Red Hat OpenStack Platform 17.1

네트워크 기능 가상화 구성

Red Hat Openstack Platform에서 NFV(네트워크 기능 가상화) 계획 및 구성

Last Updated: 2024-09-27

Red Hat OpenStack Platform 17.1 네트워크 기능 가상화 구성

Red Hat Openstack Platform에서 NFV(네트워크 기능 가상화) 계획 및 구성

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

이 안내서에는 중요한 계획 정보가 포함되어 있으며 Red Hat OpenStack Platform 배포의 NFV(네트워크 기능 가상화 인프라)를 위한 SR-IOV(Single Root Input/output virtualization) 및 DPDK(Dataplane Development Kit)의 구성 절차에 대해 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 더 나은지 알려주십시오.

Jira에서 문서 피드백 제공

Create Issue 양식을 사용하여 OpenShift (RHOSO) 또는 이전 Red Hat OpenStack Platform (RHOSP)의 Red Hat OpenStack Services 문서에 대한 피드백을 제공합니다. RHOSO 또는 RHOSP 문서에 대한 문제를 생성할 때 RHOSO Jira 프로젝트에 문제가 기록되어 피드백의 진행 상황을 추적할 수 있습니다.

문제 생성 양식을 완료하려면 Jira에 로그인해야 합니다. Red Hat Jira 계정이 없는 경우 https://issues.redhat.com 에서 계정을 생성할 수 있습니다.

1. 다음 링크를 클릭하여 **문제 생성** 페이지를 엽니다.
   https://issues.redhat.com/secure/CreateIssueDetails!init.jspa?pid=12336920&summary=Documentation%20feedback:%20%3CAdd%20summary%20here%3E&i
   <Include+the+documentation+URL,+the%20chapter+or+section+number,+and+a+detailed+descrip

2. **요약** 및 **설명** 필드를 작성합니다. **설명** 필드에 문서 URL, 장 또는 섹션 번호, 문제에 대한 자세한 설명을 포함합니다. 양식의 다른 필드를 수정하지 마십시오.

3. **생성**을 클릭합니다.

1장. RED HAT NETWORK FUNCTIONS VIRTUALIZATION(NFV) 이해

NFV(Network Functions Virtualization)는 통신 서비스 공급자(CSP)가 기존의 독점 하드웨어 이상으로 전환하여 운영 비용을 줄이면서 효율성과 민첩성을 높이는 데 도움이 되는 소프트웨어 기반 솔루션입니다.

NFV 환경은 스위치, 라우터 및 스토리지와 같은 표준 하드웨어 장치에서 실행되는 표준 가상화 기술을 사용하여 네트워크 기능 가상화(VNF)를 사용하여 가상화 인프라를 제공하여 IT 및 네트워크 통합을 허용합니다. 관리 및 오케스트레이션 논리는 이러한 서비스를 배포하고 유지합니다. NFV에는 Systems Administration, Automation 및 Life-Cycle Management도 포함되어 있으므로 필요한 수동 작업을 줄일 수 있습니다.

1.1. NFV의 이점

NFV(네트워크 기능 가상화) 구현의 주요 이점은 다음과 같습니다.

- 변화하는 요구에 대응하기 위해 새로운 네트워킹 서비스를 신속하게 배포하고 확장할 수 있으므로 시장 출시 시간을 단축합니다.
- 서비스 개발자가 프로덕션에서 사용할 플랫폼과 동일한 플랫폼을 사용하여 자체 리소스 및 프로토타입을 관리할 수 있도록 하여 혁신을 지원합니다.
- 보안이나 성능을 저하시키지 않고 고객의 요구 사항을 몇 주 또는 며칠이 아니라 몇 시간 또는 몇 분 내에 해결하십시오.
- 비용이 많이 드는 맞춤형 장비 대신 기성품 하드웨어를 사용하기 때문에 대문자를 줄입니다.
- 일상적인 작업을 최적화하여 직원의 생산성을 개선하고 운영 비용을 줄이는 간소화된 운영 및 자동화를 사용합니다.

1.2. NFV 배포에 지원되는 구성

Red Hat OpenStack Platform director 툴킷을 사용하여 외부, 프로젝트, 내부 API 등과 같은 특정 네트워크 유형을 분리할 수 있습니다. 단일 네트워크 인터페이스에 네트워크를 배포하거나 여러 호스트 네트워크 인터페이스를 통해 배포할 수 있습니다. Open vSwitch를 사용하면 단일 브리지에 여러 인터페이스를 할당하여 본딩을 만들 수 있습니다. 템플릿 파일을 사용하여 Red Hat OpenStack Platform 설치에서 네트워크 격리를 구성합니다. 템플릿 파일을 제공하지 않으면 서비스 네트워크가 provisioning 네트워크에 배포됩니다.

템플릿 구성 파일에는 다음 두 가지 유형이 있습니다.

- **network-environment.yaml**
  이 파일에는 오버클라우드 노드에 대한 서브넷 및 IP 주소 범위와 같은 네트워크 세부 정보가 포함되어 있습니다. 이 파일에는 다양한 시나리오의 기본 매개변수 값을 재정의하는 다양한 설정이 포함되어 있습니다.
- 호스트 네트워크 템플릿(예: compute.yaml 및 controller.yaml)
  이러한 템플릿은 오버클라우드 노드의 네트워크 인터페이스 구성을 정의합니다. 네트워크 세부 정보 값은 **network-environment.yaml** 파일에서 제공합니다.

이러한 heat 템플릿 파일은 언더클라우드 노드의 **/usr/share/openstack-tripleo-heat-templates/** 에 있습니다. NFV용 이러한 heat 템플릿 파일의 샘플은 샘플 DPDK SR-IOV YAML 파일을 참조하십시오.

하드웨어 요구 사항 및 소프트웨어 요구 사항 섹션에서는 Red Hat OpenStack Platform director를 사용하여 NFV의 heat 템플릿 파일을 계획하고 구성하는 방법에 대해 자세히 설명합니다.

YAML 파일을 편집하여 NFV를 구성할 수 있습니다. YAML 파일 형식에 대한 소개는 Nutshell에서 YAML을 참조하십시오.

DPDK(Data Plane Development Kit) 및 SR-IOV(Single Root I/O Virtualization)

RHOSP(Red Hat OpenStack Platform)는 자동화된 OVS-DPDK 및 SR-IOV 구성이 포함된 NFV 배포를 지원합니다.

중요

Red Hat은 비NFV 워크로드에 OVS-DPDK 사용을 지원하지 않습니다. 비NFV 워크로드에 OVS-DPDK 기능이 필요한 경우 TAM(Technical Account Manager)에 문의하거나 고객 서비스 요청 케이스를 열어 지원 예외 및 기타 옵션에 대해 논의합니다. 고객 서비스 요청 케이스를 열려면 케이스 생성으로 이동하여 **계정 > 고객 서비스**요청을 선택합니다.

HCI(하이퍼 컨버지드 인프라)

Compute 하위 시스템을 Red Hat Ceph Storage 노드와 함께 배치할 수 있습니다. 이 하이퍼 컨버지드 모델은 더 낮은 입력 비용, 초기 배포 공간, 최대 용량 활용 및 NFV 사용 사례에서 보다 효율적인 관리를 제공합니다. HCI에 대한 자세한 내용은 *하이퍼컨버지드 인프라* 배포를 참조하십시오.

구성 가능 역할

구성 가능 역할을 사용하여 사용자 지정 배포를 생성할 수 있습니다. 구성 가능 역할을 사용하면 각 역할에서 서비스를 추가하거나 제거할 수 있습니다. Composable Roles에 대한 자세한 내용은 *Red Hat OpenStack Platform 배포 사용자 지정의 Composable 서비스 및 사용자 지정 역할*을 참조하십시오.

LACP를 사용한 OVS(Open vSwitch)

OVS 2.9부터 OVS를 사용한 LACP가 완전히 지원됩니다. OVS 또는 Openstack Networking이 중단되면 관리를 방해할 수 있으므로 Openstack 컨트롤 플레인 트래픽에는 권장되지 않습니다. 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* OVS(Open vSwitch) 본딩 옵션을 참조하십시오.

OVS 하드웨어 오프로드

Red Hat OpenStack Platform은 제한 사항과 함께 OVS 하드웨어 오프로드의 배포를 지원합니다. 하드웨어 오프로드를 사용하여 OVS를 배포하는 방법에 대한 자세한 내용은 OVS 하드웨어 오프로드 구성을 참조하십시오.

OVN(Open Virtual Network)

RHOSP 16.1.4에서 다음 NFV OVN 구성을 사용할 수 있습니다.

- OVS-DPDK 및 SR-IOV를 사용하여 OVN 배포
- OVS TC Flower 오프로드를 사용하여 OVN 배포

1.3. NFV 데이터 플레인 연결

NFV가 도입됨에 따라 더 많은 네트워킹 벤더가 VNF로 기존 장치를 구현하기 시작했습니다. 대부분의 네트워킹 벤더는 가상 머신을 고려하고 있지만 일부는 설계 선택으로 컨테이너 기반 접근 방식을 조사하고 있습니다. OpenStack 기반 솔루션은 다음과 같은 두 가지 주요 이유로 인해 풍부하고 유연해야 합니다.

- **애플리케이션 준비** - 네트워크 벤더가 현재 장치를 VNF로 변환하는 중입니다. 시장 내 VNF는 서로 다른 수준의 완성도를 가지고 있습니다. 이러한 준비에 대한 일반적인 장애로는 API에서 RESTful 인터페이스 활성화, 데이터 모델을 상태 비저장으로 발전시키고, 자동화된 관리 작업을

제공하는 것이 포함됩니다. OpenStack은 모두를 위한 공통 플랫폼을 제공해야 합니다.

- **광범위한 사용 사례 -** NFV에는 다양한 사용 사례를 제공하는 다양한 애플리케이션이 포함되어 있습니다. 예를 들어, vCPE(Virtual Customer Premise Equipment)는 고객 환경에서 라우팅, 방화벽, VPN(Virtual Private Network) 및 NAT(Network Address Translation)와 같은 여러 네트워크 기능을 제공하는 것을 목표로 합니다. vEPC(Virtual Evolved Packet Core)는 LTE(Long-Term Evolution) 네트워크의 핵심 구성 요소에 비용 효율적인 플랫폼을 제공하는 클라우드 아키텍처로, 게이트웨이 및 모바일 끝점의 동적 프로비저닝을 통해 스마트 카드 및 기타 장치에서 데이터 트래픽의 양을 늘릴 수 있습니다.
  이러한 사용 사례는 서로 다른 네트워크 애플리케이션 및 프로토콜을 사용하여 구현되며 인프라의 서로 다른 연결, 격리 및 성능 특성이 필요합니다. 컨트롤 플레인 인터페이스와 프로토콜과 실제 전달 플레인을 구분하는 것도 일반적입니다. OpenStack은 다양한 데이터 경로 연결 옵션을 제공할 수 있을 만큼 유연해야 합니다.

기본적으로 가상 머신에 데이터 플레인 연결을 제공하는 두 가지 일반적인 방법이 있습니다.

- **직접 하드웨어 액세스** 는 linux 커널을 우회하고 가상 기능(VF) 및 PF(물리 기능) 패스스루 모두에 대해 PCI Passthrough 또는 SR-IOV(Single Root I/O Virtualization)와 같은 기술을 사용하여 물리적 NIC에 대한 안전한 직접 메모리 액세스(DMA)를 제공합니다.

- 하이퍼바이저의 소프트웨어 서비스로 구현된 **가상 스위치(vswitch) 사용**. 가상 머신은 가상 인터페이스(vNIC)를 사용하여 vSwitch에 연결되며 vSwitch는 가상 머신과 물리적 네트워크 간에 트래픽을 전달할 수 있습니다.

빠른 데이터 경로 옵션 중 일부는 다음과 같습니다.

- **SR-IOV(Single Root I/O Virtualization)**는 단일 PCI 하드웨어 장치를 여러 가상 PCI 장치로 표시하는 표준입니다. 물리적 하드웨어 포트를 나타내는 완전한 기능을 갖춘 PCIe 함수인 물리적 기능(PF)과 가상 머신에 할당된 경량 함수인 VF(가상 기능)를 도입하여 작동합니다. VM에 대해 VF는 하드웨어와 직접 통신하는 일반 NIC와 유사합니다. NIC는 여러 VF를 지원합니다.

- **OVS(Open vSwitch)** 는 가상화된 서버 환경 내에서 가상 스위치로 사용하도록 설계된 오픈 소스 소프트웨어 스위치입니다. OVS는 일반 L2-L3 스위치의 기능을 지원하며, OpenFlow와 같은 SDN 프로토콜을 지원하여 사용자 정의 오버레이 네트워크(예: VXLAN)를 만듭니다. OVS는 Linux 커널 네트워킹을 사용하여 가상 머신과 물리적 NIC를 사용하는 호스트 간에 패킷을 전환합니다. OVS는 iptables/ebtables를 사용하는 Linux 브리지의 오버헤드를 피하기 위해 내장 방화벽 기능이 내장된 연결 추적(Conntrack)을 지원합니다. Red Hat OpenStack Platform 환경의 Open vSwitch는 OVS와의 기본 OpenStack Networking(neutron) 통합을 제공합니다.

- **DPDK(Data Plane Development Kit)** 는 라이브러리 세트와 빠른 패킷 처리를 위한 PMD(폴링 모드 드라이버)로 구성됩니다. 대부분의 사용자 공간을 실행하도록 설계되었으므로 애플리케이션이 또는 NIC에서 직접 자체 패킷 처리를 수행할 수 있습니다. DPDK는 대기 시간을 줄이고 더 많은 패킷을 처리할 수 있습니다. DPDK PMD(Pll Mode Drivers)는 사용 중인 루프로 실행되어 패킷 도착을 위해 게스트의 호스트 및 vNIC 포트에서 NIC 포트를 지속적으로 스캔합니다.

- **DPDK 가속화 Open vSwitch(OVS-DPDK)** 는 Linux 커널 바이패스 및 DMA(직접 메모리 액세스)를 물리적 NIC로 사용하는 고성능 사용자 공간 솔루션을 위해 DPDK와 함께 제공되는 Open vSwitch입니다. 표준 OVS 커널 데이터 경로를 DPDK 기반 데이터 경로로 교체하여 패킷 전달을 위해 DPDK를 내부적으로 사용하는 호스트에서 사용자 공간 vSwitch를 생성하는 것입니다. 이 아키텍처의 장점은 사용자에게 가장 투명하다는 것입니다. OpenFlow, OVSDB와 같이 표시되는 인터페이스는 대부분 동일하게 유지됩니다.

1.4. CRYOSTATI NFV 아키텍처

European Tlecommunications Standards Institute (ETSI)는 유럽의 정보 및 통신 기술 (ICT)에 대한 표준을 개발하는 독립적 표준화 그룹입니다.

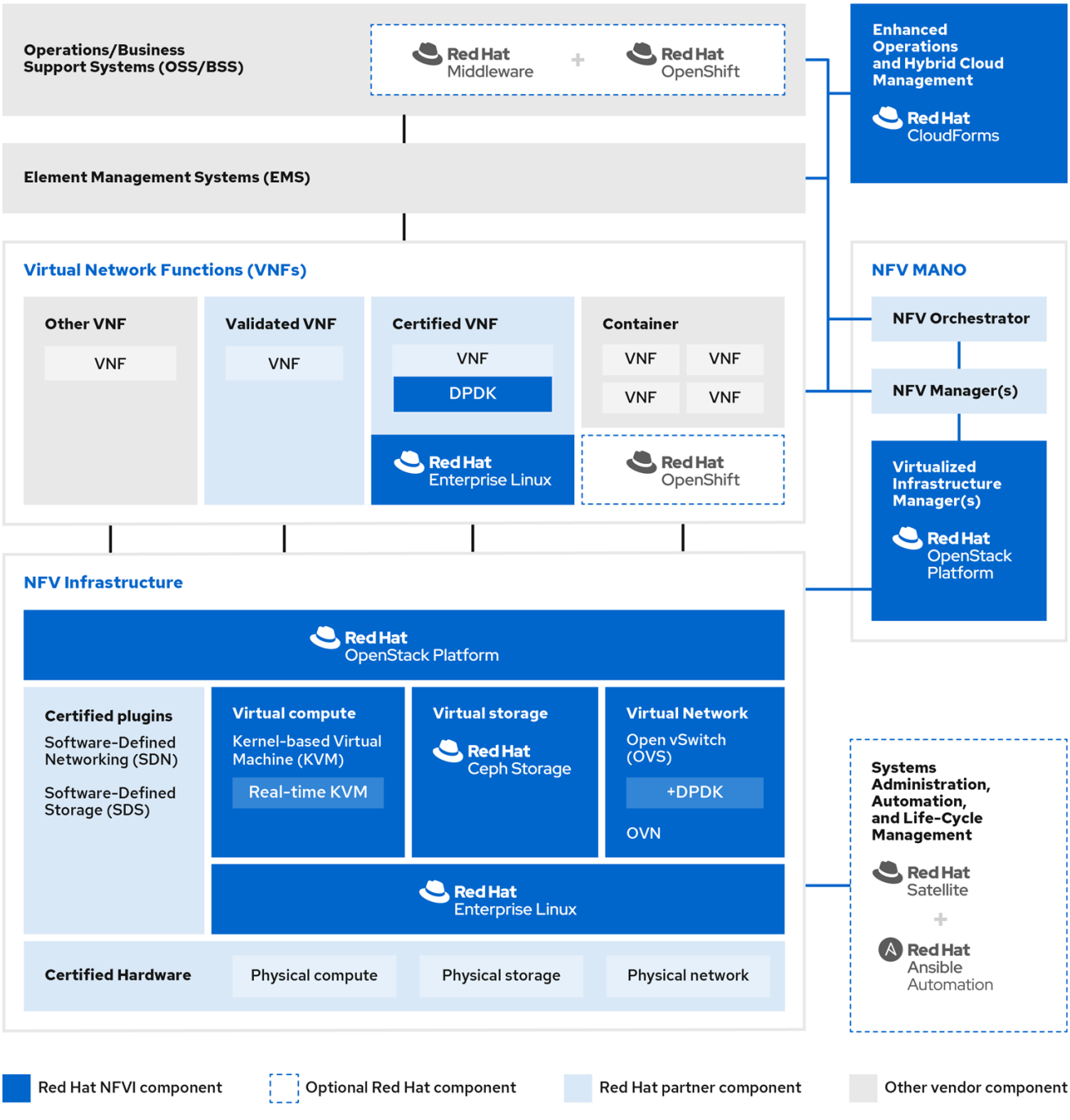
NFV(네트워크 기능 가상화)는 독점 하드웨어 장치 사용과 관련된 문제를 해결하는 데 중점을 둡니다. NFV를 사용하면 사용 사례 요구 사항 및 경제적 이점에 따라 네트워크 관련 장비 설치 필요성이 줄어듭니다. CryostatI Industry Specification Group for Network Functions Virtualization(ETSI ISG NFV)은 가상화 기능을 지원하는 데 필요한 요구 사항, 참조 아키텍처 및 인프라 사양을 설정합니다.

Red Hat은 통신 서비스 공급자(CSP)가 IT 및 네트워크 통합을 달성하는 데 도움이 되는 오픈 소스 기반 클라우드 최적화 솔루션을 제공하고 있습니다. Red Hat은 단일 루트 I/O 가상화(SR-IOV) 및 OVS-DPDK(Data Plane Development Kit)가 포함된 Open vSwitch와 같은 NFV 기능을 Red Hat OpenStack에 추가합니다.

1.5. NFV CRYOSTATI 아키텍처 및 구성 요소

일반적으로 NFV(네트워크 기능 가상화) 플랫폼에는 다음과 같은 구성 요소가 있습니다.

그림 1.1. NFV CryostatI 아키텍처 및 구성 요소



- **VNF(Virtual Network Functions)** - 라우터, 방화벽, 로드 밸런서, 초고리 게이트웨이, 모바일 패킷 프로세서, 서비스 노드, 신호링, 위치 서비스 및 기타 네트워크 기능의 소프트웨어 구현입니다.

- **NFV Infrastructure(NFVi)** - 인프라를 구성하는 물리적 리소스(컴퓨팅, 스토리지, 네트워크) 및 가상화 계층입니다. 네트워크에는 가상 시스템과 호스트 간에 패킷을 전달하는 데이터 경로가 포함됩니다. 이를 통해 기본 하드웨어의 세부 정보에 대해 우려하지 않고 VNF를 설치할 수 있습니다. NFVi는 NFV 스택의 기반을 형성합니다. NFVi는 멀티 테넌시를 지원하며 VIM(Virtual Infrastructure Manager)에서 관리합니다. 향상된 플랫폼 인식(EPA)은 낮은 수준의 CPU 및 NIC 가속 구성 요소를 VNF에 노출하여 가상 머신 패킷 전달 성능(처리량, 대기 시간, 지터)을 향상시킵니다.

- **NFV 관리 및 오케스트레이션(MANO)** - 관리 및 오케스트레이션 계층은 VNF의 라이프 사이클 전반에 걸쳐 필요한 모든 서비스 관리 작업에 중점을 둡니다. MANO의 주요 목표는 운영자가 제공하는 네트워크 기능의 서비스 정의, 자동화, 오류 분류, 모니터링 및 라이프 사이클 관리를 물리적 인프라와 분리하여 허용하는 것입니다. 이러한 분리에는 VNFM(Virtual Network Function Manager)에서 제공하는 추가 관리 계층이 필요합니다. VNFM은 가상 머신과 직접 상호 작용하거나 VNF 벤더가 제공하는 Element Management System(element Management System)을 통해 가상 머신 및 VNF의 라이프 사이클을 관리합니다. MANO가 정의하는 다른 중요한 구성 요소는 NFVO라고도 하는 Orchestrator입니다. NFVO는 상단과 VNFM의 운영/비즈니스 지원 시스템(OSS/BSS)을 포함한 다양한 데이터베이스 및 시스템과 상호 작용합니다. NFVO가 고객을 위한 새 서비스를 생성하려는 경우 VNFM에 VNF를 트리거하도록 요청하면 여러 가상 시스템이 발생할 수 있습니다.

- **운영 및 비즈니스 지원 시스템(OSS/BSS)**은 필수 비즈니스 기능 애플리케이션(예: 운영 지원 및 청구)을 제공합니다. OSS/BSS는 기존 시스템 및 새로운 MANO 구성 요소와 통합되어 NFV에 맞게 조정해야 합니다. BSS 시스템은 서비스 서브스크립션을 기반으로 정책을 설정하고 보고 및 청구를 관리합니다.

- **시스템 관리, 자동화 및 라이프 사이클 관리** - 시스템 관리, 인프라 구성 요소의 자동화 및 NFVi 플랫폼의 라이프 사이클을 관리합니다.

1.6. RED HAT NFV 구성 요소

NFV를 위한 Red Hat 솔루션에는 CryostatI 모델에서 NFV 프레임워크의 다양한 구성 요소 역할을 할 수 있는 다양한 제품이 포함되어 있습니다. Red Hat 포트폴리오의 다음 제품은 NFV 솔루션에 통합됩니다.

- Red Hat OpenStack Platform - IT 및 NFV 워크로드를 지원합니다. 향상된 플랫폼 인식(EPA) 기능은 SR-IOV 및 OVS-DPDK를 지원하는 CPU 고정, Huge 페이지, NUMA(Non-Uniform Memory Access) 선호도 및 NIC(네트워크 어댑터)를 통해 명확한 성능 향상을 제공합니다.

- Red Hat Enterprise Linux 및 Red Hat Enterprise Linux Atomic Host - 가상 머신 및 컨테이너를 VNF로 만듭니다.

- Red Hat Ceph Storage - 서비스 공급자 워크로드의 모든 요구 사항에 맞게 탄력적이고 고성능 스토리지 계층을 제공합니다.

- Red Hat JBoss Middleware 및 OpenShift Enterprise - 필요한 경우 OSS/BSS 구성 요소를 현대화할 수 있는 기능을 제공합니다.

- Red Hat CloudForms - VNF 관리자를 제공하고 통합 디스플레이에서 VIM 및 NFVi와 같은 여러 소스의 데이터를 제공합니다.

- Red Hat Satellite 및 Ansible - 필요한 경우 향상된 시스템 관리, 자동화 및 라이프 사이클 관리를 제공합니다.

1.7. NFV 설치 요약

Red Hat OpenStack Platform director는 완전한 OpenStack 환경을 설치하고 관리합니다. director는 "OpenStack-On-OpenStack"의 약어인 업스트림 OpenStack TripleO 프로젝트를 기반으로 합니다. 이 프로젝트에는 OpenStack 구성 요소를 활용하여 완전히 작동하는 OpenStack 환경을 설치합니다. 여기에는 언더클라우드라는 최소한의 OpenStack 노드가 포함됩니다. 언더클라우드는 오버클라우드를 프로비저닝하고 제어합니다(프로덕션 OpenStack 노드로 사용되는 일련의 베어 메탈 시스템). director는 간결하고 강력한 전체 Red Hat OpenStack Platform 환경을 간단하게 설치할 수 있는 방법을 제공합니다.

언더클라우드 및 오버클라우드 설치에 대한 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리를* 참조하십시오.

NFV 기능을 설치하려면 다음 추가 단계를 완료합니다.

- **network-environment.yaml** 파일에 SR-IOV 및 PCI Passthrough 매개변수를 포함하고, CPU 튜닝을 위한 **설치 후.yaml** 파일을 업데이트하고, **compute.yaml** 파일을 수정하고, **overcloud_deploy.sh** 스크립트를 실행하여 오버클라우드를 배포합니다.

- NIC에서 직접 데이터를 폴링하여 빠른 패킷 처리를 위한 DPDK 라이브러리 및 드라이버를 설치합니다. **network-environment.yaml** 파일에 DPDK 매개변수를 포함하고, CPU 튜닝을 위해 **post-install.yaml** 파일을 업데이트하고, DPDK 포트로 브릿지를 설정하도록 **compute.yaml** 파일을 업데이트하고, **controller.yaml** 파일을 업데이트하여 VLAN이 구성된 브리지 및 인터페이스를 업데이트하고 **overcloud_deploy.sh** 스크립트를 실행하여 오버클라우드를 배포합니다.

2장. NFV 성능 고려 사항

NFV(네트워크 기능 가상화) 솔루션을 사용하려면 가상화된 기능이 물리적 구현의 성능을 충족하거나 초과해야 합니다. Red Hat의 가상화 기술은 OpenStack 및 클라우드 배포에서 공통된 고성능 KVM(커널 기반 가상 시스템) 하이퍼바이저를 기반으로 합니다.

Red Hat OpenStack Platform director는 리소스 파티셔닝을 적용하고 게스트 가상 네트워크 기능(VNF)의 라인 속도 성능을 달성하기 위해 컴퓨팅 노드를 구성합니다. NFV 사용 사례의 주요 성능 요인은 처리량, 대기 시간 및 지터입니다.

DPDK(데이터 플레인 개발 키트) 가속화 가상 머신을 사용하여 물리적 NIC와 가상 머신 간에 고성능 패킷 전환을 활성화할 수 있습니다. OVS 2.10에는 DPDK 17에 대한 지원이 포함되어 있으며 vhost-user multiqueue 지원이 포함되어 확장 가능한 성능을 제공합니다. OVS-DPDK는 게스트 VNF에 대한 라인 등급 성능을 제공합니다.

SR-IOV(Single Root I/O Virtualization) 네트워킹은 특정 네트워크 및 가상 머신에 대한 처리량 향상을 포함하여 향상된 성능을 제공합니다.

성능 튜닝의 기타 중요한 기능에는 대규모 페이지, NUMA 정렬, 호스트 격리, CPU 고정이 포함됩니다. VNF 플레이버에는 성능 향상을 위해 대규모 페이지 및 에뮬레이터 스레드 격리가 필요합니다. 호스트 격리 및 CPU 고정을 통해 NFV 성능이 향상되고 패킷 손실이 발생하지 않습니다.

2.1. CPU 및 NUMA 노드

이전에는 x86 시스템의 모든 메모리가 시스템의 모든 CPU에 동일하게 액세스할 수 있었습니다. 이로 인해 시스템의 CPU가 작업을 수행하고 UMA(Uniform Memory Access)라고 하는 CPU와 관계없이 메모리 액세스 시간이 동일했습니다.

NUMA(Non-Uniform Memory Access)에서 시스템 메모리는 특정 CPU 또는 소켓에 할당된 노드라는 영역으로 나뉩니다. CPU에 로컬인 메모리에 대한 액세스는 해당 시스템의 원격 CPU에 연결된 메모리보다 빠릅니다. 일반적으로 NUMA 시스템의 각 소켓에는 로컬 노드의 메모리보다 콘텐츠를 다른 CPU 또는 모든 CPU에서 공유하는 버스의 메모리보다 더 빠르게 액세스할 수 있는 로컬 메모리 노드가 있습니다.

마찬가지로 물리적 NIC는 컴퓨팅 노드 하드웨어의 PCI 슬롯에 배치됩니다. 이러한 슬롯은 특정 NUMA 노드에 연결된 특정 CPU 소켓에 연결됩니다. 최대한 성능을 얻으려면 datapath NIC를 CPU 구성의 동일한 NUMA 노드(SR-IOV 또는 OVS-DPDK)에 연결합니다.

NUMA 누락의 성능 영향은 중요하며 일반적으로 10% 이상의 성능 적중을 시작합니다. 각 CPU 소켓에는 가상화를 위해 개별 CPU로 취급되는 여러 CPU 코어가 있을 수 있습니다.

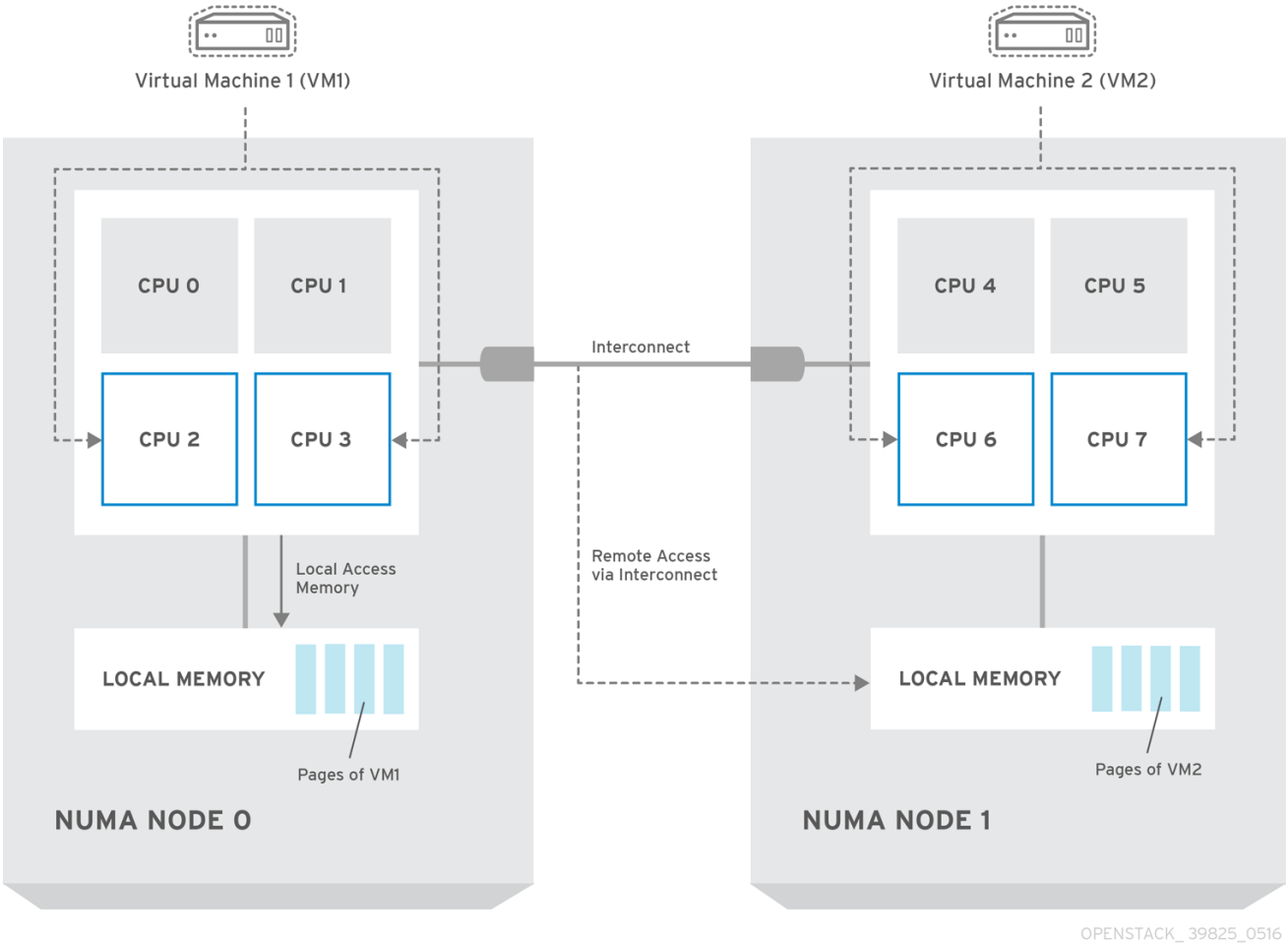
작은 정보

NUMA에 대한 자세한 내용은 NUMA란 무엇이며 Linux에서 어떻게 작동합니까?

2.1.1. NUMA 노드 예

다음 다이어그램에서는 2-노드 NUMA 시스템의 예와 CPU 코어 및 메모리 페이지를 사용할 수 있는 방법을 보여줍니다.

그림 2.1. 예: 2-노드 NUMA 시스템



참고

Interconnect를 통해 사용할 수 있는 원격 메모리는 NUMA 노드 0의 VM1에 NUMA 노드 1의 CPU 코어가 있는 **경우에만** 액세스할 수 있습니다. 이 경우 NUMA 노드 1의 메모리는 VM1의 세 번째 CPU 코어에 대해 로컬 역할을 하지만(예: 위의 다이어그램에서 VM1이 CPU 4에 할당된 경우) 동일한 VM의 다른 CPU 코어에 대한 원격 메모리 역할을 합니다.

2.1.2. NUMA 인식 인스턴스

NUMA 아키텍처가 있는 시스템에서 NUMA 토폴로지 인식을 사용하도록 OpenStack 환경을 구성할 수 있습니다. VM(가상 머신)에서 게스트 운영 체제를 실행하는 경우 다음과 같은 두 가지 NUMA 토폴로지가 포함됩니다.

- 호스트의 물리적 하드웨어의 NUMA 토폴로지
- 게스트 운영 체제에 노출되는 가상 하드웨어의 NUMA 토폴로지

가상 하드웨어를 물리적 하드웨어 NUMA 토폴로지와 조정하여 게스트 운영 체제의 성능을 최적화할 수 있습니다.

2.2. CPU 고정

CPU 고정은 지정된 호스트의 특정 물리적 CPU에서 특정 가상 시스템의 가상 CPU를 실행하는 기능입니다. vCPU 고정은 베어 메탈 시스템에서 작업을 고정하는 것과 유사한 이점을 제공합니다. 가상 머신은 호스트 운영 체제에서 사용자 공간 작업으로 실행되므로 고정으로 캐시 효율성이 향상됩니다.

CPU 고정 구성 방법에 대한 자세한 내용은 *인스턴스 생성 가이드의 Compute 서비스 구성 가이드의 link:https://access.redhat.com/documentation/en-us/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 [컴퓨팅 노드에 CPU 고정 구성]*을 참조하십시오.

2.3. HUGE PAGE

물리적 메모리는 페이지라는 연속 영역으로 분할됩니다. 효율성을 위해 시스템은 개별 메모리 대신 전체 페이지에 액세스하여 메모리를 검색합니다. 이 변환을 수행하기 위해 시스템은 가장 최근에 사용되거나 자주 사용되는 페이지의 물리적 주소 매핑이 포함된 TLB(Translation Lookaside Buffers)를 찾습니다. 시스템이 TLB에서 매핑을 찾을 수 없는 경우 프로세서는 모든 페이지 테이블을 반복하여 주소 매핑을 확인해야 합니다. 이러한 TLB 누락 중에 발생하는 성능 저하를 최소화하기 위해 TLB를 최적화합니다.

x86 시스템의 일반적인 페이지 크기는 4KB이며 다른 큰 페이지 크기를 사용할 수 있습니다. 페이지 크기가 클수록 전체 페이지가 줄어들고 따라서 가상을 물리적 주소 변환으로 저장할 수 있는 시스템 메모리 양이 늘어납니다. 결과적으로 TLB 누락이 줄어들어 성능이 향상됩니다. 페이지 크기가 클수록 프로세스에서 페이지를 할당해야 하므로 메모리가 활용도가 낮지만 일부 메모리가 필요하지는 않습니다. 결과적으로 페이지 크기를 선택하는 것은 더 큰 페이지로 더 빠른 액세스 시간을 제공하고 더 작은 페이지로 메모리 사용률을 극대화하는 것 사이의 저하입니다.

2.4. 포트 보안

포트 보안은 원래 네트워크 포트의 소스 IP 및 소스 MAC 주소와 일치하지 않는 송신 트래픽을 차단하는 방지 수단입니다. 보안 그룹 규칙을 사용하여 이 동작을 보거나 수정할 수 없습니다.

기본적으로 **port_security_enabled** 매개 변수는 OpenStack에서 새로 생성된 Neutron 네트워크에서 **활성화** 되도록 설정됩니다. 새로 생성된 포트는 생성된 네트워크에서 **port_security_enabled** 매개변수 값을 복사합니다.

방화벽 또는 라우터 빌드와 같은 일부 NFV 사용 사례의 경우 포트 보안을 비활성화해야 합니다.

단일 포트에서 포트 보안을 비활성화하려면 다음 명령을 실행합니다.

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

네트워크의 새로 생성된 포트에서 포트 보안이 활성화되지 않도록 하려면 다음 명령을 실행합니다.

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

3장. NFV 하드웨어 요구 사항

이 섹션에서는 NFV의 하드웨어 요구 사항에 대해 설명합니다.

Red Hat은 Red Hat OpenStack Platform과 함께 사용할 하드웨어를 인증합니다. 자세한 내용은 인증된 하드웨어를 참조하십시오.

3.1. NFV에 대해 테스트된 NIC

NFV용 테스트된 NIC 목록은 Red Hat Knowledgebase 솔루션 Network Adapter Fast Datapath Feature Support Matrix 를 참조하십시오.

NVIDIA (M#159nox) 네트워크 인터페이스에서 OVS-DPDK를 구성하지 않는 한 지원되는 NIC에 기본 드라이버를 사용합니다. NVIDIA 네트워크 인터페이스의 경우 j2 네트워크 구성 템플릿에 해당 커널 드라이버를 설정해야 합니다.

예제

이 예에서 **mlx5_core** 드라이버는 Mellanox ConnectX-5 네트워크 인터페이스에 대해 설정됩니다.

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

3.2. 하드웨어 오프로드 문제 해결

RHOSP(Red Hat OpenStack Platform) 17.1 배포에서 OVS Hardware Offload는 **switchdev**-able ports 및 Mellanox ConnectX5 NIC를 사용하여 VM의 흐름을 오프로드하지 않을 수 있습니다. 이 시나리오에서 오프로드 흐름의 문제를 해결하고 구성하려면 **ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox 펌웨어 매개변수를 비활성화합니다. RHOSP 17.1의 OVS Hardware Offload에 대한 자세한 내용은 Red Hat Knowledgebase 솔루션 OVS Hardware Offload with Mellanox NIC in OpenStack Platform 16.2 에서 참조하십시오.

절차

1. RHOSP 배포의 Compute 노드에 구성하려는 Mellanox NIC가 있는 컴퓨팅 노드에 로그인합니다.
2. **mstflint** 유틸리티를 사용하여 **ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox 펌웨어 매개변수를 쿼리합니다.

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

3. **ESWITCH_IPV4_TTL_MODIFY_ENABLE** 매개변수가 활성화되고 **1** 로 설정된 경우 값을 **0** 으로 설정하여 비활성화합니다.

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

4. 노드를 재부팅합니다.

3.3. NUMA 노드 토폴로지 검색

배포를 계획할 때는 정상 성능을 위해 CPU 및 메모리 리소스를 파티셔닝하려면 컴퓨팅 노드의 NUMA 토폴로지를 이해해야 합니다. NUMA 정보를 확인하려면 다음 작업 중 하나를 수행합니다.

- 하드웨어 인트로스펙션을 활성화하여 베어 메탈 노드에서 이 정보를 검색할 수 있습니다.
- 각 베어 메탈 노드에 로그온하여 수동으로 정보를 수집합니다.

참고

하드웨어 인트로스펙션을 통해 NUMA 정보를 검색하려면 언더클라우드를 설치하고 구성해야 합니다. 언더클라우드 구성에 대한 자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리를 참조하십시오*.

3.4. 하드웨어 인트로스펙션 세부 정보 검색

기본적으로 Bare Metal 서비스 하드웨어 검사 추가 기능이 활성화되므로 이 기능을 사용하여 오버클라우드 구성의 하드웨어 세부 정보를 검색할 수 있습니다. **undercloud.conf** 파일의 **inspection_extras** 매개변수에 대한 자세한 내용은 Director 구성 매개변수를 참조하십시오.

예를 들어 **numa_topology** 수집기는 하드웨어 검사 추가 기능에 포함되며 각 NUMA 노드에 대해 다음과 같은 정보가 포함되어 있습니다.

- RAM(KB)
- 물리적 CPU 코어 및 시블링 스레드
- NUMA 노드와 연결된 NIC

프로세스

- 위에 나열된 정보를 검색하려면 <UUID>를 베어 메탈 노드의 UUID로 바꿔 다음 명령을 완료하십시오.

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

다음 예제는 베어 메탈 노드에 대해 검색된 NUMA 정보를 보여줍니다.

```
{
  "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 설정

다음 표에서는 NFV에 필요한 BIOS 설정을 설명합니다.

참고

BIOS에서 SR-IOV 글로벌 및 NIC 설정을 활성화해야 합니다. 그렇지 않으면 SR-IOV 컴퓨팅 노드가 있는 RHOSP(Red Hat OpenStack Platform) 배포가 실패합니다.

표 3.1. BIOS 설정

매개변수	설정
C3 전원 상태	비활성화됨.
C6 전원 상태	비활성화됨.
MLC Streamer	활성화됨.
MLC Spatial Prefetcher	활성화됨.
DCU Data Prefetcher	활성화됨.
DCA	활성화됨.
CPU 전원 및 성능	성능.
메모리 RAS 및 성능 구성 → NUMA 구현	활성화됨.
Cryostat Boost	결정적 성능이 필요한 NFV 배포에서 비활성화되어 있습니다. 다른 모든 시나리오에서 활성화됩니다.
VT-d	VFIO 기능이 필요한 경우 Intel 카드에 대해 활성화됩니다.
NUMA 메모리 상호 작용	비활성화됨.

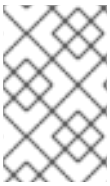
intel_idle 드라이버를 사용하는 프로세서에서 Red Hat Enterprise Linux는 BIOS 설정을 무시하고 프로세서 C-state를 다시 활성화할 수 있습니다.

intel_idle 을 비활성화하고 커널 부팅 명령줄에서 키-값 쌍 **intel_idle.max_cstate=0** 을 지정하여 **acpi_idle** 드라이버를 대신 사용할 수 있습니다.

current_driver 의 내용을 확인하여 프로세서가 **acpi_idle** 드라이버를 사용하고 있는지 확인합니다.

-

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



참고

Tuned 데몬을 시작하는 데 시간이 걸리기 때문에 드라이버를 변경한 후 약간의 대기 시간이 발생합니다. 그러나 Tuned가 로드된 후 프로세서는 더 깊은 C-state를 사용하지 않습니다.

4장. NFV 소프트웨어 요구 사항

이 섹션에서는 지원되는 구성 및 드라이버와 NFV에 필요한 서브스크립션 세부 정보에 대해 설명합니다.

4.1. 리포지토리 등록 및 활성화

Red Hat OpenStack Platform을 설치하려면 Red Hat Subscription Manager를 사용하여 Red Hat OpenStack Platform director를 등록하고 필요한 채널을 구독해야 합니다. 언더클라우드 등록 및 업데이트에 대한 자세한 내용은 언더클라우드 *등록 및 director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에 있는 서브스크립션* 연결을 참조하십시오.

프로세스

1. 메시지가 표시되면 시스템을 Content Delivery Network에 등록하고 고객 포털 사용자 이름과 암호를 입력합니다.

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

2. Red Hat OpenStack Platform director의 인타이틀먼트 풀 ID를 확인합니다(예: 다음 명령 및 출력에서 {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. 다음 명령에 **Pool ID** 값을 포함하여 Red Hat OpenStack Platform 17.1 인타이틀먼트를 연결합니다.

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

4. 기본 리포지토리를 비활성화합니다.

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

5. NFV를 사용하여 Red Hat OpenStack Platform에 필요한 리포지토리를 활성화합니다.

```
$ 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. 시스템을 업데이트하여 기본 시스템 패키지를 최신 상태로 설정합니다.

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

4.2. NFV 배포에 지원되는 구성

RHOSP(Red Hat OpenStack Platform)는 director를 사용하여 다음 NFV 배포를 지원합니다.

- SR-IOV(Single Root I/O Virtualization)
  자세한 내용은 SR-IOV 구성을 참조하십시오.
- Open vSwitch 하드웨어 오프로드
  자세한 내용은 OVS 하드웨어 오프로드 구성을 참조하십시오.
- Open vSwitch with Data Plane Development Kit (OVS-DPDK)
  자세한 내용은 OVS-DPDK 배포 구성을 참조하십시오.

또한 다음 기능을 사용하여 RHOSP를 배포할 수 있습니다.

- 구성 가능 서비스 및 사용자 지정 역할 구현.
  자세한 내용은 *Red Hat OpenStack Platform 배포 가이드의 Composable 서비스 및 사용자 지정 역할을* 참조하십시오.
- 동일한 호스트에서 Compute 및 Ceph Storage 서비스를 공동 배치합니다.
  자세한 내용은 *하이퍼컨버지드 인프라* 배포를 참조하십시오.
- Red Hat Enterprise Linux Real Time KVM(RT-KVM) 구성.
  자세한 내용은 NFV 워크로드용 RT-KVM 활성화를 참조하십시오.

4.3. NFV에 지원되는 드라이버

지원되는 드라이버의 전체 목록은 Component, Plug-In 및 Red Hat OpenStack Platform 드라이버 지원을 참조하십시오.

NFV를 사용한 Red Hat OpenStack Platform 배포용으로 테스트된 NIC 목록은 NFV 테스트 NIC를 참조하십시오.

4.4. 타사 소프트웨어와의 호환성

Red Hat OpenStack Platform에서 테스트, 지원 및 인증된 제품 및 서비스의 전체 목록은 Red Hat OpenStack Platform 과 호환되는 타사 소프트웨어를 참조하십시오. 제품 버전 및 소프트웨어 카테고리별로 목록을 필터링할 수 있습니다.

Red Hat Enterprise Linux에서 실행하도록 테스트, 지원 및 인증된 제품 및 서비스의 전체 목록은 Red Hat Enterprise Linux 와 호환되는 타사 소프트웨어를 참조하십시오. 제품 버전 및 소프트웨어 카테고리별로 목록을 필터링할 수 있습니다.

5장. NFV 네트워크 고려 사항

언더클라우드 호스트에는 최소한 다음 네트워크가 필요합니다.

- 프로비저닝 네트워크 - DHCP 및 PXE 부팅 기능을 제공하여 오버클라우드에서 사용할 베어 메탈 시스템을 검색할 수 있습니다.
- 외부 네트워크 - 모든 노드에 대한 원격 연결을 위한 별도의 네트워크입니다. 이 네트워크에 연결하는 인터페이스에는 정적으로 정의되거나 외부 DHCP 서비스에서 동적으로 생성되는 라우팅 가능한 IP 주소가 필요합니다.

최소 오버클라우드 네트워크 설정에는 다음과 같은 NIC 구성이 포함됩니다.

- 단일 NIC 설정 - 기본 VLAN 및 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN에서 provisioning 네트워크용 NIC 1개
- 듀얼 NIC 구성 - 프로비저닝 네트워크용 NIC 1개와 외부 네트워크용 다른 NIC입니다.
- 듀얼 NIC 설정 - 기본 VLAN의 프로비저닝 네트워크용 NIC 1개와 다른 오버클라우드 네트워크 유형에 서브넷을 사용하는 태그된 VLAN용 NIC입니다.
- 다중 NIC 구성 - 각 NIC에서 다른 오버클라우드 네트워크 유형에 서브넷 사용

네트워킹 요구 사항에 대한 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리*에서 언더클라우드 네트워킹 준비를 참조하십시오.

6장. SR-IOV 배포 계획

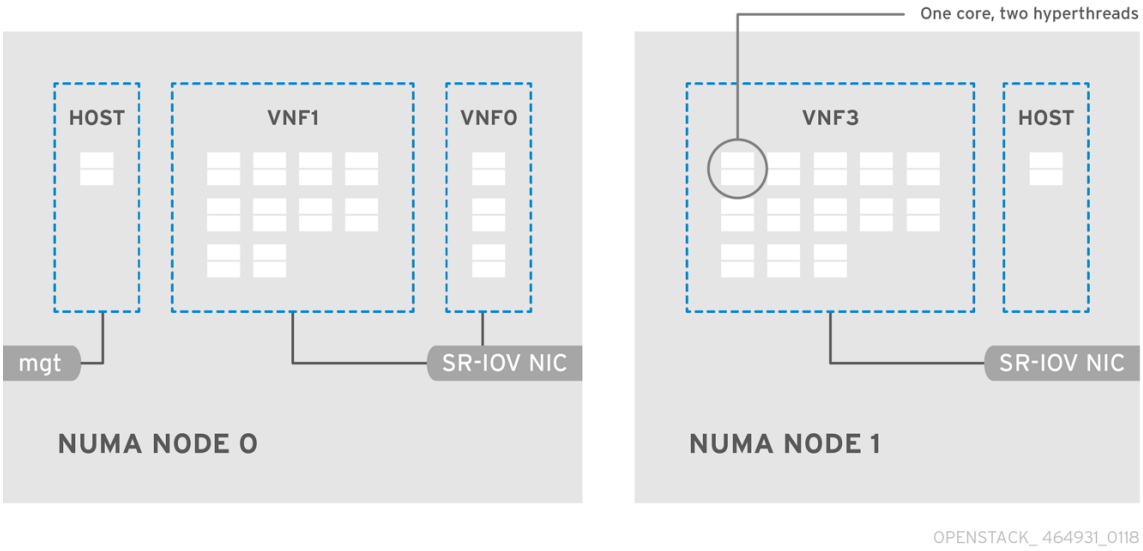
컴퓨팅 노드 하드웨어를 기반으로 개별 매개변수를 설정하여 NFV에 대한 단일 루트 I/O 가상화(SR-IOV) 배포를 최적화합니다.

SR-IOV 매개변수에 대한 하드웨어 영향을 평가 하려면 NUMA 노드 토폴로지 검색을 참조하십시오.

6.1. SR-IOV 배포를 위한 하드웨어 파티셔닝

SR-IOV를 사용하여 높은 성능을 얻으려면 호스트와 게스트 간의 리소스를 분할합니다.

그림 6.1. NUMA 노드 토폴로지

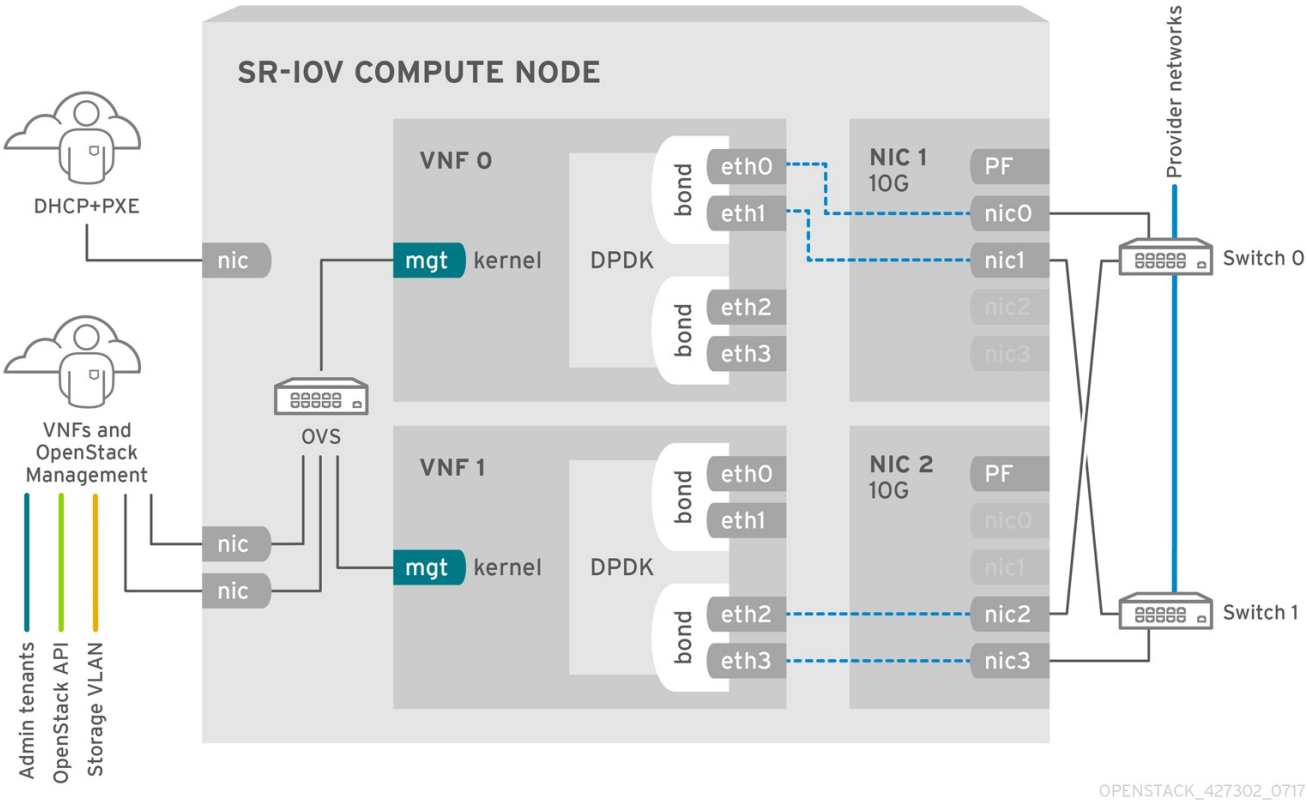


일반적인 토폴로지에는 듀얼 소켓 컴퓨팅 노드의 NUMA 노드당 14개의 코어가 포함됩니다. 하이퍼 스레딩(HT) 및 비HT 코어가 모두 지원됩니다. 각 코어에는 두 개의 형제 스레드가 있습니다. 하나의 코어는 각 NUMA 노드의 호스트에 전용됩니다. VNF(가상 네트워크 기능)는 SR-IOV 인터페이스 본딩을 처리합니다. 모든 인터럽트 요청(IRQ)은 호스트 코어에서 라우팅됩니다. VNF 코어는 VNF 전용입니다. 다른 VNF와 분리하고 호스트와의 격리를 제공합니다. 각 VNF는 단일 NUMA 노드에서 리소스를 사용해야 합니다. VNF에서 사용하는 SR-IOV NIC도 동일한 NUMA 노드와 연결되어 있어야 합니다. 이 토폴로지에는 가상화 오버헤드가 없습니다. 호스트, OpenStack Networking(neutron) 및 Compute(nova) 구성 매개변수는 쉽게, 일관성을 유지하고, 치명적인 불일치를 방지하기 위해 단일 파일에 노출되어 선점 및 패킷 손실이 발생합니다. 호스트 및 가상 머신 격리는 분리된 CPU 목록을 기반으로 부팅 매개변수와 Red Hat OpenStack Platform 수정 사항을 정의하는 **tuned** 프로필을 사용합니다.

6.2. NFV SR-IOV 배포의 토폴로지

다음 이미지에는 **mgt** 및 데이터 플레인 인터페이스로 각각 두 개의 VNF가 있습니다. 관리 인터페이스는 **ssh** 액세스를 관리합니다. 데이터 플레인 인터페이스는 DPDK 라이브러리를 사용하여 데이터 플레인 인터페이스를 결합하므로 고가용성을 보장하기 위해 VNF를 DPDK에 연결합니다. 이미지에는 중복성을 위한 두 개의 공급자 네트워크도 있습니다. 컴퓨팅 노드에는 VNF 관리와 Red Hat OpenStack Platform API 관리 간에 두 개의 일반 NIC가 결합되고 공유됩니다.

그림 6.2. NFV SR-IOV 토폴로지

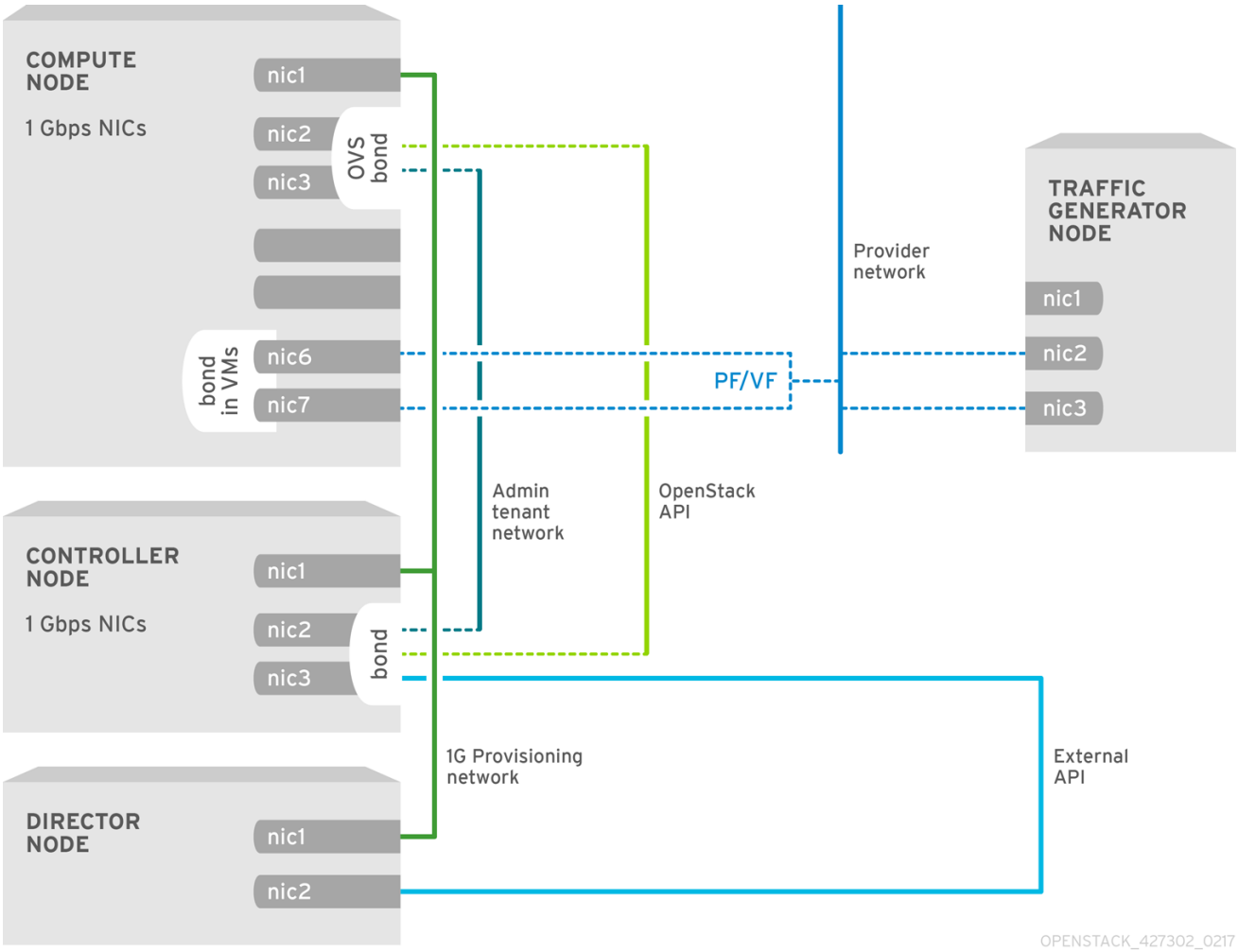


이미지에는 애플리케이션 수준에서 DPDK를 사용하고, 패브릭 구성에 따라 가용성 또는 성능을 개선하기 위해 SR-IOV 가상 기능(VF) 및 물리적 기능(PF)에 액세스할 수 있는 VNF가 표시됩니다. DPDK는 성능을 향상시키는 반면 VF/PF DPDK 본딩은 페일오버 및 고가용성을 지원합니다. VNF 벤더는 DPDK 폴링 모드 드라이버(PMD)가 VF/PF로 노출되는 SR-IOV 카드를 지원하는지 확인해야 합니다. 관리 네트워크는 OVS를 사용하므로 VNF는 표준 virtIO 드라이버를 사용하여 mgmt 네트워크 장치를 확인합니다. 해당 장치를 사용하여 처음에 VNF에 연결하고 DPDK 애플리케이션이 두 개의 VF/PF를 결합하는지 확인할 수 있습니다.

6.3. HCI 없이 NFV SR-IOV의 토폴로지

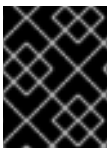
아래 이미지에서 NFV에 대한 HCI(하이퍼 컨버지드 인프라) 없이 SR-IOV의 토폴로지를 관찰합니다. 1Gbps NIC와 director 노드가 있는 컴퓨팅 및 컨트롤러 노드로 구성됩니다.

그림 6.3. HCI 없는 NFV SR-IOV 토폴로지



7장. SR-IOV 배포 구성

Red Hat OpenStack Platform NFV 배포에서는 가상 리소스를 통해 인스턴스에서 공유 PCIe 리소스로 직접 액세스를 구성할 때 SR-IOV(Single Root I/O Virtualization)를 사용하여 더 높은 성능을 얻을 수 있습니다.



중요

이 섹션에는 토폴로지 및 사용 사례에 맞게 수정해야 하는 예제가 포함되어 있습니다. 자세한 내용은 NFV 하드웨어 요구 사항을 참조하십시오.

사전 요구 사항

- RHOSP 언더클라우드.
  오버클라우드를 배포하려면 언더클라우드를 설치하고 구성해야 합니다. 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리를* 참조하십시오.

  참고

  RHOSP director는 템플릿 및 사용자 정의 환경 파일에서 지정하는 키-값 쌍을 통해 SR-IOV 구성 파일을 수정합니다. SR-IOV 파일을 직접 수정하지 않아야 합니다.

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.
- NIC가 포함된 호스트에 액세스합니다.
- NIC 펌웨어를 계속 업데이트해야 합니다.
  yum 또는 **dnf** 업데이트는 펌웨어 업데이트를 완료하지 못할 수 있습니다. 자세한 내용은 공급 업체 설명서를 참조하십시오.

절차

RHOSP(Red Hat OpenStack Platform) director를 사용하여 SR-IOV 환경에서 RHOSP를 설치 및 구성합니다. 높은 수준의 단계는 다음과 같습니다.

1. *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의 지침에 따라 오버클라우드의 물리적 네트워크를 설정하기 위해 네트워크 설정 파일* ***network_data.yaml*** *을* 생성합니다.
   https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_overcloud-networking_installing-director-on-the-undercloud
2. 역할 및 이미지 파일을 생성합니다.
3. SR-IOV 용으로 PCI 패스스루 장치를 구성합니다.
4. 역할별 매개변수 및 구성 덮어쓰기를 추가합니다.
5. 베어 메탈 노드 정의 파일을 생성합니다.
6. SR-IOV용 NIC 구성 템플릿을 생성합니다.
7. (선택 사항) 파티션 NIC.
8. 오버클라우드 네트워크 및 VIP를 프로비저닝합니다.
   자세한 내용은 다음을 참조하십시오.

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝.
- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 네트워크 VIP 구성 및 프로비저닝.

9. 오버클라우드 베어 메탈 노드를 프로비저닝합니다.
자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드의 베어 메탈 노드 프로비저닝 을 참조하십시오.

10. SR-IOV 오버클라우드를 배포합니다.

추가 리소스

- 7.7절. "NIC 파티션 구성 예"
- 7.9절. "SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 호스트 집계 생성"
- 7.10절. "SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 인스턴스 생성"

7.1. SR-IOV의 역할 및 이미지 파일 생성

RHOSP(Red Hat OpenStack Platform) director는 역할을 사용하여 노드에 서비스를 할당합니다. SR-IOV 환경에서 RHOSP를 배포할 때 **ComputeSriov** 는 기본 컴퓨팅 서비스 외에도 RHOSP 설치와 함께 제공되는 기본 역할입니다.

언더클라우드 설치에는 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 환경 파일이 필요합니다.

사전 요구 사항

- **언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.**

절차

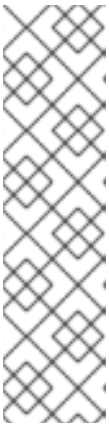
1. **stack 사용자로 언더클라우드에 로그인합니다.**

2. **stackrc 파일을 소싱합니다.**

```
$ source ~/stackrc
```

3. **Controller 및 ComputeSriov 역할을 포함하는 roles_data_compute_sriov.yaml 이라는 새 역할 데이터 파일을 생성합니다.**

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



참고

RHOSP 환경, OVS-DPDK, SR-IOV 및 OVS 하드웨어 오프로드에서 여러 기술을 사용하는 경우 모든 역할을 포함하도록 하나의 역할 데이터 파일만 생성합니다.

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

4. **이미지 파일을 생성하려면 openstack tripleo container image prepare 명령을 실행합니다. 다음 입력이 필요합니다.**

- **이전 단계에서 생성한 역할 데이터 파일(예: roles_data_compute_sriov.yaml)**
- **네트워킹 서비스 메커니즘 드라이버에 적합한 SR-IOV 환경 파일입니다.**
  - **ML2/OVN 환경**

    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml
  - **ML2/OVS 환경**

    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml

예제

이 예에서는 ML2/OVN 환경에 대해 overcloud_images.yaml 파일이 생성됩니다.

```
$ 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. 역할 데이터 파일의 경로 및 파일 이름과 사용자가 생성한 이미지 파일을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이러한 파일을 사용합니다.

다음 단계

- 7.2절. "SR-IOV의 PCI 패스스루 장치 구성" 으로 이동합니다.

추가 리소스

- 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* Composable 서비스 및 사용자 지정 역할을 참조하십시오.
- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 컨테이너 이미지 준비.

7.2. SR-IOV의 PCI 패스스루 장치 구성

SR-IOV 환경에 Red Hat OpenStack Platform을 배포할 때 사용자 정의 환경 파일에서 SR-IOV 컴퓨팅 노드에 대해 PCI 패스스루 장치를 구성해야 합니다.

사전 요구 사항

- PCI 카드가 포함된 하나 이상의 물리적 서버에 액세스합니다.
- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

절차

1. PCI 카드가 있는 물리적 서버에서 다음 명령 중 하나를 사용합니다.
   - 오버클라우드가 배포된 경우:

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

샘플 출력

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

- **오버클라우드가 배포되지 않은 경우:**

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

2. **SR-IOV 컴퓨팅 노드에서 PCI 패스스루 장치의 벤더 및 제품 ID를 유지합니다. 이후 단계에서 이러한 ID가 필요합니다.**

3. **stack 사용자로 언더클라우드에 로그인합니다.**

4. **stackrc 파일을 소싱합니다.**

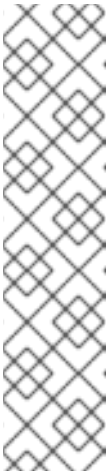
```
$ source ~/stackrc
```

5. **사용자 지정 환경 YAML 파일을 생성합니다(예: sriov-overrides.yaml). 파일에 다음 콘텐츠를 추가하여 SR-IOV 컴퓨팅 노드의 PCI 패스스루 장치를 구성합니다.**

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

- **&lt ;vendor_id>를 PCI 장치의 공급 업체 ID로 바꿉니다.**

- & lt;product_id& gt;를 PCI 장치의 제품 ID로 바꿉니다.
- & lt;NIC_address& gt;를 PCI 장치의 주소로 바꿉니다.
- <physical_network> 를 PCI 장치가 있는 물리적 네트워크의 이름으로 교체합니다.



참고

NIC의 장치 이름이 변경될 수 있으므로 PCI 패스스루를 구성할 때 devname 매개변수를 사용하지 마십시오. PF에서 Networking 서비스(neutron) 포트를 생성하려면 vendor_id, product_id 및 PCI 장치 주소를 NovaPCIPassthrough 에서 지정하고 --vnic-type direct-physical 옵션을 사용하여 포트를 만듭니다. 가상 기능(VF)에 네트워킹 서비스 포트를 생성하려면 NovaPCIPassthrough 에서 vendor_id 및 product_id 를 지정하고 --vnic-type direct 옵션을 사용하여 포트를 생성합니다. vendor_id 및 product_id 매개변수의 값은 물리적 기능(PF)과 VF 컨텍스트마다 다를 수 있습니다.

6. 또한 사용자 지정 환경 파일에서 PciPassthroughFilter 및 AggregateInstanceExtraSpecsFiltersFilter 가 NovaSchedulerEnabledFilters 매개변수의 필터 목록에 있는지 확인합니다. 이 매개변수는 Compute 서비스(nova)에서 노드를 필터링하는 데 사용합니다.

```
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. 생성한 사용자 지정 환경 파일의 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 7.3절. “역할별 매개변수 및 구성 덮어쓰기 추가” 으로 이동합니다.

추가 리소스

- *인스턴스 생성을 위해 Compute 서비스 구성에서 NovaPCIPassthrough구성 지침*

7.3. 역할별 매개변수 및 구성 덮어쓰기 추가

SR-IOV 컴퓨팅 노드에 대한 역할별 매개변수를 추가하고 SR-IOV 환경을 배포할 때 RHOSP(Red Hat OpenStack Platform) director에서 사용하는 사용자 정의 환경 YAML 파일에서 기본 구성 값을 덮어쓸 수 있습니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 7.2절. “SR-IOV의 PCI 패스스루 장치 구성” 에서 생성한 사용자 정의 환경 YAML 파일을 열거나 새 파일을 생성합니다.
4. SR-IOV 컴퓨팅 노드의 역할별 매개변수를 사용자 지정 환경 파일에 추가합니다.

예제

```
  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. **RHOSP director가 SR-IOV를 구성하는 데 사용하는 설정 기본값을 검토합니다. 이러한 기본값은 파일에 제공되며 메커니즘 드라이버에 따라 다릅니다.**

   - **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. **구성 기본값을 재정의해야 하는 경우 사용자 지정 환경 파일에 재정의를 추가합니다.**

   예를 들어 이 사용자 지정 환경 파일은 Nova PCI 허용 목록 값을 추가하거나 네트워크 유형을 설정할 수 있는 위치입니다.

   예제

   이 예에서는 Networking 서비스(neutron) 네트워크 유형이 VLAN으로 설정되고 테넌트에 범위가 추가됩니다.

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

7.

새 사용자 지정 환경 파일을 생성한 경우 해당 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 진행 중 7.4절. “SR-IOV에 대한 베어 메탈 노드 정의 파일 생성”

추가 리소스

- *Red Hat OpenStack Platform 배포 가이드 사용자* 정의에서 지원되는 사용자 정의 역할

7.4. SR-IOV에 대한 베어 메탈 노드 정의 파일 생성

RHOSP(Red Hat OpenStack Platform) director 및 정의 파일을 사용하여 SR-IOV 환경의 베어 메탈 노드를 프로비저닝합니다. 베어 메탈 노드 정의 파일에서 배포하려는 베어 메탈 노드의 수량 및 속성을 정의하고 오버클라우드 역할을 이러한 노드에 할당합니다. 노드의 네트워크 레이아웃도 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 베어 메탈 노드 프로비저닝에 지시되는 대로 **overcloud -baremetal-deploy.yaml** 과 같은 베어 메탈 노드 정의 파일을 생성합니다.*

4. 베어 메탈 노드 정의 파일에서 **Ansible** 플레이북 **cli-overcloud-node-kernelargs.yaml** 에 선언을 추가합니다.

플레이북에는 베어 메탈 노드를 프로비저닝할 때 사용할 커널 인수가 포함되어 있습니다.

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

5. 플레이북을 실행할 때 추가 **Ansible** 변수를 설정하려면 **extra_vars** 속성을 사용하여 설정합니다.

참고

extra_vars 에 추가하는 변수는 7.3절. “역할별 매개변수 및 구성 덮어쓰기 추가” 앞의 사용자 정의 환경 파일에 추가한 **SR-IOV** 컴퓨팅 노드의 역할별 매개변수여야 합니다.

예제

```
- 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. 생성한 베어 메탈 노드 정의 파일의 경로와 파일 이름을 기록해 둡니다. 나중에 **NIC**를 구성하고 노드를 프로비저닝할 때 오버클라우드 노드 프로비저닝 명령의 입력 파일로 이 파일을 사용합니다.

다음 단계

- 7.5절. “SR-IOV용 NIC 구성 템플릿 생성” 으로 이동합니다.

추가 리소스

- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 구성 가능 서비스 및 사용자 지정 역할
- NFV에 대해 테스트된 NIC
- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 베어 메탈 노드 프로비저닝 속성

7.5. SR-IOV용 NIC 구성 템플릿 생성

RHOSP(Red Hat OpenStack Platform)와 함께 제공되는 샘플 Jinja2 템플릿의 사본을 수정하여 NIC 구성 템플릿을 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 샘플 네트워크 구성 템플릿을 복사합니다.

   /usr/share/ansible/roles/tripleo_network_config/templates/ 디렉터리의 예제에서 NIC 구성 Jinja2 템플릿을 복사합니다. NIC 요구 사항에 가장 적합한 항목을 선택합니다. 필요에 따라 수정합니다.

4. NIC 구성 템플릿(예: single_nic_vlans.j2)에서 PF 및 VF 인터페이스를 추가합니다. SR-IOV VF를 생성하려면 인터페이스를 독립 실행형 NIC로 구성합니다.

예제

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

참고

numvfs 매개변수는 네트워크 구성 템플릿의 **NeutronSriovNumVFs** 매개변수를 대체합니다. **Red Hat**은 배포 후 **NeutronSriovNumVFs** 매개변수 또는 **numvfs** 매개변수를 수정할 수 없습니다. 배포 후 두 매개변수를 수정하면 해당 **PF**에 **SR-IOV** 포트가 있는 실행 중인 인스턴스가 중단될 수 있습니다. 이 경우 **SR-IOV PCI** 장치를 다시 사용할 수 있도록 이러한 인스턴스를 재부팅해야 합니다.

5. **7.4절. "SR-IOV에 대한 베어 메탈 노드 정의 파일 생성"**에서 생성한 베어 메탈 노드 정의 파일에 사용자 정의 네트워크 구성 템플릿을 추가합니다.

예제

```
- 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. 생성한 NIC 구성 템플릿의 경로와 파일 이름을 기록해 둡니다. NIC를 분할하려면 나중에 이 파일을 사용합니다.

다음 단계

1. NIC를 분할하려면 7.6절. “NIC 파티셔닝 구성” 으로 이동합니다.

2. 그렇지 않으면 다음 단계를 수행합니다.

   a. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝

   b. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 네트워크 VIP 구성 및 프로비저닝

   c. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 베어 메탈 노드 프로비저닝

   d. 7.8절. “SR-IOV 오버클라우드 배포”

7.6. NIC 파티셔닝 구성

RHOSP(Red Hat OpenStack Platform) 관리 네트워크 및 공급자 네트워크에 대해 SR-IOV(Single Root I/O Virtualization) 가상 기능(VF)을 구성하여 각 호스트에 필요한 NIC 수를 줄일 수 있습니다. 단일 고속 NIC를 여러 VF로 분할하면 컨트롤 및 데이터 플레인 트래픽에 NIC를 사용할 수 있습니다. 이 기능은 Intel Fortville NIC 및 Mellanox CX-5 NIC에서 검증되었습니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.
- NIC, 애플리케이션, VF 게스트 및 OVS가 동일한 NUMA 컴퓨팅 노드에 있는지 확인합니다.

이렇게 하면 **NUMA** 간 작업에서 성능이 저하되는 것을 방지할 수 있습니다.

- **NIC** 펌웨어를 계속 업데이트해야 합니다.

  yum 또는 **dnf** 업데이트는 펌웨어 업데이트를 완료하지 못할 수 있습니다. 자세한 내용은 공급 업체 설명서를 참조하십시오.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 이전에 **7.5절. "SR-IOV용 NIC 구성 템플릿 생성"** 에서 생성한 **NIC** 구성 템플릿(예: **single_nic_vlans.j2**)을 엽니다.

   작은 정보

   이 섹션의 단계를 완료하면 **7.7절. "NIC 파티션 구성 예"** 를 참조하십시오.

4. 인터페이스 유형 **sriov_pf** 에 대한 항목을 추가하여 호스트에서 사용할 수 있는 물리적 기능을 구성합니다.

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

   - **& lt;interface_name >**을 인터페이스 이름으로 바꿉니다.
   - **& lt;number_of_vfs& gt;**를 **VF** 수로 바꿉니다.
   -

선택 사항: 불규칙 모드를 설정하려면 < true /false >를 true로 교체하거나 false 를 사용하여 무차별 모드를 비활성화합니다. 기본값은 true입니다.

참고

numvfs 매개변수는 네트워크 구성 템플릿의 NeutronSriovNumVFs 매개변수를 대체합니다. Red Hat은 배포 후 NeutronSriovNumVFs 매개변수 또는 numvfs 매개변수를 수정할 수 없습니다. 배포 후 두 매개변수를 수정하면 해당 물리적 기능(PF)에 SR-IOV 포트가 있는 실행 중인 인스턴스가 중단될 수 있습니다. 이 경우 SR-IOV PCI 장치를 다시 사용할 수 있도록 이러한 인스턴스를 재부팅해야 합니다.

5. 인터페이스 유형 sriov_vf 항목을 추가하여 호스트에서 사용할 수 있는 가상 기능을 구성합니다.

```
- 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
```

- &lt ;bond_type >을 필수 본딩 유형(예: linux_bond)으로 바꿉니다. ovs_bond 와 같은 다른 본딩의 경우 본딩에 VLAN 태그를 적용할 수 있습니다.

- & lt;bonding_option& gt;을 다음과 같은 본딩 모드 중 하나로 바꿉니다.

- **active-backup**
- **balance-slb**

참고

LACP 본딩은 지원되지 않습니다.

- **members** 섹션에서 **sriov_vf** 를 본딩할 인터페이스 유형으로 지정합니다.

참고

OVS 브릿지를 인터페이스 유형으로 사용하는 경우 **sriov_pf** 장치의 **sriov_vf** 에서 하나의 **OVS** 브리지만 구성할 수 있습니다. 단일 **sriov_pf** 장치의 **OVS** 브리지가 두 개 이상 있으면 **VF** 간에 패킷 중복이 발생하고 성능이 저하될 수 있습니다.

- **& lt;pf_device_name& gt;**을 **PF** 장치의 이름으로 바꿉니다.
- **linux_bond** 를 사용하는 경우 **VLAN** 태그를 할당해야 합니다. **VLAN** 태그를 설정하는 경우 단일 **sriov_pf** 장치와 연결된 각 **VF**에 대해 고유한 태그를 설정해야 합니다. 동일한 **VLAN**에 동일한 **PF**의 두 개의 **VF**가 있을 수 없습니다.
- **& lt;vf_id& gt;**를 **VF ID**로 바꿉니다. 해당 **VF ID** 범위는 **0**에서 시작하여 최대 **VF** 수에서 **1**을 뺀 값으로 끝납니다.
- 스푸핑 검사를 비활성화합니다.
- **VF**를 통해 **linux_bond** 의 **sriov_vf** 에 **VLAN** 태그를 적용합니다.

6. 인스턴스의 **VF**를 예약하려면 환경 파일에 **NovaPCIPassthrough** 매개변수를 포함합니다.

예제

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

RHOSP director는 호스트 **VF**를 식별하고 인스턴스에서 사용할 수 있는 **VF**의 **PCI** 주소를 파생합니다.

7. **NIC** 파티셔닝이 필요한 모든 노드에서 **IOMMU** 를 활성화합니다.

예제

예를 들어 컴퓨팅 노드에 **NIC** 파티셔닝을 사용하려면 해당 역할에 **KernelArgs** 매개변수를 사용하여 **IOMMU**를 활성화합니다.

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

참고

먼저 **KernelArgs** 매개변수를 역할 구성에 추가하면 오버클라우드 노드가 자동으로 재부팅됩니다. 필요한 경우 노드의 자동 재부팅을 비활성화하고 대신 오버클라우드 배포 후 노드를 수동으로 재부팅할 수 있습니다.

8. 이 **NIC** 구성 템플릿(예: **single_nic_vlans.j2**)을 **7.4**절. “**SR-IOV**에 대한 베어 메탈 노드 정의 파일 생성” 에서 생성한 베어 메탈 노드 정의 파일에 추가해야 합니다.

다음 단계

1. ***director*** *가이드를 사용하여* ***Red Hat OpenStack Platform*** *설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝

2. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 네트워크 VIP 구성 및 프로비저닝

3. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 베어 메탈 노드 프로비저닝

4. 7.8절. “SR-IOV 오버클라우드 배포”

추가 리소스

- 7.7절. “NIC 파티션 구성 예”

7.7. NIC 파티션 구성 예

Red Hat OpenStack Platform SR-IOV 환경에서 NIC를 분할하려면 다음 예제 구성을 참조하십시오.

VF를 통한 Linux 본딩

다음 예제에서는 VF를 통한 Linux 본딩을 구성하고 스푸핑 검사를 비활성화하고 VLAN 태그를 sriov_vf 에 적용합니다.

```
- 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
```

VF의 OVS 브리지

다음 예제는 **VF**에서 **OVS** 브리지를 구성합니다.

```
- 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
```

VF의 OVS 사용자 브릿지

다음 예제는 **VF**에서 **OVS** 사용자 브릿지를 구성하고 **VLAN** 태그를 **ovs_user_bridge** 에 적용합니다.

```
- 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
```

추가 리소스

- 7.6절. “NIC 파티셔닝 구성”

7.8. SR-IOV 오버클라우드 배포

SR-IOV 환경에서 RHOSP(Red Hat OpenStack Platform) 오버클라우드를 구성하는 마지막 단계는 openstack overcloud deploy 명령을 실행하는 것입니다. 명령에 대한 입력에는 사용자가 구성한 모든 다양한 오버클라우드 템플릿 및 환경 파일이 포함됩니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.
- 이 섹션의 이전 절차에 나열된 모든 단계를 수행하고 **overcloud deploy** 명령에 입력으로 사용할 다양한 **heat** 템플릿 및 환경 파일을 모두 어셈블했습니다.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.
2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **openstack overcloud deploy** 명령을 입력합니다.

openstack overcloud deploy 명령에 대한 입력을 특정 순서로 나열하는 것이 중요합니다. 일반 규칙은 먼저 기본 **heat** 템플릿 파일 뒤에 사용자 지정 환경 파일 및 사용자 지정 구성이 포함된 사용자 지정 템플릿을 지정합니다(예: 기본 속성 재정의).

다음 순서로 **openstack overcloud deploy** 명령에 입력을 추가합니다.

a. 오버클라우드의 **SR-IOV** 네트워크의 사양이 포함된 사용자 지정 네트워크 정의 파일(예: **network-data.yaml**)

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 네트워크 정의 파일 구성 옵션을 참조하십시오.

b. **RHOSP director**에서 **OVS** 하드웨어 오프로드 환경을 배포하는 데 사용하는 **Controller** 및 **ComputeOvsHwOffload** 역할이 포함된 역할 파일입니다.

예: **roles_data_compute_sriov.yaml**

자세한 내용은 **7.1**절. **"SR-IOV**의 역할 및 이미지 파일 생성**"**의 내용을 참조하십시오.

c. 오버클라우드 네트워크를 프로비저닝한 출력 파일입니다.

예: **overcloud-networks-deployed.yaml**

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝 을 참조하십시오.

d. 오버클라우드 **VIP** 프로비저닝의 출력 파일입니다.

예: **overcloud-vip-deployed.yaml**

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 네트워크 VIP 구성 및 프로비저닝* 을 참조하십시오.

e. **베어 메탈 노드 프로비저닝의 출력 파일입니다.**

예: overcloud-baremetal-deployed.yaml

*자세한 내용은 **director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리** 에서* 오버클라우드의 베어 메탈 노드 프로비저닝 을 참조하십시오.

f. **director에서 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 데 사용하는 이미지 파일입니다.**

예: overcloud_images.yaml

자세한 내용은 **7.1절. "SR-IOV의 역할 및 이미지 파일 생성"의 내용을 참조하십시오.**

g. **환경에서 사용하는 Networking 서비스(neutron) 메커니즘 드라이버 및 라우터 체계용 환경 파일입니다.**

- **ML2/OVN**
  - **DCVR(Distributed virtual routing): neutron-ovn-dvr-ha.yaml**
  - **중앙 집중식 가상 라우팅: neutron-ovn-ha.yaml**
- **ML2/OVS**
  - **DCVR(Distributed virtual routing): neutron-ovs-dvr.yaml**
  - **중앙 집중식 가상 라우팅: neutron-ovs.yaml**

h. **메커니즘 드라이버에 따라 SR-IOV의 환경 파일입니다.**

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

참고

OVS-DPDK 환경도 있고 동일한 노드에서 **OVS-DPDK** 및 **SR-IOV** 인스턴스를 찾으려면 배포 스크립트에 다음 환경 파일을 포함합니다.

- **ML2/OVN**

  neutron-ovn-dpdk.yaml
- **ML2/OVS**

  neutron-ovs-dpdk.yaml

i. **다음과 같은 구성이 포함된 하나 이상의 사용자 지정 환경 파일입니다.**

- **SR-IOV 노드의 PCI 패스스루 장치입니다.**
- **SR-IOV 노드의 역할별 매개변수**
- **SR-IOV 환경의 기본 구성 값을 덮어씁니다.**

  예: sriov-overrides.yaml

자세한 내용은 다음을 참조하십시오.

- 7.2절. “SR-IOV의 PCI 패스스루 장치 구성”.
- 7.3절. “역할별 매개변수 및 구성 덮어쓰기 추가”.

예제

샘플 **openstack overcloud deploy** 명령에서 발췌한 내용은 **DVR**을 사용하는 **SR-IOV, ML2/OVN** 환경에 대한 명령 입력 순서를 올바르게 정렬하는 방법을 보여줍니다.

```
$ 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. **openstack overcloud deploy** 명령을 실행합니다.

오버클라우드 생성이 완료되면 **RHOSP director**에서 오버클라우드 액세스에 도움이 되는 세부 정보를 제공합니다.

검증

1. *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 배포 검증 단계를 수행합니다.

2. **NIC**가 올바르게 분할되었는지 확인하려면 다음을 수행하십시오.

a.

오버클라우드 컴퓨팅 노드에 **tripleo-admin** 으로 로그인하고 **VF** 수를 확인합니다.

예제

이 예에서 **p4p1** 및 **p4p2** 모두에 대한 **VF** 수는 **10** 입니다.

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

10

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

10
```

b.

OVS 연결을 표시합니다.

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

샘플 출력

출력은 다음과 유사합니다.

```
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. **SR-IOV 컴퓨팅 노드에 tripleo-admin 으로 로그인하고 Linux 본딩을 확인합니다.**

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

샘플 출력

출력은 다음과 유사합니다.

```
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. **OVS 본딩을 나열합니다.**

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

샘플 출력

출력은 다음과 유사합니다.

```
---- 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. **NovaPCIPassthrough** 를 사용하여 인스턴스에 **VF**를 전달하는 경우 **SR-IOV** 인스턴스를 배포하여 테스트합니다.

추가 리소스

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 생성
- *명령줄 인터페이스 참조에* **overcloud deploy**
- **7.10절. "SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 인스턴스 생성"**

7.9. SR-IOV 또는 OVS TC-FLOWER 하드웨어 오프로드 환경에서 호스트 집계 생성

RHOSP(Red Hat OpenStack Platform) SR-IOV 또는 **OVS TC-flower** 하드웨어 오프로드 환경에서 더 나은 성능을 얻으려면 **CPU** 고정 및 대규모 페이지가 있는 게스트를 배포합니다. 집계 메타데이터와 플레이버 메타데이터를 일치시켜 호스트 하위 집합에서 고성능 인스턴스를 예약할 수 있습니다.

사전 요구 사항

- **SR-IOV** 또는 **OVS** 하드웨어 오프로드 환경에 구성된 **RHOSP** 오버클라우드입니다.
- **RHOSP** 오버클라우드는 **AggregateInstanceExtraSpecsFilter** 용으로 구성해야 합니다.

  자세한 내용은 **7.2절. "SR-IOV의 PCI 패스스루 장치 구성"**의 내용을 참조하십시오.

절차

1. 집계 그룹을 생성하고 관련 호스트를 추가합니다.

   정의된 플레이버 메타데이터와 일치하는 메타데이터(예: **sriov=true**)를 정의합니다.

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

2. 플레이버를 만듭니다.

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

3. 추가 플레이버 속성을 설정합니다.

   정의된 메타데이터 **sriov=true** 는 **SR-IOV** 집계의 정의된 메타데이터와 일치합니다.

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

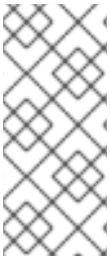
추가 리소스

- *명령줄 인터페이스 참조에* 집계
- *명령줄 인터페이스 참조의* 플레이버

7.10. SR-IOV 또는 OVS TC-FLOWER 하드웨어 오프로드 환경에서 인스턴스 생성

여러 명령을 사용하여 **RHOSP(Red Hat OpenStack Platform) SR-IOV** 또는 **OVS TC-flower** 하드웨어 오프로드 환경에서 인스턴스를 생성합니다.

호스트 집계를 사용하여 고성능 컴퓨팅 호스트를 분리합니다. 자세한 내용은 **7.9절. "SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 호스트 집계 생성"**의 내용을 참조하십시오.



참고

고정된 **CPU** 인스턴스는 고정되지 않은 인스턴스와 동일한 컴퓨팅 노드에 있을 수 있습니다. 자세한 내용은 *인스턴스 생성 가이드를 위한 컴퓨팅 서비스 구성 가이드의 컴퓨팅 노드에 CPU 고정 구성*을 참조하십시오.

사전 요구 사항

- SR-IOV 또는 OVS 하드웨어 오프로드 환경에 구성된 RHOSP 오버클라우드입니다.

절차

1. 플레이버를 만듭니다.

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

작은 정보

플레이버에 spec hw:pci_numa_affinity_policy 를 추가하여 PCI 패스스루 장치 및 SR-IOV 인터페이스에 대한 NUMA 선호도 정책을 지정할 수 있습니다. 자세한 내용은 *인스턴스 생성을 위해 Compute 서비스 구성의* 플레이버 메타데이터 를 참조하십시오.

2. 네트워크 및 서브넷을 생성합니다.

```
$ 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. VF(가상 기능) 포트 또는 물리적 기능(PF) 포트를 생성합니다.

- VF 포트:

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

- 단일 인스턴스 전용 PF 포트:

이 PF 포트는 Networking 서비스(neutron) 포트이지만 Networking 서비스에 의해 제어되지 않으며, 인스턴스에 전달되는 PCI 장치이므로 네트워크 어댑터로 표시되지 않습니다.

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

4. **인스턴스를 만듭니다.**

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

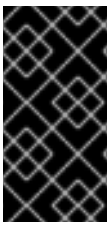
추가 리소스

- *명령줄 인터페이스 참조*에서 **플레이버 생성**
- *명령줄 인터페이스 참조*에서 **네트워크 생성**
- *명령줄 인터페이스 참조*에서 **subnet create**
- *명령줄 인터페이스 참조*에서 **port create**
- *명령줄 인터페이스 참조*에서 **server create**

8장. OVS TC-FLOWER 하드웨어 오프로드 구성

RHOSP(Red Hat OpenStack Platform) 네트워크 기능 가상화(NFV) 배포에서 OVS(Open vSwitch) TC-flower 하드웨어 오프로드를 사용하여 더 높은 성능을 얻을 수 있습니다. 하드웨어 오프로드는 네트워크 인터페이스 컨트롤러(NIC)의 전용 프로세서로 네트워킹 작업을 CPU에서 분리합니다. 이러한 특수 하드웨어 리소스는 CPU가 더 중요한 컴퓨팅 작업을 수행할 수 있는 추가 컴퓨팅 기능을 제공합니다.

OVS 하드웨어 오프로드를 위한 RHOSP 구성은 SR-IOV용 RHOSP 구성과 유사합니다.



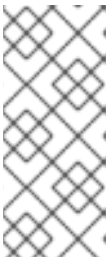
중요

이 섹션에는 토폴로지 및 기능 요구 사항에 맞게 수정해야 하는 예제가 포함되어 있습니다. 자세한 내용은 NFV 하드웨어 요구 사항을 참조하십시오.

사전 요구 사항

- RHOSP 언더클라우드.

  오버클라우드를 배포하려면 언더클라우드를 설치하고 구성해야 합니다. 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리를* 참조하십시오.

  참고

  RHOSP director는 director 템플릿 및 사용자 지정 환경 파일에서 지정하는 키-값 쌍을 통해 OVS 하드웨어 오프로드 구성 파일을 수정합니다. OVS 하드웨어 오프로드 구성 파일을 직접 수정하지 않아야 합니다.

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

- NIC, 해당 애플리케이션, VF 게스트 및 OVS가 동일한 NUMA 컴퓨팅 노드에 있는지 확인합니다.

  이렇게 하면 NUMA 간 작업에서 성능이 저하되는 것을 방지할 수 있습니다.

- NIC가 포함된 호스트에서 sudo에 액세스합니다.

- NIC 펌웨어를 계속 업데이트해야 합니다.

  yum 또는 dnf 업데이트는 펌웨어 업데이트를 완료하지 못할 수 있습니다. 자세한 내용은 공급업체 설명서를 참조하십시오.

- 연결 추적(conntrack) 모듈에 switchdev 포트에서 보안 그룹 및 포트 보안을 활성화하여 OpenFlow 흐름을 하드웨어로 오프로드합니다.

절차

RHOSP director를 사용하여 OVS 하드웨어 오프로드 환경에서 RHOSP를 설치하고 설정합니다. 높은 수준의 단계는 다음과 같습니다.

1. *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의 지침에 따라 오버클라우드의 물리적 네트워크를 설정하기 위해 네트워크 설정 파일 network_data.yaml 을 생성합니다.* https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_with_director/assembly_configuring-overcloud-networking_installing-director-on-the-undercloud

2. 역할 및 이미지 파일을 생성합니다.

3. OVS 하드웨어 오프로드에 대해 PCI 패스스루 장치를 구성합니다.

4. 역할별 매개변수 및 기타 구성 덮어쓰기를 추가합니다.

5. 베어 메탈 노드 정의 파일을 생성합니다.

6. OVS 하드웨어 오프로드에 대한 NIC 구성 템플릿을 생성합니다.

7. 오버클라우드 네트워크 및 VIP를 프로비저닝합니다.

   자세한 내용은 다음을 참조하십시오.

   -

director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드 네트워크 정의 구성 및 프로비저닝.

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 네트워크 VIP 구성 및 프로비저닝.

8. 오버클라우드 베어 메탈 노드를 프로비저닝합니다.

   자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드의 베어 메탈 노드 프로비저닝 을 참조하십시오.

9. OVS 하드웨어 오프로드 오버클라우드를 배포합니다.

추가 리소스

- 8.7절. “SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 호스트 집계 생성”
- 8.8절. “SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 인스턴스 생성”
- 8.9절. “OVS TC-flower 하드웨어 오프로드 문제 해결”
- 8.10절. “TC-flower 하드웨어 오프로드 흐름 디버깅”

8.1. OVS TC-FLOWER 하드웨어 오프로드의 역할 및 이미지 파일 생성

RHOSP(Red Hat OpenStack Platform) director는 역할을 사용하여 노드에 서비스를 할당합니다. OVS TC-flower 하드웨어 오프로드 환경에서 RHOSP를 구성할 때 RHOSP 설치와 함께 제공되는 기본 역할 Compute 를 기반으로 하는 새 역할을 생성합니다.

언더클라우드 설치에는 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 환경 파일이 필요합니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **Compute** 역할을 기반으로 **OVS** 하드웨어 오프로드에 대한 오버클라우드 역할을 생성합니다.

예제

이 예에서는 **Compute** 역할을 기반으로 하는 **ComputeOvsHwOffload** 역할이 생성됩니다. 명령이 생성하는 역할 파일의 이름은 **roles_data_compute_ovshwol.yaml** 입니다.

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

참고

RHOSP 환경에 **OVS-DPDK, SR-IOV** 및 **OVS TC-flower** 하드웨어 오프로드 기술이 혼합된 경우 **roles_data.yaml** 과 같은 하나의 역할 데이터 파일만 생성하여 모든 역할을 포함합니다.

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

4. (선택 사항) **ComputeOvsHwOffload** 역할의 **HostnameFormatDefault: '%stackname%-compute-%index%'** 이름을 변경합니다.

5. 이미지 파일을 생성하려면 **openstack tripleo container image prepare** 명령을 실행합니다. 다음 입력이 필요합니다.

- 이전 단계에서 생성한 역할 데이터 파일(예: **roles_data_compute_ovshwol.yaml**)
- 네트워킹 서비스 메커니즘 드라이버에 적합한 **SR-IOV** 환경 파일입니다.
  - **ML2/OVN** 환경

    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml
  - **ML2/OVS** 환경

    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml

    예제

    이 예에서는 **ML2/OVN** 환경에 대해 **overcloud_images.yaml** 파일이 생성됩니다.

```
$ 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. 역할 데이터 파일의 경로 및 파일 이름과 사용자가 생성한 이미지 파일을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이러한 파일을 사용합니다.

다음 단계

- **8.2절. "OVS TC-flower 하드웨어 오프로드를 위한 PCI 패스스루 장치 구성"** 으로 이동합니다.

추가 리소스

- 자세한 내용은 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리의 Composable 서비스 및 사용자 지정 역할**을 참조하십시오.

- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 컨테이너 이미지 준비.

8.2. OVS TC-FLOWER 하드웨어 오프로드를 위한 PCI 패스스루 장치 구성

OVS TC-flower 하드웨어 오프로드 환경을 위한 Red Hat OpenStack Platform을 배포할 때 사용자 지정 환경 파일에서 컴퓨팅 노드에 대해 PCI 패스스루 장치를 구성해야 합니다.

사전 요구 사항

- PCI 카드가 포함된 하나 이상의 물리적 서버에 액세스합니다.
- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

절차

1. PCI 카드가 포함된 물리적 서버에서 다음 명령 중 하나를 사용합니다.
   - 오버클라우드가 배포된 경우:

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

샘플 출력

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

   - 오버클라우드가 배포되지 않은 경우:

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

2. ComputeOvsHwOffload 노드에서 PCI 패스스루 장치의 벤더 및 제품 ID를 확인합니다. 이후

단계에서 이러한 **ID**가 필요합니다.

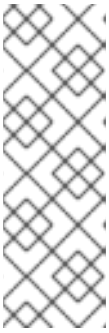
3. **stack** 사용자로 언더클라우드에 로그인합니다.

4. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

5. 사용자 지정 환경 **YAML** 파일(예**: ovshwol-overrides.yaml**)을 만듭니다. 파일에 다음 콘텐츠를 추가하여 컴퓨팅 노드의 **PCI** 패스스루 장치를 구성합니다.

```
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
    ...
```

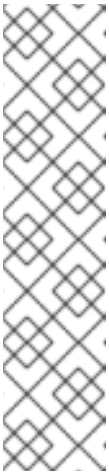


참고

Mellanox 스마트 **NIC**를 사용하는 경우 **ComputeOvsHwOffloadloadParameters** 매개변수 아래에 **DerivePciWhitelistEnabled: true** 를 추가합니다. **OVS** 하드웨어 오프로드를 사용하는 경우 **Compute** 서비스**(nova)** 스케줄러는 인스턴스 생성을 위해 **SR-IOV** 패스와 유사하게 작동합니다.

- **&lt ;vendor_id>**를 **PCI** 장치의 공급 업체 **ID**로 바꿉니다.

- & lt;product_id& gt;를 PCI 장치의 제품 ID로 바꿉니다.
- & lt;NIC_address& gt;를 PCI 장치의 주소로 바꿉니다.
- <physical_network> 를 PCI 장치가 있는 물리적 네트워크의 이름으로 교체합니다.
- VLAN의 경우 배포 후 physical_network 매개변수를 neutron에서 생성한 네트워크의 이름으로 설정합니다. 이 값은 NeutronBridgeMappings 에도 있어야 합니다.
- VXLAN의 경우 physical_network 매개 변수를 null 로 설정합니다.



참고

NIC의 장치 이름이 변경될 수 있으므로 PCI 패스스루를 구성할 때 devname 매개변수를 사용하지 마십시오. PF에서 Networking 서비스(neutron) 포트를 생성하려면 vendor_id, product_id 및 PCI 장치 주소를 NovaPCIPassthrough 에서 지정하고 --vnic-type direct-physical 옵션을 사용하여 포트를 만듭니다. 가상 기능(VF)에 네트워킹 서비스 포트를 생성하려면 NovaPCIPassthrough 에서 vendor_id 및 product_id 를 지정하고 --vnic-type direct 옵션을 사용하여 포트를 생성합니다. vendor_id 및 product_id 매개변수의 값은 물리적 기능(PF)과 VF 컨텍스트마다 다를 수 있습니다.

6. 사용자 지정 환경 파일에서 PciPassthroughFilter 및 NUMATopologyFilter 가 NovaSchedulerEnabledFilters 매개변수의 필터 목록에 있는지 확인합니다. Compute 서비스(nova)는 이 매개변수를 사용하여 노드를 필터링합니다.

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

참고

선택 사항: Mellanox ConnectX5 NIC를 사용하여 RHOSP 17.1에서 OVS 하드웨어 오프로드 문제를 해결하고 구성하는 방법에 대한 자세한 내용은 Hardware Offload 문제 해결을 참조하십시오.

7. 생성한 사용자 지정 환경 파일의 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 8.3절. "OVS TC-flower 하드웨어 오프로드에 대한 역할별 매개변수 및 구성 덮어쓰기 추가"으로 이동합니다.

추가 리소스

- *인스턴스 생성을 위해 Compute 서비스 구성에서 NovaPCIPassthrough구성 지침*

8.3. OVS TC-FLOWER 하드웨어 오프로드에 대한 역할별 매개변수 및 구성 덮어쓰기 추가

ComputeOvsHwOffload 노드에 대한 역할별 매개변수를 추가하고 RHOSP(Red Hat OpenStack Platform) director가 OVS TC-flower 하드웨어 오프로드 환경을 배포할 때 사용하는 사용자 지정 환경 YAML 파일의 기본 구성 값을 덮어쓸 수 있습니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 8.2절. "OVS TC-flower 하드웨어 오프로드를 위한 PCI 패스스루 장치 구성"에서 생성한 사

용자 정의 환경 **YAML** 파일을 열거나 새 파일을 생성합니다.

4. **ComputeOvsHwOffload** 노드의 역할별 매개 변수를 사용자 지정 환경 파일에 추가합니다.

예제

```
 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. 값이 **true** 인 역할별 매개변수 아래에 **OvsHwOffload** 매개변수를 추가합니다.

```
 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. **RHOSP director**에서 **OVS** 하드웨어 오프로드를 설정하는 데 사용하는 설정 기본값을 검토합니다. 이러한 기본값은 파일에 제공되며 메커니즘 드라이버에 따라 다릅니다.

- **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. 구성 기본값을 재정의해야 하는 경우 사용자 지정 환경 파일에 재정의를 추가합니다.

   예를 들어 이 사용자 지정 환경 파일은 **Nova PCI** 허용 목록 값을 추가하거나 네트워크 유형을 설정할 수 있는 위치입니다.

   예제

   이 예에서는 **Networking** 서비스(neutron) 네트워크 유형이 **VLAN**으로 설정되고 테넌트에 범위가 추가됩니다.

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

8. 새 사용자 지정 환경 파일을 생성한 경우 해당 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 진행 중 8.4절. “OVS TC-flower 하드웨어 오프로드에 대한 베어 메탈 노드 정의 파일 생성”

추가 리소스

- *Red Hat OpenStack Platform 배포 가이드 사용자* 정의에서 지원되는 사용자 정의 역할

8.4. OVS TC-FLOWER 하드웨어 오프로드에 대한 베어 메탈 노드 정의 파일 생성

RHOSP(Red Hat OpenStack Platform) director 및 정의 파일을 사용하여 **OVS TC-flower** 하드웨어 오프로드 환경의 베어 메탈 노드를 프로비저닝합니다. 베어 메탈 노드 정의 파일에서 배포하려는 베어 메탈 노드의 수량 및 속성을 정의하고 오버클라우드 역할을 이러한 노드에 할당합니다. 노드의 네트워크 레이아웃도 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. ***director** 가이드를 사용하여 **Red Hat OpenStack Platform** 설치 및 관리에서 오버클라우드의 베어 메탈 노드 프로비저닝에 지시되는 대로 **overcloud -baremetal-deploy.yaml** 과 같은 베어 메탈 노드 정의 파일을 생성합니다.*

4. 베어 메탈 노드 정의 파일에서 **Ansible** 플레이북 **cli-overcloud-node-kernelargs.yaml** 에 선언을 추가합니다.

   플레이북에는 베어 메탈 노드를 프로비저닝할 때 사용할 커널 인수가 포함되어 있습니다.

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

5. 플레이북을 실행할 때 추가 **Ansible** 변수를 설정하려면 **extra_vars** 속성을 사용하여 설정합니다.

참고

extra_vars 에 추가하는 변수는 8.3절. "OVS TC-flower 하드웨어 오프로드에 대한 역할별 매개변수 및 구성 덮어쓰기 추가" 의 이전 사용자 지정 환경 파일에 추가한 **ComputeOvsHwOffload** 노드의 동일한 역할별 매개변수여야 합니다.

예제

```
- 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. 생성한 베어 메탈 노드 정의 파일의 경로와 파일 이름을 기록해 둡니다. 나중에 NIC를 구성하고 노드를 프로비저닝할 때 오버클라우드 노드 프로비저닝 명령의 입력 파일로 이 파일을 사용합니다.

다음 단계

- 8.5절. “OVS TC-flower 하드웨어 오프로드에 대한 NIC 구성 템플릿 생성” 으로 이동합니다.

추가 리소스

- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 구성 가능 서비스 및 사용자 지정 역할
- NFV에 대해 테스트된 NIC
- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 베어 메탈 노드 프로비저닝 속성

8.5. OVS TC-FLOWER 하드웨어 오프로드에 대한 NIC 구성 템플릿 생성

RHOSP(Red Hat OpenStack Platform)와 함께 제공되는 샘플 Jinja2 템플릿의 사본을 수정하여 OVS TC-flower 하드웨어 오프로드 환경에 대한 NIC 구성 템플릿을 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

- **NIC, 해당 애플리케이션, VF 게스트 및 OVS가 동일한 NUMA 컴퓨팅 노드에 있는지 확인합니다.**

  이렇게 하면 NUMA 간 작업에서 성능이 저하되는 것을 방지할 수 있습니다.

절차

1. **stack 사용자로 언더클라우드에 로그인합니다.**

2. **stackrc 파일을 소싱합니다.**

```
$ source ~/stackrc
```

3. **샘플 네트워크 구성 템플릿을 복사합니다.**

   /usr/share/ansible/roles/tripleo_network_config/templates/ 디렉터리의 예제에서 NIC 구성 Jinja2 템플릿을 복사합니다. NIC 요구 사항에 가장 적합한 항목을 선택합니다. 필요에 따라 수정합니다.

4. **NIC 구성 템플릿(예: single_nic_vlans.j2)에서 PF 및 VF 인터페이스를 추가합니다. VF를 생성하려면 인터페이스를 독립 실행형 NIC로 구성합니다.**

   예제

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

참고

numvfs 매개변수는 네트워크 구성 템플릿의 **NeutronSriovNumVFs** 매개변수를 대체합니다. **Red Hat**은 배포 후 **NeutronSriovNumVFs** 매개변수 또는 **numvfs** 매개변수를 수정할 수 없습니다. 배포 후 두 매개변수를 수정하면 해당 **PF**에 **SR-IOV** 포트가 있는 실행 중인 인스턴스가 중단될 수 있습니다. 이 경우 **SR-IOV PCI** 장치를 다시 사용할 수 있도록 이러한 인스턴스를 재부팅해야 합니다.

5. **8.4절. "OVS TC-flower 하드웨어 오프로드에 대한 베어 메탈 노드 정의 파일 생성"**에서 생성한 베어 메탈 노드 정의 파일에 사용자 정의 네트워크 구성 템플릿을 추가합니다.

예제

```
- 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. **compute-sriov.yaml** 구성 파일에서 하드웨어 오프로드를 위한 하나 이상의 네트워크 인터페이스를 구성합니다.

```
  - 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
```

참고

- **OVS** 하드웨어 오프로드를 구성할 때 **NeutronSriovNumVFs** 매개변수를 사용하지 마십시오. 가상 함수 수는 **os-net-config** 에서 사용하는 네트워크 구성 파일의 **numvfs** 매개 변수를 사용하여 지정됩니다. **Red Hat**은 업데이트 또는 재배포 중에 **numvfs** 설정을 수정하는 것을 지원하지 않습니다.

- **Mellanox** 네트워크 인터페이스를 **nic-config** 인터페이스 유형 **ovs-vlan** 으로 구성하지 마십시오. 이렇게 하면 **VXLAN**과 같은 터널 끝점이 드라이버 제한으로 인해 트래픽을 전달하지 못합니다.

7. 생성한 **NIC** 구성 템플릿의 경로와 파일 이름을 기록해 둡니다. **NIC**를 분할하려면 나중에 이 파일을 사용합니다.

다음 단계

1. 오버클라우드 네트워크를 프로비저닝합니다.

   자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드 네트워크 정의 구성 및 프로비저닝* 을 참조하십시오.

2. 오버클라우드 **VIP**를 프로비저닝합니다.

   자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 네트워크 VIP 구성 및 프로비저닝을 참조하십시오.*

3. 베어 메탈 노드를 프로비저닝합니다.

   자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 베어 메탈 노드 프로비저닝* 을 참조하십시오.

4. 오버클라우드를 배포합니다.

   자세한 내용은 **8.6**절. **“OVS TC-flower** 하드웨어 오프로드 오버클라우드 배포**”**의 내용을 참조하십시오.

8.6. OVS TC-FLOWER 하드웨어 오프로드 오버클라우드 배포

OVS TC-flower 하드웨어 오프로드 환경에서 RHOSP(Red Hat OpenStack Platform) 오버클라우드를 배포하는 마지막 단계는 openstack overcloud deploy 명령을 실행하는 것입니다. 명령에 대한 입력에는 사용자가 구성한 모든 다양한 오버클라우드 템플릿 및 환경 파일이 포함됩니다.

사전 요구 사항

- **언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.**
- **NIC가 포함된 호스트에서 sudo에 액세스합니다.**
- **이 섹션의 이전 절차에 나열된 모든 단계를 수행하고 overcloud deploy 명령에 입력으로 사용할 다양한 heat 템플릿 및 환경 파일을 모두 어셈블했습니다.**

절차

1. **언더클라우드 호스트에 stack 사용자로 로그인합니다.**
2. **stackrc 언더클라우드 인증 정보 파일을 소싱합니다.**

```
$ source ~/stackrc
```

3. **openstack overcloud deploy 명령을 입력합니다.**

openstack overcloud deploy 명령에 대한 입력을 특정 순서로 나열하는 것이 중요합니다. 일반 규칙은 먼저 기본 heat 템플릿 파일 뒤에 사용자 지정 환경 파일 및 사용자 지정 구성이 포함된 사용자 지정 템플릿을 지정합니다(예: 기본 속성 재정의).

다음 순서로 openstack overcloud deploy 명령에 입력을 추가합니다.

a. **오버클라우드의 SR-IOV 네트워크의 사양이 포함된 사용자 지정 네트워크 정의 파일(예: network-data.yaml)**

자세한 내용은 director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리의 네트워크 정의 파일 구성 옵션을 **참조하십시오.**

b.

RHOSP director에서 OVS 하드웨어 오프로드 환경을 배포하는 데 사용하는 Controller 및 ComputeOvsHwOffload 역할이 포함된 역할 파일입니다.

예: roles_data_compute_ovshwol.yaml

자세한 내용은 8.1절. "OVS TC-flower 하드웨어 오프로드의 역할 및 이미지 파일 생성"의 내용을 참조하십시오.

c.

오버클라우드 네트워크를 프로비저닝한 출력 파일입니다.

예: overcloud-networks-deployed.yaml

자세한 내용은 director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드 네트워크 정의 구성 및 프로비저닝 을 참조하십시오.

d.

오버클라우드 VIP 프로비저닝의 출력 파일입니다.

예: overcloud-vip-deployed.yaml

자세한 내용은 director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 네트워크 VIP 구성 및 프로비저닝 을 참조하십시오.

e.

베어 메탈 노드 프로비저닝의 출력 파일입니다.

예: overcloud-baremetal-deployed.yaml

자세한 내용은 director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 베어 메탈 노드 프로비저닝 을 참조하십시오.

f.

director에서 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 데 사용하는 이미지 파일입니다.

예: overcloud_images.yaml

자세한 내용은 8.1절. “OVS TC-flower 하드웨어 오프로드의 역할 및 이미지 파일 생성”의 내용을 참조하십시오.

g. 환경에서 사용하는 Networking 서비스(neutron) 메커니즘 드라이버 및 라우터 체계용 환경 파일입니다.

- ML2/OVN
  - DCVR(Distributed virtual routing): neutron-ovn-dvr-ha.yaml
  - 중앙 집중식 가상 라우팅: neutron-ovn-ha.yaml
- ML2/OVS
  - DCVR(Distributed virtual routing): neutron-ovs-dvr.yaml
  - 중앙 집중식 가상 라우팅: neutron-ovs.yaml

h. 메커니즘 드라이버에 따라 SR-IOV의 환경 파일입니다.

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

참고

OVS-DPDK 환경도 있고 동일한 노드에서 OVS-DPDK 및 SR-IOV 인스턴스를 찾으려면 배포 스크립트에 다음 환경 파일을 포함합니다.

- ML2/OVN

  neutron-ovn-dpdk.yaml

- ML2/OVS

  neutron-ovs-dpdk.yaml

i. 다음과 같은 구성이 포함된 하나 이상의 사용자 지정 환경 파일입니다.

- ComputeOvsHwOffload 노드의 PCI 패스스루 장치입니다.
- ComputeOvsHwOffload 노드의 역할별 매개변수
- OVS 하드웨어 오프로드 환경의 기본 구성 값을 덮어씁니다.

  예: ovshwol-overrides.yaml

  자세한 내용은 다음을 참조하십시오.

- 8.2절. “OVS TC-flower 하드웨어 오프로드를 위한 PCI 패스스루 장치 구성”.
- 8.3절. “OVS TC-flower 하드웨어 오프로드에 대한 역할별 매개변수 및 구성 덮어쓰기 추가”.

예제

샘플 **openstack overcloud deploy** 명령에서 발췌한 내용은 **DVR**을 사용하는 **SR-IOV, ML2/OVN** 환경에 대한 명령 입력 순서를 올바르게 정렬하는 방법을 보여줍니다.

```
$ 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. **openstack overcloud deploy** 명령을 실행합니다.

오버클라우드 생성이 완료되면 **RHOSP director**에서 오버클라우드 액세스에 도움이 되는 세부 정보를 제공합니다.

검증

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 배포 검증 단계를 수행합니다.

다음 단계

1. **NIC**의 **e-switch** 모드가 **switchdev** 로 설정되어 있는지 확인합니다.

switchdev 모드는 **VF**에 매핑되는 **NIC**에 대표 포트를 설정합니다.

중요

OpenFlow 흐름을 하드웨어로 오프로드하려면 연결 추적(**conntrack**) 모듈에 대해 **switchdev** 포트에서 보안 그룹 및 포트 보안을 활성화해야 합니다.

a. 다음 명령을 실행하여 **NIC**를 확인합니다.

예제

이 예에서는 **NIC pci/0000:03:00.0** 을 쿼리합니다.

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

샘플 출력

출력은 다음과 유사합니다.

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

b. **NIC**를 **switchdev** 모드로 설정하려면 다음 명령을 실행합니다.

예제

이 예에서 **NIC pci/0000:03:00.0** 의 **e-switch** 모드는 **switchdev** 로 설정됩니다.

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

2. **switchdev-enabled NIC**에서 포트를 할당하려면 다음을 수행하십시오.

a. **admin** 역할을 사용하여 **RHOSP** 사용자로 로그인하고, **binding-profile** 값이 있는 **neutron** 포트를 만들고, 포트 보안을 비활성화합니다.

중요

OpenFlow 흐름을 하드웨어로 오프로드하려면 연결 추적(**conntrack**) 모듈에 대해 **switchdev** 포트에서 보안 그룹 및 포트 보안을 활성화해야 합니다.

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

b.

인스턴스를 생성할 때 이 포트 정보를 전달합니다.

대표 포트를 인스턴스 VF 인터페이스와 연결하고 일회성 OVS 데이터 경로 처리를 위해 대표 포트를 OVS 브리지 br-int 에 연결합니다. VF 포트는 물리적 "패치 패널" 프런트 엔드의 소프트웨어 버전과 같이 작동합니다.

새 인스턴스 생성에 대한 자세한 내용은 8.8절. “SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 인스턴스 생성” 을 참조하십시오.

3.

인터페이스 및 대표자에 다음 구성을 적용하여 TC Flower가 포트 수준에서 흐름 프로그래밍을 푸시하는지 확인합니다.

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

4.

성능 향상을 위해 각 네트워크 인터페이스의 채널 수를 조정합니다.

채널에는 인터럽트 요청(IRQ) 및 IRQ를 트리거하는 대기열 세트가 포함됩니다. mlx5_core 드라이버를 dev 모드로 설정하면 mlx5_core 드라이버가 기본적으로 하나의 결합된 채널로 설정되어 최적의 성능을 제공하지 않을 수 있습니다.

물리적 기능(PF) 대표자에서 다음 명령을 입력하여 호스트에 사용 가능한 CPU 수를 조정합니다.

예제

이 예에서 다중 목적 채널의 수는 네트워크 인터페이스 eno 3 s0f0:에서 3으로 설정됩니다.

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

추가 리소스

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 생성
- *명령줄 인터페이스 참조에* overcloud deploy

- 8.8절. “SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 인스턴스 생성”
- ethtool의 도움말 페이지
- devlink의 도움말 페이지
- *인스턴스 생성 을 위해 컴퓨팅 서비스 구성의 컴퓨팅 노드에 CPU 고정*구성

8.7. SR-IOV 또는 OVS TC-FLOWER 하드웨어 오프로드 환경에서 호스트 집계 생성

RHOSP(Red Hat OpenStack Platform) SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 더 나은 성능을 얻으려면 CPU 고정 및 대규모 페이지가 있는 게스트를 배포합니다. 집계 메타데이터와 플레이버 메타데이터를 일치시켜 호스트 하위 집합에서 고성능 인스턴스를 예약할 수 있습니다.

사전 요구 사항

- SR-IOV 또는 OVS 하드웨어 오프로드 환경에 구성된 RHOSP 오버클라우드입니다.
- RHOSP 오버클라우드는 AggregateInstanceExtraSpecsFilter 용으로 구성해야 합니다.

  자세한 내용은 8.2절. “OVS TC-flower 하드웨어 오프로드를 위한 PCI 패스스루 장치 구성”의 내용을 참조하십시오.

절차

1. 집계 그룹을 생성하고 관련 호스트를 추가합니다.

   정의된 플레이버 메타데이터와 일치하는 메타데이터(예: sriov=true)를 정의합니다.

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

2. 플레이버를 만듭니다.

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

3. 추가 플레이버 속성을 설정합니다.

   정의된 메타데이터 **sriov=true** 는 **SR-IOV** 집계의 정의된 메타데이터와 일치합니다.

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

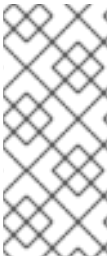
추가 리소스

- *명령줄 인터페이스 참조에* 집계
- *명령줄 인터페이스 참조의* 플레이버

8.8. SR-IOV 또는 OVS TC-FLOWER 하드웨어 오프로드 환경에서 인스턴스 생성

여러 명령을 사용하여 **RHOSP(Red Hat OpenStack Platform) SR-IOV** 또는 **OVS TC-flower** 하드웨어 오프로드 환경에서 인스턴스를 생성합니다.

호스트 집계를 사용하여 고성능 컴퓨팅 호스트를 분리합니다. 자세한 내용은 **8.7절. "SR-IOV 또는 OVS TC-flower 하드웨어 오프로드 환경에서 호스트 집계 생성"**의 내용을 참조하십시오.



참고

고정된 **CPU** 인스턴스는 고정되지 않은 인스턴스와 동일한 컴퓨팅 노드에 있을 수 있습니다. 자세한 내용은 *인스턴스 생성 가이드를 위한 컴퓨팅 서비스 구성 가이드의 컴퓨팅 노드에 CPU 고정 구성*을 참조하십시오.

사전 요구 사항

- **SR-IOV** 또는 **OVS** 하드웨어 오프로드 환경에 구성된 **RHOSP** 오버클라우드입니다.
- **OVS** 하드웨어 오프로드 환경의 경우 인스턴스를 생성할 수 있도록 **RHOSP** 관리자의 **VF**(가상 기능) 포트 또는 물리적 기능(**PF**) 포트가 있어야 합니다.

OVS 하드웨어 오프로드에는 **VF** 또는 **PF**를 생성하기 위한 바인딩 프로필이 필요합니다. **admin** 역할이 있는 **RHOSP** 사용자만 바인딩 프로필을 사용할 수 있습니다.

절차

1. 플레이버를 만듭니다.

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

작은 정보

플레이버에 **spec hw:pci_numa_affinity_policy** 를 추가하여 **PCI** 패스스루 장치 및 **SR-IOV** 인터페이스에 대한 **NUMA** 선호도 정책을 지정할 수 있습니다. 자세한 내용은 *인스턴스 생성을 위해 Compute 서비스 구성의* 플레이버 메타데이터 를 참조하십시오.

2. 네트워크 및 서브넷을 생성합니다.

```
$ 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. **admin** 역할이 있는 **RHOSP** 사용자가 아닌 경우 **RHOSP** 관리자가 인스턴스를 생성하는 데 필요한 **VF** 또는 **PF**를 제공할 수 있습니다. **5**단계로 이동합니다.

4. **admin** 역할이 있는 **RHOSP** 사용자인 경우 **VF** 또는 **PF** 포트를 생성할 수 있습니다.

- **VF** 포트:

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

- 단일 인스턴스 전용 **PF** 포트:

이 PF 포트는 Networking 서비스(neutron) 포트이지만 Networking 서비스에 의해 제어되지 않으며, 인스턴스에 전달되는 PCI 장치이므로 네트워크 어댑터로 표시되지 않습니다.

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

5. 인스턴스를 만듭니다.

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

추가 리소스

- *명령줄 인터페이스 참조*에서 플레이버 생성
- *명령줄 인터페이스 참조*에서 네트워크 생성
- *명령줄 인터페이스 참조*에서 **subnet create**
- *명령줄 인터페이스 참조*에서 **port create**
- *명령줄 인터페이스 참조*에서 **server create**

8.9. OVS TC-FLOWER 하드웨어 오프로드 문제 해결

OVS TC-flower 하드웨어 오프로드를 사용하는 RHOSP(Red Hat OpenStack Platform) 환경의 문제를 해결하는 경우 네트워크 및 인터페이스에 대한 사전 요구 사항 및 구성을 검토하십시오.

사전 요구 사항

- Linux Kernel 4.13 이상
- OVS 2.8 이상

- RHOSP 12 이상
- iproute 4.12 또는 최신 버전
- Mellanox NIC 펌웨어(예: FW ConnectX-5 16.21.0338 이상)

지원되는 사전 요구 사항에 대한 자세한 내용은 Red Hat Knowledgebase 솔루션 Network Adapter Fast Datapath Feature Support Matrix 를 참조하십시오.

네트워크 구성

HW 오프로드 배포에서 요구 사항에 따라 네트워크 구성에 대해 다음 시나리오 중 하나를 선택할 수 있습니다.

- 본딩에 연결된 동일한 인터페이스 세트를 사용하거나 유형마다 다른 NIC 세트를 사용하여 VXLAN 및 VLAN의 기본 게스트 VM을 사용할 수 있습니다.
- Linux 본딩을 사용하여 Mellanox NIC의 두 포트를 결합할 수 있습니다.
- Mellanox Linux 본딩의 VLAN 인터페이스에서 테넌트 VXLAN 네트워크를 호스팅할 수 있습니다.

개별 NIC 및 본딩이 ovs-bridge의 멤버인지 확인합니다.

다음 네트워크 구성 예를 참조하십시오.

```
...
- 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
...
```

지원되는 본딩 구성은 다음과 같습니다.

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

다음 본딩 구성은 지원되지 않습니다.

- **xmit_hash_policy=layer3+4**

인터페이스 구성

다음 절차에 따라 인터페이스 구성을 확인합니다.

절차

1. 배포하는 동안 호스트 네트워크 구성 툴 **os-net-config** 를 사용하여 **hw-tc-offload** 를 활성화합니다.

2. 컴퓨팅 노드를 재부팅할 때마다 **sriov_config** 서비스에서 **hw-tc-offload** 를 활성화합니다.

3. 본딩에 연결된 **NIC**의 **hw-tc-offload** 매개변수를 **on** 으로 설정합니다.

예제

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

hw-tc-offload: on
```

인터페이스 모드

다음 절차를 사용하여 인터페이스 모드를 확인합니다.

절차

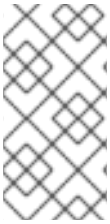
1. **HW** 오프로드에 사용하는 인터페이스에 대해 **eswitch** 모드를 **switchdev** 로 설정합니다.

2. 호스트 네트워크 구성 툴 **os-net-config** 를 사용하여 배포 중에 **eswitch** 를 활성화합니다.

3. 컴퓨팅 노드를 재부팅할 때마다 **sriov_config** 서비스에서 **eswitch** 를 활성화합니다.

예제

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



참고

PF 인터페이스의 드라이버는 **"mlx5e_rep"** 로 설정되어 있으며, 이는 **e-switch uplink** 포트의 대표자임을 나타냅니다. 이는 기능에 영향을 미치지 않습니다.

OVS 오프로드 상태

다음 절차를 사용하여 **OVS** 오프로드 상태를 확인합니다.

- 컴퓨팅 노드에서 **OVS**에서 하드웨어 오프로드를 활성화합니다.

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

"true"
```

VF 대표자 포트 이름

VF 대표 포트의 일관된 이름을 유지하기 위해 **os-net-config** 는 **udev** 규칙을 사용하여 **<PF-name>_<VF_id>** 형식의 포트 이름을 변경합니다.

절차

- 배포 후 **VF** 대표 포트의 이름이 올바르게 지정되었는지 확인합니다.

예제

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

샘플 출력

```
# 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}"
```

네트워크 트래픽 흐름

HW 오프로드 네트워크 흐름은 애플리케이션별 ASIC(Integrated circuit) 칩을 사용하는 물리적 스위치 또는 라우터와 유사한 방식으로 작동합니다.

스위치 또는 라우터의 Cryostat 쉘에 액세스하여 라우팅 테이블 및 기타 디버깅을 검사할 수 있습니다. 다음 절차에서는 Cumulus Linux 스위치의 Broadcom 칩셋을 예로 사용합니다. 환경에 적합한 값을 바꿉니다.

절차

1. **Broadcom 칩 테이블 콘텐츠를 가져오려면 bcmcmd 명령을 사용합니다.**

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

샘플 출력

```
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. **TC(트래픽 제어) 계층을 검사합니다.**

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

-

샘플 출력

```
…
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. **이 출력에서 in_hw 플래그 및 통계를 검사합니다. 하드웨어라는 용어는 하드웨어 가 네트워크 트래픽을 처리함을 나타냅니다. tc-policy=none 을 사용하는 경우 이 출력 또는 tcpdump를 확인하여 하드웨어 또는 소프트웨어가 패킷을 처리할 때 조사할 수 있습니다. 드라이버가 패킷을 오프로드할 수 없는 경우 dmesg 또는 ovs-vswitch.log 에서 해당 로그 메시지를 볼 수 있습니다.**

4. **예: Mellanox의 경우 로그 항목은 dmesg 의 syndrome 메시지와 유사합니다.**

샘플 출력

```
[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)
```

이 예에서 오류 코드 (0x6b1266)는 다음 동작을 나타냅니다.

샘플 출력

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

시스템

다음 절차에 따라 시스템을 검증합니다.

절차

1. 시스템에서 **SR-IOV** 및 **VT-d**가 활성화되어 있는지 확인합니다.

2. **GRUB**을 사용하여 커널 매개 변수에 **intel_iommu=on** 을 추가하여 **Linux**에서 **IOMMU**를 활성화합니다.

8.10. TC-FLOWER 하드웨어 오프로드 흐름 디버깅

ovs-vswitch.log 파일에 다음 메시지가 표시되는 경우 다음 절차를 사용할 수 있습니다.

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

절차

1. 오프로드 모듈에서 로깅하고 이 오류에 대한 추가 로그 정보를 얻으려면 컴퓨팅 노드에서 다음 명령을 사용하십시오.

```
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. **ovs-vswitchd** 로그를 다시 검사하여 문제에 대한 추가 세부 정보를 확인합니다.

다음 예제 로그에서는 지원되지 않는 속성 표시로 인해 오프로드가 실패했습니다.

```
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
```

Mellanox NIC 디버깅

Mellanox는 Red Hat SOS 보고서와 유사한 시스템 정보 스크립트를 제공했습니다.

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

이 명령을 실행하면 지원 사례에 유용한 관련 로그 정보의 zip 파일을 생성합니다.

절차

- **다음 명령을 사용하여 이 시스템 정보 스크립트를 실행할 수 있습니다.**

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

Mellanox 펌웨어 툴(MFT), mlxconfig, mlxlink 및 OpenFabrics Enterprise Distribution(OFED) 드라이버도 설치할 수 있습니다.

유용한 CLI 명령

다음 옵션과 함께 ethtool 유틸리티를 사용하여 진단 정보를 수집합니다.

- **ethtool -l <uplink 대표> : 채널 수 보기**
- **ethtool -l <uplink/VFs> : 통계 확인**

- **ethtool -i <uplink rep> :** 드라이버 정보 보기
- **ethtool -g <uplink rep> :** 링 크기 확인
- **ethtool -k <uplink/VFs> :** 활성화된 기능 보기

대표자 및 **PF** 포트에서 **tcpdump** 유틸리티를 사용하여 트래픽 흐름을 유사하게 확인합니다.

- 대표 포트의 링크 상태에 대한 변경 사항도 **VF** 링크 상태에 영향을 미칩니다.
- **VF** 통계가 있는 대표 포트 통계도 있습니다.

다음 명령을 사용하여 유용한 진단 정보를 가져옵니다.

```
$ 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
```

9장. OVS-DPDK 배포 계획

NFV용 OVS-DPDK(Data Plane Development Kit) 배포를 사용하여 Open vSwitch를 최적화하려면 OVS-DPDK가 컴퓨팅 노드 하드웨어(CPU, NUMA 노드, 메모리, NIC)를 사용하는 방법과 컴퓨팅 노드를 기반으로 개별 OVS-DPDK 매개변수를 결정하는 고려 사항을 이해해야 합니다.

중요

OVS-DPDK 및 OVS 기본 방화벽(conntrack을 기반으로 하는 상태 저장 방화벽)을 사용하는 경우 ICMPv4, ICMPv6, TCP 및 UDP 프로토콜을 사용하는 패킷만 추적할 수 있습니다. OVS는 다른 모든 유형의 네트워크 트래픽이 유효하지 않음으로 표시됩니다.

중요

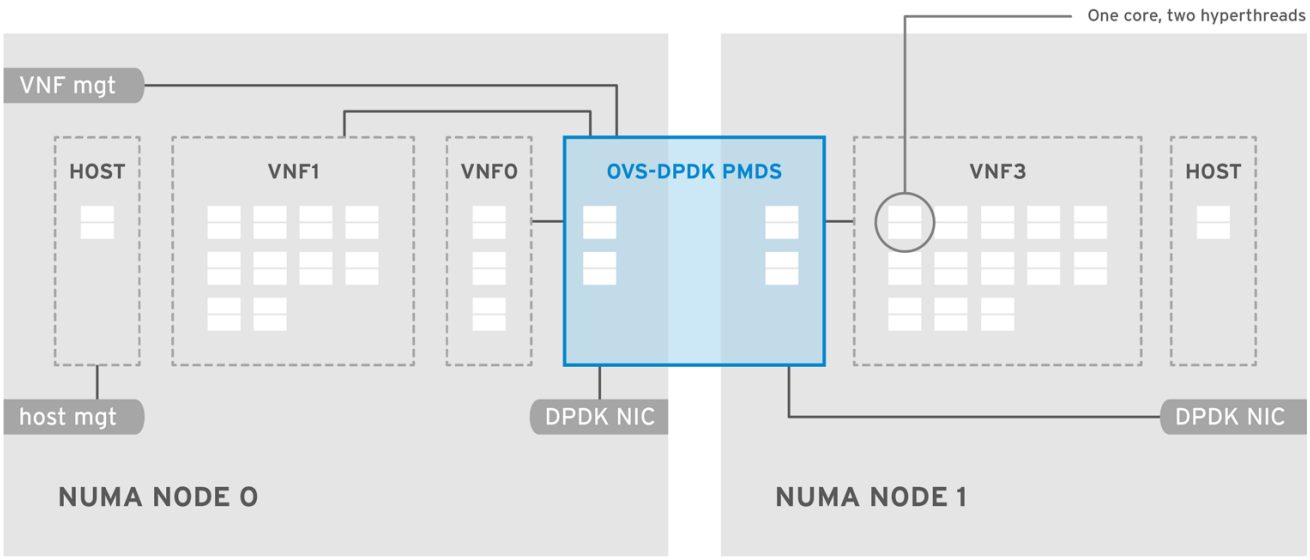
Red Hat은 비NFV 워크로드에 OVS-DPDK 사용을 지원하지 않습니다. 비NFV 워크로드에 OVS-DPDK 기능이 필요한 경우 TAM(Technical Account Manager)에 문의하거나 고객 서비스 요청 케이스를 열어 지원 예외 및 기타 옵션에 대해 논의합니다. 고객 서비스 요청 케이스를 열려면 케이스 생성으로 이동하여 계정 > 고객 서비스 요청을 선택합니다.

9.1. CPU 파티셔닝 및 NUMA 토폴로지가 포함된 OVS-DPDK

OVS-DPDK는 호스트, 게스트 및 자체에 대한 하드웨어 리소스를 분할합니다. OVS-DPDK Poll Mode Drivers(PMD)는 전용 CPU 코어가 필요한 DPDK 활성 루프를 실행합니다. 따라서 OVS-DPDK에 일부 CPU 및 대규모 페이지를 할당해야 합니다.

샘플 파티션에는 듀얼 소켓 컴퓨팅 노드의 NUMA 노드당 16개의 코어가 포함됩니다. 호스트와 OVS-DPDK 간에 NIC를 공유할 수 없기 때문에 트래픽에는 추가 NIC가 필요합니다.

그림 9.1. NUMA 토폴로지: CPU 파티션이 있는 OVS-DPDK



참고

NUMA 노드에 연결된 DPDK NIC가 없는 경우에도 두 NUMA 노드에 DPDK PMD 스레드를 예약해야 합니다.

정상 OVS-DPDK 성능의 경우 NUMA 노드에 로컬 메모리 블록을 예약합니다. 메모리 및 CPU 고정에 사용하는 동일한 NUMA 노드와 연결된 NIC를 선택합니다. 결합된 두 인터페이스가 동일한 NUMA 노드의 NIC에 있는지 확인합니다.

9.2. OVS-DPDK 매개변수

이 섹션에서는 OVS-DPDK가 director network_environment.yaml heat 템플릿 내에서 매개변수를 사용하여 최대 성능을 위해 CPU 및 메모리를 구성하는 방법을 설명합니다. 이 정보를 사용하여 컴퓨팅 노드의 하드웨어 지원과 OVS-DPDK 배포를 최적화하기 위해 하드웨어를 분할하는 방법을 평가합니다.

참고

CPU 코어를 할당할 때 항상 CPU 형제 스레드 또는 논리 CPU를 물리적 코어에 결합합니다.

컴퓨팅 노드에서 CPU 및 NUMA 노드를 결정하는 방법에 대한 자세한 내용은 NUMA 노드 토폴로지 검색을 참조하십시오. 이 정보를 사용하여 CPU 및 기타 매개변수를 매핑하여 호스트, 게스트 인스턴스 및 OVS-DPDK 프로세스 요구 사항을 지원합니다.

9.2.1. CPU 매개변수

OVS-DPDK는 CPU 파티셔닝에 다음 매개변수를 사용합니다.

OvsPmdCoreList

DPDK 폴링 모드 드라이버(PMD)에 사용되는 CPU 코어를 제공합니다. DPDK 인터페이스의 로컬 NUMA 노드와 연결된 CPU 코어를 선택합니다. OVS에서 pmd-cpu-mask 값으로 OvsPmdCoreList 를 사용합니다. OvsPmdCoreList 에 대해 다음 권장 사항을 사용하십시오.

- 형제 스레드를 페어링합니다.
- 성능은 PMD Core 목록에 할당된 물리적 코어 수에 따라 다릅니다. DPDK NIC와 연결된 NUMA 노드에서 필요한 코어를 할당합니다.
- DPDK NIC가 있는 NUMA 노드의 경우 성능 요구 사항에 따라 필요한 물리적 코어 수를 확인하고 각 물리 코어의 모든 형제 스레드 또는 논리 CPU를 포함합니다.
- DPDK NIC가 없는 NUMA 노드의 경우 NUMA 노드의 첫 번째 물리적 코어를 제외하고 모든 물리적 코어의 형제 스레드 또는 논리 CPU를 할당합니다.

참고

NUMA 노드에 연결된 DPDK NIC가 없는 경우에도 두 NUMA 노드에 DPDK PMD 스레드를 예약해야 합니다.

NovaComputeCpuDedicatedSet

고정 인스턴스 CPU의 프로세스를 예약할 수 있는 쉼표로 구분된 물리적 호스트 CPU 수 목록 또는 범위입니다. 예를 들어 NovaComputeCpuDedicatedSet: [4-12,^8,15] 는 8을 제외하고 4-12 및 15 의 코어를 예약합니다.

- OvsPmdCoreList 에서 모든 코어를 제외합니다.
- 나머지 코어를 모두 포함합니다.

- 형제 스레드를 페어링합니다.

NovaComputeCpuSharedSet

인스턴스 에뮬레이터 스레드의 호스트 **CPU**를 결정하는 데 사용되는 쉼표로 구분된 물리적 호스트 **CPU** 수 목록 또는 범위입니다.

IsolCpusList

호스트 프로세스와 분리된 **CPU** 코어 세트. **IsolCpusList** 는 **tuned-profiles-cpu-partitioning** 구성 요소의 **cpu-partitioning-variable.conf** 파일의 **isolated_cores** 값입니다. **IsolCpusList** 에 대해 다음 권장 사항을 사용하십시오.

- **OvsPmdCoreList** 및 **NovaComputeCpuDedicatedSet** 의 코어 목록을 일치시킵니다.
- 형제 스레드를 페어링합니다.

DerivePciWhitelistEnabled

VM의 **VF**(가상 기능)를 예약하려면 **NovaPCIPassthrough** 매개변수를 사용하여 **Nova**에 전달되는 **VF** 목록을 생성합니다. 목록에서 제외된 **VFS**는 호스트에서 계속 사용할 수 있습니다.

목록의 각 **VF**에 대해 **address** 매개변수를 **address** 값으로 확인하는 정규식으로 채웁니다.

다음은 수동 목록 생성 프로세스의 예입니다. **eno2** 라는 장치에서 **NIC** 파티셔닝이 활성화된 경우 다음 명령을 사용하여 **VF**의 **PCI** 주소를 나열합니다.

```
[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
```

이 경우 **VF 0, 4, 6**은 **NIC** 파티셔닝에 **eno2** 에서 사용됩니다. 다음 예와 같이 **VFs 1-3, 5** 및 **7**을 포함하도록 **NovaPCIPassthrough** 를 수동으로 구성하고 결과적으로 **VF 0,4** 및 **6**을 제외합니다.

```
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. 메모리 매개변수

OVS-DPDK는 다음 메모리 매개변수를 사용합니다.

OvsDpdkMemoryChannels

NUMA 노드당 **CPU**에 메모리 채널을 매핑합니다. **OVS**의 **OvsDpdkMemoryChannels** 는 **OVS** 의 **other_config:dpdk-extra="-n <value>"** 값입니다. **OvsDpdkMemoryChannels** 에 대한 다음 권장 사항을 확인하십시오.

- **dmidecode -t** 메모리 또는 하드웨어 설명서를 사용하여 사용 가능한 메모리 채널 수를 확인합니다.
- **ls /sys/devices/system/node/node* -d** 를 사용하여 **NUMA** 노드 수를 확인합니다.
- 사용 가능한 메모리 채널 수를 **NUMA** 노드 수로 나눕니다.

NovaReservedHostMemory

호스트의 작업에 대해 메모리(**MB**)를 예약합니다. **NovaReservedHostMemory** 는 **nova.conf** 의 컴퓨팅 노드의 **reserved_host_memory_mb** 값입니다. **NovaReservedHostMemory** 에 대한 다음 권장 사항을 확인하십시오.

- 정적 권장 값 **4096MB**를 사용합니다.

OvsDpdkSocketMemory

NUMA 노드별로 **hugepage** 풀에서 사전 할당하기 위한 메모리 양(**MB**)을 지정합니다. **OvsDpdkSocketMemory** 는 **OVS**의 **other_config:dpdk-socket-mem** 값입니다. **OvsDpdkSocketMemory** 에 대한 다음 권장 사항을 확인하십시오.

- 쉼표로 구분된 목록으로 제공합니다.

- DPDK NIC가 없는 NUMA 노드의 경우 1024MB(1GB)의 정적 권장 사항을 사용합니다.
- NUMA 노드의 각 NIC의 MTU 값에서 OvsDpdkSocketMemory 값을 계산합니다.
- 다음 공식은 OvsDpdkSocketMemory 의 값과 유사합니다.
  - MEMORY_REQD_PER_MTU = (ROUNDUP_PER_MTU + 800) * (4096 * 64) Bytes
    - 800은 오버헤드 값입니다.
    - 4096 * 64는 mempool의 패킷 수입니다.
- NUMA 노드에 설정된 각 MTU 값에 대해 MEMORY_REQD_PER_MTU를 추가하고 버퍼로 512MB를 추가합니다. 값을 1024의 배수로 반올림합니다.

샘플 계산 - MTU 2000 및 MTU 9000

DPDK NIC dpdk0 및 dpdk1은 각각 동일한 NUMA 노드 0에 있으며 MTU 9000 및 2000으로 구성됩니다. 필요한 메모리를 파생하기 위한 샘플 계산은 다음과 같습니다.

1. MTU 값을 1024바이트의 가장 가까운 배수로 반올림합니다.

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

2. 이러한 반올림된 바이트 값을 기반으로 각 MTU 값에 필요한 메모리를 계산합니다.

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

3. 필요한 총 메모리(바이트)를 계산합니다.

   2625634304 + 746586112 + 536870912 = 3909091328 bytes.

   이 계산은 **(9000**의 **MTU**에 필요한 메모리**) + (2000**의 **MTU**에 메모리 필요**) + (512MB** 버퍼**)**를 나타냅니다.

4. 필요한 총 메모리를 **MB**로 변환합니다.

   3909091328 / (1024*1024) = 3728 MB.

5. 이 값을 가장 가까운 **1024**까지 반올림합니다.

   3724 MB rounds up to 4096 MB.

6. 이 값을 사용하여 **OvsDpdkSocketMemory** 를 설정합니다.

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

샘플 계산 - MTU 2000

DPDK NIC dpdk0 및 **dpdk1**은 동일한 **NUMA** 노드 **0**에 있으며 각각 **2000**의 **MTU**로 구성됩니다. 필요한 메모리를 파생하기 위한 샘플 계산은 다음과 같습니다.

1. **MTU** 값을 **1024**바이트의 가장 가까운 배수로 반올림합니다.

   The MTU value of 2000 becomes 2048 bytes.

2. 이러한 반올림된 바이트 값을 기반으로 각 **MTU** 값에 필요한 메모리를 계산합니다.

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

3. 필요한 총 메모리(바이트)를 계산합니다.

   746586112 + 536870912 = 1283457024 bytes.

이 계산은 (2000의 MTU에 필요한 메모리) + (512MB 버퍼)를 나타냅니다.

4. 필요한 총 메모리를 MB로 변환합니다.

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

5. 이 값을 1024의 가장 가까운 배수로 반올림합니다.

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

6. 이 값을 사용하여 OvsDpdkSocketMemory 를 설정합니다.

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

9.2.3. 네트워킹 매개변수

OvsDpdkDriverType

DPDK에서 사용하는 드라이버 유형을 설정합니다. vfio-pci 의 기본값을 사용합니다.

NeutronDatapathType

OVS 브리지의 datapath 유형입니다. DPDK는 netdev 의 기본값을 사용합니다.

NeutronVhostuserSocketDir

OVS의 vhost-user 소켓 디렉터리를 설정합니다. vhost 클라이언트 모드에 /var/lib/vhost_sockets 를 사용합니다.

9.2.4. 기타 매개변수

NovaSchedulerEnabledFilters

컴퓨팅 노드에서 요청된 게스트 인스턴스에 대해 일치하는 컴퓨팅 노드를 찾는 데 사용하는 정렬된 필터 목록을 제공합니다.

VhostuserSocketGroup

vhost-user 소켓 디렉터리 그룹을 설정합니다. 기본값은 qemu 입니다. ovs-vswitchd 및 qemu 프로세스가 virtio-net 장치를 구성하는 공유 대규모 페이지 및 unix 소켓에 액세스할 수 있도록

VhostuserSocketGroup 을 **hugetlbfs** 로 설정합니다. 이 값은 역할별로 고유하며 **OVS-DPDK**를 활용하는 모든 역할에 적용해야 합니다.

중요

VhostuserSocketGroup 매개변수를 사용하려면 **NeutronVhostuserSocketDir** 도 설정해야 합니다. 자세한 내용은 9.2.3절. "네트워킹 매개변수"의 내용을 참조하십시오.

KernelArgs

부팅 시 컴퓨팅 노드의 **/etc/default/grub** 에 여러 커널 인수를 제공합니다. 구성에 따라 다음 값을 추가합니다.

- **hugepagesz: CPU**에 대규모 페이지 크기를 설정합니다. 이 값은 **CPU** 하드웨어에 따라 다를 수 있습니다. **OVS-DPDK** 배포용 **1G(default_hugepagesz=1GB hugepagesz=1G)**로 설정합니다. 이 명령을 사용하여 **CPU**가 **1G**를 지원하는지 확인하는 **pdpe1gb CPU** 플래그를 확인합니다.

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

- **hugepages count:** 사용 가능한 호스트 메모리를 기반으로 사용 가능한 대규모 페이지 수를 설정합니다. **NovaReservedHostMemory** 를 제외하고 사용 가능한 대부분의 메모리를 사용합니다. 컴퓨팅 노드의 플레이버 내에서 대규모 페이지 수 값을 구성해야 합니다.

- **iommu: For Intel CPUs, add "intel_iommu=on iommu=pt"**

- **isolcpus:** 튜닝을 위해 **CPU** 코어를 설정합니다. 이 값은 **IsolCpusList** 와 일치합니다.

CPU 분리에 대한 자세한 내용은 **RHEL 8** 및 **RHEL 9**의 **Red Hat Knowledgebase** 솔루션 **OpenStack CPU** 격리 지침을 참조하십시오.

DdpPackage

배포의 장치에 프로필 패키지를 적용하여 장치의 패킷 처리 파이프라인을 변경하도록 **DDP(Dynamic Device Personalization)**를 구성합니다. **DDP** 패키지를 포함하도록 **network_environment.yaml** 템플릿에 다음 행을 추가합니다.

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

9.2.5. VM 인스턴스 플레이버 사양

NFV 환경에 VM 인스턴스를 배포하기 전에 CPU 고정, 대규모 페이지 및 에뮬레이터 스레드 고정을 사용하는 플레이버를 생성합니다.

hw:cpu_policy

이 매개변수가 **dedicated** 로 설정된 경우 게스트는 고정된 CPU를 사용합니다. 이 매개변수 세트를 사용하여 플레이버에서 생성된 인스턴스의 오버 커밋 비율은 1:1입니다. 기본값은 **shared** 입니다.

hw:mem_page_size

이 매개변수를 표준 접미사가 있는 특정 값의 유효한 문자열(예: 4KB,8MB 또는 1GB)으로 설정합니다. **hugepagesz** 부팅 매개변수와 일치하도록 1GB를 사용합니다. 부팅 매개변수에서 **OvsDpdkSocketMemory** 를 제거하여 가상 머신에 사용할 수 있는 대규모 페이지 수를 계산합니다. 다음 값도 유효합니다.

- small(기본값) - 가장 작은 페이지 크기가 사용됩니다.
- large - 대용량 페이지 크기만 사용합니다. (2MB 또는 1GB x86 아키텍처의 경우)
- 모든 - 컴퓨팅 드라이버는 대규모 페이지를 사용하려고 시도할 수 있지만 사용할 수 없는 경우 기본값은 small입니다.

hw:emulator_threads_policy

이 매개변수의 값을 공유하여 에뮬레이터 스레드가 heat 매개변수 **NovaComputeCpuSharedSet** 에서 식별한 CPU에 고정되도록 설정합니다. 에뮬레이터 스레드가 폴링 모드 드라이버(PMD) 또는 실시간 처리가 있는 vCPU에서 실행 중인 경우 패킷 손실과 같은 부정적인 영향을 미칠 수 있습니다.

9.3. OVS-DPDK 배포에 전원 저장

전원 저장 프로필 **cpu-partitioning-powersave** 이 RHEL 9 (Red Hat Enterprise Linux 9)에 도입되었으며 RHOSP(Red Hat OpenStack Platform) 17.1.3에서 사용할 수 있습니다. 이 TuneD 프로필은 RHOSP 17.1 NFV 환경에서 전원을 절약하기 위한 기본 구성 요소입니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.
- 더 높은 **C-state**를 허용하도록 전력 절감을 달성하려는 **CPU**가 활성화됩니다.

자세한 내용은 **tuned-profiles-cpu-partitioning(7)** 도움말 페이지의 **max_power_state** 옵션을 참조하십시오.

프로세스

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **Ansible** 플레이북 **YAML** 파일을 만듭니다(예: **/home/stack/cli-overcloud-tuned-maxpower-conf.yaml**).
4. **cli-overcloud-tuned-maxpower-conf.yaml** 파일에 다음 구성을 추가합니다.

```
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. 역할 데이터 파일에 전원 저장 프로필을 추가합니다.

   자세한 내용은 10.2에서 참조하십시오. 역할 및 이미지 파일 생성.

6. cli-overcloud-tuned-maxpower-conf.yaml 플레이북을 베어 메탈 노드 정의 파일에 추가합니다.

   자세한 내용은 10.5에서 참조하십시오. 베어 메탈 노드 정의 파일 생성.

7. NIC 구성 템플릿에 큐 크기가 설정되어 있는지 확인합니다.

   자세한 내용은 10.6에서 참조하십시오. NIC 구성 템플릿 생성.

추가 리소스

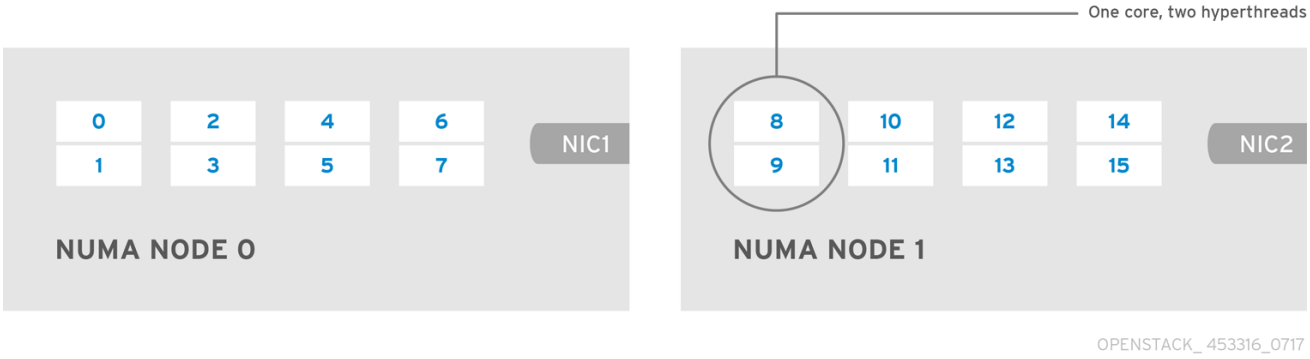
- force_latency

9.4. 두 개의 NUMA 노드 예 OVS-DPDK 배포

다음 예제의 컴퓨팅 노드에는 두 개의 NUMA 노드가 포함됩니다.

- NUMA 0에는 코어 0-7이 있습니다. 형제 스레드 쌍은 (0,1), (2,3), (4,5) 및 (6,7)입니다.
- NUMA 1에는 코어 8-15가 있습니다. 형제 스레드 쌍은 (8,9)(10,11), (12,13) 및 (14,15)입니다.
- 각 NUMA 노드는 물리적 NIC, 즉 NUMA 0의 NIC1, NUMA 1의 NIC2에 연결됩니다.

그림 9.2. ovs-DPDK: 두 개의 NUMA 노드 예



참고

데이터 경로 DPDK 프로세스가 아닌 경우 각 NUMA 노드 (0,1 및 8,9)에서 첫 번째 물리적 코어 또는 두 스레드 쌍을 예약합니다.

이 예제에서는 1500 MTU 구성도 가정하므로 OvsDpdkSocketMemory 는 모든 사용 사례에서 동일합니다.

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

DPDK용 NIC 1 - PMD에 하나의 물리적 코어

이 사용 사례에서는 PMD에 대해 NUMA 0에 하나의 물리적 코어를 할당합니다. 해당 NUMA 노드에 대한 NIC에서 DPDK가 활성화되지 않더라도 NUMA 1에서 하나의 물리적 코어도 할당해야 합니다. 나머지 코어는 게스트 인스턴스에 할당됩니다. 결과 매개변수 설정은 다음과 같습니다.

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

DPDK용 NIC 1 - PMD에 두 개의 물리적 코어

이 사용 사례에서는 **PMD**에 대해 **NUMA 0**에 두 개의 물리적 코어를 할당합니다. 해당 **NUMA** 노드에 대한 **NIC**에서 **DPDK**가 활성화되지 않더라도 **NUMA 1**에서 하나의 물리적 코어도 할당해야 합니다. 나머지 코어는 게스트 인스턴스에 할당됩니다. 결과 매개변수 설정은 다음과 같습니다.

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

DPDK용 NIC 2 (PMD 용 물리적 코어 1개)

이 사용 사례에서는 **PMD**에 대해 **NUMA 1**에 하나의 물리적 코어를 할당합니다. **DPDK**가 해당 **NUMA** 노드에 대해 **NIC**에서 활성화되지 않더라도 **NUMA 0**에서 하나의 물리적 코어도 할당해야 합니다. 나머지 코어는 게스트 인스턴스에 할당됩니다. 결과 매개변수 설정은 다음과 같습니다.

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

DPDK의 경우 NIC 2 (PMD에 두 개의 물리적 코어 포함)

이 사용 사례에서는 **PMD**에 대해 **NUMA 1**에 두 개의 물리적 코어를 할당합니다. **DPDK**가 해당 **NUMA** 노드에 대해 **NIC**에서 활성화되지 않더라도 **NUMA 0**에서 하나의 물리적 코어도 할당해야 합니다. 나머지 코어는 게스트 인스턴스에 할당됩니다. 결과 매개변수 설정은 다음과 같습니다.

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

PMD에 두 개의 물리적 코어가 있는 DPDK용 NIC 1 및 NIC2

이 사용 사례에서는 **PMD**에 대해 각 **NUMA** 노드에 두 개의 물리적 코어를 할당합니다. 나머지 코어는 게스트 인스턴스에 할당됩니다. 결과 매개변수 설정은 다음과 같습니다.

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

9.5. NFV OVS-DPDK 배포의 토폴로지

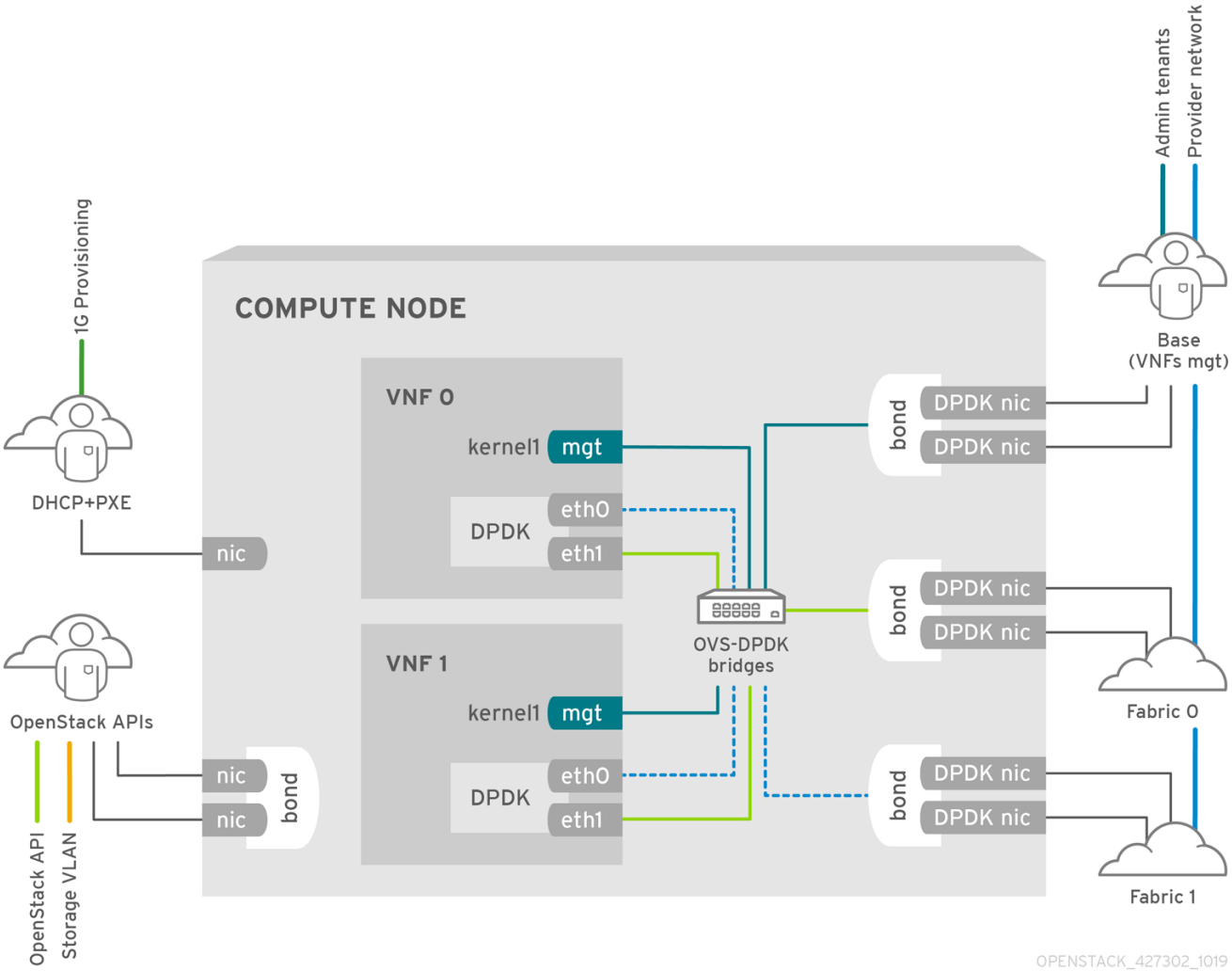
이 예제 배포는 **OVS-DPDK** 구성을 보여주고 각각 두 개의 인터페이스가 있는 두 개의 가상 네트워크 기능(**VNF**)으로 구성됩니다.

- **mgt** 로 표시되는 관리 인터페이스입니다.

- 데이터 플레인 인터페이스입니다.

OVS-DPDK 배포에서 VNF는 물리적 인터페이스를 지원하는 빌드된 DPDK와 함께 작동합니다. OVS-DPDK는 vSwitch 수준에서 본딩을 활성화합니다. OVS-DPDK 배포의 성능을 개선하려면 커널과 OVS-DPDK NIC를 분리하는 것이 좋습니다. 가상 머신의 기본 공급자 네트워크에 연결된 관리(mgt) 네트워크를 분리하려면 추가 NIC가 있는지 확인하십시오. 컴퓨팅 노드는 Ceph API에서 재사용할 수 있지만 OpenStack 프로젝트와 공유할 수 없는 Red Hat OpenStack Platform API 관리를 위한 두 개의 일반 NIC로 구성됩니다.

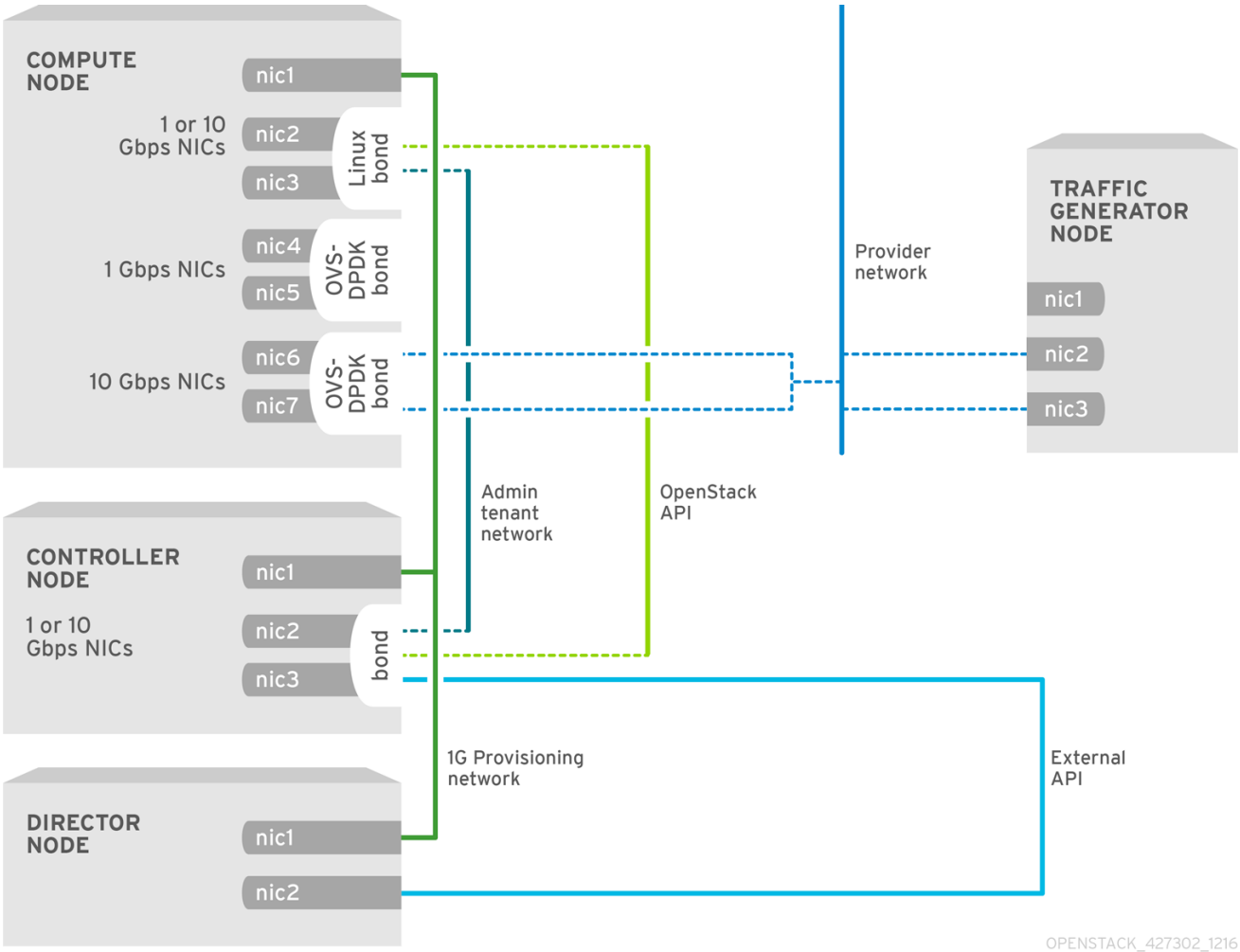
그림 9.3. Compute 노드: NFV OVS-DPDK



NFV OVS-DPDK 토폴로지

다음 이미지는 NFV용 OVS-DPDK의 토폴로지를 보여줍니다. 1개 또는 10Gbps NIC가 있는 컴퓨팅 및 컨트롤러 노드와 director 노드로 구성됩니다.

그림 9.4. NFV 토폴로지: OVS-DPDK

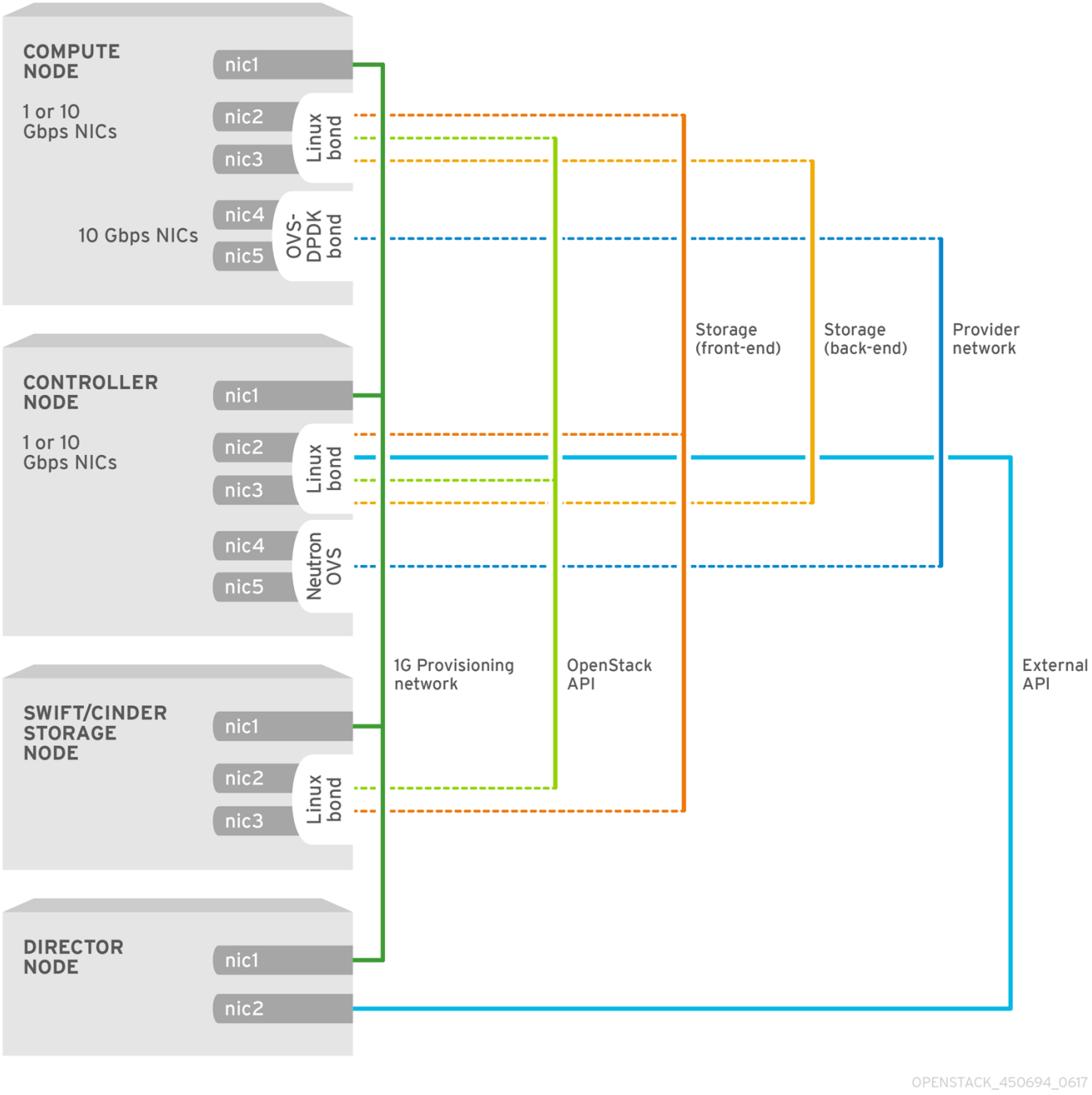


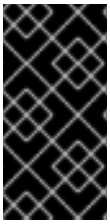
10장. OVS-DPDK 배포 구성

이 섹션에서는 RHOSP(Red Hat OpenStack Platform) 환경의 OVS-DPDK(Open vSwitch Data Plane Development Kit)를 배포, 사용 및 해결하는 방법을 설명합니다. RHOSP는 OVS-DPDK 배포를 위해 OVS 클라이언트 모드에서 작동합니다.

다음 그림은 컨트롤 플레인 및 데이터 플레인에 두 개의 결합된 포트가 있는 OVS-DPDK 토폴로지를 보여줍니다.

그림 10.1. OVS-DPDK 토폴로지 샘플





중요

이 섹션에는 토폴로지 및 사용 사례에 맞게 수정해야 하는 예제가 포함되어 있습니다. 자세한 내용은 NFV 하드웨어 요구 사항을 참조하십시오.

사전 요구 사항

- RHOSP 언더클라우드.

오버클라우드를 배포하려면 언더클라우드를 설치하고 구성해야 합니다. 자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리를* 참조하십시오.

참고

RHOSP director는 템플릿 및 사용자 지정 환경 파일에서 지정하는 키-값 쌍을 통해 OVS-DPDK 구성 파일을 수정합니다. OVS-DPDK 파일을 직접 수정하지 않아야 합니다.

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

RHOSP(Red Hat OpenStack Platform) director를 사용하여 RHOSP 환경에서 OVS-DPDK를 설치하고 구성합니다. 높은 수준의 단계는 다음과 같습니다.

1. OVS-DPDK의 알려진 제한 사항을 검토합니다.
2. 역할 및 이미지 파일을 생성합니다.
3. OVS-DPDK 사용자 지정을 위한 환경 파일을 만듭니다.
4. 보안 그룹에 대한 방화벽을 구성합니다.
5. 베어 메탈 노드 정의 파일을 생성합니다.

6. NIC 구성 템플릿을 생성합니다.

7. OVS-DPDK 인터페이스의 MTU 값을 설정합니다.

8. OVS-DPDK 인터페이스에 대한 멀티 큐를 설정합니다.

9. 노드 프로비저닝에 대한 DPDK 매개변수를 구성합니다.

10. 오버클라우드 네트워크 및 VIP를 프로비저닝합니다.

    자세한 내용은 다음을 참조하십시오.

    - *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리*에서 오버클라우드 네트워크 정의 구성 및 프로비저닝.
    - *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리*에서 오버클라우드용 네트워크 VIP 구성 및 프로비저닝.

11. 베어 메탈 노드를 프로비저닝합니다.

    *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리*에서 오버클라우드용 베어 메탈 노드 프로비저닝

12. OVS-DPDK 오버클라우드를 배포합니다.

추가 리소스

- 10.11절. “플레이버 생성 및 OVS-DPDK의 인스턴스 배포”
- 10.12절. “OVS-DPDK 구성 문제 해결”

10.1. OVS-DPDK에 대한 알려진 제한 사항

OVS-DPDK(Open vSwitch Data Plane Development Kit) 환경에서 Red Hat OpenStack Platform을 구성할 때 다음 제한 사항을 확인합니다.

- 비DPDK 트래픽에는 Linux 본딩과 내부, 관리, 스토리지, 스토리지 관리, 테넌트와 같은 컨트롤 플레인 네트워크를 사용합니다. 본딩에 사용되는 PCI 장치가 동일한 NUMA 노드에 있는지 확인합니다. Neutron Linux 브리지 구성은 Red Hat에서 지원되지 않습니다.
- OVS-DPDK가 있는 호스트에서 실행되는 모든 인스턴스에 대해 대규모 페이지가 필요합니다. 게스트에 대규모 페이지가 없으면 인터페이스가 표시되지만 작동하지 않습니다.
- OVS-DPDK에서는 DVR(Distributed Virtual Routing)과 같은 탭 장치를 사용하는 서비스의 성능 저하가 있습니다. 결과 성능은 프로덕션 환경에 적합하지 않습니다.
- OVS-DPDK를 사용하는 경우 동일한 컴퓨팅 노드의 모든 브릿지는 ovs_user_bridge 여야 합니다. director는 설정을 수락할 수 있지만 Red Hat OpenStack Platform은 동일한 노드에서 ovs_bridge 및 ovs_user_bridge 혼합을 지원하지 않습니다.

다음 단계

- 10.2절. “역할 및 이미지 파일 생성” 으로 이동합니다.

10.2. 역할 및 이미지 파일 생성

RHOSP(Red Hat OpenStack Platform) director는 역할을 사용하여 노드에 서비스를 할당합니다. OVS-DPDK 환경에 RHOSP를 배포할 때 ComputeOvsDpdk 는 기본 컴퓨팅 서비스 외에도 ComputeNeutronOvsDpdk 서비스를 포함하는 RHOSP 설치와 함께 제공되는 사용자 지정 역할입니다.

언더클라우드 설치에는 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 환경 파일이 필요합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **Controller** 및 **ComputeOvsDpdk** 역할을 포함하는 새 역할 데이터 파일(예: **roles_data_compute_ovsdpdk.yaml**)을 생성합니다.

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

참고

RHOSP 환경, **OVS-DPDK, SR-IOV** 및 **OVS** 하드웨어 오프로드에서 여러 기술을 사용하는 경우 모든 역할을 포함하도록 하나의 역할 데이터 파일만 생성합니다.

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

4. 선택 사항: **TuneD** 프로필인 **cpu-partitioning-powersave** 를 사용하여 패킷이 전달되지 않을 때 **sleep** 모드로 전환하도록 **OVS-DPDK**를 구성할 수 있습니다.

cpu-partitioning-powersave 를 구성하려면 기본 **TuneD** 프로필을 생성된 역할 데이터 파일에서 **power saving TuneD profile, cpu-partitioning-powersave** 로 바꿉니다.

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

예제

생성된 역할 데이터 파일 **/home/stack/templates/roles_data_compute_ovsdpdpdk.yaml** 에서는 **TunedProfileName** 의 기본값이 **cpu-partitioning-powersave:**로 교체됩니다.

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

5. 이미지 파일을 생성하려면 **openstack tripleo container image prepare** 명령을 실행합니다. 다음 입력이 필요합니다.

- 이전 단계에서 생성한 역할 데이터 파일(예: **roles_data_compute_ovsdpdk.yaml**)
- 네트워킹 서비스 메커니즘 드라이버에 적합한 **DPDK** 환경 파일입니다.
  - **ML2/OVN** 환경의 **neutron-ovn-dpdk.yaml** 파일
  - **ML2/OVS** 환경의 **neutron-ovs-dpdk.yaml** 파일

예제

이 예에서는 **ML2/OVN** 환경에 대해 **overcloud_images.yaml** 파일이 생성됩니다.

```
$ 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. 역할 데이터 파일의 경로 및 파일 이름과 사용자가 생성한 이미지 파일을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이러한 파일을 사용합니다.

다음 단계

- 10.3절. “OVS-DPDK 사용자 지정 환경 파일 생성” 으로 이동합니다.

추가 리소스

- OVS-DPDK 배포에 전원 저장
- *director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리의 구성 가능 서비스 및 사용자 지정 역할

- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 컨테이너 이미지 준비*

10.3. OVS-DPDK 사용자 지정 환경 파일 생성

사용자 지정 환경 YAML 파일에서 특정 Red Hat OpenStack Platform 구성 값을 사용하여 OVS-DPDK 배포를 구성할 수 있습니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. stack 사용자로 언더클라우드에 로그인합니다.

2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 사용자 지정 환경 YAML 파일(예: ovs-dpdk-overrides.yaml)을 만듭니다.

4. 사용자 지정 환경 파일에서 AggregateInstanceExtraSpecsFilter 가 Compute 서비스(nova)에서 노드를 필터링하는 데 사용하는 NovaSchedulerEnabledFilters 매개변수의 필터 목록에 있는지 확인합니다.

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

5. OVS-DPDK 컴퓨팅 노드의 역할별 매개 변수를 사용자 지정 환경 파일에 추가합니다.

예제

```
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"
```

6. 해당 파일의 구성 기본값을 재정의해야 하는 경우 3단계에서 생성한 사용자 지정 환경 파일에 재정의를 추가합니다.

   RHOSP director는 다음 파일을 사용하여 OVS-DPDK를 설정합니다.

   - ML2/OVN 배포

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

   - ML2/OVS 배포

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

7. 생성한 사용자 지정 환경 파일의 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 10.4절. “보안 그룹에 대한 방화벽 구성” 으로 이동합니다.

10.4. 보안 그룹에 대한 방화벽 구성

데이터 플레인 인터페이스에는 상태 저장 방화벽의 고성능이 필요합니다. 이러한 인터페이스를 보호하려면 RHOSP(Red Hat OpenStack Platform) OVS-DPDK 환경에서 telco-grade 방화벽을 VNF(가상 네트워크 기능)로 배포하는 것이 좋습니다.

ML2/OVS 배포에서 컨트롤 플레인 인터페이스를 구성하려면 parameter_defaults 아래의 사용자 지정 환경 파일에서 NeutronOVSFirewallDriver 매개변수를 openvswitch 로 설정합니다. OVN 배포에서 ACL(액세스 제어 목록)을 사용하여 보안 그룹을 구현할 수 있습니다.

흐름의 연결 추적 속성이 오프로드 경로에서 지원되지 않으므로 하드웨어 오프로드에는 OVS 방화벽 드라이버를 사용할 수 없습니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 10.3절. “OVS-DPDK 사용자 지정 환경 파일 생성” 에서 생성한 사용자 정의 환경 YAML 파일을 열거나 새 파일을 생성합니다.
4. parameter_defaults 에서 다음 키-값 쌍을 추가합니다.

```
parameter_defaults:
...

 NeutronOVSFirewallDriver: openvswitch
```

5. 새 사용자 지정 환경 파일을 생성한 경우 해당 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

6. 오버클라우드를 배포한 후 **openstack port set** 명령을 실행하여 데이터 플레인 인터페이스에 대한 **OVS** 방화벽 드라이버를 비활성화합니다.

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

다음 단계

- 10.5절. “베어 메탈 노드 정의 파일 생성” 으로 이동합니다.

추가 리소스

- *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 구성 가능 서비스 및 사용자 지정 역할

- NFV에 대해 테스트된 NIC

10.5. 베어 메탈 노드 정의 파일 생성

RHOSP(Red Hat OpenStack Platform) director를 사용하여 정의 파일을 사용하여 **OVS-DPDK** 환경의 베어 메탈 노드를 프로비저닝합니다. 베어 메탈 노드 정의 파일에서 배포하려는 베어 메탈 노드의 수량 및 속성을 정의하고 오버클라우드 역할을 이러한 노드에 할당합니다. 노드의 네트워크 레이아웃도 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

프로세스

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. ***director** 가이드를 사용하여 **Red Hat OpenStack Platform** 설치 및 관리에서 오버클라우드의 베어 메탈 노드 프로비저닝에 지시되는 대로 **overcloud -baremetal-deploy.yaml** 과 같은 베어 메탈 노드 정의 파일을 생성합니다.*

4. **overcloud-baremetal-deploy.yaml** 파일에서 **Ansible** 플레이북 **cli-overcloud-node-kernelargs.yaml** 에 선언을 추가합니다. 플레이북에는 베어 메탈 노드를 프로비저닝할 때 사용할 커널 인수가 포함되어 있습니다.

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

5. 플레이북을 실행할 때 추가 **Ansible** 변수를 설정하려면 **extra_vars** 속성을 사용하여 설정합니다.

자세한 내용은 ***director** 가이드를 사용하여 **Red Hat OpenStack Platform** 설치 및 관리의* 베어 메탈 노드 프로비저닝 속성을 참조하십시오.

참고

extra_vars 에 추가하는 변수는 **OVS-DPDK** 사용자 지정의 환경 파일 만들기에서 이전에 추가한 **OVS-DPDK** 컴퓨팅 노드의 동일한 역할별 매개변수여야 합니다.

예제

```
- 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. 선택 사항: TuneD 프로필인 cpu-partitioning-powersave 를 사용하여 패킷이 전달되지 않을 때 sleep 모드로 전환하도록 OVS-DPDK를 구성할 수 있습니다.

cpu-partitioning-powersave 를 구성하려면 베어 메탈 노드 정의 파일에 다음 행을 추가합니다.

```
...

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"

...
```

예제

```
- 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:
2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,3
1,33,35,37,39
    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. 생성한 베어 메탈 노드 정의 파일의 경로와 파일 이름을 기록해 둡니다. 나중에 **NIC**를 구성하고 노드를 프로비저닝할 때 오버클라우드 노드 프로비저닝 명령의 입력 파일로 이 파일을 사용합니다.

다음 단계

- 10.6절. “NIC 구성 템플릿 생성” 으로 이동합니다.

추가 리소스

- OVS-DPDK 배포에 전원 저장
- *director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리의 구성 가능 서비스 및 사용자 지정 역할

- NFV에 대해 테스트된 NIC

10.6. NIC 구성 템플릿 생성

RHOSP(Red Hat OpenStack Platform)와 함께 제공되는 샘플 Jinja2 템플릿의 사본을 수정하여 NIC 구성 템플릿을 정의합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. stack 사용자로 언더클라우드에 로그인합니다.

2. stackrc 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 샘플 네트워크 구성 템플릿을 복사합니다.

/usr/share/ansible/roles/tripleo_network_config/templates/ 디렉터리의 예제에서 NIC 구성 Jinja2 템플릿을 복사합니다. NIC 요구 사항에 가장 적합한 항목을 선택합니다. 필요에 따라 수정합니다.

4. NIC 구성 템플릿(예: single_nic_vlans.j2)에서 DPDK 인터페이스를 추가합니다.

참고

샘플 NIC 구성 템플릿 single_nic_vlans.j2 에서 노드는 VLAN의 트렁크로 하나의 단일 네트워크 인터페이스만 사용합니다. 기본 VLAN, 태그가 지정되지 않은 트래픽은 컨트롤 플레인이며 각 VLAN은 RHOSP 네트워크(내부 API, 스토리지 등)에 해당합니다.

예제

```
...
 - 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. 사용자 지정 네트워크 구성 템플릿(예: **single_nic_vlans.j2**)을 베어 메탈 노드 정의 파일(예: **10.5절. “베어 메탈 노드 정의 파일 생성”** 에서 생성한 **overcloud-baremetal-deploy.yaml**)에 추가합니다.

예제

```
- 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. 선택 사항: TuneD 프로필인 cpu-partitioning-powersave 를 사용하여 패킷이 전달되지 않을 때 sleep 모드로 전환하도록 OVS-DPDK를 구성할 수 있습니다.

   cpu-partitioning-powersave 를 구성하려면 NIC 구성 템플릿에 큐 크기를 설정해야 합니다.

   예제

```
...
 - 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. 생성한 NIC 구성 템플릿의 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 10.7절. “OVS-DPDK 인터페이스의 MTU 값 설정” 으로 이동합니다.

추가 리소스

- OVS-DPDK 배포에 전원 저장

10.7. OVS-DPDK 인터페이스의 MTU 값 설정

RHOSP(Red Hat OpenStack Platform)는 OVS-DPDK의 점보 프레임을 지원합니다. 점보 프레임의 최대 전송 단위(MTU) 값을 설정하려면 다음을 수행해야 합니다.

- 사용자 지정 환경 파일에서 네트워킹의 글로벌 MTU 값을 설정합니다.
- NIC 구성 템플릿에서 물리적 DPDK 포트 MTU 값을 설정합니다.

  이 값은 vhost 사용자 인터페이스에서도 사용됩니다.
- 컴퓨팅 노드의 모든 게스트 인스턴스 내에 MTU 값을 설정하여 구성의 마지막에 유사한 MTU 값이 있는지 확인합니다.

NIC가 DPDK PMD에 의해 제어되고 NIC 설정 템플릿에서 동일한 MTU 값을 설정하므로 물리적 NIC에 대한 특별한 구성이 필요하지 않습니다. 물리적 NIC에서 지원하는 최대 값보다 큰 MTU 값을 설정할 수 없습니다.

참고

VXLAN 패킷에는 헤더에 추가 50바이트가 포함됩니다. 이러한 추가 헤더 바이트를 기반으로 MTU 요구 사항을 계산합니다. 예를 들어 MTU 값 9000은 VXLAN 터널 MTU 값이 이러한 추가 바이트를 고려하여 8950임을 의미합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

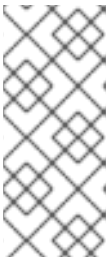
3. 10.3절. "OVS-DPDK 사용자 지정 환경 파일 생성" 에서 생성한 사용자 정의 환경 **YAML** 파일을 열거나 새 파일을 생성합니다.

4. **parameter_defaults** 에서 **NeutronGlobalPhysnetMtu** 매개변수를 설정합니다.

예제

이 예에서는 **NeutronGlobalPhysnetMtu** 가 **9000** 으로 설정됩니다.

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



참고

network-environment.yaml 파일의 **OvsDpdkSocketMemory** 값이 점보 프레임을 지원할 수 있을 만큼 충분히 큰지 확인합니다. 자세한 내용은 메모리 매개변수를 참조하십시오.

5. 10.6절. "NIC 구성 템플릿 생성" 에서 생성한 **NIC** 구성 템플릿(예: **single_nic_vlans.j2**)을 엽니다.

6. 브리지의 **MTU** 값을 컴퓨팅 노드로 설정합니다.

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

-

7. OVS-DPDK 본딩의 MTU 값을 설정합니다.

```
- 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. NIC 구성 템플릿의 경로와 파일 이름과 사용자 지정 환경 파일을 확인합니다. 오버클라우드를 배포할 때 나중에 이러한 파일을 사용합니다.

다음 단계

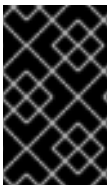
- 10.8절. “OVS-DPDK 인터페이스에 대한 멀티 큐 설정” 으로 이동합니다.

10.8. OVS-DPDK 인터페이스에 대한 멀티 큐 설정

로드 및 대기열 사용량을 기반으로 ELS (Non-isolated Poll Mode Drivers)에 대기열을 자동으로 로드 밸런싱하도록 OVS-DPDK 배포를 구성할 수 있습니다. Open vSwitch는 다음 시나리오에서 자동 큐 재조정을 트리거할 수 있습니다.

- pmd-auto-lb 값을 true 로 설정하여 RX 대기열의 사이클 기반 할당을 활성화했습니다.
- 두 개 이상의 분리되지 않은 PMD가 있습니다.

- 하나 이상의 큐가 격리되지 않은 **PMD**에 대해 폴링합니다.
- 집계된 **PMD**의 로드 값은 **1**분 동안 **95%**를 초과합니다.



중요

멀티 큐는 실험적이며 수동 대기열 고정에서만 지원됩니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

프로세스

1. **stack** 사용자로 언더클라우드에 로그인합니다.
2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 10.6절. “NIC 구성 템플릿 생성” 에서 생성한 **single_nic_vlans.j2** 와 같은 **NIC** 구성 템플릿을 엽니다.
4. 컴퓨팅 노드의 **OVS-DPDK**에서 인터페이스의 대기열 수를 설정합니다.

```
- 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. NIC 구성 템플릿의 경로와 파일 이름을 기록해 둡니다. 오버클라우드를 배포할 때 나중에 이 파일을 사용합니다.

다음 단계

- 10.9절. “노드 프로비저닝을 위한 DPDK 매개변수 구성” 으로 이동합니다.

10.9. 노드 프로비저닝을 위한 DPDK 매개변수 구성

RHOSP(Red Hat OpenStack Platform) OVS-DPDK 환경을 구성하여 OVS(Open vSwitch) Poll Mode Driver(PMD) 스레드를 자동으로 로드 밸런싱할 수 있습니다. 이를 위해 베어 메탈 노드 프로비저닝 및 오버클라우드 배포 중에 RHOSP director에서 사용하는 매개변수를 편집합니다.

OVS PMD 스레드는 사용자 공간 컨텍스트 전환에 대해 다음 작업을 수행합니다.

- 패킷의 입력 포트를 지속적으로 폴링합니다.
- 수신된 패킷을 분류합니다.
- 분류 후 패킷에서 작업을 실행합니다.

사전 요구 사항

- 언더클라우드 호스트 및 stack 사용자의 인증 정보에 액세스합니다.

프로세스

1. stack 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **10.5절. "베어 메탈 노드 정의 파일 생성"** 에서 생성한 베어 메탈 노드 정의 파일에서 매개변수를 설정합니다(예: **overcloud-baremetal-deploy.yaml**).

pmd_auto_lb

PMD 자동 로드 밸런싱을 활성화하려면 **true** 로 설정합니다.

pmd_load_threshold

PMD 스레드 중 하나가 **PMD** 로드 밸런싱을 트리거하기 전에 일관되게 사용해야 하는 처리 사이클의 백분율입니다. 정수, 범위 **0-100.**

pmd_improvement_threshold

PMD 자동 로드 밸런싱을 트리거하는 격리되지 않은 **PMD** 스레드에서 평가된 개선의 최소 백분율입니다. 정수, 범위 **0-100.**

예상 개선을 계산하기 위해 재할당의 예행 실행이 수행되고 예상 부하 분산이 현재 분산과 비교됩니다. 기본값은 **25%**입니다.

pmd_rebal_interval

연속된 **2**개의 **PMD** 자동 로드 밸런싱 작업 사이의 최소 시간(분)입니다. 범위 **0-20,000** 분.

트래픽 패턴이 변경될 수 있는 자주 재할당을 트리거하지 않도록 이 값을 구성합니다. 예를 들어 **10**분마다 또는 몇 시간마다 한 번씩 재할당을 트리거할 수 있습니다.

예제

```
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. 10.3절. “OVS-DPDK 사용자 지정 환경 파일 생성” 에서 생성한 사용자 정의 환경 YAML 파일을 열거나 새 파일을 생성합니다.

5. 사용자 지정 환경 파일에서 3단계에서 설정한 것과 동일한 베어 메탈 노드 사전 프로비저닝 값을 추가합니다. 다음과 같은 매개변수를 사용합니다.

OvsPmdAutoLb

pmd_auto_lb 와 동등한 heat입니다.

PMD 자동 로드 밸런싱을 활성화하려면 true 로 설정합니다.

OvsPmdLoadThreshold

pmd_load_threshold 와 동등한 heat입니다.

PMD 스레드 중 하나가 PMD 로드 밸런싱을 트리거하기 전에 일관되게 사용해야 하는 처리 사이클의 백분율입니다. 정수, 범위 0-100.

OvsPmdImprovementThreshold

pmd_improvement_threshold.

PMD 자동 로드 밸런싱을 트리거하는 격리되지 않은 PMD 스레드에서 평가된 개선의 최소 백분율입니다. 정수, 범위 0-100.

예상 개선을 계산하기 위해 재할당의 예행 실행이 수행되고 예상 부하 분산이 현재 분산과 비교됩니다. 기본값은 25%입니다.

OvsPmdRebalInterval

pmd_rebal_interval 과 동등한 heat입니다.

연속된 2개의 PMD 자동 로드 밸런싱 작업 사이의 최소 시간(분)입니다. 범위 0-20,000 분.

트래픽 패턴이 변경될 수 있는 자주 재할당을 트리거하지 않도록 이 값을 구성합니다. 예를 들어 10분마다 또는 몇 시간마다 한 번씩 재할당을 트리거할 수 있습니다.

예제

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

6. NIC 구성 템플릿의 경로와 파일 이름과 사용자 지정 환경 파일을 확인합니다. 베어 메탈 노드를 프로비저닝하고 오버클라우드를 배포할 때 나중에 이러한 파일을 사용합니다.

다음 단계

1. 네트워크 및 VIP를 프로비저닝합니다.

2. 베어 메탈 노드를 프로비저닝합니다.

   프로비저닝 명령을 실행하는 입력으로 overcloud-baremetal-deploy.yaml 과 같은 베어 메탈 노드 정의 파일을 사용해야 합니다.

3. 10.10절. “OVS-DPDK 오버클라우드 배포” 으로 이동합니다.

추가 리소스

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝.

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 네트워크 VIP 구성 및 프로비저닝

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 베어 메탈 노드 프로비저닝.

10.10. OVS-DPDK 오버클라우드 배포

OVS-DPDK 환경에서 RHOSP(Red Hat OpenStack Platform) 오버클라우드를 배포하는 마지막 단계는 **openstack overcloud deploy** 명령을 실행하는 것입니다. 명령에 대한 입력에는 사용자가 구성한 모든 다양한 오버클라우드 템플릿 및 환경 파일이 포함됩니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

- 이 섹션의 이전 절차에 나열된 모든 단계를 수행하고 **overcloud deploy** 명령에 입력으로 사용할 다양한 **heat** 템플릿 및 환경 파일을 모두 어셈블했습니다.

프로세스

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. **openstack overcloud deploy** 명령을 입력합니다.

openstack overcloud deploy 명령에 대한 입력을 특정 순서로 나열하는 것이 중요합니다. 일반 규칙은 먼저 기본 **heat** 템플릿 파일 뒤에 사용자 지정 환경 파일 및 사용자 지정 구성이 포함된 사용자 지정 템플릿을 지정합니다(예: 기본 속성 재정의).

다음 순서로 **openstack overcloud deploy** 명령에 입력을 추가합니다.

a. 오버클라우드의 **SR-IOV** 네트워크의 사양이 포함된 사용자 지정 네트워크 정의 파일(예: **network-data.yaml**)

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리의* 네트워크 정의 파일 구성 옵션을 참조하십시오.

b. **RHOSP director**에서 **SR-IOV** 환경을 배포하는 데 사용하는 **Controller** 및 **ComputeOvsDpdk** 역할이 포함된 역할 파일입니다.

예: **roles_data_compute_ovsdpdk.yaml**

자세한 내용은 **10.2**절. "역할 및 이미지 파일 생성"의 내용을 참조하십시오.

c. 오버클라우드 네트워크를 프로비저닝한 출력 파일입니다.

예: **overcloud-networks-deployed.yaml**

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 네트워크 정의 구성 및 프로비저닝 을 참조하십시오.

d. 오버클라우드 **VIP** 프로비저닝의 출력 파일입니다.

예: **overcloud-vip-deployed.yaml**

자세한 내용은 *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 오버클라우드의 네트워크 VIP 구성 및 프로비저닝* 을 참조하십시오.

e. 베어 메탈 노드 프로비저닝의 출력 파일입니다.

예: **overcloud-baremetal-deployed.yaml**

자세한 내용은 다음을 참조하십시오.

- 10.9절. “노드 프로비저닝을 위한 DPDK 매개변수 구성”.
- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드용 베어 메탈 노드 프로비저닝.

f. **director**에서 컨테이너 이미지를 가져올 위치와 저장 방법을 결정하는 데 사용하는 이미지 파일입니다.

예: **overcloud_images.yaml**

자세한 내용은 10.2절. “역할 및 이미지 파일 생성”의 내용을 참조하십시오.

g. 환경에서 사용하는 **Networking** 서비스(neutron) 메커니즘 드라이버 및 라우터 체계용 환경 파일입니다.

- **ML2/OVN**
  - **DCVR(Distributed virtual routing): neutron-ovn-dvr-ha.yaml**
  - 중앙 집중식 가상 라우팅: **neutron-ovn-ha.yaml**
- **ML2/OVS**
  - **DCVR(Distributed virtual routing): neutron-ovs-dvr.yaml**
  - 중앙 집중식 가상 라우팅: **neutron-ovs.yaml**

h. 메커니즘 드라이버에 따라 **OVS-DPDK**의 환경 파일입니다.

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

참고

SR-IOV 환경도 있고 동일한 노드에서 **SR-IOV** 및 **OVS-DPDK** 인스턴스를 찾으려면 배포 스크립트에 다음 환경 파일을 포함합니다.

- **ML2/OVN**

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

  neutron-sriov.yaml

i. 다음과 같은 구성이 포함된 하나 이상의 사용자 지정 환경 파일입니다.

- **OVS-DPDK** 환경의 기본 구성 값을 덮어씁니다.
- **VNF(**가상 네트워크 기능**)**로서의 방화벽.
- 점보 프레임의 최대 전송 단위**(MTU)** 값입니다.

  예**: ovs-dpdk-overrides.yaml**

자세한 내용은 다음을 참조하십시오.

- 10.3절. “OVS-DPDK 사용자 지정 환경 파일 생성”.
- 10.4절. “보안 그룹에 대한 방화벽 구성”.
- 10.7절. “OVS-DPDK 인터페이스의 MTU 값 설정”.

예제

샘플 **openstack overcloud deploy** 명령에서 발췌한 내용은 **DVR**을 사용하는 **OVS-DPDK, ML2/OVN** 환경에 대한 명령 입력 순서를 올바르게 정렬하는 방법을 보여줍니다.

```
$ 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. **openstack overcloud deploy** 명령을 실행합니다.

오버클라우드 생성이 완료되면 **RHOSP director**에서 오버클라우드 액세스에 도움이 되는 세부 정보를 제공합니다.

검증

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 배포 검증 단계를 수행합니다.

다음 단계

- 방화벽을 구성한 경우 **openstack port set** 명령을 실행하여 데이터 플레인 인터페이스에 대한 **OVS** 방화벽 드라이버를 비활성화합니다.

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

추가 리소스

- *director 가이드를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서* 오버클라우드 생성
- *명령줄 인터페이스 참조에* **overcloud deploy**

10.11. 플레이버 생성 및 OVS-DPDK의 인스턴스 배포

NFV를 사용하여 **Red Hat OpenStack Platform** 배포를 위해 **OVS-DPDK**를 구성한 후 플레이버를 생성하고 다음 단계를 사용하여 인스턴스를 배포할 수 있습니다.

1. 집계 그룹을 생성하고 **OVS-DPDK**에 대한 관련 호스트를 추가합니다. 정의된 플레이버 메타데이터와 일치하는 메타데이터(예: **dpdk=true**)를 정의합니다.

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

참고

고정된 **CPU** 인스턴스는 고정되지 않은 인스턴스와 동일한 컴퓨팅 노드에 있을 수 있습니다. 자세한 내용은 *인스턴스 생성 을 위해 컴퓨팅 서비스 구성의 컴퓨팅 노드에서 CPU 고정 구성을* 참조하십시오.

2. 플레이버를 만듭니다.

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

3. 플레이버 속성을 설정합니다. 정의된 메타데이터 **dpdk=true** 는 **DPDK** 집계에서 정의된 메타데이터와 일치합니다.

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

성능 향상을 위한 에뮬레이터 스레드 정책에 대한 자세한 내용은 *인스턴스 생성 을 위해 컴퓨팅 서비스 구성에서 에뮬레이터 스레드 구성*을 참조하십시오.

4. 네트워크를 만듭니다.

```
# 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. 선택 사항: **OVS-DPDK**와 함께 다중 큐를 사용하는 경우 인스턴스를 생성하는 데 사용할 이미지에 **hw_vif_multiqueue_enabled** 속성을 설정합니다.

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

6. 인스턴스를 배포합니다.

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

10.12. OVS-DPDK 구성 문제 해결

이 섹션에서는 **OVS-DPDK** 구성의 문제를 해결하는 단계를 설명합니다.

1. 브리지 구성을 검토하고 브리지에 **datapath_type=netdev** 가 있는지 확인합니다.

```
# 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. 선택적으로 컨테이너가 시작되지 않는 경우와 같이 오류 로그를 볼 수 있습니다.

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

3. **ovs-dpdk** 의 **Poll Mode** 드라이버 **CPU** 마스크가 **CPU**에 고정되어 있는지 확인합니다. 하이퍼 스레딩의 경우 형제 **CPU**를 사용합니다.

예를 들어 **CPU4** 의 형제를 확인하려면 다음 명령을 실행합니다.

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

CPU4 의 형제는 **CPU20** 이므로 다음 명령을 진행합니다.

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

상태를 표시합니다.

```
# 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
```

11장. RED HAT OPENSTACK PLATFORM 환경 튜닝

11.1. 에뮬레이터 스레드 고정

에뮬레이터 스레드는 가상 머신 하드웨어 에뮬레이션에 대한 인터럽트 요청 및 비차단 프로세스를 처리합니다. 이러한 스레드는 게스트가 처리하는 데 사용하는 CPU 전체에서 채워집니다. 이러한 게스트 CPU에서 폴링 모드 드라이버(PMD) 또는 실시간 처리에 사용되는 스레드가 패킷 손실 또는 누락된 데드라인이 발생할 수 있습니다.

결과적으로 자체 게스트 CPU에 스레드를 고정하여 에뮬레이터 스레드를 VM 처리 작업과 분리할 수 있습니다.

성능 향상을 위해 에뮬레이터 스레드를 호스팅하기 위해 호스트 CPU의 하위 집합을 예약합니다.

절차

1. 지정된 역할에 대해 NovaComputeCpuSharedSet 이 정의된 오버클라우드를 배포합니다. NovaComputeCpuSharedSet 의 값은 해당 역할 내의 호스트에 대한 nova.conf 파일의 cpu_shared_set 매개변수에 적용됩니다.

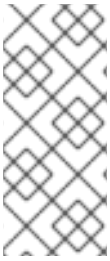
```
parameter_defaults:
  ComputeOvsDpdkParameters:
    NovaComputeCpuSharedSet: "0-1,16-17"
    NovaComputeCpuDedicatedSet: "2-15,18-31"
```

2. 플레이버를 만들어 에뮬레이터 스레드가 공유 풀로 구분된 인스턴스를 빌드합니다.

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

3. hw:emulator_threads_policy 추가 사양을 추가하고, 공유할 값을 설정합니다. 이 플레이버로 생성된 인스턴스는 nova.conf 파일의 cpu_share_set 매개변수에 정의된 인스턴스 CPU를 사용합니다.

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



참고

이 추가 사양에 대한 공유 정책을 활성화하려면 **nova.conf** 파일에 **cpu_share_set** 매개변수를 설정해야 합니다. **nova.conf** 를 수동으로 편집해도 재배포해도 유지되지 않을 수 있으므로 이 경우 **heat**를 사용해야 합니다.

검증

1. 지정된 인스턴스의 호스트 및 이름을 식별합니다.

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

2. **SSH**를 사용하여 식별된 호스트에 **tripleo-admin**으로 로그인합니다.

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

11.2. 가상 기능과 물리적 기능 간의 신뢰 구성

VF가 무차별 모드 활성화 또는 하드웨어 주소 수정과 같은 권한 있는 작업을 수행할 수 있도록 물리적 기능**(PF)**과 **VF**(가상 기능) 간 신뢰를 구성할 수 있습니다.

사전 요구 사항

- **director**를 포함한 **Red Hat OpenStack Platform**의 작동 설치

절차

물리적 기능과 가상 기능 간에 신뢰할 수 있는 오버클라우드를 구성하고 배포하려면 다음 단계를 완료합니다.

1. **parameter_defaults** 섹션에 **NeutronPhysicalDevMappings** 매개변수를 추가하여 논리 네트워크 이름과 물리적 인터페이스 간의 연결을 설정합니다.

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

2. 신뢰할 수 있는 새 속성을 **SR-IOV** 매개변수에 추가합니다.

```
parameter_defaults:
  NeutronPhysicalDevMappings:
    - sriov2:p5p2
  NovaPCIPassthrough:
    - vendor_id: "8086"
      product_id: "1572"
      physical_network: "sriov2"
      trusted: "true"
```

참고

"true" 값 주위에 큰따옴표를 포함해야 합니다.

11.3. 신뢰할 수 있는 VF 네트워크 사용

1. **vlan** 유형의 네트워크를 만듭니다.

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

2. 서브넷을 만듭니다.

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

3. 포트를 생성합니다. **vnic-type** 옵션을 **direct** 로 설정하고 **binding-profile** 옵션을 **true** 로 설정합니다.

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

4. 인스턴스를 생성하고 이전에 생성한 신뢰할 수 있는 포트에 바인딩합니다.

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

검증

하이퍼바이저에서 신뢰할 수 있는 VF 구성을 확인합니다.

1. 인스턴스를 생성한 컴퓨팅 노드에서 다음 명령을 입력합니다.

```
# 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. VF의 신뢰 상태가 신뢰할 수 있는지 확인합니다. 예제 출력에는 두 개의 포트가 포함된 환경에 대한 세부 정보가 포함되어 있습니다. vf 6 에는 에 대한 텍스트 신뢰가 포함되어 있습니다.

3. Networking 서비스(neutron) 네트워크에서 port_security_enabled: false 를 설정하거나 openstack port create 명령을 실행할 때 --disable-port-security 인수를 포함하는 경우 스푸핑 검사를 비활성화할 수 있습니다.

11.4. RX-TX 대기열 크기를 관리하여 패킷 손실 방지

다음과 같은 여러 가지 이유로 초당 3.5 만 개 이상의 패킷 손실 (mpps)에서 패킷 손실이 발생할 수 있습니다.

- 네트워크 인터럽트
- a SMI
- 가상 네트워크 기능의 패킷 처리 대기 시간

패킷 손실을 방지하려면 큐 크기를 기본값 512에서 최대 1024로 늘립니다.

사전 요구 사항

- 언더클라우드 호스트 및 **stack** 사용자의 인증 정보에 액세스합니다.

절차

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
$ source ~/stackrc
```

3. 사용자 지정 환경 **YAML** 파일을 생성하고 **parameter_defaults** 아래에 다음 정의를 추가하여 **RX** 및 **TX** 큐 크기를 늘립니다.

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

4. 배포 명령을 실행하고 코어 **heat** 템플릿, 기타 환경 파일, **RX** 및 **TX** 큐 크기 변경 사항이 포함된 환경 파일을 포함합니다.

예제

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

검증

1. **nova.conf** 파일에서 **RX** 큐 크기 및 **TX** 대기열 크기의 값을 관찰합니다.

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

다음이 표시되어야 합니다.

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

2. **Compute 호스트의 libvirt에서 생성된 VM 인스턴스 XML 파일에서 RX 큐 크기 및 TX 큐 크기의 값을 확인합니다.**

   a. **새 인스턴스를 만듭니다.**

   b. **컴퓨팅 호스트 및 인스턴스 이름을 가져옵니다.**

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

   샘플 출력

   출력은 다음과 유사합니다.

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

   c. **Compute 호스트에 로그인하고 인스턴스 정의를 덤프합니다.**

   예제

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

   샘플 출력

   출력은 다음과 유사합니다.

   ```
   ...
     <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. NUMA 인식 VSWITCH 구성

중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

NUMA 인식 **vSwitch**를 구현하기 전에 하드웨어 구성의 다음 구성 요소를 검사합니다.

- 물리적 네트워크 수입니다.
- **PCI** 카드 배치
- 서버의 물리적 아키텍처입니다.

PCIe NIC와 같은**MMIO(**메모리 매핑된 **I/O)** 장치는 특정 **NUMA** 노드와 연결됩니다. **VM**과 **NIC**가 다른 **NUMA** 노드에 있는 경우 성능이 크게 저하됩니다. 성능을 높이려면 동일한 **NUMA** 노드에서 **PCIe NIC** 배치 및 인스턴스 처리를 조정합니다.

이 기능을 사용하여 물리적 네트워크를 공유하는 인스턴스가 동일한 **NUMA** 노드에 있는지 확인합니다. 데이터 센터 하드웨어의 사용을 최적화하려면 여러 **physnet**을 사용해야 합니다.

주의

최적의 서버 사용률을 위해 NUMA 인식 네트워크를 구성하려면 PCIe 슬롯과 NUMA 노드의 매핑을 이해해야 합니다. 특정 하드웨어에 대한 자세한 내용은 벤더 설명서를 참조하십시오. NUMA 인식 vSwitch를 올바르게 계획하거나 구현하지 못하면 서버에서 단일 NUMA 노드만 사용하도록 할 수 있습니다.

NUMA 간 구성을 방지하려면 NIC의 위치를 Nova에 제공하여 VM을 올바른 NUMA 노드에 배치합니다.

사전 요구 사항

- filter NUMATopologyFilter 를 활성화했습니다.

절차

1. 새 NeutronPhysnetNUMANodesMapping 매개변수를 설정하여 물리적 네트워크를 물리적 네트워크와 연결하는 NUMA 노드에 매핑합니다.

2. VxLAN 또는 GRE와 같은 터널을 사용하는 경우 NeutronTunnelNUMANodes 매개변수도 설정해야 합니다.

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

예제

다음은 NUMA 노드 0으로 터널링된 두 개의 물리적 네트워크가 있는 예입니다.

- NUMA 노드 0과 연결된 프로젝트 네트워크 1개
- 유사성이 없는 하나의 관리 네트워크

```
parameter_defaults:
```

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

- **이 예에서는 eno2 라는 장치의 physnet을 NUMA 번호 0에 할당합니다.**

```
# 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
```

예제 heat 템플릿에서 physnet 설정을 확인합니다.

```
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
```

검증

다음 단계에 따라 NUMA 인식 vSwitch를 테스트합니다.

1. **/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf 파일의 구성을 확인합니다.**

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

2. **lscpu 명령을 사용하여 새 구성을 확인합니다.**

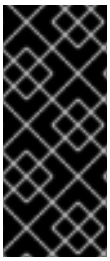
```
$ lscpu
```

3. 적절한 네트워크에 연결된 NIC를 사용하여 VM을 시작합니다.

추가 리소스

- NUMA 노드 토폴로지 검색
- 11.6절. “NUMA 인식 vSwitch에 대한 알려진 제한 사항”

11.6. NUMA 인식 VSWITCH에 대한 알려진 제한 사항



중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 Red Hat에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

이 섹션에는 RHOSP(Red Hat OpenStack Platform) 네트워크 기능 가상화 인프라(NFVi)에서 NUMA 인식 vSwitch를 구현하기 위한 제약 조건이 나열되어 있습니다.

- 두 개의 노드 게스트 NUMA 토폴로지를 지정하지 않은 경우 두 개의 NIC가 다른 NUMA 노드의 physnets에 연결된 VM을 시작할 수 없습니다.
- 두 노드 게스트 NUMA 토폴로지를 지정하지 않은 경우 하나의 NIC가 physnet에 연결되어 있고 다른 NIC가 다른 NUMA 노드의 터널링 네트워크에 연결된 VM을 시작할 수 없습니다.
- 두 개의 노드 게스트 NUMA 토폴로지를 지정하지 않은 경우 하나의 vhost 포트와 하나의 VF가 다른 NUMA 노드에 있는 VM을 시작할 수 없습니다.
- NUMA 인식 vSwitch 매개변수는 오버클라우드 역할에 따라 다릅니다. 예를 들어 컴퓨팅 노드 1 및 컴퓨팅 노드 2에는 NUMA 토폴로지가 다를 수 있습니다.
- VM의 인터페이스에 NUMA 선호도가 있는 경우 유사성이 단일 NUMA 노드에만 적용되는지 확인합니다. NUMA 노드에서 NUMA 선호도 없이 인터페이스를 찾을 수 있습니다.

- 네트워크 관리가 아닌 데이터 플레인 네트워크의 NUMA 선호도를 구성합니다.
- 터널링된 네트워크의 NUMA 선호도는 모든 VM에 적용되는 글로벌 설정입니다.

11.7. NFVI 환경에서 QOS(QUALITY OF SERVICE)

QoS(Quality of Service) 정책을 사용하여 네트워크 기능 가상화 인프라(NFVi)의 RHOSP(Red Hat OpenStack Platform) 네트워크에서 속도 제한을 적용하여 VM 인스턴스에 다양한 서비스 수준을 제공할 수 있습니다.

NFVi 환경에서 QoS 지원은 다음과 같은 규칙 유형으로 제한됩니다.

- 벤더에서 지원하는 경우 SR-IOV의 최소 대역폭 입니다.
- SR-IOV 및 OVS-DPDK 송신 인터페이스에 대한 대역폭 제한 입니다.

추가 리소스

- QoS(Quality of Service) 정책 구성

11.8. DPDK를 사용하는 HCI 오버클라우드 생성

최적화된 리소스 사용을 위해 Compute 및 Ceph Storage 서비스를 공동 배치하고 구성하여 하이퍼컨버지드 노드를 사용하여 NFV 인프라를 배포할 수 있습니다.

하이퍼 컨버지드 인프라(HCI)에 대한 자세한 내용은 *하이퍼컨버지드 인프라 배포를* 참조하십시오.

다음 섹션에서는 다양한 구성의 예를 제공합니다.

11.8.1. NUMA 노드 구성 예

성능 향상을 위해 NUMA-0 및 VNF와 같은 하나의 NUMA 노드에 테넌트 네트워크와 Ceph 개체 서비스 데몬(예: NUMA-1)을 다른 NUMA 노드에 배치하십시오.

CPU 할당:

NUMA-0	NUMA-1
Ceph OSD 수 * 4 HT	VNF 및 비NFV VM용 게스트 vCPU
DPDK lcore - 2 HT	DPDK lcore - 2 HT
DPDK PMD - 2 HT	DPDK PMD - 2 HT

CPU 할당의 예:

	NUMA-0	NUMA-1
Ceph OSD	32,34,36,38,40,42,76,78,80,82,84,86	
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 구성 파일 예

이 섹션에서는 샘플 **Red Hat Ceph Storage** 구성 파일에 대해 설명합니다. **Red Hat OpenStack Platform** 환경에 적합한 값을 대체하여 구성 파일을 모델링할 수 있습니다.

```
[osd]
osd_numa_node = 0                    # 1
osd_memory_target_autotune = true    # 2

[mgr]
mgr/cephadm/autotune_memory_target_ratio = 0.2 # 3
```

다음 매개 변수를 사용하여 **Ceph OSD**(오브젝트 스토리지 데몬) 프로세스의 **CPU** 리소스를 할당합니다. 여기에 표시된 값은 예입니다. 워크로드 및 하드웨어에 따라 값을 적절하게 조정합니다.

1

osd_numa_node: Ceph 프로세스의 선호도를 **NUMA** 노드로 설정합니다(예: **NUMA-0** 의 경우 **0, NUMA-1** 의 경우1 등). **-1** 은 선호도를 **NUMA** 노드로 설정합니다.

이 예에서 **osd_numa_node** 는 **NUMA-0** 으로 설정됩니다. 11.8.3절. “DPDK 구성 파일의 예” 에 표시된 대로 **IsolCpusList** 에는 **OvsPmdCoreList** 의 요소가 제거된 후 **NUMA-1** 의 홀수 **CPU**가 포함되어 있습니다. 대기 시간에 민감한 **Compute** 서비스(**nova**) 워크로드는 **NUMA-1** 에서 호스팅되므로 **NUMA-0** 에서 **Ceph** 워크로드를 격리해야 합니다. 이 예에서는 **stroage** 네트워크의 디스크 컨트롤러 및 네트워크 인터페이스가 **NUMA-0** 에 있다고 가정합니다.

2

osd_memory_target_autotune: true로 설정하면 **OSD** 데몬이 **osd_memory_target** 구성 옵션을 기반으로 메모리 사용을 조정합니다.

3

autotune_memory_target_ratio: OSD에 메모리를 할당하는 데 사용됩니다. 기본값은 **Cryostat** 입니다.

시스템의 총 **RAM**의 **70%**는 시작점이며, 자동 조정되지 않은 **Ceph** 데몬이 사용하는 메모리는 모두 차감됩니다. **osd_memory_target_autotune** 이 모든 **OSD**에 대해 **true**인 경우 나머지 메모리는 **OSD**로 나뉩니다. **HCI** 배포의 경우 **Compute** 서비스에 더 많은 메모리를 사용할 수 있도록 **mgr/cephadm/autotune_memory_target_ratio** 를 **0.2** 로 설정할 수 있습니다. 각 **OSD**에 **5GB** 이상의 메모리가 있는지 확인하기 위해 필요에 따라 조정합니다.

추가 리소스

- 11.8.6절. “HCI-DPDK 오버클라우드 배포”

11.8.3. DPDK 구성 파일의 예

```
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

2

IsolCpusList: 이 매개변수를 사용하여 호스트 프로세스에서 격리할 **CPU** 코어 세트를 할당합니다. **OvsPmdCoreList** 매개변수의 값을 **NovaComputeCpuDedicatedSet** 매개변수의 값에 추가하여 **IsolCpusList** 매개변수 값을 계산합니다.

3

OvsDpdkSocketMemory: OvsDpdkSocketMemory 매개변수를 사용하여 **NUMA** 노드당 **hugepage** 풀에서 사전 할당하기 위해 **MB** 단위로 메모리 양을 지정합니다. **OVS-DPDK** 매개변수 계산에 대한 자세한 내용은 **OVS-DPDK** 매개변수를 참조하십시오.

4

OvsPmdCoreList: 이 매개변수를 사용하여 **DPDK** 폴링 모드 드라이버(PMD)에 사용되는 **CPU** 코어를 지정합니다. **DPDK** 인터페이스의 로컬 **NUMA** 노드와 연결된 **CPU** 코어를 선택합니다. 각 **NUMA** 노드에 대해 **2**개의 **HT** 형제 스레드를 할당하여 **OvsPmdCoreList** 매개변수 값을 계산합니다.

11.8.4. nova 구성 파일의 예

```
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: NovaReservedHugePages 매개변수를 사용하여 **hugepage** 풀에서 메모리를 **MB**로 미리 할당합니다. 이는 **OvsDpdkSocketMemory** 매개변수 값과 동일한 메모리 합계입니다.

2

NovaReservedHostMemory: NovaReservedHostMemory 매개변수를 사용하여 호스트의 작업에 메모리를 **MB**로 예약합니다. 다음 지침을 사용하여 예약해야 하는 메모리 양을 계산합니다.

- 각 **OSD**에 대해 **5GB**입니다.

- 각 VM에 대해 0.5GB 오버헤드입니다.

- 일반 호스트 처리를 위한 4GB. NUMA 간 OSD 작업으로 인한 성능 저하를 방지하기 위해 충분한 메모리를 할당해야 합니다.

3

NovaComputeCpuDedicatedSet: OvsPmdCoreList 또는 NovaComputeCpuDedicatedSet 매개변수를 사용하여 Ceph_osd_docker_cpuset_cpus 에서 찾을 수 없는 CPU를 나열합니다. CPU는 DPDK NIC와 동일한 NUMA 노드에 있어야 합니다.

11.8.5. HCI-DPDK 배포에 권장되는 구성

표 11.1. HCI 배포를 위한 조정 가능한 매개변수

블록 장치 유형	장치당 OSD, 메모리, vCPU
NVMe	메모리 : 장치당 OSD OSD당 5GB: 장치당 4 vCPU: 3
SSD	메모리 : 장치당 OSD OSD당 5GB: 장치당 1 vCPU: 4
HDD	메모리 : 장치당 OSD OSD당 5GB: 장치당 1 vCPU: 1GB

다음 기능에 동일한 NUMA 노드를 사용합니다.

- 디스크 컨트롤러

- 스토리지 네트워크

- 스토리지 CPU 및 메모리

DPDK 공급자 네트워크의 다음 기능에 대해 다른 NUMA 노드를 할당합니다.

- NIC
- PMD CPU
- 소켓 메모리

11.8.6. HCI-DPDK 오버클라우드 배포

DPDK를 사용하는 하이퍼컨버지드 오버클라우드를 배포하려면 다음 단계를 따르십시오.

사전 요구 사항

- RHOSP(Red Hat OpenStack Platform) 17.1 이상.
- Red Hat Ceph Storage 6.1의 최신 버전입니다.

프로세스

1. Controller 및 ComputeHCIOvsDpdk 역할에 대한 roles_data.yaml 파일을 생성합니다.

```
$ openstack overcloud roles generate -o ~/<templates>/roles_data.yaml \
Controller ComputeHCIOvsDpdk
```

2. openstack flavor create 및 openstack flavor set 명령을 사용하여 새 플레이버를 생성하고 구성합니다.

3. RHOSP director 및 Ceph 구성 파일을 사용하여 Ceph를 배포합니다.

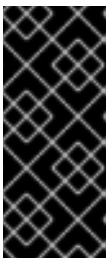
예제

```
$ openstack overcloud ceph deploy --config initial-ceph.conf
```

4. 생성한 사용자 지정 **roles_data.yaml** 파일을 사용하여 오버클라우드를 배포합니다.

예제

```
$ 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>
```



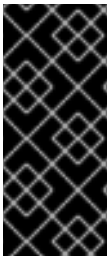
중요

이 예에서는 **Ceph RGW**(오브젝트 스토리지) 없이 **Ceph RBD**(블록 스토리지)를 배포합니다. 배포에 **RGW**를 포함하려면 **cephadm-rbd-only.yaml** 대신 **cephadm.yaml** 을 사용합니다.

추가 리소스

- *Red Hat OpenStack Platform 배포를 사용자 정의할 때 구성 가능 서비스 및 사용자 지정 역할*
- 11.8.2절. “Ceph 구성 파일 예”
- *director와 함께 Red Hat Ceph Storage 및 Red Hat OpenStack Platform 배포에서 Red Hat Ceph Storage 클러스터를 구성합니다.*

11.9. 컴퓨팅 노드를 TIMEMASTER와 동기화



중요

이 기능은 이번 릴리스에서 *기술 프리뷰*로 제공되므로 **Red Hat**에서 완전히 지원되지 않습니다. 테스트 용도로만 사용해야 하며 프로덕션 환경에 배포해서는 안 됩니다. 기술 프리뷰 기능에 대한 자세한 내용은 적용 범위 상세 정보를 참조하십시오.

시간 프로토콜을 사용하여 시스템 간 일관된 타임스탬프를 유지 관리합니다.

RHOSP(Red Hat OpenStack Platform)에는 **PTP(Precision Time Protocol)** 및 **NTP(Network Time Protocol)**가 지원됩니다.

NTP를 사용하여 밀리 초 범위에서 네트워크의 클럭을 동기화할 수 있으며 **PTP**를 사용하여 더 높은 마이크로초 단위의 정확도에 클럭을 동기화할 수 있습니다. **PTP**의 사용 사례는 가상 무선 액세스 네트워크(**vRAN**)로, 처리량이 증가하여 더 많은 간섭의 위험을 나타내는 여러 개의 **radio access network(vRAN)**가 포함되어 있습니다.

Timemaster는 **chronyd** 또는 **ntpd** 와 함께 **ptp4l** 및 **phc2sys** 를 사용하여 시스템 클럭을 **NTP** 및 **PTP** 시간 소스에 동기화하는 프로그램입니다. **phc2sys** 및 **ptp4l** 프로그램은 **SHM(Share Shared Memory Driver)** 참조 클럭을 사용하여 **PTP** 시간을 **chronyd** 또는 **ntpd** 에 전송하여 시스템 시계를 동기화하는 시간 소스를 비교합니다.

RHEL(Red Hat Enterprise Linux) 커널에서 **PTPv2** 프로토콜의 구현은 **linuxptp** 입니다.

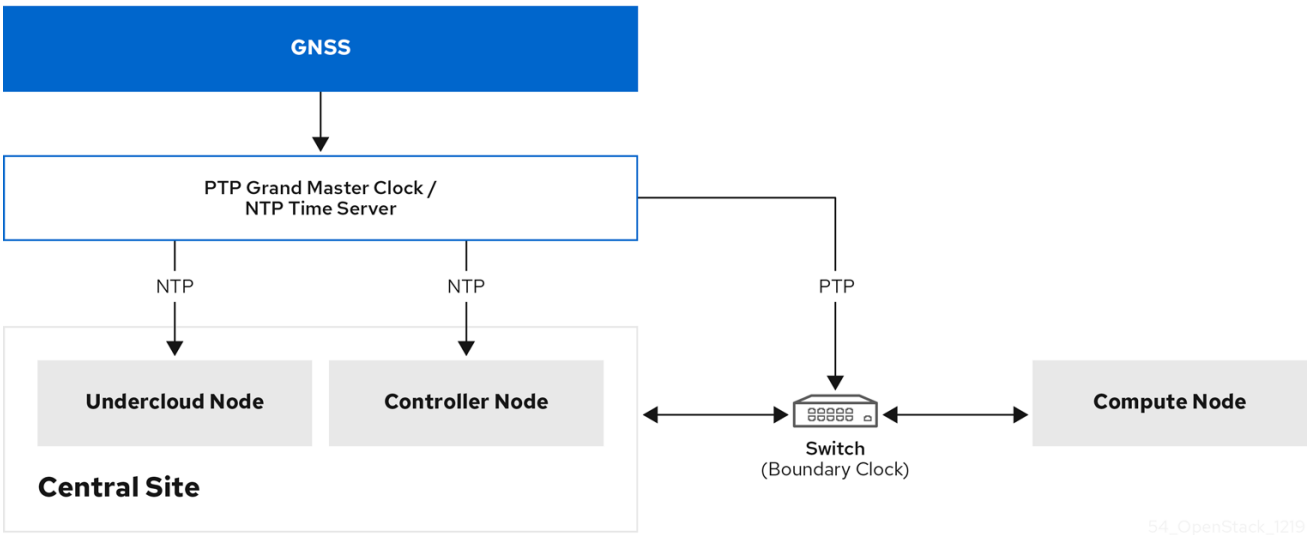
linuxptp 패키지에는 **PTP** 경계 클럭 및 일반 클럭 동기화를 위한 **ptp4l** 프로그램과 하드웨어 타임스탬프를 위한 **phc2sys** 프로그램이 포함되어 있습니다. **PTP**에 대한 자세한 내용은 ***Red Hat Enterprise Linux** 시스템 관리자 가이드의 **PTP** 소개 를 참조하십시오*.

chrony는 **NTP** 프로토콜의 구현입니다. **Chrony**의 두 가지 주요 구성 요소는 **Chrony** 데몬인 **chronyd** 와 **Chrony** 명령줄 인터페이스인 **chronyc** 입니다.

Chrony에 대한 자세한 내용은 ***Red Hat Enterprise Linux** 시스템 관리자 가이드에서 **Chrony** 모음을 사용하여 **NTP** 구성 을 참조하십시오*.

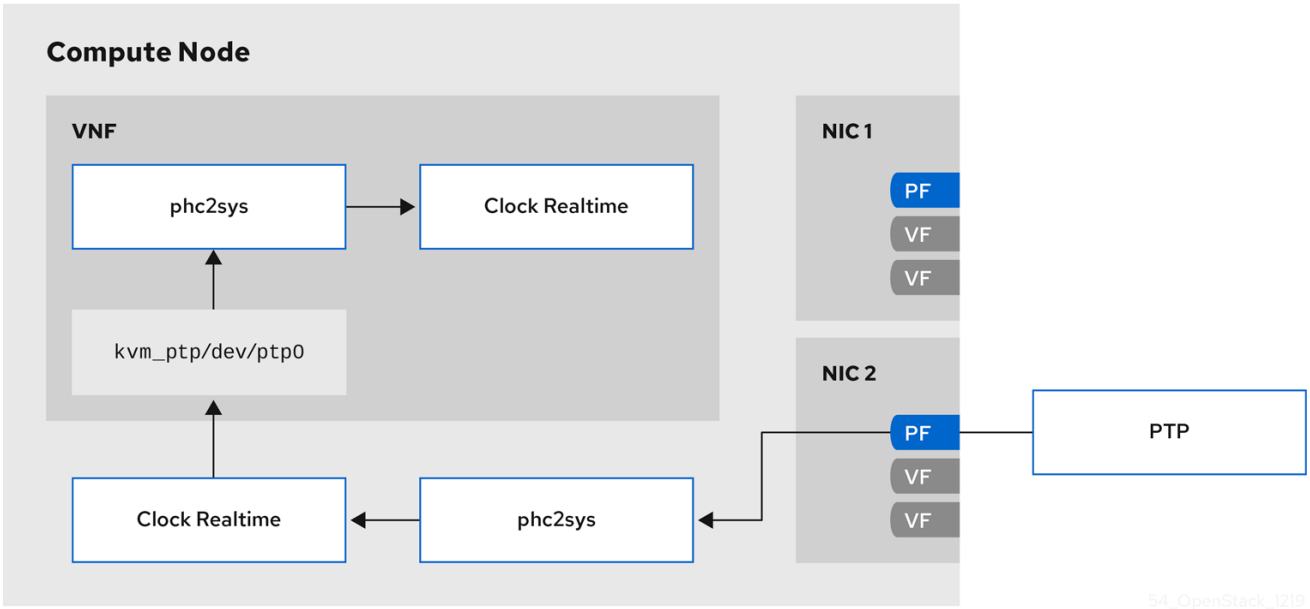
다음 이미지는 **PTP** 구성에서 패킷 이동에 대한 개요입니다.

그림 11.1. PTP 패킷 이동 개요



다음 이미지는 PTP 구성의 컴퓨팅 노드에서 패킷 이동에 대한 개요입니다.

그림 11.2. PTP 패킷 이동 세부 정보



11.9.1. Timemaster 하드웨어 요구 사항

다음과 같은 하드웨어 기능이 있는지 확인합니다.

- 하드웨어 타임스탬프 기능을 사용하여 NIC를 구성했습니다.

- 멀티 캐스트 패킷을 허용하도록 스위치를 구성했습니다.

- 스위치가 경계 또는 투명한 클록으로 작동하도록 구성되었습니다.

ethtool -T <device> 명령을 사용하여 하드웨어 타임스탬프를 확인할 수 있습니다.

```
$ 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)
```

투명성 또는 경계 클럭 스위치를 사용하여 정확도와 대기 시간을 줄일 수 있습니다. 경계 클록에 **uplink** 스위치를 사용할 수 있습니다. 경계 클럭 스위치는 **PTPv2** 헤더에서 **8**비트 **correctionField** 를 사용하여 지연 변형을 수정하고 최종 클록에서 정확성을 높입니다. 투명한 클럭 스위치에서 최종 클록은 **correctionField** 가 아닌 지연 변형을 계산합니다.

11.9.2. Timemaster 구성

오버클라우드 노드에서 시간 동기화를 위한 기본 **RHOSP(Red Hat OpenStack Platform)** 서비스는 **OS::TripleO::Services::Timesync** 입니다.

알려진 제한 사항

- 가상화된 컨트롤러의 **NTP**를 활성화하고 베어 메탈 노드에 **PTP**를 활성화합니다.

- **ptp4l** 에 호환되는 **PTP** 장치가 필요하기 때문에 **virtio** 인터페이스는 호환되지 않습니다.

-

SR-IOV가 있는 VM에 물리적 기능(PF)을 사용합니다. VF(가상 기능)는 PTP에 필요한 레지스터를 노출하지 않으며 VM은 kvm_ptp 를 사용하여 시간을 계산합니다.

- 여러 소스 및 여러 네트워크 경로가 있는 HA(고가용성) 인터페이스가 호환되지 않습니다.

프로세스

1. 선택한 역할에 속하는 노드에서 Timemaster 서비스를 활성화하려면 해당 역할의 roles_data.yaml 파일 섹션에서 OS::TripleO::Services::Timesync 가 포함된 행을 OS::TripleO::Services::TimeMaster 라인으로 교체합니다.

```
#- OS::TripleO::Services::Timesync
- OS::TripleO::Services::TimeMaster
```

2. 사용하는 compute 역할의 heat 매개변수를 구성합니다.

```
#Example
ComputeSriovParameters:
  PTPInterfaces: '0:eno1,1:eno2'
  PTPMessageTransport: 'UDPv4'
```

3. 사용자 환경과 관련된 다른 환경 파일과 함께 openstack overcloud deploy 명령에 새 환경 파일을 추가합니다.

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

…
-e <existing_overcloud_environment_files> \
-e <new_environment_file1> \
-e <new_environment_file2> \
…
```

- 을 기존 배포의 일부인 환경 파일 목록으로 바꿉니다 <existing_overcloud_environment_files>.

- <new_environment_file>을 오버클라우드 배포 프로세스에 포함할 새 환경 파일 또는 파일로 바꿉니다.

검증

- **ptp4linux 와 함께 설치된 phc_ctl 명령을 사용하여 NIC 하드웨어 클록을 쿼리합니다.**

```
# phc_ctl <clock_name> get
# phc_ctl <clock_name> cmp
```

11.9.3. timemaster 설정 예

```
$ 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 작업 예

```
$ 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
```

12장. NFV 워크로드용 RT-KVM 활성화

Red Hat Enterprise Linux Real Time KVM(RT-KVM)을 쉽게 설치하고 구성하기 위해 Red Hat OpenStack Platform은 다음과 같은 기능을 제공합니다.

- Red Hat Enterprise Linux를 실시간으로 프로비저닝하는 실시간 컴퓨팅 노드 역할.
- 추가 RT-KVM 커널 모듈.
- 컴퓨팅 노드의 자동 구성입니다.

12.1. RT-KVM 컴퓨팅 노드 계획

RT-KVM 컴퓨팅 노드를 계획할 때 다음 작업이 완료되었는지 확인합니다.

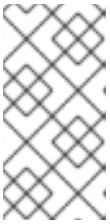
- RT-KVM 컴퓨팅 노드에 Red Hat 인증 서버를 사용해야 합니다.

  자세한 내용은 Red Hat Enterprise Linux for Real Time 인증 서버를 참조하십시오.

- 언더클라우드를 등록하고 유효한 Red Hat OpenStack Platform 서브스크립션을 연결합니다.

  자세한 내용은 다음을 참조하십시오. *언더클라우드 등록 및 director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 서브스크립션 연결*.

- RT-KVM의 rhel-9-server-nfv-rpms 리포지토리와 같이 언더클라우드에 필요한 리포지토리를 활성화하고 시스템 패키지를 최신 버전으로 업데이트합니다.



참고

이 리포지토리에 액세스하려면 별도의 Red Hat OpenStack Platform for Real Time SKU 서브스크립션이 필요합니다.

자세한 내용은 ***director** 를 사용하여 **Red Hat OpenStack Platform** 설치 및 관리에서 언더클라우드의 리포지토리 활성화*를 참조하십시오.

실시간 이미지 빌드

1. 언더클라우드에 **libguestfs-tools** 패키지를 설치하여 **virt-customize** 툴을 가져옵니다.

```
(undercloud) [stack@undercloud-0 ~]$ sudo dnf install libguestfs-tools
```

중요

언더클라우드에 **libguestfs-tools** 패키지를 설치하는 경우 언더클라우드에서 **tripleo_iscsid** 서비스와 포트 충돌을 방지하기 위해 **iscsid.socket** 을 비활성화합니다.

```
$ sudo systemctl disable --now iscsid.socket
```

2. 이미지를 추출합니다.

```
(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. 기본 이미지를 복사합니다.

```
(undercloud) [stack@undercloud-0 ~]$ cp overcloud-hardened-uefi-full.qcow2 overcloud-realtime-compute.qcow2
```

4. 사용자 지정과 관련된 **Red Hat** 리포지토리를 활성화하려면 이미지를 등록합니다. 다음 예제에서 **[username]** 및 **[password]** 를 유효한 인증 정보로 교체합니다.

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager register --username=[username] --password=[password]' \
subscription-manager release --set 9.0
```

참고

보안을 위해 명령 프롬프트에서 사용 중인 경우 기록 파일에서 인증 정보를 제거할 수 있습니다. **history -d** 명령과 행 번호를 사용하여 기록에서 개별 행을 삭제할 수 있습니다.

5. 계정의 서브스크립션에서 풀 **ID** 목록을 찾아 적절한 풀 **ID**를 이미지에 연결합니다.

```
sudo subscription-manager list --all --available | less
...
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager attach --pool [pool-ID]'
```

6. **NFV**를 사용하여 **Red Hat OpenStack Platform**에 필요한 리포지토리를 추가합니다.

```
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. 이미지에서 실시간 기능을 구성하는 스크립트를 생성합니다.

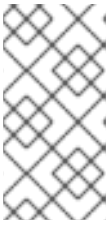
```
(undercloud) [stack@undercloud-0 ~]$ cat <<'EOF' > rt.sh
 #!/bin/bash

 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. 스크립트를 실행하여 실시간 이미지를 구성합니다.

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -
v --run rt.sh 2>&1 | tee virt-customize.log
```



참고

rt.sh 스크립트 출력에 다음 행이 표시되면 **"grubby fatal error: unable to find a suitable template". .** 에서는 이 오류를 무시할 수 있습니다.

9. 이전 명령에서 생성된 **virt-customize.log** 파일을 검사하여 **rt.sh** 스크립트를 사용하여 패키지가 올바르게 설치되었는지 확인합니다.

```
(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. **SELinux**의 레이블을 다시 지정합니다.

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 --selinux-relabel
```

11. **vmlinuz** 및 **initrd**를 추출합니다.

```
(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
```

참고

vmlinuz 및 **initramfs** 파일 이름의 소프트웨어 버전은 커널 버전에 따라 다릅니다.

12. 이미지를 업로드합니다.

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud image upload --update-existing --os-image-name overcloud-realtime-compute.qcow2
```

선택한 컴퓨팅 노드에서 **ComputeOvsDpdkRT** 구성 가능 역할과 함께 사용할 수 있는 실시간 이미지가 있습니다.

RT-KVM 컴퓨팅 노드의 **BIOS** 설정 수정

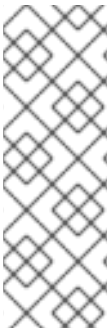
RT-KVM 컴퓨팅 노드의 대기 시간을 줄이려면 컴퓨팅 노드 **BIOS** 설정에서 다음 매개변수에 대한 모든 옵션을 비활성화합니다.

- 전원 관리
- **Hyper-Threading**
- **CPU** 절전 상태
- 논리 프로세서

12.2. RT-KVM을 사용하여 OVS-DPDK 구성

12.2.1. 실시간 컴퓨팅을 위한 노드 지정

실시간 컴퓨팅에 대한 노드를 지정하려면 새 역할 파일을 생성하여 **Real-time Compute** 역할을 구성하고 **Real-time Compute** 리소스 클래스로 베어 메탈 노드를 구성하여 컴퓨팅 노드를 실시간으로 태그합니다.



참고

다음 절차는 아직 프로비저닝하지 않은 새 오버클라우드 노드에 적용됩니다. 이미 프로비저닝된 기존 오버클라우드 노드에 리소스 클래스를 할당하려면 오버클라우드를 축소하여 노드를 프로비저닝 해제한 다음 오버클라우드를 확장하여 새 리소스 클래스 할당으로 노드를 다시 프로비저닝합니다. 자세한 내용은 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리에서** 오버클라우드 노드 스케일링을 참조하십시오.

프로세스

1. 언더클라우드 호스트에 **stack** 사용자로 로그인합니다.

2. **stackrc** 언더클라우드 인증 정보 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. **/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml** 파일을 기반으로 **ComputeRealTime** 역할의 매개변수를 설정하는 **compute-real-time.yaml** 환경 파일을 만듭니다.

4. **ComputeRealTime** 역할을 포함하는 **roles_data_rt.yaml** 이라는 새 역할 데이터 파일과 오버클라우드에 필요한 다른 역할 데이터 파일을 생성합니다. 다음 예제에서는 역할 **Controller, Compute , Compute RealTime** 을 포함하는 역할 데이터 파일 **roles_data_rt.yaml** 을 생성합니다.

```
(undercloud)$ openstack overcloud roles generate \
-o /home/stack/templates/roles_data_rt.yaml \
ComputeRealTime Compute Controller
```

5. **ComputeRealTime** 역할의 **roles_data_rt.yaml** 파일을 업데이트합니다.

```
###############################################################
# 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. 노드 정의 템플릿에 **node.json** 또는 **node.yaml.yaml**에 추가하여 오버클라우드의 **ComputeRealTime** 노드를 등록합니다.

자세한 내용은 *director 를 사용하여 Red Hat OpenStack Platform 설치 및 관리의 오버클라우드 노드 등록을* 참조하십시오.

7. 노드 하드웨어를 검사합니다.

```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

자세한 내용은 *director 를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 베어 메탈 노드 하드웨어 인벤토리 생성 을* 참조하십시오.

8. **ComputeRealTime**에 지정할 각 베어 메탈 노드에 사용자 지정 **ComputeRealTime** 리소스 클래스를 태그합니다.

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.RTCOMPUTE <node>
```

<node>를 베어 메탈 노드의 이름 또는 **UUID**로 바꿉니다.

9. 노드 정의 파일 **overcloud-baremetal-deploy.yaml** 에 **ComputeRealTime** 역할을 추가하고 노드에 할당할 예측 노드 배치, 리소스 클래스, 네트워크 토폴로지 또는 기타 속성을 정의합니다.

```
- 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>
```

- **< role_topology_file >**을 **ComputeRealTime** 역할에 사용할 토폴로지 파일의 이름으로 바꿉니다(예**: myRoleTopology.j2).** 기존 네트워크 토폴로지를 재사용하거나 역할에 대한 새 사용자 지정 네트워크 인터페이스 템플릿을 생성할 수 있습니다.

자세한 내용은 *director를 사용하여 Red Hat OpenStack Platform 설치 및 관리에서 사용자 정의 네트워크 인터페이스 템플릿* 정의를 참조하십시오. 기본 네트워크 정의 설정을

사용하려면 역할 정의에 **network_config** 를 포함하지 마십시오.

노드 정의 파일에서 노드 속성을 구성하는 데 사용할 수 있는 속성에 대한 자세한 내용은 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리에서** 베어 메탈 노드 프로비저닝 속성을 참조하십시오.

노드 정의 파일의 예는 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리의** 예제 노드 정의 파일을 참조하십시오.

10. 다음 **Ansible** 플레이북을 생성하여 노드 프로비저닝 중에 커널을 구성하고 플레이북을 **/home/stack/templates/fix_rt_kernel.yaml** 로 저장합니다.

```
# 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. 노드 프로비저닝 파일의 **ComputeOvsDpdkSriovRT** 역할 정의에 **/home/stack/templates/fix_rt_kernel.yaml** 을 플레이북으로 포함합니다.

```
- 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
```

노드 정의 파일에서 노드 속성을 구성하는 데 사용할 수 있는 속성에 대한 자세한 내용은 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리에서** 베어 메탈 노드 프로비저닝 속성을 참조하십시오.

노드 정의 파일의 예는 ***director*를 사용하여 *Red Hat OpenStack Platform* 설치 및 관리의** 예제 노드 정의 파일을 참조하십시오.

12. 역할의 새 노드를 프로비저닝합니다.

```
(undercloud)$ openstack overcloud node provision \
[--stack <stack> \ ]
[--network-config \]
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- 선택 사항: **<stack >**을 베어 메탈 노드가 프로비저닝되는 스택의 이름으로 바꿉니다. 기본값은 **overcloud** 입니다.
- 선택 사항: **--network-config** 선택적 인수를 포함하여 네트워크 정의를 **cli-overcloud-node-network-config.yaml Ansible** 플레이북에 제공합니다. **network_config** 속성을 사용하여 네트워크 정의를 정의하지 않으면 기본 네트워크 정의가 사용됩니다.
- **< deployment_file >**을 배포 명령에 포함할 **heat** 환경 파일의 이름으로 교체합니다(예: **/home/stack/templates/overcloud-baremetal-deployed.yaml).**

13. 별도의 터미널에서 프로비저닝 진행 상황을 모니터링합니다. 프로비저닝이 성공하면 노드 상태가 **available** 에서 **active** 로 변경됩니다.

```
(undercloud)$ watch openstack baremetal node list
```

14. **--network-config** 옵션 없이 **provisioning** 명령을 실행한 경우 **NIC** 템플릿 파일을 가리키도록 **network-environment.yaml** 파일에서 **< Role>NetworkConfigTemplate** 매개변수를 구성합니다.

```
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
```

< rt_compute >를 **ComputeRealTime** 역할의 네트워크 토폴로지가 포함된 파일의 이름으로 바꿉니다(예**:** 기본 네트워크 토폴로지를 사용하려면 **computert.yaml).**

15. 다른 환경 파일과 함께 스택에 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(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. OVS-DPDK 매개변수 구성

1. **parameter_defaults** 에서 터널 유형을 **vxlan** 으로 설정하고 네트워크 유형을 **vxlan,vlan:**로 설정합니다.

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

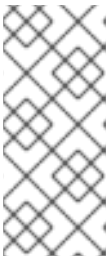
2. **parameters_defaults** 에서 브리지 매핑을 설정합니다.

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
 - dpdk-mgmt:br-link0
```

3. **parameter_defaults** 에서 **ComputeOvsDpdkSriov** 역할의 역할별 매개변수를 설정합니다.

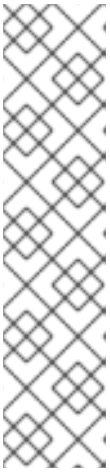
```
##########################
# OVS DPDK configuration #
##########################
ComputeOvsDpdkSriovParameters:
```

```
  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
```



참고

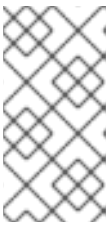
게스트 생성 중에 오류가 발생하지 않도록 하려면 각 **NUMA** 노드에 형제 스레드가 있는 하나 이상의 **CPU**를 할당합니다. 이 예에서 **OvsPmdCoreList** 매개변수 값은 **NUMA 0**의 코어 **2** 및 **22, NUMA 1**의 코어 **3** 및 **23**을 나타냅니다.



참고

이러한 대규모 페이지는 가상 머신에서 사용하고 이 절차에 표시된 대로 **OvsDpdkSocketMemory** 매개변수를 사용하는 **OVS-DPDK**에서도 사용됩니다. 가상 머신에 사용할 수 있는 대규모 페이지 수는 부팅 매개변수에서 **OvsDpdkSocketMemory** 를 뺀 값입니다.

DPDK 인스턴스와 연결된 플레이버에도 **hw:mem_page_size=1GB** 를 추가해야 합니다.



참고

OvsDpdkMemoryChannels 는 이 프로세스에 필요한 설정입니다. **optimal operation**을 위해 적절한 매개변수 및 값을 사용하여 **DPDK**를 배포하십시오.

4. **SR-IOV**의 역할별 매개변수를 구성합니다.

```
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. RT-KVM 인스턴스 시작

실시간 활성화된 컴퓨팅 노드에서 **RT-KVM** 인스턴스를 시작하려면 다음 단계를 수행합니다.

1. 오버클라우드에 **RT-KVM** 플레이버를 생성합니다.

```
$ 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. **RT-KVM** 인스턴스를 시작합니다.

```
$ openstack server create  --image <rhel> --flavor r1.small --nic net-id=<dpdk-net> test-rt
```

3. 인스턴스가 할당된 에뮬레이터 스레드를 사용하는지 확인하려면 다음 명령을 실행합니다.

```
$ 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>
```

13장. 예: VXLAN 터널링을 사용하여 OVS-DPDK 및 SR-IOV 구성

OVS-DPDK 및 SR-IOV 인터페이스를 모두 사용하여 컴퓨팅 노드를 배포할 수 있습니다. 클러스터에는 ML2/OVS 및 VXLAN 터널링이 포함됩니다.

중요

역할 구성 파일(예: roles_data.yaml)에서 오버클라우드 역할을 생성할 때 OS::TripleO::Services::Tuned 가 포함된 행을 주석 처리하거나 제거합니다.

```
ServicesDefault:
# - OS::TripleO::Services::Tuned
```

OS::TripleO::Services::Tuned 를 주석 처리하거나 제거한 경우 요구 사항에 맞게 TunedProfileName 매개변수를 설정할 수 있습니다(예: "cpu-partitioning "). OS::TripleO::Services::Tuned 행을 주석 처리하거나 제거하지 않으면 TunedProfileName 매개변수는 사용자가 설정한 다른 값 대신 "throughput-performance" 의 기본값을 가져옵니다.

13.1. 역할 데이터 구성

Red Hat OpenStack Platform은 roles_data.yaml 파일에서 기본 역할 세트를 제공합니다. 필요한 역할을 지원하기 위해 고유한 roles_data.yaml 파일을 생성할 수 있습니다.

이 예제의 목적을 위해 ComputeOvsDpdkSriov 역할이 생성됩니다.

추가 리소스

- *Red Hat OpenStack Platform 배포 사용자 지정*에서 새 역할 생성
- roles-data.yaml

13.2. OVS-DPDK 매개변수 구성

1. parameter_defaults 에서 터널 유형을 vxlan 으로 설정하고 네트워크 유형을 vxlan,vlan:로 설정합니다.

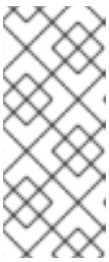
```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

2. parameters_defaults 에서 브리지 매핑을 설정합니다.

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
 - dpdk-mgmt:br-link0
```

3. parameter_defaults 에서 ComputeOvsDpdkSriov 역할의 역할별 매개변수를 설정합니다.

```
##########################
# 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
```



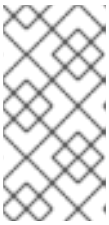
참고

게스트 생성 중에 오류가 발생하지 않도록 하려면 각 NUMA 노드에 형제 스레드가 있는 하나 이상의 CPU를 할당합니다. 이 예에서 OvsPmdCoreList 매개변수 값은 NUMA 0의 코어 2 및 22, NUMA 1의 코어 3 및 23을 나타냅니다.

참고

이러한 대규모 페이지는 가상 머신에서 사용하고 이 절차에 표시된 대로 OvsDpdkSocketMemory 매개변수를 사용하는 OVS-DPDK에서도 사용됩니다. 가상 머신에 사용할 수 있는 대규모 페이지 수는 부팅 매개변수에서 OvsDpdkSocketMemory 를 뺀 값입니다.

DPDK 인스턴스와 연결된 플레이버에도 hw:mem_page_size=1GB 를 추가해야 합니다.



참고

OvsDpdkMemoryChannels 는 이 프로세스에 필요한 설정입니다. **optimal operation**을 위해 적절한 매개변수 및 값을 사용하여 **DPDK**를 배포하십시오.

4. **SR-IOV**의 역할별 매개변수를 구성합니다.

```
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. 컨트롤러 노드 구성

1. 격리된 네트워크에 대한 **control-plane Linux** 본딩을 생성합니다.

```
- 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. 이 **Linux** 본딩에 **VLAN**을 할당합니다.

```
- 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. **OVS** 브리지를 만들어 **neutron-dhcp-agent** 및 **neutron-metadata-agent** 서비스에 액세스합니다.

```
- 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. DPDK 및 SR-IOV의 컴퓨팅 노드 구성

기본 **compute.yaml** 파일에서 **computeovsdpdksriov.yaml** 파일을 생성하고 다음과 같이 변경합니다.

1. 격리된 네트워크에 대한 control-plane Linux 본딩을 생성합니다.

```
- 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. 이 Linux 본딩에 VLAN을 할당합니다.

```
- 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. DPDK 포트가 있는 브릿지를 설정하여 컨트롤러에 연결합니다.

```
- 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
```

참고

여러 DPDK 장치를 포함하려면 추가하려는 각 DPDK 장치에 대해 유형 코드 섹션을 반복합니다.

참고

OVS-DPDK를 사용하는 경우 동일한 컴퓨팅 노드의 모든 브릿지는 ovs_user_bridge 여야 합니다. Red Hat OpenStack Platform은 동일한 노드에 있는 ovs_bridge 및 ovs_user_bridge 를 모두 지원하지 않습니다.

13.5. 오버클라우드 배포

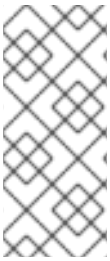
- overcloud_deploy.sh 스크립트를 실행합니다.

14장. NFV를 사용하여 RED HAT OPENSTACK 플랫폼 업그레이드

OVS-DPDK가 구성된 상태에서 **RHOSP(Red Hat OpenStack Platform)**를 업그레이드하는 방법에 대한 자세한 내용은 *업그레이드 프레임워크의 NFV(네트워크 기능 가상화 준비) 가이드를 참조하십시오 (16.2에서 17.1)* 가이드.

15장. 샘플 DPDK SR-IOV YAML 및 JINJA2 파일

이 섹션에서는 동일한 컴퓨팅 노드에 SR-IOV(Single Root I/O Virtualization) 및 DPDK(Data Plane Development Kit) 인터페이스를 추가하기 위한 참조로 샘플 yaml 파일을 제공합니다.



참고

이러한 템플릿은 완전히 구성된 환경에서 가져온 것이며 NFV와 관련이 없는 매개변수를 포함하며 배포에 적용되지 않을 수 있습니다. 구성 요소 지원 수준 목록은 Red Hat 지식베이스 솔루션 구성 요소 지원 검토를 참조하십시오.

15.1. ROLES_DATA.YAML

- openstack overcloud roles generate 명령을 실행하여 roles_data.yaml 파일을 생성합니다.

  Controller,ComputeSriov, ComputeOvsDpdkRT,ComputeOvsDpdkRT 또는 기타 역할과 같이 환경에 배포하려는 역할에 따라 명령에 역할 이름을 포함합니다.

  예제

  예를 들어 Controller 및 ComputeHCIOvsDpdkSriov 역할이 포함된 roles_data.yaml 파일을 생성하려면 다음 명령을 실행합니다.

```
$ 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
  - type: interface
    name: nic3

{% 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-tenant
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic4
    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-ex
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic5
    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
  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
  - type: interface
    name: nic3

{% 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: nic4
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: interface
            name: nic5
```

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
```