Red Hat OpenStack Platform 13

네트워킹 가이드

Red Hat OpenStack Platform Networking에 대한 고급 가이드

Last Updated: 2023-05-25

Red Hat OpenStack Platform 13 네트워킹 가이드

Red Hat OpenStack Platform Networking에 대한 고급 가이드

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

일반적인 OpenStack 네트워킹 작업을 위한 쿡북.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체 6

RED HAT 문서에 관한 피드백 제공 7

1장. 네트워킹 개요 8
1.1. 네트워킹 작동 방식 8
1.2. 두 개의 LAN 연결 8
1.3. RED HAT OPENSTACK NETWORK 흐름 매트릭스 9
1.4. OPENSTACK NETWORKING(NEUTRON) 작업 10
1.5. CIDR 형식으로 작업 10

2장. OPENSTACK 네트워킹 개념 11
2.1. OPENSTACK NETWORKING(NEUTRON) 설치 11
2.2. OPENSTACK NETWORKING 다이어그램 11
2.3. 보안 그룹 12
2.4. OPEN VSWITCH 12
2.5. OPEN VSWITCH의 OPENFLOW 인터페이스 변경 13
2.6. ML2(MODULAR LAYER 2) 네트워킹 14
2.7. ML2 유형 및 메커니즘 드라이버 호환성 15
2.8. ML2/OVN 메커니즘 드라이버의 제한 15
2.9. ML2/OVN을 사용하는 비보안 포트 제한 16
2.10. L2 채우기 드라이버 구성 17
2.11. OPENSTACK NETWORKING 서비스 17
2.12. 프로젝트 및 공급자 네트워크 18
2.13. 계층 2 및 계층 3 네트워킹 19

3장. 일반적인 관리 네트워킹 작업 22
3.1. 네트워크 생성 22
3.2. 네트워크 라우팅 추가 24
3.3. 네트워크 삭제 24
3.4. 모든 리소스 제거 및 프로젝트 삭제 25
3.5. 서브넷 작업 25
3.6. 서브넷 삭제 27
3.7. 라우터 추가 27
3.8. 라우터 삭제 28
3.9. VRRP 패킷 손실을 방지하기 위해 KEEPALIVED 튜닝 28
3.10. 인터페이스 추가 29
3.11. 인터페이스 삭제 29
3.12. 유동 IP 풀 생성 30
3.13. 특정 유동 IP 할당 31
3.14. 고급 네트워크 생성 32
3.15. 임의의 유동 IP 할당 32
3.16. 여러 유동 IP 풀 생성 34
3.17. 물리적 네트워크 브리징 35
3.18. DNS가 포트에 할당하는 이름 지정 36
3.19. 포트에 DHCP 속성 할당 39
3.20. 커널 모듈 로드 40

4장. IP 주소 사용량 계획 43
4.1. VLAN 계획 43
4.2. 네트워크 트래픽 유형 43
4.3. IP 주소 사용 45

4.4. 가상 네트워킹 45
4.5. 네트워크 계획 예 45

5장. OPENSTACK NETWORKING 라우터 포트 검토 47
5.1. 현재 포트 상태 보기 47

6장. 공급자 네트워크 문제 해결 49
6.1. 기본 PING 테스트 49
6.2. VLAN 네트워크 문제 해결 51
6.3. 프로젝트 네트워크에서 문제 해결 52

7장. 물리적 네트워크에 인스턴스 연결 54
7.1. OPENSTACK NETWORKING 토폴로지 개요 54
7.2. OPENSTACK NETWORKING 서비스 배치 54
7.3. 플랫 공급자 네트워크 구성 54
7.4. 플랫 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까? 57
7.5. 플랫 공급자 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결 61
7.6. VLAN 공급자 네트워크 구성 63
7.7. VLAN 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까? 65
7.8. VLAN 프로바이더 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결 69
7.9. ML2/OVS 배포에서 공급자 네트워크의 멀티 캐스트 스누핑 활성화 70
7.10. ML2/OVN 배포에서 멀티 캐스트 활성화 72
7.11. 컴퓨팅 메타데이터 액세스 활성화 74
7.12. 부동 IP 주소 74

8장. OPENSTACK NETWORKING의 물리적 스위치 구성 75
8.1. 물리적 네트워크 환경 계획 75
8.2. CISCO CATALYST 스위치 구성 75
8.3. CISCO NEXUS 스위치 구성 81
8.4. CUMULAS LINUX 스위치 구성 84
8.5. EXOS 스위치 구성 87
8.6. JUNIPER EX 시리즈 스위치 구성 90

9장. 최대 전송 단위(MTU) 설정 구성 96
9.1. MTU 개요 96
9.2. DIRECTOR에서 MTU 설정 구성 96
9.3. 결과 MTU 계산 검토 97

10장. QOS(QUALITY OF SERVICE) 정책 구성 98
10.1. QOS 정책 범위 98
10.2. QOS 정책 및 규칙 생성 및 적용 98
10.3. 송신 트래픽에 대한 DSCP 표시 99
10.4. QOS 정책의 RBAC 101

11장. 브릿지 매핑 구성 103
11.1. 브릿지 매핑 개요 103
11.2. 트래픽 흐름 103
11.3. 브릿지 매핑 구성 103
11.4. OVS에 대한 브리지 매핑 유지 관리 104

12장. VLAN 인식 인스턴스 107
12.1. VLAN 트렁크 및 VLAN 투명 네트워크 107
12.2. TRUNK 플러그인 검토 107
12.3. 트렁크 연결 생성 107
12.4. 트렁크에 하위 포트 추가 109

12.5. 트렁크를 사용하도록 인스턴스 구성 110
12.6. 네트워킹 서비스 RPC 타임아웃 구성 112
12.7. 트렁크 상태 이해 113

13장. RBAC 정책 구성 115
13.1. RBAC 정책 개요 115
13.2. RBAC 정책 생성 115
13.3. RBAC 정책 검토 116
13.4. RBAC 정책 삭제 116
13.5. 외부 네트워크에 대한 RBAC 정책 액세스 권한 부여 117

14장. DVR(DISTRIBUTED VIRTUAL ROUTING) 구성 119
14.1. DVR(DISTRIBUTED VIRTUAL ROUTING) 이해 119
14.2. DVR 개요 120
14.3. DVR 알려진 문제 및 주의 사항 121
14.4. 지원되는 라우팅 아키텍처 122
14.5. ML2 OVS를 사용하여 DVR 배포 122
14.6. 중앙 집중식 라우터를 분산 라우팅으로 마이그레이션 124

15장. NETWORKING LBAASV2 API를 사용하여 부하 분산-AS-A-SERVICE 구성 127
15.1. LBAAS 개요 127
15.2. OPENSTACK NETWORKING 및 LBAAS 토폴로지 128
15.3. LBAAS 구성 129

16장. IPV6를 사용한 프로젝트 네트워킹 132
16.1. IPV6 서브넷 옵션 132
16.2. STATEFUL DHCPV6을 사용하여 IPV6 서브넷 생성 133

17장. 프로젝트 할당량 관리 137
17.1. 프로젝트 할당량 구성 137
17.2. L3 할당량 옵션 137
17.3. 방화벽 할당량 옵션 138
17.4. 보안 그룹 할당량 옵션 138
17.5. 관리 할당량 옵션 138

18장. FWAAS(FIREWALL-AS-A-SERVICE) 구성 140
18.1. FWAAS(FIREWALL-AS-A-SERVICE) 개요 140
18.2. FWAAS(서비스로서의 방화벽) 활성화 141
18.3. FWAAS(서비스로서의 방화벽) 구성 141
18.4. 방화벽 생성 142

19장. 허용된 주소 쌍 구성 143
19.1. 허용되는 주소 쌍의 개요 143
19.2. 포트 생성 및 하나의 주소 쌍 허용 144
19.3. 허용되는 주소 쌍 추가 145

20장. 계층 3 고가용성(HA) 구성 147
20.1. HA(고가용성)가 없는 RHOSP 네트워킹 서비스 147
20.2. 계층 3 고가용성(HA) 개요 147
20.3. 계층 3 HA(고가용성) 페일오버 조건 148
20.4. 3 계층 HA(고가용성)에 대한 프로젝트 고려 사항 148
20.5. RHOSP NETWORKING 서비스에 대한 HA(고가용성) 변경 149
20.6. RHOSP NETWORKING 서비스 노드에서 레이어 3 HA(고가용성) 활성화 149
20.7. HA(고가용성) RHOSP NETWORKING 서비스 노드 구성 검토 152

21장. 태그가 있는 가상 장치 식별 153

21.1. 가상 장치 태그 개요 153
21.2. 가상 장치 태그 지정 153

보다 포괄적 수용을 위한 오픈 소스 용어 교체

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

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 이를 개선하는지 알려주십시오.

DDF(직접 문서 피드백) 기능 사용

특정 문장, 단락 또는 코드 블록에 대한 직접 주석은 **피드백 추가** DDF 기능을 사용하십시오.

1. *다중 페이지 HTML* 형식으로 설명서를 봅니다.
2. 문서 오른쪽 상단에 **Feedback** (피드백) 버튼이 표시되는지 확인합니다.
3. 주석 처리하려는 텍스트 부분을 강조 표시합니다.
4. **피드백 추가**를 클릭합니다.
5. 주석을 사용하여 **Add Feedback** (피드백 추가) 필드를 작성합니다.
6. 선택 사항: 설명서 팀이 문제에 대한 자세한 내용을 문의할 수 있도록 이메일 주소를 추가하십시오.
7. **Submit(제출)**을 클릭합니다.

1장. 네트워킹 개요

OpenStack Networking 서비스(코드명 *neutron*)는 Red Hat OpenStack Platform 13의 소프트웨어 정의 네트워킹 구성 요소입니다.

네트워크 관리자는 소프트웨어 정의 네트워킹(SDN)을 사용하여 낮은 수준의 기능 추상화를 통해 네트워크 서비스를 관리할 수 있습니다. 서버 워크로드가 가상 환경으로 마이그레이션되었지만 데이터를 보내고 받을 네트워크 연결을 찾는 서버일 뿐입니다. SDN은 네트워킹 장비(예: 라우터 및 스위치)를 동일한 가상화 공간으로 이동하여 이러한 요구를 충족합니다. 기본 네트워킹 개념에 이미 익숙한 경우, 연결하는 서버와 마찬가지로 이러한 물리적 네트워킹 개념이 가상화된 것으로 간주하기 쉽습니다.

1.1. 네트워킹 작동 방식

네트워킹이라는 용어는 한 컴퓨터에서 다른 컴퓨터로 정보를 이동하는 동작을 말합니다. 가장 기본적인 수준에서는 NIC(네트워크 인터페이스 카드)가 설치된 두 시스템 간에 케이블이 실행되어 수행됩니다. OSI 네트워킹 모델에서 케이블은 1 계층을 나타냅니다.

이제 두 대 이상의 컴퓨터를 대화에 참여하려면 스위치라는 장치를 추가하여 이 구성을 확장해야 합니다. 엔터프라이즈 스위치에는 추가 머신을 연결할 수 있는 여러 개의 이더넷 포트가 있습니다. 여러 시스템으로 이루어진 네트워크를 LAN(Local Area Network)이라고 합니다.

스위치는 복잡성을 증가하므로 스위치는 OSI 모델의 또 다른 계층인 2 계층을 나타냅니다. 각 NIC에는 하드웨어에 할당된 고유한 MAC 주소 번호가 있으며, 이 번호를 사용하면 동일한 스위치에 연결된 머신이 서로를 찾을 수 있습니다. 스위치는 MAC 주소가 어떤 포트에 연결되어 있는지 목록을 유지 관리하므로 한 컴퓨터에서 다른 컴퓨터에 데이터를 보내려고 할 때 스위치는 둘 다의 위치를 알고 CAM (Content Addressable Memory)의 항목을 조정하여 MAC-address-to-port 매핑을 모니터링합니다.

1.1.1. VLAN

VLAN을 사용하여 동일한 스위치에서 실행되는 컴퓨터의 네트워크 트래픽을 분할할 수 있습니다. 즉, 포트를 다른 네트워크의 구성원으로 구성하여 스위치를 논리적으로 나눌 수 있습니다. 이때 보안상의 이유로 트래픽을 분리하는 데 사용할 수 있는 미니 LAN입니다.

예를 들어 스위치에 총 24개의 포트가 있는 경우 포트 1-6을 VLAN200에 할당하고 포트 7-18을 VLAN201에 할당할 수 있습니다. 결과적으로 VLAN200에 연결된 컴퓨터는 VLAN201의 애플리케이션과 완전히 분리되며, 직접 통신할 수 없으며 원하는 경우 트래픽이 두 개의 물리적 스위치인 것처럼 라우터를 통과해야 합니다. 방화벽은 서로 통신할 수 있는 VLAN을 관리하는 데도 유용할 수 있습니다.

1.2. 두 개의 LAN 연결

두 개의 개별 스위치에서 실행되는 LAN이 두 개 있고 서로 정보를 공유하려는 경우. 이 통신을 구성하는 데는 두 가지 옵션이 있습니다.

- **802.1Q VLAN 태그를 사용하여 두 물리적 스위치에 걸쳐 있는 단일 VLAN을 구성합니다.**
  네트워크 케이블의 한 끝을 한 스위치의 포트에 연결하고 다른 스위치의 포트에 다른 스위치의 포트에 연결한 다음 이러한 포트를 802.1Q 태그 포트(트렁크 포트라고도 함)로 구성해야 합니다. 이 두 스위치는 하나의 큰 논리 스위치 역할을 하며 연결된 컴퓨터는 서로 찾을 수 있습니다.

  이 옵션의 단점은 확장성입니다. 오버헤드가 문제가 될 때까지만 Daisy-chain을 제한된 수의 스위치를 만들 수 있습니다.

- **라우터를 확보하고 케이블을 사용하여 각 스위치에 연결합니다.**
  라우터는 두 스위치 모두에 구성된 네트워크를 인식합니다. 스위치에 연결된 각 케이블 끝에는 해당 네트워크의 기본 게이트웨이라고 하는 IP 주소가 수신됩니다. 기본 게이트웨이는 대상 시스템이 소스 시스템과 동일한 LAN에 있지 않은 것이 명확할 때 트래픽이 전송되는 대상을 정의합니

다. 기본 게이트웨이를 설정함으로써 각 컴퓨터는 목적지에 대한 특정 정보를 모를 필요 없이 다른 컴퓨터에 트래픽을 보낼 수 있습니다. 각 컴퓨터는 트래픽을 기본 게이트웨이로 보내고 라우터는 트래픽을 수신하는 대상 컴퓨터를 결정합니다. 라우팅은 OSI 모델의 계층 3에서 작동하며 IP 주소 및 서브넷과 같은 친숙한 개념이 작동하는 위치입니다.

1.2.1. 방화벽

방화벽은 7 계층(실제 콘텐츠 검사용)을 포함하여 여러 OSI 계층의 트래픽을 필터링할 수 있습니다. 방화벽은 종종 라우터와 동일한 네트워크 세그먼트에 있으며, 여기에서는 모든 네트워크 간에 이동하는 트래픽을 관리합니다. 방화벽은 네트워크에 입력할 수 있는 트래픽을 규정하는 사전 정의된 규칙 세트를 나타냅니다. 이러한 규칙은 매우 세분화될 수 있습니다. 예를 들면 다음과 같습니다.

"VLAN **200의 서버는 VLAN 201** 의 컴퓨터만, 목요일 오후에만 통신할 수 있으며, 암호화된 웹 트래픽(HTTPS)을 한 방향으로 보내는 경우에만 가능합니다.".

이러한 규칙을 적용하기 위해 일부 방화벽은 계층에서 dig 패킷 (DPI)을 수행하여 패킷의 내용을 검사하여 패킷이 합법적인지 확인합니다. 해커는 트래픽 마스커레이드를 지원하지 않는 것으로 하여 데이터를 위반할 수 있습니다. DPI는 위협을 완화하는 데 사용할 수 있는 수단 중 하나입니다.

1.3. RED HAT OPENSTACK NETWORK 흐름 매트릭스

네트워크 흐름 매트릭스는 OpenStack 서비스로의 흐름을 설명하는 쉼표로 구분된 값(CSV) 파일입니다.

알림: 네트워크 흐름 매트릭스는 일반적인 트래픽 흐름을 설명합니다. 가능한 모든 흐름에 대해서는 설명하지 않습니다. 이 매트릭스에 설명되지 않은 일부 흐름은 작동에 매우 중요할 수 있습니다. 예를 들어 모든 트래픽을 차단하고 여기에 설명된 흐름만 선택적으로 여는 경우 의도치 않게 필요한 흐름을 차단할 수 있습니다. 이로 인해 문제를 해결하기 어려울 수 있습니다.

매트릭스는 다음 열의 흐름을 설명합니다.

Service

OpenStack 서비스.

프로토콜

전송 프로토콜.

디스트. 포트

대상 포트.

소스 오브젝트

데이터 소스.

디스트. 개체

데이터 대상.

소스/대상 쌍

유효한 소스 및 대상 쌍.

디스트. 네트워크

대상 네트워크.

ServiceNetMap Parent

각 서비스에 사용되는 네트워크 유형을 결정합니다.

트래픽 설명

트래픽 흐름에 대한 참고 사항입니다.

다음 위치에서 네트워크 흐름 매트릭스 파일을 다운로드합니다.

Red Hat OpenStack Network 흐름.

1.4. OPENSTACK NETWORKING(NEUTRON) 작업

이러한 네트워킹 개념은 소프트웨어 정의 네트워킹(SDN)이라고 하는 OpenStack에 적용됩니다. OpenStack Networking(neutron) 구성 요소는 가상 네트워킹 기능을 위한 API를 제공하며 스위치, 라우터 및 방화벽을 포함합니다. 가상 네트워크 인프라를 사용하면 인스턴스가 실제 네트워크를 사용하여 외부에서 서로 통신할 수 있습니다. Open vSwitch 브리지는 인스턴스에 가상 포트를 할당하며, 수신 및 발신 트래픽을 위해 네트워크 인프라 전체에서 실제 네트워크에 확장할 수 있습니다.

1.5. CIDR 형식으로 작업

일반적으로 IP 주소는 서브넷 블록에 먼저 할당됩니다. 예를 들어 서브넷 마스크가 **255.555.255.0** 인 IP 주소 범위 **192.168.100.0 - 192.168.100.255** 는 **254** IP 주소를 허용합니다(첫 번째 및 마지막 주소는 예약됨).

이러한 서브넷은 다음과 같은 여러 방법으로 표시할 수 있습니다.

- **일반적인 사용법:**
  서브넷 주소는 일반적으로 서브넷 마스크와 함께 네트워크 주소를 사용하여 표시됩니다.
  - 네트워크 주소: *192.168.100.0*
  - 서브넷 마스크: *255.255.255.0*
- **CIDR 형식:**
  서브넷 마스크는 총 활성 비트 수로 단축됩니다.

  예를 들어, **192.168.100.0/24** 에서 **/24** 는 **255.255.255.0** 의 단축된 표현이며 바이너리로 변환할 때 플립된 총 비트 수입니다.

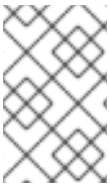
  또한 CIDR 형식은 **NETMASK** 값 대신 **ifcfg-xxx** 스크립트에서 사용할 수 있습니다.

  ```
  #NETMASK=255.255.255.0
  PREFIX=24
  ```

2장. OPENSTACK 네트워킹 개념

OpenStack Networking에는 라우팅, DHCP 및 메타데이터와 같은 핵심 서비스를 관리하는 시스템 서비스가 있습니다. 이러한 서비스는 물리 서버에 할당된 개념적 역할인 컨트롤러 노드 개념에 포함되어 있습니다.

일반적으로 물리적 서버에는 *네트워크 노드* 의 역할이 할당되며 인스턴스와의 네트워크 트래픽에 대한 3계층 라우팅 관리 작업 전용입니다. OpenStack Networking에서는 이 역할을 수행하는 여러 물리적 호스트가 있을 수 있으므로 하드웨어 장애 시 중복 서비스를 허용할 수 있습니다. 자세한 내용은 *계층 3 고가용성* 장을 참조하십시오.



참고

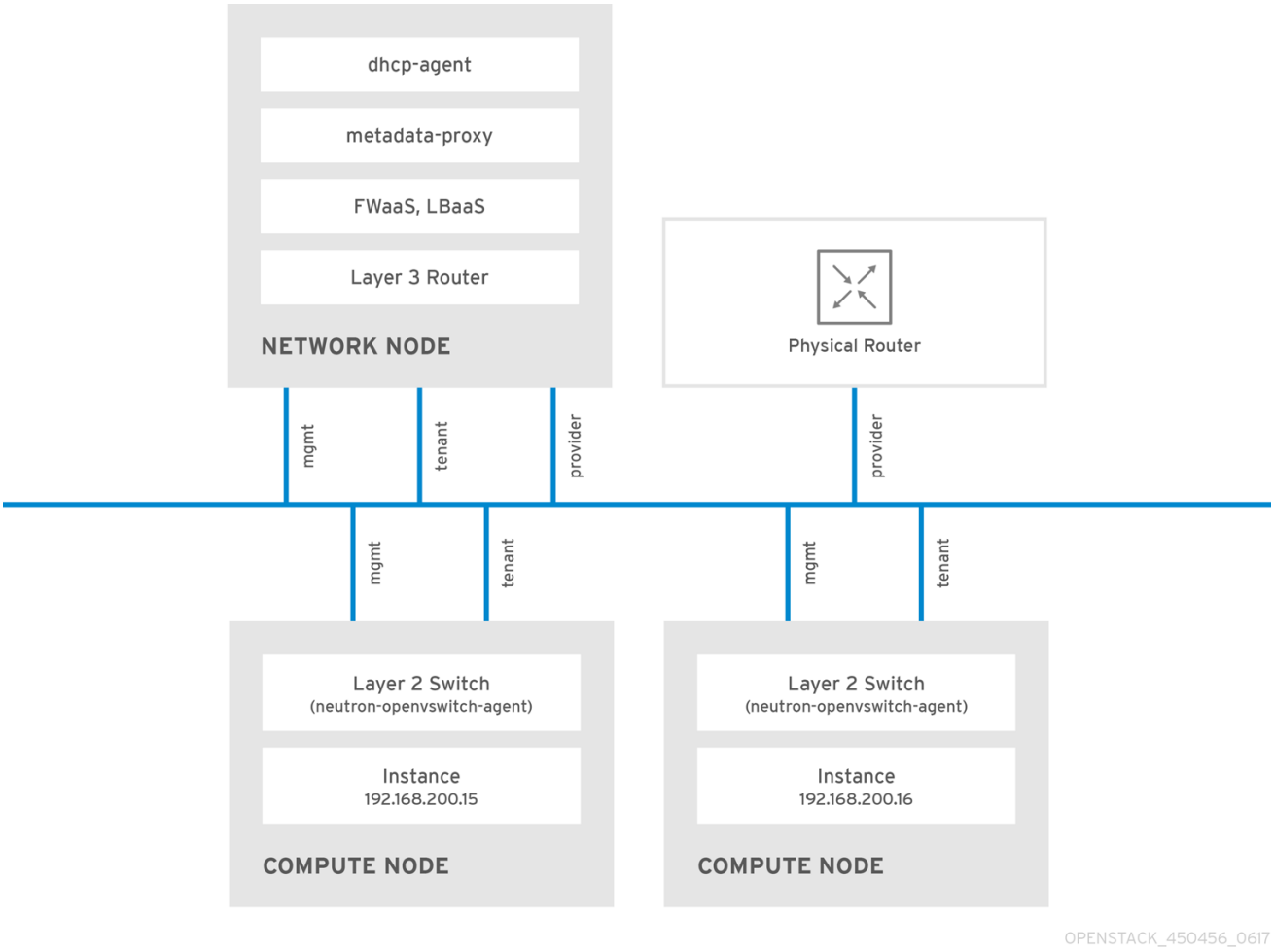
Red Hat OpenStack Platform 11은 구성 가능 역할을 지원하여 네트워크 서비스를 사용자 지정 역할로 분리할 수 있도록 했습니다. 그러나 이 가이드에서는 간편성을 위해 배포에서 기본 controller 역할을 사용한다고 가정합니다.

2.1. OPENSTACK NETWORKING(NEUTRON) 설치

OpenStack Networking 구성 요소는 Red Hat OpenStack Platform director 배포의 일부로 설치됩니다. director 배포에 대한 자세한 내용은 Director 설치 및 사용을 참조하십시오.

2.2. OPENSTACK NETWORKING 다이어그램

이 다이어그램은 계층 3 라우팅 및 DHCP를 수행하고 고급 서비스 로드 밸런싱을 서비스로서의(LBaaS)로 실행하는 전용 OpenStack Networking 노드를 사용하는 샘플 OpenStack Networking 배포를 보여줍니다. 두 개의 컴퓨팅 노드는 Open vSwitch(openvswitch-agent)를 실행하고 각각 두 개의 물리적 네트워크 카드, 즉 프로젝트 트래픽용 두 개의 물리적 네트워크 카드와 관리 연결을 위한 다른 항목이 있습니다. OpenStack Networking 노드에는 특히 공급자 트래픽을 위한 세 번째 네트워크 카드가 있습니다.



2.3. 보안 그룹

보안 그룹 및 규칙은 Neutron 포트에서 보내고 받는 네트워크 트래픽의 유형과 방향을 필터링합니다. 이는 계산 인스턴스에 있는 방화벽 규칙을 보완하는 추가 보안 계층을 제공합니다. 보안 그룹은 하나 이상의 보안 규칙이 있는 컨테이너 오브젝트입니다. 단일 보안 그룹은 여러 계산 인스턴스에 대한 트래픽을 관리할 수 있습니다.

유동 IP 주소, OpenStack Networking LBaaS VIP 및 인스턴스용으로 생성된 포트는 보안 그룹과 연결됩니다. 보안 그룹을 지정하지 않으면 포트가 default 보안 그룹과 연결됩니다. 기본적으로 이 그룹은 인바운드 트래픽을 모두 삭제하고 아웃바운드 트래픽을 모두 허용합니다. 그러나 그룹에는 자체를 가리키는 원격 그룹 ID가 있기 때문에 기본 보안 그룹의 멤버인 인스턴스 간에 트래픽이 이동합니다.

default 보안 그룹의 필터링 동작을 변경하려면 그룹에 보안 규칙을 추가하거나 완전히 새 보안 그룹을 만들 수 있습니다.

2.4. OPEN VSWITCH

OVS(Open vSwitch)는 Linux 소프트웨어 브릿지와 유사한 소프트웨어 정의 네트워킹(SDN) 가상 스위치입니다. OVS는 업계 표준, OpenFlow 및 sFlow를 지원하는 가상화된 네트워크에 전환 서비스를 제공합니다. OVS는 STP, LACP 및 802.1Q VLAN 태깅과 같은 계층 2 기능을 사용하여 물리적 스위치와 통합할 수도 있습니다. Open vSwitch 버전 1.11.0-1.el6 이상에서는 VXLAN 및 GRE를 사용한 터널링도 지원합니다.

네트워크 인터페이스 본딩에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

참고

지정된 브릿지의 멤버가 될 수 있는 단일 인터페이스 또는 단일 본딩만 OVS에서 네트워크 루프 위험을 완화합니다. 여러 개의 본딩이나 인터페이스가 필요한 경우 여러 브릿지를 설정할 수 있습니다.

중요

결합된 인터페이스에서 SR-IOV(단일 루트 I/O 가상화) 사용은 지원되지 않습니다.

2.5. OPEN VSWITCH의 OPENFLOW 인터페이스 변경

Red Hat OpenStack Platform 13에서 네트워킹 서비스(neutron)는 OpenFlow 규칙을 관리하기 위해 Open vSwitch가 의존하는 **python-ryu** 라이브러리에서 잘 작동하지 않는 Python 2.7을 사용합니다.

neutron OVS(Open vSwitch) 에이전트가 OVS에 연결할 때 시간 초과가 발생하는 경우 OpenFlow 인터페이스 및 OVS 데이터베이스 옵션의 값을 변경해야 합니다.

사전 예약

- RHOSP 13에서 Open vSwitch를 사용하고 있습니다.

절차

1. Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.

   예제

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

   작은 정보

   오케스트레이션 서비스(heat)는 templates라는 플랜 집합을 사용하여 환경을 설치하고 구성합니다. heat 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿 파일인 사용자 지정 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다.

2. **parameter_defaults** 아래에 있는 YAML 환경 파일에서 다음 Puppet 변수를 추가합니다.

   ```
   parameter_defaults:
     ExtraConfig:
       neutron::agents::ml2::ovs::of_interface: ovs-ofctl
       neutron::agents::ml2::ovs::ovsdb_interface: vsctl
       ...
   ```

   중요

   단일 콜론(:)과 값 사이에 공백 문자를 추가해야 합니다.

3. **openstack overcloud deploy** 명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.

중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

예제

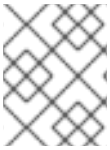
```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-ovs-environment.yaml
```

추가 리소스

- Puppet: *Advanced Overcloud Customization* 가이드에서 개별 노드의 Hieradata 사용자 정의
- *Advanced Overcloud Customization* 가이드의 환경 파일
- *Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함*

2.6. ML2(MODULAR LAYER 2) 네트워킹

ML2는 OpenStack Havana 릴리스에 도입된 OpenStack Networking 핵심 플러그인입니다. ML2 모듈식 설계를 통해 이전 모놀리식 플러그인 모델을 대체하여 혼합 네트워크 기술을 동시에 작동할 수 있습니다. 모놀리식 Open vSwitch 및 Linux Bridge 플러그인은 더 이상 사용되지 않고 제거되었습니다. 이제 ML2 메커니즘 드라이버에서 기능을 구현합니다.



참고

ML2는 기본 메커니즘 드라이버로 구성된 OVN을 사용하는 기본 OpenStack Networking 플러그인입니다.

2.6.1. ML2 뒤의 추론

이전에는 OpenStack Networking 배포에서 구현 시 선택한 플러그인만 사용할 수 있었습니다. 예를 들어 OVS(Open vSwitch) 플러그인을 실행하는 배포는 OVS 플러그인을 독점적으로 사용해야 했습니다. 모놀리식 플러그인은 linuxbridge와 같은 다른 플러그인의 동시 사용을 지원하지 않았습니다. 이러한 제한으로 인해 이기종 요구 사항이 있는 환경의 요구 사항을 충족하기 어려웠습니다.

2.6.2. ML2 네트워크 유형

여러 네트워크 세그먼트 유형을 동시에 작동할 수 있습니다. 또한 이러한 네트워크 세그먼트는 다중 세분화 네트워크에 대한 ML2 지원을 사용하여 상호 연결할 수 있습니다. 포트는 연결을 통해 세그먼트에 자동으로 바인딩됩니다. 특정 세그먼트에 포트를 바인딩할 필요가 없습니다. ML2는 메커니즘 드라이버에 따라 다음 네트워크 세그먼트 유형을 지원합니다.

- 플랫
- GRE
- 로컬
- VLAN

- VXLAN
- Geneve

ml2_conf.ini 파일의 ML2 섹션에서 Type 드라이버를 활성화합니다. 예를 들면 다음과 같습니다.

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan,geneve
```

2.6.3. ML2 메커니즘 드라이버

플러그인은 공통 코드 베이스를 사용하는 메커니즘으로 구현됩니다. 이러한 접근 방식을 통해 코드를 재사용하고 코드 유지 관리 및 테스트와 관련된 복잡성을 없앨 수 있습니다.

기본 메커니즘 드라이버는 OVN입니다. 오케스트레이션 서비스(heat) 매개변수인 **NeutronMechanismDrivers** 를 사용하여 메커니즘 드라이버를 활성화합니다. 다음은 heat 사용자 지정 환경 파일의 예입니다.

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

메커니즘 드라이버를 지정하는 순서는 중요합니다. 이전 예제에서 baremetal 메커니즘 드라이버를 사용하여 포트를 바인딩하려면 **ansible** 보다 **baremetal** 을 지정해야 합니다. 그렇지 않으면 **NeutronMechanismDrivers** 의 값 목록에서 **baremetal** 앞에 있기 때문에 ansible 드라이버는 포트를 바인딩합니다.

추가 리소스

- *Red Hat OpenStack Platform의 구성 요소, 플러그인 및 드라이버 지원의* Neutron
- *Advanced Overcloud Customization* 가이드의 환경 파일
- *Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함*

2.7. ML2 유형 및 메커니즘 드라이버 호환성

메커니즘 드라이버	드라이버 유형				
	플랫	GRE	VLAN	VXLAN	Geneve
OVN	제공됨	제공되지 않음	제공됨	제공되지 않음	제공됨
openvswitch	제공됨	제공됨	제공됨	제공됨	제공되지 않음

2.8. ML2/OVN 메커니즘 드라이버의 제한

다음 표에서는 Red Hat에서 ML2/OVN으로 아직 지원하지 않는 기능에 대해 설명합니다. Red Hat은 향후 Red Hat OpenStack Platform 릴리스에서 이러한 각 기능을 지원할 계획입니다.

또한 이번 RHOSP(Red Hat OpenStack Platform) 릴리스는 ML2/OVS 메커니즘 드라이버에서 ML2/OVN 메커니즘 드라이버로 지원되는 마이그레이션을 제공하지 않습니다. 이 RHPOSP 릴리스는 OpenStack 커뮤니티 마이그레이션 전략을 지원하지 않습니다. 마이그레이션 지원은 향후 RHOSP 릴리스에 계획되어 있습니다.

기능	참고	이 기능 추적
조각화 / 점보 프레임	OVN은 아직 ICMP "패처 지정 필요" 패킷 전송을 지원하지 않습니다. 조각화가 필요한 대규모 ICMP/UDP 패킷은 ML2/OVS 드라이버 구현과 마찬가지로 ML2/OVN에서 작동하지 않습니다. TCP 트래픽은 최대 세그먼트 크기(MSS)를 사용하여 처리합니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1547074 (ovn-network) https://bugzilla.redhat.com/show_bug.cgi?id=1702331 (코어 ovn)
포트 전달	OVN은 포트 전달을 지원하지 않습니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1654608 https://blueprints.launchpad.net/neutron/+spec/port-forwarding
보안 그룹 로깅 API	ML2/OVN은 원격 서버에서 제한된 작업을 실행하거나 제한된 포트에 액세스하려는 인스턴스와 같은 보안 그룹 이벤트를 기록하는 로그 파일을 제공하지 않습니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1619266
멀티 캐스트	ML2/OVN을 통합 브리지로 사용하는 경우 멀티캐스트 트래픽이 브로드캐스트 트래픽으로 처리됩니다. 통합 브리지는 FLOW 모드에서 작동하므로 IGMP 스누핑을 사용할 수 없습니다. 이를 지원하려면 핵심 OVN에서 IGMP 스누핑을 지원해야 합니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1672278
SR-IOV	현재 SR-IOV는 배포된 neutron DHCP 에이전트에서만 작동합니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1666684
OVN DHCP를 사용하여 베어 메탈 머신 프로비저닝	현재 OVN의 기본 제공 DHCP 서버는 베어 메탈 노드를 프로비저닝할 수 없습니다. 프로비저닝 네트워크에 DHCP를 제공할 수 없습니다. 체인부팅 iPXE에는 OVN DHCP 서버에서 지원되지 않는 태그 지정(dnsmasq의**--dhcp-match**)이 필요합니다.	https://bugzilla.redhat.com/show_bug.cgi?id=1622154
OVS_DPDK	OVS_DPDK는 현재 OVN에서 지원되지 않습니다.	

2.9. ML2/OVN을 사용하는 비보안 포트 제한

기본 ML2/OVN 메커니즘 드라이버와 많은 수의 포트를 사용하여 RHOSP(Red Hat Open Stack Platform) 배포에서 포트 보안 플러그인 확장을 비활성화하면 포트에 연결할 수 없습니다.

일부 대규모 ML2/OVN RHSOP 배포에서 ML2/OVN 내부의 흐름 체인 제한은 보안 플러그인이 비활성화된 포트를 대상으로 하는 ARP 요청을 삭제할 수 있습니다.

ML2/OVN에서 지원할 수 있는 실제 논리 스위치 포트 수에 대한 문서화된 최대 제한은 없지만 limit approximateskubelet 포트입니다.

대략적인 제한에 기여하는 속성은 ML2/OVN이 생성하는 OpenFlow 파이프라인에서 다시 제출한 수이며 전체 논리 토폴로지를 변경합니다.

2.10. L2 채우기 드라이버 구성

L2 채우기 드라이버를 사용하면 브로드캐스트, 멀티캐스트 및 유니캐스트 트래픽을 대규모 오버레이 네트워크에서 확장할 수 있습니다. 기본적으로 Open vSwitch GRE 및 VXLAN은 대상 네트워크를 호스팅하지 않는 에이전트를 포함하여 모든 에이전트에 브로드캐스트를 복제합니다. 이 설계에는 중요한 네트워크 및 처리 오버헤드가 수용되어야 합니다. L2 채우기 드라이버에서 도입한 대체 설계는 ARP 확인을 위해 부분 메시와 MAC 학습 트래픽을 구현합니다. 또한 네트워크를 호스팅하는 노드 간에만 특정 네트워크에 대한 터널을 만듭니다. 이 트래픽은 대상 유니캐스트로 캡슐화하여 필요한 에이전트에만 전송됩니다.

L2 채우기 드라이버를 활성화하려면 다음 단계를 완료합니다.

1. 메커니즘 드라이버 목록에 추가하여 L2 채우기 드라이버를 활성화합니다. 또한 GRE, VXLAN 또는 둘 다 하나 이상의 터널링 드라이버를 활성화해야 합니다. ml2_conf.ini 파일에 적절한 구성 옵션을 추가합니다.

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,linuxbridge,l2population
```

참고

Neutron의 Linux Bridge ML2 드라이버 및 에이전트는 Red Hat OpenStack Platform 11에서 더 이상 사용되지 않습니다. Open vSwitch(OVS) 플러그인 OpenStack Platform director의 기본 설정으로, Red Hat에서 일반적으로 사용하는 것이 좋습니다.

2. openvswitch_agent.ini 파일에서 L2 채우기를 활성화합니다. L2 에이전트가 포함된 각 노드에서 활성화합니다.

```
[agent]
l2_population = True
```

참고

ARP 응답 흐름을 설치하려면 **arp_responder** 플래그를 구성합니다.

```
[agent]
l2_population = True
arp_responder = True
```

2.11. OPENSTACK NETWORKING 서비스

기본적으로 Red Hat OpenStack Platform에는 ML2 및 Open vSwitch 플러그인과 통합되는 구성 요소가 포함되어 배포에 네트워킹 기능을 제공합니다.

2.11.1. L3 에이전트

L3 에이전트는 **openstack-neutron** 패키지의 일부입니다. 네트워크 네임스페이스를 사용하여 각 프로젝트에 트래픽을 지시하고 계층 2 네트워크의 게이트웨이 서비스를 제공하는 자체 격리 계층 3 라우터를 제공합니다. L3 에이전트는 이러한 라우터 관리를 지원합니다. L3 에이전트를 호스팅하는 노드에는 외부 네트워크에 연결된 네트워크 인터페이스에 수동으로 구성된 IP 주소가 없어야 합니다. 대신 OpenStack Networking에서 사용할 수 있는 외부 네트워크의 IP 주소 범위가 있어야 합니다. Neutron은 내부 및 외부 네트워크 간 링크를 제공하는 라우터에 이러한 IP 주소를 할당합니다. 선택한 IP 범위는 각 유동 IP뿐만 아니라 배포의 각 라우터에 고유한 IP 주소를 제공할 수 있을 만큼 충분히 커야 합니다.

2.11.2. DHCP 에이전트

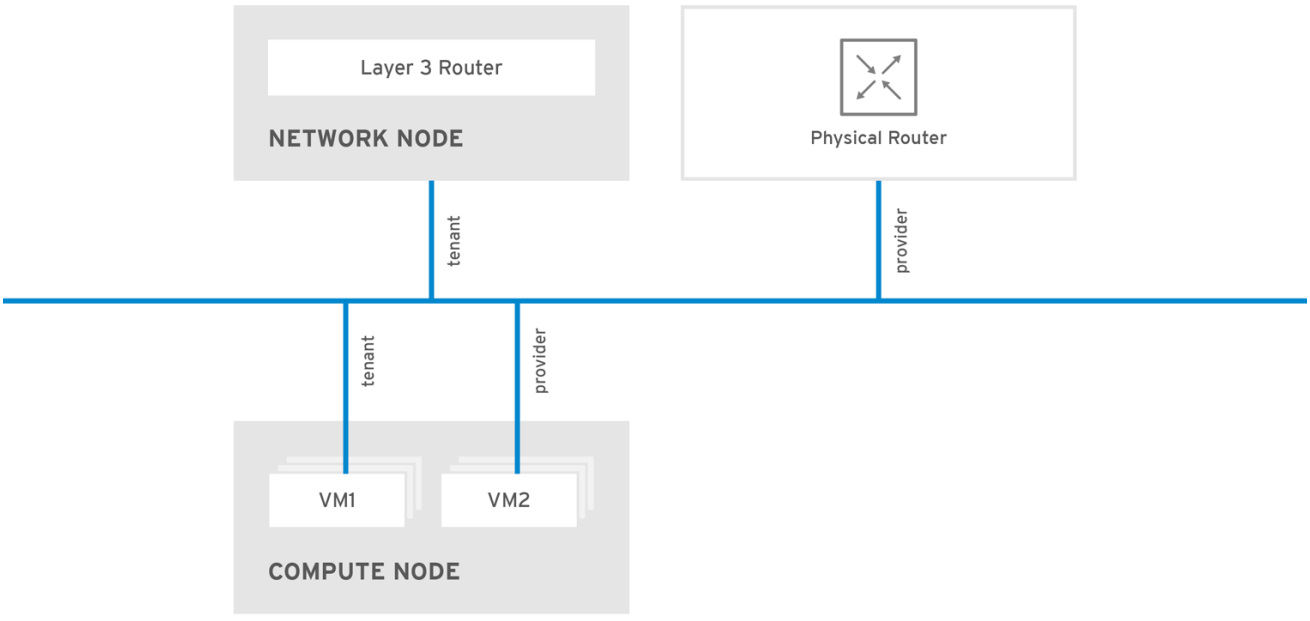
OpenStack Networking DHCP 에이전트는 DHCP 서버 역할을 하기 위해 각 프로젝트 서브넷에 대해 생성되는 네트워크 네임스페이스를 관리합니다. 각 네임스페이스는 네트워크의 가상 시스템에 IP 주소를 할당할 수 있는 dnsmasq 프로세스를 실행합니다. 서브넷이 생성될 때 에이전트가 활성화되어 실행 중인 경우 기본적으로 해당 서브넷에는 DHCP가 활성화됩니다.

2.11.3. Open vSwitch 에이전트

OVS(Open vSwitch) neutron 플러그인은 각 노드에서 실행되고 OVS 브리지를 관리하는 자체 에이전트를 사용합니다. ML2 플러그인은 전용 에이전트와 통합되어 L2 네트워크를 관리합니다. 기본적으로 Red Hat OpenStack Platform은 OVS 브리지를 사용하여 오버레이 네트워크를 구축하는 **ovs-agent** 를 사용합니다.

2.12. 프로젝트 및 공급자 네트워크

다음 다이어그램에서는 프로젝트 및 프로바이더 네트워크 유형에 대한 개요를 보여주며, 전체 OpenStack Networking 토폴로지에서 상호 작용하는 방법을 보여줍니다.



2.12.1. 프로젝트 네트워크

사용자는 프로젝트 내에서 연결을 위해 프로젝트 네트워크를 생성합니다. 프로젝트 네트워크는 기본적으로 완전히 격리되며 다른 프로젝트와 공유되지 않습니다. OpenStack Networking에서는 다음과 같은 다양한 프로젝트 네트워크 유형을 지원합니다.

- **플랫** - 모든 인스턴스가 동일한 네트워크에 있으며 호스트와 공유할 수도 있습니다. VLAN 태그 지정 또는 기타 네트워크 분리가 수행되지 않습니다.

- **VLAN** - OpenStack Networking을 사용하면 실제 네트워크에 있는 VLAN ID(802.1Q 태그)를 사용하여 여러 프로바이더 또는 프로젝트 네트워크를 만들 수 있습니다. 이렇게 하면 인스턴스가 환경 전반에서 서로 통신할 수 있습니다. 또한 동일한 계층 2 VLAN에서 전용 서버, 방화벽, 로드 밸런서 및 기타 네트워크 인프라와 통신할 수 있습니다.

- **VXLAN 및 GRE 터널** - VXLAN 및 GRE는 네트워크 오버레이를 사용하여 인스턴스 간 개인 통신을 지원합니다. GRE 또는 VXLAN 프로젝트 네트워크 외부에서 트래픽을 활성화하려면 OpenStack Networking 라우터가 필요합니다. 또한 라우터는 인터넷을 포함하여 외부 네트워크와 직접 연결된 프로젝트 네트워크를 연결하는 데 필요합니다. 라우터는 유동 IP 주소를 사용하여 외부 네트워크에서 직접 인스턴스에 연결할 수 있는 기능을 제공합니다. VXLAN 및 GRE 유형 드라이버는 ML2/OVS 메커니즘 드라이버와 호환됩니다.

- **GENEVE 터널** - GENEVE는 네트워크 가상화에서 다양한 장치의 변화하는 기능과 요구 사항을 인식하고 수용합니다. 전체 시스템에 대해 규정되어 있지 않고 터널링을 위한 프레임워크를 제공합니다. Geneve는 캡슐화 중에 추가되는 메타데이터의 콘텐츠를 유연하게 정의하고 다양한 가상화 시나리오에 적응하려고 합니다. UDP를 전송 프로토콜로 사용하며 확장 가능한 옵션 헤더를 사용하여 크기가 동적입니다. Geneve는 유니캐스트, 멀티캐스트 및 브로드캐스트를 지원합니다. GENEVE 유형 드라이버는 ML2/OVN 메커니즘 드라이버와 호환됩니다.

추가 리소스

- QoS(Quality of Service) 정책 구성

2.12.2. 공급자 네트워크

OpenStack 관리자가 프로바이더 네트워크를 생성합니다. 프로바이더 네트워크는 데이터 센터의 기존 실제 네트워크에 직접 매핑됩니다. 이 카테고리의 유용한 네트워크 유형에는 flat(태그되지 않음) 및 VLAN(802.1Q 태그)이 포함됩니다. 네트워크 생성 프로세스의 일부로 프로젝트 간에 공급자 네트워크를 공유할 수도 있습니다.

추가 리소스

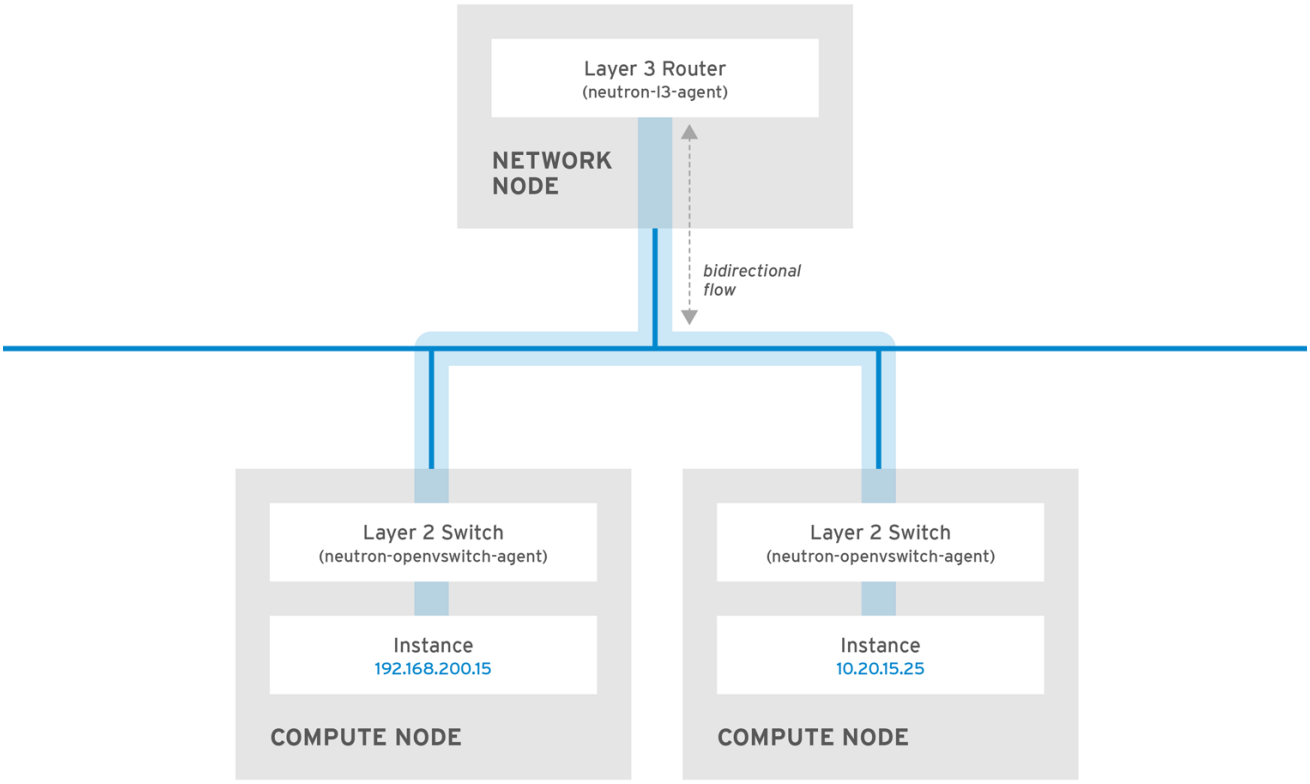
- 플랫 공급자 네트워크 구성

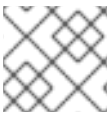
- VLAN 공급자 네트워크 구성

2.13. 계층 2 및 계층 3 네트워킹

가상 네트워크를 설계할 때 대부분의 트래픽이 전송되는 위치를 예상합니다. 네트워크 트래픽은 여러 논리적 네트워크 사이가 아니라 동일한 논리적 네트워크 내에서 더 빠르게 이동합니다. 이는 논리적 네트워크(다른 서브넷 사용) 간 트래픽이 라우터를 통과해야 하므로 대기 시간이 늘어납니다.

별도의 VLAN의 인스턴스 간 네트워크 트래픽 흐름이 있는 아래 다이어그램을 고려하십시오.





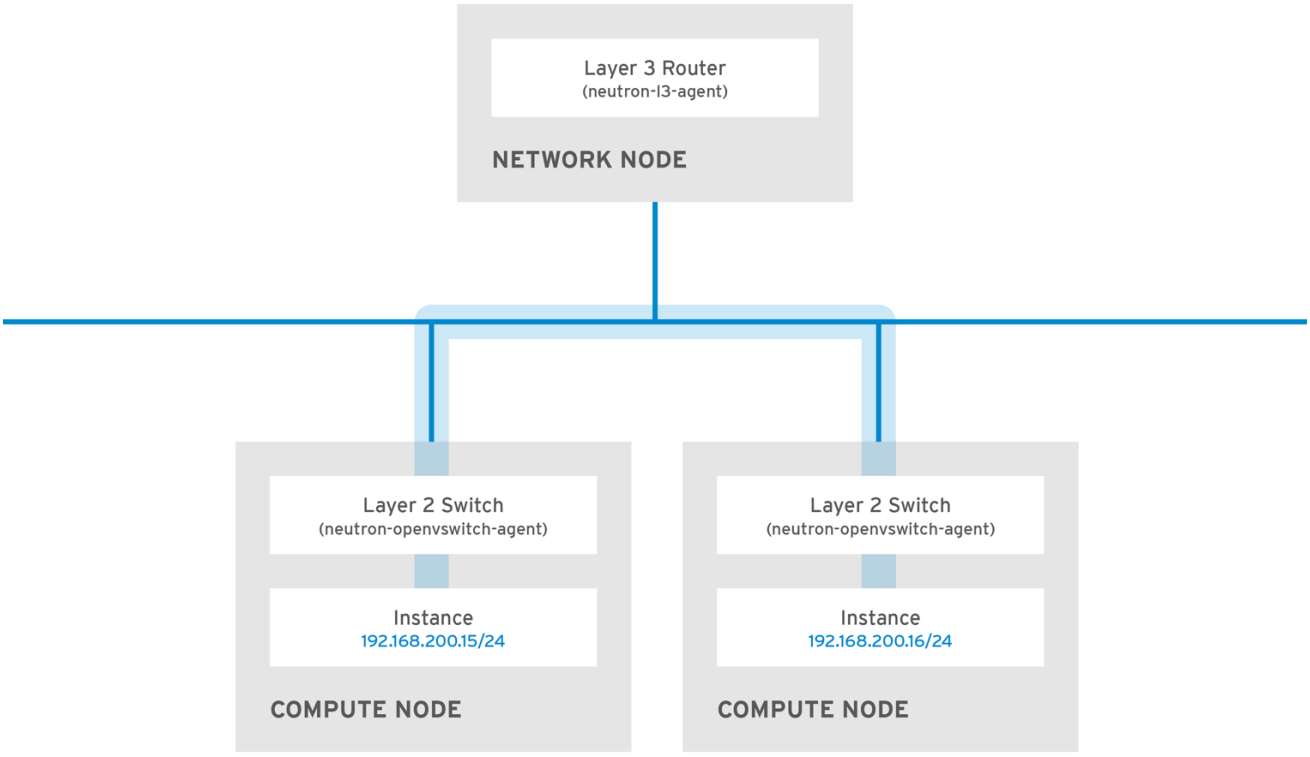
참고

고성능 하드웨어 라우터도 이 구성에 대기 시간을 추가합니다.

2.13.1. 가능한 경우 스위칭 사용

낮은 수준의 네트워크(계층 2)에서 스위칭이 발생하기 때문에 계층 3에서 발생하는 라우팅보다 더 빠르게 작동할 수 있습니다. 자주 통신하는 시스템 간에 최대한 홉을 설계합니다.

예를 들어 다음 다이어그램은 두 개의 물리적 노드를 포괄하는 전환된 네트워크를 보여주므로 두 인스턴스가 먼저 탐색에 라우터를 사용하지 않고 직접 통신할 수 있습니다. 이제 인스턴스가 동일한 논리적 네트워크에 있음을 나타내기 위해 동일한 서브넷을 공유합니다.



별도의 노드의 인스턴스가 동일한 논리적 네트워크에 있는 것처럼 통신할 수 있도록 하려면 VXLAN 또는 GRE와 같은 캡슐화 터널을 사용합니다. 터널 헤더에 필요한 추가 비트를 수용하기 위해 엔드 투 엔드에서 MTU 크기를 조정하는 것이 좋습니다. 그렇지 않으면 조각화로 인해 네트워크 성능에 부정적인 영향을 줄 수 있습니다. 자세한 내용은 *MTU 설정* 구성을 참조하십시오.

VXLAN 오프로드 기능이 있는 지원 하드웨어를 사용하여 VXLAN 터널링의 성능을 더욱 향상시킬 수 있습니다.

추가 리소스

- https://access.redhat.com/articles/1390483

3장. 일반적인 관리 네트워킹 작업

OpenStack Networking(neutron)은 Red Hat OpenStack Platform의 소프트웨어 정의 네트워킹 구성 요소입니다. 가상 네트워크 인프라를 사용하면 인스턴스와 실제 외부 네트워크 간의 연결이 가능합니다.

이 섹션에서는 Red Hat OpenStack Platform 배포에 적합한 서브넷 및 라우터 추가 및 제거와 같은 일반적인 관리 작업에 대해 설명합니다.

3.1. 네트워크 생성

인스턴스가 서로 통신하고 DHCP를 사용하여 IP 주소를 수신할 수 있도록 네트워크를 만듭니다. Red Hat OpenStack Platform 배포의 외부 네트워크와 또는 물리적 네트워크와 같은 다른 네트워크와 네트워크를 통합할 수도 있습니다. 이 통합을 통해 인스턴스에서 외부 시스템과 통신할 수 있습니다. 자세한 내용은 *실제 네트워크 브리지를* 참조하십시오.

네트워크를 만들 때 네트워크에서 여러 서브넷을 호스팅할 수 있다는 것을 알아야 합니다. 이 기능은 동일한 네트워크에서 서로 다른 시스템을 호스팅하고 시스템 간의 격리 측정값을 선호하는 경우에 유용합니다. 예를 들어 한 서브넷에 웹 서버 트래픽만 있는 반면 데이터베이스 트래픽은 다른 서브넷을 트래버스하도록 지정할 수 있습니다. 서브넷은 서로 격리되며 다른 서브넷과 통신하려는 인스턴스에는 라우터에서 보내는 트래픽이 있어야 합니다. 라우팅이 필요하지 않도록 동일한 서브넷에 많은 트래픽이 필요한 시스템을 배치하고 후속 대기 시간과 부하를 피할 수 있습니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 네트워크를 선택합니다**
2. **+Create Network** (+네트워크 만들기)를 클릭하고 다음 값을 지정합니다.

필드	설명
네트워크 이름	네트워크에서 수행할 역할에 따라 설명적인 이름입니다. 네트워크를 외부 VLAN과 통합하는 경우 VLAN ID 번호를 이름에 추가하는 것이 좋습니다. 예를 들어, **webservers_122** 에서 이 서브넷에서 HTTP 웹 서버를 호스팅하는 경우 VLAN 태그가 **122** 입니다. 또는 네트워크 트래픽을 비공개로 유지하고 네트워크를 외부 네트워크와 통합하지 않으려는 **경우 internal** 전용을 사용할 수 있습니다.
관리 상태	네트워크를 즉시 사용할 수 있는지 여부를 제어합니다. 이 필드를 사용하여 논리적으로 존재하지만 비활성 상태인 Down 상태의 네트워크를 생성합니다. 이 기능은 네트워크를 즉시 프로덕션에 입력하려는 경우 유용합니다.

3. **Next** 단추를 클릭하고 Subnet(**서브넷)** 탭에서 다음 값을 지정합니다.

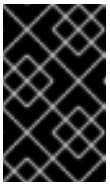
필드	설명
서브넷 만들기	서브넷을 만들지 여부를 결정합니다. 예를 들어 이 네트워크를 네트워크 연결 없이 자리 표시자로 유지하려는 경우 서브넷을 생성하지 않을 수 있습니다.
서브넷 이름	서브넷의 설명이 포함된 이름을 입력합니다.

필드	설명
네트워크 주소	IP 주소 범위와 서브넷 마스크가 포함된 CIDR 형식으로 주소를 하나의 값으로 입력합니다. 주소를 확인하려면 서브넷 마스크에 마스킹된 비트 수를 계산하고 해당 값을 IP 주소 범위에 추가합니다. 예를 들어 서브넷 마스크 255.255.255.0에는 24개의 마스킹된 비트가 있습니다. IPv4 주소 범위 192.168.122.0으로 이 마스크를 사용하려면 주소 192.168.122.0/24를 지정합니다.
IP 버전	유효한 유형이 IPv4 또는 IPv6인 인터넷 프로토콜 버전을 지정합니다. Network Address(네트워크 주소) 필드의 IP 주소 범위는 선택한 모든 버전과 일치해야 합니다.
게이트웨이 IP	기본 게이트웨이에 대한 라우터 인터페이스의 IP 주소입니다. 이 주소는 외부 위치로 향하는 트래픽을 라우팅하기 위한 다음 홉이며, 네트워크 주소 필드에 지정하는 범위 내에 있어야 합니다. 예를 들어 CIDR 네트워크 주소가 192.168.122.0/24인 경우 기본 게이트웨이는 192.168.122.1일 수 있습니다.
게이트웨이 비활성화	는 전달을 비활성화하고 서브넷을 격리합니다.

4. **다음을** 클릭하여 **DHCP** 옵션을 지정합니다.

   - **DHCP 활성화** - 이 서브넷에 DHCP 서비스를 활성화합니다. DHCP를 사용하여 인스턴스에 대한 IP 설정을 자동으로 배포할 수 있습니다.
   - **IPv6 주소** - 구성 모드. IPv6 네트워크를 생성하는 경우 IPv6 주소 및 추가 정보를 할당하는 방법을 지정해야 합니다.
     - **지정된 옵션 없음** - IP 주소를 수동으로 설정하거나 주소 할당에 OpenStack을 인식하지 않는 방법을 사용하려면 이 옵션을 선택합니다.
     - **SLAAC(상태 비저장 주소 자동 구성)** - 인스턴스에서 OpenStack Networking 라우터에서 보낸 R(라우터 알림) 메시지를 기반으로 IPv6 주소를 생성합니다. 이 구성을 사용하여 ra_mode가 slaac로 설정된 OpenStack Networking 서브넷을 만들고 address_mode를 slaac로 설정합니다.
     - **DHCPv6 상태 저장** - 인스턴스에서 IPv6 주소 및 OpenStack Networking DHCPv6 서비스에서 추가 옵션(예: DNS)을 수신합니다. 이 구성을 사용하여 ra_mode가 dhcpv6-stateful로 설정되어 있고 address_mode가 dhcpv6-stateful로 설정된 서브넷을 생성합니다.
     - **DHCPv6 상태 비저장** - 인스턴스에서 OpenStack Networking 라우터에서 전송된 라우터 알림(RA) 메시지를 기반으로 IPv6 주소를 생성합니다. 추가 옵션(예: DNS)은 OpenStack Networking DHCPv6 서비스에서 할당됩니다. 이 구성을 사용하여 ra_mode가 dhcpv6-stateless로 설정된 서브넷을 만들고 address_mode를 dhcpv6-stateless로 설정합니다.
   - **할당 풀** - DHCP가 할당할 IP 주소 범위입니다. 예를 들어 192.168.22.100,192.168.22.100 값은 해당 범위의 모든 주소를 할당에 사용할 수 있는 것으로 간주합니다.

- **DNS 이름 서버** - 네트워크에서 사용할 수 있는 DNS 서버의 IP 주소입니다. DHCP는 이름 확인을 위해 이러한 주소를 인스턴스에 배포합니다.



중요

DNS와 같은 전략적 서비스의 경우 클라우드에 호스팅하지 않는 것이 좋습니다. 예를 들어, 클라우드 호스트 DNS와 클라우드가 작동할 수 없게 되면 DNS를 사용할 수 없으며 클라우드 구성 요소는 서로 조회를 수행할 수 없습니다.

- **호스트 경로** - 정적 호스트 경로. 먼저 CIDR 형식의 대상 네트워크를 지정한 다음 라우팅에 사용할 다음 홉을 지정합니다(예: 192.168.23.0/24, 10.1.31.1). 정적 경로를 인스턴스에 배포해야 하는 경우 이 값을 제공합니다.

5. **생성**을 클릭합니다.
   Networks(네트워크) 탭에서 전체 네트워크를 볼 수 있습니다. 필요에 따라 **Edit(편집)**를 클릭하여 옵션을 변경할 수도 있습니다. 인스턴스를 만들 때 서브넷을 사용하도록 이제 구성할 수 있으며 지정된 DHCP 옵션이 제공됩니다.

3.2. 네트워크 라우팅 추가

새 네트워크에서 트래픽 라우팅을 허용하려면 서브넷을 기존 가상 라우터에 인터페이스로 추가해야 합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 라우터**를 선택합니다.

2. **Routers** (라우터) 목록에서 가상 라우터 이름을 선택하고 **Add Interface(인터페이스 추가)**를 클릭합니다.
   Subnet(서브넷) 목록에서 새 서브넷의 이름을 선택합니다. 이 필드의 인터페이스에 대한 IP 주소를 선택적으로 지정할 수 있습니다.

3. **Add Interface** (인터페이스 추가)를 클릭합니다.
   이제 네트워크의 인스턴스가 서브넷 외부의 시스템과 통신할 수 있습니다.

3.3. 네트워크 삭제

하우스키핑 또는 폐기 프로세스의 일부로 이전에 만든 네트워크를 삭제해야 하는 경우가 있습니다. 먼저 네트워크를 성공적으로 삭제하기 전에 네트워크가 아직 사용 중인 인터페이스를 제거하거나 분리해야 합니다.

프로젝트의 네트워크를 종속 인터페이스와 함께 삭제하려면 다음 단계를 완료합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 네트워크를 선택합니다**
   대상 네트워크 서브넷과 연결된 모든 라우터 인터페이스를 제거합니다.

   인터페이스를 제거하려면 **Networks** (네트워크) 목록에서 대상 네트워크를 클릭하고 ID 필드를 확인하여 삭제할 네트워크의 ID 번호를 찾습니다. 네트워크와 연결된 모든 서브넷은 **네트워크 ID** 필드에서 이 값을 공유합니다.

2. **Project(프로젝트) > Network > Routers**(네트워크 > 라우터) 로 이동하여 **Routers** (라우터) 목록에서 가상 라우터 이름을 클릭하고 삭제하려는 서브넷에 연결된 인터페이스를 찾습니다.
   게이트웨이 IP로 제공되는 IP 주소로 이 서브넷을 다른 서브넷과 구분할 수 있습니다. 인터페이스의 네트워크 ID가 이전 단계에서 기록한 ID와 일치하는지 확인하여 차이점을 추가로 확인할 수 있습니다.

3. 삭제할 인터페이스에 대해 **Delete Interface** (인터페이스 삭제) 단추를 클릭합니다.

4. **프로젝트 > 네트워크 >**네트워크를 선택하고 네트워크 이름을 클릭합니다.

5. 삭제할 서브넷의 **Delete Subnet(서브넷 삭제)** 단추를 클릭합니다.

참고

이때 서브넷을 제거할 수 없는 경우 인스턴스에서 서브넷을 아직 사용하지 않는지 확인합니다.

6. **프로젝트 > 네트워크 >**네트워크를 선택하고 삭제하려는 네트워크를 선택합니다.

7. **Delete Networks(네트워크 삭제)**를 클릭합니다.

3.4. 모든 리소스 제거 및 프로젝트 삭제

openstack project purge 명령을 사용하여 특정 프로젝트에 속하는 모든 리소스와 프로젝트 삭제도 삭제합니다.

예를 들어 **test-project** 프로젝트의 리소스를 제거한 다음 프로젝트를 삭제하려면 다음 명령을 실행합니다.

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

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

3.5. 서브넷 작업

서브넷을 사용하여 인스턴스에 네트워크 연결 권한을 부여합니다. 각 인스턴스는 인스턴스 생성 프로세스의 일부로 서브넷에 할당되므로 연결 요구 사항을 가장 잘 수용할 수 있도록 인스턴스를 적절하게 배치하는 것이 중요합니다.

기존 네트워크에서만 서브넷을 만들 수 있습니다. OpenStack Networking의 프로젝트 네트워크는 여러 서브넷을 호스팅할 수 있습니다. 이 기능은 동일한 네트워크에서 서로 다른 시스템을 호스팅하고 시스템 간의 격리 측정값을 선호하는 경우에 유용합니다.

예를 들어 하나의 서브넷에 웹 서버 트래픽만 있는 반면 데이터베이스 트래픽은 다른 서브넷을 트래버스하도록 지정할 수 있습니다.

서브넷은 서로 격리되며 다른 서브넷과 통신하려는 인스턴스에는 라우터에서 보내는 트래픽이 있어야 합니다. 따라서 서로 많은 트래픽이 필요한 동일한 서브넷에서 시스템을 그룹화하여 네트워크 대기 시간을 줄이고 부하를 줄일 수 있습니다.

3.5.1. 서브넷 생성

서브넷을 만들려면 다음 단계를 따르십시오.

1. 대시보드에서 **Project > Network > Networks(네트워크 > 네트워크)** 를 선택하고 Networks(**네트워크**) 보기에서 네트워크 이름을 클릭합니다.

2. **Create Subnet(서브넷 만들기)** 을 클릭하고 다음 값을 지정합니다.

필드	설명
서브넷 이름	설명이 포함된 서브넷 이름.
네트워크 주소	IP 주소 범위와 서브넷 마스크를 하나의 값으로 포함하는 CIDR 형식의 주소입니다. CIDR 주소를 확인하려면 서브넷 마스크에 마스킹된 비트 수를 계산하고 해당 값을 IP 주소 범위에 추가합니다. 예를 들어 서브넷 마스크 255.255.255.0에는 24개의 마스킹된 비트가 있습니다. IPv4 주소 범위 192.168.122.0으로 이 마스크를 사용하려면 주소 192.168.122.0/24를 지정합니다.
IP 버전	인터넷 프로토콜 버전입니다. 여기서 유효한 유형은 IPv4 또는 IPv6입니다. Network Address(네트워크 주소) 필드의 IP 주소 범위는 선택한 모든 프로토콜 버전과 일치해야 합니다.
게이트웨이 IP	기본 게이트웨이에 대한 라우터 인터페이스의 IP 주소입니다. 이 주소는 외부 위치로 향하는 트래픽을 라우팅하기 위한 다음 홉이며, 네트워크 주소 필드에 지정하는 범위 내에 있어야 합니다. 예를 들어 CIDR 네트워크 주소가 192.168.122.0/24인 경우 기본 게이트웨이는 192.168.122.1일 수 있습니다.
게이트웨이 비활성화	는 전달을 비활성화하고 서브넷을 격리합니다.

3. **다음을** 클릭하여 **DHCP** 옵션을 지정합니다.

   - **DHCP 활성화** - 이 서브넷에 DHCP 서비스를 활성화합니다. DHCP를 사용하여 인스턴스에 대한 IP 설정을 자동으로 배포할 수 있습니다.

   - **IPv6 주소** - 구성 모드. IPv6 네트워크를 생성하는 경우 IPv6 주소 및 추가 정보를 할당하는 방법을 지정해야 합니다.

     - **지정된 옵션 없음** - IP 주소를 수동으로 설정하거나 주소 할당에 OpenStack을 인식하지 않는 방법을 사용하려면 이 옵션을 선택합니다.

     - **SLAAC(상태 비저장 주소 자동 구성)** - 인스턴스에서 OpenStack Networking 라우터에서 보낸 R(라우터 알림) 메시지를 기반으로 IPv6 주소를 생성합니다. 이 구성을 사용하여 ra_mode가 slaac로 설정된 OpenStack Networking 서브넷을 만들고 address_mode를 slaac로 설정합니다.

     - **DHCPv6 상태 저장** - 인스턴스에서 IPv6 주소 및 OpenStack Networking DHCPv6 서비스에서 추가 옵션(예: DNS)을 수신합니다. 이 구성을 사용하여 ra_mode가 dhcpv6-stateful로 설정되어 있고 address_mode가 dhcpv6-stateful로 설정된 서브넷을 생성합니다.

     - **DHCPv6 상태 비저장** - 인스턴스에서 OpenStack Networking 라우터에서 전송된 라우터

알림(RA) 메시지를 기반으로 IPv6 주소를 생성합니다. 추가 옵션(예: DNS)은 OpenStack Networking DHCPv6 서비스에서 할당됩니다. 이 구성을 사용하여 ra_mode가 dhcpv6-stateless로 설정된 서브넷을 만들고 address_mode를 dhcpv6-stateless로 설정합니다.

- **할당 풀** - DHCP가 할당할 IP 주소 범위입니다. 예를 들어 192.168.22.100,192.168.22.100 값은 해당 범위의 모든 주소를 할당에 사용할 수 있는 것으로 간주합니다.
- **DNS 이름 서버** - 네트워크에서 사용할 수 있는 DNS 서버의 IP 주소입니다. DHCP는 이름 확인을 위해 이러한 주소를 인스턴스에 배포합니다.
- **호스트 경로** - 정적 호스트 경로. 먼저 CIDR 형식의 대상 네트워크를 지정한 다음 라우팅에 사용할 다음 홉을 지정합니다(예: 192.168.23.0/24, 10.1.31.1). 정적 경로를 인스턴스에 배포해야 하는 경우 이 값을 제공합니다.

4. **생성**을 클릭합니다.
   Subnets(서브넷) 목록에서 서브넷을 볼 수 있습니다. 필요에 따라 **Edit(편집)**를 클릭하여 옵션을 변경할 수도 있습니다. 인스턴스를 만들 때 서브넷을 사용하도록 이제 구성할 수 있으며 지정된 DHCP 옵션이 제공됩니다.

3.6. 서브넷 삭제

서브넷이 더 이상 사용되지 않는 경우 삭제할 수 있습니다. 그러나 서브넷을 사용하도록 인스턴스가 여전히 구성된 경우 삭제 시도가 실패하고 대시보드에 오류 메시지가 표시됩니다.

네트워크에서 특정 서브넷을 삭제하려면 다음 단계를 완료합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 네트워크를 선택합니다**
2. 네트워크 이름을 클릭합니다.
3. 대상 서브넷을 선택하고 **Delete Subnets(서브넷 삭제)**를 클릭합니다.

3.7. 라우터 추가

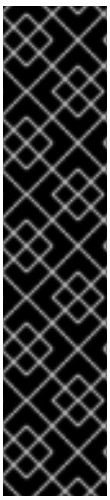
OpenStack Networking은 SDN 기반 가상 라우터를 사용하여 라우팅 서비스를 제공합니다. 라우터는 인스턴스가 실제 네트워크의 서브넷을 포함하여 외부 서브넷과 통신하기 위해 필요합니다. 라우터 및 서브넷은 인터페이스를 사용하여 연결되며, 각 서브넷에는 자체 인터페이스가 필요한 라우터가 있습니다.

라우터의 기본 게이트웨이는 라우터에서 수신한 모든 트래픽의 다음 홉을 정의합니다. 해당 네트워크는 일반적으로 가상 브리지를 사용하여 트래픽을 외부 물리적 네트워크로 라우팅하도록 구성됩니다.

라우터를 생성하려면 다음 단계를 완료합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 라우터를 선택하고Create Router(라우터 만들기)**를 클릭합니다.
2. 새 라우터의 설명이 포함된 이름을 입력하고 **Create router(라우터 만들기)**를 클릭합니다.
3. **Routers** (라우터) 목록에서 새 라우터의 항목 옆에 있는 **Set Gateway** (게이트웨이 설정)를 클릭합니다.
4. **External Network** (외부 네트워크) 목록에서 외부 위치로 향하는 트래픽을 수신하려는 네트워크를 지정합니다.
5. **Set Gateway** (게이트웨이 설정)를 클릭합니다.

라우터를 추가한 후 이 라우터를 사용하여 트래픽을 보내도록 만든 서브넷을 구성해야 합니다. 서브넷과 라우터 간에 인터페이스를 만들어 이 작업을 수행합니다.



중요

서브넷의 기본 경로를 덮어쓸 수 없습니다. 서브넷의 기본 경로가 제거되면 L3 에이전트에서 라우터 네임스페이스의 해당 경로도 자동으로 제거하며 네트워크 트래픽이 연결된 서브넷으로 또는 연결된 서브넷에서 이동할 수 없습니다. 기존 라우터 네임스페이스 경로가 제거된 경우 이 문제를 해결하려면 다음 단계를 수행하십시오.

1. 서브넷의 모든 유동 IP의 연결을 끊습니다.
2. 서브넷에서 라우터를 분리합니다.
3. 라우터를 서브넷에 다시 연결합니다.
4. 모든 유동 IP를 다시 연결합니다.

3.8. 라우터 삭제

연결된 인터페이스가 없는 경우 라우터를 삭제할 수 있습니다.

인터페이스를 제거하고 라우터를 삭제하려면 다음 단계를 완료합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 라우터**를 선택하고 삭제하려는 라우터 이름을 클릭합니다.
2. **내부 인터페이스 유형의 인터페이스를 선택하고Delete Interfaces(인터페이스 삭제)를 클릭합니다**.
3. Routers(라우터) 목록에서 대상 라우터를 선택하고 **Delete Routers(라우터 삭제)**를 클릭합니다.

3.9. VRRP 패킷 손실을 방지하기 위해 KEEPALIVED 튜닝

단일 호스트의 HA(고가용성) 라우터 수가 높은 경우 HA 라우터가 오버라이드되면 VRRP(Virtual Router Redundancy Protocol) 메시지가 IRQ 큐를 오버플로할 수 있습니다. 이 오버플로는 OVS(Open vSwitch)가 이러한 VRRP 메시지를 응답하고 전달하지 못하도록 중지합니다.

VRRP 패킷 과부하를 방지하려면 Networking 서비스(neutron) 에이전트 구성 파일 **l3_agent.conf** 에서 **ha_vrrp_advert_int** 옵션을 사용하여 VRRP 알림 간격을 늘려야 합니다.

절차

1. Overcloud Networking controller 호스트에 stack 사용자로 로그인하고 텍스트 편집기에서 **l3_agent.conf** 를 엽니다.

   예제

   ```
   vi /var/lib/config-data/neutron/etc/neutron/l3_agent.conf
   ```

2. **ha_vrrp_advert_int** 옵션을 찾고 VRRP 광고 간격을 늘립니다. (기본값은 **2** 초입니다.)

   예제

   ```
   [DEFAULT]
   ```

```
ha_vrrp_advert_int = 7
ha_vrrp_garp_master_repeat = 5
ha_vrrp_garp_master_delay = 5
```

설정할 수 있는 적절한 ARP 메시지에 대한 옵션도 있습니다.

ha_vrrp_garp_master_repeat

master 상태로 전환된 후 한 번에 전송할 적절한 ARP 메시지 수입니다. (기본값은 5 메시지입니다.)

ha_vrrp_garp_master_delay

더 낮은 우선 순위 advert가 master 상태에서 수신된 후 좋은 ARP 메시지의 두 번째 집합에 대한 지연이 발생합니다. (기본값은 5초입니다.)

작은 정보

다음에 **openstack overcloud deploy** 명령을 실행하면 이 명령과 같이 수동 변경 사항을 **l3_agent.conf** 로 덮어씁니다. 향후 오버클라우드 업데이트 후 변경 사항을 다시 적용하는 배포 후 스크립트를 작성할 수 있습니다. 스크립트의 다음 문서에 설명된 puppet hieradata 역할 이름을 통합하는 경우 스크립트는 **l3_agent.conf** 변경 사항이 다시 적용되어야 하는 오버클라우드 노드 역할에서만 실행됩니다. 자세한 내용은 구성 후를 참조하십시오. *Advanced Overcloud Customization 가이드에서 All Overcloud* Roles 사용자 지정.

추가 리소스

- 2.1.2 데이터 전달 규칙, *RFC 4541*의 하위 섹션 2

3.10. 인터페이스 추가

라우터가 중간 서브넷 외부의 대상으로 보내는 트래픽을 보낼 수 있도록 인터페이스를 사용하여 서브넷과 라우터를 상호 연결할 수 있습니다.

라우터 인터페이스를 추가하고 새 인터페이스를 서브넷에 연결하려면 다음 단계를 완료합니다.

참고

이 절차에서는 네트워크 토폴로지 기능을 사용합니다. 이 기능을 사용하면 네트워크 관리 작업을 수행하는 동안 모든 가상 라우터 및 네트워크를 그래픽으로 표시할 수 있습니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 네트워크 토폴로지**를 선택합니다.
2. 관리할 라우터를 찾아 마우스 커서를 올려 놓은 다음 **Add Interface(인터페이스 추가)**를 클릭합니다.
3. 라우터에 연결할 서브넷을 지정합니다.
   IP 주소도 지정할 수 있습니다. 이 인터페이스에 대한 ping은 트래픽이 예상대로 라우팅되고 있음을 나타내므로 주소가 테스트 및 문제 해결에 유용합니다.
4. **Add interface** (인터페이스 추가)를 클릭합니다.
   네트워크 토폴로지 다이어그램은 라우터와 서브넷 간의 새 인터페이스 연결을 반영하도록 자동으로 업데이트됩니다.

3.11. 인터페이스 삭제

서브넷의 트래픽을 전달하는 데 라우터가 더 이상 필요하지 않은 경우 서브넷으로 인터페이스를 제거할 수 있습니다.

인터페이스를 삭제하려면 다음 단계를 완료합니다.

1. 대시보드에서 **프로젝트 > 네트워크 > 라우터**를 선택합니다.
2. 삭제할 인터페이스를 호스팅하는 라우터 이름을 클릭합니다.
3. 인터페이스 유형(**내부 인터페이스**)을 선택하고 **Delete Interfaces(인터페이스 삭제)**를 클릭합니다.

3.12. 유동 IP 풀 생성

유동 IP 주소를 사용하여 수신 네트워크 트래픽을 OpenStack 인스턴스로 보낼 수 있습니다. 먼저, 동적으로 인스턴스에 할당할 수 있는 유효한 라우팅 가능한 외부 IP 주소 풀을 정의해야 합니다. OpenStack Networking은 해당 유동 IP로 향하는 들어오는 모든 트래픽을 유동 IP와 연결된 인스턴스로 라우팅합니다.

참고

OpenStack Networking에서는 CIDR 형식의 동일한 IP 범위의 모든 프로젝트(테넌트)에 유동 IP 주소를 할당합니다. 결과적으로 모든 프로젝트에서 모든 유동 IP 서브넷의 유동 IP를 사용할 수 있습니다. 특정 프로젝트의 할당량을 사용하여 이 동작을 관리할 수 있습니다. 예를 들어 Project **C** 및 ProjectB의 할당량을 **0** 으로 설정하는 동안 **ProjectA 및 ProjectB** 의 기본값을 **10** 으로 설정할 수 있습니다.

절차

- 외부 서브넷을 만들 때 유동 IP 할당 풀을 정의할 수도 있습니다.

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

서브넷 호스트에서 유동 IP 주소만 호스팅하는 경우 **openstack subnet create** 명령에서 **--no-dhcp** 옵션을 사용하여 DHCP 할당을 비활성화하는 것이 좋습니다.

예제

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

검증 단계

- 인스턴스에 임의의 유동 IP를 할당하여 풀이 올바르게 구성되었는지 확인할 수 있습니다. (다음의 이후 링크를 참조하십시오.)

추가 리소스

- *명령줄 인터페이스 참조*에서 서브넷 생성
- 임의의 유동 IP 할당

3.13. 특정 유동 IP 할당

VM 인스턴스에 특정 유동 IP 주소를 할당할 수 있습니다.

절차

- **openstack server add floating ip 명령을 사용하여 인스턴스에 유동** IP 주소를 할당합니다.

  예제

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

검증 단계

- **openstack server show** 명령을 사용하여 유동 IP가 인스턴스와 연결되어 있는지 확인합니다.

  예제

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

  샘플 출력

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

```
|                          | 45f76ffced91096196f646b5                |
| volumes_attached         |                                         |
+--------------------------+-----------------------------------------+
```

추가 리소스

- server add floating ip in the *Command Line Interface Reference*
- 명령줄 인터페이스 참조에 서버 표시
- *임의의 유동 IP 할당*

3.14. 고급 네트워크 생성

Admin (관리자) 보기에서 네트워크를 만들 때 관리자가 고급 네트워크 옵션을 사용할 수 있습니다. 이러한 옵션을 사용하여 프로젝트를 지정하고 사용할 네트워크 유형을 정의합니다.

절차

1. *대시보드에서 **Admin > Networks > Create Network > Project**를 선택합니다.*
2. ***Project(프로젝트)** 드롭다운 목록에서 새 네트워크를 호스팅할 프로젝트를 선택합니다.*
3. ***Provider Network Type** 의 옵션을 검토합니다.*
   - ***local** - 트래픽은 로컬 계산 호스트에 유지되며 외부 네트워크와 효과적으로 격리됩니다.*
   - ***플랫** - 트래픽은 단일 네트워크에 유지되며 호스트와 공유할 수도 있습니다. VLAN 태그 지정 또는 기타 네트워크 분리가 수행되지 않습니다.*
   - ***VLAN** - 실제 네트워크에 있는 VLAN에 해당하는 VLAN ID를 사용하여 네트워크를 만듭니다. 이 옵션을 사용하면 인스턴스가 동일한 계층 2 VLAN에서 시스템과 통신할 수 있습니다.*
   - ***GRE** - 여러 노드에 걸쳐 있는 네트워크 오버레이를 사용하여 인스턴스 간 개별 통신을 수행합니다. 오버레이를 송신하는 트래픽의 경로가 지정되어야 합니다.*
   - ***VXLAN** - GRE와 유사하며 네트워크 오버레이를 사용하여 인스턴스 간 개인 통신에 여러 노드를 확장합니다. 오버레이를 송신하는 트래픽의 경로가 지정되어야 합니다.*
4. ***Create Network(네트워크 만들기)**를 클릭합니다.*
   ***Project(프로젝트) Network Topology(네트워크 토폴로지)**를 검토하여 네트워크가 성공적으로 생성되었는지 확인합니다.*

추가 리소스

- *특정 유동 IP 할당*
- *임의의 유동 IP 할당*

3.15. 임의의 유동 IP 할당

외부 IP 주소 풀에서 VM 인스턴스에 유동 IP 주소를 동적으로 할당할 수 있습니다.

사전 요구 사항

- 라우팅 가능한 외부 IP 주소 풀입니다.
  자세한 내용은 3.12 절. "유동 IP 풀 생성" 의 내용을 참조하십시오.

절차

1. 다음 명령을 입력하여 풀에서 유동 IP 주소를 할당합니다. 이 예에서 네트워크의 이름은 **public** 입니다.

   예제

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

   샘플 출력

   다음 예에서 새로 할당된 유동 IP는 **192.0.2.200** 입니다. 인스턴스에 할당할 수 있습니다.

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

2. 다음 명령을 입력하여 인스턴스를 찾습니다.

   ```
   $ openstack server list
   ```

   샘플 출력

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

3. 인스턴스 이름 또는 ID를 유동 IP와 연결합니다.

   예제

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

검증 단계

- 다음 명령을 입력하여 유동 IP가 인스턴스와 연결되어 있는지 확인합니다.

예제

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

샘플 출력

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

추가 리소스

- *명령줄 인터페이스 참조에서 부동 ip create*
- *server add floating ip in the Command Line Interface Reference*
- *명령줄 인터페이스 참조에 서버 표시*
- *부동 IP 풀 생성*

3.16. 여러 유동 IP 풀 생성

OpenStack Networking에서는 각 L3 에이전트에 대해 하나의 유동 IP 풀을 지원합니다. 따라서 추가 유동 IP 풀을 생성하려면 L3 에이전트를 확장해야 합니다.

절차

- 사용자 환경의 L3 에이전트에 대해 **/var/lib/config-data/neutron/neutron.conf** 속성 **handle_internal_only_routers** 가 **True** 로 설정되어 있는지 확인합니다. 이 옵션은 외부가 아닌 라우터만 관리하도록 L3 에이전트를 구성합니다.

추가 리소스

- 부동 IP 풀 생성
- 임의의 유동 IP 할당

3.17. 물리적 네트워크 브리징

가상 네트워크를 실제 네트워크로 연결하여 가상 인스턴스와 연결할 수 있도록 합니다.

이 절차에서 실제 인터페이스의 예로 **eth0** 이 브리지 **br-ex** 에 매핑됩니다. 가상 브리지는 실제 네트워크와 모든 가상 네트워크 간의 중간 역할을 합니다.

결과적으로 **eth0** 을 통과하는 모든 트래픽은 구성된 Open vSwitch를 사용하여 인스턴스에 연결합니다.

물리적 NIC를 가상 Open vSwitch 브리지에 매핑하려면 다음 단계를 완료합니다.

절차

1. 텍스트 편집기에서 **/etc/sysconfig/network-scripts/ifcfg-eth0** 을 열고 사이트의 네트워크에 적합한 값으로 다음 매개변수를 업데이트합니다.
   - IPADDR
   - 넷마스크 게이트웨이
   - DNS1 (이름 서버)
     예를 들면 다음과 같습니다.

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

2. 텍스트 편집기에서 **/etc/sysconfig/network-scripts/ifcfg-br-ex** 를 열고 이전에 eth0에 할당된 IP 주소 값으로 가상 브리지 매개변수를 업데이트합니다.

   ```
   # vi /etc/sysconfig/network-scripts/ifcfg-br-ex
   DEVICE=br-ex
   DEVICETYPE=ovs
   TYPE=OVSBridge
   BOOTPROTO=static
   IPADDR=192.168.120.10
   NETMASK=255.255.255.0
   ```

```
GATEWAY=192.168.120.1
DNS1=192.168.120.1
ONBOOT=yes
```

이제 인스턴스에 유동 IP 주소를 할당하고 실제 네트워크에서 사용할 수 있도록 설정할 수 있습니다.

추가 리소스

- 브릿지 매핑 구성

3.18. DNS가 포트에 할당하는 이름 지정

포트 확장(dns_domain_**ports**)에 대해 RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스(neutron) dns_domain을 활성화할 때 내부 DNS에서 포트에 할당된 이름을 지정할 수 있습니다.

YAML 형식의 환경 파일에서 RHOSP Orchestration(heat) **NeutronPluginExtensions** 매개변수를 선언하여 포트 확장에 대해 dns_domain을 활성화합니다. 해당 매개변수인 **NeutronDnsDomain** 을 사용하여 도메인 이름을 지정하여 기본값인 **openstacklocal** 을 덮어씁니다. 오버클라우드를 재배포한 후 OpenStack 클라이언트 포트 명령, 포트 **세트 또는 포트 생성을 --dns-name** 과 함께 사용하여 포트 이름을 할당할 수 있습니다.

또한 포트 확장의 dns_domain이 활성화되면 계산 서비스에서 VM 인스턴스 부팅 중에 인스턴스의 **hostname** 속성으로 **dns_name** 속성을 자동으로 채웁니다. 부팅 프로세스가 끝나면 dnsmasq는 인스턴스 호스트 이름으로 할당된 포트를 인식합니다.

절차

1. stack 사용자로 언더클라우드에 로그인하고 **stackrc** 파일을 가져와 director 명령줄 툴을 활성화합니다.

   예제

   ```
   $ source ~/stackrc
   ```

2. 사용자 지정 YAML 환경 파일(**my-neutron-environment.yaml**)을 만듭니다.

   참고

   괄호 안의 값은 이 절차의 예제 명령에 사용되는 샘플 값입니다. 이러한 샘플 값을 사이트에 적합한 값으로 바꿉니다.

   예제

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

   작은 정보

   언더클라우드에는 오버클라우드 생성 계획을 구성하는 오케스트레이션 서비스 템플릿 세트가 포함되어 있습니다. 핵심 오케스트레이션 서비스 템플릿 컬렉션의 매개변수와 리소스를 재정의하는 YAML 형식의 파일인 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다. 환경 파일은 필요한 수만큼 추가할 수 있습니다.

3. *환경 파일에서* ***parameter_defaults*** *섹션을 추가합니다. 이 섹션에서 포트 확장자* dns_domain_ ***ports에 대해 dns_domain*** *을 추가합니다.*

   예제

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

   참고

   dns_domain_ports *를 설정하는 경우 배포에서 DNS 통합 확장 기능인* ***dns_domain*** *도 사용하지 않는지 확인합니다. 이러한 확장 기능은 호환되지 않으며 두 확장을 동시에 정의할 수 없습니다.*

4. *또한* ***parameter_defaults*** *섹션에서* ***NeutronDnsDomain*** *매개변수를 사용하여 도메인 이름 (****example.com****)을 추가합니다.*

   예제

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

5. ***openstack overcloud deploy*** *명령을 실행하고 핵심 오케스트레이션 템플릿, 환경 파일 및 이 새 환경 파일을 포함합니다.*

   중요

   후속 환경 파일에 정의된 매개 변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다.

   예제

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

검증 단계

1. *Overcloud에 로그인하고 네트워크(****공용****)에 새 포트(****new_port****)를 만듭니다. 포트에 DNS 이름 (****my_port****)을 할당합니다.*

   예제

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

2. *포트(new****_port****)의 세부 정보를 표시합니다.*

   예제

```
$ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
```

출력 결과

```
+-------------------------+-----------------------------------------------+
| Field                   | Value                                         |
+-------------------------+-----------------------------------------------+
| dns_assignment          | fqdn='my_port.example.com',                   |
|                         | hostname='my_port',                           |
|                         | ip_address='10.65.176.113'                    |
| dns_domain              | example.com                                   |
| dns_name                | my_port                                       |
| name                    | new_port                                      |
+-------------------------+-----------------------------------------------+
```

dns_assignment *에서 포트의 정규화된 도메인 이름(* ***fqdn****) 값에는* ***NeutronDnsDomain*** *을 사용하여 이전에 설정한 DNS 이름(****my_port****) 및 도메인 이름(****example.com****)의 연결이 포함됩니다.*

3. *방금 만든 포트(new****_port)를 사용하여 새 VM 인스턴스(*** ***my_vm****)를 만듭니다.*

   예제

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

4. *포트(new****_port****)의 세부 정보를 표시합니다.*

   예제

   ```
   $ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
   ```

   출력 결과

   ```
   +-------------------------+-----------------------------------------------+
   | Field                   | Value                                         |
   +-------------------------+-----------------------------------------------+
   | dns_assignment          | fqdn='my_vm.example.com',                     |
   |                         | hostname='my_vm',                             |
   |                         | ip_address='10.65.176.113'                    |
   | dns_domain              | example.com                                   |
   | dns_name                | my_vm                                         |
   | name                    | new_port                                      |
   +-------------------------+-----------------------------------------------+
   ```

   계산 서비스는 ***dns_name*** *속성을 원래 값(****my_port****)에서 포트가 연결된 인스턴스 이름(****my_vm****)으로 변경합니다.*

추가 리소스

- *Advanced Overcloud Customization 가이드의 환경 파일*
- *Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함*
- *명령줄 인터페이스 참조의 포트*

- 명령줄 인터페이스 참조에서 서버 생성

3.19. 포트에 DHCP 속성 할당

RHOSP(Red Hat Openstack Plaform) Networking 서비스(neutron) 확장을 사용하여 네트워킹 기능을 추가할 수 있습니다. 추가 DHCP 옵션 확장(**extra_dhcp_opt**)을 사용하여 DHCP 속성으로 DHCP 클라이언트 포트를 구성할 수 있습니다. 예를 들어 **tftp-server,server-ip-address** 또는 **bootfile-name** 과 같은 PXE 부팅 옵션을 DHCP 클라이언트 포트에 추가할 수 있습니다.

extra_dhcp_opt 특성 값은 DHCP 옵션 오브젝트의 배열이며, 각 오브젝트에는 **opt_name** 및 **opt_value** 가 포함되어 있습니다. IPv4는 기본 버전이지만 세 번째 옵션 **ip-version=6** 을 포함하여 IPv6로 변경할 수 있습니다.

VM 인스턴스가 시작되면 RHOSP Networking 서비스는 DHCP 프로토콜을 사용하여 포트 정보를 인스턴스에 제공합니다. 실행 중인 인스턴스에 이미 연결된 포트에 DHCP 정보를 추가하는 경우 인스턴스가 재시작될 때 인스턴스에서 새 DHCP 포트 정보만 사용합니다.

가장 일반적인 DHCP 포트 속성은 **bootfile-name**,**dns-server**,**domain-name**,**mtu**,**server-ip-address**, **tftp-server** 입니다. **opt_name** 에 허용되는 값의 전체 세트는 DHCP 사양을 참조하십시오.

사전 요구 사항

- RHOSP 관리자 권한이 있어야 합니다.

절차

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

```
$ source ~/stackrc
```

3. 사용자 지정 YAML 환경 파일을 생성합니다.

예제

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

4. 환경 파일에는 **parameter_defaults** 키워드가 포함되어야 합니다. 이러한 키워드 아래에 **extra_dhcp_opt** 를 추가 DHCP 옵션 확장을 추가합니다.

예제

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

5. 배포 명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.

> The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

예제

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

검증

1. 자격 증명 파일을 가져옵니다.

   예제

   ```
   $ source ~/overcloudrc
   ```

2. 네트워크(**공용**)에 새 포트(**new_port**)를 만듭니다. DHCP 사양의 유효한 속성을 새 포트에 할당합니다.

   예제

   ```
   $ openstack port create --extra-dhcp-option name=domain-name,value=test.domain --extra-dhcp-option name=ntp-server,value=192.0.2.123 --network public new_port
   ```

3. 포트(new_**port**)의 세부 정보를 표시합니다.

   예제

   ```
   $ openstack port show new_port -c extra_dhcp_opts
   ```

   샘플 출력

   ```
   +-----------------+--------------------------------------------------------------------------+
   | Field           | Value                                                                    |
   +-----------------+--------------------------------------------------------------------------+
   | extra_dhcp_opts | ip_version='4', opt_name='domain-name', opt_value='test.domain'    |
   |                 | ip_version='4', opt_name='ntp-server', opt_value='192.0.2.123'     |
   +-----------------+--------------------------------------------------------------------------+
   ```

추가 리소스

- OVN 지원 DHCP 옵션
- DHCP(Dynamic Host Configuration Protocol) 및 Bootstrap Protocol(BOOTP) 매개 변수
- Advanced Overcloud Customization 가이드의 환경 파일
- Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함
- 명령줄 인터페이스 참조에서 포트 생성
- 명령줄 인터페이스참조에 포트 표시

3.20. 커널 모듈 로드

RHOSP(Red Hat OpenStack Platform)의 일부 기능을 사용하려면 특정 커널 모듈을 로드해야 합니다. 예를 들어 OVS 방화벽 드라이버에서는 두 VM 인스턴스 간 GRE 터널링을 지원하도록 the ***nf_conntrack_proto_gre*** *커널 모듈을 로드해야 합니다.*

특수 오케스트레이션 서비스(heat) 매개변수인 ***ExtraKernelModule을 사용하면*** *heat에서 GRE 터널링과 같은 기능에 필요한 커널 모듈에 대한 구성 정보를 저장할 수 있습니다. 나중에 일반 모듈 관리 중에 이러한 필수 커널 모듈이 로드됩니다.*

절차

1. *Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.*

   예제

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

   작은 정보

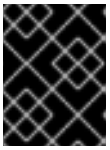
   Heat는 템플릿 이라는 플랜 세트를 사용하여 환경을 설치하고 구성합니다. heat 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿 파일인 사용자 지정 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다.

2. ***parameter_defaults*** *아래의 YAML 환경 파일에서* ***ExtraKernelMod****les를 로드할 모듈 이름으로 설정합니다.*

   예제

   ```
   ComputeParameters:
     ExtraKernelModules:
       nf_conntrack_proto_gre: {}
   ControllerParameters:
     ExtraKernelModules:
       nf_conntrack_proto_gre: {}
   ```

3. ***openstack overcloud deploy*** *명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.*

   중요

   후속 환경 파일에 정의된 매개 변수와 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

   예제

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

검증 단계

- *Heat에 모듈을 올바르게 로드한 경우 컴퓨팅 노드에서 **lsmod** 명령을 실행할 때 출력이 표시됩니다.*

  예제

  ```
  sudo lsmod | grep nf_conntrack_proto_gre
  ```

추가 리소스

- *Advanced Overcloud Customization 가이드의 환경 파일*
- *Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함*

4장. IP 주소 사용량 계획

OpenStack 배포에서는 예상보다 많은 수의 IP 주소를 사용할 수 있습니다. 이 섹션에는 필요한 주소 수량을 올바르게 예상하고 사용자 환경에서 주소가 사용되는 위치를 확인하는 데 대한 정보가 포함되어 있습니다.

4.1. VLAN 계획

Red Hat OpenStack Platform 배포를 계획할 때 개별 IP 주소를 할당하는 여러 서브넷으로 시작합니다. 여러 서브넷을 사용하는 경우 시스템 간에 트래픽을 VLAN으로 분리할 수 있습니다.

예를 들어 관리 또는 API 트래픽이 웹 트래픽을 제공하는 시스템과 동일한 네트워크에 있지 않은 것이 이상적입니다. VLAN 간 트래픽은 트래픽 흐름을 제어하는 방화벽을 구현할 수 있는 라우터를 통해 이동합니다.

배포 시 다양한 유형의 가상 네트워킹 리소스에 대한 트래픽 격리, 고가용성 및 IP 주소 사용률을 포함하는 전체 계획의 일부로 VLAN을 계획해야 합니다.

참고

단일 네트워크의 최대 VLAN 수 또는 네트워크 노드에 대한 하나의 OVS 에이전트는 4094입니다. 최대 VLAN 수가 필요한 경우 여러 개의 프로바이더 네트워크(VXLAN 네트워크)와 네트워크당 하나씩 여러 네트워크 노드를 생성할 수 있습니다. 각 노드에는 최대 4094개의 사설 네트워크를 포함할 수 있습니다.

4.2. 네트워크 트래픽 유형

호스팅하려는 다양한 네트워크 트래픽 유형에 별도의 VLAN을 할당할 수 있습니다. 예를 들어 이러한 각 네트워크 유형에 대해 별도의 VLAN이 있을 수 있습니다. 외부 네트워크만 외부 물리적 네트워크로 라우팅할 수 있어야 합니다. 이번 릴리스에서 director는 DHCP 서비스를 제공합니다.

참고

이 섹션의 모든 OpenStack 배포에 대해 격리된 VLAN이 모두 필요하지는 않습니다. 예를 들어, 클라우드 사용자가 필요에 따라 임시 가상 네트워크를 생성하지 않으면 프로젝트 네트워크가 필요하지 않을 수 있습니다. 각 VM이 다른 물리적 시스템과 동일한 스위치에 직접 연결하려면 Compute 노드를 공급자 네트워크에 직접 연결하고 해당 프로바이더 네트워크를 직접 사용하도록 인스턴스를 구성합니다.

- **프로비저닝 네트워크** - 이 VLAN은 PXE 부팅을 통해 director를 사용하여 새 노드를 배포하는 전용입니다. OpenStack Orchestration(heat)은 Overcloud 베어 메탈 서버에 OpenStack을 설치합니다. 이러한 서버는 실제 네트워크에 연결하여 Undercloud 인프라에서 플랫폼 설치 이미지를 받습니다.
- **내부 API 네트워크** - OpenStack 서비스는 API 통신, RPC 메시지 및 데이터베이스 통신을 포함하여 내부 API 네트워크를 통신에 사용합니다. 또한 이 네트워크는 컨트롤러 노드 간 작동 메시지에도 사용됩니다. IP 주소 할당을 계획할 때 각 API 서비스에 고유한 IP 주소가 필요합니다. 특히 다음 각 서비스에 대한 IP 주소를 계획해야 합니다.
  - vip-msg (ampq)
  - vip-keystone-int
  - av-glance-int

- *vip-cinder-int*
- *vip-nova-int*
- *vip-neutron-int*
- *vip-horizon-int*
- *vip-heat-int*
- *vip-ceilometer-int*
- *vip-swift-int*
- *vip-keystone-pub*
- *vip-glance-pub*
- *vip-cinder-pub*
- *vip-nova-pub*
- *vip-neutron-pub*
- *vip-horizon-pub*
- *vip-heat-pub*
- *vip-ceilometer-pub*
- *vip-swift-pub*

참고

고가용성을 사용하는 경우 Pacemaker는 물리 노드 간에 VIP 주소를 이동합니다.

- ***스토리지*** *- 블록 스토리지, NFS, iSCSI 및 기타 스토리지 서비스. 성능상의 이유로 이 네트워크를 분리하여 물리적 이더넷 링크를 분리합니다.*
- ***스토리지 관리*** *- OpenStack Object Storage(swift)는 이 네트워크를 사용하여 참여한 복제본 노드 간에 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 Ceph와 직접 상호 작용하지 않고 프론트엔드 서비스를 사용하는 스토리지 관리 네트워크를 통해 연결됩니다. RBD 드라이버는 예외입니다. 이 트래픽은 Ceph에 직접 연결됩니다.*
- ***프로젝트 네트워크*** *- Neutron은 VLAN 분리(각 프로젝트 네트워크가 네트워크 VLAN인) 또는 VXLAN 또는 GRE를 사용하여 터널링을 사용하여 각 프로젝트에 자체 네트워크를 제공합니다. 각 프로젝트 네트워크 내에서 네트워크 트래픽이 격리됩니다. 각 프로젝트 네트워크에는 연결된 IP 서브넷이 있으며, 여러 프로젝트 네트워크에서 동일한 주소를 사용할 수 있습니다.*
- ***external*** *- 외부 네트워크는 공용 API 엔드포인트 및 대시보드(horizon)에 대한 연결을 호스팅합니다. SNAT에 이 네트워크를 사용할 수도 있습니다. 프로덕션 배포에서는 일반적으로 유동 IP 주소와 NAT에 별도의 네트워크를 사용하는 것이 일반적입니다.*
- ***프로바이더 네트워크*** *- 프로바이더 네트워크를 사용하여 인스턴스를 기존 네트워크 인프라에 연결합니다. 프로바이더 네트워크를 사용하면 플랫 네트워킹 또는 VLAN 태그를 사용하여 데이터*

센터의 기존 실제 네트워크에 직접 매핑할 수 있습니다. 이렇게 하면 인스턴스에서 OpenStack Networking 인프라 외부에 있는 시스템과 동일한 계층-2 네트워크를 공유할 수 있습니다.

4.3. IP 주소 사용

다음 시스템은 할당된 범위의 IP 주소를 사용합니다.

- ***물리 노드** - 각 물리적 NIC에는 하나의 IP 주소가 필요합니다. 물리적 NIC를 특정 기능에 전용으로 지정하는 것이 일반적입니다. 예를 들어, 관리 및 NFS 트래픽을 별도의 물리적 NIC에 할당하고, 중복을 위해 여러 NIC를 서로 다른 스위치에 연결하는 경우가 있습니다.*

- ***고가용성을 위한 VIP(가상 IP)** - 컨트롤러 노드가 공유하는 각 네트워크에 대해 VIP를 1~3개에 할당할 계획입니다.*

4.4. 가상 네트워킹

다음 가상 리소스는 OpenStack Networking에서 IP 주소를 사용합니다. 이러한 리소스는 클라우드 인프라로 로컬로 간주되며 외부 물리적 네트워크의 시스템에서 연결할 필요가 없습니다.

- ***프로젝트 네트워크 - 각 프로젝트 네트워크에** IP 주소를 인스턴스에 할당하는 데 사용할 수 있는 서브넷이 필요합니다.*

- ***가상 라우터** - 서브넷에 연결하는 각 라우터 인터페이스에는 하나의 IP 주소가 필요합니다. DHCP를 사용하려면 각 라우터 인터페이스에 두 개의 IP 주소가 필요합니다.*

- ***인스턴스** - 각 인스턴스에는 인스턴스를 호스팅하는 프로젝트 서브넷의 주소가 필요합니다. 인그레스 트래픽이 필요한 경우 지정된 외부 네트워크에서 인스턴스에 유동 IP 주소를 할당해야 합니다.*

- ***관리 트래픽** - OpenStack 서비스 및 API 트래픽이 포함됩니다. 모든 서비스는 적은 수의 VIP를 공유합니다. API, RPC 및 데이터베이스 서비스는 내부 API VIP에서 통신합니다.*

4.5. 네트워크 계획 예

이 예에서는 각 서브넷이 다양한 IP 주소가 할당되는 여러 서브넷을 수용하는 여러 네트워크를 보여줍니다.

표 4.1. 서브넷 계획 예

서브넷 이름	주소 범위	주소 수	서브넷 마스크
프로비저닝 네트워크	192.168.100.1 - 192.168.100.250	250	255.255.255.0
내부 API 네트워크	172.16.1.10 - 172.16.1.250	241	255.255.255.0
스토리지	172.16.2.10 - 172.16.2.250	241	255.255.255.0
스토리지 관리	172.16.3.10 - 172.16.3.250	241	255.255.255.0
테넌트 네트워크 (GRE/VXLAN)	172.16.4.10 - 172.16.4.250	241	255.255.255.0

서브넷 이름	주소 범위	주소 수	서브넷 마스크
외부 네트워크(유동 IP 포함)	10.1.2.10 - 10.1.3.222	469	255.255.254.0
공급자 네트워크(인프라)	10.10.3.10 - 10.10.3.250	241	255.255.252.0

5장. OPENSTACK NETWORKING 라우터 포트 검토

OpenStack Networking의 가상 라우터는 포트를 사용하여 서브넷과 상호 연결합니다. 이러한 포트의 상태를 검토하여 해당 포트가 예상대로 연결되는지 여부를 확인할 수 있습니다.

5.1. 현재 포트 상태 보기

특정 라우터에 연결된 모든 포트를 나열하고 포트(DOWN 또는ACTIVE)의 현재 상태를 검색하려면 다음 단계를 완료합니다.

1. **r1** 라우터에 연결된 모든 포트를 보려면 다음 명령을 실행합니다.

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

결과 예:

```
+--------------------------------------+------+-------------------+-------------------------------------
----------------------------------------+
| id                                   | name | mac_address       | fixed_ips
|
+--------------------------------------+------+-------------------+-------------------------------------
----------------------------------------+
| b58d26f0-cc03-43c1-ab23-ccdb1018252a |      | fa:16:3e:94:a7:df | {"subnet_id": "a592fdba-
babd-48e0-96e8-2dd9117614d3", "ip_address": "192.168.200.1"} |
| c45e998d-98a1-4b23-bb41-5d24797a12a4 |      | fa:16:3e:ee:6a:f7 | {"subnet_id": "43f8f625-
c773-4f18-a691-fd4ebfb3be54", "ip_address": "172.24.4.225"}  |
+--------------------------------------+------+-------------------+-------------------------------------
----------------------------------------+
```

2. 각 포트의 세부 정보를 보려면 다음 명령을 실행합니다. 보려는 포트의 포트 ID를 포함합니다. 결과적으로 다음 예에 **ACTIVE** 상태가 있는 것으로 표시된 포트 상태가 포함됩니다.

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

결과 예:

```
+-----------------------+-------------------------------------------------------------------------------------
+
| Field                 | Value                                                                               |
+-----------------------+-------------------------------------------------------------------------------------
+
| admin_state_up        | True                                                                                |
| allowed_address_pairs |                                                                                     |
| binding:host_id       | node.example.com                                                                    |
| binding:profile       | {}                                                                                  |
| binding:vif_details   | {"port_filter": true, "ovs_hybrid_plug": true}                                      |
| binding:vif_type      | ovs                                                                                 |
| binding:vnic_type     | normal                                                                              |
| device_id             | 49c6ebdc-0e62-49ad-a9ca-58cea464472f                                                |
| device_owner          | network:router_interface                                                            |
| extra_dhcp_opts       |                                                                                     |
| fixed_ips             | {"subnet_id": "a592fdba-babd-48e0-96e8-2dd9117614d3", "ip_address":
"192.168.200.1"} |
| id                    | b58d26f0-cc03-43c1-ab23-ccdb1018252a                                                |
```

```
| mac_address           | fa:16:3e:94:a7:df                                                                                  |
| name                  |                                                                                                    |
| network_id            | 63c24160-47ac-4140-903d-8f9a670b0ca4
|
| security_groups       |                                                                                                    |
| status                | ACTIVE                                                                                             |
| tenant_id             | d588d1112e0f496fb6cac22f9be45d49                                                                   |
+-----------------------+----------------------------------------------------------------------------------------------------
+
```

3. *각 포트에 대해 2단계를 수행하여 상태를 검색합니다.*

6장. 공급자 네트워크 문제 해결

SDN(소프트웨어 정의 네트워킹)이라고도 하는 가상 라우터 및 스위치의 배포는 복잡성을 유발할 수 있습니다. 그러나 OpenStack Networking에서 네트워크 연결 문제를 해결하는 진단 프로세스는 실제 네트워크의 진단 프로세스와 유사합니다. VLAN을 사용하는 경우 가상 인프라를 완전히 분리된 환경이 아니라 실제 네트워크의 트렁크 확장으로 간주할 수 있습니다.

6.1. 기본 PING 테스트

ping 명령은 네트워크 연결 문제를 분석하는 데 유용한 도구입니다. 결과는 네트워크 연결의 기본 지표 역할을 하지만 실제 애플리케이션 트래픽을 차단하는 방화벽과 같은 모든 연결 문제를 완전히 제외하지는 않을 수 있습니다. ping 명령은 트래픽을 특정 대상으로 전송한 다음 시도가 성공했는지 여부를 다시 보고합니다.

참고

ping 명령은 ICMP 작업입니다. **ping** 을 사용하려면 ICMP 트래픽이 중간 방화벽을 통과하는 것을 허용해야 합니다.

ping 테스트는 네트워크 문제가 발생한 시스템에서 실행할 때 가장 유용하므로 시스템이 완전히 오프라인인 것처럼 보이면 VNC 관리 콘솔을 통해 명령줄에 연결해야 할 수도 있습니다.

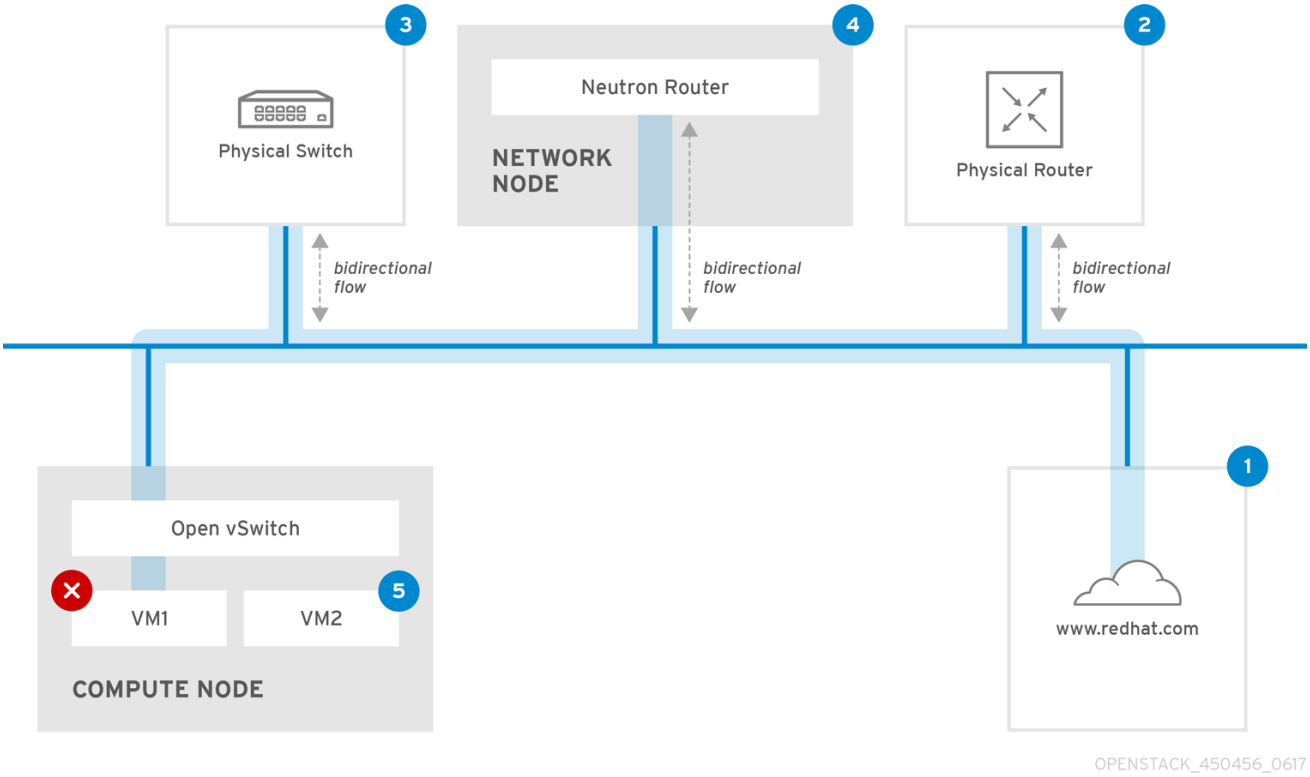
예를 들어 다음 ping test 명령은 성공하기 위해 여러 계층의 네트워크 인프라의 유효성을 검사합니다. 이름 확인, IP 라우팅 및 네트워크 스위칭이 모두 올바르게 작동해야 합니다.

```
$ ping www.redhat.com

PING e1890.b.akamaiedge.net (125.56.247.214) 56(84) bytes of data.
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=1
ttl=54 time=13.4 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=2
ttl=54 time=13.5 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com (125.56.247.214): icmp_seq=3
ttl=54 time=13.4 ms
^C
```

결과 요약이 표시되는 Ctrl-c를 사용하여 ping 명령을 종료할 수 있습니다. 제로 비율의 패킷 손실은 연결이 안정적이며 시간 초과되지 않았음을 나타냅니다.

```
--- e1890.b.akamaiedge.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.461/13.498/13.541/0.100 ms
```

테스트 대상에 따라 ping 테스트 결과가 매우 표시될 수 있습니다. 예를 들어 다음 다이어그램 VM1에서는 몇 가지 형태의 연결 문제가 발생했습니다. 가능한 대상에는 파란색으로 번호가 지정되며 성공 또는 실패한 결과에서 얻은 결론이 표시됩니다.



*1. **인터넷** - 일반적인 첫 번째 단계는 인터넷 위치(예: www.redhat.com)로 ping 테스트를 보내는 것입니다.*

- *성공: 이 테스트는 시스템과 인터넷 사이의 모든 다양한 네트워크 지점이 올바르게 작동하고 있음을 나타냅니다. 여기에는 가상 및 물리적 네트워크 인프라가 포함됩니다.*
- *실패: 오래된 인터넷 위치에 대한 ping 테스트가 실패할 수 있는 여러 가지 방법이 있습니다. 네트워크의 다른 시스템이 인터넷을 ping 할 수 있으면 인터넷 연결이 작동 중임을 증명하고 로컬 시스템 구성 내에서 문제가 발생할 수 있습니다.*

*2. **물리적 라우터** - 네트워크 관리자가 트래픽을 외부 대상으로 전송하도록 지정하는 라우터 인터페이스입니다.*

- *성공: 물리적 라우터로 Ping 테스트를 수행하면 로컬 네트워크 및 기본 스위치가 작동하는지 확인할 수 있습니다. 이러한 패킷은 라우터를 통과하지 않으므로 기본 게이트웨이에 라우팅 문제가 있는지를 증명하지 않습니다.*
- *실패: 이는 VM1과 기본 게이트웨이 사이에 문제가 있음을 나타냅니다. 라우터/하위가 꺼져 있거나 잘못된 기본 게이트웨이를 사용 중일 수 있습니다. 사용자가 알고 있는 다른 서버에서 구성을 올바르게 작동하고 있는 구성과 비교합니다. 로컬 네트워크에서 다른 서버를 ping 합니다.*

*3. **Neutron 라우터** - Red Hat OpenStack Platform에서 가상 시스템의 트래픽을 전송하는 데 사용하는 가상 SDN(소프트웨어 정의 네트워킹) 라우터입니다.*

- *성공: 방화벽은 ICMP 트래픽을 허용하며 네트워킹 노드는 온라인 상태입니다.*
- *실패: 인스턴스의 보안 그룹에 ICMP 트래픽이 허용되는지 확인합니다. Networking 노드가 온라인 상태인지 확인하고 모든 필수 서비스가 실행 중인지 확인하고 L3 에이전트 로그 (/var/log/neutron/l3-agent.log)를 검토합니다.*

*4. **물리적 스위치** - 물리적 스위치는 동일한 실제 네트워크의 노드 간 트래픽을 관리합니다.*

- *성공: VM에서 물리적 스위치로 전송되는 트래픽은 이 세그먼트가 올바르게 작동함을 나타내는 가상 네트워크 인프라를 통과해야 합니다.*

- *실패: 물리적 스위치 포트가 필요한 VLAN을 트렁크하도록 구성되어 있는지 확인합니다.*

5. VM2 - 동일한 컴퓨팅 노드의 동일한 서브넷에서 VM을 ping 합니다.

- *성공: VM1의 NIC 드라이버 및 기본 IP 구성이 작동합니다.*
- *실패: VM1의 네트워크 구성을 확인합니다. 또는 VM2의 방화벽은 단순히 ping 트래픽을 차단하는 것일 수 있습니다. 또한 가상 전환 구성을 확인하고 Open vSwitch(또는 Linux 브리지) 로그 파일을 검토합니다.*

6.2. VLAN 네트워크 문제 해결

OpenStack Networking은 VLAN 네트워크를 SDN 스위치로 트렁크할 수 있습니다. VLAN 태그가 지정된 프로바이더 네트워크를 지원하므로 가상 인스턴스가 실제 네트워크의 서버 서브넷과 통합할 수 있습니다.

VLAN 공급자 네트워크에 대한 연결 문제를 해결하려면 다음 단계를 완료합니다.

1. ***ping <gateway-IP-address>** 를 사용하여 게이트웨이를 ping 합니다.*
   다음 명령을 사용하여 네트워크를 생성하는 이 예제를 고려하십시오.

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

   *이 예에서 게이트웨이 IP 주소는 **192.168.120.254** 입니다.*

```
$ ping 192.168.120.254
```

2. *ping이 실패하면 다음을 수행합니다.*

   a. *연결된 VLAN의 네트워크 흐름이 있는지 확인합니다.*
   *VLAN ID가 설정되지 않았을 수 있습니다. 이 예에서 OpenStack Networking은 VLAN 120을 프로바이더 네트워크로 트렁크하도록 구성되어 있습니다. (1단계 예제의 **--provider:segmentation_id=120** 참조)*

   b. ***ovs-ofctl dump-flows <bridge-name>** 명령을 사용하여 브리지 인터페이스에서 VLAN 흐름을 확인합니다.*
   *이 예에서는 브리지의 이름은 **br-ex**:*

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

 NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=987.521s, table=0, n_packets=67897, n_bytes=14065247, idle_age=0, priority=1 actions=NORMAL
  cookie=0x0, duration=986.979s, table=0, n_packets=8, n_bytes=648, idle_age=977, priority=2,in_port=12 actions=drop
```

6.2.1. VLAN 구성 및 로그 파일 검토

- ***OpenStack Networking(neutron) 에이전트 - openstack network agent list** 명령을 사용하여 모든 에이전트가 올바른 이름으로 설정 및 등록되었는지 확인합니다.*

```
(overcloud)[stack@undercloud~]$ openstack network agent list
+--------------------------------------+--------------------+----------------------+-------+----------------+
| id                                   | agent_type         | host                 | alive | admin_state_up |
+--------------------------------------+--------------------+----------------------+-------+----------------+
| a08397a8-6600-437d-9013-b2c5b3730c0c | Metadata agent     | rhelosp.example.com  | :-)   | True           |
| a5153cd2-5881-4fc8-b0ad-be0c97734e6a | L3 agent           | rhelosp.example.com  | :-)   | True           |
| b54f0be7-c555-43da-ad19-5593a075ddf0 | DHCP agent         | rhelosp.example.com  | :-)   | True           |
| d2be3cb0-4010-4458-b459-c5eb0d4d354b | Open vSwitch agent | rhelosp.example.com  | :-)   | True           |
+--------------------------------------+--------------------+----------------------+-------+----------------+
```

- **/var/log/containers/neutron/openvswitch-agent.log** 를 검토합니다. 이 로그는 생성 프로세스에서 VLAN 트렁킹을 구성하기 위해 **ovs-ofctl** 명령을 사용했는지 확인합니다.
- **/etc/neutron/l3_agent.ini** 파일에서 **external_network_bridge** 의 유효성을 검사합니다. **external_network_bridge** 매개변수에 하드 코딩된 값이 있는 경우 L3-agent와 함께 프로바이더 네트워크를 사용할 수 없으며 필요한 흐름을 생성할 수 없습니다. **external_network_bridge** 값은 'external_network_bridge = "" ' 형식이어야 합니다.
- **/etc/neutron/plugin.ini** 파일에서 **network_vlan_ranges** 값을 확인합니다. 공급자 네트워크의 경우 숫자 VLAN ID를 지정하지 마십시오. VLAN 분리 프로젝트 네트워크를 사용하는 경우에만 ID를 지정합니다.
- OVS 에이전트 구성 파일 브리지 매핑의 유효성을 검사하고 **phy-eno1에 매핑된 브리지가 있으며 eno1** 에 올바르게 연결되어 있는지 확인합니다.

6.3. 프로젝트 네트워크에서 문제 해결

OpenStack Networking에서는 프로젝트에서 서로 방해하지 않고 네트워크를 구성할 수 있도록 모든 프로젝트 트래픽이 네트워크 네임스페이스에 포함됩니다. 예를 들어 네트워크 네임스페이스를 사용하면 서로 다른 프로젝트 간에 간섭 없이 동일한 서브넷 범위를 192.168.1.1/24로 설정할 수 있습니다.

프로젝트 네트워크 문제 해결을 시작하려면 먼저 네트워크가 포함된 네트워크 네임스페이스를 결정합니다.

1. **openstack network list 명령을 사용하여 모든 프로젝트 네트워크를 나열합니다.**

```
# (overcloud)[stack@osp13-undercloud ~]$ openstack network list
+--------------------------------------+-------------+--------------------------------------------------------------+
| id                                   | name        | subnets                                                      |
+--------------------------------------+-------------+--------------------------------------------------------------+
| 9cb32fe0-d7fb-432c-b116-f483c6497b08 | web-servers | 453d6769-fcde-4796-a205-66ee01680bba 192.168.212.0/24 |
| a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81 | private     | c1e58160-707f-44a7-bf94-8694f29e74d3 10.0.0.0/24      |
| baadd774-87e9-4e97-a055-326bb422b29b | private     | 340c58e1-7fe7-4cf2-96a7-96a0a4ff3231 192.168.200.0/24 |
| 24ba3a36-5645-4f46-be47-f6af2a7d8af2 | public      | 35f3d2cb-6e4b-4527-a932-952a395c4bb3 172.24.4.224/28  |
+--------------------------------------+-------------+--------------------------------------------------------------+
```

*이 예에서 **web-servers** 네트워크를 살펴봅니다. web-server 행에서 id 값을 기록합니다(**9cb32fe0-d7fb-432c-b116-f483c6497b08**). 이 값은 네트워크 네임스페이스에 추가되어 다음 단계에서 네임스페이스를 식별하는 데 도움이 됩니다.*

2. ***ip netns list** 명령을 사용하여 모든 네트워크 네임스페이스를 나열합니다.*

```
# ip netns list
qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08
qrouter-31680a1c-9b3e-4906-bd69-cb39ed5faa01
qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b
qdhcp-a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81
qrouter-e9281608-52a6-4576-86a6-92955df46f56
```

*출력에는 **web-servers** 네트워크 ID와 일치하는 네임스페이스가 포함되어 있습니다. 이 예에서 네임스페이스는 **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08** 입니다.*

3. *네임스페이스 내에서 명령을 실행하고 **ip netns exec <namespace>** 를 사용하여 문제 해결 명령 앞에 추가하여 **web-servers** 네트워크의 구성을 검사합니다.*

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

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.24.4.225    0.0.0.0         UG    0      0        0 qg-8d128f89-87
172.24.4.224    0.0.0.0         255.255.255.240 U     0      0        0 qg-8d128f89-87
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 qr-8efd6357-96
```

6.3.1. 네임 스페이스 내에서 고급 ICMP 테스트 수행

1. ***tcpdump** 명령을 사용하여 ICMP 트래픽을 캡처합니다.*

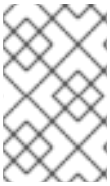
```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b tcpdump -qnntpi any icmp
```

2. *별도의 명령줄 창에서 외부 네트워크에 대한 ping 테스트를 수행합니다.*

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

3. ***tcpdump** 세션을 실행하는 터미널에서 ping 테스트의 자세한 결과를 관찰합니다.*

```
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
IP (tos 0xc0, ttl 64, id 55447, offset 0, flags [none], proto ICMP (1), length 88)
    172.24.4.228 > 172.24.4.228: ICMP host 192.168.200.20 unreachable, length 68
 IP (tos 0x0, ttl 64, id 22976, offset 0, flags [DF], proto UDP (17), length 60)
    172.24.4.228.40278 > 192.168.200.21: [bad udp cksum 0xfa7b -> 0xe235!] UDP, length 32
```



참고

트래픽에 대한 tcpdump 분석을 수행할 때 인스턴스 대신 라우터 인터페이스로 향하는 패킷이 응답하는 것을 관찰할 수 있습니다. qrouter가 반환 패킷에서 DNAT를 수행하므로 이는 예상된 동작입니다.

7장. 물리적 네트워크에 인스턴스 연결

이 장에서는 프로바이더 네트워크를 사용하여 인스턴스를 외부 네트워크에 직접 연결하는 방법을 설명합니다.

7.1. OPENSTACK NETWORKING 토폴로지 개요

OpenStack Networking(neutron)에는 여러 노드 유형에 분산된 두 가지 서비스 범주가 있습니다.

- **Neutron 서버** - 이 서비스는 OpenStack Networking API 서버와 상호 작용하는 최종 사용자 및 서비스에 API를 제공하는 OpenStack Networking API 서버를 실행합니다. 또한 이 서버는 기본 데이터베이스와 통합되어 프로젝트 네트워크, 라우터 및 로드 밸런서 세부 정보를 저장하고 검색합니다.
- **Neutron 에이전트** - OpenStack 네트워킹의 네트워크 기능을 수행하는 서비스입니다.
  - **neutron-dhcp-agent** - 프로젝트 사설 네트워크의 DHCP IP 주소 지정을 관리합니다.
  - **neutron-l3-agent** - 프로젝트 사설 네트워크, 외부 네트워크 등의 계층 3 라우팅을 수행합니다.
- **계산 노드** - 이 노드는 가상 머신(인스턴스라고도 함)을 실행하는 하이퍼바이저를 호스팅합니다. 인스턴스에 외부 연결을 제공하려면 컴퓨팅 노드를 네트워크에 직접 연결해야 합니다. 일반적으로 이 노드는 **neutron-openvswitch-agent** 와 같이 l2 에이전트가 실행되는 위치입니다.

추가 리소스

- OpenStack Networking 서비스 배치

7.2. OPENSTACK NETWORKING 서비스 배치

OpenStack Networking 서비스는 동일한 물리적 서버에서 또는 해당 역할에 따라 이름이 지정된 별도의 전용 서버에서 함께 실행할 수 있습니다.

- Controller node(컨트롤러 노드) - API 서비스를 실행하는 서버입니다.
- 네트워크 노드 - OpenStack Networking 에이전트를 실행하는 서버입니다.
- 계산 노드 - 인스턴스를 호스팅하는 하이퍼바이저 서버입니다.

이 장의 단계는 이러한 세 가지 노드 유형을 포함하는 환경에 적용됩니다. 배포에 동일한 물리 노드에 컨트롤러 및 네트워크 노드 역할이 모두 있는 경우 해당 서버의 두 섹션에서 단계를 수행해야 합니다. 세 개의 노드 모두 컨트롤러 노드와 HA를 사용하여 네트워크 노드 서비스를 실행할 수 있는 HA(고가용성) 환경에도 적용됩니다. 결과적으로 세 노드 모두에서 컨트롤러 및 네트워크 노드에 적용되는 섹션의 단계를 완료해야 합니다.

추가 리소스

- OpenStack Networking 토폴로지 개요

7.3. 플랫 공급자 네트워크 구성

플랫 프로바이더 네트워크를 사용하여 인스턴스를 외부 네트워크에 직접 연결할 수 있습니다. 이 기능은 물리적 네트워크와 물리적 인터페이스가 여러 개 있고 각 컴퓨팅 및 네트워크 노드를 해당 외부 네트워크에 연결하려는 경우에 유용합니다.

사전 요구 사항

- *물리적 네트워크가 여러 개 있습니다.*
  *이 예에서는 각각 **physnet1 및 physnet 2** 라는 물리적 네트워크를 사용합니다.*
- *개별 물리적 인터페이스가 있습니다.*
  *이 예에서는 각각 별도의 물리적 인터페이스인 **eth0** 및 **eth1** 을 사용합니다.*

절차

1. *Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.*

   예제

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

   작은 정보

   Red Hat OpenStack Platform Orchestration 서비스(heat)는 템플릿 이라는 일련의 계획을 사용하여 환경을 설치하고 구성합니다. 오케스트레이션 템플릿에 대한 사용자 지정을 제공하는 특수한 유형의 템플릿인 사용자 지정 환경 파일을 사용하여 오버클라우드의 부분을 사용자 지정할 수 있습니다.

2. ***parameter_defaults** 아래의 YAML 환경 파일에서 **NeutronBridgeMappings** 를 사용하여 외부 네트워크에 액세스하는 데 사용되는 OVS 브리지를 지정합니다.*

   예제

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

3. *컨트롤러 및 컴퓨팅 노드에 대한 사용자 정의 NIC 구성 템플릿에서 인터페이스가 연결된 브릿지를 구성합니다.*

   예제

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

```
            mtu: 1500
            use_dhcp: false
            members:
            - type: interface
              name: eth1
              mtu: 1500
              use_dhcp: false
              primary: true
...
```

4. ***openstack overcloud deploy*** *명령을 실행하고 수정된 사용자 지정 NIC 템플릿 및 새 환경 파일을 포함하여 템플릿과 환경 파일을 포함합니다.*

중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다.

예제

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

검증 단계

1. *외부 네트워크(**public1**)를 플랫 네트워크로 생성하고 구성된 물리적 네트워크(phys **net1**)와 연결합니다.*
   *다른 사용자가 외부 네트워크에 직접 연결하는 VM 인스턴스를 생성하도록 공유 네트워크(**--share**사용)로 구성합니다.*

   예제

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

2. ***openstack subnet create** 명령을 사용하여 서브넷(**public_subnet**)을 생성합니다.*

   예제

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

3. *VM 인스턴스를 생성하고 새로 생성한 외부 네트워크에 직접 연결합니다.*

   예제

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

추가 리소스

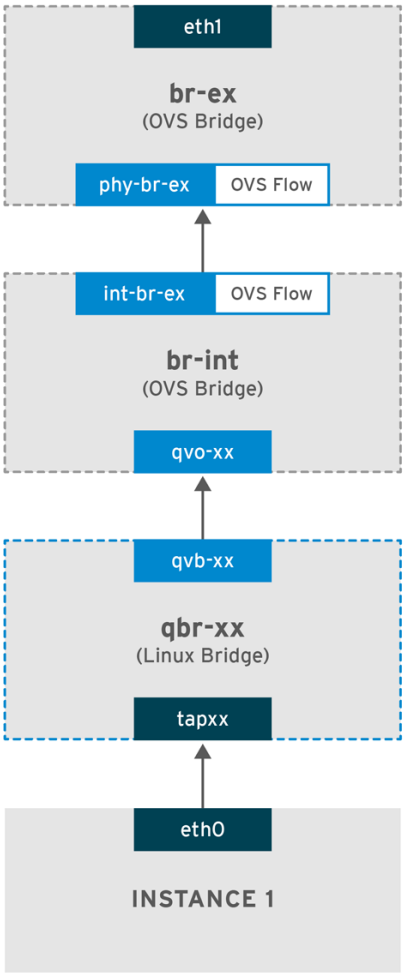
- *Advanced Overcloud Customization 가이드의 사용자 지정 네트워크 인터페이스 템플릿*
- *Advanced Overcloud Customization 가이드의 환경 파일*
- *Advanced Overcloud Customization 가이드 의 Overcloud 생성에 환경 파일 포함*
- *명령줄 인터페이스 참조에서 네트워크 생성*
- *명령줄 인터페이스 참조에서 서브넷 생성*
- *명령줄 인터페이스 참조에서 서버 생성*

7.4. 플랫 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까?

이 섹션에서는 플랫 프로바이더 네트워크 구성이 있는 인스턴스와의 트래픽 흐름 방법을 자세히 설명합니다.

플랫 공급자 네트워크에서 나가는 트래픽 흐름

다음 다이어그램에서는 인스턴스를 벗어나고 외부 네트워크에 직접 도착하는 트래픽의 패킷 흐름을 설명합니다. ***br-ex*** *외부 브리지를 구성한 후 실제 인터페이스를 브리지에 추가하고 Compute 노드에 인스턴스를 생성합니다. 결과적으로 생성된 인터페이스 및 브리지 구성은 다음 다이어그램의 구성과 비슷합니다(* ***iptables_hybrid*** *방화벽 드라이버를 사용하는 경우).*



OPENSTACK_450456_0617

1. *패킷은 인스턴스의* ***eth0*** *인터페이스를 종료하고 linux bridge* ***qbr-xx에 도달합니다.***

2. *bridge **qbr-xx** 는 veth 쌍 **qvb -xx <-> qvo-xxx를 사용하여 br- int** 에 연결됩니다. 이는 브리지가 보안 그룹에서 정의한 인바운드/아웃바운드 방화벽 규칙을 적용하는 데 사용되기 때문입니다.*

3. ***qvb-xx** 인터페이스는 **qbr-xx** Linux 브리지에 연결되고 **qvoxx** 는 **br-int** Open vSwitch(OVS) 브리지에 연결됩니다.*

'qbr-xx'Linux 브리지의 구성 예:

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

br-int의 qvo- xx 구성 :

```
# ovs-vsctl show
 Bridge br-int
      fail_mode: secure
         Interface "qvof63599ba-8f"
      Port "qvo269d4d73-e7"
         tag: 5
         Interface "qvo269d4d73-e7"
```

참고

*포트 **qvo-xx** 는 flat 프로바이더 네트워크와 연결된 내부 VLAN 태그로 태그가 지정됩니다. 이 예에서 VLAN 태그는 **5** 입니다. 패킷이 **qvo-xx** 에 도달하면 VLAN 태그가 패킷 헤더에 추가됩니다.*

*그런 다음 patch-peer **int -br-ex <-> phy-br-ex를 사용하여 패킷이 br -ex** OVS 브리지로 이동합니다.*

***br-int** 의 patch-peer 구성 예 :*

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

***br-ex** 의 patch-peer 구성 예 :*

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

*이 패킷이 **br -ex의 phy-br -ex** 에 도달하면 **br-ex** 내부의 OVS 흐름이 VLAN 태그를 제거하고 실제 인터페이스로 전달합니다.*

*다음 예에서 출력에는 **phy-br-ex** 의 포트 번호가 **2** 가 표시되어 있습니다.*

```
 # ovs-ofctl show br-ex
OFPT_FEATURES_REPLY (xid=0x2): dpid:00003440b5c90dc6
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE

 2(phy-br-ex): addr:ba:b5:7b:ae:5c:a2
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

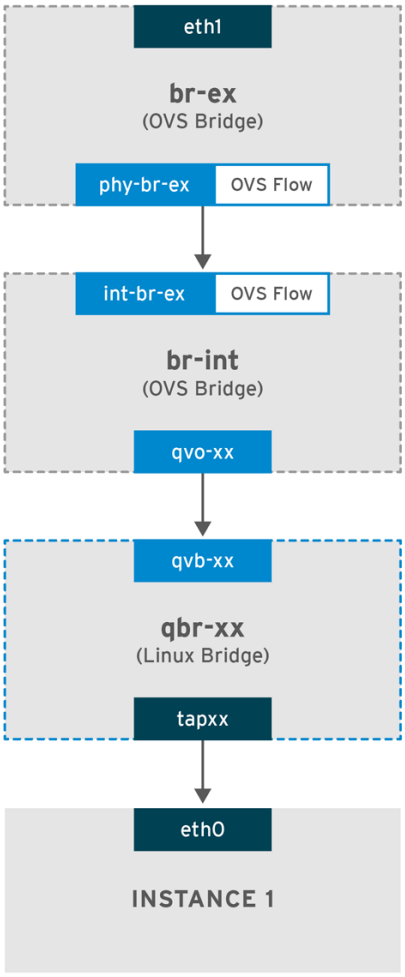
*다음 출력은 VLAN 태그가 **5** (dl_**vlan=5**) 인 **phy-br-ex** (**in_port=2**) 에 도착하는 모든 패킷을 보여줍니다. 또한 br-ex의 OVS 흐름은 VLAN 태그를 제거하고 패킷을 실제 인터페이스로 전달합니다.*

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4703.491s, table=0, n_packets=3620, n_bytes=333744, idle_age=0, priority=1
actions=NORMAL
 cookie=0x0, duration=3890.038s, table=0, n_packets=13, n_bytes=1714, idle_age=3764,
priority=4,in_port=2,dl_vlan=5 actions=strip_vlan,NORMAL
 cookie=0x0, duration=4702.644s, table=0, n_packets=10650, n_bytes=447632, idle_age=0,
priority=2,in_port=2 actions=drop
```

실제 인터페이스가 또 다른 VLAN 태그 지정된 인터페이스인 경우 실제 인터페이스는 패킷에 태그를 추가합니다.

플랫 공급자 네트워크에서 들어오는 트래픽 흐름

이 섹션에는 인스턴스의 인터페이스에 도달할 때까지 외부 네트워크에서 들어오는 트래픽 흐름에 대한 정보가 포함되어 있습니다.



OPENSTACK_450456_0617

1. *들어오는 트래픽은 물리적 노드의* ***eth1*** *에 도달합니다.*
2. *패킷은* ***br-ex*** *브리지에 전달됩니다.*
3. *패킷은 patch-peer* ***phy -br-ex <--> int-br-ex를 통해 br-int 로 이동합니다.***

다음 예에서 ***int-br-ex*** *는 포트 번호* ***15*** *를 사용합니다.* ***15(int-br-ex)*** *가 포함된 항목을 참조하십시오.*

```
 # ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:00004e67212f644d
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 15(int-br-ex): addr:12:4e:44:a9:50:f4
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

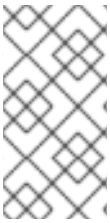
br-int에서 트래픽 흐름 관찰

1. *패킷이* ***int-br-ex*** *에 도달하면* ***br-int*** *브리지 내의 OVS 흐름 규칙이 패킷을 수정하여 내부 VLAN 태그* ***5*** *를 추가합니다.* ***actions=mod_vlan_vid:5*** *에 대한 항목을 참조하십시오.*

```
 # ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
```

```
 cookie=0x0, duration=5351.536s, table=0, n_packets=12118, n_bytes=510456, idle_age=0,
priority=1 actions=NORMAL
 cookie=0x0, duration=4537.553s, table=0, n_packets=3489, n_bytes=321696, idle_age=0,
priority=3,in_port=15,vlan_tci=0x0000 actions=mod_vlan_vid:5,NORMAL
 cookie=0x0, duration=5350.365s, table=0, n_packets=628, n_bytes=57892, idle_age=4538,
priority=2,in_port=15 actions=drop
 cookie=0x0, duration=5351.432s, table=23, n_packets=0, n_bytes=0, idle_age=5351, priority=0
actions=drop
```

1. 두 번째 규칙은 VLAN 태그(vlan_tci=0x0000)가 없는 int-br-ex(in_port=15)에 도착하는 패킷을 관리합니다. 이 규칙은 VLAN 태그 5를 패킷(**actions=mod_vlan_vid:5,NORMAL**)에 추가하고 **qvoxxx** 로 전달합니다.
2. **qvoXXX** 는 VLAN 태그를 제거한 후 패킷을 수락하고 **qvbxx** 로 전달합니다.
3. 그런 다음 패킷이 인스턴스에 도달합니다.



참고

VLAN 태그 5는 플랫 프로바이더 네트워크가 있는 테스트 컴퓨팅 노드에서 사용된 VLAN의 예입니다. 이 값은 **neutron-openvswitch-agent** 에서 자동으로 할당했습니다. 이 값은 자체 플랫 프로바이더 네트워크에 따라 다를 수 있으며, 별도의 두 컴퓨팅 노드의 동일한 네트워크에 따라 다를 수 있습니다.

추가 리소스

- 플랫 공급자 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결

7.5. 플랫 공급자 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결

"Flat provider 네트워크 패킷 흐름은 어떻게 작동합니까?"에 제공된 출력은 모든 문제가 발생하는 경우 플랫 프로바이더 네트워크 문제 해결에 필요한 충분한 디버깅 정보를 제공합니다. 다음 단계에는 문제 해결 프로세스에 대한 추가 정보가 포함되어 있습니다.

절차

1. bridge_mappings를 검토합니다.

사용하는 물리적 네트워크 이름(예: **physnet1)이** 다음 예와 같이 **bridge_mapping** 구성 내용과 일치하는지 확인합니다.

```
 # grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
bridge_mappings = physnet1:br-ex

 # openstack network show provider-flat
...
| provider:physical_network | physnet1
...
```

2. 네트워크 구성을 검토합니다.

네트워크가 **외부로** 생성되고 **flat** 유형을 사용하는지 확인합니다.

```
 # openstack network show provider-flat
...
```

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

3. patch-peer를 검토합니다.

ovs-vsctl show *명령을 실행하고 patch-peer* ***int -br -ex <--> phy-br-ex*** *를 사용하여* ***br-int*** *및* ***br -ex*** *가 연결되어 있는지 확인합니다.*

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

br-ex *의 patch-peer 구성 예 :*

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

이 연결은 ***bridge_mapping이 /etc/neutron/plugins/ml2/ openvswitch_ agent.ini에 올바르게 구성된 경우*** ***neutron- openvswitch-agent*** *서비스를 다시 시작할 때 생성됩니다. 서비스를 다시 시작한 후 연결이 생성되지 않는 경우* ***bridge_mapping*** *설정을 다시 확인합니다.*

4. 네트워크 흐름을 검토합니다.

ovs-ofctl dump-flows br-ex *및* ***ovs-ofctl dump-flows br-int*** *를 실행하고 흐름이 나가는 패킷의 내부 VLAN ID를 제거하고 들어오는 패킷의 VLAN ID를 추가하는지 검토합니다. 이 흐름은 먼저 특정 컴퓨팅 노드에서 이 네트워크에 인스턴스를 생성할 때 추가됩니다.*

- *인스턴스를 생성한 후에 이 흐름을 생성하지 않으면 네트워크가* ***flat***, external, ***physical _network*** *이름이 올바른지 확인합니다. 또한* ***bridge_mapping*** *설정을 검토합니다.*
- *마지막으로* ***ifcfg-br-ex*** *및* ***ifcfg-ethx*** *구성을 검토합니다.* ***ethX*** *가* ***br-ex 내의 포트로 추가되고 ifcfg- br-ex 및 ifcfg- ethx*** *에* ***ip a*** *의 출력에* ***UP*** *플래그가 있는지 확인합니다.*

다음 출력은 ***eth1*** *이* ***br-ex*** *의 포트임을 보여줍니다.*

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

다음 예제에서는 **eth1** 이 OVS 포트로 구성되어 있고 커널이 인터페이스에서 모든 패킷을 전송하고 OVS 브리지 **br-ex** 로 전송한다는 것을 보여줍니다. 이는 **master ovs-system** 항목에서 확인할 수 있습니다.

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

추가 리소스

- 플랫 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까?
- 브릿지 매핑 구성

7.6. VLAN 공급자 네트워크 구성

단일 NIC의 VLAN 태그 지정 인터페이스를 여러 공급자 네트워크에 연결하면 이러한 새 VLAN 공급자 네트워크가 VM 인스턴스를 외부 네트워크에 직접 연결할 수 있습니다.

사전 요구 사항

- VLAN 범위가 있는 물리적 네트워크가 있습니다.
  이 예에서는 VLAN 범위가 **171-172** 인 **physnet1** 이라는 물리적 네트워크를 사용합니다.
- 네트워크 노드와 컴퓨팅 노드는 물리적 인터페이스를 사용하여 물리적 네트워크에 연결됩니다.
  이 예에서는 물리적 인터페이스 **eth1을 사용하여 물리적 네트워크 physnet1 에 연결된 네트워크 노드 및 컴퓨팅 노드를 사용합니다.**
- 이러한 인터페이스가 연결되는 스위치 포트는 필수 VLAN 범위를 트렁크하도록 구성해야 합니다.

절차

1. Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.

   예제

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

   작은 정보

   Red Hat OpenStack Platform Orchestration 서비스(heat)는 템플릿 이라는 일련의 계획을 사용하여 환경을 설치하고 구성합니다. 오케스트레이션 템플릿에 대한 사용자 지정을 제공하는 특수한 유형의 템플릿인 사용자 지정 환경 파일을 사용하여 오버클라우드의 부분을 사용자 지정할 수 있습니다.

2. **parameter_defaults** 아래의 YAML 환경 파일에서 **NeutronTypeDrivers** 를 사용하여 네트워크 유형 드라이버를 지정합니다.

   예제

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

3. *사용 중인 물리적 네트워크 및 VLAN 범위를 반영하도록* ***NeutronNetworkVLANRanges*** *설정을 구성합니다.*

   예제

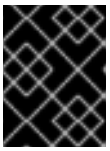
   ```
   parameter_defaults:
     NeutronTypeDrivers: 'vxlan,flat,vlan'
     NeutronNetworkVLANRanges: 'physnet1:171,172'
   ```

4. *외부 네트워크 브리지(br-ex)를 만들고 포트(eth1)와 연결합니다.*
   이 예제에서는 br-ex 를 사용하도록 eth1 을 구성합니다.

   예제

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

5. ***openstack overcloud deploy*** *명령을 실행하고 이 새 환경 파일을 포함하여 코어 템플릿과 환경 파일을 포함합니다.*

   중요

   후속 환경 파일에 정의된 매개 변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다.

   예제

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

검증 단계

1. *외부 네트워크를* ***vlan*** *유형으로 생성하고 구성된* ***physical_network*** *와 연결합니다.*
   외부 네트워크를 생성할 때 다른 프로젝트의 사용자가 외부 네트워크를 공유하고 VM 인스턴스를 직접 연결할 수 있도록 ***--shared*** *옵션을 사용합니다.*

   다음 예제 명령을 실행하여 두 개의 네트워크를 생성합니다. 하나는 VLAN 171용이고 다른 하나는 VLAN 172에 사용됩니다.

   예제

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

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

2. 여러 서브넷을 만들고 외부 네트워크를 사용하도록 구성합니다.
openstack subnet create 또는 대시보드를 사용하여 이러한 서브넷을 만들 수 있습니다. 네트워크 관리자가 수신한 외부 서브넷 세부 정보가 각 VLAN에 올바르게 연결되어 있는지 확인합니다.

   이 예제에서 VLAN 171은 서브넷 **10.65.217.0/24** 를 사용하고 VLAN 172는 **10.65.218.0/24** 를 사용합니다.

   예제

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

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

추가 리소스

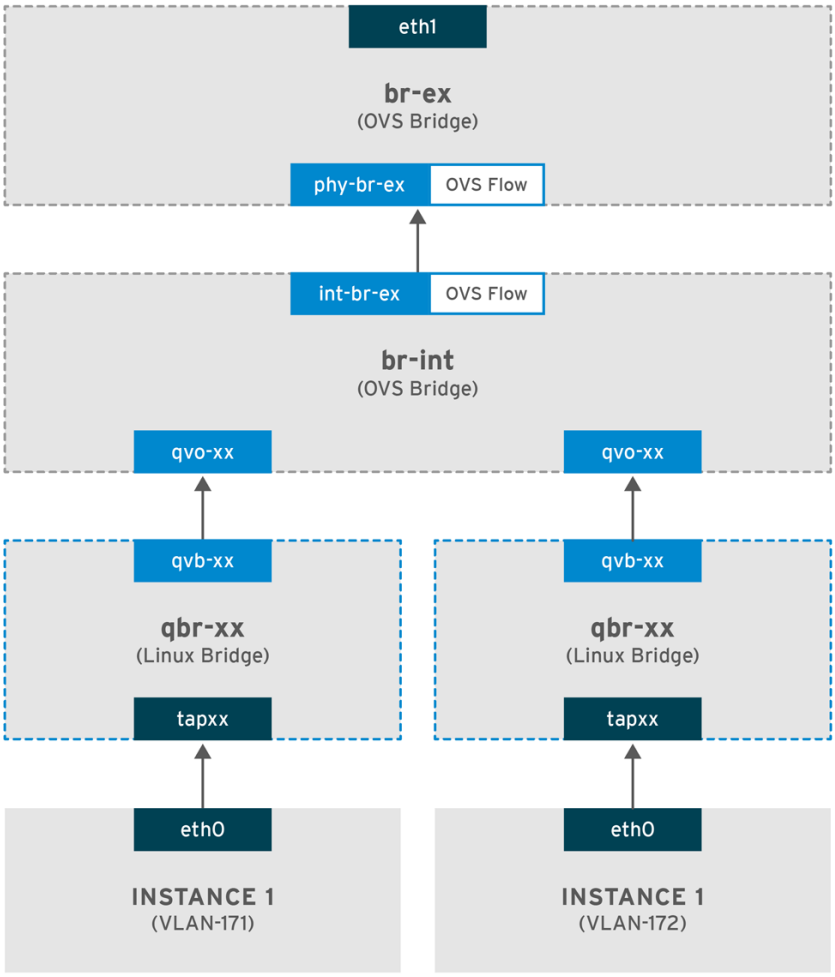
- Advanced Overcloud Customization 가이드의 사용자 지정 네트워크 인터페이스 템플릿
- Advanced Overcloud Customization 가이드의 환경 파일
- Advanced Overcloud Customization 가이드 의 Overcloud 생성에 환경 파일 포함
- 명령줄 인터페이스 참조에서 네트워크 생성
- 명령줄 인터페이스 참조에서 서브넷 생성

7.7. VLAN 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까?

이 섹션에서는 VLAN 프로바이더 네트워크 구성이 있는 인스턴스와의 트래픽 흐름 방법을 자세히 설명합니다.

VLAN 공급자 네트워크에서 나가는 트래픽 흐름

다음 다이어그램에서는 인스턴스를 벗어나고 VLAN 프로바이더 외부 네트워크에 직접 도착하는 트래픽의 패킷 흐름을 설명합니다. 이 예에서는 두 개의 VLAN 네트워크(171 및 172)에 연결된 두 개의 인스턴스를 사용합니다. br-ex 를 구성한 후 물리적 인터페이스를 추가하고 Compute 노드에 인스턴스를 생성합니다. 결과적으로 생성된 인터페이스 및 브리지 구성은 다음 다이어그램의 구성과 유사합니다.



OPENSTACK_450456_0617

1. *인스턴스의 eth0 인터페이스를 나가는 패킷은 인스턴스에 연결된 linux bridge qbr-xx 에 도달합니다.*

2. *qbr-xx 는 veth 쌍 qvbxx <ECDHE qvoxxx 를 사용하여 br-int 에 연결됩니다.*

3. *qvbxx 는 linux bridge qbr-xx 에 연결되고 qvoxx 는 Open vSwitch 브리지 br-int 에 연결됩니다.*

Linux 브리지의 qbr-xx 구성 예.

이 예에서는 2개의 인스턴스와 2개의 해당 linux 브리지를 제공합니다.

```
# brctl show
bridge name bridge id  STP enabled interfaces
qbr84878b78-63  8000.e6b3df9451e0 no  qvb84878b78-63
      tap84878b78-63

qbr86257b61-5d  8000.3a3c888eeae6 no  qvb86257b61-5d
      tap86257b61-5d
```

br-int 의 qvoxx 구성 :

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

            Interface "qvo86257b61-5d"
```

```
    Port "qvo84878b78-63"
        tag: 2
        Interface "qvo84878b78-63"
```

- ***qvoxx*** 는 VLAN 프로바이더 네트워크와 연결된 내부 VLAN 태그로 태그됩니다. 이 예에서 내부 VLAN 태그 2는 VLAN 프로바이더 네트워크 ***provider-171*** 과 연결되며, VLAN 태그 3은 VLAN 프로바이더 네트워크 ***provider-172*** 와 연결됩니다. 패킷이 qvoxx 에 도달하면 이 VLAN 태그가 패킷 헤더에 추가됩니다.

- 그런 다음 patch-peer ***int -br-ex <❑ phy-br-ex*** 를 사용하여 패킷이 ***br -ex*** OVS 브리지로 이동합니다. br-int의 patch- peer 예 :

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

br-ex 의 패치 피어 구성 예 :

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

- 이 패킷이 br -ex에서 phy-br -ex 에 도달하면 br-ex 내부의 OVS 흐름이 내부 VLAN 태그를 VLAN 프로바이더 네트워크와 연결된 실제 VLAN 태그로 교체합니다.

다음 명령의 출력은 phy-br-ex 의 포트 번호가 ***4*** 임을 보여줍니다.

```
# ovs-ofctl show br-ex
 4(phy-br-ex): addr:32:e7:a1:6b:90:3e
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

다음 명령은 VLAN 태그 2(dl_***vlan=2***)가***있는*** ***phy-br-ex(in_port=4***)에 도착하는 패킷을 보여줍니다. Open vSwitch는 VLAN 태그를 171(***actions=mod_vlan_vid:171,NORMAL***)으로 교체하고 패킷을 실제 인터페이스로 전달합니다. 또한 명령은 VLAN 태그 3(dl_***vlan=3***)이***있는*** ***phy-br-ex(in_port=4***)에 도착하는 모든 패킷도 표시합니다. Open vSwitch는 VLAN 태그를 172(***actions=mod_vlan_vid:172,NORMAL***)로 교체하고 패킷을 실제 인터페이스로 전달합니다. neutron-openvswitch-agent는 이러한 규칙을 추가합니다.

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6527.527s, table=0, n_packets=29211, n_bytes=2725576, idle_age=0,
priority=1 actions=NORMAL
 cookie=0x0, duration=2939.172s, table=0, n_packets=117, n_bytes=8296, idle_age=58,
priority=4,in_port=4,dl_vlan=3 actions=mod_vlan_vid:172,NORMAL
```

```
 cookie=0x0, duration=6111.389s, table=0, n_packets=145, n_bytes=9368, idle_age=98,
priority=4,in_port=4,dl_vlan=2 actions=mod_vlan_vid:171,NORMAL
 cookie=0x0, duration=6526.675s, table=0, n_packets=82, n_bytes=6700, idle_age=2462,
priority=2,in_port=4 actions=drop
```

- 그런 다음 이 패킷은 물리적 인터페이스 eth1 로 전달됩니다.

VLAN 공급자 네트워크에서 들어오는 트래픽 흐름

다음 예제 흐름은 프로바이더 네트워크 provider-171에 VLAN 태그 2를 사용하고 프로바이더 네트워크 provider-172의 VLAN 태그 3을 사용하여 계산 노드에서 테스트되었습니다. 흐름에서는 통합 브리지 br-int 에서 18 포트를 사용합니다.

VLAN 프로바이더 네트워크에는 다른 구성이 필요할 수 있습니다. 또한 네트워크의 구성 요구 사항은 두 개의 서로 다른 컴퓨팅 노드마다 다를 수 있습니다.

다음 명령의 출력은 int-br-ex 를 포트 번호 18과 함께 표시합니다.

```
# ovs-ofctl show br-int
 18(int-br-ex): addr:fe:b7:cb:03:c5:c1
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

다음 명령의 출력은 br-int의 흐름 규칙을 보여줍니다.

```
# ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6770.572s, table=0, n_packets=1239, n_bytes=127795, idle_age=106,
priority=1 actions=NORMAL

 cookie=0x0, duration=3181.679s, table=0, n_packets=2605, n_bytes=246456, idle_age=0,
 priority=3,in_port=18,dl_vlan=172 actions=mod_vlan_vid:3,NORMAL

 cookie=0x0, duration=6353.898s, table=0, n_packets=5077, n_bytes=482582, idle_age=0,
 priority=3,in_port=18,dl_vlan=171 actions=mod_vlan_vid:2,NORMAL

 cookie=0x0, duration=6769.391s, table=0, n_packets=22301, n_bytes=2013101, idle_age=0,
priority=2,in_port=18 actions=drop

 cookie=0x0, duration=6770.463s, table=23, n_packets=0, n_bytes=0, idle_age=6770, priority=0
actions=drop
```

수신 흐름 예

이 예제에서는 다음 br-int OVS 흐름을 보여줍니다.

```
cookie=0x0, duration=3181.679s, table=0, n_packets=2605, n_bytes=246456, idle_age=0,
priority=3,in_port=18,dl_vlan=172 actions=mod_vlan_vid:3,NORMAL
```

- 외부 네트워크에서 VLAN 태그 172가 있는 패킷은 물리적 노드의 eth1 을 통해 br-ex 브리지에 도달합니다.
- 패킷은 patch-peer **phy -br-ex <-> int-br-ex**를 통해 **br-int** 로 이동합니다.
- 패킷은 흐름의 기준(**in_port=18,dl_vlan=172**)과 일치합니다.

- 흐름 작업(**actions=mod_vlan_vid:3,NORMAL**)은 VLAN 태그 172를 내부 VLAN 태그 3으로 교체하고 패킷을 일반 계층 2 프로세싱으로 인스턴스에 전달합니다.

추가 리소스

- *VLAN 공급자 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결*

7.8. VLAN 프로바이더 네트워크에서 인스턴스 물리적 네트워크 연결 문제 해결

VLAN 프로바이더 네트워크에서 연결 문제를 해결할 때 "VLAN 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까?"에 설명된 패킷 흐름을 참조하십시오. 또한 다음 구성 옵션을 검토합니다.

절차

1. 물리적 네트워크 이름이 일관되게 사용되는지 확인합니다. 이 예에서는 네트워크를 만드는 동안 및 **bridge_mapping** 구성 내에서 **physnet1** 이 일관되게 사용됩니다.

```
# grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
bridge_mappings = physnet1:br-ex

# openstack network show provider-vlan171
...
| provider:physical_network | physnet1
...
```

2. 네트워크가 **외부로** 생성되었고, 유형 **vlan** 이며 올바른 **segmentation_id** 값을 사용하는지 확인합니다.

```
# openstack network show provider-vlan171
...
| provider:network_type     | vlan                         |
| provider:physical_network | physnet1                     |
| provider:segmentation_id  | 171                          |
...
```

3. **ovs-vsctl show** 를 실행하고 patch -peer **int-br -ex <❑ phy -br-ex를 사용하여 br- int 및 br** -ex 가 연결되어 있는지 확인합니다.
이 연결은 **bridge_mapping이 /etc/neutron/plugins/ml2/ openvswitch_ agent.ini에 올바르게 구성된 경우 neutron- openvswitch-agent.ini** 를 다시 시작하는 동안 생성됩니다.
서비스를 다시 시작한 후에도 생성되지 않은 경우 bridge_mapping 설정을 다시 확인합니다.

4. 나가는 패킷의 흐름을 검토하려면 **ovs-ofctl dump-flows br-ex** 및 **ovs-ofctl dump-flows br-int** 를 실행하고 흐름이 내부 VLAN ID를 외부 VLAN ID(segmentation_id)에 매핑하는지 확인합니다. 들어오는 패킷의 경우 외부 VLAN ID를 내부 VLAN ID에 매핑합니다.
이 흐름은 이 네트워크에 인스턴스를 처음 생성할 때 neutron OVS 에이전트에서 추가합니다. 인스턴스를 생성한 후 이 흐름을 생성하지 않으면 네트워크가 **vlan** 으로 생성되고 **외부** 이며 **physical_network** 이름이 올바른지 확인합니다. 또한 **bridge_mapping** 설정을 다시 확인합니다.

5. 마지막으로 ifcfg-br-ex 및 ifcfg- ethx 구성을 다시 확인합니다. br-ex 에 포트 ethX 가 포함되고 ifcfg-br-ex 및 ifcfg-ethx 모두 **ip a** 명령의 출력에 **UP** 플래그가 있는지 확인합니다.
예를 들어 다음 출력은 eth1 이 br-ex 의 포트임을 보여줍니다.

```
Bridge br-ex
    Port phy-br-ex
```

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

다음 명령은 eth1 이 포트로 추가되었으며, 커널이 인터페이스에서 OVS 브리지 br-ex 로 모든 패킷을 이동하도록 구성되어 있음을 보여줍니다. 이는 **master ovs-system** 항목에서 보여줍니다.

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

추가 리소스

- VLAN 프로바이더 네트워크 패킷 흐름은 어떻게 작동합니까?

7.9. ML2/OVS 배포에서 공급자 네트워크의 멀티 캐스트 스누핑 활성화

RHOSP(Red Hat OpenStack Platform) 공급자 네트워크의 모든 포트에 플러딩 멀티캐스트 패킷을 방지하려면 멀티캐스트 스누핑을 활성화해야 합니다. Open vSwitch 메커니즘 드라이버(ML2/OVS)와 함께 Modular Layer 2 플러그인을 사용하는 RHOSP 배포에서 사용자 지정 환경 파일에 적절한 Puppet 변수를 추가하고 **openstack overcloud deploy** 명령을 실행하여 이 작업을 수행합니다.

중요

프로덕션 환경에 적용하기 전에 멀티캐스트 스누핑 구성을 철저하게 테스트하고 이해해야 합니다. 잘못된 구성으로 멀티캐스팅이 중단되거나 잘못된 네트워크 동작이 발생할 수 있습니다.

사전 요구 사항

- 구성에서는 ML2/OVS 공급자 네트워크만 사용해야 합니다.
- 실제 라우터에 IGMP 스누핑이 활성화되어 있어야 합니다.
  즉, 물리적 라우터에서 OVS(및 물리적 네트워킹)에서 스누핑 캐시를 유지 관리하도록 멀티캐스트 그룹 구성원에서 일반 IGMP가 보고하도록 하려면 프로바이더 네트워크의 IGMP 쿼리 패킷을 보내야 합니다.
- VM 인스턴스(또는 포트 보안 비활성화)에 대한 인바운드 IGMP를 허용하려면 RHOSP Networking 서비스 보안 그룹 규칙이 있어야 합니다.
  이 예에서는 **ping_ssh** 보안 그룹에 대한 규칙이 생성됩니다.

  예제

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

절차

1. Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.

   예제

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

작은 정보

오케스트레이션 서비스(heat)는 templates라는 플랜 집합을 사용하여 환경을 설치하고 구성합니다. heat 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿 파일인 사용자 지정 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다.

2. 적절한 역할의 **ExtraConfig** 섹션에 있는 YAML 환경 파일에서 Puppet 변수 **igmp_snooping_enable** 을 **true** 로 설정합니다.

예제

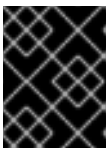
사용된 역할이 **ComputeOvsDpdk** 인 경우 사용자 지정 envirnoment 파일에 추가하는 행은 다음과 같습니다.

```
parameter_defaults:
  ComputeOvsDpdkExtraConfig:
    neutron::agents::ml2::ovs::igmp_snooping_enable: true
```

중요

단일 콜론(:)과 값 사이에 공백 문자를 추가해야 합니다.

3. **openstack overcloud deploy** 명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.



중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

예제

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

검증 단계

- 멀티 캐스트 스누핑이 활성화되었는지 확인합니다.

예제

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

샘플 출력

```
...
mcast_snooping_enable: true
```

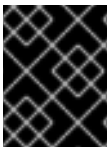
```
...
other_config: {mac-table-size="50000", mcast-snooping-disable-flood-unregistered=True}
...
```

추가 리소스

- Red Hat OpenStack Platform의 구성 요소, 플러그인 및 드라이버 지원의 Neutron
- Advanced Overcloud Customization 가이드의 환경 파일
- Advanced Overcloud Customization 가이드 의 Overcloud 생성에 환경 파일 포함

7.10. ML2/OVN 배포에서 멀티 캐스트 활성화

멀티 캐스트 트래픽을 지원하려면 멀티 캐스트 트래픽이 멀티캐스트 그룹의 가상 머신(VM) 인스턴스에 도달할 수 있도록 배포의 보안 구성을 수정합니다. 멀티 캐스트 트래픽 플러핑을 방지하려면 IGMP 스누핑을 활성화하십시오.



중요

멀티 캐스트 스누핑 구성을 테스트 및 이해하고 프로덕션 환경에 적용합니다. 잘못된 구성으로 멀티캐스팅이 중단되거나 잘못된 네트워크 동작이 발생할 수 있습니다.

사전 요구 사항

- ML2/OVN 메커니즘 드라이버를 사용한 OpenStack 배포.

절차

1. 적절한 VM 인스턴스로 멀티 캐스트 트래픽을 허용하도록 보안을 구성합니다. 예를 들어 IGMP querier의 IGMP 트래픽을 허용하는 보안 그룹 규칙과 VM 인스턴스를 입력하고 종료하는 세 번째 규칙과 멀티 캐스트 트래픽을 허용하는 세 번째 규칙을 만듭니다.

   예제

   보안 그룹 mySG 를 사용하면 IGMP 트래픽이 VM 인스턴스에 들어가 종료할 수 있습니다.

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

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

   또 다른 규칙을 사용하면 멀티 캐스트 트래픽이 VM 인스턴스에 도달할 수 있습니다.

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

   보안 그룹 규칙을 설정하는 대신 일부 Operator는 네트워크에서 포트 보안을 선택적으로 비활성화하도록 선택합니다. 포트 보안을 비활성화하도록 선택하는 경우 관련 보안 위험을 고려하고 계획하십시오.

2. heat 매개변수 **NeutronEnableIgmpSnooping을 설정합니다. 언더클라우드 노드의 환경 파일에서 true** 예를 들어 ovn-extras.yaml에 다음 행을 추가합니다.

   예제

```
parameter_defaults:
      NeutronEnableIgmpSnooping: True
```

3. 해당 환경과 관련된 기타 환경 파일과 함께 **openstack overcloud deploy** 명령에 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
$ openstack overcloud deploy \
--templates \
…
-e <other_overcloud_environment_files> \

-e ovn-extras.yaml \
…
```

<other_overcloud_environment_files> 를 기존 배포의 일부인 환경 파일 목록으로 바꿉니다.

검증 단계

1. 멀티 캐스트 스누핑이 활성화되었는지 확인합니다. northbound 데이터베이스 Logical_Switch 테이블을 나열합니다.

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

샘플 출력

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

Networking 서비스(neutron) igmp_snooping_enable 구성은 OVN Northbound 데이터베이스의 Logical_Switch 테이블에 있는 other_config 열에 설정된 mcast_snoop 옵션으로 변환됩니다. mcast_flood_unregistered는 항상 "false" 입니다.

2. IGMP 그룹을 표시합니다.

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

샘플 출력

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

추가 리소스

- Red Hat OpenStack Platform의 구성 요소, 플러그인 및 드라이버 지원의 Neutron
- Advanced Overcloud Customization 가이드의 환경 파일

- Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함

7.11. 컴퓨팅 메타데이터 액세스 활성화

이 장에 설명된 대로 연결된 인스턴스는 프로바이더 외부 네트워크에 직접 연결되며, 기본 게이트웨이로 외부 라우터가 구성되어 있습니다. OpenStack Networking(neutron) 라우터가 사용되지 않습니다. 즉, neutron 라우터를 사용하여 인스턴스에서 nova-metadata 서버로 메타데이터 요청을 프록시할 수 없으므로 cloud- init 를 실행하는 동안 오류가 발생할 수 있습니다. 그러나 이 문제는 메타데이터 요청을 프록시하도록 dhcp 에이전트를 구성하여 해결할 수 있습니다. **/etc/neutron/dhcp_agent.ini** 에서 이 기능을 활성화할 수 있습니다. 예를 들면 다음과 같습니다.

```
enable_isolated_metadata = True
```

7.12. 부동 IP 주소

유동 IP가 이미 사설 네트워크와 연결되어 있는 경우에도 동일한 네트워크를 사용하여 인스턴스에 유동 IP 주소를 할당할 수 있습니다. 이 네트워크에서 유동 IP로 할당한 주소는 네트워크 노드의 qrouter-xxx 네임스페이스에 바인딩되고 연결된 개인 IP 주소에 DNAT-SNAT 를 수행합니다. 반대로 외부 네트워크 액세스를 위해 할당한 IP 주소는 인스턴스 내부에서 직접 바인딩되며 인스턴스에서 외부 네트워크와 직접 통신할 수 있습니다.

8장. OPENSTACK NETWORKING의 물리적 스위치 구성

이 장에서는 OpenStack Networking에 필요한 일반적인 실제 스위치 구성 단계를 설명합니다. 특정 스위치에는 벤더별 구성이 포함됩니다.

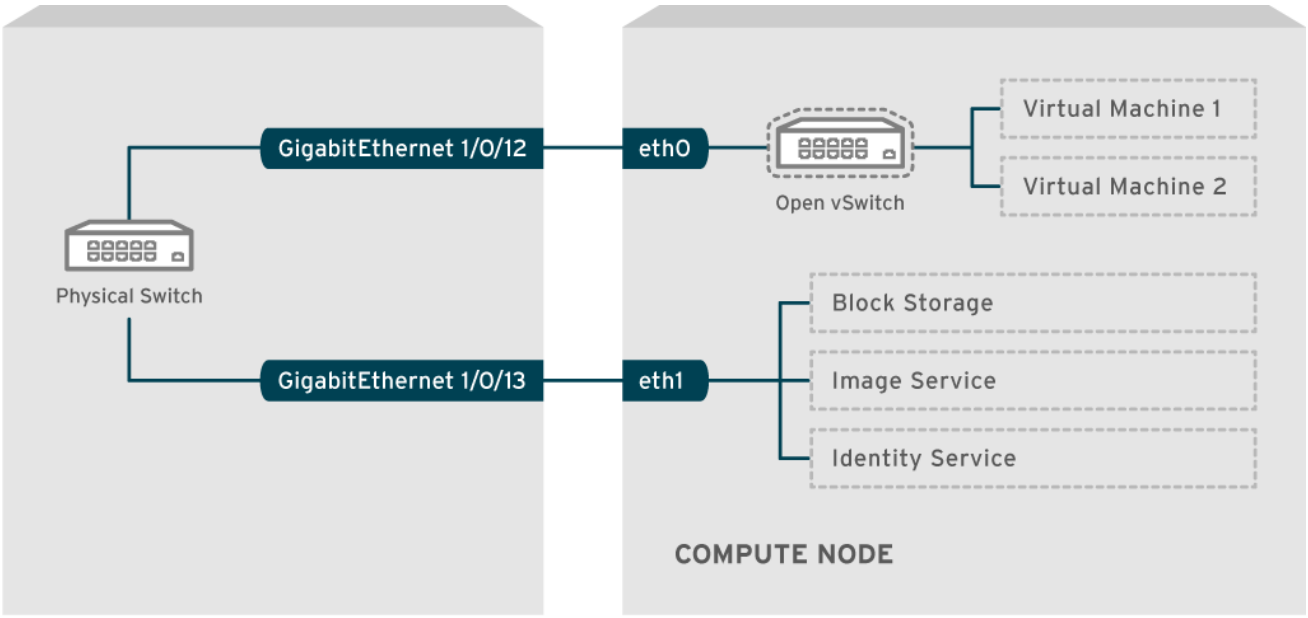
8.1. 물리적 네트워크 환경 계획

OpenStack 노드의 실제 네트워크 어댑터는 인스턴스 트래픽, 스토리지 데이터 또는 인증 요청과 같은 다양한 유형의 네트워크 트래픽을 전달합니다. 이러한 NIC가 전송하는 트래픽 유형은 물리적 스위치에서 포트를 구성해야 하는 방법에 영향을 미칩니다.

먼저 어떤 유형의 트래픽이 전달될 컴퓨팅 노드의 물리적 NIC를 결정해야 합니다. 그런 다음 NIC가 물리적 스위치 포트에 케이블링되면 트렁크 또는 일반 트래픽을 허용하도록 스위치 포트를 구성해야 합니다.

예를 들어 다음 다이어그램에서는 두 개의 NIC, eth0 및 eth1이 있는 컴퓨팅 노드를 보여줍니다. 각 NIC는 물리적 스위치의 기가비트 이더넷 포트로 연결되며 eth0은 인스턴스 트래픽을 전달하고 eth1은 OpenStack 서비스에 대한 연결을 제공합니다.

그림 8.1. 네트워크 레이아웃 샘플



참고

이 다이어그램에는 내결함성에 필요한 추가 중복 NIC가 포함되지 않습니다.

네트워크 인터페이스 본딩에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

8.2. CISCO CATALYST 스위치 구성

8.2.1. 트렁크 포트 정보

OpenStack 네트워킹을 사용하면 인스턴스를 실제 네트워크에 이미 존재하는 VLAN에 연결할 수 있습니다. trunk 이라는 용어는 여러 VLAN이 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다. 이러한 포트를 사용하면 VLAN이 가상 스위치를 비롯한 여러 스위치 간에 확장할 수 있습니다. 예를 들어 실제

네트워크의 VLAN110으로 태그된 트래픽이 계산 노드에 도달합니다. 여기서 8021q 모듈은 태그된 트래픽을 vSwitch의 적절한 VLAN으로 전달합니다.

8.2.2. Cisco catalyst 스위치의 트렁크 포트 구성

- *Cisco catalyst 스위치를 사용하는 경우 다음 구성 구문을 사용하여 VLAN 110 및 111의 트래픽이 인스턴스로 전달될 수 있습니다.*
  이 구성에서는 물리적 노드에 실제 스위치의 인터페이스 GigabitEthernet1/0/12에 연결된 이더넷 케이블이 있다고 가정합니다.

중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

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

다음 목록을 사용하여 이러한 매개변수를 파악합니다.

필드	설명
interface GigabitEthernet1/0/12	X 노드의 NIC가 연결되는 스위치 포트입니다. **GigabitEthernet1/0/12** 값을 사용자 환경에 대한 올바른 포트 값으로 교체해야 합니다. show interface 명령을 사용하여 포트 목록을 확인합니다.
컴퓨팅 노드 간략 정보	이 인터페이스를 식별하는 데 사용할 수 있는 고유하고 설명적인 값입니다.
spanning-tree portfast 트렁크	환경에서 STP를 사용하는 경우 이 값을 설정하여 이 포트가 트래픽을 트렁크하는 데 사용되는 Port Fast에 지시합니다.
switchport 트렁크 캡슐화 dot1q	802.1q 트렁킹 표준(ISL이 아닌) 활성화. 이 값은 스위치에서 지원하는 구성에 따라 다릅니다.
Switchport 모드 트렁크	이 포트를 액세스 포트 대신 트렁크 포트로 구성합니다. 즉 VLAN 트래픽이 가상 스위치로 전달할 수 있습니다.
Switchport 트렁크 네이티브 vlan 2	기본 VLAN을 설정하여 태그가 지정되지 않은 (VLAN 제외) 트래픽을 보낼 스위치에 지시합니다.

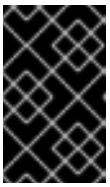
필드	설명
Switchport 트렁크 허용 vlan 2,110,111	트렁크를 통해 허용되는 VLAN을 정의합니다.

8.2.3. 액세스 포트 정보

Compute 노드의 모든 NIC가 인스턴스 트래픽을 전송하는 것은 아니므로 여러 VLAN이 통과할 수 있도록 모든 NIC를 구성할 필요가 없습니다. 액세스 포트에는 하나의 VLAN만 필요하며, 관리 트래픽 또는 블록 스토리지 데이터 전송과 같은 다른 운영 요구 사항을 충족할 수 있습니다. 이러한 포트는 일반적으로 액세스 포트라고 하며, 일반적으로 트렁크 포트보다 더 간단한 구성이 필요합니다.

8.2.4. Cisco catalyst 스위치의 액세스 포트 구성

- *그림 8.1. "네트워크 레이아웃 샘플" 다이어그램의 예제를 사용하여 GigabitEthernet1/0/13(Cisco catalyst 스위치에서)은 **eth1** 의 액세스 포트로 구성됩니다.*
  이 구성에서 물리적 노드에는 실제 스위치의 GigabitEthernet1/0/12 인터페이스에 연결된 이더넷 케이블이 있습니다.



중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

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

이러한 설정은 다음과 같습니다.

필드	설명
interface GigabitEthernet1/0/13	X 노드의 NIC가 연결되는 스위치 포트입니다. **GigabitEthernet1/0/12** 값을 사용자 환경에 대한 올바른 포트 값으로 교체해야 합니다. show interface 명령을 사용하여 포트 목록을 확인합니다.
컴퓨팅 노드의 액세스 포트 설명	이 인터페이스를 식별하는 데 사용할 수 있는 고유하고 설명적인 값입니다.
스위치 포트 모드 액세스	이 포트를 트렁크 포트 대신 액세스 포트로 구성합니다.
Switchport 액세스 vlan 200	VLAN 200에서 트래픽을 허용하도록 포트를 구성합니다. 이 VLAN의 IP 주소를 사용하여 컴퓨팅 노드를 구성해야 합니다.

필드	설명
spaning-tree portfast	STP를 사용하는 경우 STP가 트렁크로 초기화하지 않도록 이 값을 설정하여 초기 연결(예: 서버 재부팅) 중에 더 빠른 포트 핸드셰이크를 허용합니다.

8.2.5. LACP 포트 집계 정보

LACP를 사용하여 여러 물리적 NIC를 함께 번들하여 단일 논리적 채널을 구성할 수 있습니다. 또한 802.3ad(또는 Linux의 본딩 모드 4)라고도 하는 LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 만듭니다. 실제 NIC 및 물리적 스위치 포트에 있는 물리적 엔드 모두에서 LACP를 구성해야 합니다.

8.2.6. 물리적 NIC 에서 LACP 구성

1. */home/stack/network-environment.yaml 파일을 편집합니다.*

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. *LACP를 사용하도록 Open vSwitch 브리지를 구성합니다.*

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

네트워크 본딩 구성에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

8.2.7. Cisco catalyst 스위치의 LACP 구성

이 예제에서 Compute 노드에는 VLAN 100을 사용하는 두 개의 NIC가 있습니다.

절차

1. *컴퓨팅 노드의 두 NIC를 모두 스위치에 물리적으로 연결합니다(예: 포트 12 및 13).*

2. *LACP 포트 채널을 생성합니다.*

```
interface port-channel1
  switchport access vlan 100
  switchport mode access
```

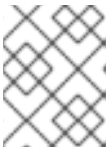
```
spanning-tree guard root
```

3. *스위치 포트 12(Gi1/0/12) 및 13(Gi1/0/13)을 구성합니다.*

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

sw01(config) interface GigabitEthernet1/0/12
  switchport access vlan 100
  switchport mode access
  speed 1000
  duplex full
  channel-group 10 mode active
  channel-protocol lacp

interface GigabitEthernet1/0/13
  switchport access vlan 100
  switchport mode access
  speed 1000
  duplex full
  channel-group 10 mode active
  channel-protocol lacp
```

4. *새 포트 채널을 검토합니다. 결과 출력에는 멤버 포트 Gi1/**0/12 및 Gi 1/0/13이 포함된 새 포트-채널 Po 1** 이 나열됩니다.*

```
sw01# show etherchannel summary
<snip>

Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------------------
1      Po1(SD)         LACP      Gi1/0/12(D)  Gi1/0/13(D)
```



참고

*running-config를 startup-config에 복사하여 변경 사항을 적용하십시오. **running-config startup-config를 복사합니다**.*

8.2.8. MTU 설정 정보

특정 유형의 네트워크 트래픽에 맞게 MTU 크기를 조정해야 합니다. 예를 들어 특정 NFS 또는 iSCSI 트래픽에는 점보 프레임(9000바이트)이 필요합니다.

참고

가상 스위치를 포함하여 트래픽이 통과할 것으로 예상되는 모든 홉의 엔드 투 엔드에서 MTU 설정을 변경해야 합니다.

추가 리소스

- *최대 전송 단위(MTU) 설정 구성*

8.2.9. Cisco catalyst 스위치의 MTU 설정 구성

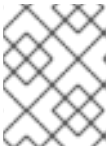
Cisco Catalyst 3750 스위치에서 점보 프레임을 활성화하려면 이 예제 절차의 단계를 완료합니다.

1. 현재 MTU 설정을 검토합니다.

```
sw01# show system mtu

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

2. MTU 설정은 3750 스위치에서 개별 인터페이스에는 변경되지 않습니다. 다음 명령을 실행하여 점보 프레임 9000 바이트를 사용하도록 스위치를 구성합니다. 스위치에서 이 기능을 지원하는 경우 개별 인터페이스에 대한 MTU 설정을 구성하려는 경우가 있습니다.

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

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



참고

running-config 를 startup-config 에 복사하여 변경 사항을 저장하십시오. **running-config startup-config 를 복사합니다**.

3. 스위치를 다시 로드하여 변경 사항을 적용합니다.

중요

스위치를 다시 로드하면 스위치에 종속된 모든 장치에 대해 네트워크 중단이 발생합니다. 따라서 예약된 유지 관리 기간 동안만 스위치를 다시 로드합니다.

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

4. 스위치가 다시 로드된 후 새 점보 MTU 크기를 확인합니다.
정확한 출력은 스위치 모델에 따라 다를 수 있습니다. 예를 들어 **시스템 MTU** 는 기가비트가 아닌 인터페이스에 적용할 수 있으며 **Jumbo MTU** 는 모든 기가비트 인터페이스를 설명할 수 있습니다.

```
sw01# show system mtu

System MTU size is 1600 bytes
System Jumbo MTU size is 9000 bytes
System Alternate MTU size is 1600 bytes
Routing MTU size is 1600 bytes
```

8.2.10. LLDP 검색 정보

ironic-python-agent 서비스는 연결된 스위치에서 LLDP 패킷을 수신 대기합니다. 수집된 정보에는 스위치 이름, 포트 세부 정보, 사용 가능한 VLAN이 포함될 수 있습니다. CDP(Cisco Discovery Protocol)와 유사하게 LLDP는 director 인트로스펙션 프로세스 중에 물리적 하드웨어 검색을 지원합니다.

8.2.11. Cisco catalyst 스위치의 LLDP 구성

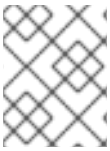
절차

1. Cisco catalyst 스위치에서 LLDP를 전역적으로 활성화하려면 **lldp run 명령을 실행합니다**.

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

sw01(config)# lldp run
```

2. 인접 LLDP 호환 장치를 확인합니다.

```
sw01# show lldp neighbor
Capability codes:
   (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
   (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID          Local Intf     Hold-time  Capability      Port ID
DEP42037061562G3     Gi1/0/11       180        B,T             422037061562G3:P1

Total entries displayed: 1
```



참고

running-config를 startup-config에 복사하여 변경 사항을 저장하십시오. **running-config startup-config를 복사합니다**.

8.3. CISCO NEXUS 스위치 구성

8.3.1. 트렁크 포트 정보

OpenStack 네트워킹을 사용하면 인스턴스를 실제 네트워크에 이미 존재하는 VLAN에 연결할 수 있습니다. trunk 이라는 용어는 여러 VLAN이 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다. 이러한 포트를 사용하면 VLAN이 가상 스위치를 비롯한 여러 스위치 간에 확장할 수 있습니다. 예를 들어 실제 네트워크의 VLAN110으로 태그된 트래픽이 계산 노드에 도달합니다. 여기서 8021q 모듈은 태그된 트래픽을 vSwitch의 적절한 VLAN으로 전달합니다.

8.3.2. Cisco Nexus 스위치의 트렁크 포트 구성

- Cisco Nexus를 사용하는 경우 다음 구성 구문을 사용하여 VLAN 110 및 111의 트래픽이 인스턴스로 전달될 수 있습니다.
  이 구성에서는 물리적 노드에 물리적 스위치의 인터페이스 **Ethernet1/12에 연결된 이더넷** 케이블이 있다고 가정합니다.

중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

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

8.3.3. 액세스 포트 정보

Compute 노드의 모든 NIC가 인스턴스 트래픽을 전송하는 것은 아니므로 여러 VLAN이 통과할 수 있도록 모든 NIC를 구성할 필요가 없습니다. 액세스 포트에는 하나의 VLAN만 필요하며, 관리 트래픽 또는 블록 스토리지 데이터 전송과 같은 다른 운영 요구 사항을 충족할 수 있습니다. 이러한 포트는 일반적으로 액세스 포트라고 하며, 일반적으로 트렁크 포트보다 더 간단한 구성이 필요합니다.

8.3.4. Cisco Nexus 스위치의 액세스 포트 구성

절차

- *그림 8.1. "네트워크 레이아웃 샘플" 다이어그램의 예제를 사용하여 Ethernet1/13(Cisco Nexus 스위치에서)이 **eth1** 의 액세스 포트로 구성됩니다. 이 구성에서는 물리적 노드에 물리적 스위치의 인터페이스 **Ethernet1/13에 연결된 이더넷** 케이블이 있다고 가정합니다.*

중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

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

8.3.5. LACP 포트 집계 정보

LACP를 사용하여 여러 물리적 NIC를 함께 번들하여 단일 논리적 채널을 구성할 수 있습니다. 또한 802.3ad(또는 Linux의 본딩 모드 4)라고도 하는 LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 만듭니다. 실제 NIC 및 물리적 스위치 포트에 있는 물리적 엔드 모두에서 LACP를 구성해야 합니다.

8.3.6. 물리적 NIC에서 LACP 구성

1. */home/stack/network-environment.yaml 파일을 편집합니다.*

```
- type: linux_bond
  name: bond1
  mtu: 9000
```

```
bonding_options:{get_param: BondInterfaceOvsOptions};
members:
  - type: interface
    name: nic3
    mtu: 9000
    primary: true
  - type: interface
    name: nic4
    mtu: 9000
```

2. LACP를 사용하도록 Open vSwitch 브리지를 구성합니다.

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

네트워크 본딩 구성에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

8.3.7. Cisco Nexus 스위치에 대한 LACP 구성

이 예제에서 Compute 노드에는 VLAN 100을 사용하는 두 개의 NIC가 있습니다.

절차

1. 컴퓨팅 노드 NIC를 스위치에 물리적으로 연결합니다(예: 포트 12 및 13).
2. LACP가 활성화되었는지 확인합니다.

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

3. 포트 1/12 및 1/13을 액세스 포트로 구성하고 채널 그룹의 구성원으로 구성합니다.
   배포에 따라 액세스 인터페이스가 아닌 트렁크 인터페이스를 배포할 수 있습니다.

   예를 들어 Cisco UCI의 경우 NIC는 가상 인터페이스이므로 액세스 포트를 독점적으로 구성하는 것이 좋습니다. 대체로 이러한 인터페이스에 VLAN 태그 지정 구성이 포함됩니다.

```
interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 channel-group 10 mode active

interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 channel-group 10 mode active
```

8.3.8. MTU 설정 정보

특정 유형의 네트워크 트래픽에 맞게 MTU 크기를 조정해야 합니다. 예를 들어 특정 NFS 또는 iSCSI 트래픽에는 점보 프레임(9000 바이트)이 필요합니다.

참고

가상 스위치를 포함하여 트래픽이 통과할 것으로 예상되는 모든 홉의 엔드 투 엔드에서 MTU 설정을 변경해야 합니다.

추가 리소스

- *최대 전송 단위(MTU) 설정 구성*

8.3.9. Cisco Nexus 7000 스위치의 MTU 설정 구성

7000 시리즈 스위치의 단일 인터페이스에 MTU 설정을 적용합니다.

절차

- *다음 명령을 실행하여 9000 바이트의 점보 프레임을 사용하도록 인터페이스 1/12를 구성합니다.*

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

8.3.10. LLDP 검색 정보

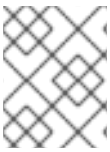
***ironic-python-agent** 서비스는 연결된 스위치에서 LLDP 패킷을 수신 대기합니다. 수집된 정보에는 스위치 이름, 포트 세부 정보, 사용 가능한 VLAN이 포함될 수 있습니다. CDP(Cisco Discovery Protocol)와 유사하게 LLDP는 director 인트로스펙션 프로세스 중에 물리적 하드웨어 검색을 지원합니다.*

8.3.11. Cisco Nexus 7000 스위치의 LLDP 구성

절차

- *Cisco Nexus 7000 시리즈 스위치의 개별 인터페이스에 LLDP를 활성화할 수 있습니다.*

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

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



참고

*running-config를 startup-config에 복사하여 변경 사항을 저장하십시오. **running-config startup-config를 복사합니다**.*

8.4. CUMULAS LINUX 스위치 구성

8.4.1. 트렁크 포트 정보

OpenStack 네트워킹을 사용하면 인스턴스를 실제 네트워크에 이미 존재하는 VLAN에 연결할 수 있습니다. trunk 이라는 용어는 여러 VLAN이 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다. 이러한 포트를 사용하면 VLAN이 가상 스위치를 비롯한 여러 스위치 간에 확장할 수 있습니다. 예를 들어 실제 네트워크의 VLAN110으로 태그된 트래픽이 계산 노드에 도달합니다. 여기서 8021q 모듈은 태그된 트래픽을 vSwitch의 적절한 VLAN으로 전달합니다.

8.4.2. Cumlbs Linux 스위치의 트렁크 포트 구성

이 구성에서는 물리적 노드에 실제 스위치에서 포트 swp1 및 swp2를 전환하는 데 연결된 트랜시버가 있다고 가정합니다.

중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

절차

- 다음 구성 구문을 사용하여 VLAN 100 및 200의 트래픽이 인스턴스로 전달되도록 허용합니다.

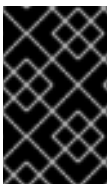
```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200
```

8.4.3. 액세스 포트 정보

Compute 노드의 모든 NIC가 인스턴스 트래픽을 전송하는 것은 아니므로 여러 VLAN이 통과할 수 있도록 모든 NIC를 구성할 필요가 없습니다. 액세스 포트에는 하나의 VLAN만 필요하며, 관리 트래픽 또는 블록 스토리지 데이터 전송과 같은 다른 운영 요구 사항을 충족할 수 있습니다. 이러한 포트는 일반적으로 액세스 포트라고 하며, 일반적으로 트렁크 포트보다 더 간단한 구성이 필요합니다.

8.4.4. Cumlbs Linux 스위치의 액세스 포트 구성

이 구성에서는 물리적 노드에 실제 스위치의 인터페이스에 연결된 이더넷 케이블이 있다고 가정합니다. Cumuls Linux 스위치는 관리 인터페이스의 경우 **eth** 를 사용하고 액세스/트렁크 포트에는 **swp** 를 사용합니다.



중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

절차

- 그림 8.1. "네트워크 레이아웃 샘플" 다이어그램의 예제를 사용하여 **swp1(**Cumlbs Linux 스위치에서)이 액세스 포트로 구성됩니다.

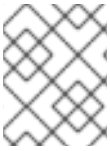
```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200


auto swp1
iface swp1
  bridge-access 100


auto swp2
iface swp2
  bridge-access 200
```

8.4.5. LACP 포트 집계 정보

LACP를 사용하여 여러 물리적 NIC를 함께 번들하여 단일 논리적 채널을 구성할 수 있습니다. 또한 802.3ad(또는 Linux의 본딩 모드 4)라고도 하는 LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 만듭니다. 실제 NIC 및 물리적 스위치 포트에 있는 물리적 엔드 모두에서 LACP를 구성해야 합니다.

8.4.6. MTU 설정 정보

특정 유형의 네트워크 트래픽에 맞게 MTU 크기를 조정해야 합니다. 예를 들어 특정 NFS 또는 iSCSI 트래픽에는 점보 프레임(9000 바이트)이 필요합니다.



참고

가상 스위치를 포함하여 트래픽이 통과할 것으로 예상되는 모든 홉의 엔드 투 엔드에서 MTU 설정을 변경해야 합니다.

추가 리소스

- 최대 전송 단위(MTU) 설정 구성

8.4.7. Cumlbs Linux 스위치의 MTU 설정 구성

절차

- 이 예에서는 Cumlbs Linux 스위치에서 점보 프레임을 활성화합니다.

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

참고

업데이트된 구성을 다시 로드하여 변경 사항을 적용하십시오. **sudo ifreload -a**

8.4.8. LLDP 검색 정보

ironic-python-agent 서비스는 연결된 스위치에서 LLDP 패킷을 수신 대기합니다. 수집된 정보에는 스위치 이름, 포트 세부 정보, 사용 가능한 VLAN이 포함될 수 있습니다. CDP(Cisco Discovery Protocol)와 유사하게 LLDP는 director 인트로스펙션 프로세스 중에 물리적 하드웨어 검색을 지원합니다.

8.4.9. Cumlbs Linux 스위치에 대한 LLDP 구성

기본적으로 LLDP 서비스 lldpd는 데몬으로 실행되며 스위치가 부팅될 때 시작됩니다.

절차

- 모든 포트/인터페이스에서 LLDP 인접 항목을 모두 보려면 다음 명령을 실행합니다.

```
cumulus@switch$ netshow lldp
Local Port  Speed  Mode        Remote Port   Remote Host Summary
----------  ---    ----------    -----  -----  ----------- --------
eth0        10G    Mgmt        ====   swp6   mgmt-sw     IP: 10.0.1.11/24
swp51       10G    Interface/L3 ====   swp1   spine01     IP: 10.0.0.11/32
swp52       10G    Interface/L  ====   swp1   spine02     IP: 10.0.0.11/32
```

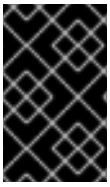
8.5. EXOS 스위치 구성

8.5.1. 트렁크 포트 정보

OpenStack 네트워킹을 사용하면 인스턴스를 실제 네트워크에 이미 존재하는 VLAN에 연결할 수 있습니다. trunk 이라는 용어는 여러 VLAN이 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다. 이러한 포트를 사용하면 VLAN이 가상 스위치를 비롯한 여러 스위치 간에 확장할 수 있습니다. 예를 들어 실제 네트워크의 VLAN110으로 태그된 트래픽이 계산 노드에 도달합니다. 여기서 8021q 모듈은 태그된 트래픽을 vSwitch의 적절한 VLAN으로 전달합니다.

8.5.2. network EXOS 스위치에서 트렁크 포트 구성

X-670 시리즈 스위치를 사용하는 경우 다음 예제를 참조하여 VLAN 110 및 111의 트래픽이 인스턴스로 전달되도록 허용합니다.



중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

절차

- 이 구성에서는 물리적 노드에 실제 스위치의 인터페이스 24에 연결된 이더넷 케이블이 있다고 가정합니다. 이 예에서 DATA 및 MNGT는 VLAN 이름입니다.

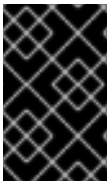
```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

8.5.3. 액세스 포트 정보

Compute 노드의 모든 NIC가 인스턴스 트래픽을 전송하는 것은 아니므로 여러 VLAN이 통과할 수 있도록 모든 NIC를 구성할 필요가 없습니다. 액세스 포트에는 하나의 VLAN만 필요하며, 관리 트래픽 또는 블록 스토리지 데이터 전송과 같은 다른 운영 요구 사항을 충족할 수 있습니다. 이러한 포트는 일반적으로 액세스 포트라고 하며, 일반적으로 트렁크 포트보다 더 간단한 구성이 필요합니다.

8.5.4. Networks EXOS(네트워크 EXOS) 스위치의 액세스 포트 구성

*이 구성에서는 물리적 노드에 실제 스위치의 인터페이스 **10** 에 연결된 이더넷 케이블이 있다고 가정합니다.*



중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

절차

- *이 구성 예제에서는 network X-670 시리즈 스위치에서 **10** 을 **eth1** 의 액세스 포트로 사용합니다.*

```
create vlan VLANNAME tag NUMBER
configure vlan Default delete ports PORTSTRING
configure vlan VLANNAME add ports PORTSTRING untagged
```

예를 들면 다음과 같습니다.

```
#create vlan DATA tag 110
#configure vlan Default delete ports 10
#configure vlan DATA add ports 10 untagged
```

8.5.5. LACP 포트 집계 정보

LACP를 사용하여 여러 물리적 NIC를 함께 번들하여 단일 논리적 채널을 구성할 수 있습니다. 또한 802.3ad(또는 Linux의 본딩 모드 4)라고도 하는 LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 만듭니다. 실제 NIC 및 물리적 스위치 포트에 있는 물리적 엔드 모두에서 LACP를 구성해야 합니다.

8.5.6. 물리적 NIC에서 LACP 구성

1. */home/stack/network-environment.yaml 파일을 편집합니다.*

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. LACP를 사용하도록 Open vSwitch 브리지를 구성합니다.

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

네트워크 본딩 구성에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

8.5.7. network EXOS 스위치에서 LACP 구성

절차

- 이 예제에서 Compute 노드에는 VLAN 100을 사용하는 두 개의 NIC가 있습니다.

```
enable sharing MASTERPORT grouping ALL_LAG_PORTS lacp
configure vlan VLANNAME add ports PORTSTRING tagged
```

예를 들면 다음과 같습니다.

```
#enable sharing 11 grouping 11,12 lacp
#configure vlan DATA add port 11 untagged
```

참고

LACP 협상 스크립트에서 시간제한 기간을 조정해야 할 수도 있습니다. 자세한 내용은 https://gtacknowledge.extremenetworks.com/articles/How_To/LACP-configured-ports-interfere-with-PXE-DHCP-on-servers의 내용을 참조하십시오.

8.5.8. MTU 설정 정보

특정 유형의 네트워크 트래픽에 맞게 MTU 크기를 조정해야 합니다. 예를 들어 특정 NFS 또는 iSCSI 트래픽에는 점보 프레임(9000바이트)이 필요합니다.

참고

가상 스위치를 포함하여 트래픽이 통과할 것으로 예상되는 모든 홉의 엔드 투 엔드에서 MTU 설정을 변경해야 합니다.

추가 리소스

- 최대 전송 단위(MTU) 설정 구성

8.5.9. network EXOS 스위치에서 MTU 설정 구성

절차

- 이 예에서 명령을 실행하여 network EXOS 스위치에서 점보 프레임을 활성화하고 9000바이트의 IP 패킷 전달 지원을 구성합니다.

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

예제

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

8.5.10. LLDP 검색 정보

ironic-python-agent 서비스는 연결된 스위치에서 LLDP 패킷을 수신 대기합니다. 수집된 정보에는 스위치 이름, 포트 세부 정보, 사용 가능한 VLAN이 포함될 수 있습니다. CDP(Cisco Discovery Protocol)와 유사하게 LLDP는 director 인트로스펙션 프로세스 중에 물리적 하드웨어 검색을 지원합니다.

8.5.11. 프리뷰 네트워크 EXOS 스위치에서 LLDP 설정 구성

절차

- 이 예에서 LLDP는 Networks EXOS 스위치에서 활성화됩니다. **11** 은 포트 문자열을 나타냅니다.

```
enable lldp ports 11
```

8.6. JUNIPER EX 시리즈 스위치 구성

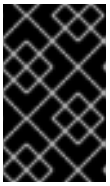
8.6.1. 트렁크 포트 정보

OpenStack 네트워킹을 사용하면 인스턴스를 실제 네트워크에 이미 존재하는 VLAN에 연결할 수 있습니다. trunk 이라는 용어는 여러 VLAN이 동일한 포트를 통과할 수 있는 포트를 설명하는 데 사용됩니다. 이러한 포트를 사용하면 VLAN이 가상 스위치를 비롯한 여러 스위치 간에 확장할 수 있습니다. 예를 들어 실제 네트워크의 VLAN110으로 태그된 트래픽이 계산 노드에 도달합니다. 여기서 8021q 모듈은 태그된 트래픽을 vSwitch의 적절한 VLAN으로 전달합니다.

8.6.2. Juniper EX 시리즈 스위치의 트렁크 포트 구성

절차

- Juniper EX 시리즈가 실행되는 Juniper JunOS를 사용하는 경우 다음 구성 구문을 사용하여 VLAN 110 및 111의 트래픽이 인스턴스로 전달되도록 허용합니다.
  이 구성에서는 물리적 노드에 실제 스위치의 인터페이스 ge-1/0/12에 연결된 이더넷 케이블이 있다고 가정합니다.



중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

```
ge-1/0/12 {
        description Trunk to Compute Node;
            unit 0 {
                family ethernet-switching {
                    port-mode trunk;
                    vlan {
```

```
                    members [110 111];
                    }
                native-vlan-id 2;
            }
        }
    }
```

8.6.3. 액세스 포트 정보

Compute 노드의 모든 NIC가 인스턴스 트래픽을 전송하는 것은 아니므로 여러 VLAN이 통과할 수 있도록 모든 NIC를 구성할 필요가 없습니다. 액세스 포트에는 하나의 VLAN만 필요하며, 관리 트래픽 또는 블록 스토리지 데이터 전송과 같은 다른 운영 요구 사항을 충족할 수 있습니다. 이러한 포트는 일반적으로 액세스 포트라고 하며, 일반적으로 트렁크 포트보다 더 간단한 구성이 필요합니다.

8.6.4. Juniper EX 시리즈 스위치의 액세스 포트 구성

이 예제의 Juniper EX 시리즈 스위치는 **ge-1/0/13** 을 **eth1** 의 액세스 포트로 보여줍니다.

+

중요

이러한 값은 예제입니다. 사용자 환경의 값과 일치하도록 이 예제의 값을 변경해야 합니다. 조정 없이 이러한 값을 스위치 구성에 복사하고 붙여넣으면 예기치 않은 중단이 발생할 수 있습니다.

절차

이 구성에서는 물리적 노드에 실제 스위치의 인터페이스 **ge-1/0/13** 에 연결된 이더넷 케이블이 있다고 가정합니다.

+

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

8.6.5. LACP 포트 집계 정보

LACP를 사용하여 여러 물리적 NIC를 함께 번들하여 단일 논리적 채널을 구성할 수 있습니다. 또한 802.3ad(또는 Linux의 본딩 모드 4)라고도 하는 LACP는 부하 분산 및 내결함성을 위한 동적 본딩을 만듭니다. 실제 NIC 및 물리적 스위치 포트에 있는 물리적 엔드 모두에서 LACP를 구성해야 합니다.

8.6.6. 물리적 NIC 에서 LACP 구성

1. */home/stack/network-environment.yaml* 파일을 편집합니다.

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. *LACP*를 사용하도록 *Open vSwitch* 브리지를 구성합니다.

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

네트워크 본딩 구성에 대한 자세한 내용은 Advanced Overcloud Customization 가이드의 Network Interface Bonding 장을 참조하십시오.

8.6.7. Juniper EX 시리즈 스위치에 대한 LACP 구성

이 예제에서 Compute 노드에는 VLAN 100을 사용하는 두 개의 NIC가 있습니다.

절차

1. *컴퓨팅 노드의 두 NIC를 스위치에 물리적으로 연결합니다(예: 포트 12 및 13).*

2. *포트 집계를 생성합니다.*

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

3. *포트 집계* ***ae1*** *에 참여하도록 스위치 포트 12 (ge-1/0/12) 및 13 (ge-1/0/13)을 구성합니다.*

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

```
        }
    }
}
```

참고

Red Hat OpenStack Platform director 배포의 경우 본딩에서 PXE 부팅을 위해 인트로스펙션 및 첫 번째 부팅 중에 하나의 본딩 멤버만 표시되도록 lacp force-up으로 구성해야 합니다. lacp force-up으로 구성하는 본딩 멤버는 instackenv.json에 MAC 주소가 있는 동일한 본딩 멤버여야 합니다. ironic으로 알려진 MAC 주소는 force-up으로 구성된 것과 동일한 MAC 주소여야 합니다.

4. *포트 집계* ***ae1*** *에서 LACP를 활성화합니다.*

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

5. *VLAN 100* ***에 집계 ae1*** *을 추가합니다.*

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

6. *새 포트 채널을 검토합니다. 결과 출력에는 멤버 포트* ***ge-1/0/12 및 ge-1/0 /13*** *이 있는 새 포트 집계* ***ae1*** *이 나열됩니다.*

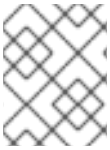
```
> show lacp statistics interfaces ae1

Aggregated interface: ae1
LACP Statistics: LACP Rx LACP Tx Unknown Rx Illegal Rx
ge-1/0/12 0 0 0 0
ge-1/0/13 0 0 0 0
```

참고

커밋 *명령을 실행하여 변경 사항을 적용해야 합니다.*

8.6.8. MTU 설정 정보

특정 유형의 네트워크 트래픽에 맞게 MTU 크기를 조정해야 합니다. 예를 들어 특정 NFS 또는 iSCSI 트래픽에는 점보 프레임(9000 바이트)이 필요합니다.



참고

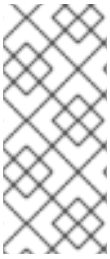
가상 스위치를 포함하여 트래픽이 통과할 것으로 예상되는 모든 홉의 엔드 투 엔드에서 MTU 설정을 변경해야 합니다.

추가 리소스

- *최대 전송 단위(MTU) 설정 구성*

8.6.9. Juniper EX 시리즈 스위치의 MTU 설정 구성

이 예에서는 Juniper EX4200 스위치에서 점보 프레임을 활성화합니다.



참고

*MTU 값은 Juniper 또는 Cisco 장치를 사용하는지에 따라 다르게 계산됩니다. 예를 들어 Juniper의 **9216** 은 Cisco의 경우 **9202** 입니다. 추가 바이트는 L2 헤더에 사용되며, Cisco는 이 값을 지정된 MTU 값에 자동으로 추가하지만 Juniper를 사용할 때 사용 가능한 MTU는 지정된 것보다 14 바이트 더 작습니다. 따라서 VLAN에서 MTU **9000** 을 지원하려면 Juniper에서 **9014** 의 MTU를 구성해야 합니다.*

절차

1. *Juniper EX 시리즈 스위치의 경우 개별 인터페이스에 대해 MTU 설정이 설정됩니다. 이러한 명령은 **ge-1/0/14 및 ge-1/0 /15** 포트에서 점보 프레임을 구성합니다.*

```
set interfaces ge-1/0/14 mtu 9216
set interfaces ge-1/0/15 mtu 9216
```

참고

***commit** 명령을 실행하여 변경 사항을 저장해야 합니다.*

2. *LACP 집계를 사용하는 경우 멤버 NIC가 아닌 MTU 크기를 설정해야 합니다. 예를 들어 이 설정은 ae1 집계의 MTU 크기를 구성합니다.*

```
set interfaces ae1 mtu 9216
```

8.6.10. LLDP 검색 정보

***ironic-python-agent** 서비스는 연결된 스위치에서 LLDP 패킷을 수신 대기합니다. 수집된 정보에는 스위치 이름, 포트 세부 정보, 사용 가능한 VLAN이 포함될 수 있습니다. CDP(Cisco Discovery Protocol)와 유사하게 LLDP는 director 인트로스펙션 프로세스 중에 물리적 하드웨어 검색을 지원합니다.*

8.6.11. Juniper EX 시리즈 스위치에 대한 LLDP 구성

모든 인터페이스에 대해 LLDP를 전역적으로 활성화하거나 개별 인터페이스에 대해서만 활성화할 수 있습니다.

절차

- *Juniper EX 4200 스위치에서 전 세계적으로 LLDP를 다음과 같이 사용하십시오.*

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

- *다음을 사용하여 단일 인터페이스 **ge-1/0/14** 에 LLDP를 활성화합니다.*

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

참고

커밋 *명령을 실행하여 변경 사항을 적용해야 합니다.*

9 장. 최대 전송 단위(MTU) 설정 구성

9.1. MTU 개요

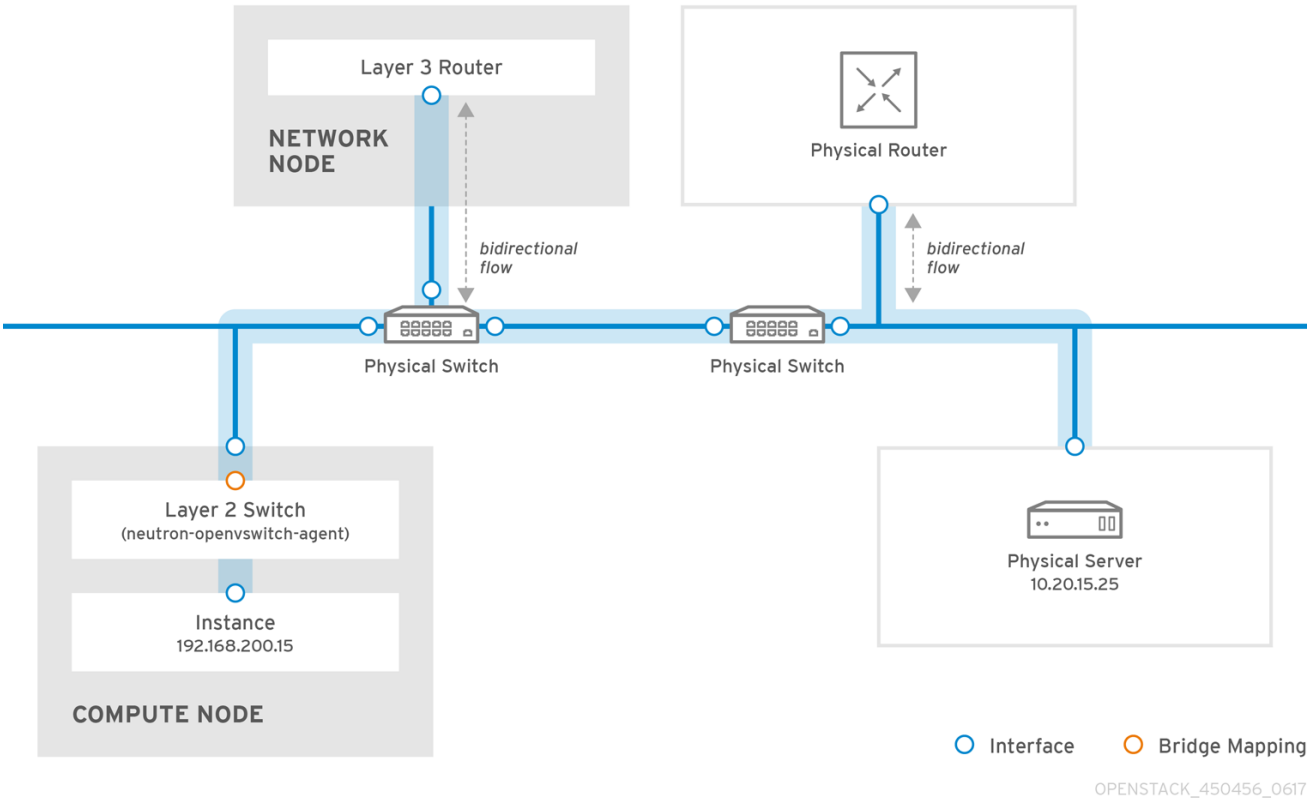
OpenStack Networking 은 인스턴스에 안전하게 적용할 수 있는 가능한 최대 MTU(최대 전송 단위) 크기를 계산할 수 있습니다. MTU 값은 단일 네트워크 패킷이 전송할 수 있는 최대 데이터 양을 지정합니다. 이 숫자는 애플리케이션에 가장 적합한 크기에 따라 다릅니다. 예를 들어 NFS 공유에는 VoIP 애플리케이션보다 다른 MTU 크기가 필요할 수 있습니다.

참고

openstack network show <network_name> 명령을 사용하여 OpenStack Networking 에서 계산하는 가능한 최대 MTU 값을 확인할 수 있습니다. **net-mtu** 는 일부 구현에 없는 neutron API 확장입니다. 필요한 MTU 값은 인스턴스에서 지원하는 경우 자동 구성을 위해 DHCPv4 클라이언트에 알리고 라우터 알림(RA) 패킷을 통해 IPv6 클라이언트에 알릴 수 있습니다. 라우터 알림을 보내려면 네트워크를 라우터에 연결해야 합니다.

엔드 투 엔드에서 MTU 설정을 일관되게 구성해야 합니다. 즉, VM, 가상 네트워크 인프라, 물리적 네트워크 및 대상 서버를 포함하여 패킷이 통과하는 모든 지점에서 MTU 설정이 동일해야 합니다.

예를 들어 다음 다이어그램의 원은 인스턴스와 물리적 서버 간의 트래픽에 맞게 MTU 값을 조정해야 하는 다양한 지점을 나타냅니다. 특정 MTU 크기의 패킷을 수용하도록 네트워크 트래픽을 처리하는 인터페이스의 MTU 값을 변경해야 합니다. 트래픽이 인스턴스 192.168.200.15 에서 실제 서버 10. 20.15.25 로 이동하는 경우 이 작업이 필요합니다.



일관되지 않은 MTU 값은 여러 네트워크 문제가 발생할 수 있으며, 가장 일반적인 패킷 손실로 인해 연결이 끊어지고 네트워크 성능이 저하됩니다. 이러한 문제는 올바른 MTU 값이 있는지 확인하기 위해 가능한 모든 네트워크 지점을 식별하고 검사해야 하므로 문제를 해결하는 데 문제가 있습니다.

9.2. DIRECTOR 에서 MTU 설정 구성

이 예제에서는 NIC 구성 템플릿을 사용하여 MTU를 설정하는 방법을 보여줍니다. 브리지, 본딩(해당되는 경우), 인터페이스 및 VLAN에 MTU를 설정해야 합니다.

```
-
  type: ovs_bridge
  name: br-isolated
  use_dhcp: false
  mtu: 9000    # <--- Set MTU
  members:
    -
      type: ovs_bond
      name: bond1
      mtu: 9000    # <--- Set MTU
      ovs_options: {get_param: BondInterfaceOvsOptions}
      members:
        -
          type: interface
          name: ens15f0
          mtu: 9000    # <--- Set MTU
          primary: true
        -
          type: interface
          name: enp131s0f0
          mtu: 9000    # <--- Set MTU
    -
      type: vlan
      device: bond1
      vlan_id: {get_param: InternalApiNetworkVlanID}
      mtu: 9000    # <--- Set MTU
      addresses:
      -
        ip_netmask: {get_param: InternalApiIpSubnet}
    -
      type: vlan
      device: bond1
      mtu: 9000    # <--- Set MTU
      vlan_id: {get_param: TenantNetworkVlanID}
      addresses:
      -
        ip_netmask: {get_param: TenantIpSubnet}
```

9.3. 결과 MTU 계산 검토

인스턴스가 사용할 수 있는 가장 큰 MTU 값인 계산된 MTU 값을 볼 수 있습니다. 계산된 MTU 값을 사용하여 네트워크 트래픽 경로와 관련된 모든 인터페이스를 구성합니다.

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

10 장. QOS(QUALITY OF SERVICE) 정책 구성

Red Hat OpenStack Platform 네트워크 서비스 품질(QoS) 정책을 사용하면 OpenStack 관리자가 인스턴스의 송신 트래픽에 속도 제한을 적용하여 다양한 서비스 수준을 제공할 수 있습니다. QoS 정책을 구현한 결과 지정된 속도를 초과하는 모든 트래픽이 삭제됩니다.

10.1. QOS 정책 범위

개별 포트 또는 특정 테넌트 네트워크에 QoS 정책을 적용할 수 있습니다. 여기서 특정 정책이 연결되지 않은 포트는 정책을 상속합니다.

10.2. QOS 정책 및 규칙 생성 및 적용

QoS(Quality of Service) 정책 및 규칙을 생성하고 정책을 포트에 적용하려면 다음 단계를 완료합니다.

1. /etc/neutron/plugins/ml2/<agent_name>_agent.ini의 OpenStack 네트워킹에 대해 **qos** 확장이 아직 활성화되어 있지 않은 경우 다음 단계를 따르십시오.

   a. 사용자 지정 Heat 환경 파일을 만들고 다음 행을 추가합니다.

   ```
   parameter_defaults:
     NeutronSriovAgentExtensions: 'qos'
   ```

   중요

   YAML 파일은 파일에서 매개 변수를 배치하는 위치에 대해 매우 민감합니다. **parameter_defaults:** 가 첫 번째 열(위의 공백 문자 없음)에서 시작되고 매개 변수 값 쌍이 3 열에서 시작됩니다(매개 변수 앞에 공백 두 문자가 있음).

   b. Red Hat OpenStack Platform director 명령을 실행하고 **openstack overcloud deploy**를 실행하고 현재 환경 파일과 이 사용자 지정 새 환경 파일을 포함합니다.
   자세한 내용은 Director 설치 및 사용 가이드 의 "Overcloud 환경 수정" 을 참조하십시오.

2. 테넌트 목록을 검토하고 QoS 정책을 생성해야 하는 위치의 ID를 확인합니다.

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

3. **admin** 테넌트에 **bw-limiter** 라는 QoS 정책을 만듭니다.

   ```
   # openstack network qos policy create --project 98a2f53c20ce4d50a40dac4a38016c69 bw-limiter
   ```

4. **bw-limiter** 정책에 대해 policing 규칙을 구성합니다.

```
# openstack network qos rule create --type bandwidth-limit --max-kbps 3000 --max-burst-kbits 30000 bw-limiter
```

5. ***bw-limiter*** *정책을 적용하도록 neutron 포트를 구성합니다.*

```
# openstack port set --qos-policy bw-limiter port_name_or_id
```

6. *QoS 규칙을 검토합니다. 예를 들면 다음과 같습니다.*

```
# openstack network qos policy show 9be535c3-daa2-4d7b-88ea-e8de16

+-------------------+--------------------------------+
| Field             | Value                          |
+-------------------+--------------------------------+
| id                | 9be535c3-daa2-4d7b-88ea-e8de16 |
| rule_type         | bandwidth_limit                |
| description       |                                |
| max_kbps          | 3000                           |
| max_burst_kbps    | 300                            |
+-------------------+--------------------------------+
```

이러한 값을 사용하면 필요에 따라 정책 알고리즘을 구성할 수 있습니다.

- ***max_kbps*** *- 인스턴스가 보낼 수 있는 최대 속도(Kbps) 입니다.*
- ***max_burst_kbps*** *- 토큰 버퍼가 가득 찼을 경우 포트가 즉시 보낼 수 있는 최대 데이터 양(kbits) 입니다. 토큰 버퍼는 "max_kbps" 속도로 사용됩니다.*

10.3. 송신 트래픽에 대한 DSCP 표시

IP 헤더에 관련 값을 포함하여 네트워크에 QoS(Quality-of-service) 정책을 구현하기 위해 차별화된 서비스 코드 지점(DSCP) 정책을 사용할 수 있습니다. Networking 서비스(neutron) QoS 정책은 DSCP 표시를 사용하여 neutron 포트 및 네트워크에서 송신 트래픽을 관리할 수 있습니다.

절차

1. *다음과 같은 경우*
   - *ML2/OVN을 사용하여 2단계로 건너뜁니다.*
   - *터널링 프로토콜 없이 ML2/OVS를 사용하여 2단계로 건너뜁니다.*
   - *터널링 프로토콜(VXLAN 및 GRE)에서 ML2/OVS를 사용한 다음 다음 단계를 수행합니다.*
     a. *언더클라우드 호스트에 **stack** 사용자로 로그인합니다.*

```
$ source ~/stackrc
```

     b. *사용자 지정 YAML 환경 파일을 생성합니다.*

     예제

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

c. ***parameter_defaults*** *의 YAML 환경 파일에서 다음 행을 추가합니다.*

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

dscp_inherit *이* ***true*** *인 경우 Networking 서비스는 내부 헤더의 DSCP 값을 외부 헤더에 복사합니다.*

d. *배포 명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.*

중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

예제

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

2. *자격 증명 파일을 가져옵니다.*

예제

```
$ source ~/overcloudrc
```

3. *새 QoS 정책을 생성합니다.*

예제

```
openstack network qos policy create --project 98a2f53c20ce4d50a40dac4a38016c69 qos-web-servers
```

4. *DSCP 규칙을 생성하고 정책에 적용합니다.*

예제

이 예에서 DSCP 규칙은 DSCP 마크 ***18*** *을 사용하여 생성되며* ***qos-web-servers*** *정책에 적용됩니다.*

```
openstack network qos rule create --type dscp-marking --dscp-mark 18 qos-web-servers
```

샘플 출력

```
Created a new dscp_marking_rule:
+-----------+--------------------------------------+
```

```
| Field     | Value                                |
+-----------+--------------------------------------+
| dscp_mark | 18                                   |
| id        | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----------+--------------------------------------+
```

5. *규칙에 할당된 DSCP 값을 변경할 수 있습니다.*

   예제

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

6. *DSCP 규칙을 삭제할 수 있습니다.*

   예제

   ```
   openstack network qos rule delete qos-web-servers d7f976ec-7fab-4e60-af70-f59bf88198e6
   ```

검증

- *DSCP 규칙(d7f976ec-7fab-4e60-af70-f59bf88198e6)이 QoS 정책(qos-web-servers)에 적용되는지 확인합니다.*

  예제

  ```
  openstack network qos rule list qos-web-servers
  ```

  샘플 출력

  ```
  +-----------+--------------------------------------+
  | dscp_mark | id                                   |
  +-----------+--------------------------------------+
  |        18 | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
  +-----------+--------------------------------------+
  ```

추가 리소스

- *Advanced Overcloud Customization 가이드의 환경 파일*
- *Advanced Overcloud Customization 가이드 의 Overcloud 생성에 환경 파일 포함*
- *명령줄 인터페이스 참조에서 네트워크 qos 규칙 생성*
- *명령줄 인터페이스 참조에서 네트워크 qos 규칙 세트*
- *명령줄 인터페이스 참조에서 네트워크 qos 규칙 삭제*
- *명령줄 인터페이스 참조의 네트워크 qos 규칙 목록*

10.4. QOS 정책의 RBAC

QoS(Quality of Service) 정책에 대해 RBAC(역할 기반 액세스 제어)를 추가할 수 있습니다. 따라서 이제 특정 프로젝트에서 QoS 정책을 사용할 수 있습니다.

*예를 들어 우선순위가 낮은 네트워크 트래픽을 허용하고 특정 프로젝트에만 적용되도록 하는 QoS 정책을 생성할 수 있습니다. 다음 명령을 실행하여 **bw-limiter** 정책을 프로젝트 **demo** 에 할당합니다.*

```
# openstack network rbac create --type qos_policy --target-project
80bf5732752a41128e612fe615c886c6 --action access_as_shared rbac_name
```

11장. 브릿지 매핑 구성

이 장에서는 Red Hat OpenStack Platform에서 브릿지 매핑을 구성하는 방법에 대해 설명합니다.

11.1. 브릿지 매핑 개요

브리지 매핑은 물리적 네트워크 이름(인터페이스 레이블)을 OVS 또는 OVN으로 생성된 브리지에 연결합니다. 이 예에서 실제 이름(datacentre)은 외부 브리지(br-ex)에 매핑됩니다.

```
bridge_mappings = datacentre:br-ex
```

브리지 매핑을 사용하면 프로바이더 네트워크 트래픽이 실제 네트워크에 도달할 수 있습니다. 트래픽은 라우터의 **qg-xxx** 인터페이스에서 프로바이더 네트워크를 벗어나 **br-int** 에 도착합니다. OVS의 경우 **br-int** 와 **br-ex** 간의 패치 포트로 트래픽이 프로바이더 네트워크의 브리지를 통과하고 실제 네트워크로 이동할 수 있습니다. OVN은 포트가 필요한 하이퍼바이저에 바인딩된 VM이 있는 경우에만 하이퍼바이저에 패치 포트를 생성합니다.

라우터가 예약된 네트워크 노드에서 브리지 매핑을 구성합니다. 라우터 트래픽은 프로바이더 네트워크에서 나타내는 올바른 물리적 네트워크를 사용하여 송신할 수 있습니다.

11.2. 트래픽 흐름

각 외부 네트워크는 **qg-xxx** 포트에 태그가 지정된 내부 VLAN ID로 표시됩니다. 패킷이 **phy-br-ex** 에 도달하면 **br-ex** 포트는 VLAN 태그를 제거하고 패킷을 실제 인터페이스로 이동한 다음 외부 네트워크로 이동합니다.

외부 네트워크의 반환 패킷은 **br-ex에 도착하고 phy-br -ex <-> int-br-ex를 사용하여 br-int 로 이동합니다.** 패킷이 **br-ex를 통해 br -int** 로 이동하면 패킷의 외부 vlan ID가 **br-int** 의 내부 vlan 태그로 교체되고, **qg-xxx** 가 패킷을 수락할 수 있습니다.

출력 패킷의 경우 패킷의 내부 vlan 태그는 br-ex(또는 **network_vlan_ranges** 매개 변수에 정의된 외부 브리지)의 외부 vlan 태그로 교체됩니다.

11.3. 브릿지 매핑 구성

RHOSP(Red Hat OpenStack Platform) director는 사전 정의된 NIC 템플릿을 사용하여 초기 네트워킹 구성을 설치하고 구성합니다.

사용자 지정 환경 파일에서 **NeutronBridgeMappings** 매개변수를 사용하여 브리지 매핑과 같은 초기 네트워킹 구성을 사용자 지정할 수 있습니다. **openstack overcloud deploy** 명령에서 환경 파일을 호출합니다.

사전 요구 사항

- 라우터가 예약된 네트워크 노드에서 브리지 매핑을 구성해야 합니다.
- ML2/OVS 및 ML2/OVN DVR 구성의 경우 계산 노드에 대한 브리지 매핑도 구성해야 합니다.

절차

1. 사용자 지정 환경 파일을 만들고 사이트에 적합한 값을 사용하여 **NeutronBridgeMappings** heat 매개변수를 추가합니다.

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
```

NeutronBridgeMappings heat 매개변수는 물리적 이름(datacentre)을 브리지(br**-ex)**에 연결합니다.

참고

NeutronBridgeMappings 매개변수를 사용하지 않는 경우 기본값은 호스트(br-ex)의 외부 브리지를 물리적 이름(datacentre)에 매핑합니다.

2. 이 구성을 적용하려면 오버클라우드를 배포하고 사용자 지정 환경 파일을 다른 환경 파일과 함께 스택에 추가합니다.

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

3. 다음 단계를 수행할 준비가 되었습니다. 이는 다음과 같습니다.

   a. 네트워크 VLAN 범위를 사용하여 해당 외부 네트워크를 나타내는 프로바이더 네트워크를 만듭니다. (neutron 프로바이더 네트워크 또는 유동 IP 네트워크를 만들 때 물리적 이름을 사용합니다.)

   b. 라우터 인터페이스를 사용하여 프로젝트 네트워크에 외부 네트워크를 연결합니다.

추가 리소스

- Advanced Overcloud Customization 가이드의 네트워크 환경 매개변수
- Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함

11.4. OVS에 대한 브리지 매핑 유지 관리

OVS 브리지 매핑을 제거한 후에는 후속 정리를 수행하여 브리지 구성이 연결된 패치 포트 항목으로 지워야 합니다. 다음 방법으로 이 작업을 수행할 수 있습니다.

- 수동 포트 정리 - 수퍼유저 패치 포트를 신중하게 제거해야 합니다. 네트워크 연결 중단이 필요하지 않습니다.
- 자동화된 포트 정리 - 자동화된 정리를 수행하지만 중단이 필요하며 필요한 브릿지 매핑을 다시 추가해야 합니다. 네트워크 연결 중단을 허용할 수 있는 경우 예약된 유지 관리 기간 동안 이 옵션을 선택합니다.

참고

OVN 브리지 매핑이 제거되면 OVN 컨트롤러에서 연결된 패치 포트를 자동으로 정리합니다.

11.4.1. OVS 패치 포트 수동으로 정리

OVS 브리지 매핑을 제거한 후 연결된 패치 포트도 제거해야 합니다.

사전 요구 사항

- *정리 중인 패치 포트는 OVS(Open vSwitch) 포트여야 합니다.*
- *수동 패치 포트 정리를 수행하는 데 시스템 중단이 필요하지* ***않습니다****.*
- *이름 지정 규칙에 따라 정리할 패치 포트를 식별할 수 있습니다.*
  - ***br-$external_bridge*** *패치 포트의 이름은* ***phy-<external bridge name>*** *(예: phy-br-ex2) 입니다.*
  - ***br-int*** *패치 포트* ***의 이름은 int-<external bridge name>*** *(예:* ***int-br-ex2)입니다.***

절차

1. ***ovs-vsctl*** *을 사용하여 제거된 브리지 매핑 항목과 연결된 OVS 패치 포트를 제거합니다.*

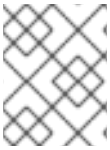
```
# ovs-vsctl del-port br-ex2 datacentre
# ovs-vsctl del-port br-tenant tenant
```

2. ***neutron-openvswitch-agent*** *를 다시 시작하십시오.*

```
# service neutron-openvswitch-agent restart
```

11.4.2. OVS 패치 포트 자동 정리

OVS 브리지 매핑을 제거한 후 연결된 패치 포트도 제거해야 합니다.



참고

OVN 브리지 매핑이 제거되면 OVN 컨트롤러에서 연결된 패치 포트를 자동으로 정리합니다.

사전 요구 사항

- *정리 중인 패치 포트는 OVS(Open vSwitch) 포트여야 합니다.*
- ***neutron-ovs-cleanup*** *명령을 사용하여 패치 포트를 자동으로 정리하면 네트워크 연결이 중단되며 예약된 유지 관리 기간 동안만 수행해야 합니다.*
- *플래그* ***--ovs_all_ports*** *를 사용하여* ***br-int에서 모든 패치 포트를 제거하고, br- tun*** *에서 터널 끝을 정리하며, 브리지에서 브리지로의 패치 포트를 사용합니다.*
- ***neutron-ovs-cleanup*** *명령은 모든 OVS 브리지에서 모든 패치 포트(인스턴스, qdhcp/qrouter)를 언플러그합니다.*

절차

1. ***-- ovs_all_ports 플래그를 사용하여 neutron-ovs -cleanup*** *명령을 실행합니다.*

중요

이 단계에 따라 네트워킹이 중단됩니다.

```
# /usr/bin/neutron-ovs-cleanup
--config-file /etc/neutron/plugins/ml2/openvswitch_agent.ini
--log-file /var/log/neutron/ovs-cleanup.log --ovs_all_ports
```

2. *Overcloud를 재배포하여 연결을 복원합니다.*
   ***openstack overcloud deploy** 명령을 다시 실행하면 브리지 매핑 값이 다시 적용됩니다.*

참고

재시작 후 OVS 에이전트는 bridge_mappings에 없는 연결을 방해하지 않습니다. 따라서 br- ***ex2에 연결된 br- int*** *가 있고* ***br-ex2*** *에 일부 흐름이 있는 경우, OVS 에이전트 또는 노드를 다시 시작할 때 bridge_mappings 구성에서* ***br-int*** *를 제거해도 두 브리지의 연결이 끊어지지 않습니다.*

추가 리소스

- *Advanced Overcloud Customization 가이드의 네트워크 환경 매개변수*
- *Advanced Overcloud Customization 가이드에서 오버클라우드 생성에 환경 파일 포함*

12장. VLAN 인식 인스턴스

12.1. VLAN 트렁크 및 VLAN 투명 네트워크

인스턴스는 단일 가상 NIC를 통해 VLAN 태그가 지정된 트래픽을 보내고 받을 수 있습니다. 이 기능은 VLAN 태그가 지정된 트래픽을 예상하는 NFV 애플리케이션(VNF)에 특히 유용하므로 단일 가상 NIC가 여러 고객 또는 서비스를 제공할 수 있습니다.

인스턴스는 단일 vNIC를 통해 VLAN 태그가 지정된 트래픽을 보내고 받을 수 있습니다. 이 기능은 VLAN 태그가 지정된 트래픽을 예상하는 NFV 애플리케이션(VNF)에 특히 유용하므로 단일 vNIC가 여러 고객 또는 서비스를 제공할 수 있습니다.

예를 들어 프로젝트 데이터 네트워크는 VLAN 또는 터널링(VXLAN/GRE) 분할을 사용할 수 있으며 인스턴스에서 VLAN ID로 태그가 지정된 트래픽을 확인할 수 있습니다. 결과적으로 네트워크 패킷은 인스턴스에 삽입되기 직전에 태그가 지정되며 전체 네트워크 전체에서 태그를 지정할 필요가 없습니다.

VLAN 태그 지정된 트래픽을 구현하려면 상위 포트를 생성하고 기존 neutron 네트워크에 새 포트를 연결합니다. 새 포트를 연결하면 OpenStack Networking에서 생성한 상위 포트에 트렁크 연결을 추가합니다. 그런 다음 하위 포트를 만듭니다. 이러한 하위 포트는 VLAN을 인스턴스에 연결하여 트렁크에 연결할 수 있습니다. 인스턴스 운영 체제 내에서 하위 포트와 연결된 VLAN의 트래픽에 태그를 지정하는 하위 인터페이스를 생성해야 합니다.

12.2. TRUNK 플러그인 검토

Red Hat openStack 배포 중에 trunk 플러그인은 기본적으로 활성화됩니다. 컨트롤러 노드의 구성을 검토할 수 있습니다.

- 컨트롤러 노드에서 /var/lib/config-data/neutron/etc/neutron/neutron.conf 파일에서 trunk 플러그인이 활성화되어 있는지 확인합니다.

```
service_plugins=router,qos,trunk
```

12.3. 트렁크 연결 생성

VLAN 태그 지정된 트래픽에 대한 트렁크를 구현하려면 상위 포트를 생성하고 기존 neutron 네트워크에 새 포트를 연결합니다. 새 포트를 연결하면 OpenStack Networking에서 생성한 상위 포트에 트렁크 연결을 추가합니다. 그런 다음 하위 포트를 만듭니다. 이러한 하위 포트는 VLAN을 인스턴스에 연결하여 트렁크에 연결할 수 있습니다. 인스턴스 운영 체제 내에서 하위 포트와 연결된 VLAN의 트래픽에 태그를 지정하는 하위 인터페이스를 생성해야 합니다.

1. 트렁크 포트 연결이 필요한 네트워크를 식별합니다. 이 네트워크는 트렁크된 VLAN에 액세스해야 하는 인스턴스가 포함됩니다. 이 예제에서는 공용 네트워크입니다.

```
openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 82845092-4701-4004-add7-838837837621 | private | 434c7982-cd96-4c41-a8c9-
b93adbdcb197 |
| 8d8bc6d6-5b28-4e00-b99e-157516ff0050 | public  | 3fd811b4-c104-44b5-8ff8-
7a86af5e332c |
+--------------------------------------+---------+--------------------------------------+
```

2. *상위 트렁크 포트를 만들어 인스턴스가 연결되는 네트워크에 연결합니다. 이 예에서는 공용 네트워크에 parent-trunk-port 라는 neutron 포트를 만듭니다. 이 트렁크는 상위 포트입니다. 이를 사용하여 하위 포트를 생성할 수 있습니다.*

```
openstack port create --network public parent-trunk-port
+-----------------------+--------------------------------------------------------------------------+
| Field                 | Value                                                                    |
+-----------------------+--------------------------------------------------------------------------+
| admin_state_up        | UP                                                                       |
| allowed_address_pairs |                                                                          |
| binding_host_id       |                                                                          |
| binding_profile       |                                                                          |
| binding_vif_details   |                                                                          |
| binding_vif_type      | unbound                                                                  |
| binding_vnic_type     | normal                                                                   |
| created_at            | 2016-10-20T02:02:33Z                                                     |
| description           |                                                                          |
| device_id             |                                                                          |
| device_owner          |                                                                          |
| extra_dhcp_opts       |                                                                          |
| fixed_ips             | ip_address='172.24.4.230', subnet_id='dc608964-9af3-4fed-9f06-
6d3844fb9b9b' |
| headers               |                                                                          |
| id                    | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39                                     |
| mac_address           | fa:16:3e:33:c4:75                                                        |
| name                  | parent-trunk-port                                                        |
| network_id            | 871a6bd8-4193-45d7-a300-dcb2420e7cc3                                     |
| project_id            | 745d33000ac74d30a77539f8920555e7                                         |
| project_id            | 745d33000ac74d30a77539f8920555e7                                         |
| revision_number       | 4                                                                        |
| security_groups       | 59e2af18-93c6-4201-861b-19a8a8b79b23                                     |
| status                | DOWN                                                                     |
| updated_at            | 2016-10-20T02:02:33Z                                                     |
+-----------------------+--------------------------------------------------------------------------+
```

3. *2단계에서 생성한 포트를 사용하여 트렁크를 생성합니다. 이 예에서 트렁크의 이름은* ***parent-trunk*** *입니다.*

```
openstack network trunk create --parent-port parent-trunk-port parent-trunk
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| created_at      | 2016-10-20T02:05:17Z                 |
| description     |                                      |
| id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name            | parent-trunk                         |
| port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                                    |
| status          | DOWN                                 |
| sub_ports       |                                      |
| tenant_id       | 745d33000ac74d30a77539f8920555e7     |
| updated_at      | 2016-10-20T02:05:17Z                 |
+-----------------+--------------------------------------+
```

4. *트렁크 연결을 확인합니다.*

```
openstack network trunk list
+--------------------------------------+-------------+--------------------------------------+-------------+
| ID                                   | Name        | Parent Port                          | Description |
+--------------------------------------+-------------+--------------------------------------+-------------+
| 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 | parent-trunk | 20b6fdf8-0d43-475a-a0f1-
ec8f757a4a39 |             |
+--------------------------------------+-------------+--------------------------------------+-------------+
```

5. *트렁크 연결 세부 정보를 확인합니다.*

```
openstack network trunk show parent-trunk
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| created_at      | 2016-10-20T02:05:17Z                 |
| description     |                                      |
| id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name            | parent-trunk                         |
| port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                                    |
| status          | DOWN                                 |
| sub_ports       |                                      |
| tenant_id       | 745d33000ac74d30a77539f8920555e7     |
| updated_at      | 2016-10-20T02:05:17Z                 |
+-----------------+--------------------------------------+
```

12.4. 트렁크에 하위 포트 추가

1. *neutron 포트를 만듭니다.*
   이 포트는 트렁크에 대한 하위 포트 연결입니다. 또한 상위 포트에 할당한MAC 주소를 지정해야 합니다.

```
openstack port create --network private --mac-address fa:16:3e:33:c4:75 subport-trunk-port
+-----------------------+--------------------------------------------------------------------------+
| Field                 | Value                                                                    |
+-----------------------+--------------------------------------------------------------------------+
| admin_state_up        | UP                                                                       |
| allowed_address_pairs |                                                                          |
| binding_host_id       |                                                                          |
| binding_profile       |                                                                          |
| binding_vif_details   |                                                                          |
| binding_vif_type      | unbound                                                                  |
| binding_vnic_type     | normal                                                                   |
| created_at            | 2016-10-20T02:08:14Z                                                     |
| description           |                                                                          |
| device_id             |                                                                          |
| device_owner          |                                                                          |
| extra_dhcp_opts       |                                                                          |
| fixed_ips             | ip_address='10.0.0.11', subnet_id='1a299780-56df-4c0b-a4c0-
c5a612cef2e8' |
| headers               |                                                                          |
```

```
| id                  | 479d742e-dd00-4c24-8dd6-b7297fab3ee9                                 |
| mac_address         | fa:16:3e:33:c4:75                                                    |
| name                | subport-trunk-port                                                   |
| network_id          | 3fe6b758-8613-4b17-901e-9ba30a7c4b51                                 |
| project_id          | 745d33000ac74d30a77539f8920555e7                                     |
| project_id          | 745d33000ac74d30a77539f8920555e7                                     |
| revision_number     | 4                                                                    |
| security_groups     | 59e2af18-93c6-4201-861b-19a8a8b79b23                                 |
| status              | DOWN                                                                 |
| updated_at          | 2016-10-20T02:08:15Z                                                 |
+---------------------+----------------------------------------------------------------------+
```

참고

오류가 발생하면 ***HttpException: conflict****, 다른 네트워크에서 상위 트렁크 포트가 있는 포트에 하위 포트를 생성 중인지 확인합니다. 이 예에서는 상위 트렁크 포트에 공용 네트워크를 사용하고 하위 포트에 private 네트워크를 사용합니다.*

2. *포트를 trunk(****parent-trunk****)과 연결하고 VLAN ID(55)를 지정합니다****.***

```
openstack network trunk set --subport port=subport-trunk-port,segmentation-
type=vlan,segmentation-id=55 parent-trunk
```

12.5. 트렁크를 사용하도록 인스턴스 구성

하위 포트에 할당된 RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스(neutron)의 MAC 주소를 사용하도록 VM 인스턴스 운영 체제를 구성해야 합니다. 하위 포트 생성 단계에서 특정 MAC 주소를 사용하도록 하위 포트를 구성할 수도 있습니다.

사전 요구 사항

- *컴퓨팅 노드의 실시간 마이그레이션을 수행하는 경우 RHOSP 네트워킹 서비스 RPC 응답 타임아웃이 RHOSP 배포에 적절하게 설정되어 있는지 확인합니다. RPC 응답 시간 초과 값은 사이트마다 다를 수 있으며 시스템 속도에 따라 달라집니다. 일반적인 권장 사항은 값을 트렁크 포트당 120초 이상으로 설정하는 것입니다.*
  RHOSP 배포의 트렁크 포트 바인딩 프로세스를 측정한 다음 RHOSP Networking 서비스 RPC 응답 타임아웃을 적절하게 설정하는 것이 좋습니다. RPC 응답 시간 초과 값을 낮게 유지하려고 하지만 RHOSP 네트워킹 서비스에서 RPC 응답을 받을 수 있는 충분한 시간도 제공합니다. 자세한 내용은 12.6 절. "네트워킹 서비스 RPC 타임아웃 구성" 의 내용을 참조하십시오.

절차

1. *네트워크 트렁크* ***명령을*** *사용하여 네트워크 트렁크의 구성을 검토합니다.*

   예제

```
$ openstack network trunk list
```

   샘플 출력

```
+--------------------+-------------+--------------------+-------------+
| ID                 | Name        | Parent Port        | Description |
+--------------------+-------------+--------------------+-------------+
```

```
| 0e4263e2-5761-4cf6- | parent-trunk | 20b6fdf8-0d43-475a- |             |
| ab6d-b22884a0fa88   |              | a0f1-ec8f757a4a39   |             |
+---------------------+--------------+---------------------+-------------+
```

예제

```
$ openstack network trunk show parent-trunk
```

샘플 출력

```
+-----------------+-------------------------------------------------------+
| Field           | Value                                                 |
+-----------------+-------------------------------------------------------+
| admin_state_up  | UP                                                    |
| created_at      | 2021-10-20T02:05:17Z                                  |
| description     |                                                       |
| id              | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88                  |
| name            | parent-trunk                                          |
| port_id         | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39                  |
| revision_number | 2                                                     |
| status          | DOWN                                                  |
| sub_ports       | port_id='479d742e-dd00-4c24-8dd6-b7297fab3ee9', segm |
|                 | entation_id='55', segmentation_type='vlan'            |
| tenant_id       | 745d33000ac74d30a77539f8920555e7                      |
| updated_at      | 2021-08-20T02:10:06Z                                  |
+-----------------+-------------------------------------------------------+
```

2. 상위 **port-id** 를 vNIC로 사용하는 인스턴스를 생성합니다.

예제

```
openstack server create --image cirros --flavor m1.tiny --security-group default --key-name
sshaccess --nic port-id=20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 testInstance
```

샘플 출력

```
+--------------------------------------+--------------------------------+
| Property                             | Value                          |
+--------------------------------------+--------------------------------+
| OS-DCF:diskConfig                    | MANUAL                         |
| OS-EXT-AZ:availability_zone          |                                |
| OS-EXT-SRV-ATTR:host                 | -                              |
| OS-EXT-SRV-ATTR:hostname             | testinstance                   |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                              |
| OS-EXT-SRV-ATTR:instance_name        |                                |
| OS-EXT-SRV-ATTR:kernel_id            |                                |
| OS-EXT-SRV-ATTR:launch_index         | 0                              |
| OS-EXT-SRV-ATTR:ramdisk_id           |                                |
| OS-EXT-SRV-ATTR:reservation_id       | r-juqco0el                     |
| OS-EXT-SRV-ATTR:root_device_name     | -                              |
| OS-EXT-SRV-ATTR:user_data            | -                              |
| OS-EXT-STS:power_state               | 0                              |
| OS-EXT-STS:task_state                | scheduling                     |
| OS-EXT-STS:vm_state                  | building                       |
```

```
| OS-SRV-USG:launched_at              | -                                    |
| OS-SRV-USG:terminated_at            | -                                    |
| accessIPv4                          |                                      |
| accessIPv6                          |                                      |
| adminPass                           | uMyL8PnZRBwQ                         |
| config_drive                        |                                      |
| created                             | 2021-08-20T03:02:51Z                 |
| description                         | -                                    |
| flavor                              | m1.tiny (1)                          |
| hostId                              |                                      |
| host_status                         |                                      |
| id                                  | 88b7aede-1305-4d91-a180-67e7eac      |
|                                     | 8b70d                                |
| image                               | cirros (568372f7-15df-4e61-a05f      |
|                                     | -10954f79a3c4)                       |
| key_name                            | sshaccess                            |
| locked                              | False                                |
| metadata                            | {}                                   |
| name                                | testInstance                         |
| os-extended-volumes:volumes_attached | []                                   |
| progress                            | 0                                    |
| security_groups                     | default                              |
| status                              | BUILD                                |
| tags                                | []                                   |
| tenant_id                           | 745d33000ac74d30a77539f8920555e      |
|                                     | 7                                    |
| updated                             | 2021-08-20T03:02:51Z                 |
| user_id                             | 8c4aea738d774967b4ef388eb41fef5      |
|                                     | e                                    |
+-------------------------------------+--------------------------------------+
```

추가 리소스

- *네트워킹 서비스 RPC 타임아웃 구성*

12.6. 네트워킹 서비스 RPC 타임아웃 구성

RHOSP(Red Hat OpenStack Platform) Networking 서비스(neutron) RPC 응답 타임아웃을 수정해야 하는 경우가 있을 수 있습니다. 예를 들어 시간 초과 값이 너무 작으면 트렁크 포트를 사용하는 컴퓨팅 노드의 실시간 마이그레이션이 실패할 수 있습니다.

RPC 응답 시간 초과 값은 사이트마다 다를 수 있으며 시스템 속도에 따라 달라집니다. 일반적인 권장 사항은 값을 트렁크 포트당 120초 이상으로 설정하는 것입니다.

사이트에서 트렁크 포트를 사용하는 경우 가장 좋은 방법은 RHOSP 배포의 트렁크 포트 바인딩 프로세스 시간을 측정한 다음 RHOSP Networking 서비스 RPC 응답 타임아웃을 적절하게 설정하는 것입니다. RPC 응답 시간 초과 값을 낮게 유지하려고 하지만 RHOSP 네트워킹 서비스에서 RPC 응답을 받을 수 있는 충분한 시간도 제공합니다.

수동 hieradata 덮어쓰기 **rpc_response_timeout** 을 사용하면 RHOSP 네트워킹 서비스의 RPC 응답 시간 초과 값을 설정할 수 있습니다.

절차

1. Undercloud 호스트에서 stack 사용자로 로그인한 사용자 지정 YAML 환경 파일을 만듭니다.

예제

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

작은 정보

RHOSP Orchestration 서비스(heat)에서는 템플릿 이라는 플랜 세트를 사용하여 환경을 설치하고 구성합니다. heat 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿 파일인 사용자 지정 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다.

2. ***ExtraConfig** 의 YAML 환경 파일에서 **rpc_response_timeout** 에 적절한 값(초)을 설정합니다. (기본값은 60초입니다.)*

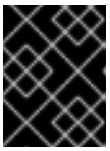
예제

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

참고

RHOSP Orchestration 서비스(heat)는 사용자 정의 환경 파일에 설정한 값으로 모든 RHOSP 노드를 업데이트하지만 이 값은 RHOSP 네트워킹 구성 요소에만 영향을 미칩니다.

3. ***openstack overcloud deploy** 명령을 실행하고 코어 heat 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.*



중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하므로 환경 파일의 순서가 중요합니다.

예제

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

추가 리소스

- *Advanced Overcloud Customization 가이드의 환경 파일*
- *Advanced Overcloud Customization 가이드 의 Overcloud 생성에 환경 파일 포함*

12.7. 트렁크 상태 이해

- ***활성**: 트렁크가 예상대로 작동하고 현재 요청이 없습니다.*

- ***DOWN***: *트렁크의 가상 및 실제 리소스가 동기화되지 않습니다. 이는 협상 중에 임시 상태일 수 있습니다.*
- ***BUILD***: *요청이 있고 리소스가 프로비저닝 중입니다. 성공적으로 완료되면 트렁크가* ***ACTIVE*** *로 돌아갑니다.*
- ***DEGRADED***: *배포 요청이 완료되지 않았으므로 트렁크가 부분적으로만 프로비저닝되었습니다. 하위 포트를 제거하고 다시 시도하는 것이 좋습니다.*
- ***ERROR***: *배포 요청이 실패했습니다. 오류로 트렁크를 되돌려주는 리소스를 제거합니다.* ***ERROR*** *상태에서는 하위 포트를 더 추가하지 마십시오. 이로 인해 더 많은 문제가 발생할 수 있기 때문입니다.*

13 장. RBAC 정책 구성

13.1. RBAC 정책 개요

OpenStack Networking의 역할 기반 액세스 제어(RBAC) 정책을 통해 공유 Neutron 네트워크를 세부적으로 제어할 수 있습니다. OpenStack Networking에서는 RBAC 테이블을 사용하여 프로젝트 간 Neutron 네트워크 공유를 제어하므로 관리자가 네트워크에 인스턴스를 연결할 수 있는 권한이 부여된 프로젝트를 제어할 수 있습니다.

따라서 클라우드 관리자는 일부 프로젝트에서 네트워크를 생성하는 기능을 제거할 수 있으며 대신 해당 프로젝트에 해당하는 기존 네트워크에 연결할 수 있습니다.

13.2. RBAC 정책 생성

이 예제 절차에서는 역할 기반 액세스 제어(RBAC) 정책을 사용하여 공유 네트워크에 대한 프로젝트 액세스 권한을 부여하는 방법을 설명합니다.

1. 사용 가능한 네트워크 목록을 확인합니다.

```
# openstack network list
+--------------------------------------+-------------+--------------------------------------------------------+
| id                                   | name        | subnets                                                |
+--------------------------------------+-------------+--------------------------------------------------------+
| fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 | web-servers | 20512ffe-ad56-4bb4-b064-2cb18fecc923 192.168.200.0/24 |
| bcc16b34-e33e-445b-9fde-dd491817a48a | private     | 7fe4a05a-4b81-4a59-8c47-82c965b0e050 10.0.0.0/24      |
| 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 | public      | 2318dc3b-cff0-43fc-9489-7d4cf48aaab9 172.24.4.224/28  |
+--------------------------------------+-------------+--------------------------------------------------------+
```

2. 프로젝트 목록을 확인합니다.

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

3. auditors 프로젝트 (4b0b98f8c040f38ba4f7146e8680f5)에 대한 액세스 권한을 부여하는 **web-servers** 네트워크에 대한 RBAC 항목을 생성합니다.

```
# openstack network rbac create --type network --target-project 4b0b98f8c6c040f38ba4f7146e8680f5 --action access_as_shared web-servers
Created a new rbac_policy:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_shared                     |
```

```
| id            | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id     | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type   | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5    |
| project_id    | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

*결과적으로 **auditors** 프로젝트의 사용자는 인스턴스를 **web-servers** 네트워크에 연결할 수 있습니다.*

13.3. RBAC 정책 검토

1. ***openstack network rbac list** 명령을 실행하여 기존 역할 기반 액세스 제어(**RBAC**) 정책의 **ID**를 검색합니다.*

```
# openstack network rbac list
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. ***openstack network rbac-show** 명령을 실행하여 특정 **RBAC** 항목의 세부 정보를 확인합니다.*

```
# openstack network rbac show 314004d0-2261-4d5e-bda7-0181fcf40709
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_shared                     |
| id            | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id     | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type   | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5    |
| project_id    | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

13.4. RBAC 정책 삭제

1. ***openstack network rbac list** 명령을 실행하여 기존 역할 기반 액세스 제어(**RBAC**) 정책의 **ID**를 검색합니다.*

```
# openstack network rbac list
```

```
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. *삭제하려는 **RBAC**의 **ID**를 사용하여 **openstack network rbac delete** 명령을 실행하여 **RBAC**를 삭제합니다.*

```
# openstack network rbac delete 314004d0-2261-4d5e-bda7-0181fcf40709
Deleted rbac_policy: 314004d0-2261-4d5e-bda7-0181fcf40709
```

13.5. 외부 네트워크에 대한 RBAC 정책 액세스 권한 부여

***--action access_as_external** 매개 변수를 사용하여 외부 네트워크에 대한 **RBAC**(역할 기반 액세스 제어) 정책 액세스(게이트웨이 인터페이스가 연결된 네트워크)에 액세스할 수 있습니다.*

*웹 서버 네트워크에 대한 **RBAC**를 생성하고 엔지니어링 프로젝트 (**c717f263785d4679b16a122516247deb**)에 대한 액세스 권한을 부여하려면 다음 예제 절차의 단계를 완료합니다.*

- *__--action access_as_external__ 옵션을 사용하여 새 **RBAC** 정책을 생성합니다.*

```
# openstack network rbac create --type network --target-project
c717f263785d4679b16a122516247deb --action access_as_external web-servers
 Created a new rbac_policy:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_external                   |
| id            | ddef112a-c092-4ac1-8914-c714a3d3ba08 |
| object_id     | 6e437ff0-d20f-4483-b627-c3749399bdca |
| object_type   | network                              |
| target_project | c717f263785d4679b16a122516247deb     |
| project_id    | c717f263785d4679b16a122516247deb     |
+---------------+--------------------------------------+
```

*결과적으로 **Engineering** 프로젝트의 사용자는 네트워크를 보거나 인스턴스를 연결할 수 있습니다.*

```
$ openstack network list
```

```
+--------------------------------------+-------------+--------------------------------------------------------+
| id                                   | name        | subnets                                                |
+--------------------------------------+-------------+--------------------------------------------------------+
| 6e437ff0-d20f-4483-b627-c3749399bdca | web-servers | fa273245-1eff-4830-b40c-57eaeac9b904 192.168.10.0/24 |
+--------------------------------------+-------------+--------------------------------------------------------+
```

14장. DVR(DISTRIBUTED VIRTUAL ROUTING) 구성

14.1. DVR(DISTRIBUTED VIRTUAL ROUTING) 이해

Red Hat OpenStack Platform을 배포할 때 중앙 집중식 라우팅 모델이나 DVR 중에서 선택할 수 있습니다.

각 모델에는 장단점이 있습니다. 이 문서를 사용하여 중앙 집중식 라우팅 또는 DVR이 요구 사항에 더 잘 맞는지 여부를 신중하게 계획하십시오.

DVR은 새 ML2/OVN 배포에서 기본적으로 활성화되며 새 ML2/OVS 배포에서는 기본적으로 비활성화됩니다. OpenStack Networking(neutron) API(deployment/neutron/neutron-api-container-puppet.yaml)에 대한 Heat 템플릿에는 DVR(Distributed Virtual Routing)을 활성화 및 비활성화하는 매개 변수가 포함되어 있습니다. DVR을 비활성화하려면 환경 파일에서 다음을 사용하십시오.

```
parameter_defaults:
  NeutronEnableDVR: false
```

14.1.1. 계층 3 라우팅 개요

Red Hat OpenStack Platform Networking 서비스(neutron)는 프로젝트 네트워크에 대한 라우팅 서비스를 제공합니다. 라우터가 없으면 프로젝트 네트워크의 VM 인스턴스가 공유 L2 브로드캐스트 도메인을 통해 다른 인스턴스와 통신할 수 있습니다. 라우터를 만들고 프로젝트 네트워크에 할당하면 해당 네트워크의 인스턴스가 다른 프로젝트 네트워크 또는 업스트림과 통신할 수 있습니다(라우터에 대해 외부 게이트웨이가 정의된 경우).

14.1.2. 라우팅 흐름

OpenStack의 라우팅 서비스는 다음 세 가지 주요 흐름으로 분류할 수 있습니다.

- East-West 라우팅 - 동일한 프로젝트에서 서로 다른 네트워크 간 트래픽 라우팅. 이 트래픽은 OpenStack 배포를 그대로 두지 않습니다. 이 정의는 IPv4 및 IPv6 서브넷 모두에 적용됩니다.

- 유동 IP를 사용한 North-South 라우팅 - 유동 IP 주소 지정은 수정할 수 있는 1대 1개의 네트워크 주소 변환(NAT)이며 VM 인스턴스 간에 이상일 수 있습니다. 유동 IP는 유동 IP와 네트워킹 서비스(neutron) 포트 간의 일대일 연결로 모델링되지만 NAT 변환을 수행하는 네트워킹 서비스 라우터와 연결하여 구현됩니다. 유동 IP 자체는 라우터에 외부 연결을 제공하는 uplink 네트워크에서 가져옵니다. 따라서 인스턴스는 외부 리소스(예: 인터넷의 끝점) 또는 다른 방법과 통신할 수 있습니다. 유동 IP는 IPv4 개념이며 IPv6에는 적용되지 않습니다. 프로젝트에서 사용하는 IPv6

주소 지정은 프로젝트 전체에 겹치지 않고 **GUA(Global Unicast Addresses)**를 사용하므로 **NAT** 없이 라우팅할 수 있습니다.

- 유동 **IP(SNAT**라고도 함)가 없는 **North-South** 라우팅 **-** 네트워킹 서비스는 유동 **IP**가 할당되지 않은 인스턴스에 대한 기본 포트 주소 변환**(PAT)** 서비스를 제공합니다. 이 서비스를 사용하면 인스턴스가 라우터를 통해 외부 엔드포인트와 통신할 수 있지만 다른 방법으로는 통신할 수 없습니다. 예를 들어 인스턴스는 인터넷에서 웹 사이트를 검색할 수 있지만 외부의 웹 브라우저는 인스턴스 내에서 호스팅되는 웹 사이트를 검색할 수 없습니다. **SNAT**는 **IPv4** 트래픽에만 적용됩니다. 또한 **GUAs** 접두사가 할당된 네트워킹 서비스 네트워크에는 네트워킹 서비스 라우터 외부 게이트웨이 포트에 **NAT**가 필요하지 않습니다.

14.1.3. 중앙 집중식 라우팅

원래 네트워킹 서비스**(neutron)**는 **Neutron L3** 에이전트가 관리하는 프로젝트의 가상 라우터가 모두 전용 노드 또는 노드(네트워크 노드 또는 컨트롤러 노드라고 함)에 배포되는 중앙 집중식 라우팅 모델로 설계되었습니다. 즉 라우팅 기능이 필요할 때마다(동부/서부 **IP** 또는 **SNAT)** 트래픽이 토폴로지의 전용 노드를 통과합니다. 이로 인해 여러 과제가 발생했고 차선의 트래픽 흐름이 발생했습니다. 예를 들면 다음과 같습니다.

- 인스턴스 간 트래픽이 컨트롤러 노드를 통과합니다. **L3**를 사용하여 두 인스턴스가 서로 통신해야 하는 경우 트래픽이 컨트롤러 노드에 도달해야 합니다. 인스턴스가 동일한 컴퓨팅 노드에 예약되어 있더라도 트래픽은 여전히 컴퓨팅 노드를 종료하고 컨트롤러를 통과하며 컴퓨팅 노드로 다시 라우팅해야 합니다. 이는 성능에 부정적인 영향을 미칩니다.

- 유동 **IP**가 있는 인스턴스는 컨트롤러 노드를 통해 패킷을 수신하고 전송합니다. 외부 네트워크 게이트웨이 인터페이스는 컨트롤러 노드에서만 사용할 수 있으므로 트래픽이 인스턴스에서 발생하거나 외부 네트워크에서 인스턴스로 향하든 컨트롤러 노드를 통과해야 합니다. 결과적으로 대규모 환경에서 컨트롤러 노드는 트래픽 로드가 많은 영향을 받습니다. 이 경우 성능과 확장성에 영향을 줄 수 있으며 외부 네트워크 게이트웨이 인터페이스에서 대역폭을 충분히 수용하기 위한 신중한 계획이 필요합니다. **SNAT** 트래픽에 동일한 요구 사항이 적용됩니다.

L3 에이전트를 더 잘 확장하기 위해 네트워킹 서비스는 **L3 HA** 기능을 사용하여 여러 노드에 가상 라우터를 배포할 수 있습니다. 컨트롤러 노드가 손실된 경우 **HA** 라우터는 다른 노드의 대기 모드로 장애 조치를 수행하며 **HA** 라우터 페일오버가 완료될 때까지 패킷 손실이 발생합니다.

14.2. DVR 개요

배포된 가상 라우팅**(DVR)**은 대체 라우팅 설계를 제공합니다. **DVR**은 컨트롤러 노드의 장애 도메인을 격리하고 **L3** 에이전트를 배포하고 모든 컴퓨팅 노드에 라우터를 예약하여 네트워크 트래픽을 최적화합니다. **DVR**에는 이러한 특성이 있습니다.

- *East-West 트래픽은 분산 방식으로 컴퓨팅 노드에 직접 라우팅됩니다.*

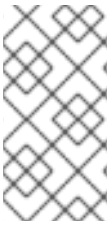
- *유동 IP를 사용한 North-South 트래픽이 계산 노드에 배포되고 라우팅됩니다. 이를 위해서는 외부 네트워크를 모든 컴퓨팅 노드에 연결해야 합니다.*

- *유동 IP가 없는 North-South 트래픽은 분산되지 않으며 전용 컨트롤러 노드가 필요합니다.*

- *노드가 SNAT 트래픽만 제공하도록 컨트롤러 노드의 L3 에이전트는 dvr_snat 모드를 사용합니다.*

- *neutron 메타데이터 에이전트는 모든 계산 노드에 배포되고 배포됩니다. 메타데이터 프록시 서비스는 모든 분산 라우터에서 호스팅됩니다.*

14.3. DVR 알려진 문제 및 주의 사항



참고

Red Hat OpenStack Platform 13의 경우 커널 버전이 kernel-3.10.0-514.1.1.el7 이상인 경우 DVR을 사용하지 마십시오.

- *DVR에 대한 지원은 ML2 코어 플러그인 및 OVS(Open vSwitch) 메커니즘 드라이버 또는 ML2/OVN 메커니즘 드라이버로 제한됩니다. 다른 백엔드는 지원되지 않습니다.*

- *OVS 및 OVN DVR 배포 모두에서 Red Hat OpenStack Platform 로드 밸런싱 서비스(octavia)의 네트워크 트래픽은 계산 노드 대신 컨트롤러 및 네트워크 노드를 통과합니다.*

- *ML2/OVS 메커니즘 드라이버 네트워크 백엔드 및 DVR을 사용하면 VIP를 생성할 수 있습니다. 그러나 allowed_address_cidrs를 사용하여 바인딩된 포트에 할당된 IP 주소는 가상 포트 IP 주소(/32)와 일치해야 합니다.*

  대신 바인딩된 포트에 CIDR 형식 IP 주소를 사용하는 경우 포트 전달은 백엔드에 구성되지 않으며 바인딩된 IP 포트에 도달하는 데 필요한 CIDR의 모든 IP에 대한 트래픽이 실패합니다.

- *DVR이 활성화된 경우에도 SNAT(소스 네트워크 주소 변환) 트래픽이 분산되지 않습니다. SNAT는 작동하지만 모든 수신/ 송신 트래픽은 중앙 집중식 컨트롤러 노드를 통과해야 합니다.*

- *DVR이 활성화된 경우에도 IPv6 트래픽이 분산되지 않습니다. IPv6 라우팅은 작동하지만 모든 수신/ 송신 트래픽은 중앙 집중식 컨트롤러 노드를 통과해야 합니다. IPv6 라우팅을 광범위하게 사용하는 경우 DVR을 사용하지 마십시오.*

- *DVR은 L3 HA와 함께 지원되지 않습니다. Red Hat OpenStack Platform 13 director와 함께 DVR을 사용하면 L3 HA가 비활성화됩니다. 즉, 네트워크 노드(및 L3 에이전트 간 부하 공유)에 라우터가 예약되지만 하나의 에이전트가 실패하면 이 에이전트에서 호스팅하는 모든 라우터도 실패합니다. 이는 SNAT 트래픽에만 영향을 미칩니다. 이러한 경우 allow_automatic_l3agent_failover 기능을 사용하는 것이 좋습니다. 이 경우 하나의 네트워크 노드가 실패하면 라우터를 다른 노드로 다시 스케줄링할 수 있습니다.*

- *neutron DHCP 에이전트에서 관리하는 DHCP 서버는 분산되지 않으며 여전히 컨트롤러 노드에 배포됩니다. DHCP 에이전트는 라우팅 설계(중앙화 또는 DVR)에 관계없이 컨트롤러 노드의 고가용성 구성에 배포됩니다.*

- *유동 IP를 사용하려면 각 컴퓨팅 노드에 외부 네트워크의 인터페이스가 필요합니다. 또한 각 Compute 노드에는 하나의 추가 IP 주소가 필요합니다. 이는 외부 게이트웨이 포트 및 유동 IP 네트워크 네임스페이스를 구현하기 때문입니다.*

- *프로젝트 데이터 분리에는 VLAN, GRE, VXLAN이 모두 지원됩니다. GRE 또는 VXLAN을 사용하는 경우 L2 채우기 기능을 활성화해야 합니다. Red Hat OpenStack Platform director는 설치하는 동안 L2 채우기를 적용합니다.*

14.4. 지원되는 라우팅 아키텍처

RHOSP(Red Hat OpenStack Platform)는 나열된 RHOSP 버전에서 중앙 집중식 고가용성(HA) 라우팅 및 분산 가상 라우팅(DVR)을 모두 지원합니다.

- *RHOSP 중앙 집중식 HA 라우팅 지원이 RHOSP 8에서 시작되었습니다.*

- *RHOSP 12에서 RHOSP 분산 라우팅 지원이 시작되었습니다.*

14.5. ML2 OVS를 사용하여 DVR 배포

ML2/OVS 배포에서 DVR(분산 가상 라우팅)을 배포하고 관리하려면 heat 템플릿 및 환경 파일에서 설정을 구성합니다.

heat 템플릿 설정을 사용하여 호스트 네트워킹을 프로비저닝합니다.

- *Compute 및 Controller 노드 둘 다에서 외부 네트워크 트래픽에 대해 실제 네트워크에 연결된 인터페이스를 구성합니다.*
- *외부 네트워크 트래픽용 인터페이스를 사용하여 계산 및 컨트롤러 노드에 브리지를 만듭니다.*

또한 프로비저닝된 네트워킹 환경과 일치하도록 Networking 서비스(neutron)를 구성하고 해당 브릿지를 사용하도록 트래픽을 허용합니다.

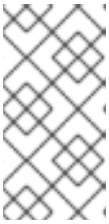
기본 설정은 지침으로만 제공됩니다. 네트워크 격리, 전용 NIC 또는 기타 여러 변수 요인에 대한 사용자 지정이 필요할 수 있는 프로덕션 또는 테스트 환경에서는 작동하지 않습니다. 환경을 설정할 때는 L2 에이전트에서 사용하는 브리지 매핑 유형 매개 변수와 L3 에이전트와 같은 기타 에이전트의 외부 브리지를 올바르게 구성해야 합니다.

다음 예제 절차에서는 일반적인 기본값을 사용하여 개념 증명 환경을 구성하는 방법을 보여줍니다.

절차

1. *OS::TripleO::Compute::Net::SoftwareConfig 의 값이 파일 overcloud-resource-registry.yaml 또는 배포 명령에 포함된 환경 파일의 OS::TripleO::Controller::Net::SoftwareConfig 값과 일치하는지 확인합니다.*

   이 값은 net_config_bridge.yaml 과 같은 파일의 이름을 지정합니다. 명명된 파일은 외부 네트워크 컴퓨팅 노드 L2 에이전트에 대한 Neutron 브리지 매핑을 구성합니다. 브리지는 DVR 배포의 컴퓨팅 노드에 호스팅된 유동 IP 주소의 트래픽을 라우팅합니다. 일반적으로 이 파일 이름 값은 environments/net-multiple-nics.yaml 과 같이 오버클라우드를 배포할 때 사용하는 네트워크 환경 파일에서 찾을 수 있습니다.

   참고

   컴퓨팅 노드의 네트워크 구성을 사용자 지정하는 경우 대신 사용자 지정 파일에 적절한 구성을 추가해야 할 수 있습니다.

2. *계산 노드에 외부 브리지가 있는지 확인합니다.*

   a. ***openstack-tripleo-heat-templates** 디렉터리의 로컬 복사본을 만듭니다.*

   b. ***$ cd <local_copy_of_templates_directory.***

   c. ***process-templates** 스크립트를 실행하여 템플릿을 임시 출력 디렉터리에 렌더링합니다.*

   ```
   $ ./tools/process-templates.py -r <roles_data.yaml> \
     -n <network_data.yaml> -o <temporary_output_directory>
   ```

   d. ***<temporary_output_directory>/network/config** 에서 역할 파일을 확인합니다.*

3. *필요한 경우 컨트롤러 노드와 일치하는 외부 브리지를 포함하도록 계산 템플릿을 사용자 지정하고 환경 파일의 **OS::TripleO::Compute::Net::SoftwareConfig** 에 사용자 지정 파일 경로 이름을 지정합니다.*

4. *오버클라우드를 배포할 때 배포 명령에 **environments/services/neutron-ovs-dvr.yaml** 파일을 포함합니다.*

   ```
   $ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
   templates/environments/services/neutron-ovs-dvr.yaml
   ```

5. ***L3 HA**가 비활성화되었는지 확인합니다.*

참고

***L3** 에이전트의 외부 브리지 구성은 **Red Hat OpenStack Platform 13**에서 더 이상 사용되지 않으며 **Red Hat OpenStack Platform 15**에서 제거되었습니다.*

14.6. 중앙 집중식 라우터를 분산 라우팅으로 마이그레이션

*이 섹션에서는 **L3 HA** 중앙 집중식 라우팅을 사용하는 **Red Hat OpenStack Platform** 배포를 위한 분산 라우팅으로 업그레이드하는 방법에 대해 설명합니다.*

절차

1. *배포를 업그레이드하고 올바르게 작동하는지 확인합니다.*

2. ***director** 스택 업데이트를 실행하여 **DVR**을 구성합니다.*

3. *기존 라우터를 통해 라우팅이 올바르게 작동하는지 확인합니다.*

4. ***L3 HA** 라우터를 직접 배포 하도록 전환할 수 없습니다. 대신 각 라우터에 대해 **L3 HA** 옵션을 비활성화한 다음 분산 옵션을 활성화합니다.*

   a. *라우터를 비활성화합니다.*

   예제

   ```
   $ openstack router set --disable router1
   ```

   b. *고가용성 지우기:*

   예제

   ```
   $ openstack router set --no-ha router1
   ```

   c. ***DVR**을 사용하도록 라우터를 구성합니다.*

   예제

```
$ openstack router set --distributed router1
```

d. *라우터를 활성화합니다.*

예제

```
$ openstack router set --enable router1
```

e. *분산 라우팅이 올바르게 작동하는지 확인합니다.*

추가 리소스

- *ML2 OVS를 사용하여 DVR 배포*

15장. NETWORKING LBAASV2 API를 사용하여 부하 분산-AS-A-SERVICE 구성

15.1. LBAAS 개요

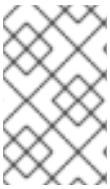
LBaaS(Load Balancing-as-a-Service)를 사용하면 OpenStack Networking에서 지정된 인스턴스 간에 들어오는 요청을 균등하게 배포할 수 있습니다. 이 섹션의 단계를 완료하여 Open vSwitch(OVS) 플러그인과 함께 LBaaS를 사용하도록 OpenStack 네트워킹을 구성합니다.

LBaaS(Load Balancing-as-a-Service)를 사용하면 OpenStack Networking에서 지정된 인스턴스 간에 들어오는 요청을 균등하게 배포할 수 있습니다. 따라서 인스턴스 간에 워크로드를 예측 가능한 방식으로 공유할 수 있으며 시스템 리소스를 보다 효과적으로 사용할 수 있습니다. 들어오는 요청은 다음 로드 밸런싱 방법 중 하나를 사용하여 분산됩니다.

- 라운드 로빈 - 여러 인스턴스 간에 요청을 균등하게 순환합니다.
- 소스 IP - 고유 소스 IP 주소의 요청이 동일한 인스턴스로 일관되게 전달됩니다.
- least connections (최소 연결) - 활성 연결 수가 가장 적은 인스턴스에 요청을 할당합니다.

표 15.1. LBaaS 기능

기능	설명
모니터	LBaaS는 PING, TCP, HTTP 및 HTTPS GET 방법을 사용하여 가용성 모니터링을 제공합니다. 모니터는 풀 멤버가 요청을 처리하는 데 사용할 수 있는지 여부를 결정합니다.
관리	LBaaS는 다양한 툴 세트를 사용하여 관리합니다. REST API는 프로그래밍 방식의 관리 및 스크립팅에 사용할 수 있습니다. 사용자는 CLI(neutron) 또는 OpenStack 대시보드를 통해 로드 밸런서를 관리합니다.
연결 제한	인그레스 트래픽은 연결 제한이 있는 제약이 있을 수 있습니다. 이 기능을 사용하면 워크로드 제어가 허용되며 DoS(Denial of Service) 공격을 완화하는 데도 도움이 될 수 있습니다.
세션 지속성	LBaaS는 들어오는 요청이 여러 인스턴스 풀 내에서 동일한 인스턴스로 라우팅되도록 하여 세션 지속성을 지원합니다. LBaaS는 쿠키 및 소스 IP 주소를 기반으로 라우팅 결정을 지원합니다.



참고

LBaaS는 현재 IPv4 주소 지정에서만 지원됩니다.

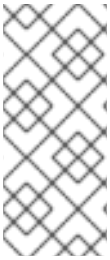
15.2. OPENSTACK NETWORKING 및 LBAAS 토폴로지

OpenStack Networking(neutron) 서비스는 다음 두 가지 범주로 광범위하게 분류할 수 있습니다.

Neutron API 서버 - 이 서비스는 OpenStack Networking API 서버를 실행합니다. 이 서버는 OpenStack 네트워킹과 상호 작용하기 위해 최종 사용자 및 서비스에 API를 제공하는 주요 역할을 합니다. 또한 이 서버는 기본 데이터베이스와 상호 작용하여 테넌트 네트워크, 라우터 및 로드 밸런서 세부 정보를 저장하고 검색합니다.

Neutron 에이전트 - OpenStack 네트워킹을 위한 다양한 네트워크 기능을 제공하는 서비스입니다.

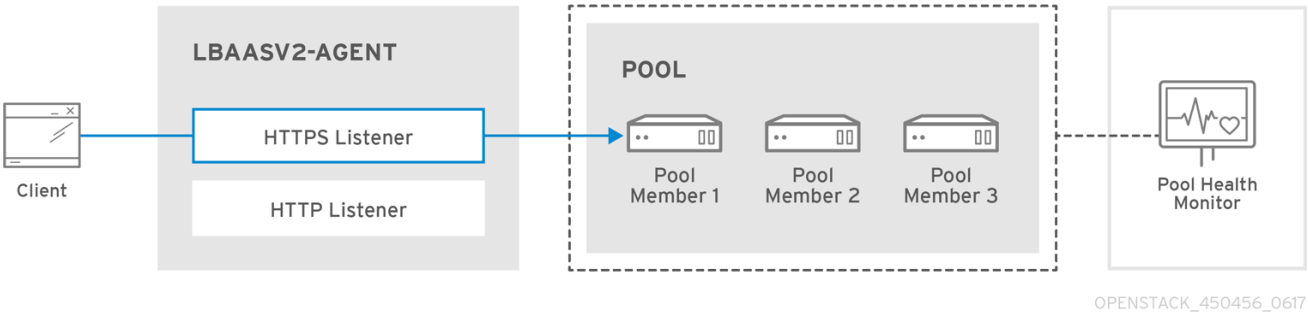
- neutron-dhcp-agent - 테넌트 사설 네트워크의 DHCP IP 주소 지정을 관리합니다.
- neutron-l3-agent - 테넌트 사설 네트워크, 외부 네트워크 및 기타 네트워크 간의 계층 3 라우팅을 지원합니다.



참고

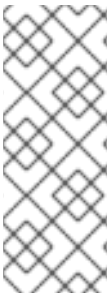
neutron-lbaasv2-agent (HAProxy 사용)는 더 이상 사용되지 않습니다. octavia를 사용한 기본 로드 밸런싱 참조 구현은 Load Balancing-as-a-Service에 Octavia 사용 가이드를 참조하십시오.

다음 다이어그램에서는 풀 멤버로 향하는 HTTPS 트래픽의 흐름을 보여줍니다.



15.2.1. LBaaS 지원 상태

- LBaaS v1 API가 버전 10에서 제거되었습니다.
- LBaaS v2 API는 Red Hat OpenStack Platform 13 이후에 더 이상 사용되지 않으며 제거됩니다. (Octavia는 대체입니다.)
- LBaaS 배포는 현재 Red Hat OpenStack Platform director에서 지원되지 않습니다.



참고

neutron-lbaasv2-agent (HAProxy 사용)는 더 이상 사용되지 않습니다. Octavia를 사용한 기본 로드 밸런싱 참조 구현은 Octavia for Load Balancing-as-a-Service 가이드를 참조하십시오. 타사 플러그인 지원을 위한 API를 지원하는 데 neutron-lbaas RPM을 계속 사용할 수 있습니다.

15.3. LBAAS 구성

이 절차에서는 OVS(Open vSwitch) 플러그인과 함께 LBaaS를 사용하도록 OpenStack Networking(neutron)을 구성합니다.

참고

neutron-server 서비스를 실행하는 노드에서 다음 단계를 수행합니다.

절차

컨트롤러 노드(API 서버)에서 다음을 수행합니다.

1. LBaaS를 활성화합니다.

```
# yum install openstack-neutron-lbaas -y
```

2. LBaaS 테이블을 neutron 데이터베이스에 추가합니다.

```
$ neutron-db-manage --subproject neutron-lbaas --config-file /var/lib/config-
data/neutron/etc/neutron/neutron.conf --config-file /var/lib/config-
data/neutron/etc/neutron/plugins/ml2/ml2_conf.ini upgrade head
```

3. */var/lib/config-data/neutron/etc/neutron/neutron_lbaas.conf* 에서 서비스 프로바이더를 변경합니다. **[service provider]** 섹션에서 이 항목을 제외한모든항목을 주석 처리합니다.

```
service_provider=LOADBALANCERV2:Haproxy:neutron_lbaas.drivers.haproxy.plugin_driver.H
aproxyOnHostPluginDriver:default
```

4. */var/lib/config-data/neutron/etc/neutron/neutron.conf* 에서 **service_plugins** 에 구성된 **LBaaS v2** 플러그인이 있는지 확인합니다.

```
service_plugins=neutron_lbaas.services.loadbalancer.plugin.LoadBalancerPluginv2
```

이전에 추가한 다른 플러그인도 볼 수 있습니다.

참고

lbaasv1 이 구성된 경우 **lbaasv2** 에 대한 위의 설정으로 바꿉니다.

5. */var/lib/config-data/neutron/etc/neutron/lbaas_agent.ini* 에서 **[DEFAULT]** 섹션에 다음을 추가합니다.

```
ovs_use_veth = False
interface_driver =neutron.agent.linux.interface.OVSInterfaceDriver
```

6. */var/lib/config-data/neutron/etc/neutron/services_lbaas.conf* 에 다음을 **[haproxy]** 섹션에 추가합니다.

```
user_group = haproxy
```

a. 다른 장치 드라이버 항목을 주석 처리합니다.

참고

***l3-agent** 가 실패 모드인 경우 **l3_agent** 로그 파일을 참조하십시오. **/var/lib/config-data/neutron/etc/neutron/neutron.conf** 를 편집하고 **[DEFAULT]** 의 특정 값을 주석 처리하고 로그 파일에 설명된 대로 **in oslo_messaging_rabbit** 값의 주석을 해제해야 할 수도 있습니다.*

7. ***LbaaS** 서비스를 구성하고 상태를 검토합니다.*

   a. ***lbaasv1** 서비스를 중지하고 **lbaasv2** 를 시작합니다 :*

   ```
   # systemctl disable neutron-lbaas-agent.service
   # systemctl stop neutron-lbaas-agent.service
   # systemctl mask neutron-lbaas-agent.service
   # systemctl enable neutron-lbaasv2-agent.service
   # systemctl start neutron-lbaasv2-agent.service
   ```

   b. ***lbaasv2** 의 상태를 검토합니다 :*

   ```
   # systemctl status neutron-lbaasv2-agent.service
   ```

   c. ***neutron-server** 를 다시 시작하고 상태를 확인합니다.*

   ```
   # systemctl restart neutron-server.service
   # systemctl status neutron-server.service
   ```

   d. ***Loadbalancerv2** 에이전트를 확인합니다.*

   ```
   $ openstack network agent list
   ```

16장. IPV6를 사용한 프로젝트 네트워킹

16.1. IPV6 서브넷 옵션

RHOSP(Red Hat OpenStack Platform) 프로젝트 네트워크에서 IPv6 서브넷을 만들 때 주소 모드와 라우터 알림 모드를 지정하여 다음 표에 설명된 특정 결과를 얻을 수 있습니다.

참고

RHOSP는 ML2/OVN 배포에서 IPv6 접두사 위임을 지원하지 않습니다. Global Unicast Address 접두사를 수동으로 설정해야 합니다.

RRA 모드	주소 모드	결과
ipv6_ra_mode=not set	ipv6-address-mode=slaac	인스턴스는 SLAAC(상태 비저장 주소 자동 구성)를 사용하여 외부 라우터(OpenStack 네트워킹에서 관리되지 않음)에서 IPv6 주소를 받습니다.  **참고** OpenStack Networking은 SLAAC에 대해 EUI-64 IPv6 주소 할당만 지원합니다. 이를 통해 기본 64비트와 MAC 주소를 기반으로 호스트 자체 할당 주소로 IPv6 네트워킹을 간소화할 수 있습니다. 다른 넷마스크 및 SLAAC의 *address_assign_type* 을 사용하여 서브넷을 생성할 수 없습니다.
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	인스턴스에서 **DHCPv6 상태를 사용하여 IPv6 주소 및 선택적 정보를 OpenStack Networking(dnsmasq)에서 받습니다**.

RRA 모드	주소 모드	결과
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateless	인스턴스에서 SLAAC를 사용하여 외부 라우터에서 IPv6 주소를 수신하고 **DHCPv6 상태 비저장** 을 사용하여 OpenStack Networking(dnsmasq)에서 선택적 정보를 받습니다.
ipv6_ra_mode=slaac	ipv6-address-mode=not-set	인스턴스에서 SLAAC를 사용하여 OpenStack Networking(ra**dvd**)에서 IPv6 주소를 받습니다.
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=not-set	인스턴스에서 DHCPv6 상태를 사용하여 외부 DHCPv6 서버에서 IPv6 주소 및 선택적 정보를 **받습니다**.
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=not-set	인스턴스에서 SLAAC를 사용하여 OpenStack Networking(ra**dvd**)에서 IPv6 주소를 수신하고 DHCPv6 **상태 비저장** 을 사용하는 외부 DHCPv6 서버의 선택적 정보를 받습니다.
ipv6_ra_mode=slaac	ipv6-address-mode=slaac	인스턴스에서 **SLAAC** 를 사용하여 OpenStack Networking(ra**dvd**)에서 IPv6 주소를 받습니다.
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=dhcpv6-stateful	인스턴스에서 DHCPv6 **상태 저장을 사용하여dnsmasq(OpenStack 네트워킹)에서 IPv6 주소를 수신하고 DHCPv6 상태를** 사용하는**dnsmasq**(OpenStack Networking)의 선택적 정보를 받습니다.
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	인스턴스에서 **SLAAC** 를 사용하여 OpenStack Networking(ra**dvd**)에서 IPv6 주소를 수신하고 **DHCPv6 상태 비저장** 을 사용하는**dnsmasq**(OpenStack Networking)의 선택적 정보를 받습니다.

16.2. STATEFUL DHCPV6을 사용하여 IPV6 서브넷 생성

RHOSP(Red Hat OpenStack) 프로젝트 네트워크에서 IPv6 서브넷을 만들 수 있습니다.

*예를 들어 **QA**라는 프로젝트에서 **database-servers**라는 네트워크에서 **Stateful DHCPv6**을 사용하여 **IPv6** 서브넷을 생성할 수 있습니다.*

절차

1. ***IPv6** 서브넷을 만들 프로젝트의 프로젝트 **ID**를 검색합니다. 이러한 값은 **OpenStack** 배포 간에 고유하므로 이 예제의 값과 값이 다릅니다.*

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 25837c567ed5458fbb441d39862e1399 |    QA    |
| f59f631a77264a8eb0defc898cb836af |  admin   |
| 4e2e1951e70643b5af7ed52f3ff36539 |   demo   |
| 8561dff8310e4cd8be4b6fd03dc8acf5 | services |
+----------------------------------+----------+
```

2. ***OpenStack Networking(neutron)**에 있는 모든 네트워크 목록을 검색하고 **IPv6** 서브넷을 호스팅하려는 네트워크의 이름을 확인합니다.*

```
# openstack network list
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| id                                   | name            | subnets                                               |
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private         | c17f74c4-db41-4538-af40-
48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public          | 303ced03-6019-4e79-a21c-
1942a460b920 172.24.4.224/28        |
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers |
|
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
```

3. ***openstack subnet create** 명령에 프로젝트 **ID**, 네트워크 이름 및 **ipv6** 주소 모드를 포함합니다.*

```
# openstack subnet create --ip-version 6 --ipv6-address-mode dhcpv6-stateful --project
25837c567ed5458fbb441d39862e1399 --network database-servers --subnet-range
fdf8:f53b:82e4::53/125 subnet_name

Created a new subnet:
+-------------------+-------------------------------------------------------+
| Field             | Value                                                 |
+-------------------+-------------------------------------------------------+
```

```
| allocation_pools  | {"start": "fdf8:f53b:82e4::52", "end": "fdf8:f53b:82e4::56"} |
| cidr              | fdf8:f53b:82e4::53/125                                       |
| dns_nameservers   |                                                              |
| enable_dhcp       | True                                                         |
| gateway_ip        | fdf8:f53b:82e4::51                                           |
| host_routes       |                                                              |
| id                | cdfc3398-997b-46eb-9db1-ebbd88f7de05                         |
| ip_version        | 6                                                            |
| ipv6_address_mode | dhcpv6-stateful                                              |
| ipv6_ra_mode      |                                                              |
| name              |                                                              |
| network_id        | 6aff6826-4278-4a35-b74d-b0ca0cbba340                         |
| tenant_id         | 25837c567ed5458fbb441d39862e1399                             |
+-------------------+--------------------------------------------------------------+
```

검증 단계

1. *네트워크 목록을 검토하여 이 구성을 확인합니다. **database-servers** 에 대한 항목은 이제 새로 생성된 **IPv6** 서브넷을 반영합니다.*

```
# openstack network list
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| id                                   | name            | subnets                                               |
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers | cdfc3398-997b-46eb-9db1-
ebbd88f7de05 fdf8:f53b:82e4::50/125 |
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-db41-4538-af40-
48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-6019-4e79-a21c-
1942a460b920 172.24.4.224/28        |
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
```

결과

*이 구성의 결과로 **QA** 프로젝트에서 생성하는 인스턴스에서 **database-servers** 서브넷에 추가할 때 **DHCP IPv6** 주소를 수신할 수 있습니다.*

```
# openstack server list
+--------------------------------------+------------+--------+------------+-------------+-----------------------
----------------+
| ID                                   | Name       | Status | Task State | Power State | Networks
|
+--------------------------------------+------------+--------+------------+-------------+-----------------------
----------------+
| fad04b7a-75b5-4f96-aed9-b40654b56e03 | corp-vm-01 | ACTIVE | -          | Running     |
database-servers=fdf8:f53b:82e4::52 |
+--------------------------------------+------------+--------+------------+-------------+-----------------------
----------------+
```

추가 리소스

- ***IPv6** 서브넷에서 특정 결과를 얻기 위해 라우터 알림 모드와 주소 모드 조합을 찾으려면 네트워킹 가이드의 **IPv6** 서브넷 옵션을 참조하십시오.*

17장. 프로젝트 할당량 관리

17.1. 프로젝트 할당량 구성

OpenStack Networking(neutron)에서는 할당량 사용을 지원하여 테넌트/프로젝트에서 생성한 리소스 수를 제한할 수 있습니다.

절차

- /var/lib/config-data/neutron/etc/neutron/neutron.conf 파일에서 다양한 네트워크 구성 요소에 대한 프로젝트 할당량을 설정할 수 있습니다.

  예를 들어 프로젝트에서 생성할 수 있는 라우터 수를 제한하려면 quota_router 값을 변경합니다.

  ```
  quota_router = 10
  ```

  이 예제에서 각 프로젝트는 최대 10개의 라우터로 제한됩니다.

할당량 설정 목록은 바로 따르는 섹션을 참조하십시오.

17.2. L3 할당량 옵션

다음은 계층 3 (L3) 네트워킹에 사용할 수있는 할당량 옵션입니다.

- quota_floatingip - 프로젝트에 사용할 수 있는 유동 IP 수입니다.
- quota_network - 프로젝트에서 사용할 수 있는 네트워크 수입니다.
- quota_port - 프로젝트에서 사용할 수 있는 포트 수입니다.
- quota_router - 프로젝트에서 사용할 수 있는 라우터 수입니다.

- *quota_subnet - 프로젝트에서 사용할 수 있는 서브넷 수입니다.*

- *quota_vip - 프로젝트에서 사용할 수 있는 가상 IP 주소 수입니다.*

17.3. 방화벽 할당량 옵션

다음은 프로젝트의 방화벽 관리에 사용할 수 있는 할당량 옵션입니다.

- *quota_firewall - 프로젝트에서 사용할 수 있는 방화벽 수입니다.*

- *quota_firewall_policy - 프로젝트에서 사용할 수 있는 방화벽 정책 수입니다.*

- *quota_firewall_rule - 프로젝트에 사용할 수 있는 방화벽 규칙 수입니다.*

17.4. 보안 그룹 할당량 옵션

네트워킹 서비스 할당량 엔진은 보안 그룹 및 보안 그룹 규칙을 관리하며 default 보안 그룹(및 IPv4 및 IPv6에 대한 모든 송신 트래픽을 수락하는 두 개의 기본 보안 그룹 규칙)을 생성하기 전에 모든 할당량을 0으로 설정할 수 없습니다. 새 프로젝트를 생성할 때 Networking 서비스는 네트워크 또는 포트가 생성될 때까지 또는 보안 그룹 또는 보안 그룹 규칙을 나열할 때까지 기본 보안 그룹을 생성하지 않습니다.

다음은 프로젝트에서 생성할 수 있는 보안 그룹 수를 관리하는 데 사용할 수 있는 할당량 옵션입니다.

- *quota_security_group - 프로젝트에서 사용할 수 있는 보안 그룹 수입니다.*

- *quota_security_group_rule - 프로젝트에 사용할 수 있는 보안 그룹 규칙 수입니다.*

17.5. 관리 할당량 옵션

다음은 관리자가 프로젝트 할당량을 관리하기 위해 사용할 수 있는 추가 옵션입니다.

- ***default_quota* -** 프로젝트에서 사용할 수 있는 리소스의 기본 수입니다.*

- ***quota_health_monitor* -** 프로젝트에서 사용할 수 있는 상태 모니터 수입니다.*

  *상태 모니터는 리소스를 사용하지 않지만 **OpenStack Networking**에서는 상태 모니터를 리소스 소비자로 간주하므로 할당량 옵션을 사용할 수 있습니다.*

- ***quota_member -** 프로젝트에서 사용할 수 있는 풀 구성원 수입니다.*

  *풀 구성원은 리소스를 사용하지 않지만 **OpenStack Networking**에서는 풀 멤버를 리소스 소비자로 간주하므로 할당량 옵션을 사용할 수 있습니다.*

- ***quota_pool -** 프로젝트에서 사용할 수 있는 풀 수입니다.*

18장. FWAAS(FIREWALL-AS-A-SERVICE) 구성

18.1. FWAAS(FIREWALL-AS-A-SERVICE) 개요

FWaaS(Firewall-as-a-Service) 플러그인은 **OpenStack Networking(neutron)**에 경계 방화벽 관리를 추가합니다. **FWaaS**는 **iptables**를 사용하여 프로젝트 내의 모든 가상 라우터에 방화벽 정책을 적용하며 각 프로젝트에 대해 하나의 방화벽 정책과 논리적 방화벽 인스턴스를 지원합니다.

FWaaS는 **OpenStack Networking(neutron)** 라우터에서 트래픽을 필터링하여 경계에서 작동합니다. **FWaaS**를 인스턴스 수준에서 작동하는 보안 그룹과 구별합니다.

참고

FWaaS는 현재 기술 프리뷰 상태에 있습니다. 테스트되지 않은 작업은 권장되지 않습니다. 향후 릴리스에서는 **FWaaS**를 사용할 수 없습니다.

다음 예제 다이어그램에서는 **VM2** 인스턴스의 수신 및 송신 트래픽 흐름을 보여줍니다.

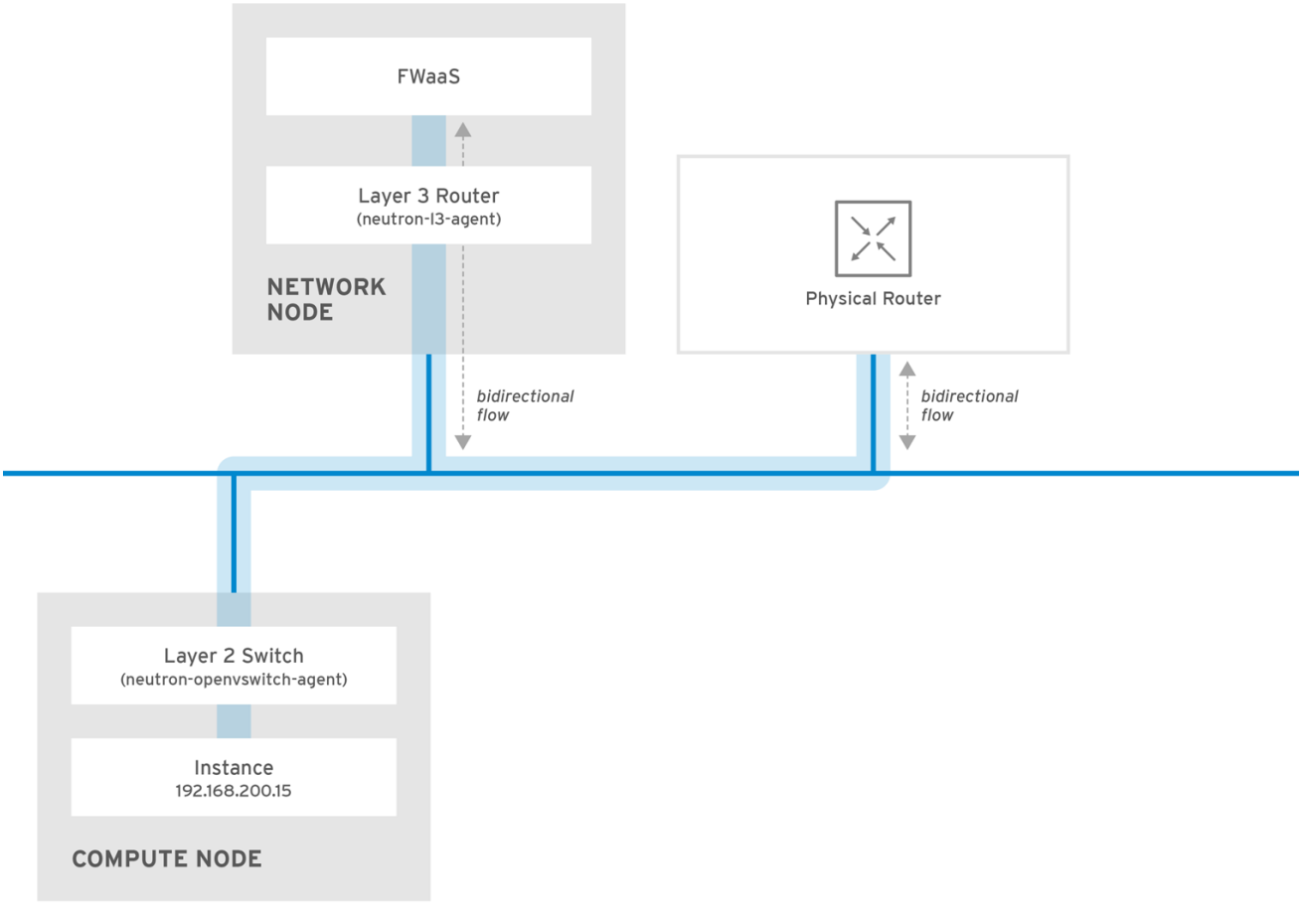


그림 1. FWaaS 아키텍처

18.2. FWAAS(서비스로서의 방화벽) 활성화

1. FWaaS 패키지를 설치합니다.

```
# dnf install openstack-neutron-fwaas python-neutron-fwaas
```

2. /var/lib/config-data/neutron/etc/neutron/neutron.conf 파일에서 FWaaS 플러그인을 활성화합니다.

```
service_plugins = neutron.services.firewall.fwaas_plugin.FirewallPlugin
```

3. fwaas_driver.ini 파일에서 FWaaS를 구성합니다.

```
[fwaas]
driver = neutron.services.firewall.drivers.linux.iptables_fwaas.IptablesFwaasDriver
enabled = True

[service_providers]
service_provider =
LOADBALANCER:Haproxy:neutron_lbaas.services.loadbalancer.drivers.haproxy.plugin_driver.
HaproxyOnHostPluginDriver:default
```

4. 일반적으로 컨트롤러 노드에 있는 local_settings.py 파일에서 FWaaS 대시보드 관리 옵션을 활성화합니다.

```
/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py
'enable_firewall' = True
```

5. neutron-server 를 다시 시작하여 변경 사항을 적용합니다.

```
# systemctl restart neutron-server
```

18.3. FWAAS(서비스로서의 방화벽) 구성

먼저 방화벽 규칙을 생성하고 포함할 정책을 생성한 다음 방화벽을 생성하고 정책을 적용합니다.

1. *방화벽 규칙을 생성합니다.*

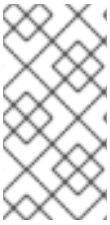
```
$ neutron firewall-rule-create --protocol <tcp|udp|icmp|any> --destination-port <port-range> --action <allow|deny>
```

CLI에는 프로토콜 값이 필요합니다. 규칙이 프로토콜과 무관한 경우 값을 사용할 수 있습니다 .

2. *방화벽 정책을 생성합니다.*

```
$ neutron firewall-policy-create --firewall-rules "<firewall-rule IDs or names separated by space>" myfirewallpolicy
```

neutron firewall-policy-create 명령에서 지정하는 규칙의 순서는 중요합니다. 업데이트 작업(여러 규칙 추가 시) 또는 삽입 규칙 작업(단일 규칙을 추가할 때)에서 빈 방화벽 정책을 생성하고 나중에 규칙을 추가할 수 있습니다.



참고

FWaaS는 항상 각 정책의 우선 순위에서 항상 모든 규칙을 거부합니다. 따라서 규칙이 없는 방화벽 정책은 기본적으로 모든 트래픽을 차단합니다.

18.4. 방화벽 생성

- *openstack security group create 명령을 사용하여 방화벽을 생성합니다.*

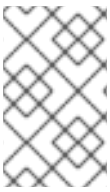
```
$ openstack security group create <firewall-policy-uuid>
```

방화벽은 OpenStack 네트워킹 라우터를 생성하고 인터페이스를 연결할 때까지 PENDING_CREATE 상태로 유지됩니다.

19장. 허용된 주소 쌍 구성

19.1. 허용되는 주소 쌍의 개요

허용되는 주소 쌍은 특정 MAC 주소, IP 주소 또는 둘 다 서브넷에 관계없이 네트워크 트래픽이 포트를 통과하도록 허용하는 경우입니다. 허용된 주소 쌍을 정의할 때 빠른 데이터 플레인 페일오버를 활성화하기 위해 두 VM 인스턴스 간에 IP 주소를 복제하는 VRRP(Virtual Router Redundancy Protocol)와 같은 프로토콜을 사용할 수 있습니다.



참고

allowed-address 쌍 확장은 현재 ML2 및 Open vSwitch 플러그인에서만 지원됩니다.

Red Hat OpenStack Platform 명령줄 클라이언트 openstack port 명령을 사용하여 허용되는 주소 쌍을 정의합니다.

중요

허용되는 주소 쌍에서 더 넓은 IP 주소 범위를 가진 default 보안 그룹을 사용해서는 안 됩니다. 이렇게 하면 단일 포트가 동일한 네트워크 내의 다른 모든 포트에 대해 보안 그룹을 우회할 수 있습니다.

예를 들어 이 명령은 네트워크의 모든 포트에 영향을 미치고 모든 보안 그룹을 무시합니다.

```
# openstack port set --allowed-address
mac_address=3e:37:09:4b,ip_address=0.0.0.0/0
9e67d44eab334f07bf82fa1b17d824b6
```

참고

ML2/OVN 메커니즘 드라이버 네트워크 백엔드를 사용하면 VIP를 생성할 수 있습니다. 그러나 allowed_address_cidrs를 사용하여 바인딩된 포트에 할당된 IP 주소는 가상 포트 IP 주소(/32)와 일치해야 합니다.

대신 바인딩된 포트에 CIDR 형식 IP 주소를 사용하는 경우 포트 전달은 백엔드에 구성되지 않으며 바인딩된 IP 포트에 도달하는 데 필요한 CIDR의 모든 IP에 대한 트래픽이 실패합니다.

추가 리소스

- 명령줄 인터페이스 참조 에서 포트 명령
- 포트 생성 및 하나의 주소 쌍 허용
- 허용되는 주소 쌍 추가

19.2. 포트 생성 및 하나의 주소 쌍 허용

허용되는 주소 쌍을 사용하여 포트를 만들면 서브넷에 관계없이 네트워크 트래픽이 포트를 통과할 수 있습니다.

사전 요구 사항

- **ML2/OVS** 플러그인을 사용하고 있습니다.

중요

허용되는 주소 쌍에서 **IP** 주소 범위가 더 넓은 기본 보안 그룹을 사용하지 마십시오. 이렇게 하면 단일 포트가 동일한 네트워크 내의 다른 모든 포트에 대해 보안 그룹을 우회할 수 있습니다.

절차

- 다음 명령을 사용하여 포트를 생성하고 하나의 주소 쌍을 허용합니다.

```
# openstack port create <port-name> --network <network> --allowed-address mac_address=<mac-address>,ip_address=<ip-cidr>
```

추가 리소스

- 명령줄 인터페이스 참조 에서 포트 명령
- 허용되는 주소 쌍의 개요

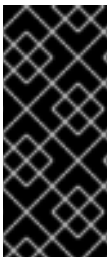
- 허용되는 주소 쌍 추가

19.3. 허용되는 주소 쌍 추가

허용되는 주소 쌍을 포트에 추가하여 서브넷에 관계없이 네트워크 트래픽이 포트를 통과하도록 할 수 있습니다.

사전 요구 사항

- **ML2/OVS** 플러그인을 사용하고 있습니다.



중요

허용되는 주소 쌍에서 **IP** 주소 범위가 더 넓은 기본 보안 그룹을 사용하지 마십시오. 이렇게 하면 단일 포트가 동일한 네트워크 내의 다른 모든 포트에 대해 보안 그룹을 우회할 수 있습니다.

절차

- 다음 명령을 사용하여 허용되는 주소 쌍을 추가합니다.

```
# openstack port set <port-uuid> --allowed-address mac_address=
<mac_address>,ip_address=<ip_cidr>
```

참고

mac_address 및 포트의 **ip_address**와 일치하는 **allowed-address** 쌍을 설정할 수 없습니다. **mac_address** 및 **ip_address** 와 일치하는 트래픽이 이미 포트를 통과할 수 있으므로 이러한 설정이 적용되지 않기 때문입니다.

추가 리소스

- 명령줄 인터페이스 참조 에서 포트 명령

- 허용되는 주소 쌍의 개요

- *포트 생성 및 하나의 주소 쌍 허용*

20장. 계층 3 고가용성(HA) 구성

20.1. HA(고가용성)가 없는 RHOSP 네트워킹 서비스

HA(고가용성) 기능이 없는 RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스는 물리적 노드 장애에 취약합니다.

일반적인 배포에서는 프로젝트에서 물리적 네트워킹 서비스 계층 3(L3) 에이전트 노드에서 실행되도록 예약된 가상 라우터를 만듭니다. 이는 L3 에이전트 노드가 손실되고 종속 가상 시스템이 나중에 외부 네트워크와의 연결이 끊어지면 문제가 됩니다. 유동 IP 주소도 사용할 수 없습니다. 또한 라우터가 호스트하는 모든 네트워크 간에 연결이 끊어집니다.

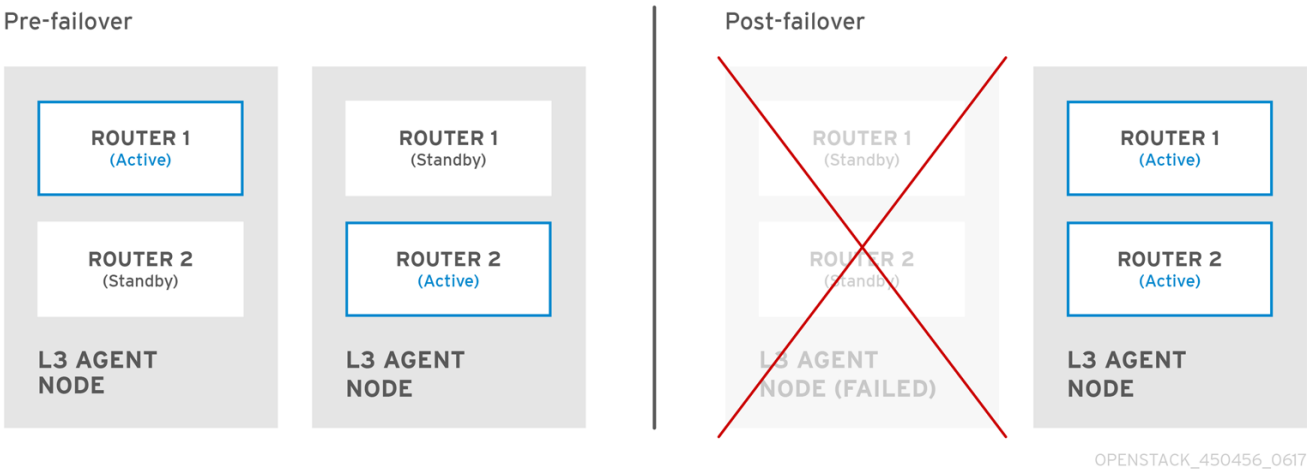
20.2. 계층 3 고가용성(HA) 개요

이 액티브/패시브 고가용성(HA) 구성은 업계 표준 VRRP(RFC 3768에 정의된 대로)를 사용하여 프로젝트 라우터 및 유동 IP 주소를 보호합니다. 가상 라우터는 여러 RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스 노드에 무작위로 예약되며, 하나는 활성 라우터로 지정되고 나머지 라우터는 standby 역할로 사용됩니다.

참고

L3) HA를 배포하려면 유동 IP 범위 및 외부 네트워크에 대한 액세스를 포함하여 중복 네트워킹 서비스 노드에서 유사한 구성을 유지 관리해야 합니다.

다음 다이어그램에서 활성 라우터 1 및 라우터 2 라우터 는 별도의 물리적 L3 네트워킹 서비스 에이전트 노드에서 실행되고 있습니다. L3 HA는 해당 노드에 백업 가상 라우터를 예약했으며 물리적 노드 장애가 발생하는 경우 서비스를 다시 시작할 준비가 되었습니다. L3 에이전트 노드가 실패하면 L3 HA가 영향을 받는 가상 라우터 및 유동 IP 주소를 작업 노드에 다시 예약합니다.



장애 조치 이벤트 중에 유동 IP를 통한 인스턴스 TCP 세션은 영향을 받지 않고 새 L3 노드로 마이그레이션됩니다. SNAT 트래픽만 페일오버 이벤트의 영향을 받습니다.

L3 에이전트는 액티브/액티브 HA 모드에서 추가로 보호됩니다.

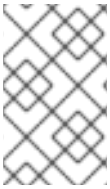
추가 리소스

- VRRP(Virtual Router Redundancy Protocol)

20.3. 계층 3 HA(고가용성) 페일오버 조건

RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스의 계층 3(L3) HA(고가용성)는 다음 이벤트에서 보호 리소스를 자동으로 다시 예약합니다.

- 네트워킹 서비스 L3 에이전트 노드가 종료되거나 하드웨어 장애로 인해 전원이 손실됩니다.
- L3 에이전트 노드는 물리적 네트워크에서 격리되며 연결이 끊어집니다.



참고

L3 에이전트 서비스를 수동으로 중지해도 장애 조치 이벤트가 발생하지 않습니다.

20.4. 3 계층 HA(고가용성)에 대한 프로젝트 고려 사항

RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스 계층 3(L3) HA(고가용성) 구성은 백엔드에서 발생하며 프로젝트에 표시되지 않습니다. 프로젝트는 정상적으로 가상 라우터를 계속 만들고 관리할 수 있지만 L3 HA 구현을 설계할 때 알아야 할 몇 가지 제한 사항이 있습니다.

- L3 HA는 프로젝트당 최대 255개의 가상 라우터를 지원합니다.
- 내부 VRRP 메시지는 각 프로젝트에 대해 자동으로 생성되는 별도의 내부 네트워크 내에서 전송됩니다. 이 프로세스는 사용자에게 투명하게 수행됩니다.
-

ML2/OVS에서 HA(고가용성) 라우터를 구현할 때 각 L3 에이전트는 각 라우터에 haproxy 및 neutron-keepalived-state-change-monitor 프로세스를 생성합니다. 각 프로세스는 약 20MB의 메모리를 사용합니다. 기본적으로 각 HA 라우터는 3개의 L3 에이전트에 상주하며 각 노드의 리소스를 사용합니다. 따라서 RHOSP 네트워크 크기를 조정할 때 구현하려는 HA 라우터 수를 지원하기에 충분한 메모리를 할당했는지 확인합니다.

20.5. RHOSP NETWORKING 서비스에 대한 HA(고가용성) 변경

관리자가 라우터를 생성할 때 --ha=True/False 플래그를 설정할 수 있도록 RHOSP(Red Hat OpenStack Platform) 네트워킹 서비스(neutron) API가 업데이트되어 /var/ lib/config-data/neutron/etc/neutron.conf 에서 l3_ha 의 기본 구성을 덮어씁니다.

- neutron-server로 HA(고가용성) 변경:
  - L3 계층 3(L3) HA는 네트워킹 서비스에서 사용하는 스케줄러(랜덤 또는 최소 라우터)에 관계없이 활성 역할을 무작위로 할당합니다.
  - 데이터베이스 스키마가 가상 라우터에 대한 VIP(가상 IP 주소) 할당을 처리하도록 수정되었습니다.
  - L3 HA 트래픽을 지시하기 위해 전송 네트워크가 생성됩니다.
- Networking 서비스 L3 에이전트에 대한 HA 변경 사항:
  - 부하 분산 및 HA 기능을 제공하는 새로운 keepalived 관리자가 추가되었습니다.
  - IP 주소는 VIP로 변환됩니다.

20.6. RHOSP NETWORKING 서비스 노드에서 레이어 3 HA(고가용성) 활성화

RHOSP(Red Hat OpenStack Platform) director는 RHOSP 컨트롤러가 두 개 이상 있고 분산 가상 라우팅(DVR)을 사용하지 않는 경우 기본적으로 가상 라우터에 고가용성(HA)을 활성화합니다. RHOSP Orchestration 서비스(heat) 매개변수인 max_l3_agents_per_router 를 사용하면 HA 라우터가 예약된 RHOSP Networking 서비스 계층 3(L3) 에이전트의 최대 수를 설정할 수 있습니다.

사전 요구 사항

- ***RHOSP** 배포에서 **DVR**을 사용하지 않습니다.*
- *최소 두 개의 **RHOSP** 컨트롤러가 배포되어 있습니다.*

절차

1. ***stack** 사용자로 언더클라우드에 로그인하고 **stackrc** 파일을 가져와 **director** 명령줄 툴을 활성화합니다.*

   예제

   ```
   $ source ~/stackrc
   ```

2. *사용자 지정 **YAML** 환경 파일을 생성합니다.*

   예제

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

   작은 정보

   *오케스트레이션 서비스(**heat**)에서는 템플릿 이라는 플랜 세트를 사용하여 환경을 설치하고 구성합니다. **heat** 템플릿에 대한 사용자 지정을 제공하는 특수 유형의 템플릿 파일인 사용자 지정 환경 파일을 사용하여 오버클라우드의 특정 부분을 사용자 지정할 수 있습니다.*

3. ***YAML** 환경 파일에서 **NeutronL3HA** 매개 변수를 **true** 로 설정합니다. 이렇게 하면 **director** 가 기본적으로 설정하지 않은 경우에도 **HA**가 활성화됩니다.*

```
parameter_defaults:
  NeutronL3HA: 'true'
```

4. ***HA** 라우터가 예약된 최대 **L3** 에이전트 수를 설정합니다.*

***max_l3_agents_per_router** 매개변수를 배포의 최솟값과 총 네트워크 노드 수의 값으로 설정합니다. (**0** 값은 라우터가 모든 에이전트에 예약됨을 나타냅니다.)*

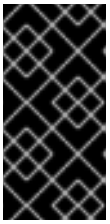
예제

```
parameter_defaults:
  NeutronL3HA: 'true'
  ControllerExtraConfig:
    neutron::server::max_l3_agents_per_router: 2
```

*이 예에서는 **4**개의 **Networking** 서비스 노드를 배포하는 경우 **L3** 에이전트 두 개만 각 **HA** 가상 라우터(활성 하나의 활성 및 하나의 대기)를 보호합니다.*

***max_l3_agents_per_router** 값을 사용 가능한 네트워크 노드 수보다 크게 설정하는 경우 새 **L3** 에이전트를 추가하여 대기 라우터 수를 확장할 수 있습니다. 배포하는 모든 새로운 **L3** 에이전트 노드에 대해 네트워킹 서비스는 **max_l3_agents_per_router** 제한에 도달할 때까지 가상 라우터의 추가 대기 버전을 예약합니다.*

5. ***openstack overcloud deploy** 명령을 실행하고 코어 **heat** 템플릿, 환경 파일 및 이 새 사용자 지정 환경 파일을 포함합니다.*



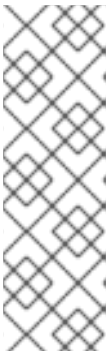
중요

후속 환경 파일에 정의된 매개 변수와 리소스가 우선하기 때문에 환경 파일의 순서가 중요합니다.

예제

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

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



참고

NeutronL3HA 가 **true** 로 설정되면 기본적으로 생성된 모든 가상 라우터를 **HA** 라우터로 설정합니다. 라우터를 생성할 때 **openstack router create** 명령에 **--no-ha** 옵션을 포함하여 **HA** 옵션을 덮어쓸 수 있습니다.

```
# openstack router create --no-ha
```

추가 리소스

- **Advanced Overcloud Customization** 가이드의 환경 파일
- **Advanced Overcloud Customization** 가이드 의 **Overcloud** 생성에 환경 파일 포함

20.7. HA(고가용성) RHOSP NETWORKING 서비스 노드 구성 검토

절차

- 가상 라우터 네임스페이스에서 **ip address** 명령을 실행하여 **ha-** 접두사가 붙은 결과에서 **HA(고가용성)** 장치를 반환합니다.

```
# ip netns exec qrouter-b30064f9-414e-4c98-ab42-646197c74020 ip address
<snip>
2794: ha-45249562-ec: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 12:34:56:78:2b:5d brd ff:ff:ff:ff:ff:ff
inet 169.254.0.2/24 brd 169.254.0.255 scope global ha-54b92d86-4f
```

계층 **3 HA**를 활성화하면 가상 라우터 및 유동 **IP** 주소가 개별 노드 장애로 보호됩니다.

21장. 태그가 있는 가상 장치 식별

21.1. 가상 장치 태그 개요

여러 네트워크 인터페이스 또는 블록 장치가 있는 인스턴스를 시작하는 경우 장치 태그를 사용하여 각 장치의 의도한 역할을 인스턴스 운영 체제에 전달할 수 있습니다. 태그는 인스턴스 부팅 시 장치에 할당되며 메타데이터 **API** 및 구성 드라이브(활성화된 경우)를 통해 인스턴스 운영 체제에서 사용할 수 있습니다.

태그는 다음 매개변수를 사용하여 설정됩니다.

- **--block-device tag=device metadata**
- **--NIC tag=device** 메타데이터

21.2. 가상 장치 태그 지정

절차

- 가상 장치에 태그를 지정하려면 인스턴스를 만들 때 태그 매개 변수인 **--block-device** 및 **--nic** 을 사용합니다.

  예를 들면 다음과 같습니다.

```
$ nova boot test-vm --flavor m1.tiny --image cirros \
--nic net-id=55411ca3-83dd-4036-9158-bf4a6b8fb5ce,tag=nfv1 \
--block-device id=b8c9bef7-aa1d-4bf4-a14d-17674b370e13,bus=virtio,tag=database-server
NFVappServer
```

  결과 태그는 기존 인스턴스 메타데이터에 추가되며 메타데이터 **API** 및 구성 드라이브 모두를 통해 사용할 수 있습니다.

  이 예에서 다음 **devices** 섹션에서는 메타데이터를 채웁니다.

  meta_data.json 파일의 샘플 내용:

```
{
```

```
  "devices": [
    {
        "type": "nic",
        "bus": "pci",
        "address": "0030:00:02.0",
        "mac": "aa:00:00:00:01",
        "tags": ["nfv1"]
    },
    {
        "type": "disk",
        "bus": "pci",
        "address": "0030:00:07.0",
        "serial": "disk-vol-227",
        "tags": ["database-server"]
    }
  ]
}
```

*장치 태그 메타데이터는 메타데이터 **API**에서 **GET /openstack/latest/meta_data.json** 을 사용하여 사용할 수 있습니다.*

*구성 드라이브가 활성화되어 인스턴스 운영 체제의 **/configdrive** 에 마운트되면 메타데이터도 **/configdrive/openstack/latest/meta_data.json** 에 있습니다.*