Red Hat OpenStack Platform 13

Red Hat OpenDaylight 설치 및 구성 가이드

Red Hat OpenStack Platform을 사용하여 OpenDaylight 설치 및 구성

Last Updated: 2023-05-25

Red Hat OpenStack Platform 13 Red Hat OpenDaylight 설치 및 구성 가이드

Red Hat OpenStack Platform을 사용하여 OpenDaylight 설치 및 구성

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

이 가이드에서는 Red Hat OpenDaylight 설치 및 구성에 대한 정보를 제공합니다.

차례

머리말

이 문서에서는 OpenDaylight 소프트웨어 정의 네트워크(SDN) 컨트롤러를 사용하여 Red Hat OpenStack Platform 13을 배포하는 방법을 설명합니다. OpenDaylight 컨트롤러는 neutron **ML2/OVS** 플러그인 및 **L2** 및 **L3** 에이전트의 드롭인 대체이며 Red Hat OpenStack 환경 내에서 네트워크 가상화를 제공합니다.

1장. 개요

1.1. OPENDAYLIGHT는 무엇입니까?

OpenDaylight 플랫폼은 OpenStack 환경을 위한 네트워크 가상화에 사용할 수 있는 Java로 작성된 프로그래밍 가능한 SDN 컨트롤러입니다. 컨트롤러 아키텍처는 분리된 northbound 및 southbound 인터페이스로 구성됩니다. OpenStack 통합을 위해 기본 northbound 인터페이스는 OpenStack Networking 서비스인 neutron과 통신하는 **NeutronNorthbound** 프로젝트를 사용합니다. **OVSDB** 및 **OpenFlow** 플러그인인 southbound OpenDaylight 프로젝트는 OVS(**Open vSwitch**) 컨트롤 및 데이터 플레인과 통신하는 데 사용됩니다. Neutron 구성을 네트워크 가상화로 변환하는 기본 OpenDaylight 프로젝트는 **NetVirt** 프로젝트입니다.

1.2. OPENDAYLIGHT는 OPENSTACK에서 어떻게 작동합니까?

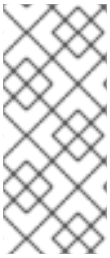
1.2.1. 기본 neutron 아키텍처

neutron 참조 아키텍처는 일련의 에이전트를 사용하여 OpenStack 내의 네트워크를 관리합니다. 이러한 에이전트는 다른 플러그인으로 neutron에 제공됩니다. 코어 플러그인은 *계층 2* 오버레이 기술과 데이터 플레인 유형을 관리하는 데 사용됩니다. 서비스 플러그인은 방화벽, DHCP, 라우팅, NAT와 같은 OSI 모델에서 *계층 3* 이상의 네트워크 작업을 관리하는 데 사용됩니다.

기본적으로 Red Hat OpenStack Platform은 각 컴퓨팅 및 컨트롤러 노드에서 OVS를 구성하는 에이전트를 제공하는 OVS 메커니즘 드라이버와 함께 *ML2(*Modular Layer 2) 코어 플러그인을 사용합니다. 서비스 플러그인인 DHCP 에이전트, 메타데이터 에이전트는 *L3* 에이전트와 함께 컨트롤러에서 실행됩니다.

1.2.2. OpenDaylight 기반 네트워킹 아키텍처

OpenDaylight는 **networking-odl** 이라는 자체 드라이버를 제공하여 *ML2* 코어 플러그인과 통합됩니다. 따라서 모든 노드에서 OVS 에이전트를 사용할 필요가 없습니다. OpenDaylight는 개별 노드에서 에이전트가 없어도 각 OVS 인스턴스를 환경에서 직접 프로그래밍할 수 있습니다. *계층 3* 서비스의 경우 neutron은 OpenDaylight *L3* 플러그인을 사용하도록 구성되어 있습니다. 이러한 접근 방식을 사용하면 OpenDaylight가 데이터 플레인을 직접 프로그래밍하여 분산 가상 라우팅 기능을 처리할 수 있기 때문에 라우팅 및 NAT(네트워크 주소 변환)를 처리하는 여러 노드의 에이전트 수가 줄어듭니다. neutron DHCP 및 메타데이터 에이전트는 여전히 DHCP 및 메타데이터(cloud-init) 요청을 관리하는 데 사용됩니다.



참고

OpenDaylight는 DHCP 서비스를 제공합니다. 그러나 현재 Red Hat OpenStack Platform director 아키텍처를 배포할 때 neutron DHCP 에이전트를 사용하면 HA(고가용성) 및 가상 머신(VM) 인스턴스 메타데이터(**cloud-init**)에 대한 지원을 제공하므로 Red Hat은 이러한 기능에 대해 OpenDaylight를 사용하지 않고 neutron DHCP 에이전트를 배포할 것을 권장합니다.

1.3. RED HAT OPENSTACK PLATFORM DIRECTOR란 무엇이며 어떻게 설계됩니까?

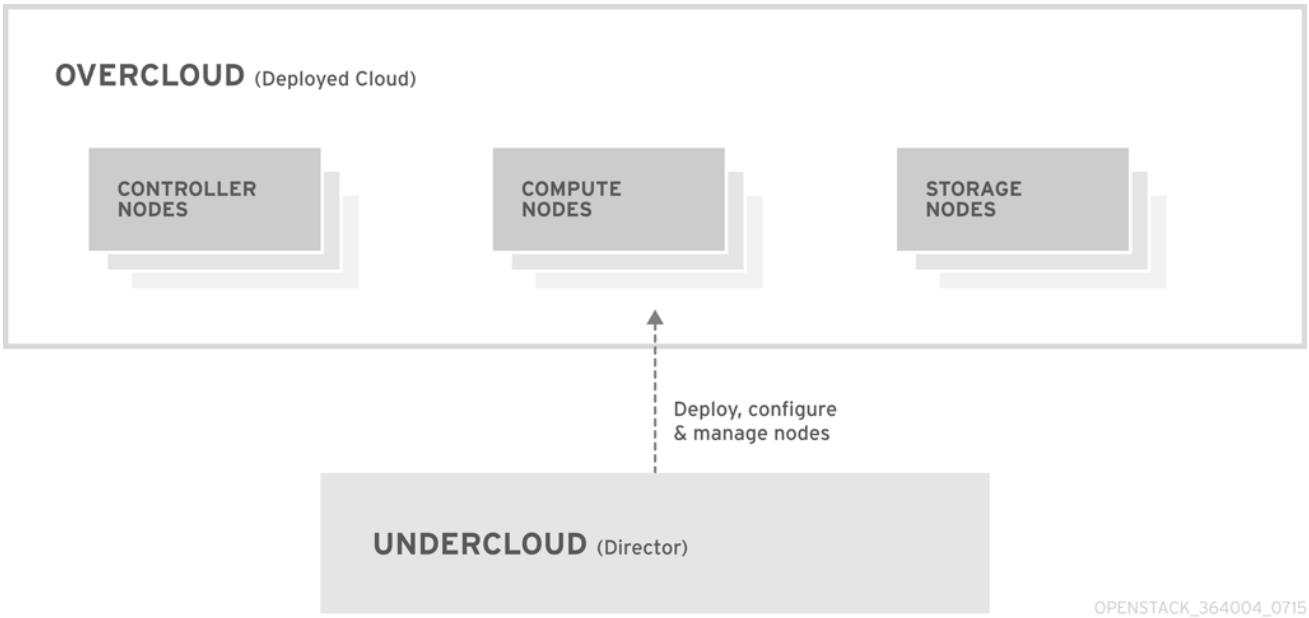
Red Hat OpenStack Platform director는 완전한 OpenStack 환경을 설치하고 관리하기 위한 툴셋입니다. 주로 OpenStack TripleO (OpenStack-On-OpenStack) 프로젝트를 기반으로 합니다.

이 프로젝트는 OpenStack 구성 요소를 사용하여 완전히 작동하는 OpenStack 환경을 설치합니다. 또한 베어 메탈 시스템을 프로비저닝하고 OpenStack 노드로 작동하는 새로운 OpenStack 구성 요소가 포함되어 있습니다. 이 방법을 사용하면 가볍고 강력한 완전한 Red Hat OpenStack Platform 환경을 설치할 수

있습니다.

Red Hat OpenStack Platform director는 *언더클라우드* 와 *오버클라우드* 의 두 가지 주요 개념을 사용합니다. 언더클라우드에서 오버클라우드를 설치 및 구성합니다. Red Hat OpenStack Platform director 아키텍처에 대한 자세한 내용은 Director Installation and Usage 를 참조하십시오.

그림 1.1. Red Hat OpenStack Platform 디렉터 -undercloud 및 오버클라우드



1.3.1. Red Hat OpenStack Platform director 및 OpenDaylight

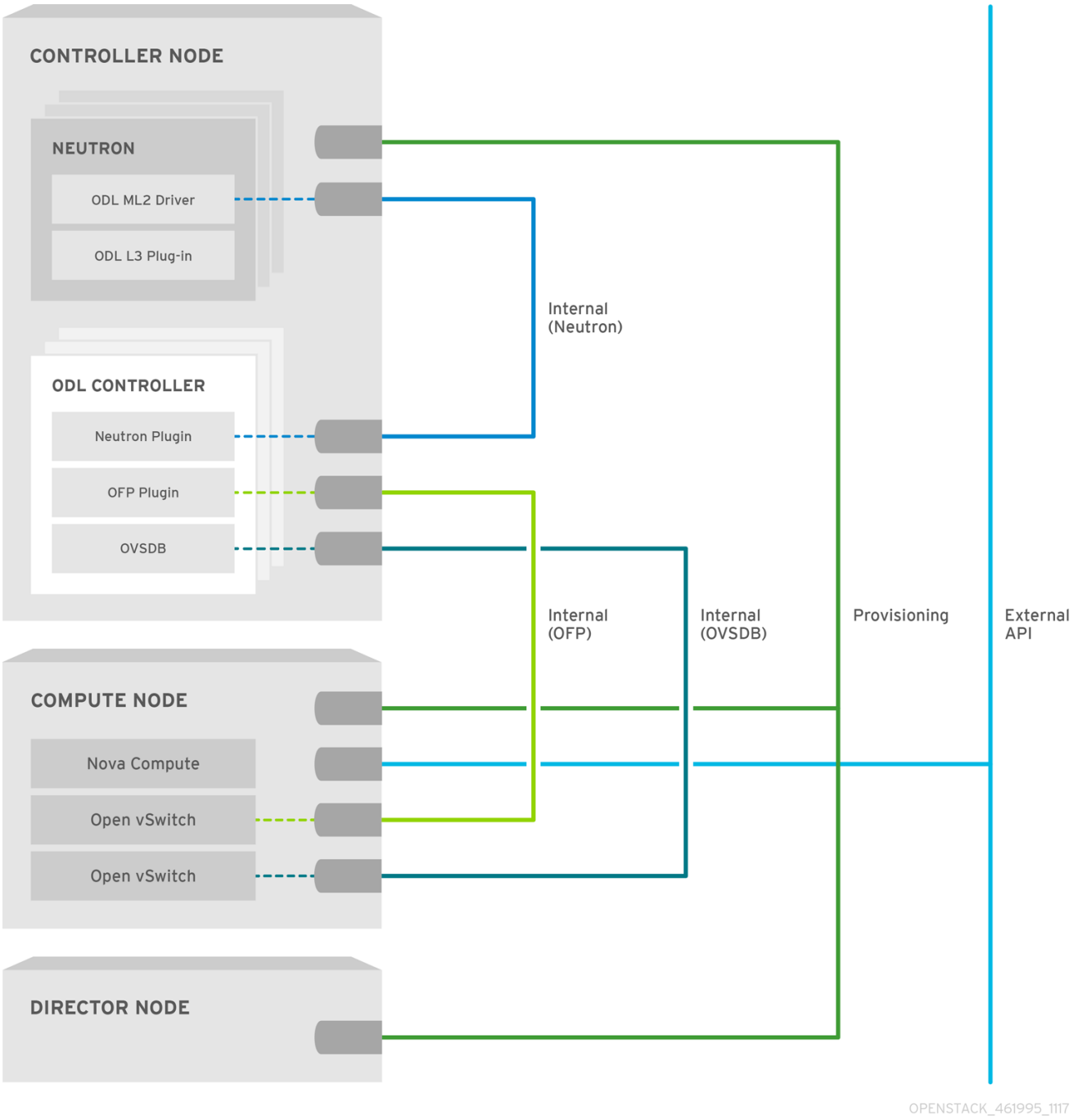
Red Hat OpenStack Platform director는 구성 가능 서비스 및 사용자 지정 역할에 대한 개념을 도입합니다. 구성 가능 서비스는 필요한 경우 역할별로 포함 및 활성화할 수 있는 격리된 리소스를 형성합니다. 사용자 지정 역할을 통해 사용자는 기본 컨트롤러 및 컴퓨팅 역할과 관계없이 자체 역할을 생성할 수 있습니다. 이제 배포할 OpenStack 서비스와 이를 호스팅할 노드를 선택할 수 있는 옵션이 있습니다.

OpenDaylight를 director와 통합하기 위해 다음 두 가지 서비스가 추가되었습니다.

- OpenDaylight *SDN* 컨트롤러를 실행하기 위한 **OpenDaylightApi** 서비스
- OpenDaylight는 각 노드에서 OVS를 설정하여 OpenDaylight와 올바르게 통신할 수 있도록 하는 **OpenDaylightOvs** 서비스입니다.

기본적으로 **OpenDaylightApi** 서비스는 컨트롤러 역할에서 실행되며 **OpenDaylightOvs** 서비스는 컨트롤러 및 Compute 역할에서 실행됩니다. OpenDaylight는 **OpenDaylightApi** 서비스 인스턴스 수를 확장하여 HA(*고가용성*)를 제공합니다. 기본적으로 컨트롤러 수를 3개 이상으로 스케일링하면 *HA* 가 자동으로 활성화됩니다. OpenDaylight *HA* 아키텍처에 대한 자세한 내용은 High Availability and Clustering with OpenDaylight 를 참조하십시오.

그림 1.2. OpenDaylight 및 OpenStack#189 기반 아키텍처



1.3.2. Red Hat OpenStack Platform director에서 네트워크 격리

Red Hat OpenStack Platform director는 개별 서비스를 사전 정의된 특정 네트워크 유형으로 구성할 수 있습니다. 이러한 네트워크 트래픽 유형은 다음과 같습니다.

IPMI	노드의 전원 관리 네트워크입니다. 언더클라우드를 설치하기 전에 이 네트워크를 설정해야 합니다.
프로비저닝(ctlplane)	director는 이 네트워크 트래픽 유형을 사용하여 *DHCP* 및 *PXE* 부팅을 통해 새 노드를 배포하고 오버클라우드 베어 메탈 서버에 OpenStack Platform 설치를 오케스트레이션합니다. 언더클라우드를 설치하기 전에 네트워크를 설정해야 합니다. 또는 ironic에서 직접 운영 체제 이미지를 배포할 수 있습니다. 이 경우 *PXE* 부팅이 필요하지 않습니다.

내부 API(internal_api)	*내부 API* 네트워크는 API 통신, RPC 메시지 및 데이터베이스 통신을 사용하는 OpenStack 서비스 간 통신 및 로드 밸런서의 내부 통신에 사용됩니다.
테넌트(테넌트)	Neutron은 각 테넌트에 *VLAN (각 테넌트 네트워크가 네트워크 VLAN임)* 또는 오버레이 터널을 사용하여 자체 네트워크를 제공합니다. 네트워크 트래픽은 각 테넌트 네트워크 내에서 격리됩니다. 터널을 사용하는 경우 여러 테넌트 네트워크가 충돌 없이 동일한 IP 주소 범위를 사용할 수 있습니다.

참고

GRE(Generic Routing Encapsulation) 및 VXLAN(Virtual eXtensible Local Area Network)은 코드베이스에서 사용할 수 있지만 *VXLAN* 은 OpenDaylight와 함께 사용하는 것이 권장되는 터널링 프로토콜입니다. *VXLAN* 은 RFC 7348 에 정의되어 있습니다. 이 문서의 나머지 부분에서는 터널링이 사용될 때마다 *VXLAN* 에 중점을 둡니다.

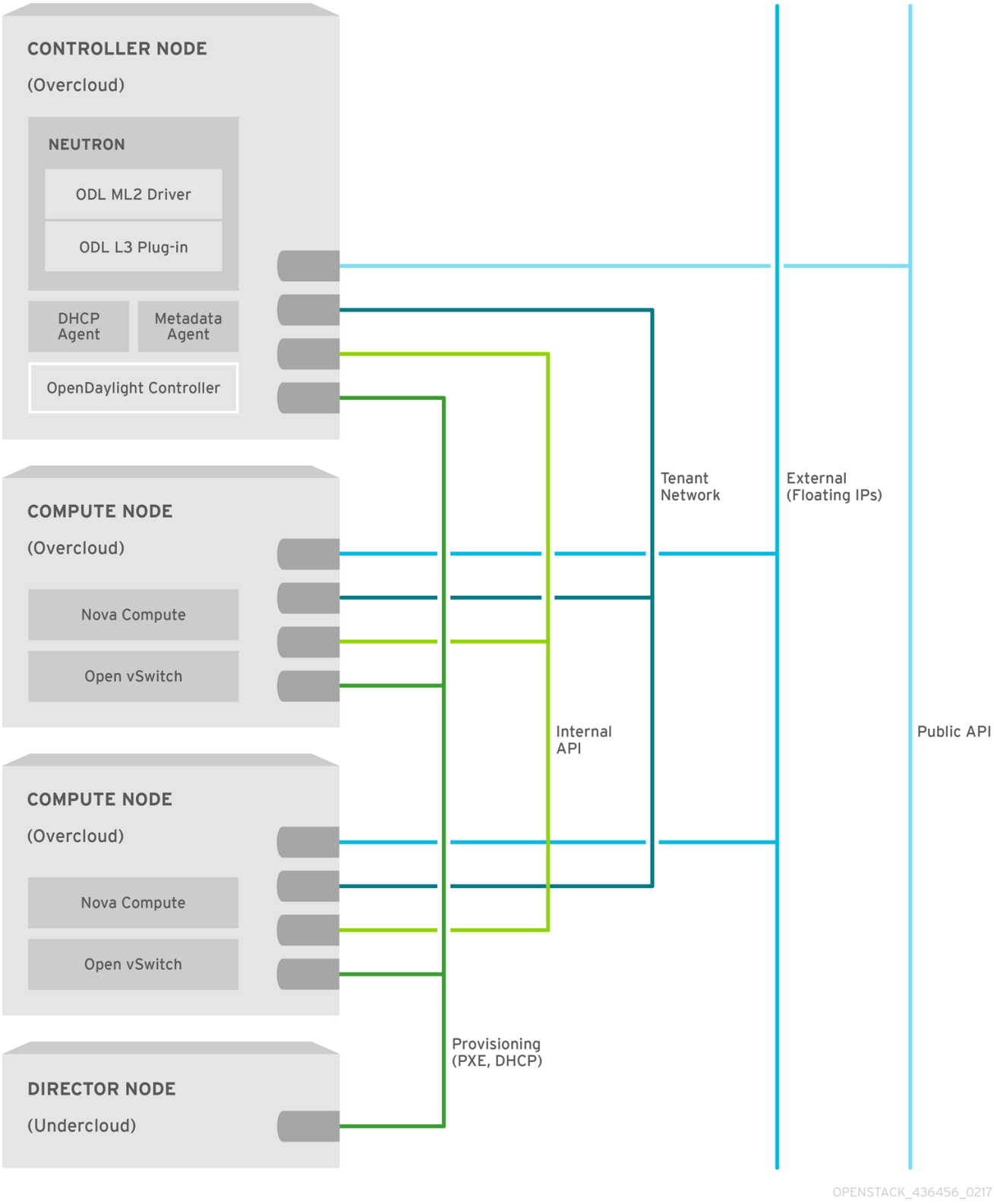
스토리지(스토리지)	블록 스토리지, NFS, iSCSI 등. 이상적으로 이는 성능 최적화를 위해 완전히 별도의 스위치 패브릭으로 격리됩니다.
스토리지 관리 (storage_mgmt)	OpenStack Object Storage(swift)는 이 네트워크를 사용하여 복제본 노드 간 데이터 오브젝트를 동기화합니다. 프록시 서비스는 사용자 요청과 기본 스토리지 계층 간의 중간 인터페이스 역할을 합니다. 프록시는 들어오는 요청을 수신하고 요청된 데이터를 검색하는 데 필요한 복제본을 찾습니다. *Ceph* 백엔드를 사용하는 서비스는 스토리지 관리 네트워크를 통해, *Ceph* 와 직접 상호 작용하지 않고 프런트 엔드 서비스를 사용하기 때문입니다. 이 트래픽이 *Ceph* 에 직접 연결되므로 **RBD** 드라이버가 예외입니다.
외부/공용 API	이 API는 그래픽 시스템 관리, OpenStack 서비스의 공용 API인 OpenStack 대시보드(horizon)를 호스팅하고 인스턴스로 들어오는 트래픽에 대해 SNAT를 수행합니다. 외부 네트워크에서 프라이빗 IP 주소(RFC-1918에 따라)를 사용하는 경우 인터넷에서 들어오는 트래픽에 대해 추가 NAT를 수행해야 합니다.
부동 IP	테넌트 네트워크의 인스턴스에 할당된 유동 IP 주소와 고정 IP 주소 간의 일대일 *IPv4* 주소 매핑을 사용하여 들어오는 트래픽이 인스턴스에 연결할 수 있습니다. 일반적인 구성은 별도의 유지 관리 대신 외부 및 유동 IP 네트워크를 결합하는 것입니다.
관리	SSH 액세스, DNS 트래픽 및 NTP 트래픽과 같은 시스템 관리 기능에 대한 액세스를 제공합니다. 이 네트워크는 또한 컨트롤러가 아닌 노드의 게이트웨이 역할을 합니다.

일반적인 Red Hat OpenStack Platform 설치에서는 종종 네트워크 유형의 수가 물리적 네트워크 연결 수를 초과합니다. 모든 네트워크를 적절한 호스트에 연결하기 위해 Overcloud에서 **802.1q** *VLAN* 태그를 사용하여 인터페이스당 두 개 이상의 네트워크를 제공할 수 있습니다. 대부분의 네트워크는 서브넷이 분리되어 있지만 일부는 인터넷 액세스 또는 인프라 네트워크 연결을 위해 라우팅을 제공하기 위해 *계층 3* 게이트웨이가 필요합니다.

OpenDaylight의 경우 관련 네트워크에는 **ServiceNetMap** *내부의 각 네트워크에 매핑되는 내부 API* 및 *테넌트* 서비스가 포함됩니다. 기본적으로 **ServiceNetMap** 은 **OpenDaylightApi** 네트워크를 *내부 API* 네트워크에 매핑합니다. 이 구성은 **OVS** 에 대한 southbound 트래픽뿐만 아니라 northbound 트래픽을 *내부 API* 네트워크로 격리한다는 것을 의미합니다.

OpenDaylight가 분산 라우팅 아키텍처를 사용하므로 각 컴퓨팅 노드가 유동 *IP* 네트워크에 연결되어 있어야 합니다. 기본적으로 Red Hat OpenStack Platform director는 *외부* 네트워크가 OVS 브리지 *br-ex* 에 매핑된 물리적 neutron 네트워크 *datacentre* 에서 실행됩니다. 따라서 컴퓨팅 노드 NIC 템플릿의 기본 구성에 *br-ex* 브릿지를 포함해야 합니다.

그림 1.3. OpenDaylight 및 OpenStack#189 네트워크 격리 예



1.3.3. 네트워크 및 방화벽 구성

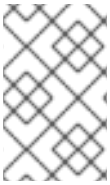
제한적인 방화벽이 있는 경우와 같은 일부 배포에서는 OpenStack 및 OpenDaylight 서비스 트래픽을 활성화하려면 방화벽을 수동으로 구성해야 할 수 있습니다.

기본적으로 OpenDaylight Northbound는 *8080* 포트를 사용합니다. *8080* 포트를 사용하는 swift 서비스와 충돌하지 않도록 Red Hat OpenStack Platform director와 함께 설치할 때 OpenDaylight 포트가 *8081*로 설정됩니다. Red Hat OpenDaylight 솔루션의 Southbound는 일반적으로 OVS 인스턴스가 연결되는 포트 *6640* 및 *6653* 에서 수신 대기하도록 구성됩니다.

OpenStack에서 각 서비스에는 일반적으로 고유한 VIP(가상 IP 주소)가 있으며 OpenDaylight는 동일한 방식으로 작동합니다. **HAProxy** 는 8081 포트 를 공용으로 열고 OpenStack에 이미 있는 플레인 VIP를 제어하도록 구성되어 있습니다. VIP 및 포트가 **ML2** 플러그인에 표시되고 neutron은 모든 통신을 전송합니다. OVS 인스턴스는 Southbound에 대해 OpenDaylight가 실행되고 있는 노드의 물리적 IP에 직접 연결합니다.

서비스	프로토콜	기본 포트	네트워크
OpenStack Neutron API	TCP	9696	내부 API
OpenStack Neutron API(SSL)	TCP	13696	내부 API
OpenDaylight Northbound	TCP	8081	내부 API
OpenDaylight Southbound: OVSDB	TCP	6640	내부 API
OpenDaylight Southbound: OpenFlow	TCP	6653	내부 API
OpenDaylight High Availability	TCP	2550	내부 API
VXLAN	UDP	4789	테넌트

표 1: 네트워크 및 방화벽 구성



참고

이 섹션에서는 OpenDaylight 통합과 관련된 서비스 및 프로토콜에 중점을 두고 완전한 것은 아닙니다. Red Hat OpenStack에서 실행되는 서비스에 필요한 전체 네트워크 포트 목록은 Red Hat OpenStack Platform용 방화벽 규칙 을 참조하십시오.

2장. OPENDAYLIGHT를 실행하려면 어떻게 해야 합니까?

다음 섹션에서는 OpenDaylight를 사용한 오버클라우드 배포 요구 사항에 대해 설명합니다. Red Hat OpenDaylight를 올바르게 설치하고 실행할 수 있는 충분한 컴퓨터 리소스가 있어야 합니다. 다음 정보를 사용하여 최소 요구 사항을 파악합니다.

2.1. 컴퓨팅 노드 요구 사항

컴퓨팅 노드는 가상 머신 인스턴스가 시작된 후 이를 실행하는 역할을 합니다. 모든 컴퓨팅 노드는 하드웨어 가상화를 지원해야 합니다. 또한 호스팅하는 가상 머신 인스턴스의 요구 사항을 지원하기에 충분한 메모리 및 디스크 공간이 있어야 합니다.

프로세서	Intel 64 또는 AMD64 CPU 확장 기능을 지원하고 AMD-V 또는 Intel VT 하드웨어 가상화 확장 기능이 활성화된 64비트 프로세서. 이 프로세서에 최소 4개의 코어가 있는 것이 좋습니다.
메모리	최소 6GB의 RAM. 가상 머신 인스턴스에서 사용하려는 메모리 크기에 따라 이 요구 사항에 RAM을 추가합니다.
디스크 공간	최소 40GB의 사용 가능한 디스크 공간
네트워크 인터페이스 카드	최소 1개의 1Gbps 네트워크 인터페이스 카드가 필요합니다. 프로덕션 환경에서 NIC(네트워크 인터페이스 카드)를 두 개 이상 사용하는 것이 좋습니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용하십시오. NIC에 대한 자세한 내용은 Tested NIC를 참조하십시오.
전원 관리	각 컨트롤러 노드에는 서버의 마더보드에서 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스가 필요합니다.

2.2. 컨트롤러 노드 요구 사항

컨트롤러 노드는 Red Hat OpenStack Platform 환경에서 horizon 대시보드, 백엔드 데이터베이스 서버, keystone 인증, 고가용성 서비스와 같은 핵심 서비스를 호스팅합니다.

프로세서	Intel 64 또는 AMD64 CPU 확장을 지원하는 64비트 프로세서입니다.

메모리	최소 메모리 용량은 20GB입니다. 그러나 권장 메모리 양은 CPU 코어 수에 따라 다릅니다. 다음 계산을 지침으로 사용하십시오. **컨트롤러의 최소 RAM 계산:** 코어당 1.5GB 메모리를 사용합니다. 예를 들어 코어가 48개인 머신에는 72GB RAM이 있어야 합니다. **컨트롤러의 권장 RAM 계산:** 코어당 3GB의 메모리를 사용합니다. 예를 들어 코어가 48개인 머신에는 144GB RAM이 있어야 합니다. 메모리 요구 사항 측정에 대한 자세한 내용은 Red Hat 고객 포털의 고가용성 컨트롤러에 대한 Red Hat OpenStack Platform 하드웨어 요구 사항 을 참조하십시오.
디스크 공간	최소 40GB의 사용 가능한 디스크 공간
네트워크 인터페이스 카드	최소 2개의 1GB의 네트워크 인터페이스 카드가 필요합니다. 본딩된 인터페이스나 태그된 VLAN 트래픽 위임에는 추가 네트워크 인터페이스 카드를 사용하십시오.
전원 관리	각 컨트롤러 노드에는 서버의 마더보드에서 IPMI(Intelligent Platform Management Interface) 기능과 같은 지원되는 전원 관리 인터페이스가 필요합니다.

3장. 오버클라우드에 OPENDAYLIGHT 설치

이 문서에서는 OpenDaylight 설치에만 중점을 둡니다. OpenDaylight를 배포하기 전에 작동 중인 언더클라우드 환경이 있고 오버클라우드 노드가 실제 네트워크에 연결되어 있는지 확인해야 합니다.

언더클라우드 및 오버클라우드 배포에 필요한 절차 를 설명하는 Director 설치 및 사용 가이드 의 CLI 툴을 사용하여 Undercloud 설치 및 기본 Overcloud 요구 사항 설치를 참조하십시오.

Red Hat OpenStack 플랫폼에 OpenDaylight를 설치하는 방법은 여러 가지가 있습니다. 다음 장에서는 OpenDaylight의 가장 유용한 시나리오와 설치 방법을 소개합니다.

3.1. 기본 구성 이해 및 설정 사용자 정의

OpenDaylight를 설치하는 데 권장되는 방법은 기본 환경 파일 **neutron-opendaylight.yaml** 을 사용하여 언더클라우드에서 배포 명령에 인수로 전달하는 것입니다. 그러면 OpenDaylight의 기본 설치가 배포됩니다.

기타 OpenDaylight 설치 및 구성 시나리오는 이 설치 방법을 기반으로 합니다. 배포 명령에 특정 환경 파일을 제공하여 다양한 시나리오를 사용하여 OpenDaylight를 배포할 수 있습니다.

3.1.1. 기본 환경 파일 이해

기본 환경 파일은 **/usr/share/openstack-tripleo-heat-templates/environments/services** 디렉터리의 **neutron-opendaylight.yaml** 입니다. 이 환경 파일은 OpenDaylight가 지원하는 서비스를 활성화하거나 비활성화합니다. 환경 파일은 배포 중에 director가 설정하는 필수 매개변수도 정의합니다.

다음 파일은 Docker 기반 배포에 사용할 수 있는 **neutron-opendaylight.yaml** 파일의 예입니다.

```
# A Heat environment that can be used to deploy OpenDaylight with L3 DVR using Docker containers
resource_registry:
  OS::TripleO::Services::NeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronCorePlugin: OS::Heat::None
  OS::TripleO::Services::OpenDaylightApi: ../../docker/services/opendaylight-api.yaml
  OS::TripleO::Services::OpenDaylightOvs: ../../puppet/services/opendaylight-ovs.yaml
  OS::TripleO::Services::NeutronL3Agent: OS::Heat::None
  OS::TripleO::Docker::NeutronMl2PluginBase: ../../puppet/services/neutron-plugin-ml2-odl.yaml

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronPluginExtensions: 'port_security'
  NeutronMechanismDrivers: 'opendaylight_v2'
  NeutronServicePlugins: 'odl-router_v2,trunk'
  OpenDaylightLogMechanism: 'console'
```

Red Hat OpenStack Platform director는 **resource_registry** 를 사용하여 배포에 대한 리소스를 해당 리소스 정의 yaml 파일에 매핑합니다. 서비스는 매핑할 수 있는 하나의 유형의 리소스입니다. 특정 서비스를 비활성화하려면 **OS::Heat::None** 값을 설정합니다. 기본 파일에서 **OpenDaylightApi** 및 **OpenDaylightOvs** 서비스는 활성화되어 있지만 OpenDaylight가 기능을 상속하면 기본 Neutron 에이전트가 명시적으로 비활성화됩니다.

heat 매개변수를 사용하여 director로 배포 설정을 구성할 수 있습니다. 환경 파일의 **parameter_defaults** 섹션을 사용하여 기본값을 재정의할 수 있습니다.

이 예에서 **NeutronEnableForceMetadata**,**NeutronMechanismDrivers** 및 **NeutronServicePlugins** 매개 변수는 OpenDaylight를 사용하도록 설정됩니다.

참고

기타 서비스 및 해당 구성 옵션 목록은 이 가이드의 뒷부분에 제공되어 있습니다.

3.1.2. OpenDaylight API 서비스 구성

필요에 맞게 **/usr/share/openstack-tripleo-heat-templates/puppet/services/opendaylight-api.yaml** 파일의 기본값을 변경할 수 있습니다. 이 파일의 설정을 직접 덮어쓰지 마십시오. 이 파일을 중복하고 백업 솔루션으로 원본을 유지합니다. 중복만 수정하고 중복을 배포 명령에 전달합니다.

참고

후자의 환경 파일의 매개변수는 이전 환경 파일에서 설정된 변수를 재정의합니다. 실수로 매개 변수를 덮어쓰지 않도록 환경 파일의 순서에 주의하십시오.

3.1.2.1. 구성 가능한 옵션

OpenDaylight **API 서비스**를 구성할 때 다음과 같은 몇 가지 매개변수를 설정할 수 있습니다.

OpenDaylightPort	Northbound 통신에 사용되는 포트입니다. 기본값은 **0**입니다. 이 매개변수는 OSP 13에서 더 이상 사용되지 않습니다.
OpenDaylightUsername	OpenDaylight의 로그인 사용자 이름입니다. 기본값은 **admin** 입니다.
OpenDaylightPassword	OpenDaylight의 로그인 암호입니다. 기본값은 **admin** 입니다.
OpenDaylightEnableDHCP	OpenDaylight가 DHCP 서비스로 작동하도록 활성화합니다. 기본값은 **false**입니다.
OpenDaylightFeatures	OpenDaylight에서 부팅할 쉼표로 구분된 기능 목록입니다. 기본값은 **[odl-netvirt-openstack, odl-jolokia]** 입니다.
OpenDaylightConnectionProtocol	REST 액세스에 사용되는 L7 프로토콜입니다. 기본값은 **http** 입니다.
OpenDaylightManageRepositories	OpenDaylight 리포지토리를 관리할지 여부를 정의합니다. 기본값은 **false**입니다.
OpenDaylightSNATMechanism	OpenDaylight에서 사용할 SNAT 메커니즘입니다. **conntrack** 또는 **controller** 를 선택합니다. 기본값은**conntrack** 입니다.
OpenDaylightLogMechanism	OpenDaylight를 위한 로깅 메커니즘입니다. **파일** 또는 **콘솔** 을 선택합니다. 기본값은 **file** 입니다.
OpenDaylightTLSKeystorePassword	OpenDaylight TLS 키 저장소의 암호입니다. 기본값은 **opendaylight** 입니다. 암호는 6자 이상이어야 합니다.

EnableInternalTLS	내부 네트워크에서 TLS를 활성화하거나 비활성화합니다. **true** 또는 **false** 값을 사용할 수 있습니다. 기본값은**false**입니다.
InternalTLSCAFile	내부 네트워크에서 서비스에 TLS를 활성화하는 경우 **InternalTLSCAFile** 매개변수를 사용하여 기본 CA 인증서를 지정해야 합니다. 기본값은 **/etc/ipa/ca.crt** 입니다.

TLS를 사용하여 배포하는 방법에 대한 자세한 내용은 Advanced Overcloud Customization Guide 를 참조하십시오.

3.1.3. OpenDaylight OVS 서비스 구성

필요에 맞게 **/usr/share/openstack-tripleo-heat-templates/puppet/services/opendaylight-ovs.yaml** 파일의 기본값을 변경할 수 있습니다. 이 파일의 설정을 직접 덮어쓰지 마십시오. 이 파일을 중복하고 백업 솔루션으로 원본을 유지합니다. 중복만 수정하고 중복을 배포 명령에 전달합니다.

참고

후자의 환경 파일의 매개변수는 이전 환경 파일에서 설정된 변수를 재정의합니다. 실수로 매개 변수를 덮어쓰지 않도록 환경 파일의 순서에 주의하십시오.

3.1.3.1. 구성 가능한 옵션

OpenDaylight **OVS 서비스** 를 구성할 때 다음과 같은 몇 가지 매개변수를 설정할 수 있습니다.

OpenDaylightPort	OpenDaylight와 Northbound 통신에 사용되는 포트입니다. 기본값은 **0**입니다. OVS 서비스는 Northbound를 사용하여 OpenDaylight를 쿼리하여 연결하기 전에 완전히 가동되고 있는지 확인합니다. 이 매개변수는 OSP 13에서 더 이상 사용되지 않습니다.
OpenDaylightConnectionProtocol	REST 액세스에 사용되는 계층 7 프로토콜입니다. 기본값은 **http** 입니다. **HTTP** 는 OpenDaylight에서 지원되는 유일한 프로토콜입니다. 이 매개변수는 OSP 13에서 더 이상 사용되지 않습니다.
OpenDaylightCheckURL	OVS가 연결되기 전에 OpenDaylight가 완전히 up인지 확인하는 URL입니다. 기본값은 **restconf/operational/network-topology:network-topology/topology/netvirt:1**입니다.
OpenDaylightProviderMappings	논리 네트워크와 물리적 인터페이스 간의 쉼표로 구분된 매핑 목록입니다. 이 설정은 VLAN 배포에 필요합니다. 기본값은 **datacentre:br-ex** 입니다.
OpenDaylightUsername	OpenDaylight OVS 서비스의 사용자 지정 사용자 이름입니다. 기본값은 **admin** 입니다.
OpenDaylightPassword	OpenDaylight OVS 서비스의 사용자 지정 암호입니다. 기본값은 **admin** 입니다.

HostAllowedNetworkTypes	이 OVS 호스트에 대해 허용된 테넌트 네트워크 유형을 정의합니다. 호스트 또는 역할에 따라 nova 인스턴스와 네트워크가 예약된 호스트를 제한할 수 있습니다. 기본값은 **['local', 'vlan', 'vxlan', 'gre', 'flat'입니다**.
OvsEnableDpdk	OVS에서 DPDK를 활성화하거나 비활성화합니다. 기본값은 **false** 입니다.
OvsVhostuserMode	vhostuser 포트 생성이 있는 OVS 모드입니다. 클라이언트 모드에서 하이퍼바이저는 vhostuser 소켓을 생성합니다. 서버 모드에서 OVS는 이를 생성합니다. 기본값은 **client** 입니다.
VhostuserSocketDir	vhostuser 소켓에 사용할 디렉터리입니다. 기본값은 **/var/run/openvswitch** 입니다.
OvsHwOffload	OVS Hardware Offload를 활성화하거나 비활성화합니다. **true** 또는 **false** 를 사용할 수 있습니다. 기본값은**false**입니다. 이 매개변수는 이 릴리스의 기술 프리뷰에 있습니다.
EnableInternalTLS	내부 네트워크에서 TLS를 활성화하거나 비활성화합니다. **true** 또는 **false** 값을 사용할 수 있습니다. 기본값은**false**입니다.
InternalTLSCAFile	내부 네트워크에서 서비스에 TLS를 활성화하는 경우 **InternalTLSCAFile** 매개변수를 사용하여 기본 CA 인증서를 지정해야 합니다. 기본값은 **/etc/ipa/ca.crt** 입니다.
ODLUpdateLevel	OpenDaylight 업데이트 수준입니다. **1** 또는 **2** 값을 사용할 수 있습니다. 기본값은 **1** 입니다.
VhostuserSocketGroup	vhost-user 소켓 디렉터리 그룹입니다. vhostuser가 기본 **dpdkvhostuserclient** 모드에 있는 경우 qemu는 vhost 소켓을 생성합니다. **VhostuserSocketGroup** 의 기본값은**qemu** 입니다.
VhostuserSocketUser	vhost-user 소켓 디렉터리 사용자 이름입니다. vhostuser가 기본 **dpdkvhostuserclient** 모드에 있는 경우 qemu는 vhost 소켓을 생성합니다. **VhostuserSocketUser** 의 기본값은**qemu** 입니다.

3.1.4. OpenDaylight에서 neutron 메타데이터 서비스 사용

OpenStack Compute 서비스를 사용하면 가상 머신에서 특수 주소 *169.254.169.254* 에 웹 요청을 수행하여 연결된 메타데이터를 쿼리할 수 있습니다. OpenStack Networking은 요청이 격리된 또는 IP 주소가 겹치는 여러 네트워크에서 발생하는 경우에도 **nova-api** 에 대한 요청을 프록시합니다.

메타데이터 서비스는 neutron L3 에이전트 라우터를 사용하여 메타데이터 요청 또는 DHCP 에이전트 인스턴스를 제공합니다. 계층 3 라우팅 플러그인이 활성화된 OpenDaylight를 배포하면 neutron L3 에이전트가 비활성화됩니다. 따라서 테넌트 네트워크에 라우터가 있는 경우에도 DHCP 인스턴스를 통해 전달하도록 메타데이터를 구성해야 합니다. 이 기능은 기본 환경 파일 **neutron-opendaylight.yaml** 에서 활성화되어 있습니다. 이를 비활성화하려면 **NeutronEnableForceMetadata** 를 **false** 로 설정합니다.

VM 인스턴스에는 DHCP 옵션 **121** 을 사용하여 **169.254.169.254/32** 의 정적 호스트 경로가 설치되어 있습

니다. 이 정적 경로가 배치되면 **169.254.169.254:80** 에 대한 메타데이터 요청이 DHCP 네트워크 네임스페이스의 메타데이터 이름 서버 프록시로 이동합니다. 그런 다음 네임스페이스 프록시는 인스턴스의 IP와 함께 HTTP 헤더를 요청에 추가하고 Unix 도메인 소켓을 통해 메타데이터 에이전트에 연결합니다. 메타데이터 에이전트는 소스 IP 및 네트워크 ID에 해당하는 인스턴스 ID에 대해 neutron을 쿼리하고 nova Metadata 서비스에 프록시합니다. 테넌트 간에 격리를 유지하고 중복되는 IP 지원을 허용하려면 추가 HTTP 헤더가 필요합니다.

3.1.5. 네트워크 구성 및 NIC 템플릿 이해

Red Hat OpenStack Platform director에서 물리적 neutron 네트워크 데이터 센터는 기본적으로 **br-ex** 라는 OVS 브릿지에 매핑됩니다. 이는 OpenDaylight 통합과 일관되게 동일합니다. 기본 **OpenDaylightProviderMappings** 를 사용하고 **flat** 또는 **VLAN _External** 네트워크를 생성하는 경우 컴퓨팅 노드의 NIC 템플릿에서 OVS br-ex 브릿지를 구성해야 합니다. 레이어 3 플러그인은 이러한 노드에 분산 라우팅을 사용하므로 더 이상 컨트롤러 역할 NIC 템플릿에서 br-ex를 구성할 필요가 없습니다.

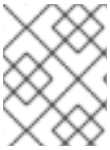
br-ex 브리지는 네트워크 격리의 모든 네트워크에 매핑할 수 있지만 일반적으로 예제에 표시된 대로 외부 네트워크에 매핑됩니다.

```
type: ovs_bridge
  name: {get_input: bridge_name}
  use_dhcp: false
  members:
    -
      type: interface
      name: nic3
      # force the MAC address of the bridge to this interface
      primary: true
  dns_servers: {get_param: DnsServers}
  addresses:
    -
      ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    -
      default: true
      ip_netmask: 0.0.0.0/0
      next_hop: {get_param: ExternalInterfaceDefaultRoute}
```

DPDK를 사용하여 일반적으로 **br-phy** 라는 다른 OVS 브리지를 만들고 ovs-dpdk-port를 제공해야 합니다. 브리지의 IP 주소는 VXLAN 오버레이 네트워크 터널용으로 구성됩니다.

```
type: ovs_user_bridge
      name: br-phy
      use_dhcp: false
      addresses:
          -
              ip_netmask: {get_param: TenantIpSubnet}
              members:
                  -
                      type: ovs_dpdk_port
                      name: dpdk0
                      driver: uio_pci_generic
                      members:
                          -
                              type: interface
```

```
        name: nic1
        # force the MAC address of the bridge to this interface
        primary: true
```



참고

네트워크 분리를 사용하는 경우 컴퓨팅 노드의 이 브릿지에 IP 주소 또는 기본 경로를 배치할 필요가 없습니다.

또는 **br-ex** 브리지를 사용하지 않고 외부 네트워크 액세스를 구성할 수도 있습니다. 이 방법을 사용하려면 오버클라우드 컴퓨팅 노드의 인터페이스 이름을 사전에 알고 있어야 합니다. 예를 들어 **eth3** 이 컴퓨팅 노드에서 세 번째 인터페이스의 결정적인 이름인 경우 이를 사용하여 컴퓨팅 노드의 NIC 템플릿에서 인터페이스를 지정할 수 있습니다.

```
-
  type: interface
  name: eth3
  use_dhcp: false
```

3.2. OPENDAYLIGHT의 기본 설치

이 섹션에서는 표준 환경 파일을 사용하여 OpenDaylight를 배포하는 방법을 설명합니다.

3.2.1. 오버클라우드에 대해 OpenDaylight 환경 파일 준비

시작하기 전에

- 언더클라우드를 설치합니다. 자세한 내용은 언더클라우드 설치를 참조하십시오.
- 필요한 경우 오버클라우드 및 OpenDaylight 설치 중에 사용하려는 컨테이너 이미지로 로컬 레지스트리를 생성합니다. 자세 한 내용은 *Director 설치 및 사용* 가이드의 컨테이너 이미지 소스 구성을 참조하십시오.

절차

1. 언더클라우드에 로그인하여 관리자 자격 증명을 로드합니다.

```
$ source ~/stackrc
```

2. OpenStack 및 OpenDaylight 설치에 필요한 Docker 컨테이너 이미지에 대한 참조가 포함된 Docker 레지스트리 파일 **odl-images.yaml** 을 생성합니다.

```
$ openstack overcloud container image prepare -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml --output-env-file /home/stack/templates/odl-images.yaml
```

이제 오버클라우드를 배포할 환경을 준비하고 3.2.2절. “OpenDaylight를 사용하여 오버클라우드 설치” 에 설명된 설치를 시작할 준비가 되었습니다.

더 많은 정보

openstack overcloud image prepare 명령은 오버클라우드 및 OpenDaylight를 설치할 컨테이너 이미지 환경 파일을 준비합니다. 이 명령은 다음 옵션을 사용합니다.

-e

OpenDaylight 및 OVS와 같이 해당 환경에 필요한 특정 컨테이너 이미지를 추가하는 서비스 환경 파일을 지정합니다.

--env-file

설치에 사용할 컨테이너 이미지 목록이 포함된 새 컨테이너 이미지 환경 파일을 생성합니다.

--pull-source

Docker 컨테이너 레지스트리의 위치 설정

--namespace

Docker 컨테이너의 버전을 설정합니다.

--prefix

이미지 이름에 접두사 추가

--suffix

이미지 이름에 접미사 추가

--tag

이미지 릴리스를 정의합니다.

3.2.2. OpenDaylight를 사용하여 오버클라우드 설치

시작하기 전에

- 오버클라우드 절차의 OpenDaylight 환경 파일 을 수행하여 배포에 필요한 환경 파일을 생성합니다.

절차

1. 언더클라우드에 로그인하여 관리자 자격 증명을 로드합니다.

```
$ source ~/stackrc
```

2. 이전에 생성된 환경 파일을 사용하여 오버클라우드를 배포합니다.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates \
 -e <other environment files>
 -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml \
 -e /home/stack/templates/odl-images.yaml
```

참고

배포 명령에 있는 환경 파일은 명령에 앞서 포함하는 환경 파일을 덮어씁니다. 실수로 매개변수를 덮어쓰지 않도록 포함하는 환경 파일의 순서에 주의하십시오.

작은 정보

변경하려는 매개변수만 설정하고 기본 환경 파일과 결합하는 최소 환경 파일을 생성하여 일부 매개변수를 재정의할 수 있습니다.

더 많은 정보

이 절차의 **openstack overcloud deploy** 명령에서는 다음 옵션을 사용합니다.

--templates

heat 템플릿 디렉터리의 경로를 정의합니다.

-e

환경 파일을 지정합니다.

3.3. 사용자 지정 역할에 OPENDAYLIGHT 설치

Custom 역할에 OpenDaylight를 설치하면 컨트롤러 노드와 다른 지정된 OpenDaylight 노드에서 실행되는 분리된 **OpenDaylightApi** 서비스가 생성됩니다.

OpenDaylight에 Custom 역할을 사용하려면 노드 레이아웃 및 기능 구성이 포함된 역할 파일을 만들어야 합니다.

3.3.1. 기본 역할에 따라 역할 파일 사용자 지정

사용자 정의 서비스 목록을 실행하는 각 역할에 대한 사용자 정의 역할 목록을 사용하여 OpenStack을 배포할 수 있습니다. 역할은 개별 서비스 또는 구성을 포함하는 노드 그룹입니다. 예를 들어 **nova API** 서비스가 포함된 *Controller* 역할을 생성할 수 있습니다. **openstack-tripleo-heat-templates** 에서 예제 역할을 볼 수 있습니다.

이러한 역할을 사용하여 오버클라우드 노드에 원하는 역할이 포함된 **roles_data.yaml** 파일을 생성합니다. 디렉터리에 개별 파일을 생성하여 사용자 지정 역할을 생성하고 이를 사용하여 새 **roles_data.yaml** 을 생성할 수도 있습니다.

특정 OpenStack 역할만 설치하는 사용자 지정 환경 파일을 생성하려면 다음 단계를 완료합니다.

절차

- admin 자격 증명을 로드합니다.

```
$ source ~/stackrc
```

- 사용자 지정 **roles_data.yaml** 파일을 생성하는 데 사용할 수 있는 기본 역할을 나열합니다.

```
$ openstack overcloud role list
```

- 이러한 역할을 모두 사용하려면 다음 명령을 실행하여 **roles_data.yaml** 파일을 생성합니다.

```
$ openstack overcloud roles generate -o roles_data.yaml
```

- 일부 역할만 포함하도록 역할 파일을 사용자 지정하려면 이전 단계의 명령에 인수로 역할 이름을 전달할 수 있습니다. 예를 들어 **컨트롤러**,**Compute** 및 **Telemetry** 역할을 사용하여 **roles_data.yaml** 파일을 생성하려면 다음 명령을 실행합니다.

```
$ openstack overcloud roles generate - roles_data.yaml Controller Compute Telemetry
```

3.3.2. OpenDaylight를 위한 사용자 정의 역할 생성

사용자 지정 역할을 생성하려면 역할 파일 디렉터리에 새 역할 파일을 생성하고 새 **roles_data.yaml** 파일을 생성합니다. 생성하는 각 사용자 지정 역할에 대해 새 역할 파일을 생성해야 합니다. 각 사용자 지정 역할 파일에는 특정 역할에 대한 데이터만 포함해야 하며 사용자 지정 역할 파일 이름은 역할 이름과 일치해야 합니다.

최소한 파일은 다음 매개변수를 정의해야 합니다.

- name **:** 역할의 이름을 정의합니다. 이름은 항상 비어 있지 않은 고유 문자열이어야 합니다.

```
- Name: Custom_role
```

- **ServicesDefault:** 이 역할에 사용된 서비스를 나열합니다. 사용된 서비스가 없는 경우 변수는 비어 있을 수 있습니다. 예제 형식은 다음과 같습니다.

```
ServicesDefault:
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
```

필수 매개변수 외에도 추가 설정을 정의할 수도 있습니다.

- **CountDefault:** 기본 노드 수를 정의합니다. **CountDefault:** 가 비어 있으면 기본값은 0입니다.

```
CountDefault: 1
```

- **HostnameFormatDefault:** 호스트 이름에 대한 형식 문자열을 정의합니다. 값은 선택 사항입니다.

```
HostnameFormatDefault: '%stackname%-computeovsdpdk-%index%'
```

- **설명 :** 역할에 대한 정보를 설명하고 추가합니다.

```
Description:
  Compute OvS DPDK Role
```

절차

1. 기본 역할 파일을 새 디렉터리에 복사하고 원본 파일을 백업으로 유지합니다.

```
$ mkdir ~/roles
$ cp /usr/share/openstack-tripleo-heat-templates/roles/* ~/roles
```

2. **~/roles** 의 **Controller.yaml** 파일에서 기본 Controller 역할을 수정하고 파일에서 **OpenDaylightApi** 행을 제거하여 컨트롤러 노드에서 **OpenDaylightAPI** 서비스를 비활성화합니다.

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
```

```
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::OpenDaylightApi #<--Remove this
    - OS::TripleO::Services::OpenDaylightOvs
```

3. **~/roles** 디렉터리에 새 **OpenDaylight.yaml** 파일을 생성하고 OpenDaylight 역할 설명을 추가합니다.

```
- name: OpenDaylight
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Ptp
    - OS::TripleO::Services::OpenDaylightApi
```

4. 파일을 저장합니다.

5. 사용자 지정 역할에서 OpenDaylight를 사용하여 OpenStack 오버클라우드를 배포할 때 사용할 새 역할 파일을 생성합니다.

```
$ openstack overcloud roles generate --roles-path ~/roles -o ~/roles_data.yaml Controller
Compute OpenDaylight
```

3.3.3. 사용자 정의 역할에 OpenDaylight를 사용하여 OverCloud 설치

시작하기 전에

- 언더클라우드를 설치합니다. 자세한 내용은 언더클라우드 설치를 참조하십시오.
- 오버클라우드 컨테이너 이미지에 대한 링크로 환경 파일을 생성합니다. 자세한 내용은 OpenDaylight를 사용하여 오버클라우드 설치 준비 를 참조하십시오.

- 사용자 지정 역할에 OpenDaylight를 구성하도록 역할 파일을 준비합니다. 자세한 내용은 OpenDaylight를 위한 사용자 지정 역할 만들기를 참조하십시오.

절차

1. 사용자 지정 역할을 생성합니다. 환경 파일에서 다음 매개변수 값을 설정합니다.

```
- OvercloudOpenDaylightFlavor: opendaylight
- OvercloudOpenDaylightCount: 3
```

   자세한 내용은 roles_data 파일 생성을 참조하십시오.

2. 기본 역할 정의를 재정의하려면 **-r** 인수와 함께 배포 명령을 실행합니다. 이 옵션은 사용자 지정 역할이 포함된 **roles_data.yaml** 파일을 사용하도록 배포 명령에 지시합니다. 이전 단계에서 생성한 **odl-composable.yaml** 환경 파일을 이 배포 명령에 전달합니다. 이 예제에는 총 3개의 ironic 노드가 있습니다. 사용자 정의 OpenDaylight 역할을 위해 하나의 ironic 노드가 예약되어 있습니다.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e /usr/share/openstack-tripleo-heat-templates/environments/docker.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e network-environment.yaml --compute-scale 1 --ntp-server 0.se.pool.ntp.org --control-flavor control --compute-flavor compute -r ~/roles_data.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
-e /home/stack/templates/odl-composable.yaml
```

참고

배포 명령에 있는 환경 파일은 명령에 앞서 포함하는 환경 파일을 덮어씁니다. 실수로 매개변수를 덮어쓰지 않도록 포함하는 환경 파일의 순서에 주의하십시오.

작은 정보

변경하려는 매개변수만 설정하고 기본 환경 파일과 결합하는 최소 환경 파일을 생성하여 일부 매개변수를 재정의할 수 있습니다.

더 많은 정보

- **-r** 옵션은 설치 시 역할 정의를 재정의합니다.

```
-r <roles_data>.yaml
```

- 사용자 정의 역할에는 설치 중에 추가 ironic 노드가 필요합니다.
- 사용자 지정 역할에 대한 rhosp13 composable 역할의 노드 카운터를 재정의하려면 다음 예의 구문을 사용하십시오. <role-name>Count: <value> 역할 이름 업데이트 role_data.yaml 파일에서 정확한 이름 세부 정보로 업데이트합니다.

3.3.4. 사용자 지정 역할에서 OpenDaylight 설치 확인

시작하기 전에

- 사용자 지정 역할에 OpenDaylight를 사용하여 Overcloud를 설치합니다. 자세한 내용은 사용자 지정 역할에 OpenDaylight를 사용하여 Overcloud 설치 를 참조하십시오.

절차

1. 기존 인스턴스를 나열합니다.

```
$ openstack server list
```

2. 새 OpenDaylight 역할이 인스턴스로 전용인지 확인합니다.

```
+--------------------------------------+------------------------+--------+------------+-------------+-------
-------------+
| ID                                   | Name                   | Status | Task State | Power State | Networks
|
+--------------------------------------+------------------------+--------+------------+-------------+-------
-------------+
| 360fb1a6-b5f0-4385-b68a-ff19bcf11bc9 | overcloud-controller-0   | BUILD  | spawning   |
NOSTATE     | ctlplane=192.0.2.4 |
| e38dde02-82da-4ba2-b5ad-d329a6ceaef1 | overcloud-novacompute-0  | BUILD  | spawning
| NOSTATE     | ctlplane=192.0.2.5 |
| c85ca64a-77f7-4c2c-a22e-b71d849a72e8 | overcloud-opendaylight-0 | BUILD  | spawning   |
NOSTATE     | ctlplane=192.0.2.8 |
+--------------------------------------+------------------------+--------+------------+-------------+-------
-------------+
```

3.4. SR-IOV 지원을 사용하여 OPENDAYLIGHT 설치

SR-IOV(*Single Root Input/Output Virtualization*)를 지원하는 컴퓨팅 노드와 함께 OpenDaylight를 배포할 수 있습니다. 이 배포에서 컴퓨팅 노드는 전용 SR-IOV 노드로 작동해야 하며 OVS를 기반으로 하는 nova 인스턴스와 혼합해서는 안 됩니다. 하나의 OpenDaylight 배포에 OVS 및 SR-IOV 컴퓨팅 노드를 둘 다 배포할 수 있습니다.

이 시나리오에서는 사용자 정의 SR-IOV Compute 역할을 사용하여 이러한 종류의 배포를 수행합니다.

SR-IOV 배포에서는 neutron SR-IOV 에이전트를 사용하여 VF(가상 기능)를 구성해야 합니다. 그런 다음 이러한 함수가 배포 시 직접 Compute 인스턴스에 전달되고 네트워크 포트로 사용됩니다. VF는 컴퓨팅 노드의 호스트 NIC에서 파생되므로 배포를 시작하기 전에 호스트 인터페이스에 대한 일부 정보가 필요합니다.

3.4.1. SR-IOV 컴퓨팅 역할 준비

OpenDaylight In Custom Role 에 표시된 것과 동일한 방법론을 따라 SR-IOV 기반 인스턴스를 생성할 수 있도록 SR-IOV 컴퓨팅 노드에 대한 사용자 정의 역할을 생성해야 기본 Compute 역할은 OVS 기반 Nova 인스턴스를 제공합니다.

시작하기 전에

- 사용자 지정 역할에서 OpenDaylight 설치 장에 대해 알아봅니다.

절차

1. 기본 역할 파일을 새 디렉터리에 복사하고 원본 파일을 백업으로 유지합니다.

```
$ mkdir ~/roles
$ cp /usr/share/openstack-tripleo-heat-templates/roles/* ~/roles
```

2. **~/roles** 디렉터리에 새 **ComputeSriov.yaml** 파일을 생성하고 다음 역할 설명을 추가합니다.

```
- name: ComputeSRIOV
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::Securetty
```

3. 파일을 저장합니다.

4. 기본 Compute 역할에서 **NeutronSriovAgent** 및 **NeutronSriovHostConfig** 서비스를 제거하고 해당 정보를 roles_data.yaml에 저장합니다.

```
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NeutronSriovAgent
```

5. OpenDaylight Compute SR-IOV 지원을 사용하여 OpenStack 오버클라우드를 배포하는 데 사용할 새 역할 파일을 생성합니다.

```
$ openstack overcloud roles generate --roles-path ~/roles -o ~/roles_data.yaml Controller
Compute ComputeSriov
```

6. 로컬 레지스트리를 생성합니다.

```
openstack overcloud container image prepare   --namespace=192.168.24.1:8787/rhosp13   -
-prefix=openstack-   --tag=2018-05-07.2
-e /home/stack/templates/environments/services-docker/neutron-opendaylight.yaml -e
/home/stack/templates/environments/services-docker/neutron-opendaylight-sriov.yaml --
output-env-file=/home/stack/templates/overcloud_images.yaml --roles-file
/home/stack/templates/roles_data.yaml
```

3.4.2. SR-IOV 에이전트 서비스 구성

SR-IOV 지원을 사용하여 OpenDaylight를 배포하려면 **neutron-opendaylight.yaml** 파일의 기본 매개변수를 재정의해야 합니다. **/usr/share/openstack-tripleo-heat-templates** 및 **neutron-opendaylight.yaml** 환경 파일에 있는 표준 SR-IOV 환경 파일을 사용할 수 있습니다. 그러나 원본 파일을 편집하지 않는 것이 좋습니다. 대신 원래 환경 파일을 복사하고 중복 파일에서 매개 변수를 수정합니다.

또는 변경하려는 매개변수만 제공하는 새 환경 파일을 만들고 배포에 두 파일을 모두 사용할 수도 있습니다. 사용자 지정 OpenDaylight를 배포하려면 두 파일을 모두 배포 명령에 전달합니다. 최신 환경 파일은

이전 설정을 재정의하므로 배포 명령에 올바른 순서로 포함해야 합니다. 올바른 순서는 먼저 **neutron-opendaylight.yaml** 이고, 그 다음 **neutron-opendaylight-sriov.yaml**.

기본 설정으로 OpenDaylight 및 SR-IOV를 배포하려면 Red Hat에서 제공하는 **neutron-opendaylight-sriov.yaml** 을 사용할 수 있습니다. 매개변수를 변경하거나 추가해야 하는 경우 기본 SR-IOV 환경 파일을 복사한 후 새로 생성된 파일을 편집합니다.

다음은 사용자 정의된 **neutron-opendaylight-sriov.yaml** 파일의 설명적인 예입니다.

```
# A Heat environment that can be used to deploy OpenDaylight with SRIOV
resource_registry:
  OS::TripleO::Services::NeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronOvsAgent: OS::Heat::None
  OS::TripleO::Services::ComputeNeutronCorePlugin: ../puppet/services/neutron-plugin-ml2.yaml
  OS::TripleO::Services::NeutronCorePlugin: ../puppet/services/neutron-plugin-ml2-odl.yaml
  OS::TripleO::Services::OpenDaylightApi: ../docker/services/opendaylight-api.yaml
  OS::TripleO::Services::OpenDaylightOvs: ../puppet/services/opendaylight-ovs.yaml
  OS::TripleO::Services::NeutronSriovAgent: ../puppet/services/neutron-sriov-agent.yaml
  OS::TripleO::Services::NeutronL3Agent: OS::Heat::None

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronPluginExtensions: 'port_security'
  NeutronMechanismDrivers: ['sriovnicswitch','opendaylight_v2']
  NeutronServicePlugins: 'odl-router_v2,trunk'

  # Add PciPassthroughFilter to the scheduler default filters
  #NovaSchedulerDefaultFilters:
['RetryFilter','AvailabilityZoneFilter','RamFilter','ComputeFilter','ComputeCapabilitiesFilter',
'ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter']

  #NovaSchedulerAvailableFilters:
["nova.scheduler.filters.all_filters","nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter"]

  #NeutronPhysicalDevMappings: "datacentre:ens20f2"

  # Number of VFs that needs to be configured for a physical interface
  #NeutronSriovNumVFs: "ens20f2:5"

  #NovaPCIPassthrough:
  #  - devname: "ens20f2"
  #    physical_network: "datacentre"
```

더 많은 정보

neutron-opendaylight-sriov.yaml 파일에서 다음 옵션을 구성할 수 있습니다. 표에서는 개별 옵션을 설명하고 SR-IOV 기능을 활성화하는 데 필요한 설정을 표시합니다.

NovaSchedulerDefaultFilters	SR-IOV에 PCI 패스스루를 사용할 수 있습니다. 환경 파일에서 주석 처리를 제거하고 **PciPassthroughFilter**를 포함해야 합니다.

NovaSchedulerAvailableFilters	Nova 기본 필터에 대한 PCI 패스스루 필터 지정을 활성화합니다. **nova.scheduler.filters.all_filters**를 설정하고 포함해야 합니다.
NeutronPhysicalDevMappings	논리적 neutron 네트워크를 호스트 네트워크 인터페이스에 매핑합니다. neutron이 가상 네트워크를 물리적 포트에 바인딩할 수 있도록 지정해야 합니다.
NeutronSriovNumVFs	호스트 네트워크 인터페이스에 생성할 VF 수입니다. 구문: &lt **;Interface name>:<number of VFs>**

NovaPCIPassthrough	nova에서 허용된 PCI 장치의 화이트리스트를 목록 형식으로 설정합니다. 예를 들면 다음과 같습니다.
	`NovaPCIPassthrough:`
	`- vendor_id: "8086"`
	`product_id: "154c"`
	`address: "0000:05:00.0"`
	`physical_network: "datacentre"`
	또한 특정 하드웨어 속성 대신 논리적 장치 이름을 사용할 수도 있습니다.
	`NovaPCIPassthrough:`
	`- devname: "ens20f2"`
	`physical_network: "datacentre"`

3.4.3. SR-IOV를 사용하여 OpenDaylight 설치

시작하기 전에

- 언더클라우드를 설치합니다. 자세한 내용은 언더클라우드 설치를 참조하십시오.
- 오버클라우드 컨테이너 이미지에 대한 링크로 환경 파일을 생성합니다. 자세한 내용은 OpenDaylight를 사용하여 오버클라우드 설치 준비 를 참조하십시오.
- SR-IOV 지원을 사용하여 사용자 지정 역할에 OpenDaylight를 구성하도록 역할 파일을 준비합니다. 자세한 내용은 SR-IOV 컴퓨팅 역할 준비를 참조하십시오.

절차

1. 사용자 정의 역할 파일 및 필요한 환경 파일을 포함하도록 **-r** 인수와 함께 배포 명령을 실행하여 OpenDaylight로 SR-IOV 기능을 활성화합니다.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-sriov.yaml
```

```
-e network-environment.yaml --compute-scale 1 --ntp-server 0.se.pool.ntp.org --control-flavor control --compute-flavor compute -r my_roles_data.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

참고

배포 명령에 있는 환경 파일은 명령에 앞서 포함하는 환경 파일을 덮어씁니다. 실수로 매개변수를 덮어쓰지 않도록 포함하는 환경 파일의 순서에 주의하십시오.

작은 정보

변경하려는 매개변수만 설정하고 기본 환경 파일과 결합하는 최소 환경 파일을 생성하여 일부 매개변수를 재정의할 수 있습니다.

더 많은 정보

- **-r** 옵션은 설치 시 역할 정의를 재정의합니다.

  ```
  -r <roles_data>.yaml
  ```

- 사용자 정의 역할에는 설치 중에 추가 ironic 노드가 필요합니다.

3.5. OVS-DPDK 지원을 사용하여 OPENDAYLIGHT 설치

OpenDaylight는 director와 함께 *Open vSwitch DPDK(Data Plane Development Kit)* 가속을 통해 배포할 수 있습니다. 이 배포는 패킷이 커널 대신 사용자 공간에서 처리되므로 더 높은 데이터 플레인 성능을 제공합니다. OVS-DPDK를 사용하여 배포하려면 잠재적인 성능 이점을 활용하기 위해 각 컴퓨팅 노드의 하드웨어 물리적 레이아웃에 대한 지식이 필요합니다.

특히 다음을 고려해야 합니다.

- 호스트의 네트워크 인터페이스에서 DPDK를 지원하는 경우
- 컴퓨팅 노드의 NUMA 노드 토폴로지(소켓당 소켓 수, CPU 코어 및 메모리 수)
- DPDK NIC PCI 버스가 각 NUMA 노드에 근접
- 컴퓨팅 노드에서 사용 가능한 RAM 양
- 네트워크 기능 가상화 계획 및 구성 가이드 참조.

3.5.1. OVS-DPDK 배포 파일 준비

OVS-DPDK를 배포하려면 다른 환경 파일을 사용합니다. 이 파일은 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker** 디렉터리의 **neutron-opendaylight.yaml** 환경 파일에 설정된 일부 매개 변수를 재정의합니다. 원래 환경 파일을 수정하지 마십시오. 대신 필요한 매개변수(예: **neutron-opendaylight-dpdk.yaml**)가 포함된 새 환경 파일을 생성합니다.

기본 설정을 사용하여 OVS-DPDK를 사용하여 OpenDaylight를 배포하려면 **/usr/share/openstack-tripleo-heat-templates/environments/services-docker** 디렉터리의 기본 **neutron-opendaylight-dpdk.yaml** 환경 파일을 사용합니다.

기본 파일에는 다음 값이 포함되어 있습니다.

```
# A Heat environment that can be used to deploy OpenDaylight with L3 DVR and DPDK.
# This file is to be used with neutron-opendaylight.yaml

parameter_defaults:
  NovaSchedulerDefaultFilters:
"RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,NUMA
TopologyFilter"
  OpenDaylightSNATMechanism: 'controller'

  ComputeOvsDpdkParameters:
    OvsEnableDpdk: True

    ## Host configuration Parameters
    #TunedProfileName: "cpu-partitioning"
    #IsolCpusList: ""               # Logical CPUs list to be isolated from the host process (applied via cpu-
partitioning tuned).
                                    # It is mandatory to provide isolated cpus for tuned to achive optimal
performance.
                                    # Example: "3-8,12-15,18"
    #KernelArgs: ""                 # Space separated kernel args to configure hugepage and IOMMU.
                                    # Deploying DPDK requires enabling hugepages for the overcloud compute
nodes.
                                    # It also requires enabling IOMMU when using the VFIO (vfio-pci)
OvsDpdkDriverType.
                                    # This should be done by configuring parameters via host-config-and-
reboot.yaml environment file.

    ## Attempting to deploy DPDK without appropriate values for the below parameters may lead to
unstable deployments
    ## due to CPU contention of DPDK PMD threads.
    ## It is highly recommended to to enable isolcpus (via KernelArgs) on compute overcloud nodes
and set the following parameters:
    #OvsDpdkSocketMemory: ""       # Sets the amount of hugepage memory to assign per NUMA
node.
                                  # It is recommended to use the socket closest to the PCIe slot used for the
                                  # desired DPDK NIC.  Format should be comma separated per socket string
such as:
                                  # "<socket 0 mem MB>,<socket 1 mem MB>", for example: "1024,0".
    #OvsDpdkDriverType: "vfio-pci" # Ensure the Overcloud NIC to be used for DPDK supports this
UIO/PMD driver.
    #OvsPmdCoreList: ""           # List or range of CPU cores for PMD threads to be pinned to.  Note,
NIC
                                  # location to cores on socket, number of hyper-threaded logical cores, and
                                  # desired number of PMD threads can all play a role in configuring this setting.
                                  # These cores should be on the same socket where OvsDpdkSocketMemory is
assigned.
                                  # If using hyperthreading then specify both logical cores that would equal the
                                  # physical core.  Also, specifying more than one core will trigger multiple PMD
                                  # threads to be spawned, which may improve dataplane performance.
    #NovaVcpuPinSet: ""           # Cores to pin Nova instances to.  For maximum performance, select
cores
                                  # on the same NUMA node(s) selected for previous settings.
```

3.5.2. OVS-DPDK 배포 구성

neutron-opendaylight-dpdk.yaml 의 값을 변경하여 OVS-DPDK 서비스를 구성할 수 있습니다.

TunedProfileName	OVS-DPDK와 함께 사용할 CPU 코어에서 분리하기 위해 IRQ를 고정할 수 있습니다. 기본 프로필: **cpu-partitioning**
IsolCpusList	커널 스케줄러에서 OVS-DPDK 전용으로 할당할 수 있는 이러한 코어를 사용하지 못하도록 CPU 코어 목록을 지정합니다. 형식은 쉼표로 구분된 개인 또는 코어 범위(예: **1,2,3,4-8,10-12)를 사용합니다.**
KernelArgs	부팅 시 커널로 전달할 인수를 나열합니다. OVS-DPDK의 경우 IOMMU 및 Hugepages를 활성화해야 합니다 **.** 예 **를** 들면 다음과 같습니다. ---- **intel_iommu=on iommu=pt default_hugepagesz=1GB hugepagesz=1G hugepages=60** ---- 지정된 RAM 용량은 hugepages의 경우 60GB입니다. 이 값을 설정할 때 컴퓨팅 노드에서 사용 가능한 RAM 용량을 고려하는 것이 중요합니다.
OvsDpdkSocketMemory	각 NUMA 노드에 할당할 대규모 페이지 메모리(MB)의 양을 지정합니다. 최대 성능을 위해 DPDK NIC에 가장 가까운 소켓에 메모리를 할당합니다. 소켓당 메모리 형식 나열: ---- "<socket 0 mem MB>,<socket 1 mem MB>" ---- 예: "1024,0"
OvsDpdkDriverType	PMD 스레드에서 사용할 UIO 드라이버 유형을 지정합니다. DPDK NIC는 지정된 드라이버를 지원해야 합니다. Red Hat OpenStack Platform 배포는 드라이버 유형 **vfio-pci** 를 지원합니다. Red Hat OpenStack Platform 배포는 **uio_pci_generic** 및 **igb_uio** 를 포함한 UIO 드라이버를 지원하지 않습니다.
OvsPmdCoreList	PMD 스레드가 고정될 단일 코어 또는 코어 범위를 나열합니다. 여기에 지정된 코어는 **OvsDpdkSocketMemory** 설정을 사용하여 메모리가 할당된 동일한 NUMA 노드에 있어야 합니다. 하이퍼 스레딩을 사용하는 경우 호스트의 물리적 코어를 구성하는 논리 코어를 지정합니다.
OvsDpdkMemoryChannels	소켓당 메모리 채널 수를 지정합니다.
NovaVcpuPinSet	**libvirtd** 를 사용하여 nova 인스턴스를 에 고정하는 코어입니다. 최상의 성능을 위해 OVS PMD Core가 에 고정된 동일한 소켓의 코어를 사용합니다.

3.5.3. OVS-DPDK를 사용하여 OpenDaylight 설치

시작하기 전에

- 언더클라우드를 설치합니다. 자세한 내용은 언더클라우드 설치를 참조하십시오.
- 오버클라우드 컨테이너 이미지에 대한 링크로 환경 파일을 생성합니다. 자세한 내용은 OpenDaylight를 사용하여 오버클라우드 설치 준비 를 참조하십시오.

- SR-IOV 지원을 사용하여 사용자 지정 역할에 OpenDaylight를 구성하도록 역할 파일을 준비합니다. 자세한 내용은 OVS-DPDK 배포 파일 준비를 참조하십시오.

절차

1. 필요한 환경 파일과 함께 배포 명령을 실행하여 OpenDaylight에서 DPDK 기능을 활성화합니다.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-dpdk.yaml
-e network-environment.yaml --compute-scale 1 --ntp-server 0.se.pool.ntp.org --control-flavor control
--compute-flavor compute -r my_roles_data.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

참고

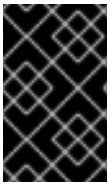
배포 명령에 있는 환경 파일은 명령에 앞서 포함하는 환경 파일을 덮어씁니다. 실수로 매개변수를 덮어쓰지 않도록 포함하는 환경 파일의 순서에 주의하십시오.

작은 정보

변경하려는 매개변수만 설정하고 기본 환경 파일과 결합하는 최소 환경 파일을 생성하여 일부 매개변수를 재정의할 수 있습니다.

3.5.4. 예: ODL 및 VXLAN 터널을 사용하여 OVS-DPDK 구성

이 섹션에서는 ODL 및 VXLAN 터널을 사용한 OVS-DPDK의 구성에 대해 설명합니다.



중요

OVS-DPDK의 OpenStack 네트워크를 최적화하려면 **network-environment.yaml** 파일에서 설정한 OVS-DPDK 매개변수에 가장 적합한 값을 결정해야 합니다. 자세한 내용은 워크플로를 사용하여 DPDK 매개변수 분리를 참조하십시오.

3.5.4.1. ComputeOvsDpdk 구성 가능 역할 생성

ComputeOvsDpdk 역할에 대한 **roles_data.yaml** 을 생성합니다.

```
# openstack overcloud roles generate --roles-path templates/openstack-tripleo-heat-templates/roles -o roles_data.yaml Controller ComputeOvsDpdk
```

3.5.4.2. OVS-DPDK 매개변수 구성

중요

OVS-DPDK의 OpenStack 네트워크를 최적화하려면 **network-environment.yaml** 파일에서 설정한 OVS-DPDK 매개변수에 가장 적합한 값을 결정해야 합니다. 자세한 내용은 https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_planning_and_configure#proc_derive-dpdk 을 참조하십시오.

1. **resource_registry** 에서 OVS-DPDK에 대한 사용자 정의 리소스를 추가합니다.

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to use for override the default.
  OS::TripleO::ComputeOvsDpdk::Net::SoftwareConfig: nic-configs/computeovsdpdk.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml
```

2. **parameter_defaults** 아래에서 터널 유형과 테넌트 유형을 **vxlan** 으로 설정합니다.

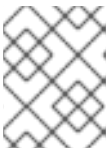
```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan'
```

3. **parameters_defaults** 에서 브리지 매핑을 설정합니다.

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings: 'tenant:br-link0'
OpenDaylightProviderMappings: 'tenant:br-link0'
```

4. **parameter_defaults** 에서 **ComputeOvsDpdk** 역할에 대한 역할별 매개변수를 설정합니다.

```
  ############################
  # OVS DPDK configuration #
  ############################
  ComputeOvsDpdkParameters:
    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"
    NovaVcpuPinSet: ['4-19,24-39']
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "4096,4096"
    OvsDpdkMemoryChannels: "4"
    OvsDpdkCoreList: "0,20,1,21"
    OvsPmdCoreList: "2,22,3,23"
    OvsEnableDpdk: true
```



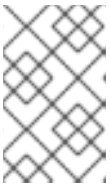
참고

게스트 인스턴스 생성에 실패를 방지하려면 DPDK PMD에 대해 DPDK NIC를 포함하거나 포함하지 않은 각 NUMA 노드에 하나 이상의 CPU를 할당해야 합니다.

참고

이러한 대규모 페이지는 가상 머신에서 사용하며 이 프로세스에 표시된 대로 **OvsDpdkSocketMemory** 매개변수를 사용하는 OVS-DPDK에서도 사용합니다. 가상 머신에서 사용할 수 있는 대규모 페이지 수는 **부팅** 매개변수입니다. **OvsDpdkSocketMemory**.

DPDK 인스턴스와 연결된 플레이버에 **hw:mem_page_size=1GB** 도 추가해야 합니다.



참고

OvsDPDKCoreList 및 **OvsDpdkMemoryChannels** 는 이 절차에 **필요한** 설정입니다. 적절한 값 없이 DPDK를 배포하려고 하면 배포에 실패하거나 불안정한 배포가 발생합니다.

3.5.4.3. 컨트롤러 노드 구성

1. 격리된 네트워크의 컨트롤 플레인 Linux 본딩을 생성합니다.

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  addresses:
  - ip_netmask:
      list_join:
      - /
      - - get_param: ControlPlaneIp
        - get_param: ControlPlaneSubnetCidr
  routes:
  - ip_netmask: 169.254.169.254/32
    next_hop:
      get_param: EC2MetadataIp
  members:
  - type: interface
    name: eth1
    primary: true
```

2. VLAN을 이 Linux 본딩에 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. 클라우드 네트워크에 유동 IP에 액세스할 수 있도록 OVS 브리지를 만듭니다.

```
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: eth2
    mtu: 9000
  - type: vlan
    vlan_id:
      get_param: TenantNetworkVlanID
    mtu: 9000
    addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
```

3.5.4.4. DPDK 인터페이스에 대한 컴퓨팅 노드 구성

기본 **compute.yaml** 파일에서 **compute-ovs-dpdk.yaml** 파일을 생성하고 다음과 같이 변경합니다.

1. 격리된 네트워크의 컨트롤 플레인 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: nic7
      primary: true
    - type: interface
      name: nic8
```

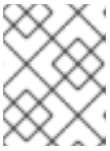
2. VLAN을 이 Linux 본딩에 할당합니다.

```
- 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: nic3
        - type: ovs_dpdk_port
          name: dpdk1
          members:
            - type: interface
              name: nic4
```



참고

여러 DPDK 장치를 포함하려면 추가할 각 DPDK 장치에 대한 **유형** 코드 섹션을 반복합니다.

참고

OVS-DPDK를 사용하는 경우 동일한 컴퓨팅 노드의 **모든** 브릿지는 **ovs_user_bridge** 여야 합니다. director는 설정을 허용하지만 Red Hat OpenStack Platform은 동일한 노드에서 **ovs_bridge** 및 **ovs_user_bridge** 혼합을 지원하지 않습니다.

3.5.4.5. 오버클라우드 배포

overcloud_deploy.sh 스크립트를 실행하여 오버클라우드를 배포합니다.

```
#!/bin/bash

openstack overcloud deploy \
--templates \
-r /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight-dpdk.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ovs-dpdk-permissions.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/docker-images.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-odl-ctlplane-dataplane-bonding-hybrid/network-environment.yaml
```

3.6. L2GW 지원을 사용하여 OPENDAYLIGHT 설치

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.

계층 2 게이트웨이 서비스를 사용하면 테넌트의 가상 네트워크를 물리적 네트워크에 연결할 수 있습니다. 이러한 통합을 통해 사용자는 라우팅된 계층 3 연결을 통하지 않고 계층 2 네트워크 연결을 통해 물리적 서버의 리소스에 액세스할 수 있습니다. 즉, L3 또는 유동 IP를 거치지 않고 계층 2 브로드캐스트 도메인을 확장합니다.

3.6.1. L2GW 배포 파일 준비

L2GW 지원을 사용하여 OpenDaylight를 배포하려면 **/usr/share/openstack-tripleo-heat-templates/environments** 디렉터리에 있는 **neutron-l2gw-opendaylight.yaml** 파일을 사용합니다. 해당 파일에서 설정을 변경해야 하는 경우 기존 파일을 수정하지 마십시오. 대신 필요한 매개 변수가 포함된 환경 파일의 새 복사본을 생성합니다.

기본 설정으로 OpenDaylight 및 L2GW를 배포하려면 **/usr/share/openstack-tripleo-heat-templates/environment-docker 디렉터리에서 neutron-l2gw- opendaylight.yaml** 을 사용할 수 있습니다.

기본 파일에는 다음 값이 포함되어 있습니다.

```
# A Heat environment file that can be used to deploy Neutron L2 Gateway service
#
# Currently there are only two service provider for Neutron L2 Gateway
# This file enables L2GW service with OpenDaylight as driver.
#
# - OpenDaylight:
L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default
resource_registry:
  OS::TripleO::Services::NeutronL2gwApi: ../../docker/services/neutron-l2gw-api.yaml

parameter_defaults:
  NeutronServicePlugins: "odl-router_v2,trunk,l2gw"
  L2gwServiceProvider:
['L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default']

  # Optional
  # L2gwServiceDefaultInterfaceName: "FortyGigE1/0/1"
  # L2gwServiceDefaultDeviceName: "Switch1"
  # L2gwServiceQuotaL2Gateway: 10
  # L2gwServicePeriodicMonitoringInterval: 5
```

3.6.2. OpenDaylight L2GW 배포 구성

neutron-l2gw-opendaylight.yaml 파일에서 값을 변경하여 서비스를 구성할 수 있습니다.

NeutronServicePlugins	**neutron.service_plugins** 네임스페이스에서 로드할 서비스 플러그인 진입점의 쉼표로 구분된 목록입니다. 기본값은 **router** 입니다.
L2gwServiceProvider	이 서비스를 제공하는 데 사용해야 하는 공급자를 정의합니다. 기본값은 **L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default**
L2gwServiceDefaultInterfaceName	기본 인터페이스의 이름을 설정합니다.
L2gwServiceDefaultDeviceName	기본 장치의 이름을 설정합니다.
L2gwServiceQuotaL2Gateway	L2 게이트웨이의 서비스 할당량을 지정합니다. 기본값은 **10** 입니다.
L2gwServicePeriodicMonitoringInterval	L2GW 서비스의 모니터링 간격을 지정합니다.

3.6.3. L2GW로 OpenDaylight 설치

시작하기 전에

- 언더클라우드를 설치합니다. 자세한 내용은 언더클라우드 설치를 참조하십시오.
- 오버클라우드 컨테이너 이미지에 대한 링크로 환경 파일을 생성합니다. 자세한 내용은 OpenDaylight를 사용하여 오버클라우드 설치 준비 를 참조하십시오.

- SR-IOV 지원을 사용하여 사용자 지정 역할에 OpenDaylight를 구성하도록 역할 파일을 준비합니다. 자세한 내용은 L2GW 배포 파일 준비를 참조하십시오.

절차

1. 필요한 환경 파일과 함께 배포 명령을 실행하여 OpenDaylight로 L2GW 기능을 활성화합니다.

```
$ openstack overcloud deploy --templates /usr/share/openstack-tripleo-heat-templates
-e <other environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-opendaylight.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-l2gw-opendaylight.yaml
-e /home/stack/templates/docker-images.yaml
-e /home/stack/templates/odl-images.yaml
```

참고

배포 명령에 있는 환경 파일은 명령에 앞서 포함하는 환경 파일을 덮어씁니다. 실수로 매개변수를 덮어쓰지 않도록 포함하는 환경 파일의 순서에 주의하십시오.

작은 정보

변경하려는 매개변수만 설정하고 기본 환경 파일과 결합하는 최소 환경 파일을 생성하여 일부 매개변수를 재정의할 수 있습니다.

4장. 배포 테스트

4.1. 기본 테스트 수행

기본 테스트에서는 인스턴스가 서로 ping할 수 있는지 확인합니다. 테스트에서는 유동 *IP SSH* 액세스도 확인합니다. 이 예제에서는 언더클라우드에서 이 테스트를 수행하는 방법을 설명합니다.

이 절차를 수행하려면 많은 개별 단계를 수행해야 합니다. 편의를 위해 이 절차는 더 작은 부분으로 나뉩니다. 그러나 다음 순서로 모든 단계를 수행해야 합니다.

참고

In this setup, a flat network is used to create the _External_ network, and _VXLAN_ is used for the _Tenant_ networks. _VLAN External_ networks and _VLAN Tenant_ networks are also supported, depending on the desired deployment.

4.1.1. 테스트를 위한 새 네트워크 만들기

1. 오버클라우드에 액세스하기 위해 인증 정보를 가져옵니다.

```
$ source /home/stack/overcloudrc
```

2. 외부 neutron 네트워크를 생성하여 오버클라우드 외부에서 인스턴스에 액세스합니다.

```
$ openstack network create --external --project service --external  --provider-network-type flat --provider-physical-network datacentre external
```

3. 이전 단계에서 생성한 새 외부 네트워크에 해당하는 neutron 서브넷을 생성합니다.

```
$ openstack subnet create  --project service --no-dhcp --network external --gateway 192.168.37.1 --allocation-pool start=192.168.37.200,end=192.168.37.220 --subnet-range 192.168.37.0/24 external-subnet
```

4. 오버클라우드 인스턴스를 생성하는 데 사용할 cirros 이미지를 다운로드합니다.

```
$ wget http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img
```

5. 오버클라우드의 glance에 cirros 이미지를 업로드합니다.

```
$ openstack image create cirros --public --file ./cirros-0.3.4-x86_64-disk.img --disk-format qcow2 --container-format bare
```

6. 오버클라우드 인스턴스에 사용할 **작** 은 플레이버를 만듭니다.

```
$ openstack flavor create m1.tiny --ram 512 --disk 1 --public
```

7. 인스턴스를 호스팅할 VXLAN 테넌트 네트워크를 만듭니다.

```
$ openstack network create net_test --provider-network-type=vxlan --provider-segment 100
```

8. 이전 단계에서 만든 테넌트 네트워크의 서브넷을 생성합니다.

```
$ openstack subnet create --network net_test --subnet-range 123.123.123.0/24 test
```

9. 테넌트 네트워크의 ID를 찾아서 저장합니다.

```
$ net_mgmt_id=$(openstack network list | grep net_test | awk '{print $2}')
```

10. **cirros1** 인스턴스를 만들고 **net_test** 네트워크 및 **SSH** 보안 그룹에 연결합니다.

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-id=$vlan1 --security-group
SSH --key-name RDO_KEY --availability-zone nova:overcloud-novacompute-0.localdomain
cirros1
```

11. **net_test** 네트워크 및 **SSH** 보안 그룹에도 연결된 두 번째 인스턴스 **cirros2** 를 만듭니다.

```
$ openstack server create --flavor m1.tiny --image cirros --nic net-id=$vlan1 --security-group
SSH --key-name RDO_KEY --availability-zone nova:overcloud-novacompute-0.localdomain
cirros2
```

4.1.2. 테스트 환경에서 네트워킹 설정

1. admin 프로젝트의 ID를 찾아서 저장합니다.

```
$ admin_project_id=$(openstack project list | grep admin | awk '{print $2}')
```

2. admin 프로젝트의 기본 보안 그룹을 찾아 저장합니다.

```
$ admin_sec_group_id=$(openstack security group list | grep $admin_project_id | awk '{print
$2}')
```

3. ICMP 트래픽 수신을 허용하도록 admin default 보안 그룹에 규칙을 추가합니다.

```
$ openstack security group rule create $admin_sec_group_id --protocol icmp --ingress
```

4. ICMP 트래픽 송신을 허용하도록 admin default 보안 그룹에 규칙을 추가합니다.

```
$ openstack security group rule create $admin_sec_group_id --protocol icmp --egress
```

5. admin default 보안 그룹에 규칙을 추가하여 SSH 트래픽 수신을 허용합니다.

```
$ openstack security group rule create $admin_sec_group_id --protocol tcp --dst-port 22 --
ingress
```

6. admin default 보안 그룹에 규칙을 추가하여 SSH 트래픽 송신을 허용합니다.

```
$ openstack security group rule create $admin_sec_group_id --protocol tcp --dst-port 22 --
egress
```

4.1.3. 연결 테스트

1. horizon에서 인스턴스의 *novnc* 콘솔에 액세스할 수 있어야 합니다. ***overcloudrc*** 의 암호를 사용하여 *admin* 으로 horizon에 로그인합니다. *cirros* 이미지의 기본 인증 정보는 사용자 이름 **cirros** 및 암호 **cubswin:)** 입니다.

2. *novnc* 콘솔에서 인스턴스에서 DHCP 주소를 수신하는지 확인합니다.

```
$ ip addr show
```

참고

언더클라우드에서 **nova console-log <instance id** > 명령을 실행하여 인스턴스에서 DHCP 주소를 수신하는지 확인할 수도 있습니다.

3. 다른 모든 인스턴스에 대해 1 및 2단계를 반복합니다.

4. 한 인스턴스에서 다른 인스턴스를 ping합니다. 이렇게 하면 오버클라우드의 *기본 테넌트* 네트워크 연결이 검증됩니다.

5. 유동 *IP* 를 사용하여 다른 인스턴스에 연결할 수 있는지 확인합니다.

4.1.4. 장치 생성

1. **cirros1** 인스턴스와 연결할 외부 네트워크에 유동 IP를 만듭니다.

```
$ openstack floating ip create external
```

2. 유동 IP와 **cirros1** 테넌트 IP 간에 NAT를 처리하는 라우터를 만듭니다.

```
$ openstack router create test
```

3. 라우터의 게이트웨이를 외부 네트워크로 설정합니다.

```
$ openstack router set test --external-gateway external
```

4. 테넌트 네트워크에 연결된 라우터에 인터페이스를 추가합니다.

```
$ openstack router add subnet test test
```

5. 1단계에서 만든 유동 IP를 찾아 저장합니다.

```
$ floating_ip=$(openstack floating ip list | head -n -1 | grep -Eo '[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+')
```

6. 유동 IP를 **cirros1** 인스턴스와 연결합니다.

```
$ openstack server add floating ip cirros1 $floating_ip
```

7. 외부 네트워크 액세스 권한이 있는 노드에서 인스턴스에 로그인합니다.

```
$ ssh cirros@$floating_ip
```

4.2. 고급 테스트 수행

OpenDaylight를 배포한 후 OpenDaylight 구성 및 배포의 여러 구성 요소를 테스트할 수 있습니다. 설치의 특정 부분을 테스트하려면 여러 절차를 따라야 합니다. 각 절차는 개별적으로 설명됩니다.

오버클라우드 노드에서 절차를 수행해야 합니다.

4.2.1. 오버클라우드 노드에 연결

오버클라우드 노드에 연결하고 올바르게 작동하는지 확인하려면 다음 단계를 완료합니다.

절차

1. 언더클라우드에 로그인합니다.
2. 다음 명령을 입력하여 프로세스를 시작합니다.

```
$ source /home/stack/stackrc
```

3. 모든 인스턴스를 나열합니다.

```
$ openstack server list
```

4. 필요한 인스턴스를 선택하고 목록에서 인스턴스 IP 주소를 기록합니다.
5. 이전 단계에서 얻은 목록에서 IP 주소를 사용하여 머신에 연결합니다.

```
$ ssh heat-admin@<IP from step 4>
```

6. superuser로 전환합니다.

```
$ sudo -i
```

4.2.2. OpenDaylight 테스트

OpenDaylight가 올바르게 작동하는지 테스트하려면 서비스가 작동하고 지정된 기능이 올바르게 로드되었는지 확인해야 합니다.

절차

1. 수퍼유저로 OpenDaylight를 실행하는 오버클라우드 노드에 로그인하거나 사용자 지정 역할로 실행 중인 OpenDaylight 노드에 로그인합니다.
2. OpenDaylight 컨트롤러가 모든 컨트롤러 노드에서 실행 중인지 확인합니다.

```
# docker ps | grep opendaylight
2363a99d514a        192.168.24.1:8787/rhosp13/openstack-opendaylight:latest
"kolla_start"           4 hours ago         Up 4 hours (healthy)                        opendaylight_api
```

3. HAProxy가 포트 8081에서 수신 대기하도록 올바르게 구성되었는지 확인합니다.

```
# docker exec -it haproxy-bundle-docker-0 grep -A7 opendaylight /etc/haproxy/haproxy.cfg
listen opendaylight
  bind 172.17.0.10:8081 transparent
  bind 192.168.24.10:8081 transparent
```

```
  mode http
  balance source
  server overcloud-controller-0.internalapi.localdomain 172.17.0.22:8081 check fall 5 inter
2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.17.0.12:8081 check fall 5 inter
2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.17.0.13:8081 check fall 5 inter
2000 rise 2
```

4. HAproxy IP를 사용하여 karaf 계정을 연결합니다. karaf 암호는 **karaf**:

```
# ssh -p 8101 karaf@localhost
```

5. 설치된 기능을 나열합니다.

```
# feature:list -i | grep odl-netvirt-openstack
```

절차 중에 생성된 것처럼 목록의 세 번째 열에 **x** 가 있는 경우 기능이 올바르게 설치됩니다.

6. API가 작동하는지 확인합니다.

```
  # web:list | grep neutron
```

이 API 엔드포인트는 **/etc/neutron/plugins/ml2/ml2_conf.ini** 에 설정되어 있으며 neutron이 OpenDaylight와 통신하는 데 사용됩니다.

7. 노드 간 *VXLAN* 터널이 작동 중인지 확인합니다.

```
# vxlan:show
```

8. REST API가 올바르게 응답하는지 테스트하려면 해당 API를 사용하는 모듈을 나열합니다.

```
# curl -u "admin:admin" http://localhost:8081/restconf/modules
```

출력은 유사할 것입니다(예가 단축됨).

```
{"modules":{"module":[{"name":"netty-event-executor","revision":"2013-11-
12","namespace":"urn:opendaylight:params:xml:ns:yang:controller:netty:eventexecutor"},
{"name" ...
```

9. 호스트 internal_API IP를 사용하는 REST 스트림을 나열합니다.

```
# curl -u "admin:admin" http://localhost:8081/restconf/streams
```

다음과 같은 출력이 표시됩니다.

```
{"streams":{}}
```

10. 다음 명령을 내부_API IP와 함께 실행하여 NetVirt가 작동하는지 확인합니다.

```
# curl -u "admin:admin" http://localhost:8081/restconf/operational/network-topology:network-
topology/topology/netvirt:1
```

다음 출력에서 NetVirt가 작동하는지 확인합니다.

```
{"topology":[{"topology-id":"netvirt:1"}]}
```

4.2.3. Open vSwitch 테스트

Open vSwitch 를 확인하려면 컴퓨팅 노드 중 하나에 연결하고 올바르게 구성되어 OpenDaylight에 연결되어 있는지 확인합니다.

절차

1. 수퍼유저로 오버클라우드의 컴퓨팅 노드 중 하나에 연결합니다.
2. Open vSwitch 설정을 나열합니다.

```
# ovs-vsctl show
```

3. 출력의 여러 관리자를 확인합니다. 이 예제에서는 2행과 3행이 여러 관리자를 표시합니다.

```
    6b003705-48fc-4534-855f-344327d36f2a
        Manager "ptcp:6639:127.0.0.1"
        Manager "tcp:172.17.1.16:6640"
            is_connected: true
        Bridge br-ex
            fail_mode: standalone
            Port br-ex-int-patch
                Interface br-ex-int-patch
                    type: patch
                    options: {peer=br-ex-patch}
            Port br-ex
                Interface br-ex
                    type: internal
            Port "eth2"
                Interface "eth2"
        Bridge br-isolated
            fail_mode: standalone
            Port "eth1"
                Interface "eth1"
            Port "vlan50"
                tag: 50
                Interface "vlan50"
                    type: internal
            Port "vlan30"
                tag: 30
                Interface "vlan30"
                    type: internal
            Port br-isolated
                Interface br-isolated
                    type: internal
            Port "vlan20"
                tag: 20
                Interface "vlan20"
                    type: internal
        Bridge br-int
```

```
        Controller "tcp:172.17.1.16:6653"
            is_connected: true
        fail_mode: secure
        Port br-ex-patch
            Interface br-ex-patch
                type: patch
                options: {peer=br-ex-int-patch}
        Port "tun02d236d8248"
            Interface "tun02d236d8248"
                type: vxlan
                options: {key=flow, local_ip="172.17.2.18", remote_ip="172.17.2.20"}
        Port br-int
            Interface br-int
                type: internal
        Port "tap1712898f-15"
            Interface "tap1712898f-15"
    ovs_version: "2.7.0"
```

4. **tcp** 관리자가 OpenDaylight가 실행 중인 노드의 IP를 가리키는지 확인합니다.

5. OVS의 OpenDaylight에 연결하고 OVSDB 프로토콜을 사용하는지 확인하려면 Managers show **is_connected: true** 를 확인합니다.

6. 각 브리지(*br-int*이외의)가 있으며 Compute 역할과 함께 배포에 사용되는 NIC 템플릿에 해당하는지 확인합니다.

7. *tcp* 연결이 OpenDaylight 서비스가 실행 중인 IP에 해당하는지 확인합니다.

8. *br-int* 에 **is_connected: true** 브리지가 표시되고 OpenDaylight에 대한 OpenFlow 프로토콜 연결이 설정되었는지 확인합니다.

더 많은 정보

- OpenDaylight는 *br-int* 브리지를 자동으로 생성합니다.

4.2.4. 컴퓨팅 노드에서 Open vSwitch 구성을 확인합니다.

1. 수퍼유저로 컴퓨팅 노드에 연결합니다.

2. *Open vSwitch* 구성 설정을 나열합니다.

```
# ovs-vsctl list open_vswitch
```

3. 출력을 읽습니다. 이 예와 유사합니다.

```
 _uuid               : 4b624d8f-a7af-4f0f-b56a-b8cfabf7635d
 bridges             : [11127421-3bcc-4f9a-9040-ff8b88486508, 350135a4-4627-4e1b-8bef-
56a1e4249bef]
 cur_cfg             : 7
 datapath_types      : [netdev, system]
 db_version          : "7.12.1"
 external_ids        : {system-id="b8d16d0b-a40a-47c8-a767-e118fe22759e"}
 iface_types         : [geneve, gre, internal, ipsec_gre, lisp, patch, stt, system, tap, vxlan]
 manager_options     : [c66f2e87-4724-448a-b9df-837d56b9f4a9, defec179-720e-458e-8875-
ea763a0d8909]
```

```
next_cfg            : 7
other_config        : {local_ip="11.0.0.30", provider_mappings="datacentre:br-ex"}
ovs_version         : "2.7.0"
ssl                 : []
statistics          : {}
system_type         : RedHatEnterpriseServer
system_version      : "7.4-Maipo"
```

4. **other_config** 옵션의 값에 *VXLAN* 터널을 통해 테넌트 네트워크에 연결되는 로컬 인터페이스에 대해 올바른 **local_ip** 세트가 있는지 확인합니다.

5. **other_config** 옵션 아래의 **provider_mappings** 값이 **OpenDaylightProviderMappings** heat 템플릿 매개변수의 값과 일치하는지 확인합니다. 이 구성은 neutron 논리적 네트워크를 해당 물리 인터페이스에 매핑합니다.

4.2.5. neutron 구성 확인

절차

1. 컨트롤러 역할 노드 중 하나에서 수퍼유저 계정에 연결합니다.

2. **/etc/neutron/neutron.conf** 파일에 **service_plugins=odl-router_v2,trunk** 가 포함되어 있는지 확인합니다.

3. **/etc/neutron/plugin.ini** 파일에 다음 **ml2** 구성이 포함되어 있는지 확인합니다.

```
[ml2]
mechanism_drivers=opendaylight_v2

[ml2_odl]
password=admin
username=admin
url=http://192.0.2.9:8081/controller/nb/v2/neutron
```

4. **Overcloud** 컨트롤러 중 하나에서 neutron 에이전트가 올바르게 실행되고 있는지 확인합니다.

```
# openstack network agent list
```

5. 메타데이터 및 DHCP 에이전트의 **admin_state_up** 값이 **True** 인지 확인합니다.

```
+--------------------------------------+----------------+-------------------------+-------------------+------
-+----------------+-----------------------+
| id                                   | agent_type     | host                    | availability_zone | alive |
admin_state_up | binary                |
+--------------------------------------+----------------+-------------------------+-------------------+------
-+----------------+-----------------------+
| 3be198c5-b3aa-4d0e-abb4-51b29db3af47 | Metadata agent | controller-0.localdomain |
| :-)   | True           | neutron-metadata-agent |
| 79579d47-dd7d-4ef3-9614-cd2f736043f3 | DHCP agent     | controller-0.localdomain | nova
| :-)   | True           | neutron-dhcp-agent     |
+--------------------------------------+----------------+-------------------------+-------------------+------
-+----------------+-----------------------+
```

더 많은 정보

- 3단계에서 언급한 **plugin.ini** 의 IP는 **InternalAPI** 가상 IP 주소(VIP)여야 합니다.
- 이제 OpenDaylight에서 모두 관리되므로 5단계 출력에 나열된 Open vSwitch 에이전트 또는 L3 에이전트가 없습니다.

5장. 디버깅

5.1. 로그 찾기

5.1.1. OpenDaylight 로그 액세스

OpenDaylight는 **/var/log/containers/opendaylight** 디렉터리의 컨테이너에 로그를 저장합니다. 가장 최근의 로그의 이름은 **karaf.log** 입니다. OpenDaylight는 이전 로그에 증분 번호를 추가합니다(예: **karaf.log.1**,**karaf.log.** 2).

5.1.2. OpenStack Networking 로그에 액세스

OpenStack 네트워킹 명령이 실패하면 먼저 neutron 로그를 검사합니다. **/var/log/containers/neutron** 디렉터리의 각 neutron 노드의 **server.log** 파일에서 neutron 로그를 찾을 수 있습니다.

server.log 파일에는 OpenDaylight와의 통신 관련 오류도 포함되어 있습니다. neutron 오류가 OpenDaylight와 상호 작용에서 시작되는 경우 OpenDaylight 로그를 검사하여 실패 원인을 찾아야 합니다.

5.2. 네트워킹 오류 디버그

인스턴스 연결 손실과 같은 네트워크 오류가 발생하지만 OpenStack 명령 또는 neutron 로그를 실행할 때 오류가 보고되지 않으면 OVS 노드에서 네트워크 트래픽 및 OpenFlow 흐름을 검사하는 것이 유용할 수 있습니다.

1. 네트워크 오류가 발생하는 노드에 수퍼유저로 로그인합니다.

2. **br-int 스위치에 대한 정보를 표시합니다.**

```
# ovs-ofctl -O openflow13 show br-int
```

3. **출력을 검사합니다. 다음 예와 유사해야 합니다.**

```
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000e4c153bdb306
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS
QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(br-ex-patch): addr:ae:38:01:09:66:5b
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 2(tap1f0f610c-8e): addr:00:00:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 3(tun1147c81b59c): addr:66:e3:d2:b3:b8:e3
     config:     0
```

```
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:e4:c1:53:bd:b3:06
    config:     PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x5): frags=normal miss_send_len=0
```

4. **br-int** 스위치의 통계를 나열합니다.

```
# ovs-ofctl -O openflow13 dump-ports br-int
```

5. 출력을 검사합니다. 다음 예와 유사해야 합니다.

```
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
 port LOCAL: rx pkts=101215, bytes=6680190, drop=0, errs=0, frame=0, over=0, crc=0
          tx pkts=0, bytes=0, drop=0, errs=0, coll=0
          duration=90117.708s
 port  1: rx pkts=126887, bytes=8970074, drop=0, errs=0, frame=0, over=0, crc=0
          tx pkts=18764, bytes=2067792, drop=0, errs=0, coll=0
          duration=90117.418s
 port  2: rx pkts=1171, bytes=70800, drop=0, errs=0, frame=0, over=0, crc=0
          tx pkts=473, bytes=44448, drop=0, errs=0, coll=0
          duration=88644.819s
 port  3: rx pkts=120197, bytes=8776126, drop=0, errs=0, frame=0, over=0, crc=0
          tx pkts=119408, bytes=8727254, drop=0, errs=0, coll=0
          duration=88632.426s
```

더 많은 정보

- **3**단계에서 는 이 **OVS** 노드에 세 개의 포트가 있습니다. 첫 번째 포트는 이 시나리오에서 외부 네트워크 연결 포트인 **br-ex** 브리지로 이동하는 패치 포트입니다. 두 번째 포트는 **DHCP** 에이전트 인스턴스에 연결하는 탭 포트입니다. 호스트가 컨트롤러이기 때문에 이를 알고 있습니다. 그렇지 않으면 **Compute** 역할에 따라 인스턴스가 됩니다. 세 번째 포트는 테넌트 트래픽에 대해 생성된 **VXLAN** 터널 포트입니다.

- 각 포트가 무엇인지 이해하면 포트 통계를 검사하여 포트가 트래픽을 수신하고 있는지 확인할 수 있습니다(단계 **4** 단계참조).

- **5**단계 의 출력에서 각 포트가 수신**(rx pkts)** 및 패킷 전송**(tx pkts)**을 확인합니다.

5.2.1. OpenFlow 흐름을 사용한 고급 디버깅

OpenFlow에 익숙한 고급 사용자의 경우 스위치의 흐름을 검사하여 트래픽이 떨어지는 위치를 탐지할 수 있습니다.

1. 흐름을 나열하고 패킷 수를 보려면 다음 명령을 입력합니다.

```
# ovs-ofctl -O openflow13 dump-flows br-int
```

2. 명령 출력을 검사하여 필요한 정보를 가져옵니다.

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x8000000, duration=90071.665s, table=0, n_packets=126816, n_bytes=8964820,
priority=1,in_port=1
actions=write_metadata:0x20000000001/0xffffff0000000001,goto_table:17
 cookie=0x8000000, duration=88967.292s, table=0, n_packets=473, n_bytes=44448,
priority=4,in_port=2
actions=write_metadata:0x40000000000/0xffffff0000000001,goto_table:17
 cookie=0x8000001, duration=88954.901s, table=0, n_packets=120636, n_bytes=8807869,
priority=5,in_port=3
actions=write_metadata:0x70000000001/0x1fffff0000000001,goto_table:36
 cookie=0x8000001, duration=90069.534s, table=17, n_packets=126814, n_bytes=8964712,
priority=5,metadata=0x20000000000/0xffffff0000000000
actions=write_metadata:0xc0000200000222e0/0xfffffffffffff
ffe,goto_table:19
 cookie=0x8040000, duration=90069.533s, table=17, n_packets=126813, n_bytes=8964658,
priority=6,metadata=0xc000020000000000/0xffffff0000000000
actions=write_metadata:0xe00002138a000000/0xffffffff
fffffffe,goto_table:48
 cookie=0x8040000, duration=88932.689s, table=17, n_packets=396, n_bytes=36425,
priority=6,metadata=0xc000040000000000/0xffffff0000000000
actions=write_metadata:0xe00004138b000000/0xfffffffffffff
ffe,goto_table:48
```

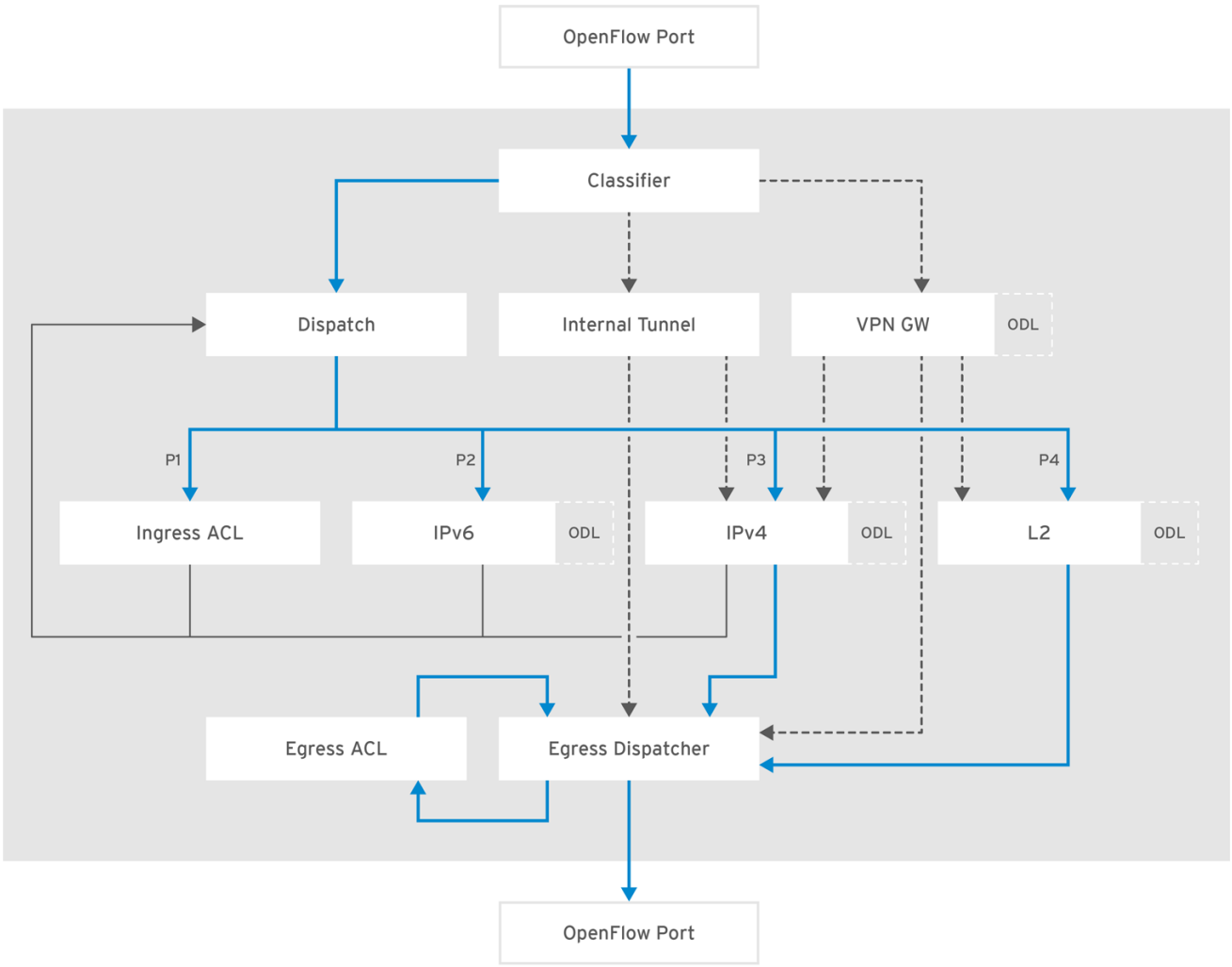
참고

이 출력은 길이에 맞게 편집되었습니다.

5.2.2. OpenFlow에서 패킷 트래버스

중요한 점은 패킷에서 수행되는 네트워크 기능이 서로 다른 **OpenFlow** 테이블로 분할되고 패킷이 **0**부터 시작하여 해당 테이블을 순서대로 통과한다는 것입니다. 들어오는 패킷은 표 **0**에 있는 다음, 포트 밖으로 전송되거나 ***OpenDaylight Controller****로 삭제될 때까지* ***OpenFlow*** *파이프라인* 을 통해 진행됩니다. 패킷이 하나 이상의 테이블을 건너뛸 수 있으며, 이 테이블은 어떤 네트워크 기능으로 이동해야 하는지에 따라 달라집니다. 테이블의 전체 다이어그램 및 네트워크 기능에 해당하는 방법은 다음과 같습니다.

그림 5.1. OpenDaylight NetVirt OpenFlow *Pipeline*



6장. 배포 예

6.1. 테넌트 네트워크를 사용하는 모델 설치 시나리오

이 섹션에서는 프로덕션 환경에서 **OpenStack**을 사용한 **OpenDaylight** 설치 예를 살펴봅니다. 이 시나리오에서는 테넌트 트래픽 분리에 터널링(*VXLAN*)을 사용합니다.

6.1.1. 물리적 토폴로지

이 시나리오의 토폴로지는 **6**개의 노드로 구성됩니다.

- **x director** 언더클라우드 노드 **1**개
- 다른 **OpenStack** 서비스 외에도 **OpenDaylight SDN** 컨트롤러가 설치된 **x OpenStack overcloud** 컨트롤러 **3**
- 두 개의 **x OpenStack** 오버클라우드 컴퓨팅 노드

6.1.2. 물리적 네트워크 환경 계획

오버클라우드 컨트롤러 노드는 각각 **3**개의 **NIC(**네트워크 인터페이스 카드**)**를 사용합니다.

이름	목적
nic1	관리 네트워크(예: SSH를 통해 노드에 액세스)
nic2	테넌트 (*VXLAN*) 캐리어, 프로비저닝 (PXE, DHCP), *내부 API* 네트워크
nic3	공용 API 네트워크 액세스

오버클라우드 컴퓨팅 노드에는 다음 세 개의 **NIC**가 제공됩니다.

이름	목적
nic1	관리 네트워크

이름	목적
nic2	테넌트 통신업체, 프로비저닝 및 *내부 API* 네트워크
nic3	*external* (Floating IP) 네트워크

언더클라우드 노드에는 두 개의 NIC가 제공됩니다.

이름	목적
nic1	관리 네트워크에 사용
nic2	프로비저닝 네트워크에 사용

6.1.3. 계획 NIC Connectivity

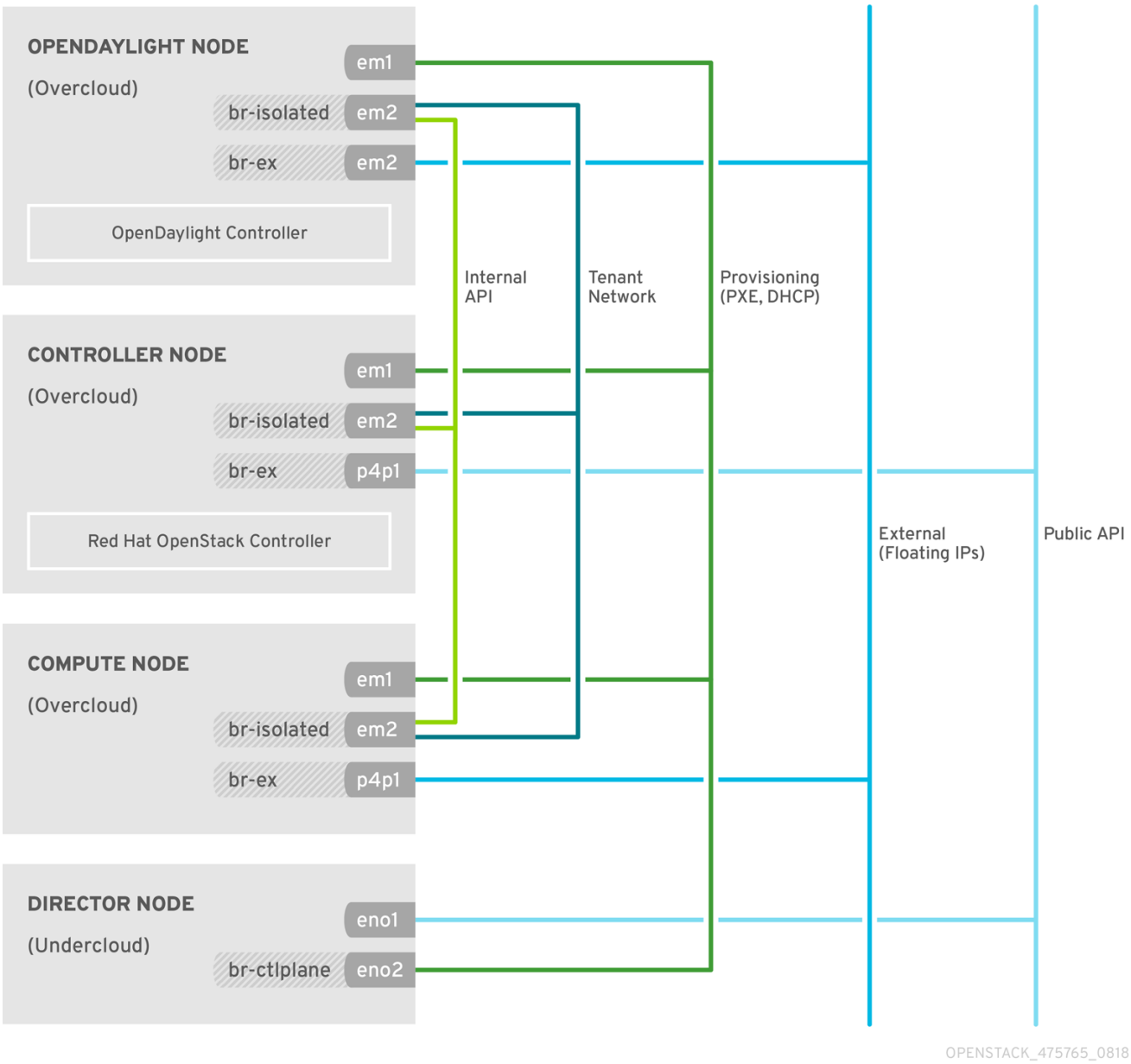
이 경우 환경 파일은 추상 번호가 지정된 인터페이스(nic1,nic2)를 사용하며 호스트 운영 체제(eth0 또는 eno2)에 제공되는 실제 장치 이름은 아닙니다. 동일한 역할에 속하는 호스트에는 동일한 네트워크 인터페이스 장치 이름이 필요하지 않습니다. 한 호스트에서 em1 및 em2 인터페이스를 사용하는 반면 다른 호스트에서 eno1 및 eno2 를 사용하는 경우 문제가 없습니다. 각 NIC를 nic1 및 nic2 라고 합니다.

추상화된 NIC 스키마는 실시간 및 연결된 인터페이스에만 의존합니다. 호스트에 다른 수의 인터페이스가 있는 경우 호스트를 연결하는 데 필요한 최소한의 인터페이스를 사용하는 것으로 충분합니다. 예를 들어, 한 호스트에 4개의 물리적 인터페이스가 있고 다른 호스트에 6개의 물리적 인터페이스가 있는 경우 nic1,nic2,nic3, nic4 를 사용하고 두 호스트의 4개의 케이블로 연결해야합니다.

6.1.4. 네트워크, VLAN 및 IP 계획

이 시나리오에서는 네트워크 분리를 사용하여 관리, 프로비저닝, *내부 API*, 테넌트, 공용 API 및 유동 IP 네트워크 트래픽을 분리합니다. 이 그래픽은 네트워크 구성의 예입니다. 사용자 지정 역할 배포를 보여줍니다. 필요한 경우 Red Hat OpenStack Platform conroller에 OpenDaylight를 포함할 수도 있습니다. 이는 기본 설정입니다. 이 스키마 IPMI 네트워크에 NIC 및 라우팅이 표시되지 않습니다. OpenStack 구성에 따라 추가 네트워크가 필요할 수 있습니다.

그림 6.1. 이 시나리오에서 사용되는 세부 네트워크 토폴로지

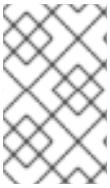


표에는 각 네트워크와 연결된 **VLAN ID** 및 **IP** 서브넷이 표시됩니다.

네트워크	VLAN ID	IP Subnet
프로비저닝	실제 하드웨어에 적용	192.0.5.0/24
내부 API	600	172.17.0.0/24
테넌트	603	172.16.0.0/24
공용 API	411	10.35.184.144/28
부동 IP	412	10.35.186.146/28

OpenStack Platform director는 **br-isolated OVS** 브리지를 생성하고 네트워크 구성 파일에 정의된 대로 각 네트워크의 ***VLAN*** 인터페이스를 추가합니다. **director**는 관련 네트워크 인터페이스가 연결된 ***br-ex*** 브리지도 생성합니다.

호스트 간에 연결을 제공하는 물리적 네트워크 스위치가 해당 ***VLAN ID*** 를 전달하도록 올바르게 구성되었는지 확인합니다. ***VLAN*** 을 사용하여 호스트를 대상으로 하는 모든 스위치 포트를 **"trunks"**로 구성해야 합니다. 여기서 **"trunk"**라는 용어는 여러 ***VLAN ID*** 가 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다.



참고

물리적 스위치에 대한 구성 지침은 이 문서의 범위를 벗어납니다.

참고

network-environment.yaml 파일에서 **TenantNetworkVlanID** 특성을 사용하여 ***VXLAN*** 터널링을 사용할 때 테넌트 네트워크에 대한 **VLAN** 태그를 정의할 수 있습니다. 예를 들어 ***VXLAN*** 테넌트 트래픽이 **VLAN** 태그된 오버레이 네트워크를 통해 전송됩니다. 테넌트 네트워크가 기본 **VLAN**을 통해 실행하려는 경우에도 이 값을 비워 둘 수 있습니다. 또한 **VLAN** 테넌트 유형 네트워크를 사용할 때 **TenantNetworkVlanID** 에 제공된 값 이외의 **VLAN** 태그를 사용할 수 있습니다.

6.1.5. 이 시나리오에서 사용되는 OpenDaylight 구성 파일

OpenStack 및 **OpenDaylight**의 이 시나리오를 배포하려면 언더클라우드 노드에 다음 배포 명령이 입력되었습니다.

```
$ openstack overcloud deploy --debug \
  --templates \
  --environment-file "$HOME/extra_env.yaml" \
  --libvirt-type kvm \
  -e /home/stack/baremetal-vlan/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-opendaylight.yaml \
  --log-file overcloud_install.log &> overcloud_install.log
```

또한 이 가이드에서는 이 시나리오에서 사용되는 구성 파일, 콘텐츠도 표시되며 사용된 설정에 대한 설명도 제공합니다.

6.1.5.1. extra_env.yaml 파일

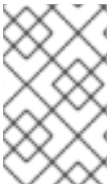
파일에는 하나의 매개변수만 있습니다.

```
parameter_defaults:
  OpenDaylightProviderMappings: 'datacentre:br-ex,tenant:br-isolated'
```

이러한 매핑은 **OpenDaylight**에서 제어하는 각 노드에서 사용할 매핑입니다. 물리적 네트워크 데이터센터 는 **br-ex OVS** 브리지에 매핑되고 테넌트 네트워크 트래픽은 **br-isolated OVS** 브리지에 매핑됩니다.

6.1.5.2. undercloud.conf 파일

이 파일은 **/home/stack/baremetal-vlan/** 디렉터리에 있습니다.



참고

파일 경로는 사용자 지정된 구성 파일의 버전을 가리킵니다.

```
[DEFAULT]
local_ip = 192.0.5.1/24
network_gateway = 192.0.5.1
undercloud_public_vip = 192.0.5.2
undercloud_admin_vip = 192.0.5.3
local_interface = eno2
network_cidr = 192.0.5.0/24
masquerade_network = 192.0.5.0/24
dhcp_start = 192.0.5.5
dhcp_end = 192.0.5.24
inspection_iprange = 192.0.5.100,192.0.5.120
```

이 예에서는 프로비저닝 네트워크의 **192.0.5.0/24** 서브넷이 사용됩니다. 물리적 인터페이스 **eno2** 는 프로비저닝을 위해 언더클라우드 노드에서 사용됩니다.

6.1.5.3. network-environment.yaml 파일

이 파일은 네트워크를 구성하는 데 사용되는 기본 파일입니다. **/home/stack/baremetal-vlan/** 디렉터리에 있습니다. 다음 파일에서 **VLAN ID** 및 **IP** 서브넷은 다른 네트워크와 공급자 매핑에 대해 지정됩니다. **nic-configs** 디렉터리 **controller.yaml** 및 **compute.yaml** 의 두 파일은 컨트롤러 및 컴퓨팅 노드의 네트워크 구성을 지정하는 데 사용됩니다.

컨트롤러 노드 **(3)** 및 컴퓨팅 노드**(2)**의 수가 예에 지정되어 있습니다.

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to use for
  # override the default.
  OS1::TripleO::Compute::Net::SoftwareConfig: nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml

  # Network isolation configuration
  # Service section
  # If some service should be disable, use the following example
  # OS::TripleO::Network::Management: OS::Heat::None
    OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-templates/network/external.yaml
    OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
    OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-templates/network/tenant.yaml
    OS::TripleO::Network::Management: OS::Heat::None
    OS::TripleO::Network::StorageMgmt: OS::Heat::None
    OS::TripleO::Network::Storage: OS::Heat::None

  # Port assignments for the VIP addresses
    OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
    OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
    OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/vip.yaml
    OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
    OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml

  # Port assignments for the controller role
    OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
    OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
    OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant.yaml
    OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
    OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
    OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml

  # Port assignments for the Compute role
    OS::TripleO::Compute::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
    OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
    OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant.yaml
    OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
    OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
    OS::TripleO::Compute::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
```

```
templates/network/ports/noop.yaml

  # Port assignments for service virtual IP addresses for the controller role
    OS::TripleO::Controller::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/vip.yaml

parameter_defaults:
  # Customize all these values to match the local environment
  InternalApiNetCidr: 172.17.0.0/24
  TenantNetCidr: 172.16.0.0/24
  ExternalNetCidr: 10.35.184.144/28
  # CIDR subnet mask length for provisioning network
  ControlPlaneSubnetCidr: '24'
  InternalApiAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  TenantAllocationPools: [{'start': '172.16.0.100', 'end': '172.16.0.200'}]
  # Use an External allocation pool which will leave room for floating IP addresses
  ExternalAllocationPools: [{'start': '10.35.184.146', 'end': '10.35.184.157'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.35.184.158
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.5.254
  # Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.5.1
  InternalApiNetworkVlanID: 600
  TenantNetworkVlanID: 603
  ExternalNetworkVlanID: 411
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["10.35.28.28","8.8.8.8"]
  # May set to br-ex if using floating IP addresses only on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: """"
  # The tunnel type for the tenant network (vxlan or gre). Set to '' to disable tunneling.
  NeutronTunnelTypes: ''
  # The tenant network type for Neutron (vlan or vxlan).
  NeutronNetworkType: 'vxlan'
  # The OVS logical->physical bridge mappings to use.
  # NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-isolated'
  # The Neutron ML2 and OpenVSwitch vlan mapping range to support.
  NeutronNetworkVLANRanges: 'datacentre:412:412'
  # Nova flavor to use.
  OvercloudControlFlavor: baremetal
  OvercloudComputeFlavor: baremetal
  # Number of nodes to deploy.
  ControllerCount: 3
  ComputeCount: 2

  # Sets overcloud nodes custom names
  # http://docs.openstack.org/developer/tripleo-
docs/advanced_deployment/node_placement.html#custom-hostnames
  ControllerHostnameFormat: 'controller-%index%'
  ComputeHostnameFormat: 'compute-%index%'
  CephStorageHostnameFormat: 'ceph-%index%'
  ObjectStorageHostnameFormat: 'swift-%index%'
```

6.1.5.4. controller.yaml 파일

파일은 **/home/stack/baremetal-vlan/nic-configs/** 디렉터리에 있습니다. 이 예제에서는 **br-isolated** 및 **br-ex** 의 스위치 두 개를 정의합니다. **nic2** 는 **br-isolated** 및 **nic3** 아래에 있습니다.

```
heat_template_version: pike

description: >
  Software Config to drive os-net-config to configure VLANs for the
  controller role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  ExternalNetworkVlanID:
    default: ''
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: ''
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
    type: number
```

```
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ControlPlaneSubnetCidr: # Override this with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  DnsServers: # Override this with parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations) that will be added to
resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this with parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                -
                  ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
              members:
                -
                  type: interface
                  name: nic2
                  # force the MAC address of the bridge to this interface
                  primary: true
                -
                  type: vlan
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: InternalApiIpSubnet}
                -
                  type: vlan
                  vlan_id: {get_param: TenantNetworkVlanID}
```

```
            addresses:
              -
                ip_netmask: {get_param: TenantIpSubnet}
          -
            type: ovs_bridge
            name: br-ex
            use_dhcp: false
            dns_servers: {get_param: DnsServers}
            members:
              -
                type: interface
                name: nic3
                # force the MAC address of the bridge to this interface
              -
                type: vlan
                vlan_id: {get_param: ExternalNetworkVlanID}
                addresses:
                -
                  ip_netmask: {get_param: ExternalIpSubnet}
                routes:
                  -
                    default: true
                    next_hop: {get_param: ExternalInterfaceDefaultRoute}

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

6.1.5.5. compute.yaml 파일

파일은 /home/stack/baremetal-vlan/nic-configs/ 디렉터리에 있습니다. 컴퓨팅 구성의 대부분의 옵션은 컨트롤러 구성과 동일합니다. 이 예제에서 nic3 는 외부 연결에 사용할 br-ex 아래에 있습니다(IP 네트워크 연결)

```
heat_template_version: pike

description: >
  Software Config to drive os-net-config to configure VLANs for the
  Compute role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
```

```
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  InternalApiNetworkVlanID:
    default: ''
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  ControlPlaneSubnetCidr: # Override this with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this with parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this with parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations) that will be added to
resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this with parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
```

```
        network_config:
          -
            type: ovs_bridge
            name: br-isolated
            use_dhcp: false
            dns_servers: {get_param: DnsServers}
            addresses:
              -
                ip_netmask:
                  list_join:
                    - '/'
                    - - {get_param: ControlPlaneIp}
                      - {get_param: ControlPlaneSubnetCidr}
            routes:
              -
                ip_netmask: 169.254.169.254/32
                next_hop: {get_param: EC2MetadataIp}
              -
                next_hop: {get_param: ControlPlaneDefaultRoute}
            members:
              -
                type: interface
                name: nic2
                # force the MAC address of the bridge to this interface
                primary: true
              -
                type: vlan
                vlan_id: {get_param: InternalApiNetworkVlanID}
                addresses:
                  -
                    ip_netmask: {get_param: InternalApiIpSubnet}
              -
                type: vlan
                vlan_id: {get_param: TenantNetworkVlanID}
                addresses:
                  -
                    ip_netmask: {get_param: TenantIpSubnet}
          -
            type: ovs_bridge
            name: br-ex
            use_dhcp: false
            members:
              -
                type: interface
                name: nic3

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

6.1.6. 이 시나리오에서 사용되는 Red Hat OpenStack Platform director 구성 파일

6.1.6.1. neutron.conf 파일

이 파일은 **/etc/neutron/** 디렉터리에 있으며 다음 정보가 포함되어야 합니다.

```
[DEFAULT]
service_plugins=odl-router_v2,trunk
```

6.1.6.2. ml2_conf.ini 파일

이 파일은 **/etc/neutron/plugins/ml2/** 디렉터리에 있으며 다음 정보가 포함되어야 합니다.

```
[ml2]
type_drivers = vxlan,vlan,flat,gre
tenant_network_types = vxlan
mechanism_drivers = opendaylight_v2

[ml2_type_vlan]
network_vlan_ranges = datacentre:412:412

[ml2_odl]
password = admin
username = admin
url = http://172.17.1.18:8081/controller/nb/v2/neutron
```

1. **[ml2]** 섹션에서 **VXLAN**은 네트워크 유형으로 사용되며 **opendaylight_v2** 메커니즘 드라이버입니다.

2. **[ml2_type_vlan]**에서 **network-environment.yaml** 파일에 구성된 것과 동일한 매핑을 설정해야 합니다.

3. **[ml2_odl]** 아래에 **OpenDaylightController**에 액세스하는 구성이 표시되어야 합니다.

이러한 세부 정보를 사용하여 **OpenDaylight** 컨트롤러에 대한 액세스를 확인할 수 있습니다.

```
$ curl -H "Content-Type:application/json" -u admin:admin
http://172.17.1.18:8081/controller/nb/v2/neutron/networks
```

6.2. 공급자 네트워크를 사용하여 설치 시나리오 모델링

이 설치 시나리오에서는 테넌트 네트워크 대신 공급자 네트워크를 사용한 **OpenStack** 및 **OpenDaylight**의 예를 보여줍니다. 외부 **neutron** 공급자 네트워크는 **VM** 인스턴스를 레이어**-3(L3)** 및 기타 네트워크 서비스를 제공하는 물리적 네트워크 인프라에 연결합니다. 대부분의 경우 공급자 네트워크는

VLAN ID를 사용하여 **L2** 계층(**L2**) 세그먼트를 구현합니다. 공급자 네트워크는 공급자 네트워크에서 **VM** 인스턴스 시작을 지원하는 각 컴퓨팅 노드의 공급자 브릿지에 매핑됩니다.

6.2.1. 물리적 토폴로지

이 시나리오의 토폴로지는 **6**개의 노드로 구성됩니다.

- **x director** 언더클라우드 노드 **1**개
- 다른 **OpenStack** 서비스 외에도 **OpenDaylight SDN** 컨트롤러가 설치된 **x OpenStack overcloud** 컨트롤러 **3**
- 두 개의 **x OpenStack** 오버클라우드 컴퓨팅 노드

6.2.2. 물리적 네트워크 환경 계획

오버클라우드 컨트롤러 노드는 각각 **4**개의 **NIC**(네트워크 인터페이스 카드)를 사용합니다.

이름	목적
nic1	관리 네트워크(예: SSH를 통해 노드에 액세스)
nic2	provisioning(PXE, DHCP), *내부 API* 네트워크
nic3	테넌트 네트워크
nic4	공용 API 네트워크, 유동 IP 네트워크

오버클라우드 컴퓨팅 노드에는 네 개의 **NIC**가 있습니다.

이름	목적
nic1	관리 네트워크
nic2	프로비저닝 및 *내부 API* 네트워크
nic3	테넌트 네트워크

이름	목적
nic4	부동 IP 네트워크

언더클라우드 노드에는 두 개의 NIC가 제공됩니다.

이름	목적
nic1	관리 네트워크에 사용
nic2	프로비저닝 네트워크에 사용

6.2.3. 계획 NIC Connectivity

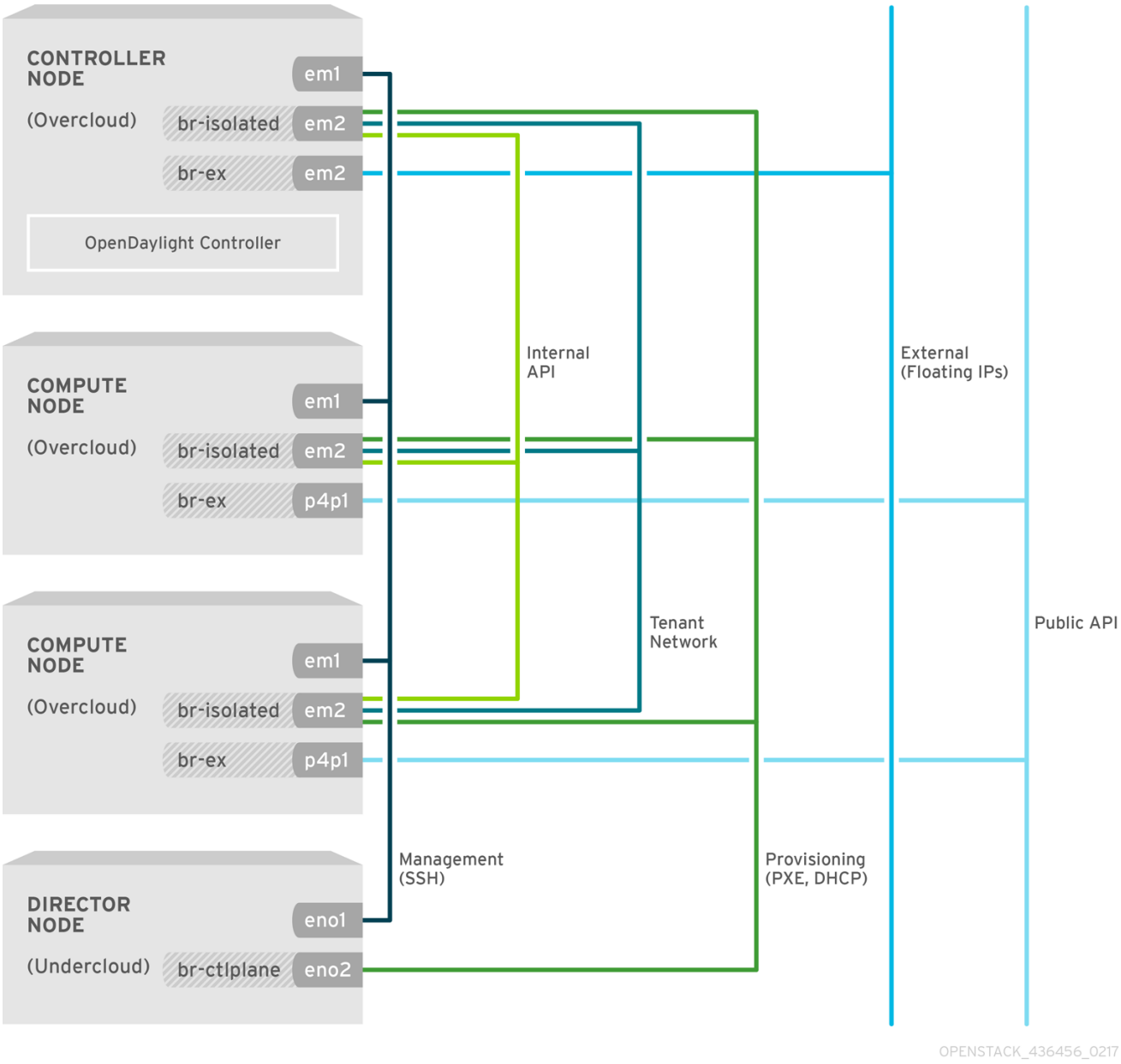
이 경우 환경 파일은 추상 번호가 지정된 인터페이스(nic1,nic2)를 사용하며 호스트 운영 체제(예: eth0 또는 eno 2)에 제공되는 실제 장치 이름은 아닙니다. 동일한 역할에 속하는 호스트에는 동일한 네트워크 인터페이스 장치 이름이 필요하지 않습니다. 한 호스트에서 em1 및 em2 인터페이스를 사용하는 반면 다른 호스트에서 eno1 및 eno2 를 사용하는 경우 문제가 없습니다. 각 NIC를 nic1 및 nic2 라고 합니다.

추상화된 NIC 스키마는 실시간 및 연결된 인터페이스에만 의존합니다. 호스트에 다른 수의 인터페이스가 있는 경우 호스트를 연결하는 데 필요한 최소한의 인터페이스를 사용하는 것으로 충분합니다. 예를 들어, 한 호스트에 4개의 물리적 인터페이스가 있고 다른 호스트에 6개의 물리적 인터페이스가 있는 경우 nic1,nic2,nic3, nic4 를 사용하고 두 호스트의 4개의 케이블로 연결해야합니다.

6.2.4. 네트워크, VLAN 및 IP 계획

이 시나리오에서는 네트워크 분리를 사용하여 관리, 프로비저닝, *내부 API*, 테넌트, 공용 API 및 유동 IP 네트워크 트래픽을 분리합니다.

그림 6.2. 이 시나리오에서 사용되는 세부 네트워크 토폴로지



표에는 각 네트워크와 연결된 **VLAN ID** 및 **IP** 서브넷이 표시됩니다.

네트워크	VLAN ID	IP Subnet
프로비저닝	실제 하드웨어에 적용	192.0.5.0/24
내부 API	600	172.17.0.0/24
테넌트	554,555-601	172.16.0.0/24
공용 API	552	192.168.210.0/24
부동 IP	553	10.35.186.146/28

OpenStack Platform director는 ***br-isolated*** **OVS** 브리지를 만들고 네트워크 구성 파일에 정의된 대로 각 네트워크의 ***VLAN*** 인터페이스를 추가합니다. **director**는 또한 연결된 관련 네트워크 인터페이스를 사용하여 **br-ex** 브릿지를 자동으로 생성합니다.

호스트 간에 연결을 제공하는 물리적 네트워크 스위치가 해당 ***VLAN ID*** 를 전달하도록 올바르게 구성되었는지 확인합니다. ***VLAN*** 을 사용하여 호스트에 있는 모든 스위치 포트를 *트렁크* 로 구성해야 합니다. 여기서 **"trunk"**라는 용어는 여러 ***VLAN ID*** 가 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다.

참고

물리적 스위치에 대한 구성 지침은 이 문서의 범위를 벗어납니다.

참고

network-environment.yaml 의 **TenantNetworkVlanID** 는 ***VXLAN*** 터널링을 사용할 때 테넌트 네트워크에 대해 **VLAN** 태그를 정의할 수 있습니다(예: **VLAN** 태그를 붙은 **VLAN** 태그를 통해 전송한 ***VXLAN*** 테넌트 트래픽). 테넌트 네트워크가 기본 **VLAN**을 통해 실행하려는 경우에도 이 값이 비어 있을 수 있습니다. 또한 **VLAN** 테넌트 유형 네트워크를 사용하는 경우 **TenantNetworkVlanID** 에 제공된 값 이외의 **VLAN** 태그를 사용할 수 있습니다.

6.2.5. 이 시나리오에서 사용되는 OpenDaylight 구성 파일

OpenStack 및 **OpenDaylight**의 이 시나리오를 배포하려면 언더클라우드 노드에 다음 배포 명령이 입력되었습니다.

```
$ openstack overcloud deploy --debug \
  --templates \
  --environment-file "$HOME/extra_env.yaml" \
  --libvirt-type kvm \
  -e /home/stack/baremetal-vlan/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-opendaylight.yaml \
  --log-file overcloud_install.log &> overcloud_install.log
```

이 가이드에서는 또한 이 시나리오의 구성 파일, 구성 파일 콘텐츠 및 구성에 대한 설명 정보를 보여줍니다.

6.2.5.1. extra_env.yaml 파일

파일에는 하나의 매개변수만 있습니다.

```
parameter_defaults:
  OpenDaylightProviderMappings: 'datacentre:br-ex,tenant:br-vlan'
```

이러한 매핑은 **OpenDaylight**에서 제어하는 각 노드에서 사용할 매핑입니다. 물리적 네트워크 데이터 센터 는 **br-ex OVS** 브리지에 매핑되며 테넌트 네트워크 트래픽은 **br-vlan OVS** 브리지에 매핑됩니다.

6.2.5.2. undercloud.conf 파일

이 파일은 **/home/stack/** 디렉터리에 있습니다.

참고

파일 경로는 사용자 지정된 구성 파일의 버전을 가리킵니다.

```
[DEFAULT]
local_ip = 192.0.5.1/24
network_gateway = 192.0.5.1
undercloud_public_vip = 192.0.5.2
undercloud_admin_vip = 192.0.5.3
local_interface = eno2
network_cidr = 192.0.5.0/24
masquerade_network = 192.0.5.0/24
dhcp_start = 192.0.5.5
dhcp_end = 192.0.5.24
inspection_iprange = 192.0.5.100,192.0.5.120
```

이 예에서는 프로비저닝 네트워크에 **192.0.5.0/24** 서브넷을 사용합니다. 물리적 인터페이스 **eno2** 는 프로비저닝을 위해 언더클라우드 노드에서 사용됩니다.

6.2.5.3. network-environment.yaml 파일

이 파일은 네트워크를 구성하는 데 사용되는 기본 파일입니다. **/home/stack/baremetal-vlan/** 디렉터리에 있습니다. 다음 파일은 다른 네트워크의 **VLAN ID** 및 **IP** 서브넷을 지정하고 공급자 매핑을 보여줍니다. **nic-configs** 디렉터리의 **controller.yaml** 및 **compute.yaml** 파일은 컨트롤러 및 컴퓨팅 노드의 네트워크 구성을 지정하는 데 사용됩니다.

컨트롤러 노드 **(3)** 및 컴퓨팅 노드**(2)**의 수가 예에 지정되어 있습니다.

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to use for override the default.
  OS::TripleO::Compute::Net::SoftwareConfig: nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml

  # Network isolation configuration
  # Service section
  # If some service should be disabled, use the following example
  # OS::TripleO::Network::Management: OS::Heat::None
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-templates/network/tenant.yaml
  OS::TripleO::Network::Management: OS::Heat::None
  OS::TripleO::Network::StorageMgmt: OS::Heat::None
  OS::TripleO::Network::Storage: OS::Heat::None

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/vip.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external.yaml
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant.yaml
  OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Compute::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
```

```
  # Port assignments for service virtual IPs for the controller role
  OS::TripleO::Controller::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/vip.yaml
  OS::TripleO::NodeUserData: /home/stack/baremetal-vlan/firstboot-config.yaml

parameter_defaults:
  # Customize all these values to match the local environment
  InternalApiNetCidr: 172.17.0.0/24
  TenantNetCidr: 172.16.0.0/24
  ExternalNetCidr: 192.168.210.0/24
  # CIDR subnet mask length for provisioning network
  ControlPlaneSubnetCidr: '24'
  InternalApiAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  TenantAllocationPools: [{'start': '172.16.0.100', 'end': '172.16.0.200'}]
  # Use an External allocation pool which will leave room for floating IPs
  ExternalAllocationPools: [{'start': '192.168.210.2', 'end': '192.168.210.12'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.210.1
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.5.1
  # Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.5.1
  InternalApiNetworkVlanID: 600
  TenantNetworkVlanID: 554
  ExternalNetworkVlanID: 552
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["10.35.28.28","8.8.8.8"]
  # May set to br-ex if using floating IPs only on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: """"
  # The tunnel type for the tenant network (vxlan or gre). Set to '' to disable tunneling.
  NeutronTunnelTypes: ''
  # The tenant network type for Neutron (vlan or vxlan).
  NeutronNetworkType: 'vlan'
  # The OVS logical->physical bridge mappings to use.
  #  NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-isolated'
  # The Neutron ML2 and OpenVSwitch vlan mapping range to support.
  NeutronNetworkVLANRanges: 'datacentre:552:553,tenant:555:601'
  # Nova flavor to use.
  OvercloudControlFlavor: baremetal
  OvercloudComputeFlavor: baremetal
  # Number of nodes to deploy.
  ControllerCount: 3
  ComputeCount: 2

  # Sets overcloud nodes custom names
  # http://docs.openstack.org/developer/tripleo-
docs/advanced_deployment/node_placement.html#custom-hostnames
  ControllerHostnameFormat: 'controller-%index%'
  ComputeHostnameFormat: 'compute-%index%'
  CephStorageHostnameFormat: 'ceph-%index%'
  ObjectStorageHostnameFormat: 'swift-%index%'
```

6.2.5.4. controller.yaml 파일

이 파일은 **/home/stack/baremetal-vlan/nic-configs/** 디렉터리에 있습니다. 이 예제에서는 **br-isolated,br-vlan, br-ex** 등의 스위치를 정의합니다. **NIC2** 는 **br-isolated** 에 있으며 **nic3** 는 **br-ex** 에 있습니다.

```
heat_template_version: pike

description: >
  Software Config to drive os-net-config to configure VLANs for the
  controller role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  ExternalNetworkVlanID:
    default: ''
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: ''
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
```

```
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ControlPlaneSubnetCidr: # Override this with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  DnsServers: # Override this with parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations) that will be added to
resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this with parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: interface
              name: nic1
              use_dhcp: false
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                -
                  ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
              members:
                -
                  type: interface
                  name: nic2
                  # force the MAC address of the bridge to this interface
                  primary: true
                -
                  type: vlan
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
```

```
            -
              ip_netmask: {get_param: InternalApiIpSubnet}
        -
          type: ovs_bridge
          name: br-ex
          use_dhcp: false
          dns_servers: {get_param: DnsServers}
          members:
            -
              type: interface
              name: nic4
              # force the MAC address of the bridge to this interface
            -
              type: vlan
              vlan_id: {get_param: ExternalNetworkVlanID}
              addresses:
              -
                ip_netmask: {get_param: ExternalIpSubnet}
              routes:
                -
                  default: true
                  next_hop: {get_param: ExternalInterfaceDefaultRoute}
        -
          type: ovs_bridge
          name: br-vlan
          use_dhcp: false
          dns_servers: {get_param: DnsServers}
          members:
            -
              type: interface
              name: nic3
            -
              type: vlan
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

6.2.5.5. compute.yaml 파일

이 파일은 /home/stack/baremetal-vlan/nic-configs/ 디렉터리에 있습니다. 컴퓨팅 구성의 대부분의 옵션은 컨트롤러 구성과 동일합니다. 이 예제에서 nic4 는 외부 연결에 사용할 br-ex 아래에 있습니다(IP 네트워크 연결)

```
heat_template_version: pike

description: >
  Software Config to drive os-net-config to configure VLANs for the
```

```
  compute role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  InternalApiNetworkVlanID:
    default: ''
    description: Vlan ID for the internal_api network traffic.
    type: number
  TenantNetworkVlanID:
    default: ''
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 23
    description: Vlan ID for the management network traffic.
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  ControlPlaneSubnetCidr: # Override this with parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this with parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this with parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations) that will be added to
resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this with parameter_defaults
```

```
    description: The IP address of the EC2 metadata server.
    type: string
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: interface
              name: nic1
              use_dhcp: false
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
               -
                 ip_netmask:
                   list_join:
                     - '/'
                     - - {get_param: ControlPlaneIp}
                       - {get_param: ControlPlaneSubnetCidr}
              routes:
               -
                 ip_netmask: 169.254.169.254/32
                 next_hop: {get_param: EC2MetadataIp}
               -
                 next_hop: {get_param: ControlPlaneDefaultRoute}
                 default: true
              members:
                -
                  type: interface
                  name: nic2
                  # force the MAC address of the bridge to this interface
                  primary: true
                -
                  type: vlan
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    -
                      ip_netmask: {get_param: InternalApiIpSubnet}
            -
              type: ovs_bridge
              name: br-ex
              use_dhcp: false
              members:
                -
```

```
            type: interface
            name: nic4
        -
          type: ovs_bridge
          name: br-vlan
          use_dhcp: false
          dns_servers: {get_param: DnsServers}
          members:
            -
              type: interface
              name: nic3
            -
              type: vlan
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}


outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}
```

6.2.6. 이 시나리오에서 사용되는 Red Hat OpenStack Platform director 구성 파일

6.2.6.1. neutron.conf 파일

이 파일은 **/etc/neutron/** 디렉터리에 있으며 다음 정보를 포함합니다.

```
[DEFAULT]
service_plugins=odl-router_v2,trunk
```

6.2.6.2. ml2_conf.ini file

이 파일은 **/etc/neutron/plugins/ml2/** 디렉터리에 있으며 다음 정보를 포함합니다.

```
[DEFAULT]
[ml2]
type_drivers = vxlan,vlan,flat,gre
tenant_network_types = vlan
mechanism_drivers = opendaylight_v2
extension_drivers = qos,port_security
path_mtu = 0

[ml2_type_flat]
flat_networks = datacentre

[ml2_type_geneve]
```

```
[ml2_type_gre]
tunnel_id_ranges = 1:4094

[ml2_type_vlan]
network_vlan_ranges = datacentre:552:553,tenant:555:601

[ml2_type_vxlan]
vni_ranges = 1:4094
vxlan_group = 224.0.0.1

[securitygroup]
[ml2_odl]
password=<PASSWORD>
username=<USER>
url=http://172.17.0.10:8081/controller/nb/v2/neutron
```

1. **[ml2] 섹션에서 VXLAN은 네트워크 유형으로 사용되며 opendaylight_v2 메커니즘 드라이버입니다.**

2. **[ml2_type_vlan]에서 network-environment.yaml 파일에서와 동일한 매핑을 설정합니다.**

3. **[ml2_odl] 아래에 OpenDaylightController에 액세스하는 구성이 표시되어야 합니다.**

이러한 세부 정보를 사용하여 OpenDaylight 컨트롤러에 대한 액세스를 확인할 수 있습니다.

```
$ curl -H "Content-Type:application/json" -u admin:admin
http://172.17.1.18:8081/controller/nb/v2/neutron/networks
```

7장. OPENDAYLIGHT를 통한 고가용성 및 클러스터링

Red Hat OpenStack Platform 13은 **neutron** 및 **OpenDaylight** 컨트롤러 모두에서 고가용성 클러스터링을 지원합니다. 다음 표에서는 고가용성 클러스터를 실행하는 데 권장되는 아키텍처를 보여줍니다.

노드 유형	노드 수	노드 모드
Neutron	3	활성/활성/활성
OpenDaylight	3	활성/활성/활성
컴퓨팅 노드(nova 또는 OVS)	Any	

OpenDaylight 역할은 구성 가능하므로 **neutron** 노드와 동일한 노드에 배포하거나 별도의 노드에 배포할 수 있습니다. 설정은 **all-active** 설정입니다. 모든 노드는 요청을 처리할 수 있습니다. 수신 노드가 요청을 처리할 수 없는 경우 노드는 요청을 다른 적절한 노드로 전달합니다. 모든 노드는 서로 동기화를 유지합니다. **OVS(Open vSwitch** 데이터베이스 스키마) **Southbound**에서 사용 가능한 컨트롤러 노드는 **Open vSwitch**를 공유하므로 클러스터의 특정 노드가 각 스위치를 처리합니다.

7.1. 고가용성 및 클러스터링을 위한 OPENDAYLIGHT 구성

Red Hat OpenStack Platform director는 **OpenDaylight** 컨트롤러 노드를 배포하므로 **OpenDaylight**에 대한 클러스터링을 구성하는 데 필요한 모든 정보가 있습니다. 각 **OpenDaylight** 노드에는 노드 *역할*(클러스터의 이름)을 식별하는 **akka.conf** 구성 파일이 필요하며, 초기 노드인 클러스터의 다른 노드 중 적어도 일부를 나열합니다. *노드에는 클러스터에서 데이터가 복제되는 방법을 정의하는* ***module-shards.conf*** *파일도 필요합니다.* ***Red Hat OpenStack Platform director****는 선택한 배포 구성에 따라 올바른 설정을 수행합니다.* ***akka.conf*** *파일은 노드에 따라 다르지만* ***module-shards.conf*** *파일은 노드 및 설치된 데이터 저장소(따라서 설치되는 데이터 저장소)에 따라 달라집니다.*

akka.conf *파일의 예:*

```
$ docker exec -it opendaylight_api cat
/var/lib/kolla/config_files/src/opt/opendaylight/configuration/initial/akka.conf


odl-cluster-data {
  akka {
    remote {
      netty.tcp {
        hostname = "192.0.2.1"
      }
    },
    cluster {
      seed-nodes = [
```

```
        "akka.tcp://opendaylight-cluster-data@192.0.2.1:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.2:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.3:2550"],
      roles = [ "member-1" ]
    }
  }
}
```

이 예제 노드는 시 드 노드 입니다. 현재 클러스터 설정을 전체적으로 반영할 필요가 없습니다. 시드 노드 목록을 사용하여 현재 클러스터의 실제 노드 중 하나에 연결할 수 있는 경우 시작 노드가 클러스터에 참여할 수 있습니다. 구성 파일에서 **IP** 주소 대신 이름을 사용할 수 있습니다.

7.2. 클러스터 동작

클러스터는 동적으로 정의되지 않으므로 자동으로 조정되지 않습니다. 새 노드만 구성하여 새 노드를 시작하고 기존 클러스터에 연결할 수 없습니다. 클러스터 관리 **RPC** 를 통해 노드의 추가 및 제거에 대해 클러스터에 알려야 합니다.

클러스터는 리더/**Followers** 모델입니다. 활성 노드 중 하나가 리더로 선택되고 나머지 활성 노드는 팔로워가 됩니다. 클러스터는 **Raft** 합의 기반 모델에 따라 지속성을 처리합니다. 이 원칙에 따라 클러스터의 대부분의 노드가 동의한 경우에만 트랜잭션이 커밋됩니다.

OpenDaylight에서 노드와 클러스터 연결이 끊어지면 로컬 트랜잭션이 더 이상 진행되지 않습니다. 결국 기본적으로 **10**분 동안 시간 초과되고 프런트 엔드 액터가 중지됩니다. 이 모든 것이 **shard**마다 적용되므로 다른 **shard**는 다른 리더를 가질 수 있습니다. 이 동작은 다음 중 하나로 이어집니다.

- **10** 분 이내에 통신이 부족하면 소수의 노드가 대다수의 리더와 다시 연결됩니다. 모든 트랜잭션이 롤백되고 대부분의 트랜잭션이 재생됩니다.
- **10**분 이상 통신이 없으면 소수 노드가 작동을 중지하고 해당 정보를 로그 메시지에 기록합니다. 읽기 전용 요청은 계속 완료되지만 변경 사항은 유지되지 않으며 노드는 자율적으로 클러스터에 다시 참여할 수 없습니다.

즉, 사용자가 노드를 모니터링해야 합니다. 사용자는 가용성 및 클러스터 동기화를 확인하고 동기화가 너무 긴 경우 다시 시작해야 합니다. 노드를 모니터링하기 위해 사용자는 **Jolokia REST** 서비스를 사용할 수 있습니다. 자세한 내용은 **Monitoring with Jolokia** 를 참조하십시오.

7.3. 클러스터 요구 사항

본딩 또는 **MTU**와 같은 클러스터를 지원하기 위한 특정 네트워킹 요구 사항은 없습니다. 클러스터 통신

은 높은 대기 시간을 지원하지 않지만 데이터 검증 수준에 대기 시간이 허용됩니다.

7.4. OPEN VSWITCH 구성

Red Hat OpenStack Platform director는 각 스위치를 모든 컨트롤러와 자동으로 설정합니다. **OVSDB**는 특정 수준의 부하 분산을 허용하기 위해 클러스터 노드 간 공유 스위치를 지원합니다. 그러나 각 스위치는 클러스터의 모든 노드에 연결하여 먼저 답변하는 노드를 선택하고 기본적으로 마스터 스위치로 만듭니다. 이 동작으로 인해 노드에 가장 빠르게 응답되는 경우 대부분의 스위치를 처리할 때 컨트롤러 할당을 클러스터링할 수 있습니다.

7.5. 클러스터 모니터링

7.5.1. 모니터링 with Jolokia

클러스터 상태를 모니터링하려면 **OpenDaylight**에서 **Jolokia** 지원을 활성화해야 합니다.

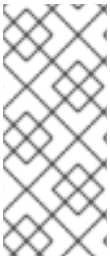
Jolokia 주소에서 구성 데이터 저장소 클러스터링 보고서를 가져옵니다.

```
# curl -u <odl_username>
<odl_password>http://<odl_ip>:8081/jolokia/read/org.opendaylight.controller:type=DistributedConfigData
store,Category=ShardManager,name=shard-manager-config
```

Jolokia 주소에서 운영 데이터 저장소 클러스터링 보고서를 받으십시오.

```
# curl -u <odl_username>
<odl_password>http://<odl_ip>:8081/jolokia/read/org.opendaylight.controller:type=DistributedOperationa
lDatastore,Category=ShardManager,name=shard-manager-operational
```

보고서는 **JSON** 문서입니다.



참고

사용자 환경과 일치하도록 **IP** 주소 와 **member-1** 값을 변경해야 합니다. 응답할 노드가 우선 순위가 없는 경우 **IP** 주소는 **VIP**를 가리킬 수 있습니다. 그러나 특정 컨트롤러를 주소 지정하면 더 관련성이 높은 결과를 얻을 수 있습니다.

이 설명은 각 노드에서 동일한 리더를 표시해야 합니다.

참고

*업스트림 **OpenDaylight** 팀에서 개발한 **Cluster Monitor** 툴 을 사용하여 클러스터를 모니터링할 수도 있습니다. **OpenDaylight Github** 저장소에서 찾을 수 있습니다.*

*이 툴은 **Red Hat OpenStack Platform 13**의 일부가 아니며 **Red Hat**에서 지원하거나 제공하지 않습니다.*

7.6. OPENDAYLIGHT 포트 이해

*모든 **OpenDaylight** 포트의 공식 목록은 **OpenDaylight wiki** 페이지에서 확인할 수 있습니다. 이 시나리오와 관련된 포트는 다음과 같습니다.*

포트 번호	사용자 이름
2550	클러스터링
6653	OpenFlow
6640, 6641	OVSDB
8087	Neutron
8081	RESTCONF, Jolokia

컨트롤러에서 이러한 포트에 대한 트래픽을 차단하면 다음과 같은 효과가 있습니다.

클러스터링

클러스터형 노드는 통신할 수 없습니다. 클러스터형 모드로 실행할 때 각 노드에는 하나 이상의 피어가 있어야 합니다. 모든 트래픽이 차단되면 컨트롤러가 중지됩니다.

OpenFlow

스위치는 컨트롤러에 도달할 수 없습니다.

OVSDB

***Open vSwitch**는 컨트롤러에 연결할 수 없습니다. 컨트롤러는 활성 **OVS** 연결을 시작할 수 있지만, 스위치에서 컨트롤러로 전환하는 **ping**이 실패하고 결국 스위치가 다른 컨트롤러로 장애됩니다.*

Neutron

Neutron은 컨트롤러에 연결할 수 없습니다.

RESTCONF

REST 엔드포인트를 사용하는 외부 툴은 컨트롤러에 연결할 수 없습니다. 이 시나리오에서는 모니터링 도구에만 영향을 미칩니다.

OpenDaylight 측에서 다른 포트가 ODL 컨트롤러와 통신하는 데 사용되므로 로그에는 클러스터링에 대해 차단된 트래픽만 표시됩니다.

대상 장치에서 이러한 포트에 대한 트래픽을 차단하면 다음과 같은 효과가 있습니다.

클러스터링

클러스터형 노드는 통신할 수 없습니다. 클러스터형 모드로 실행할 때 각 노드에는 하나 이상의 피어가 있어야 합니다. 모든 트래픽이 차단되면 컨트롤러가 중지됩니다.

OpenFlow

컨트롤러는 흐름을 푸시할 수 없습니다.

OVSDB

컨트롤러는 스위치에 도달할 수 없습니다(컨트롤러는 패시브 OVS 연결에 응답할 수 있음).

후자의 모든 상황에서 OpenDaylight는 구성 및 운영 상태를 별도의 트리에 유지 관리하기 때문에 구성은 여전히 연결할 수 없는 장치를 가리키며 컨트롤러는 계속 연결을 시도합니다.

7.7. OPENDAYLIGHT FLOWS 이해

flow	설명
Neutron → ODL	ODL의 HA 프록시에 대한 Neutron. Pacemaker에서 VIP를 관리합니다(세 개의 백업 PIP 포함). 드라이버는 TCP 세션을 열어 두려고 시도하여 영향을 줄 수 있습니다(https://review.openstack.org/#/c/440866/).
ODL → Neutron	ODL-initiated 통신이 없습니다.
ODL → ODL	ODL 노드는 포트 2550(구성 가능)에서 서로 통신합니다.

flow	설명
ODL → OVS	ODL은 OVSDB(포트 6640 및 6641) 및 OpenFlow(포트 6633)를 사용하여 스위치와 통신합니다. VIP는 관련이 없습니다. ODL은 모든 스위치의 IP 주소를 알고 있으며 각 ODL 노드는 모든 스위치에 대해 알고 있습니다.
OVS → ODL	ODL은 OVSDB(포트 6640 및 6641) 및 OpenFlow(포트 6633)를 사용하여 스위치와 통신합니다. VIP는 포함되어 있지 않으며, ODL은 모든 컨트롤러를 알 수 있도록 모든 스위치를 구성합니다. 스위치에서 컨트롤러로의 알림은 모든 노드로 전송됩니다.

7.8. NEUTRON DHCP 에이전트 HA

*기본 설정은 **OVS** 에이전트와 함께 모든 **neutron** 노드에서 **DHCP** 에이전트를 실행합니다. 역할은 구성 가능하므로 에이전트가 컨트롤러와 분리할 수 있습니다. **DHCP** 에이전트는 포트 가져 오기 단계 및 리스 갱신 중에만 **HA**에 중요합니다. 포트 생성 시 **neutron**은 **IP** 및 **MAC** 주소를 할당하고 포트가 시작되기 전에 모든 **DHCP** 에이전트를 적절하게 구성합니다. 이 단계에서 모든 **DHCP** 에이전트가 결과 **DHCP** 요청에 응답합니다.*

*__DHCP__ 에이전트 실패 시 데이터 플레인 가용성을 극대화하기 위해 리스가 긴 리스 시간으로 구성되고 노드는 짧은 갱신 지연으로 구성됩니다. 따라서 **DHCP** 에이전트는 거의 필요하지 않지만, 요청 노드에서는 사용할 수 없는 **DHCP** 에이전트가 빠르게 실패하고 브로드캐스트 요청을 발행하여 나머지 **DHCP** 에이전트를 자동으로 선택합니다.*

*에이전트에는 자체 프로세스 모니터가 있습니다. **systemd** 는 에이전트를 시작하고, 네임스페이스를 생성하고, 그 안에 프로세스를 시작합니다. 에이전트가 종료되면 네임스페이스가 그대로 유지되면 **systemd** 는 종료하거나 다른 프로세스를 재시작하지 않고 에이전트를 다시 시작합니다(소유하지 않음). 그런 다음 에이전트는 네임스페이스에 다시 연결하고 실행 중인 모든 프로세스와 함께 다시 사용합니다.*

7.9. NEUTRON 메타데이터 에이전트 HA

*참조 구현에서 메타데이터 서비스는 해당 **DHCP** 에이전트와 동일한 네임스페이스에서 네트워크 노드와 결합된 컨트롤러에서 실행됩니다. 메타데이터 프록시는 포트 **80**에서 수신 대기하고, 정적 경로는 잘 알려진 메타데이터 주소를 사용하여 가상 시스템에서 프록시로 트래픽을 리디렉션합니다. 프록시는 **Unix** 소켓을 사용하여 동일한 노드에 있는 메타데이터 서비스와 통신하고 후자는 **nova**와 통신합니다. **Unix** 소켓을 사용하면 프록시와 서비스 간에 **IP**를 라우팅할 수 없으므로 노드가 라우팅되지 않은 경우에도 메타데이터 서비스를 사용할 수 있습니다. **HA**는 유지 관리 및 **VRRP** 선거를 사용하여 처리됩니다. 장애 조치 **(failover)** 시간은 **2-5s**입니다. 에이전트는 **DHCP** 에이전트와 동일한 방식으로 처리됩니다(**systemd** 및 네임스페이스 사용).*

***Red Hat OpenStack Platform 11**의 메타데이터 서비스는 사용자 지정 **Python** 스크립트이며 **Red Hat OpenStack Platform 13**에서는 메모리 사용량을 **30**으로 낮추는 **HAProxy**입니다. 많은 사용자가 라우터마다 하나의 프록시를 실행하고 컨트롤러당 수천 개의 라우터가 아닌 경우 수백 개의 프록시를 실행하므로 특히 중요합니다.*

8장. RED HAT OPENSTACK PLATFORM 및 OPENDAYLIGHT에 대한 자세한 정보는 어디에서 찾을 수 있습니까?

구성 요소	reference
OpenDaylight	이 문서에서 다루지 않는 자세한 내용은 OpenDaylight Carbon 설명서를 참조하십시오.
Red Hat OpenDaylight 제품 가이드	Red Hat OpenDaylight 및 Red Hat OpenStack Platform과 관련된 자세한 내용은 Red Hat OpenDaylight 제품 가이드를 참조하십시오.
Red Hat Enterprise Linux	Red Hat OpenStack Platform은 Red Hat Enterprise Linux 7.4에서 지원됩니다. Red Hat Enterprise Linux 설치에 대한 자세한 내용은 Red Hat Enterprise Linux 설치 가이드의 해당 설치 가이드를 참조하십시오.
Red Hat OpenStack Platform	OpenStack 구성 요소 및 해당 종속 항목을 설치하려면 Red Hat OpenStack Platform director를 사용합니다. director는 기본 OpenStack 언더클라우드를 사용합니다. 그런 다음 최종 오버클라우드에서 OpenStack 노드를 프로비저닝하고 관리하는 데 사용됩니다. 배포된 오버클라우드에 필요한 환경 외에도 언더클라우드를 설치하기 위해 하나의 추가 호스트 시스템이 필요합니다. 자세한 내용은 Director Installation and Usage 를 참조하십시오. 네트워크 격리, 스토리지 구성, SSL 통신 및 일반 설정 방법과 같은 Red Hat OpenStack Platform director를 사용하는 Red Hat OpenStack Platform 엔터프라이즈 환경에 대한 고급 기능을 구성하는 방법에 대한 자세한 내용은 Advanced Overcloud Customization 을 참조하십시오.
NFV 문서	NFV를 통한 Red Hat OpenStack Platform 배포 계획에 대한 자세한 내용은 Network Functions Virtualization Planning and Configuration Guide를 참조하십시오.