Red Hat OpenStack Platform 14

ネットワークガイド

OpenStack Networking の詳細ガイド

Last Updated: 2020-03-11

Red Hat OpenStack Platform 14 ネットワークガイド

OpenStack Networking の詳細ガイド

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

一般的な OpenStack Networking タスクのガイドです。

目次

前書き 7

第1章 ネットワークの概要 9
1.1. ネットワークの仕組み 9
1.1.1. VLAN 9
1.2. 2 つの LAN の接続 9
1.2.1. ファイアウォール 10
1.3. OPENSTACK NETWORKING (NEUTRON) 10
1.4. CIDR 形式の使用 10

第2章 OPENSTACK NETWORKING の概念 12
2.1. OPENSTACK NETWORKING (NEUTRON) のインストール 12
2.1.1. サポートされるインストール 12
2.2. OPENSTACK NETWORKING の図 12
2.3. セキュリティーグループ 13
2.4. OPEN VSWITCH 13
2.5. MODULAR LAYER 2 (ML2) 14
2.5.1. ML2 が導入された理由 14
2.5.2. ML2 ネットワーク種別 14
2.5.3. ML2 メカニズムドライバー 15
2.6. L2 POPULATION 15
2.7. OPENSTACK NETWORKING サービス 16
2.7.1. L3 エージェント 16
2.7.2. DHCP エージェント 16
2.7.3. Open vSwitch エージェント 16
2.8. テナントとプロバイダーネットワーク 17
2.8.1. テナントネットワーク 17
2.8.2. プロバイダーネットワーク 17
2.8.2.1. フラットプロバイダーネットワーク 18
2.8.2.2. コントローラーノードの設定 18
2.8.2.3. ネットワークノードとコンピュートノードの設定 18
2.8.2.4. ネットワークノードの設定 19
2.9. レイヤー 2 およびレイヤー 3 ネットワーク 20
2.9.1. 可能な範囲でのスイッチの使用 20

パート I. 一般的なタスク 22

第3章 一般的な管理タスク 23
3.1. ネットワークの作成 23
3.2. 高度なネットワークの作成 25
3.3. ネットワークルーティングの追加 26
3.4. ネットワークの削除 26
3.5. テナントのネットワークの削除 26
3.6. サブネットの作成 27
3.6.1. 新規サブネットの作成 27
3.7. サブネットの削除 29
3.8. ルーターの追加 29
3.9. ルーターの削除 30
3.10. インターフェースの追加 30
3.11. インターフェースの削除 30
3.12. IP アドレスの設定 30
3.12.1. Floating IP アドレスプールの作成 30

3.12.2. 特定の Floating IP アドレスの割り当て 31
3.12.3. Floating IP アドレスの無作為な割り当て 31
3.13. 複数の FLOATING IP アドレスプールの作成 32
3.14. 物理ネットワークのブリッジ 32

第4章 IP アドレス使用のプランニング 34
4.1. 複数の VLAN の使用 34
4.2. VLAN トラフィックの分離 34
4.3. IP アドレスの消費 36
4.4. 仮想ネットワーク 36
4.5. ネットワークプランの例 36

第5章 OPENSTACK NETWORKING ルーターポートのレビュー 38
5.1. ポートの現在のステータスの表示 38

第6章 プロバイダーネットワークのトラブルシューティング 40
6.1. 基本的な PING 送信テスト 40
6.2. VLAN ネットワークのトラブルシューティング 42
6.2.1. VLAN 設定とログファイルの確認 43
6.3. テナントネットワーク内からのトラブルシューティング 43
6.3.1. 名前空間内での高度な ICMP テストの実行 44

第7章 物理ネットワークへのインスタンスの接続 46
7.1. フラットプロバイダーネットワークの使用 46
7.1.1. 送信トラフィックのフロー 49
7.1.2. 受信トラフィックのフロー 51
7.1.3. トラブルシューティング 53
7.2. VLAN プロバイダーネットワークの使用 55
7.2.1. 送信トラフィックのフロー 57
7.2.2. 受信トラフィックのフロー 60
7.2.3. トラブルシューティング 61
7.3. コンピュートのメタデータアクセスの有効化 62
7.4. FLOATING IP アドレス 62

第8章 OPENSTACK NETWORKING の物理スイッチの設定 63
8.1. 物理ネットワーク環境のプランニング 63
8.2. CISCO CATALYST スイッチの設定 64
8.2.1. トランクポートの設定 64
8.2.1.1. Cisco Catalyst スイッチのトランクポートの設定 64
8.2.2. アクセスポートの設定 65
8.2.2.1. Cisco Catalyst スイッチのアクセスポートの設定 65
8.2.3. LACP ポートアグリゲーションの設定 66
8.2.3.1. 物理 NIC 上での LACP の設定 66
8.2.3.2. Cisco Catalyst スイッチ上での LACP の設定 67
8.2.4. MTU の設定 68
8.2.4.1. Cisco Catalyst スイッチ上での MTU の設定 68
8.2.5. LLDP ディスカバリーの設定 69
8.2.5.1. Cisco Catalyst スイッチ上での LLDP の設定 69
8.3. CISCO NEXUS スイッチの設定 69
8.3.1. トランクポートの設定 69
8.3.1.1. Cisco Nexus スイッチのトランクポートの設定 70
8.3.2. アクセスポートの設定 70
8.3.2.1. Cisco Nexus スイッチのアクセスポートの設定 70
8.3.3. LACP ポートアグリゲーションの設定 70

8.3.3.1. 物理 NIC 上での LACP の設定 71
8.3.3.2. Cisco Nexus スイッチ上での LACP の設定 71
8.3.4. MTU の設定 72
8.3.4.1. Cisco Nexus 7000 スイッチ上での MTU の設定 72
8.3.5. LLDP ディスカバリーの設定 72
8.3.5.1. Cisco Nexus 7000 スイッチ上での LLDP の設定 72
8.4. CUMULUS LINUX スイッチの設定 72
8.4.1. トランクポートの設定 73
8.4.1.1. Cumulus Linux スイッチのトランクポートの設定 73
8.4.2. アクセスポートの設定 73
8.4.2.1. Cumulus Linux スイッチのアクセスポートの設定 73
8.4.3. LACP ポートアグリゲーションの設定 74
8.4.3.1. 物理 NIC 上での LACP の設定 74
8.4.3.2. Cumulus Linux スイッチでの LACP の設定 74
8.4.4. MTU の設定 74
8.4.4.1. Cumulus Linux スイッチでの MTU の設定 74
8.4.5. LLDP ディスカバリーの設定 75
8.5. EXTREME NETWORKS EXOS スイッチの設定 75
8.5.1. トランクポートの設定 75
8.5.1.1. Extreme Networks EXOS スイッチでのトランクポートの設定 75
8.5.2. アクセスポートの設定 75
8.5.2.1. Extreme Networks EXOS スイッチのアクセスポートの設定 76
8.5.3. LACP ポートアグリゲーションの設定 76
8.5.3.1. 物理 NIC 上での LACP の設定 76
8.5.3.2. Extreme Networks EXOS スイッチでの LACP の設定 77
8.5.4. MTU の設定 77
8.5.4.1. Extreme Networks EXOS スイッチでの MTU の設定 77
8.5.5. LLDP ディスカバリーの設定 77
8.5.5.1. Extreme Networks EXOS スイッチでの LLDP の設定 78
8.6. JUNIPER EX シリーズスイッチの設定 78
8.6.1. トランクポートの設定 78
8.6.1.1. Juniper EX シリーズスイッチでのトランクポートの設定 78
8.6.2. アクセスポートの設定 78
8.6.2.1. Juniper EX シリーズスイッチのアクセスポートの設定 79
8.6.3. LACP ポートアグリゲーションの設定 79
8.6.3.1. 物理 NIC 上での LACP の設定 79
8.6.3.2. Juniper EX シリーズスイッチでの LACP の設定 80
8.6.4. MTU の設定 81
8.6.4.1. Juniper EX シリーズスイッチでの MTU の設定 81
8.6.5. LLDP ディスカバリーの設定 82
8.6.5.1. Juniper EX シリーズスイッチでの LLDP の設定 82

パート II. 高度な設定 **84**

第9章 MTU の設定 **85**
9.1. MTU の概要 85
9.1.1. MTU 広告の設定 86
9.1.2. director での MTU の設定 86
9.1.3. MTU 計算の確認 87

第10章 QUALITY-OF-SERVICE (QOS) の設定 **88**
10.1. QOS ポリシースコープ 88
10.2. QOS ポリシー管理 88
10.3. 送信トラフィックの DSCP マーキング 89

10.4. QOS ポリシーの RBAC 90

第11章 ブリッジマッピングの設定 91
11.1. ブリッジマッピングを使用する目的 91
11.1.1. ブリッジマッピングの設定 91
11.1.2. コントローラーノードの設定 91
11.1.3. トラフィックの流れ 91
11.2. ブリッジマッピングのメンテナンス 92
11.2.1. 手動ポートクリーンアップ 92
11.2.2. 「neutron-ovs-cleanup」を使用した自動ポートクリーンアップ 93
11.2.2.1. neutron-ovs-cleanup の使用例 93

第12章 VLAN 対応のインスタンス 94
12.1. 概要 94
12.2. トランクプラグインのレビュー 94
12.3. トランク接続の作成 94
12.4. トランクへのサブポートの追加 96
12.5. トランクを使用するためのインスタンスの設定 97
12.6. トランクの状態 98

第13章 RBAC の設定 100
13.1. 新規 RBAC ポリシーの作成 100
13.2. 設定したRBAC ポリシーの確認 101
13.3. RBAC ポリシーの削除 101
13.4. 外部ネットワークの RBAC 102

第14章 分散仮想ルーター (DVR) の設定 103
14.1. レイヤー 3 ルーティングの概要 103
14.1.1. ルーティングのフロー 103
14.1.2. 集中ルーティング 103
14.2. DVR の概要 104
14.3. 既知の問題および警告 104
14.4. サポートされているルーティングアーキテクチャー 105
14.5. ML2 OVS を使用した DVR のデプロイ 105
14.6. 集中ルーティングから分散ルーティングへの移行 106

第15章 NEUTRON LBAASV2 API を使用した LOAD BALANCING-AS-A-SERVICE の設定 108
15.1. OPENSTACK NETWORKING および LBAAS トポロジー 109
15.1.1. LBaaS のサポートステータス 109
15.1.2. サービスの配置 110

第16章 OCTAVIA を使用した LOAD BALANCING-AS-A-SERVICE (LBAAS) 111
16.1. OCTAVIA の概要 111
16.2. ソフトウェア要件 112
16.3. アンダークラウドの前提条件 112
16.4. OCTAVIA デプロイメントのプランニング 112
16.4.1. Octavia の証明書とキーの設定 113
16.5. OCTAVIA のデプロイ 114
16.5.1. ロードバランシングネットワークの設定 114
16.5.2. 管理ポートの設定 114
16.5.3. director を使用した Octavia のデプロイ 114
16.6. HTTP ロードバランサーの設定 115
16.7. ロードバランサーの検証 115

第17章 IPV6 を使用したテナントネットワーク 119

17.1. IPV6 サブネットのオプション 119
17.1.1. Stateful DHCPv6 を使用した IPv6 サブネットの作成 120

第18章 テナントクォータの管理 123
18.1. L3 クォータオプション 123
18.2. ファイアウォールのクォータオプション 123
18.3. セキュリティーグループのクォータオプション 123
18.4. 管理クォータオプション 123

第19章 FIREWALL-AS-A-SERVICE (FWAAS) の設定 124
19.1. FWAAS の有効化 124
19.2. FWAAS の設定 125
19.3. ファイアウォールの作成 125

第20章 ALLOWED-ADDRESS-PAIRS の設定 127

第21章 レイヤー 3 高可用性の設定 128
21.1. 高可用性なしの OPENSTACK NETWORKING 128
21.2. レイヤー 3 高可用性の概要 128
21.2.1. フェイルオーバーの状況 129
21.3. テナントに関する留意事項 129
21.4. 背景の変更 129
21.4.1. neutron-server への変更 129
21.4.2. L3 エージェントへの変更 129
21.5. 設定手順 129
21.5.1. OpenStack Networking ノードの設定 130
21.5.2. 設定の確認 130

第22章 仮想デバイスの識別に対するタグの使用 131

第23章 レイヤー 2 ゲートウェイの使用 132
23.1. 概要 132
23.1.1. ネットワークトポロジー 132
23.1.2. 要件 133
23.2. ユースケース 1: 1 台の SR-IOV コンピュートノード上に 2 つのインスタンスがある場合 133
23.3. ユースケース 2: 別のコンピュートノード (複数) 上にインスタンスがある場合 135
23.4. ユースケース 3: ハードウェア VEB 上のインスタンスにソフトウェア VEB 上のインスタンスが接続する場合 135
23.5. ユースケース 4: SR-IOV PHYSICAL FUNCTION (PF) に接続されたインスタンスの場合 136
23.6. ユースケース 5: TOR スイッチが 2 つある場合 137
23.7. ユースケース 6 : 同じインターフェースを共有する異なる複数のネットワークに接続されたインスタンスの場合 138
23.8. SDN トポロジーの構築 139
23.9. OPENDAYLIGHT の設定 139
23.9.1. トランスポートゾーンの設定 140
23.10. OPENSTACK ネットワークの作成 142
23.11. レビュー 144

前書き

OpenStack Networking (コード名: **neutron**) は、Red Hat OpenStack Platform 14 のソフトウェア定義ネットワークのコンポーネントです。

OpenStack Networking および SDN

ソフトウェア定義ネットワーク (SDN: Software-defined Networking) は、下層の機能を抽象化することでネットワーク管理者がネットワークサービスを管理できるようにするコンピューターネットワーキングの方法の 1 つです。サーバーのワークロードを仮想環境に移行しても、それらのサーバーがデータの送受信のためのネットワーク接続を必要とすることに変わりありません。SDN は、ルーターやスイッチなどのネットワーク装置を同じ仮想化領域に移動することで、このニーズに対応します。すでにネットワークの基本概念に精通している場合には、SDN によって接続されるサーバーと同様に、ネットワークが仮想化されていると考えても飛躍ではありません。

本書は、パート 1 で管理者が基本的な管理およびトラブルシューティングタスクについて理解するための内容を紹介し、パート 2 で OpenStack Networking の高度な機能についてクックブック形式で解説しています。ネットワークの一般概念をすでに把握している場合には、本書の内容は理解しやすいはずです (ネットワークに精通されていない場合には、パート 1 のネットワークの一般的な概要が役立ちます)。

本ガイドの構成

- **前書き**: SDN に関連した大規模な組織内の政治について説明し、一般的なネットワークの概念について簡単に紹介します。
- **パート 1**: 一般的な管理タスクと基本的なトラブルシューティングのステップを説明します。
  - ネットワークリソースの追加と削除
  - 基本的なネットワークのトラブルシューティング
  - テナントネットワークのトラブルシューティング
- **パート 2**: 以下のように、高度な OpenStack Networking 機能についてクックブック形式のシナリオがまとめられています。
  - 仮想ルーターのレイヤー 3 高可用性の設定
  - DVR およびその他の Neutron 機能の設定

仮想ネットワークをめぐる組織内の政治

ソフトウェア定義ネットワーク (SDN: Software-defined Networking) により、エンジニアは仮想ルーターやスイッチを OpenStack または RHV ベースの仮想化環境にデプロイすることが可能です。また、SDN により、コンピューター間におけるデータパケット転送は従来とは異なる領域に移行されます。これらのルーターやスイッチは、以前はさまざまなケーブルで接続された物理デバイスでしたが、SDN を使用するとボタンを数回クリックするだけでデプロイして稼働させることができます。

大規模な仮想化環境では、ソフトウェア定義ネットワーク (SDN) を採用すると、組織内政治の緊張を生む可能性があります。高度なネットワーク概念に精通していない仮想化エンジニアは、クラウドデプロイメントの仮想ルーターやスイッチの管理を突然迫られ、IP アドレスの確保、VLAN の分離、サブネット化について合理的に判断する必要があります。仮想化エンジニアがこのような状況に直面している一方で、ネットワークエンジニアは、かつて自分たちの独占領域であった技術について他のチームが協議しているのを目の当たりにするため、動揺したり、雇用の先行き不安が生じたりする可能性があり

ます。このような境界があると、トラブルシューティングもかなり複雑になります。システムがダウンして相互接続できない場合には、仮想化エンジニアはパケットが物理スイッチに到達していることを確認した時点で、そのトラブルシューティングをネットワークエンジニアに引き継ぐべきでしょうか。

このような緊張は、仮想ネットワークを物理ネットワークの延長と考えると、容易に緩和することができます。デフォルトゲートウェイ、ルーター、サブネットには、物理ネットワークと同じ概念が適用され、ネットワークはすべて TCP/IPを使用して稼働することには変わりありません。

この問題に対して、どのような政治的対応を図るとしても、技術的な手段で対処することもできます。たとえば、Cisco の Nexus 製品を使用することで、OpenStack オペレーターは精通した Cisco NX-OS を実行する仮想ルーターをデプロイすることができます。これにより、ネットワークエンジニアは、これまで既存の物理 Cisco ネットワーク装置で行ってきた方法で、ネットワークポートにログインし、管理することができます。または、ネットワークエンジニアが仮想ネットワークを管理しない場合でも、初期の段階からネットワークエンジニアを交えて進めていくことが賢明です。OpenStack のノードには、物理ネットワークのインフラストラクチャーが必要な点は変わりなく、IP アドレスの割り当て、VLAN のトランク接続、VLAN をトランク接続するためのスイッチポートの設定を行う必要があります。トラブルシューティングに加えて、両チーム間での広範囲にわたる連携が要求されます。たとえば、仮想マシンの MTU サイズを調整する場合には、全仮想/物理スイッチおよびルーターなど、エンドツーエンドで実行する必要があり、両チーム間で慎重に準備して変更を行っていく必要があります。

ネットワークエンジニアは、引き続き、仮想化デプロイメントにおける極めて重要な役割を果たします。SDN の導入後にはその重要性はさらに高まります。業務が複雑化するため、ネットワークエンジニアのスキルを確実に活用していかなければなりません。特に問題発生時には、ネットワークエンジニアの知識が必要になります。

第1章 ネットワークの概要

1.1. ネットワークの仕組み

ネットワーク という用語は、コンピューター間で情報を移動させる動作のことを指します。最も基本的なレベルでは、ネットワークインターフェースカード (NIC) がインストールされた 2 つのマシンをケーブルでつなぐことで達成されます。

注記

OSI ネットワークモデルでは、これはレイヤー 1 に相当します。

3 台以上のコンピューターを使用する場合には、スイッチというデバイスを追加してこの構成をスケールアウトする必要があります。スイッチとは、追加のマシンを結線するための複数のイーサネットポートがついた専用デバイスです。スイッチを追加してマシンを結線すると、ローカルエリアネットワーク (LAN) と呼ばれるネットワークが完成します。

スイッチについての内容は、OSI モデルのレイヤー 2 が対象で、下層のレイヤー 1 よりもインテリジェントな機能が適用されます。各 NIC には、ハードウェアごとに一意に割り当てられる MAC アドレス番号があり、この番号を使用することにより、複数のマシンを同じスイッチに結線して、相互に認識できるようになります。スイッチは、どの MAC アドレスがどのポートにプラグインされるかのリストが管理するので、コンピューター間でデータ送信を試みると、スイッチはそれらの両方のコンピューターがどこに配置されているかを認識し、MAC アドレスからポートへのマッピングを記録する CAM (Content Addressable Memory) のエントリーを調整します。

1.1.1. VLAN

VLAN により、同じスイッチ上で実行されるコンピューターのネットワークトラフィックを分割することができます。言い換えると、別のネットワークのメンバーであるポートを設定することで、スイッチを論理的に分割することができます。つまり、セキュリティー上の理由別にトラフィックを分割できる小規模な LAN ということになります。たとえば、このポートがスイッチに合計で 24 個ある場合に、ポート 1 - 6 は **VLAN200**に所属し、ポート 7 - 18 は **VLAN201** に所属するなどと分けることができます。ここで、**VLAN200** に接続されているコンピューターは、**VLAN201** のコンピューターと完全に分割でき、直接通信できなくなります。任意で、2 つの別個の物理スイッチであるかのように、トラフィックをルーター経由で転送させるようにすることも可能です (このように考えると便利です)。このような場合には、VLAN 同士での通信の制御にファイアウォールも便利になります。

1.2. 2 つの LAN の接続

2 つの別個のスイッチ上で稼働する LAN が 1 つずつあり、このスイッチ間で情報を共有させる場合には、設定オプションが 2 種類あります。

- **1 番目のオプション**: 802.1Q VLAN タグを使用して、両方の物理スイッチにまたがる単一の VLAN を設定します。この接続を機能させるには、ネットワークケーブルを 1 本用意して、各スイッチのポートに差し込んでから、それらのポートを 802.1Q タグが付いたポートとして設定します (このようなポートは **トランク** ポートとも呼ばれます)。基本的には、これで 2 つのスイッチが 1 つの大きな論理スイッチとして設定され、接続されているコンピューター同士が相互に検出できるようになります。このオプションの難点はスケーラビリティーで、オーバーヘッドの問題が発生することなくデイジーチェーン接続できるスイッチの数は限られています。

- **2 番目のオプション**: ルーターと呼ばれるデバイスを購入して、各スイッチからケーブルを接続します。これにより、ルーターは両スイッチで設定したネットワークを認識するようになりま

す。スイッチに差し込んだケーブルの両端には、ネットワークのデフォルトゲートウェイとして知られる IP アドレスが割り当てられます。デフォルトゲートウェイの「デフォルト」とは、宛先のマシンが同じ LAN 上にないことが明らかな場合のトラフィックの送信先のことです。各コンピューターにデフォルトゲートウェイを設定することで、トラフィック送信のために、他のネットワーク上にある全コンピューターを認識する必要がなくなります。これで、デフォルトゲートウェイのみにトラフィックが送信されるようになり、そこからの処理はルーターが行うようにします。ルーターは、どのネットワークがどのインターフェースに存在するかを把握しているため、指定の宛先に、問題なくパケットを送信することができます。ルーティングは、IP アドレスやサブネットなどの一般的に知られている概念と同様に、OSI モデルのレイヤー 3 で機能します。

注記

この概念は、インターネット自体の仕組みと同じです。さまざまな組織で稼働する多数の個別ネットワークはすべて、スイッチやルーターを使用して相互に接続しています。適切なデフォルトゲートウェイを辿ることで、最終的にトラフィックは適切な宛先に到着します。

1.2.1. ファイアウォール

ファイアウォールは、レイヤー 7 (実際のコンテンツを検査するレイヤー) を含む複数の OSI レイヤーにわたってトラフィックをフィルターすることができます。多くの場合、ファイアウォールはルーターと同じネットワークセグメントに存在し、全ネットワーク間で移動するトラフィックを制御します。そのためには、ファイアウォールが、ネットワークに出入りできるトラフィックを決定する事前定義済みのルールに基づいて制御を行います。これらのルールは粒度を高くすることが可能です。以下に例を示します。

「**VLAN200** のサーバーは、Web (HTTP) トラフィックを一方向のみに転送している場合に、木曜の午後のみ、**VLAN201** のコンピューターにだけ通信できるものとする」といった設定が可能です。

このようなルールを強化するために、一部のファイアウォールは、レイヤー 5 から 7 でディープパケットインスペクション (DPI) も実行し、パケットのコンテンツを検証して、パケットの実際の内容がパケットが主張する内容と同じであることを確認します。ハッカーは、トラフィックを実際の内容とは別のものとして転送して、密かにデータを抜き出すことが知られているため、DPI はこのような脅威を軽減する手段の 1 つとなっています。

1.3. OPENSTACK NETWORKING (NEUTRON)

OpenStack では、これと同じネットワーク概念が適用されており、ソフトウェア定義ネットワーク (SDN) として知られています。OpenStack Networking (neutron) のコンポーネントは、仮想ネットワーク機能向けの API を提供します。これには、スイッチ、ルーター、ファイアウォールが含まれます。仮想ネットワークインフラストラクチャーにより、インスタンスは相互に通信することができます。また、物理ネットワークを使用した外部との通信を許可することも可能です。Open vSwitch のブリッジは、仮想ポートをインスタンスに割り当て、送受信トラフィックを物理ネットワークに橋渡しします。

1.4. CIDR 形式の使用

一般的には、IP アドレスはサブネットのブロックにまず割り当てられます。たとえば、IP アドレスの範囲が **192.168.100.0 - 192.168.100.255** で、サブネットマスクが **255.555.255.0** の場合には、IP アドレス **254** 個分を割り当てることができます (最初と最後のアドレスは予約されています)。

これらのサブネットは、複数の方法で表現することができます。

- 一般的な使用法: サブネットアドレスは一般的に、サブネットマスクとネットワークアドレスを使用して表示されます。例を以下に示します。
  - ネットワークアドレス: 192.168.100.0
  - サブネットマスク: 255.255.255.0
- CIDR 形式の使用: この形式は、サブネットマスクをアクティブな合計ビット数に短縮します。たとえば、**192.168.100.0/24** では、**/24** は **255.255.255.0** の略式表現で、バイナリーに変換した際に反転したビット合計数のことを指します。たとえば、CIDR 形式は **NETMASK** の値ではなく **ifcfg-xxx** スクリプトで使用することができます。

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

第2章 OPENSTACK NETWORKING の概念

OpenStack Networking には、ルーティング、DHCP、メタデータなどのコアサービスを管理するシステムサービスがあります。これらのサービスが 1 つにまとまって、物理サーバーに割り当てられる概念的なロールであるコントローラーノードの概念に含まれます。物理サーバーは通常、**ネットワークノード** のロールが割り当てられ、インスタンス間のネットワークトラフィックのレイヤー 3 ルーティングを管理するタスクに特化して稼働します。OpenStack Networking では、このロールを実行する複数の物理ホストを指定することができ、ハードウェア障害が発生した場合に向けたサービスの冗長化が可能です。詳しい情報は、**「レイヤー 3 の高可用性」**の章を参照してください。

注記

Red Hat OpenStack Platform 11 では、コンポーザブルロールのサポートが追加され、ネットワークサービスをカスタムロール別に分類することができます。ただし、本ガイドでは、内容をわかりやすくするために、デプロイメントにはデフォルトのコントローラーノードを使用することを前提とします。

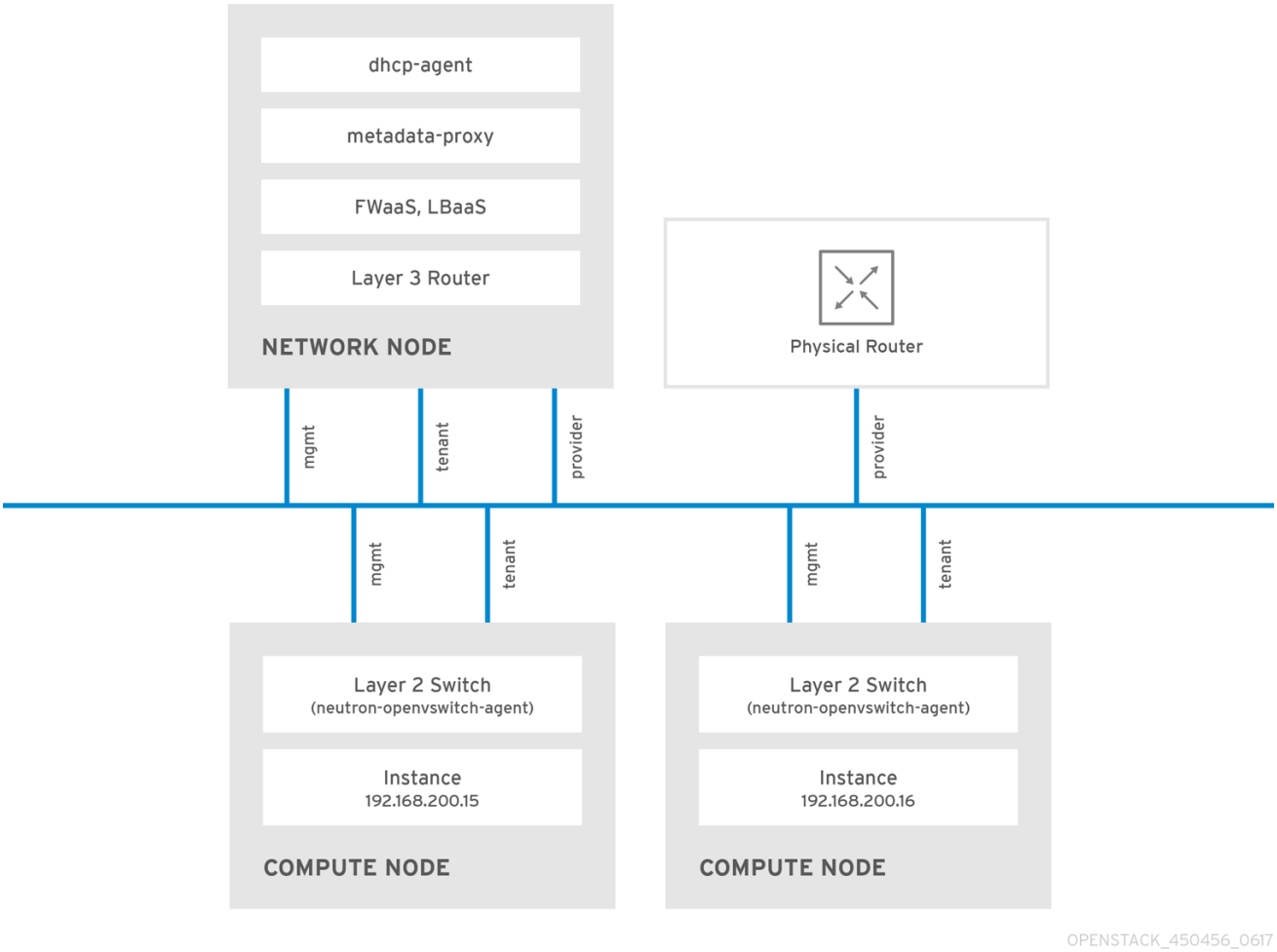
2.1. OPENSTACK NETWORKING (NEUTRON) のインストール

2.1.1. サポートされるインストール

OpenStack Networking コンポーネントは、Red Hat OpenStack Platform director デプロイメントの一部としてインストールされます。詳しくは、『director のインストールと使用方法』を参照してください。

2.2. OPENSTACK NETWORKING の図

以下の図は、専用の OpenStack Networking ノードが L3 ルーティングと DHCP の機能を果たし、**FWaaS** や **LBaaS** の高度なサービスを実行する OpenStack Networking のデプロイメントの例です。2 つのコンピュートノードは Open vSwitch (**openvswitch-agent**) を実行し、それぞれにテナントトラフィックと管理用の接続向けに物理ネットワークカードが 2 つ搭載されています。また、OpenStack Networking ノードには、プロバイダートラフィック専用の 3 枚目のネットワークカードがあります。



2.3. セキュリティーグループ

セキュリティーグループおよびルールを使用して、任意の neutron ポートに送信 (および任意の neustron ポートから受信) されるネットワークトラフィックの種別と方向をフィルタリングします。これにより、セキュリティーにもう 1 つレイヤーが追加されて、コンピュートインスタンスに存在するファイアウォールルールが補完されます。セキュリティーグループとは、1 つ以上のセキュリティールールを含むコンテナーオブジェクトのことで、1 つのセキュリティーグループで複数のコンピュートインスタンスへのトラフィックを管理することができます。Floating IP アドレス、OpenStack Networking LBaaS の仮想 IP、およびインスタンスのために作成されたポートは、セキュリティーグループに割り当てられます。セキュリティーグループが指定されなかった場合には、ポートは default のセキュリティーグループに割り当てられます。デフォルトでは、このグループは全受信トラフィックをドロップし、全送信トラフィックを許可します。追加のセキュリティールールを default セキュリティーグループに追加して動作を変更したり、必要に応じて新規セキュリティーグループを作成したりすることができます。

2.4. OPEN VSWITCH

Open vSwitch は、レガシーの Linux ソフトウェアブリッジと同様の、ソフトウェア定義ネットワーク (SDN: Software-Defined Networking) の仮想スイッチです。OVS は業界標準の **NetFlow**、**OpenFlow**、および **sFlow** をサポートし、仮想ネットワークにスイッチングサービスを提供します。Open vSwitch は、**STP**、**LACP**、**802.1Q VLAN タグ付け** などのレイヤー 2 (L2) 機能を使用することで物理スイッチとの統合も可能です。VXLAN と GRE を使用したトンネリングは、Open vSwitch のバージョン **1.11.0-1.el6** 以降でサポートされています。

注記

LACP は OVS ベースのボンディングでは使用しないでください。この構成は問題があるため、サポートされていません。この機能の代わりとして、**bond_mode=balance-slb** を使用することを検討してください。また、LACP は Linux ボンディングで使用することも可能です。この要件に関する技術的な詳細は、BZ#1267291 を参照してください。

注記

1 つのブリッジには単一のインターフェースまたは単一のボンディングのみをメンバーにすると、Open vSwitch でネットワークループが発生するリスクを緩和することができます。複数のボンディングまたはインターフェースが必要な場合には、複数のブリッジを設定することが可能です。

2.5. MODULAR LAYER 2 (ML2)

ML2 とは、OpenStack Havana リリースで導入された OpenStack Networking コアプラグインです。以前のモノリシックなプラグインのモデルに置き換わる、ML2 のモジュラー型設計により、複数のネットワーク技術を組み合わせた操作を同時に実行できます。モノリシックな Open vSwitch および Linux Bridge プラグインは、非推奨となり、削除されました。これらの機能は、代わりに ML2 メカニズムドライバーとして再実装されています。

注記

ML2 はデフォルトの OpenStack Networking プラグインで、Open vSwitch がデフォルトのメカニズムドライバーとして設定されています。

2.5.1. ML2 が導入された理由

以前は、OpenStack Networking のデプロイでは、実装時に選択したプラグインしか使用することができませんでした。たとえば、Open vSwitch プラグインを実行するデプロイは、Open vSwitch のみを単独にしか使えず、linuxbridge などの別のプラグインを同時に実行することは不可能でした。複数の異なる要件を伴う環境では、これは制限事項となっていました。

2.5.2. ML2 ネットワーク種別

ML2 ネットワーク種別では、複数のネットワークセグメントタイプを同時に操作することができます。また、これらのネットワークセグメントは、ML2 のマルチセグメントネットワークサポートを利用して相互接続することが可能です。ポートは接続されているセグメントに自動的にバインドされ、特定のセグメントにバインドする必要はありません。メカニズムドライバーにより、ML2 は、次のネットワークセグメントタイプをサポートします。

- flat
- GRE
- local
- VLAN
- VXLAN

ml2_conf.ini ファイルの ML2 セクションでは、さまざまな ***タイプの*** ドライバーが有効化されます。

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

2.5.3. ML2 メカニズムドライバー

共通のコードベースを使用するメカニズムとしてプラグインが再実装されています。このアプローチにより、コードの再利用が可能になる上、コードのメンテナンスとテストにおける複雑性が大幅に軽減されます。

注記

サポートされているメカニズムドライバーの一覧は、『リリースノート』参照してください。

ml2_conf.ini ファイルの ML2 セクションでさまざまなメカニズムドライバーが有効化されます。

```
[ml2]
mechanism_drivers = openvswitch,linuxbridge,l2population
```

注記

デプロイメントで Red Hat OpenStack Platform director を使用している場合には、これらの設定は director によって管理されるので、手動で変更すべきではありません。

注記

Neutron の Linux Bridge ML2 ドライバーおよびエージェントは Red Hat OpenStack Platform 11 で非推奨となりました。OpenStack Platform director によってデフォルトでデプロイされるのは、Open vSwitch (OVS) のプラグインです。一般的な用途の場合には、Red Hat はこのプラグインを推奨しています。

2.6. L2 POPULATION

L2 Population ドライバーはブロードキャスト、マルチキャスト、およびユニキャストのトラフィックを有効化して、大型のオーバーレイネットワークをスケールアウトします。デフォルトでは、Open vSwitch GRE および VXLAN がブロードキャストを各エージェントに複製します。これには、送信先のネットワークをホストしていないエージェントも含まれます。この設計には、多大なネットワークとプロセスのオーバーヘッドを受容する必要があります。L2 Population ドライバーにより導入される代替の設計は、ARP 解決および MAC 学習トラフィックのための部分的なメッシュを実装し、特定のネットワークをホストするノード間に、そのネットワーク用のトンネルも作成します。このトラフィックは、対象設定済みのユニキャストとしてカプセル化されることによって、必要なエージェントにのみ送信されます。

1. L2 population ドライバーを有効化するには、メカニズムドライバーのリストに追加します。また、少なくとも 1 つのトンネリングドライバーが有効化されている必要もあります (GRE と VXLAN のいずれか一方または両方)。ml2_conf.ini ファイルに適切な設定オプションを追加します。

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

注記

Neutron の Linux Bridge ML2 ドライバーおよびエージェントは Red Hat OpenStack Platform 11 で非推奨となりました。OpenStack Platform director によってデフォルトでデプロイされるのは、Open vSwitch (OVS) のプラグインです。一般的な用途の場合には、Red Hat はこのプラグインを推奨しています。

2. openvswitch_agent.ini ファイルで L2 Population を有効化します。これは、L2 エージェントを実行する各ノードで有効化する必要があります。

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

注記

ARP 応答フローをインストールするには、**arp_responder** フラグを設定する必要があります。以下に例を示します。

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

2.7. OPENSTACK NETWORKING サービス

Red Hat OpenStack Platform にはデフォルトで、ML2 および Open vSwitch のプラグインと統合してデプロイメントのネットワーク機能を提供するコンポーネントが含まれています。

2.7.1. L3 エージェント

L3 エージェントは **openstack-neutron** パッケージに含まれています。ネットワーク名前空間は、各プロジェクトに独自の分離されたレイヤー 3 ルーターを提供するのに使用されます。レイヤー 3 ルーターは、トラフィックを誘導し、レイヤー 2 ネットワーク向けのゲートウェイサービスを提供します。L3 エージェントはこれらのルーターの管理を支援します。L3 エージェントをホストするノードでは、外部ネットワークに接続されたネットワークインターフェースに手動で IP アドレスを設定することはできません。代わりに、OpenStack Networking で利用可能な外部ネットワークの IP アドレスの範囲内で指定する必要があります。これらの IP アドレスは、内部ネットワークと外部ネットワークの間を接続するルーターに割り当てられます。選択した範囲は、デプロイメント内の各ルーターに一意の IP アドレスと、必要な各 Floating IP を指定するのに十分な大きさである必要があります。

2.7.2. DHCP エージェント

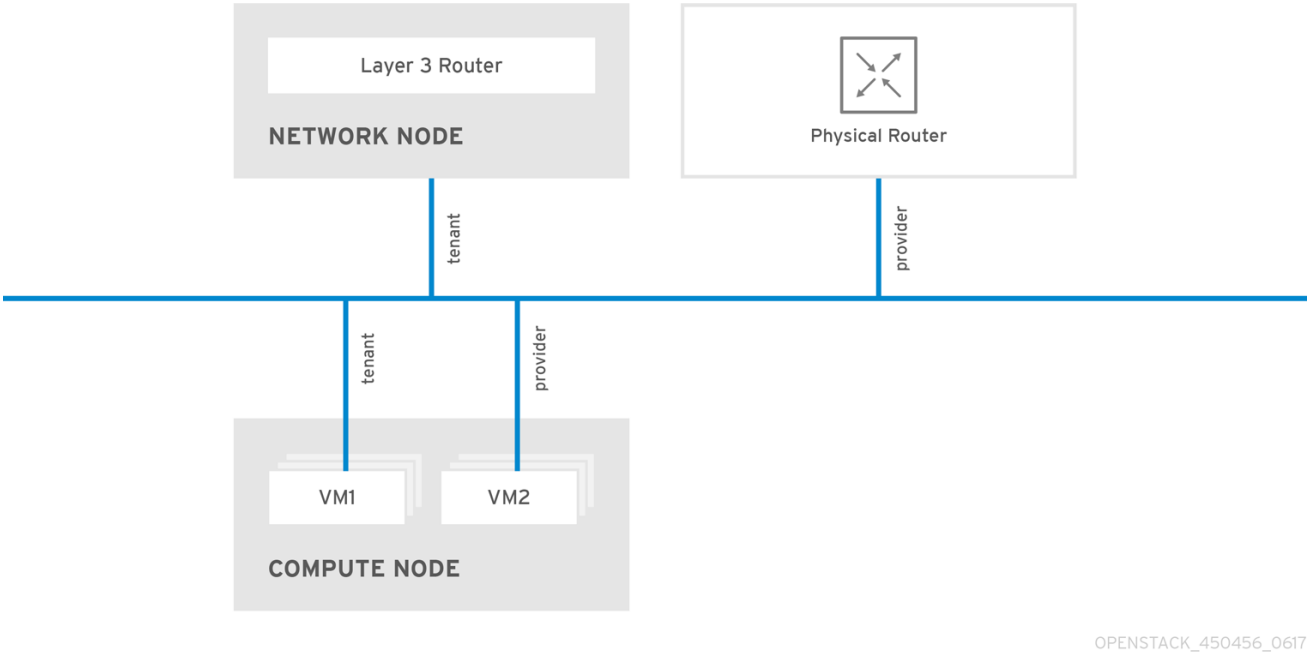
OpenStack Networking DHCP エージェントは、各プロジェクトのサブネットが DHCP サーバーとして機能するために作成されるネットワークの名前空間を管理します。各名前空間は、ネットワーク上で実行される仮想マシンへの IP アドレス確保が可能な dnsmasq プロセスを実行します。サブネットの作成時にこのエージェントが有効化されて稼働している場合には、そのサブネットにはデフォルトで DHCP が有効化されます。

2.7.3. Open vSwitch エージェント

Open vSwitch (OVS) neutron プラグインは、独自のエージェントを使用します。このエージェントは、各ノードで稼働し、OVS ブリッジを管理します。ML2 プラグインは専用のエージェントと統合して L2 ネットワークを管理します。デフォルトでは、Red Hat OpenStack Platform は **ovs-agent** を使用します。このエージェントは、OVS ブリッジを使用してオーバーレイネットワークを構築します。

2.8. テナントとプロバイダーネットワーク

以下の図には、テナントとプロバイダーネットワークの種別の概要と、それらが OpenStack Networking トポロジー全体でどのように対話するかを図解しています。



2.8.1. テナントネットワーク

テナントネットワークは、プロジェクト内の接続のためにユーザーが作成します。これらは、デフォルトで完全に分離され、他のプロジェクトとは共有されません。OpenStack Networking はさまざまな種別のテナントネットワークをサポートしています。

- **フラット**: 全インスタンスが同じネットワークに存在し、そのネットワークは、ホストと共有することも可能です。VLAN タグ付けやその他のネットワーク分離は行われません。

- **VLAN**: OpenStack Networking では、物理ネットワークにある VLAN に対応する VLAN ID (802.1Q タグ付け)を使用してユーザーが複数のプロバイダーネットワークまたはテナントネットワークを作成することができます。これにより、インスタンスは環境全体で相互に通信を行うことが可能になります。また、専用のサーバー、ファイアウォール、ロードバランサー、および同じレイヤー 2 上にあるその他のネットワークインフラストラクチャーと通信することもできます。

- **VXLAN および GRE のトンネル**: VXLAN および GRE は、ネットワークオーバーレイを使用して、インスタンス間のプライベートの通信をサポートします。OpenStack Networking ルーターは、トラフィックが GRE または VXLAN テナントネットワークの外部に通過できるようにするために必要です。また、ルーターは、直接接続されたテナントネットワークを外部ネットワーク (インターネットを含む) に接続するのにも必要とされ、Floating IP アドレスを使用して外部ネットワークから直接インスタンスに接続する機能を提供します。

注記

テナントネットワークの QoS ポリシーを設定することが可能です。詳しくは、「10 章*Quality-of-Service (QoS) の設定*」を参照してください。

2.8.2. プロバイダーネットワーク

プロバイダーネットワークは OpenStack の管理者によって作成され、データセンター内の既存の物理

ネットワークに直接マップします。このカテゴリーの中で有用なネットワークタイプには、フラット (タグなし) と VLAN (802.1Q タグ付き) があります。ネットワーク作成プロセスの一環としてプロバイダーネットワークがテナント間で共有されるように許可することができます。

2.8.2.1. フラットプロバイダーネットワーク

フラットプロバイダーネットワークを使用してインスタンスを直接外部ネットワークに接続することができます。これは、複数の物理ネットワーク (例: **physnet1** および **physnet2**)、さらにそれぞれ別の物理インターフェース (**eth0 → physnet1** および **eth1 → physnet2**) があり、各コンピュートおよびネットワークノードをこれらの外部ネットワークに接続する予定にしている場合に便利です。単一のインターフェース上に VLAN タグ付けされたインターフェースを複数使用する場合には、「VLAN プロバイダーネットワークの使用」の項を参照してください。

2.8.2.2. コントローラーノードの設定

1. **/etc/neutron/plugin.ini** (**/etc/neutron/plugins/ml2/ml2_conf.ini** へのシンボリックリンク) を編集して、既存の値リストに **flat** を追加し、**flat_networks** を * に設定します。以下に例を示します。

```
type_drivers = vxlan,flat
flat_networks =*
```

2. フラットネットワークとして外部ネットワークを作成して、設定済みの **physical_network** に関連付けます。このネットワークを共有ネットワークとして設定すると (**--shared** を使用)、他のユーザーはそのネットワークに直接接続されたインスタンスを作成することができます。

```
neutron net-create public01 --provider:network_type flat --provider:physical_network physnet1 --
router:external=True --shared
```

3. **neutron subnet-create** またはダッシュボードを使用してサブネットを作成します。以下に例を示します。

```
# neutron subnet-create --name public_subnet --enable_dhcp=False --allocation_pool
start=192.168.100.20,end=192.168.100.100 --gateway=192.168.100.1 public01 192.168.100.0/24
```

4. **neutron-server** サービスを再起動して、変更を適用します。

```
systemctl restart neutron-server
```

2.8.2.3. ネットワークノードとコンピュートノードの設定

ネットワークノードとコンピュートノードで以下の手順を実行すると、ノードは外部ネットワークに接続され、インスタンスが外部ネットワークと直接通信できるようになります。

1. 外部ネットワークのブリッジ (br-ex) を作成して、関連付けたポート(eth1) を追加します。

/etc/sysconfig/network-scripts/ifcfg-br-ex に外部ネットワークのブリッジを作成します。

```
DEVICE=br-ex
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

/etc/sysconfig/network-scripts/ifcfg-eth1 で **eth1** が **br-ex** に接続するように設定します。

```
DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

ノードを再起動するか、ネットワークサービスを再起動して、変更を適用します。

2. **/etc/neutron/plugins/ml2/openvswitch_agent.ini** で物理ネットワークを設定して、ブリッジをその物理ネットワークにマッピングします。

```
bridge_mappings = physnet1:br-ex
```

注記

ブリッジマッピングについての詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

3. ネットワークノードとコンピュートノードの両方で **neutron-openvswitch-agent** サービスを再起動して、変更を適用します。

```
systemctl restart neutron-openvswitch-agent
```

2.8.2.4. ネットワークノードの設定

1. **/etc/neutron/l3_agent.ini** で **external_network_bridge =** に空の値を設定します。

以前のバージョンでは、OpenStack Networking は、外部のネットワークとの接続に単一のブリッジのみを使用する場合に **external_network_bridge** を使用していました。今回のリリースでは、この値に空白の文字列を設定することができるようになり、複数の外部ネットワークブリッジを使用できます。この設定により、OpenStack Networking は各外部ブリッジから **br-int** へのパッチポートを作成します。

```
# Name of bridge used for external network traffic. This should be set to
# empty value for the linux bridge
external_network_bridge =
```

2. **neutron-l3-agent** を再起動して、変更を有効にします。

```
systemctl restart neutron-l3-agent
```

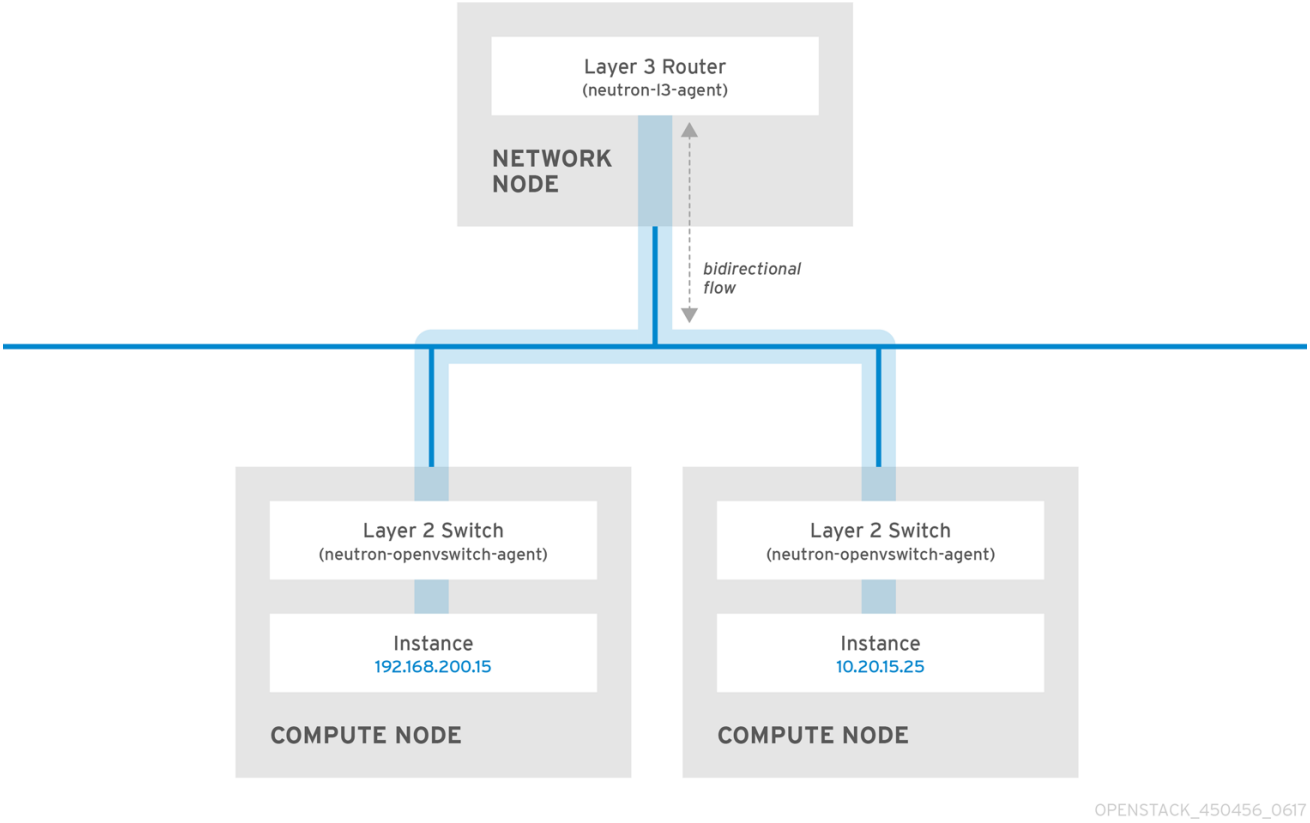
注記

複数のプロバイダーネットワークが存在する場合には、ネットワークごとに異なる物理インターフェースとブリッジを使用して外部ネットワークに接続すべきです。**ifcfg-*** スクリプトを適切に設定し、**bridge_mappings** オプションでコンマ区切りリストでネットワーク別のマッピングを指定する必要があります。詳しくは、「11章 *ブリッジマッピングの設定*」を参照してください。

2.9. レイヤー 2 およびレイヤー 3 ネットワーク

仮想ネットワークを設計する場合には、トラフィックの大半がどこに発生するかを予測する必要があります。ネットワークトラフィックは、異なるネットワーク間よりも同じ論理ネットワーク内のほうが早く移動します。これは、(異なるサブネットを使用した) 論理ネットワーク間のトラフィックではルーターを経由する必要があり、追加でレイテンシーが発生するためです。

以下の図で、別の VLAN 上にあるインスタンス間を流れるネットワークトラフィックを見てみましょう。

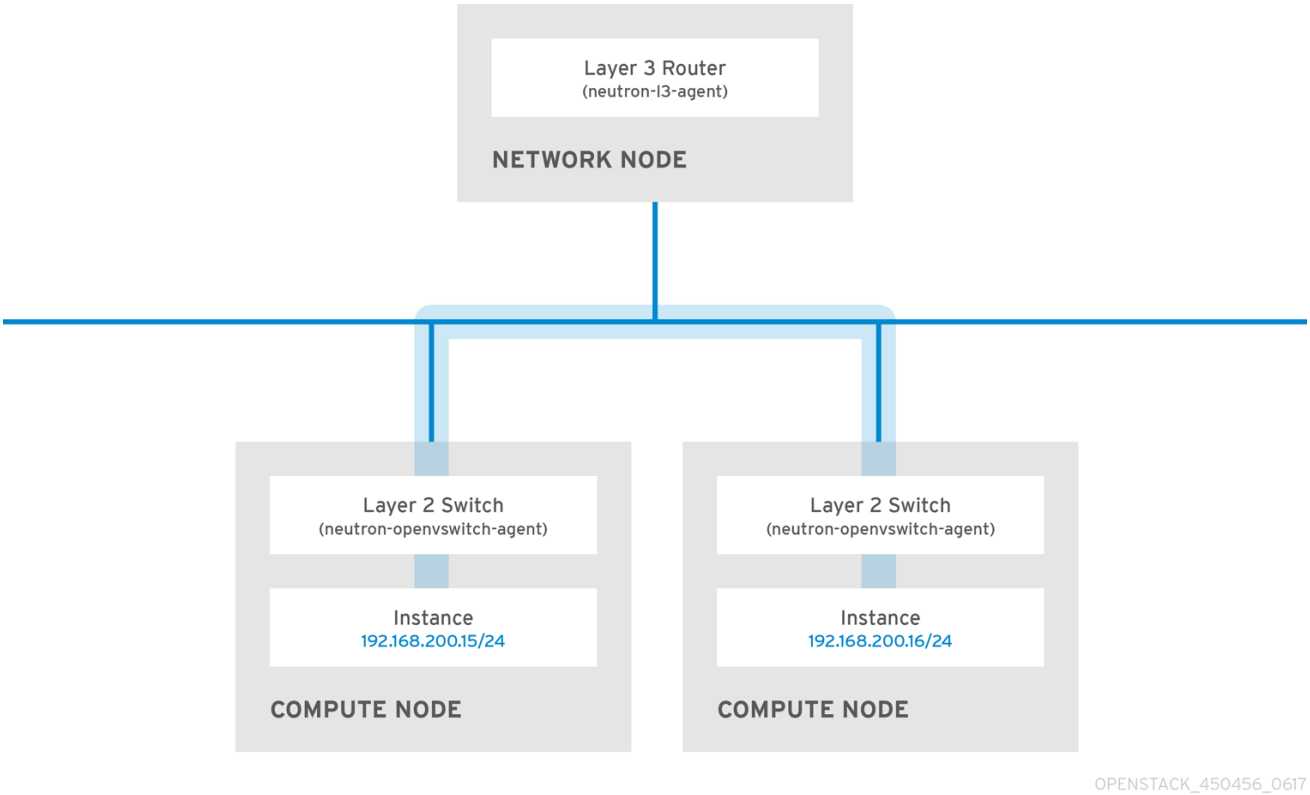


注記

パフォーマンスが高いハードウェアルーターでも、この設定にレイテンシーが加えられます。

2.9.1. 可能な範囲でのスイッチの使用

スイッチは、ネットワークの下層 (レイヤー 2) で行われるため、レイヤー 3 で行われるルーティングよりもはるかに早く機能することが可能です。頻繁に通信のあるシステム間では、ホップ数をできる限り少なくすることを推奨します。たとえば、この図ではスイッチ付きのネットワークで 2 つの物理ノードをつなげ、ナビゲーションにルーターを使用せずに 2 つのインスタンスが直接通信できるようにしている点が描写されています。これらのインスタンスで同じサブネットが共有されており、同じ論理ネットワークに存在することが分かります。



別のノードにあるインスタンスが同じ論理ネットワークにあるかのように通信できるようにするためには、VXLAN または GRE などのカプセル化されたトンネルを使用する必要があります。トンネルヘッダーに必要な追加のビットに対応するためにエンドツーエンドでの MTU サイズの調節を検討することを推奨します。そうしなかった場合には、断片化が原因でネットワークのパフォーマンスが悪影響を受ける可能性があります。詳しい情報は、**「MTU の設定」**を参照してください。

VXLAN オフロード機能を搭載したサポート対象のハードウェアを使用すると、VXLAN トンネリングのパフォーマンスをさらに向上させることができます。完全な一覧は「Network Adapter Feature Support in RHEL」の記事を参照してください。

パート I. 一般的なタスク

一般的な管理タスクと基本的なトラブルシューティングのステップを説明します。

第3章 一般的な管理タスク

OpenStack Networking (neutron) は、Red Hat OpenStack Platform のソフトウェア定義ネットワークのコンポーネントです。仮想ネットワークインフラストラクチャーにより、インスタンスと外部の物理ネットワークとの間の接続が可能になります。

本項では、お使いの Red Hat OpenStack Platform デプロイメントに合わせてサブネットやルーターを追加/削除するなど、一般的な管理タスクについて説明します。

3.1. ネットワークの作成

インスタンスが相互に通信する場所を提供し、DHCP を使用して IP アドレスを受け取るためにネットワークを作成します。ネットワークは、Red Hat OpenStack Platform デプロイメントまたは別の場所にある物理ネットワークなどの外部ネットワークと統合することも可能です。このような統合を行うと、インスタンスは外部のシステムと通信できるようになります。詳しい情報は、**「物理ネットワークのブリッジ」**を参照してください。

ネットワークの作成時には、ネットワークで複数のサブネットをホスト可能である点を認識しておくことが重要です。これは、明らかに異なるシステムを同じネットワークでホストする場合やそれらのシステムを分離する手段を希望する場合に役立ちます。たとえば、Web サーバーのトラフィックが 1 つのサブネット上のみで伝送される一方で、データベースのトラフィックは別のサブネット上を通過するように指定することができます。サブネットは相互に分離され、別のサブネットと通信する必要のあるインスタンスのトラフィックは、ルーターによって転送される必要があります。大量のトラフィックを必要とする複数のシステムを、同じサブネットに配置すると、ルーティングの必要がなく、それに伴うレイテンシーや負荷を回避することができます。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ネットワーク**を選択します。

2. **+ネットワークの作成** をクリックして、以下を指定します。

フィールド	説明
ネットワーク名	そのネットワークが果たす役割に基づいた説明的な名前。ネットワークを外部の VLAN と統合する場合には、名前に VLAN ID 番号を追記することを検討してください。たとえば、このサブネットで HTTP Web サーバーをホストし、VLAN タグが **122** の場合には **webservers_122** とします。また、ネットワークトラフィックをプライベートにして、外部ネットワークと統合しない場合には **internal-only** とします。
管理状態	このオプションにより、ネットワークを即時に利用可能にするかどうかを制御することができます。ネットワークを作成して、Down の状態に維持することが可能です。この場合、そのネットワークは論理的には存在しますが、アクティブではありません。このような設定は、そのネットワークを直ちに稼働させない場合に有用です。

3. **次へ** ボタンをクリックして **サブネット** タブで以下を指定します。

フィールド	説明
サブネットの作成	サブネットを作成するかどうかを決定します。たとえば、ネットワーク接続のないプレースホルダーとしてこのネットワークを維持する場合には、サブネットを作成しない方がよいでしょう。
サブネット名	サブネットの説明的な名前を入力します。
ネットワークアドレス	IP アドレス範囲とサブネットマスクが 1 つの値としてまとめられた CIDR 形式でアドレスを入力します。アドレスを判断するには、サブネットマスクでマスキングされたビット数を算出して、IP アドレス範囲の値に追記します。たとえば、サブネットマスク 255.255.255.0 のマスクビットは 24 で、このマスクを IPv4 アドレス範囲 192.168.122.0 で使用するには、アドレスは 192.168.122.0/24 に指定します。
IP バージョン	インターネットプロトコルバージョンを指定します (有効なタイプは IPv4 または IPv6)。**ネットワークアドレス** フィールドの IP アドレス範囲は、選択したバージョンと一致する必要があります。
ゲートウェイ IP	デフォルトのゲートウェイに指定したルーターのインターフェースの IP アドレス。このアドレスは、外部ロケーションを宛先とするトラフィックのルーティングのネクストホップとなり、ネットワークアドレスの値で指定した範囲内でなければなりません。たとえば、CIDR ネットワークアドレスが 192.168.122.0/24 の場合には、デフォルトのゲートウェイは 192.168.122.1 となる可能性が高くなります。
ゲートウェイなし	転送を無効にして、サブネットを分離した状態を保ちます。

4. **次へ** をクリックして **DHCP** オプションを指定します。

- **DHCP 有効**: そのサブネットの DHCP サービスを有効にします。DHCP により、インスタンスへの IP 設定の割り当てを自動化することができます。
- **IPv6 アドレス**: IPv6 ネットワークを作成する際の設定モード。IPv6 アドレスと追加の情報をどのように割り当てるかを指定します。
  - **オプション指定なし**: IP アドレスを手動で設定する場合または OpenStack が対応していない方法を使用してアドレスを割り当てる場合にはこのオプションを選択します。
  - **SLAAC (Stateless Address Autoconfiguration)**: インスタンスは、OpenStack Networking ルーターから送信されるルーター広告 (RA) メッセージに基づいて IPv6 アドレスを生成します。この設定を使用すると、ra_mode が slaac、address_mode が slaac に設定された OpenStack Networking サブネットが作成されます。
  - **DHCPv6 stateful**: インスタンスは、OpenStack Networking DHCPv6 サービスから、IPv6

アドレスや追加のオプション (例: DNS) を受信します。この設定を使用すると、ra_mode が dhcpv6-stateful、address_mode が dhcpv6-stateful に設定されたサブネットが作成されます。

  - **DHCPv6 stateless**: インスタンスは、OpenStack Networking ルーターから送信されるルーター広告 (RA) メッセージに基づいて IPv6 アドレスを生成します。追加のオプション (例: DNS) は、OpenStack Networking DHCPv6 サービスから割り当てられます。この設定を使用すると、ra_mode が dhcpv6-stateless、address_mode が dhcpv6-stateless に設定されたサブネットが作成されます。

- **割り当てプール**: DHCP によって割り当てられる IP アドレスの範囲。たとえば、192.168.22.100,192.168.22.100 という値を指定すると、その範囲内で使用可能なアドレスはすべて **割り当ての対象** として考慮されます。

- **DNS 名前サーバー**: ネットワーク上で利用可能な DNS サーバーの IP アドレス。DHCP はこれらの IP アドレスをインスタンスに割り当てて名前解決します。

- **追加のルート設定**: 静的ホストルート。最初に CIDR 形式の宛先ネットワークを指定し、その後にルーティングに使用する必要のあるネクストホップを指定します (例: 192.168.23.0/24, 10.1.31.1)。静的ルートをインスタンスに分散する必要がある場合には、この値を指定します。

5. **作成** をクリックします。

作成が完了したネットワークは、**ネットワーク** タブに表示されます。必要に応じて、**編集** をクリックしてオプションを変更することができます。これで、インスタンスの作成時には、このサブネットを使用できるようになりました。作成後には、指定した DHCP オプションがインスタンスに適用されます。

3.2. 高度なネットワークの作成

管理者は、**管理** の画面からネットワークを作成する際に高度なネットワークオプションを使用することができます。これらのオプションは、使用するネットワークタイプを定義し、テナントの指定を可能にします。

1. Dashboard で、 **管理 > ネットワーク > ネットワークの作成 > プロジェクト**を選択します。 **プロジェクト** から新規ネットワークをホストする先のプロジェクトを選択します。

2. **プロバイダーネットワーク種別** でオプションを確認します。

- **ローカル**: トラフィックはローカルの Compute ホストに残り、実質的には外部のネットワークから分離されます。

- **フラット**: トラフィックは単一のネットワーク上に残り、ホストと共有することも可能となります。VLAN タグ付けやその他のネットワーク分離は行われません。

- **VLAN**: 物理ネットワークに存在する VLAN に対応した VLAN ID を使用してネットワークを作成します。インスタンスは、同じレイヤー 2 VLAN 上のシステムと通信することが可能となります。

- **GRE**: 複数のノードにまたがるネットワークオーバーレイを使用して、インスタンス間のプライベート通信を行います。オーバーレイの外部に送信されるトラフィックは、ルーティングする必要があります。

- **VXLAN**: GRE と同様に、複数のノードにまたがるネットワークオーバーレイを使用して、インスタンス間のプライベート通信を行います。オーバーレイの外部に送信されるトラフィックは、ルーティングする必要があります。

ネットワークの作成 をクリックし、プロジェクトのネットワークトポロジーをチェックして、ネットワークが適切に作成されたことを確認します。

3.3. ネットワークルーティングの追加

新規ネットワークからのトラフィックのルーティングを許可するには、そのサブネットを既存の仮想ルーターへのインターフェースとして追加する必要があります。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ルーター**を選択します。

2. **ルーター** 一覧で仮想ルーターの名前をクリックしてから、**+インターフェースの追加** をクリックします。サブネット一覧では、新規サブネットの名前を選択します。このフィールドでインターフェースの IP アドレスを任意で指定することができます。

3. **インターフェースの追加** をクリックします。

ネットワーク上のインスタンスで、サブネットの外部のシステムとの通信ができるようになりました。

3.4. ネットワークの削除

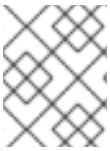
以前に作成したネットワークを削除する必要がある場合があります (例: ハウスキーピングやデコミッションプロセスの一環としての処理など)。ネットワークを正常に削除するには、まず始めにまだネットワークを使用しているインターフェースを削除または切断する必要があります。以下の手順で、関連するインターフェースと共にプロジェクト内のネットワークを削除する方法について説明します。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ネットワーク**を選択します。対象のネットワークのサブネットに関連付けられたルーターインターフェースをすべて削除します。インターフェースを削除するには、**ネットワーク** 一覧にある対象のネットワークをクリックして ID フィールドを確認し、削除するネットワークの ID 番号を特定します。そのネットワークの割り当て済みサブネットにはすべて、この値が **ネットワーク ID** フィールドに示されます。

2. **プロジェクト** > **ネットワーク** > **ルーター**を選択して、**ルーター** の一覧から対象の仮想ルーターの名前をクリックし、削除するサブネットに接続されているインターフェースを特定します。ゲートウェイ IP として機能していた IP アドレスで他のインターフェースと区別することができます。また、以前のステップで書き留めた ID とインターフェースのネットワーク ID が一致するかどうかを確認することによってさらに識別することができます。

3. 対象のインターフェースの **インターフェースの削除** ボタンをクリックします。

プロジェクト > ネットワーク > ネットワークを選択して、対象のネットワークの名前をクリックします。対象のサブネットのサブネットの削除ボタンをクリックします。



注記

この時点でサブネットをまだ削除できない場合には、インスタンスがすでにそのサブネットを使用していないかどうかを確認してください。

4. **プロジェクト** > **ネットワーク** > **ネットワーク**を選択し、削除するネットワークを選択します。

5. **ネットワークの削除** をクリックします。

3.5. テナントのネットワークの削除

以前のリリースでは、ネットワーク、ルーター、ポートなど、過去にプロジェクトに割り当てられた古

いリソースがプロジェクトの削除後にも残っている場合がありました。以前のリリースでは、これらの古いリソースは、正しい順序で慎重に手動で削除する必要がありました。Red Hat OpenStack Platform 12 は、この問題に対処し、**neutron purge** コマンドを使用して、特定のプロジェクトに以前に割り当てられた neutron リソースをすべて削除できるようになりました。

たとえば、削除前に **test-project** の neutron リソースを削除するには、以下を実行します。

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

# neutron purge 02e501908c5b438dbc73536c10c9aac0
Purging resources: 100% complete.
Deleted 1 security_group, 1 router, 1 port, 1 network.

# openstack project delete 02e501908c5b438dbc73536c10c9aac0
```

3.6. サブネットの作成

サブネットは、インスタンスにネットワーク接続を許可する手段です。インスタンスの作成プロセスの一環として、各インスタンスはサブネットに割り当てられるため、最適なインスタンスの配置を考慮してインスタンスの接続性の要件に対応することが重要です。サブネットは既存のネットワークで作成されます。OpenStack Networking のテナントネットワークでは、複数のサブネットをホストできることを念頭に入れておいてください。これは、明らかに異なるシステムを同じネットワークでホストする場合やそれらのシステムを分離する手段を希望する場合に役立ちます。たとえば、1 つのサブネットでは Web サーバーのトラフィックがのみが伝送されるようにする一方で、別のサブネットはデータベースのトラフィックが通過するように指定することができます。サブネットは相互に分離され、別のサブネットと通信する必要のあるインスタンスのトラフィックは、ルーターによって転送される必要があります。大量のトラフィックを必要とする複数のシステムを、同じサブネットに配置すると、ルーティングの必要がなく、それに伴うレイテンシーや負荷を回避することができます。

3.6.1. 新規サブネットの作成

Dashboard で **プロジェクト** > **ネットワーク** > **ネットワーク**を選択して、**ネットワーク** ビューで使用するネットワーク名をクリックします。

1. **サブネットの作成** をクリックして、以下を指定します。

フィールド	説明
サブネット名	サブネットの説明的な名前

フィールド	説明
ネットワークアドレス	IP アドレス範囲とサブネットマスクが 1 つの値に含まれた CIDR 形式のアドレス。アドレスを決定するには、サブネットマスクでマスクされるビット数を計算して、その値を IP アドレス範囲に追記します。たとえば、サブネットマスク 255.255.255.0 でマスクされるビット数は 24 です。このマスクを IPv4 アドレス範囲 192.168.122.0 に使用するには、アドレスを 192.168.122.0/24 と指定します。
IP バージョン	インターネットプロトコルのバージョン。有効な値は IPv4 または IPv6 です。ネットワークアドレスフィールドの IP アドレスの範囲は、選択したバージョンと一致する必要があります。
ゲートウェイ IP	デフォルトのゲートウェイに指定したルーターのインターフェースの IP アドレス。このアドレスは、外部ロケーションを宛先とするトラフィックのルーティングのネクストホップとなり、ネットワークアドレスの値で指定した範囲内でなければなりません。たとえば、CIDR ネットワークアドレスが 192.168.122.0/24 の場合には、デフォルトのゲートウェイは 192.168.122.1 となる可能性が高くなります。
ゲートウェイなし	転送を無効にして、サブネットを分離した状態を保ちます。

2. **次へ** をクリックして DHCP オプションを指定します。

- **DHCP 有効**: そのサブネットの DHCP サービスを有効にします。DHCP により、インスタンスへの IP 設定の割り当てを自動化することができます。
- **IPv6 アドレス**: IPv6 ネットワークを作成する際の設定モード。IPv6 アドレスと追加の情報をどのように割り当てるかを指定します。
  - **オプション指定なし**: IP アドレスを手動で設定する場合または OpenStack が対応していない方法を使用してアドレスを割り当てる場合にはこのオプションを選択します。
  - **SLAAC (Stateless Address Autoconfiguration)**: インスタンスは、OpenStack Networking ルーターから送信されるルーター広告 (RA) メッセージに基づいて IPv6 アドレスを生成します。この設定を使用すると、ra_mode が slaac、address_mode が slaac に設定された OpenStack Networking サブネットが作成されます。
  - **DHCPv6 stateful**: インスタンスは、OpenStack Networking DHCPv6 サービスから、IPv6 アドレスや追加のオプション (例: DNS) を受信します。この設定を使用すると、ra_mode が dhcpv6-stateful、address_mode が dhcpv6-stateful に設定されたサブネットが作成されます。
  - **DHCPv6 stateless**: インスタンスは、OpenStack Networking ルーターから送信されるルーター広告 (RA) メッセージに基づいて IPv6 アドレスを生成します。追加のオプション (例:

DNS) は、OpenStack Networking DHCPv6 サービスから割り当てられます。この設定を使用すると、ra_mode が dhcpv6-stateless、address_mode が dhcpv6-stateless に設定されたサブネットが作成されます。

- **割り当てプール**: DHCP によって割り当てられる IP アドレスの範囲。たとえば、192.168.22.100,192.168.22.100 という値を指定すると、その範囲内で使用可能なアドレスはすべて **割り当ての対象** として考慮されます。
- **DNS 名前サーバー**: ネットワーク上で利用可能な DNS サーバーの IP アドレス。DHCP はこれらの IP アドレスをインスタンスに割り当てて名前解決します。
- **追加のルート設定**: 静的ホストルート。最初に CIDR 形式の宛先ネットワークを指定し、その後にルーティングに使用する必要のあるネクストホップを指定します (例: 192.168.23.0/24, 10.1.31.1)。静的ルートをインスタンスに分散する必要がある場合には、この値を指定します。

3. **作成** をクリックします。

作成した新規サブネットは、ネットワークのサブネット一覧に表示されます。必要に応じて編集をクリックしてオプションを変更することができます。インスタンスの作成時には、このサブネットを使用するよう設定できるようになりました。作成後には、指定した DHCP オプションがインスタンスに適用されます。

3.7. サブネットの削除

使用しなくなったサブネットは削除することができます。ただし、インスタンスがまだそのサブネットを使用するように設定されている場合には、削除を試みても失敗し、Dashboard にエラーメッセージが表示されます。以下の手順では、ネットワーク内にある特定のサブネットを削除する方法を説明します。

Dashboard で **プロジェクト** > **ネットワーク** > **ネットワーク**を選択して、使用するネットワーク名をクリックします。対象のサブネットを選択して、**サブネットの削除** をクリックします。

3.8. ルーターの追加

OpenStack Networking は、SDN をベースとする仮想ルーターを使用したルーティングサービスを提供します。インスタンスが外部のサブネット (物理ネットワーク内のサブネットを含む) と通信するには、ルーターは必須です。ルーターとサブネットはインターフェースを使用して接続します。各サブネットにはルーターに接続するための独自のインターフェースが必要です。ルーターのデフォルトゲートウェイは、そのルーターが受信するトラフィックのネクストホップを定義します。そのネットワークは通常、仮想ブリッジを使用して、外部の物理ネットワークにトラフィックをルーティングするように設定されます。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ルーター**を選択し、**+ルーターの作成** をクリックします。

2. 新規ルーターの説明的な名前を入力し、**ルーターの作成** をクリックします。

3. **ルーター** 一覧に新たに追加されたルーターのエントリーの横にある **ゲートウェイの設定** をクリックします。

4. **外部ネットワーク** の一覧で、外部ロケーション宛のトラフィックを受信するネットワークを指定します。

5. **ゲートウェイの設定** をクリックします。ルーターを追加した後に行う次のステップでは、作成済みのサブネットがこのルーターを使用してトラフィックを受信できるように設定します。これは、サブネットとルーター間のインターフェースを作成することによって行います。

3.9. ルーターの削除

インターフェースが接続されていないルーターは削除することができます。以下の手順では、最初にルーターのインターフェースを削除してからルーター自体を削除するステップを説明します。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ルーター**を選択し、削除するルーター名をクリックします。

2. **内部インターフェース** タイプのインターフェースを選択します。**インターフェースの削除** をクリックします。

3. **ルーター** 一覧から対象のルーターを選択して **ルーターの削除** をクリックします。

3.10. インターフェースの追加

インターフェースにより、ルーターをサブネットと相互接続することができます。これにより、ルーターは、インスタンスを中継するサブネットの外部にある宛先にインスタンスが送信するトラフィックを転送することが可能となります。以下の手順では、ルーターのインターフェースを追加して、サブネットに接続します。以下の手順では、ネットワークトポロジー機能を使用します。この画面には、全仮想ルーターとネットワークを示した図が表示され、ネットワーク管理タスクを実行することができます。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ネットワークトポロジー**を選択します。

2. 管理するルーターを特定して **インターフェースの追加** をクリックします。

3. ルーターを接続するサブネットを指定します。オプションで IP アドレスを指定することができます。このアドレスを指定しておくと、インターフェースに対して ping を実行して成功した場合にはトラフィックのルーティングが想定通りに機能していることを確認できるので、テストやトラブルシューティングに役立ちます。

4. **インターフェースの追加** をクリックします。

ネットワークトポロジー の図が自動的に更新され、ルーターとサブネットの間の新規インターフェース接続が反映されます。

3.11. インターフェースの削除

ルーターがトラフィックを転送する必要がなくなった場合には、サブネットからインターフェースを削除することができます。以下の手順では、インターフェースの削除に必要なステップを説明します。

1. Dashboard で **プロジェクト** > **ネットワーク** > **ルーター**を選択します。

2. 削除するインターフェースをホストしているルーターの名前をクリックします。

3. (**内部インターフェース** タイプの) インターフェースを選択し、**インターフェースの削除** をクリックします。

3.12. IP アドレスの設定

本項に記載する手順に従って OpenStack Networking における IP アドレスの確保を管理することができます。

3.12.1. Floating IP アドレスプールの作成

Floating IP アドレスにより、ネットワークの受信トラフィックを OpenStack インスタンスに転送することができます。最初に、有効でルーティング可能な外部 IP アドレスのプールを定義して、それらの IP アドレスをインスタンスに動的に割り当てられるようにすると、OpenStack Networking は、特定の Floating IP アドレス宛の受信トラフィックをすべて、その Floating IP アドレスが割り当てられたインスタンスにルーティングする必要があることを認識します。

注記

OpenStack Networking は、同じ IP 範囲/CIDR からの全プロジェクト (テナント) に、Floating IP アドレスを割り当てます。これは、Floating IP のサブネットはすべて、全プロジェクトまたはいずれかのプロジェクトで使用できることを意味します。この動作は、個別のプロジェクトごとのクォータを使用することで管理できます。たとえば、**ProjectA** と **ProjectB** のクォータのデフォルト値を **10** に設定して、**ProjectC** を **0** に設定することができます。

Floating IP アドレスを確保するプールは、外部サブネットの作成時に定義されます。サブネットが Floating IP アドレスのみをホストする場合には、**enable_dhcp=False** オプションを使用して DHCP の割り当てを無効にすることを検討してください。

```
# neutron subnet-create --name SUBNET_NAME --enable_dhcp=False --allocation_pool
start=IP_ADDRESS,end=IP_ADDRESS --gateway=IP_ADDRESS NETWORK_NAME CIDR
```

例:

```
# neutron subnet-create --name public_subnet --enable_dhcp=False --allocation_pool
start=192.168.100.20,end=192.168.100.100 --gateway=192.168.100.1 public 192.168.100.0/24
```

3.12.2. 特定の Floating IP アドレスの割り当て

nova コマンドを使用して、特定の Floating IP アドレスをインスタンスに割り当てることが可能です。

```
# nova floating-ip-associate INSTANCE_NAME IP_ADDRESS
```

以下の例では、Floating IP アドレスは corp-vm-01 という名前のインスタンスに割り当てられます。

```
# nova floating-ip-associate corp-vm-01 192.168.100.20
```

3.12.3. Floating IP アドレスの無作為な割り当て

Floating IP アドレスはインスタンスに動的に割り当てることができます。特定の IP アドレスを選択する代わりに、OpenStack Networking がプールからアドレスを 1 つ確保するように要求します。以前に作成したプールから Floating IP アドレスを確保します。

```
# neutron floatingip-create public
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 192.168.100.20                       |
| floating_network_id | 7a03e6bc-234d-402b-9fb2-0af06c85a8a3 |
| id                  | 9d7e2603482d                         |
| port_id             |                                      |
```

```
| router_id          |                                      |
| status             | ACTIVE                               |
| tenant_id          | 9e67d44eab334f07bf82fa1b17d824b6     |
+--------------------+--------------------------------------+
```

IP アドレスが確保されたら、特定のインスタンスに割り当てることができます。インスタンスに関連付けられているポートの ID を特定します (これは、インスタンスに確保されている Fixed IP と一致します)。このポート ID は、以下のステップでインスタンスのポート ID を Floating IP アドレスの ID に関連付けるのに使用します。3 番目のコラムの MAC アドレスがインスタンスの MAC アドレスと一致するようにすることで、正しいポート ID をさらに区別することが可能です。

```
# neutron port-list
+--------+------+-------------+-------------------------------------------------------+
| id     | name | mac_address | fixed_ips                                             |
+--------+------+-------------+-------------------------------------------------------+
| ce8320 |      | 3e:37:09:4b | {"subnet_id": "361f27", "ip_address": "192.168.100.2"} |
| d88926 |      | 3e:1d:ea:31 | {"subnet_id": "361f27", "ip_address": "192.168.100.5"} |
| 8190ab |      | 3e:a3:3d:2f | {"subnet_id": "b74dbb", "ip_address": "10.10.1.25"}|
+--------+------+-------------+-------------------------------------------------------+
```

neutron コマンドを使用して、対象となるインスタンスのポート ID に Floating IP アドレスを割り当てます。

```
# neutron floatingip-associate 9d7e2603482d 8190ab
```

3.13. 複数の FLOATING IP アドレスプールの作成

OpenStack Networking は 1 つの L3 エージェントあたり 1 つの Floating IP プールをサポートします。このため、L3 エージェントをスケールアウトすることによって追加の Floating IP プールを作成することができます。

注記

/etc/neutron/neutron.conf で **handle_internal_only_routers** の値が環境内の 1 つの L3 に対してのみ **True** に設定されていることを確認してください。このオプションにより、L3 エージェントは、外部以外のルーターのみを管理するようになります。

3.14. 物理ネットワークのブリッジ

以下の手順では、仮想ネットワークを物理ネットワークにブリッジして仮想インスタンスとの間の接続を可能にします。例として示した物理 eth0 インターフェースは br-ex ブリッジにマップされます。この仮想ブリッジは、物理ネットワークと仮想ネットワークを中継する機能を果たします。これにより、eth0 を通過するトラフィックはすべて、設定した Open vSwitch を使用してインスタンスに到達します。物理 NIC を仮想 Open vSwitch ブリッジにマッピングします (詳しくは、「11章 *ブリッジマッピングの設定*」を参照)。

注記

IPADDR、**NETMASK GATEWAY**、**DNS1** (ネームサーバー) は対象のネットワークと一致するように更新する必要があります。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
DEVICE=eth0
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

eth0 に以前確保されていた IP アドレスの情報を使用して仮想ブリッジを設定します。

```
# 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
```

インスタンスに Floating IP アドレスを割り当てて、物理ネットワークが利用できるようにすることができます。

第4章 IP アドレス使用のプランニング

OpenStack のデプロイメントでは、予想以上の数の IP アドレスが使用される可能性があります。本項は、必要な数のアドレスを適切に予測できるようにサポートすることを目的とし、さらに IP アドレスがどこで使用されるかを説明します。

注記

VIP (別称: 仮想 IP アドレス): 仮想 IP アドレスとは HA サービスをホストし、基本的に複数のコントローラーノード間で共有される IP アドレスです。

4.1. 複数の VLAN の使用

OpenStack のデプロイメントを計画する際には、デプロイメントに使用するサブネットの数から開始します。この数に基づいて、各アドレスをどのように使用していくかを割り当てることができるはずです。複数のサブネットを使用すると、システム間のトラフィックを VLAN に分割することができます。たとえば、通常は、管理または API トラフィックは、Web トラフィックにサービスを提供するシステムと同じネットワークを共有しないようにする必要があります。また VLAN 間のトラフィックは、ルーター経由で伝送する必要があります。この構成では、ファイアウォールを設置してトラフィックフローをさらに制御することができます。

4.2. VLAN トラフィックの分離

通常、異種のネットワークトラフィックをホストする場合には、別個の VLAN をトラフィックに割り当てます。たとえば、各種ネットワークごとに別の VLAN を指定します。当然ながら、外部ネットワークは外部の物理ネットワークにのみルーティングが可能です。本リリースでは、DHCP サービスは **director** により提供されます。

注記

本項で分離する VLAN は、すべての OpenStack デプロイメントで必要なわけではありません。たとえば、クラウドユーザーがアドホックの仮想ネットワークをオンデマンドで作成する必要がない場合には、テナントネットワークが必要ない可能性があります。各仮想マシンを他の物理システムと同じスイッチに直接接続する必要があるだけの場合には、コンピュートノードをプロバイダーネットワークに直接接続し、インスタンスがプロバイダーネットワークを直接使用するようにするだけでよいでしょう。

- **プロビジョニングネットワーク**: この VLAN は、PXE ブートで director を使用して新規ノードをデプロイするためだけに特化されています。OpenStack Orchestration (heat) は、OpenStack をオーバークラウドのベアメタルサーバーにインストールします。これらのサーバーは、物理ネットワークにアタッチされており、アンダークラウドのインフラストラクチャーから OpenStack Platform のインストールイメージを取得します。

- **内部 API ネットワーク**: 内部 API ネットワークは、API 通信、RPC メッセージ、データベース通信経由で OpenStack のサービス間の通信を行う際に使用します。さらに、このネットワークは、コントローラーノード間の稼働メッセージの通信にも使用されます。IP アドレスの割り当てを計画する際には、各 API サービスには独自の IP アドレスが必要である点を念頭に置いてください。具体的には、以下のサービスに IP アドレスが必要です。

  - vip-msg (ampq)
  - vip-keystone-int
  - vip-glance-int

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

注記

高可用性を使用する場合には、Pacemaker による仮想 IP アドレスの物理ノード間の移動が可能でなければなりません。

- **ストレージ**: Block Storage、NFS、iSCSI など。理想的には、これはパフォーマンスの関係上、別の物理イーサネットリンクに分離します。
- **Storage Management**: OpenStack Object Storage (swift) は、参加するレプリカノード間でデータオブジェクトを同期するためにこのネットワークを使用します。プロキシーサービスは、ユーザー要求と下層のストレージレイヤの間の仲介インターフェースとして機能します。プロキシーは、受信要求を受け取り、必要なレプリカの位置を特定して要求データを取得します。Ceph バックエンドを使用するサービスは、Ceph と直接対話せずにフロントエンドのサービスを使用するため、ストレージ管理ネットワーク経由で接続を確立します。RBD ドライバーは例外で、このトラフィックは直接 Ceph に接続する点に注意してください。
- **テナントネットワーク**: Neutron は、VLAN 分離 (各テナントネットワークがネットワーク VLAN) または VXLAN か GRE 経由のトンネリングを使用した独自のネットワークを各テナントに提供します。ネットワークトラフィックは、テナントのネットワークごとに分割されます。テナントネットワークには IP サブネットが割り当てられており、複数のテナントネットワークが同じアドレスを使用する場合があります。
- **外部**: 外部ネットワークは、パブリック API エンドポイントと Dashboard (horizon) への接続をホストします。オプションでこれと同じネットワークを SNAT に使用することもできますが、必須ではありません。実稼働環境のデプロイでは、大抵の場合、Floating IP アドレスと NAT に

別のネットワークが使用されます。

- **プロバイダーネットワーク**: これらのネットワークでは、インスタンスを既存のネットワークインフラストラクチャーにアタッチすることができます。フラットネットワークまたは VLAN タグでデータセンターの既存の物理ネットワークに直接マッピングするために、プロバイダーネットワークを使用することができます。これにより、インスタンスは、OpenStack Networking インフラストラクチャー外部のシステムと同じレイヤー 2 ネットワークを共有することができます。

4.3. IP アドレスの消費

以下のシステムは割り当てられた範囲からの IP アドレスを消費します。

- **物理ノード**: 物理 NIC ごとに IP アドレスが 1 つ必要です。物理 NIC に固有の機能を割り当てるのが一般的な慣習です。たとえば、管理トラフィックと NFS トラフィックは、それぞれ 1つずつ物理 NIC が割り当てられます (時には複数の NIC を使用して冗長化の目的で異なるスイッチ間を接続します)。

- **高可用性の仮想 IP (VIP)**: コントローラーノード間で共有されるネットワーク 1 つにつき 1 - 3 個程度割り当てられることが予想されます。

4.4. 仮想ネットワーク

これらの仮想リソースは、OpenStack Networking の IP アドレスを消費します。これらはクラウドインフラストラクチャーではローカルとみなされ、外部の物理ネットワークにあるシステムから到達可能である必要はありません。

- **テナントネットワーク**: 各テナントネットワークには、IP アドレスをインスタンスに割り当てるためのサブネットが必要です。

- **仮想ルーター**: サブネットに結線する各ルーターのインターフェースには IP アドレス が 1 つ必要です (DHCP が有効化されている場合には追加でもう 1 つアドレスが必要です)。

- **インスタンス**: 各インスタンスには、ホストされるテナントのサブネットからのアドレスが必要です。受信トラフィックが必要な場合には、指定の外部ネットワークから、追加の Floating IP アドレスを割り当てる必要があります。

- **管理トラフィック**: OpenStack サービスと API トラフィックを含みます。Red Hat OpenStack Platform 11 では、仮想 IP アドレスが要件が軽減され、代わりに全サービスが共有する仮想 IP が少数になりました。API、RPC、データベースサービスは、内部 APIの仮想 IP で通信します。

4.5. ネットワークプランの例

以下の例には、複数のサブネットに対応する、さまざまなネットワークを示しています。各サブネットには IP アドレスの範囲が 1 つ割り当てられます。

表4.1 サブネットプランの例

サブネット名	アドレス範囲	アドレス数	サブネットマスク
プロビジョニングネットワーク	192.168.100.1 - 192.168.100.250	250	255.255.255.0

サブネット名	アドレス範囲	アドレス数	サブネットマスク
内部 API ネットワーク	172.16.1.10 - 172.16.1.250	241	255.255.255.0
ストレージ	172.16.2.10 - 172.16.2.250	241	255.255.255.0
ストレージ管理	172.16.3.10 - 172.16.3.250	241	255.255.255.0
テナントネットワーク (GRE/VXLAN)	172.19.4.10 - 172.16.4.250	241	255.255.255.0
外部ネットワーク (Floating IP など)	10.1.2.10 - 10.1.3.222	469	255.255.254.0
プロバイダーネットワーク (インフラストラクチャー)	10.10.3.10 - 10.10.3.250	241	255.255.252.0

第5章 OPENSTACK NETWORKING ルーターポートのレビュー

OpenStack Networking の仮想ルーターは、ポートを使用してサブネットと相互接続します。これらのポートの状態を確認して、想定通りに接続されているかどうかを判断できます。

5.1. ポートの現在のステータスの表示

以下の手順では、特定のルーターに接続されたポートをすべて一覧表示し、ポートの状態 (DOWN または ACTIVE) を取得する方法を解説します。

1. r1 という名前のルーターにアタッチされているポートをすべて表示します。

```
# neutron router-port-list r1
```

結果の例:

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

2. ポートの ID (値は左のコラムに記載) に対して以下のコマンドを実行して、各ポートの詳細を表示します。コマンドの結果にはポートの **ステータス** が含まれており、以下の例ではステータスが **ACTIVE** であることが分かります。

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

結果の例:

```
+-----------------------+-----------------------------------------------------------------------------------------+
| Field                 | Value                                                                                   |
+-----------------------+-----------------------------------------------------------------------------------------+
| admin_state_up        | True                                                                                    |
| allowed_address_pairs |                                                                                         |
| binding:host_id       | node.example.com                                                                        |
| binding:profile       | {}                                                                                      |
| binding:vif_details   | {"port_filter": true, "ovs_hybrid_plug": true}                                          |
| binding:vif_type      | ovs                                                                                     |
| binding:vnic_type     | normal                                                                                  |
| device_id             | 49c6ebdc-0e62-49ad-a9ca-58cea464472f                                                    |
| device_owner          | network:router_interface                                                                |
| extra_dhcp_opts       |                                                                                         |
| fixed_ips             | {"subnet_id": "a592fdba-babd-48e0-96e8-2dd9117614d3", "ip_address":
"192.168.200.1"} |
| id                    | b58d26f0-cc03-43c1-ab23-ccdb1018252a                                                    |
| mac_address           | fa:16:3e:94:a7:df                                                                       |
```

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

ポートごとにこの手順を実行して、ステータスを取得します。

第6章 プロバイダーネットワークのトラブルシューティング

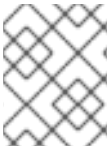
仮想ルーターとスイッチのデプロイメントは、ソフトウェア定義ネットワーク (SDN) としても知られており、一見すると (デプロイメントが) 複雑化しているように感じる場合がありますが、OpenStack Networking のネットワークの接続性をトラブルシュートする診断プロセスは、物理ネットワークの診断プロセスとよく似ています。VLAN を使用する場合は、仮想インフラストラクチャーは、全く別の環境ではなく、物理ネットワークのトランク接続による広帯域化と考えることができます。

本項の構成

- 基本的な ping 送信テスト
- VLAN ネットワークのトラブルシューティング
- テナントネットワーク内からのトラブルシューティング

6.1. 基本的な PING 送信テスト

ping コマンドは、ネットワーク接続の問題解析に役立つツールです。ping コマンドで返される結果は、ネットワーク接続に関する基本的な指標として機能しますが、実際のアプリケーショントラフィックをブロックするファイアウォールなど、すべての接続性の問題を完全に除外するわけではありません。ping コマンドは、指定の宛先にトラフィックを送信することで機能し、次に ping 送信の試行に問題がなかったかどうかを報告します。



注記

ping コマンドは、ICMP トラフィックが途中にあるファイアウォールを通過できることを想定します。

ping テストは、ネットワークの問題が発生しているマシンから実行すると最も有効です。そのため、マシンが完全にオフラインの場合には、VNC 管理コンソール経由でコマンドラインに接続する必要がある場合があります。

たとえば、以下の ping のテストコマンドを成功させるには、複数のネットワークインフラストラクチャー層を検証する必要があります。つまり、名前の解決、IP ルーティング、ネットワークスイッチのすべてが正常に機能していなければなりません。

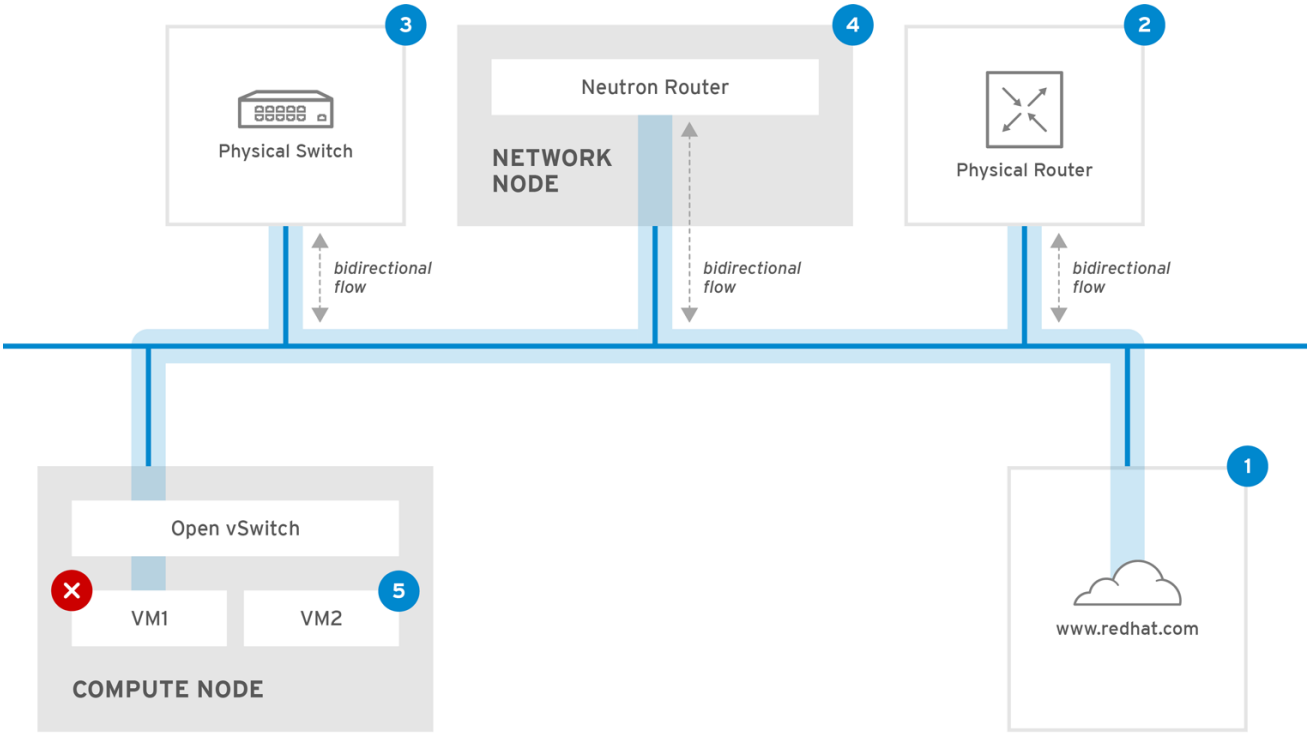
```
$ ping www.redhat.com

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
```

ping コマンドの結果のサマリーが表示されたら、Ctrl-c で ping コマンドを終了することができます。パケットロスがない場合は、タイムリーかつ安定した接続であることが分かります。

```
--- 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
```

さらに、テストする宛先によっては、ping テストの結果は非常に明確な場合があります。たとえば、以下の図では、**VM1** において何らかの接続性の問題が発生しています。接続が可能な宛先を青の番号で示しています。また、成功結果または失敗結果から導かれた結論を記載しています。



OPENSTACK_450456_0617

1. **インターネット**: 一般的な最初のステップは、www.redhat.com などのインターネットロケーションに ping テストを送信します。

- **成功**: このテストは、送信元と送信先の間にあるさまざまなネットワークポイントすべてが期待通りに機能していることを示します。これには、仮想/物理インフラストラクチャーが含まれます。

- **失敗**: 遠隔にあるインターネットロケーションへの ping テストは、さまざまな部分で失敗する可能性があります。ネットワーク上の他のマシンがインターネットに正常に ping 送信できる場合には、インターネット接続は機能していることが分かり、使用しているマシンにもう少し近い箇所でトラブルシューティングを行う必要があります。

2. **物理ルーター**: これは、外部の宛先にトラフィックを転送するために、ネットワーク管理者が指定したルーターインターフェースです。

- **成功**: 物理ルーターに ping テストを行って、ローカルネットワークと基盤のスイッチが機能しているかどうかを検証することができます。このパケットは、ルーターを通過しないため、デフォルトのゲートウェイにルーティングの問題があるかどうかは分かりません。

- **失敗**: これは、VM1 とデフォルトゲートウェイの間で問題があることを示しています。ルーター/スイッチがダウンしているか、不正なデフォルトゲートウェイを使用している可能性があります。機能していることを確認済みの別のサーバーと、設定内容を比較してください。また、ローカルネットワーク上の別のサーバーに ping 送信を試行してみてください。

3. **Neutron ルーター**: これは、Red Hat OpenStack Platform が仮想マシントラフィックの転送に使用する仮想 SDN (ソフトウェア定義ネットワーク) ルーターです。

- **成功**: ファイアウォールが ICMP トラフィックを許可し、ネットワークノードがオンラインの状態です。

- **失敗**: インスタンスのセキュリティーグループで、ICMP トラフィックが許可されているかどうかを確認してください。また、ネットワークノードがオンラインで、必要なサービスすべてが実行中であることをチェックし、L3 エージェントのログ (**/var/log/neutron/l3-agent.log**) を確認してください。

4. **物理スイッチ**: 物理スイッチは、同じ物理ネットワーク上にあるノード間のトラフィックを管理する役割を果たします。

- **成功**: 仮想マシンが物理スイッチへ送信したトラフィックは、仮想ネットワークインフラストラクチャーを通過する必要があります。つまり、このセグメントが想定通りに機能していることが分かります。

- **失敗**: 必要な VLAN をトランク接続するように、物理スイッチポートは設定されていますか？

5. **VM2**: 同じコンピュートノード上にある、同じサブネットの仮想マシンに ping 送信を試行します。

- **成功**: VM1 上の NIC ドライバーと基本的な IP 設定が機能しています。

- **失敗**: VM1 のネットワーク設定を検証します。問題がない場合には、VM2 のファイアウォールが単に ping トラフィックをブロックしている可能性があります。また、仮想スイッチが正しく設定されていることをチェックし、Open vSwitch (または Linux Bridge) のログファイルを確認します。

6.2. VLAN ネットワークのトラブルシューティング

OpenStack Newtorking は、VLAN ネットワークをトランク接続して SDN スイッチに到達することができます。VLAN のタグ付けがされたプロバイダーネットワークに対するサポートがあると、仮想インスタンスを物理ネットワークにあるサーバーのサブネットと統合することができます。

VLAN プロバイダーネットワークへの接続のトラブルシューティングを行うには、ネットワークの作成時に指定したゲートウェイ IP に ping 送信をしてみてください。たとえば、以下のコマンドでネットワークを作成します。

```
# neutron net-create provider --provider:network_type=vlan --provider:physical_network=phy-eno1 --
provider:segmentation_id=120
# neutron subnet-create "provider" --allocation-pool start=192.168.120.1,end=192.168.120.253 --
disable-dhcp --gateway 192.168.120.254 192.168.120.0/24
```

この場合には、定義済みのゲートウェイの IP **192.168.120.254** に ping を送信してください。

これに失敗した場合には、関連付けられた VLAN (ネットワーク作成時に定義済み) へのネットワークフローがあることを確認してください。上記の例では、OpenStack Networking は VLAN 120 をプロバイダーネットワークにトランク接続するように設定されています。このオプションは **--provider:segmentation_id=120** のパラメーターを使用して設定します。

ブリッジインターフェース (今回の場合は **br-ex** という名前) の VLAN フローを確認します。

```
# 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 設定とログファイルの確認

- **OpenStack Networking (neutron) エージェント**: **openstack** コマンドを使用して、すべてのエージェントが稼働しており、正しい名前で登録されていることを確認します。

```
(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** を確認します。このログでは、作成プロセスで **ovs-ofctl** コマンドを使用して VLAN のトランク接続が設定されたことが確認できるはずです。

- **/etc/neutron/l3_agent.ini** ファイルで **external_network_bridge** を確認します。ここで値がハードコードされている場合は、L3 エージェント経由でプロバイダーネットワークを使用できず、必要なフローが作成されないので、この値は **external_network_bridge = ""** のようになるはずです。

- **/etc/neutron/plugin.ini** ファイルで **network_vlan_ranges** を確認します。プロバイダーネットワークを使用している場合には、数字の VLAN ID を指定する必要はありません。VLAN を分離したテナントネットワークを使用している場合にのみ、ここに ID を指定する必要があります。

- OVS エージェントの設定ファイルのブリッジマッピングを検証し、**phy-eno1** にマッピングされているブリッジが存在することと、**eno1** に適切に接続されていることを確認します。

6.3. テナントネットワーク内からのトラブルシューティング

OpenStack Networking では、テナントトラフィックはすべて、ネットワークの名前空間に含まれます。これにより、テナントは、テナント間の干渉なしにネットワークの設定が可能になります。たとえば、ネットワーク名前空間を使用することで異なるテナントが干渉することなしに 192.168.1.1/24 の同じサブネット範囲を指定することができます。

テナントネットワークのトラブルシューティングを開始するには、まず対象のネットワークがどのネットワーク名前空間に含まれているかを確認します。

1. **openstack** コマンドを使用して全テナントネットワークを一覧表示します。

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

この例では、**web-servers** ネットワークを検証します。**web-server** の行の **id** の値をメモしてください (この例では **9cb32fe0-d7fb-432c-b116-f483c6497b08** です)。この値は、ネットワーク名前空間に追加されており、次のステップで特定がしやすくなります。

2. ip コマンドを使用してネットワーク名前空間をすべて一覧表示します。

```
# 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
```

この結果では、**web-server** のネットワーク ID と一致する名前空間が存在します。この例では、**qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08** と表示されています。

3. この名前空間内でコマンドを実行して **web-servers** ネットワークの設定を検証します。これには、トラブルシューティングのコマンドの先頭に **ip netns exec (名前空間)** を追加します。例を以下に示します。

a) web-servers ネットワークのルーティングテーブルを表示する場合:

```
# 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
```

b) web-servers ネットワークのルーティングテーブルを表示する場合:

```
# 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. 名前空間内での高度な ICMP テストの実行

1. tcpdump コマンドを使用して ICMP トラフィックを取得します。

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

次のステップを実行するまで何も出力が表示されない可能性があります。

2. 別のコマンドラインウィンドウで、外部ネットワークへの ping テストを実行します。

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

3. **tcpdump** セッションを実行するターミナルで、ping テストの詳細結果を確認します。

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

注記

トラフィックの **tcpdump** 分析を実行する際には、インスタンスではなくルーターインターフェース方向の応答パケットが確認される場合があります。これは、qrouter によりリターンパケットで DNAT が実行されるため、想定通りの動作です。

第7章 物理ネットワークへのインスタンスの接続

本章では、プロバイダーネットワークを使用して外部ネットワークに直接インスタンスを接続する方法を説明します。

OpenStack Networking トポロジーの概要

OpenStack Networking (neutron) には、複数のノード種別に分散される 2 種類のサービスがあります。

- **Neutron サーバー**: このサービスは、エンドユーザーとサービスが OpenStack Networking と対話するための API を提供する OpenStack Networking API サーバーを実行します。このサーバーは、下層のデータベースと統合して、テナントネットワーク、ルーター、ロードバランサーの詳細などを保管します。
- **Neutron エージェント**: これらは、OpenStack Networking のネットワーク機能を実行するサービスです。
  - **neutron-dhcp-agent**: テナントプライベートネットワークの DHCP IP アドレスを管理します。
  - **neutron-l3-agent**: テナントプライベートネットワーク、外部ネットワークなどの間のレイヤー 3 ルーティングを実行します。
  - **neutron-lbaas-agent**: テナントにより作成された LBaaS ルーターをプロビジョニングします。
- **コンピュートノード**: このノードは、仮想マシン (別称: インスタンス) を実行するハイパーバイザーをホストします。コンピュートノードは、インスタンスに外部への接続を提供するために、ネットワークに有線で直接接続する必要があります。このノードは通常、**neutron-openvswitch-agent** などの L2 エージェントが実行される場所です。

サービスの配置:

OpenStack Networking サービスは、同じ物理サーバーまたは別の専用サーバー (役割によって名付けられる) で実行することができます。

- **コントローラーノード**: API サービスを実行するサーバー
- **ネットワークノード**: OpenStack Networking エージェントを実行するサーバー
- **コンピュートノード**: インスタンスをホストするハイパーバイザー

本章の以下の手順では、環境に上記の 3 種類のノード種別がデプロイされていることが前提です。お使いのデプロイメントで、同じ物理ノードがコントローラーノードとネットワークノードの両方の役割を果たしている場合には、そのサーバーで両ノードのセクションの手順を実行する必要があります。これは、3 つの全ノードにおいてコントローラーノードおよびネットワークノードサービスが HA で実行されている高可用性 (HA) 環境にも適用されます。そのため、コントローラーノードとネットワークノードに該当するセクションの手順を全 3 ノードで実行する必要があります。

7.1. フラットプロバイダーネットワークの使用

以下の手順では、外部ネットワークの直接インスタンスを接続可能なフラットプロバイダーネットワークを作成します。複数の物理ネットワーク (**physnet1**、**physnet2**) およびそれぞれ別の物理インターフェース (**eth0 -> physnet1** および **eth1 -> physnet2**) があり、各コンピュートノードとネットワークノードをこれらの外部ネットワークに接続する必要がある場合に実行します。

注記

単一の NIC 上の VLAN タグ付けされた複数のインターフェースを複数のプロバイダーネットワークに接続する場合には、「VLAN プロバイダーネットワークの使用」を参照してください。

コントローラーノードの設定

1. **/etc/neutron/plugin.ini** (シンボリックリンク: **/etc/neutron/plugins/ml2/ml2_conf.ini**) を編集して、既存の値リストに **flat** を追加して、**flat_networks** を * に設定します。

```
type_drivers = vxlan,flat
flat_networks =*
```

2. フラットな外部ネットワークを作成して、設定済みの **physical_network** に関連付けます。共有ネットワークとしてこのネットワークを作成することで、他のユーザーが直接インスタンスを接続できるようにします。

```
neutron net-create public01 --provider:network_type flat --provider:physical_network physnet1 --
router:external=True --shared
```

3. **neutron subnet-create** または OpenStack Dashboard を使用して、外部ネットワーク内にサブネットを作成します。以下に例を示します。

```
neutron subnet-create --name public_subnet --enable_dhcp=False --allocation_pool
start=192.168.100.20,end=192.168.100.100 --gateway=192.168.100.1 public01 192.168.100.0/24
```

4. **neutron-server** サービスを再起動して、この変更を適用します。

```
# systemctl restart neutron-server.service
```

ネットワークノードとコンピュートノードの両方の設定

以下のステップは、ネットワークノードとコンピュートノードで完了する必要があります。このステップを完了するとノードが外部ネットワークに接続され、インスタンスが直接外部ネットワークと通信できるようになります。

1. Open vSwitch ブリッジとポートを作成します。この手順では外部ネットワークのブリッジ (**br-ex**) を作成して、対応のポート (**eth1**) を追加します。

i. **/etc/sysconfig/network-scripts/ifcfg-eth1** を編集します。

```
DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

ii. **/etc/sysconfig/network-scripts/ifcfg-br-ex** を編集します。

```
DEVICE=br-ex
TYPE=OVSBridge
```

```
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

2. **network** サービスを再起動してこれらの変更を適用します。

```
# systemctl restart network.service
```

3. **/etc/neutron/plugins/ml2/openvswitch_agent.ini** で物理ネットワークを設定して、ブリッジを物理ネットワークにマッピングします。

注記

bridge_mappings の設定に関する詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

```
bridge_mappings = physnet1:br-ex
```

4. ネットワークノードとコンピュートノード上で **neutron-openvswitch-agent** サービスを再起動して、変更を適用します。

```
# systemctl restart neutron-openvswitch-agent
```

ネットワークノードの設定

1. **/etc/neutron/l3_agent.ini** で **external_network_bridge =** を空の値に設定します。これにより、外部プロバイダーネットワークが使用できるようになります。

```
# Name of bridge used for external network traffic. This should be set to
# empty value for the linux bridge
external_network_bridge =
```

2. **neutron-l3-agent** を再起動して変更を適用します。

```
# systemctl restart neutron-l3-agent.service
```

注記

複数のフラットプロバイダーネットワークが存在する場合には、それぞれに異なる物理インターフェースとブリッジを使用して外部ネットワークに接続すべきです。**ifcfg-*** スクリプトを適切に設定し、**bridge_mappings** にコンマ区切りリストでネットワーク別のマッピングを指定してください。**bridge_mappings** の設定に関する詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

外部ネットワークへのインスタンスの接続

このネットワークが作成されると、インスタンスを接続して、接続性をテストすることができます。

1. 新規インスタンスを作成します。

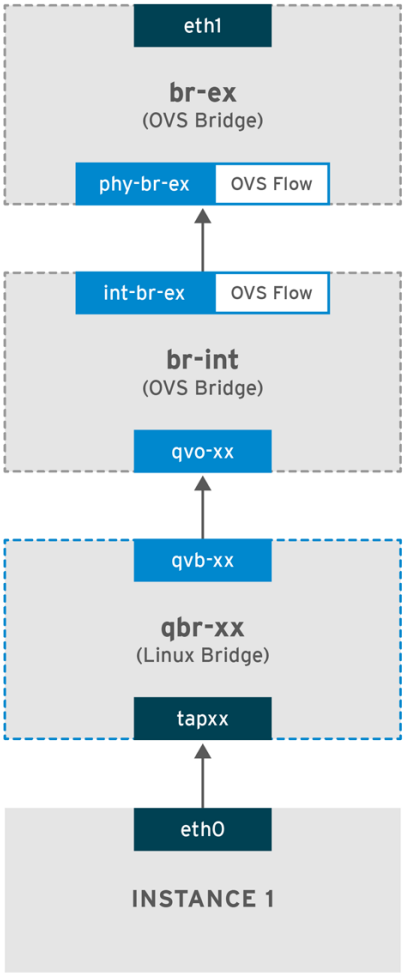
2. Dashboard の **ネットワーク** タブから、新たに作成した外部ネットワークに新規インスタンスを直接追加します。

パケットフローについて

フラットネットワークが設定され、本項ではインスタンスに対するトラフィックの流れについて詳しく説明します。

7.1.1. 送信トラフィックのフロー

インスタンスから送信され、直接外部ネットワークに到達するトラフィックのパケットフロー: **br-ex** を設定し、物理インターフェースを追加してインスタンスをコンピュートノードに作成すると、作成されるインターフェースとブリッジは以下の図のようになります (**iptables_hybrid** ファイアウォールドライバーを使用する場合)。



OPENSTACK_450456_0617

1. インスタンスの **eth0** インターフェースからのパケットは最初に linux ブリッジ **qbr-xx** に到達します。

2. ブリッジ **qbr-xx** は veth ペア **qvb-xx <-> qvo-xxx** を使用して、**br-int** に接続します。これは、セキュリティーグループによって定義されている受信/送信のファイアウォールルールの適用にブリッジが使用されるためです。

3. インターフェース **qvb-xx** は **qbr-xx** linux ブリッジに、**qvoxx** は **br-int** Open vSwitch (OVS) ブリッジに接続されます。

「qbr-xx」Linux ブリッジの設定例を以下に示します。

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

br-int 上の **qvo-xx** の設定:

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

注記

ポート **qvo-xx** は、フラットなプロバイダーネットワークに関連付けられた内部 VLAN タグでタグ付けされます。この例では VLAN タグは **5** です。パケットが **qvo-xx** に到達すると、VLAN タグがパケットのヘッダーに追加されます。

次にこのパケットは、パッチピア **int-br-ex <-> phy-br-ex** を使用して **br-ex** OVS ブリッジに移動します。

br-int でのパッチピアの設定例

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

br-ex でのパッチピアの設定例

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

このパケットが **br-ex** の **phy-br-ex** に到達すると、**br-ex** 内の OVS フローにより VLAN タグ (5) が取り除かれ、物理インターフェースに転送されます。

以下の出力例では、**phy-br-ex** のポート番号は **2** となっています。

```
 # ovs-ofctl show br-ex
OFPT_FEATURES_REPLY (xid=0x2): dpid:00003440b5c90dc6
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
```

```
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE

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

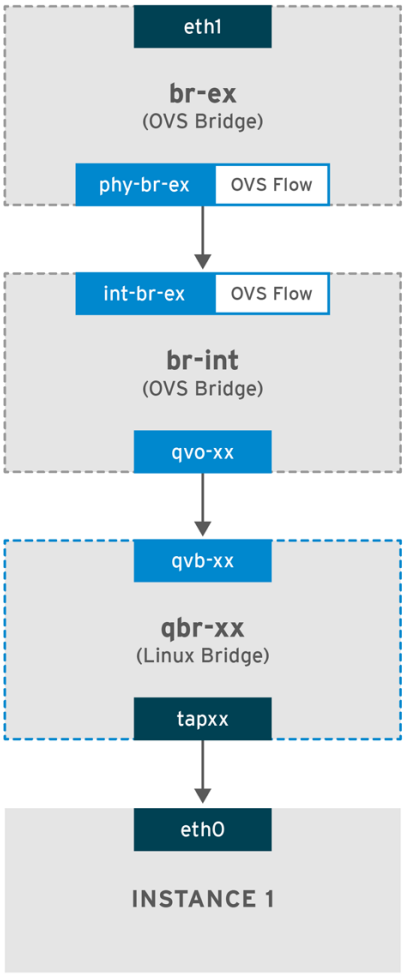
以下の出力例では、VLAN タグが **5** (**dl_vlan=5**) の **phy-br-ex** (**in_port=2**) に到達するパケットを示しています。さらに、VLAN タグは削除され、パケットが転送されます (**actions=strip_vlan,NORMAL**)。

```
# 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
```

このパケットは、次に物理インターフェースに転送されます。物理インターフェースが別の VLAN タグ付きインターフェースの場合、そのインターフェースはパケットにタグを追加します。

7.1.2. 受信トラフィックのフロー

以下の項では、外部ネットワークからインスタンスのインターフェースに到達するまでの受信トラフィックのフローを説明します。



OPENSTACK_450456_0617

1. 最初に受信トラフィックは物理ノードの **eth1** に到達します。

2. 次にパケットは **br-ex** ブリッジに渡されます。

3. パッチピア **phy-br-ex <--> int-br-ex** を使用して、このパケットは **br-int** に移動します。

以下の例では、**int-br-ex** がポート番号 **15** を使用します。**15(int-br-ex)** が含まれるエントリーに注目してください。

```
 # 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-int のトラフィックフローの確認

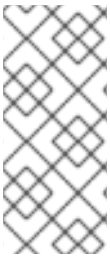
1. パケットが **int-br-ex** に到達すると、**br-int** ブリッジ内の OVS フロールールにより、内部 VLAN タグ **5** を追加するようにパケットが変更されます。**actions=mod_vlan_vid:5** のエントリーを参照してください。

```
 # ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5351.536s, table=0, n_packets=12118, n_bytes=510456, idle_age=0,
priority=1 actions=NORMAL
 cookie=0x0, duration=4537.553s, table=0, n_packets=3489, n_bytes=321696, idle_age=0,
priority=3,in_port=15,vlan_tci=0x0000 actions=mod_vlan_vid:5,NORMAL
 cookie=0x0, duration=5350.365s, table=0, n_packets=628, n_bytes=57892, idle_age=4538,
priority=2,in_port=15 actions=drop
 cookie=0x0, duration=5351.432s, table=23, n_packets=0, n_bytes=0, idle_age=5351, priority=0
actions=drop
```

2. 2 番目のルールは、VLAN タグ (vlan_tci=0x0000) のない int-br-ex (in_port=15) に到達するパケットを管理します。これにより、VLAN タグ 5 はパケット (**actions=mod_vlan_vid:5,NORMAL**) に追加され、**qvoxxx** に転送されます。

3. **qvoxxx** は、VLAN タグを削除した後に、パケットを受け入れて **qvbxx** に転送します。

4. 最終的にパケットはインスタンスに到達します。



注記

VLAN tag 5 は、フラットプロバイダーネットワークを使用するテスト用コンピュートノードで使用したサンプルの VLAN です。この値は **neutron-openvswitch-agent** により自動的に割り当てられました。お使いのフラットプロバイダーネットワークの値とは異なる可能性があり、2 種のコンピュートノード上にある同じネットワークにおいても異なる可能性があります。

7.1.3. トラブルシューティング

前項「**パケットフローについて**」で提供された出力では、問題が発生した場合にフラットプロバイダーネットワークをトラブルシューティングするためのデバッグ情報が十分に提供されません。以下の手順では、トラブルシューティングのプロセスについて説明します。

1. bridge_mappings を確認します。

使用する物理ネットワーク名 (例: **physnet1**) が **bridge_mapping** 設定の内容と一致していることを確認します。以下に例を示します。

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

 # neutron net-show provider-flat
...
| provider:physical_network | physnet1
...
```

2. ネットワークの設定を確認します。

ネットワークが **external** として作成され、**flat** の種別が使用されていることを確認します。

```
 # neutron net-show provider-flat
...
| provider:network_type     | flat                                 |
| router:external           | True                                 |
...
```

■

3. パッチピアを確認します。

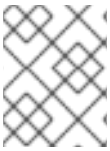
ovs-vsctl show を実行し、**int-br-ex <--> phy-br-ex**を使用して **br-int** および **br-ex** が接続されていることを確認します。

```
# 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 でのパッチピアの設定例

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

この接続は、**/etc/neutron/plugins/ml2/openvswitch_agent.ini** で **bridge_mapping** が正しく設定されている場合にのみ、**neutron-openvswitch-agent** サービスが再起動された時点で作成されます。サービスを再起動した後でもこれが作成されない場合には、**bridge_mapping** の設定を再確認してください。



注記

bridge_mappings の設定に関する詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

4. ネットワークフローを確認します。

ovs-ofctl dump-flows br-ex と **ovs-ofctl dump-flows br-int** を実行して、フローにより送信パケットの内部 VLAN ID が削除されたかどうかを確認します。まず、このフローは、特定のコンピュートノード上のこのネットワークにインスタンスを作成すると追加されます。

- インスタンスの起動後にこのフローが作成されなかった場合には、ネットワークが **flat** として作成されており、**external** であることと、**physical_network** の名前が正しいことを確認します。また、**bridge_mapping** の設定を確認します。
- 最後に **ifcfg-br-ex** と **ifcfg-ethx** の設定をチェックします。**ethX** が **br-ex** 内のポートとして追加されており、**ip a** の出力でどちらのインターフェースにも **UP** フラグが指定されていることを確認します。

たとえば、以下の出力では **eth1** は **br-ex** のポートであることが分かります。

```
  Bridge br-ex
      Port phy-br-ex
          Interface phy-br-ex
              type: patch
```

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

以下の例では **eth1** は OVS ポートとして設定されており、カーネルはこのインターフェースからのパケットをすべて転送して OVS ブリッジ **br-ex** に送信することを認識していることが分かります。これは、**master ovs-system** のエントリーで確認することができます。

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

7.2. VLAN プロバイダーネットワークの使用

以下の手順では、外部ネットワークに直接インスタンスを接続可能な VLAN プロバイダーネットワークを作成します。複数のプロバイダーネットワークに (単一の NIC 上で) VLAN タグが付けられたインターフェースを複数接続するには、この手順を実行します。以下の例では、VLAN 範囲が **171-172** の **physnet1** と呼ばれる物理ネットワークを使用します。ネットワークノードとコンピュートノードは、**eth1** という名前の物理インターフェースを使用して物理ネットワークに接続します。これらのインターフェースの接続先のスイッチポートは、必要な VLAN 範囲をトランク接続するように設定する必要があります。
以下の手順では、サンプルの VLAN ID と上記で指定した名前を使用して VLAN プロバイダーネットワークを設定します。

コントローラーノードの設定

1. **/etc/neutron/plugin.ini** (シンボリックリンク: **/etc/neutron/plugins/ml2/ml2_conf.ini**) を編集して**vlan** メカニズムドライバーを有効化して、**vlan** を既存の値リストに追加します。以下に例を示します。

```
[ml2]
type_drivers = vxlan,flat,vlan
```

2. **network_vlan_ranges** の設定を行い、使用する物理ネットワークおよび VLAN 範囲を反映します。以下に例を示します。

```
[ml2_type_vlan]
network_vlan_ranges=physnet1:171:172
```

3. **neutron-server** サービスを再起動して変更を適用します。

```
systemctl restart neutron-server
```

4. 外部ネットワークを **vlan** 種別として作成して、設定済みの **physical_network** に関連付けます。**--shared** ネットワークとして作成して、他のユーザーが直接インスタンスに接続できるようにします。以下の例では、VLAN 171 と VLAN 172 の 2 つのネットワークを作成します。

```
neutron net-create provider-vlan171 \
  --provider:network_type vlan \
  --router:external true \
  --provider:physical_network physnet1 \
  --provider:segmentation_id 171 --shared

neutron net-create provider-vlan172 \
```

```
  --provider:network_type vlan \
  --router:external true \
  --provider:physical_network physnet1 \
  --provider:segmentation_id 172 --shared
```

5. 複数のサブネットマスクを作成して、外部ネットワークを使用するように設定します。これは、**neutron subnet-create** または Dashboard のいずれかを使用して設定できます。ネットワーク管理者から取得した外部サブネットの詳細が正しく各 VLAN に関連付けられていることを確認します。以下の例では、VLAN 171 はサブネット **10.65.217.0/24**、VLAN 172 は **10.65.218.0/24** を使用します。

```
neutron subnet-create \
  --name subnet-provider-171 provider-171 10.65.217.0/24 \
  --enable-dhcp \
  --gateway 10.65.217.254 \

neutron subnet-create \
  --name subnet-provider-172 provider-172 10.65.218.0/24 \
  --enable-dhcp \
  --gateway 10.65.218.254 \
```

ネットワークノードとコンピュートノードの設定

以下の手順は、ネットワークノードとコンピュートノードで実行する必要があります。この手順を実行すると、外部ネットワークにノードを接続して、インスタンスが外部ネットワークと直接通信できるようになります。

1. 外部ネットワークブリッジ (**br-ex**) を作成して、ポート (**eth1**) に関連付けます。

- 以下の例では、**eth1** が **br-ex** を使用するように設定します。

```
/etc/sysconfig/network-scripts/ifcfg-eth1

DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

- 以下の例では、**br-ex** ブリッジを設定します。

```
/etc/sysconfig/network-scripts/ifcfg-br-ex:

DEVICE=br-ex
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

2. ノードを再起動するか、**network** サービスを再起動して、ネットワークの変更を有効にします。以下に例を示します。

```
# systemctl restart network
```

3. **/etc/neutron/plugins/ml2/openvswitch_agent.ini** で物理ネットワークを設定して、物理ネットワークに応じてブリッジをマッピングします。

```
bridge_mappings = physnet1:br-ex
```

注記

bridge_mappings の設定に関する詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

4. ネットワークノードとコンピュートノードで **neutron-openvswitch-agent** サービスを再起動して、変更を有効にします。

```
systemctl restart neutron-openvswitch-agent
```

ネットワークノードの設定

1. **/etc/neutron/l3_agent.ini** で **external_network_bridge =** を空の値に設定します。これは、ブリッジベースの外部ネットワークではなく、プロバイダーの外部ネットワークを使用するために必要です。ブリッジベースの外部ネットワークの場合は **external_network_bridge = br-ex** を指定します。

```
# Name of bridge used for external network traffic. This should be set to
# empty value for the linux bridge
external_network_bridge =
```

2. **neutron-l3-agent** を再起動して、変更を有効にします。

```
systemctl restart neutron-l3-agent
```

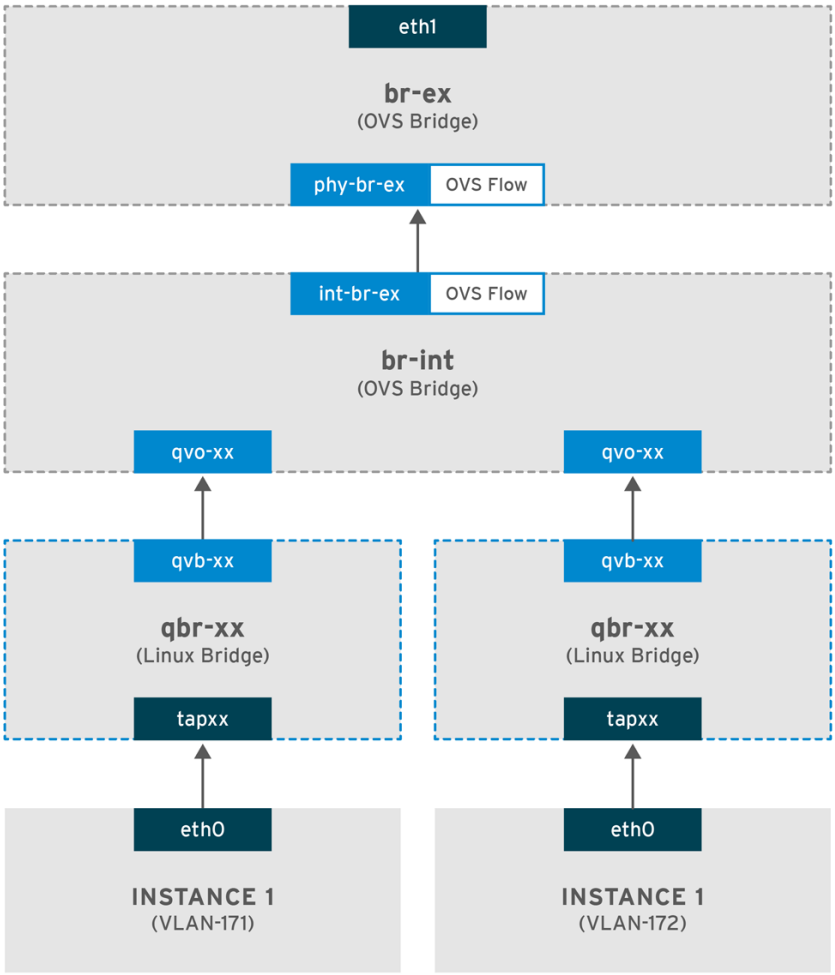
3. 新規インスタンスを作成して、Dashboard の **ネットワーク** タブを使用して新規作成した外部ネットワークに直接、新しいインスタンスを追加します。

パケットフローについて

VLAN プロバイダーネットワークが設定され、本項ではインスタンスに対するトラフィックの流れについて詳しく説明します。

7.2.1. 送信トラフィックのフロー

以下の項では、インスタンスから直接 VLAN プロバイダーの外部ネットワークに到達するトラフィックのパケットフローについて説明します。この例では、2 つの VLAN ネットワーク (171 および 172) にアタッチされた 2 つのインスタンスを使用します。**br-ex** を設定して物理インターフェースを追加し、コンピュートノードにインスタンスを作成すると、作成されたインターフェースとブリッジは以下の図のようになります。



OPENSTACK_450456_0617

1. 上記の図のように、インスタンスの **eth0** を出たパケットは、まずインスタンスに接続された linux ブリッジ **qbr-xx** に到達します。

2. **qbr-xx** は **qvbxx <→ qvoxxx** veth ペアを使用して **br-int** に接続します。

3. **qvbxx** は linux ブリッジ **qbr-xx** に、**qvoxx** は Open vSwitch ブリッジ **br-int** に接続します。

Linux ブリッジ上の qbr-xx の設定

インスタンスが 2 つあるため、linux ブリッジが 2 つになります。

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

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

br-int 上の qvoxx の設定

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

            Interface "qvo86257b61-5d"
```

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

- **qvoxx** には、VLAN プロバイダーネットワークが関連付けられた内部 VLAN のタグが付けられます。以下の例では、内部 VLAN タグ 2 には VLAN プロバイダーネットワーク **provider-171**、VLAN タグ 3 には VLAN プロバイダーネットワーク **provider-172** が関連付けられます。パケットが **qvoxx** に到達すると、パケットのヘッダーにこの VLAN タグが追加されます。

- パケットは次に、パッチピア **int-br-ex** <→ **phy-br-ex** を使用して **br-ex** OVS ブリッジに移動します。**br-int** 上のパッチピアの例を以下に示します。

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

br-ex 上のパッチピアの設定例を以下に示します。

```
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
```

- このパケットが **br-ex** 上の **phy-br-ex** に到達すると、**br-ex** 内の OVS フローが内部 VLAN タグを VLAN プロバイダーネットワークに関連付けられた実際の VLAN タグに置き換えます。

以下のコマンドの出力では、**phy-br-ex** のポート番号は **4** となっています。

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

以下のコマンドでは、VLAN タグ 2 (**dl_vlan=2**) が付いた phy-br-ex (**in_port=4**) に到達するパケットを表示します。Open vSwitch により、VLAN タグは 171 に置き換えられ (**actions=mod_vlan_vid:171,NORMAL**)、パケットを次の宛先に転送します。また、このコマンドで、VLAN タグ 3 (**actions=mod_vlan_vid:172,NORMAL**) が付いた phy-br-ex (**in_port=4**) に到達するパケットが表示され、Open vSwitch により VLAN タグは 172 に置き換えられ (**actions=mod_vlan_vid:172,NORMAL**)、次の宛先にパケットを転送します。

```
# 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
```

- このパケットは、次に物理インターフェース **eth1** に転送されます。

7.2.2. 受信トラフィックのフロー

- 外部ネットワークから受信するインスタンスのパケットは、**eth1** に到達してから **br-ex** に届きます。
- このパケットは、br-ex からパッチピア **phy-br-ex <-> int-br-ex** を経由して br-int に移動します。

以下のコマンドを実行すると、ポート番号 18 を使用する **int-br-ex** が表示されます。

```
# 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
```

- パケットが **int-br-ex** に到着すると、**br-int** 内の OVS フローにより、**provider-171** の場合は内部 VLAN タグ 2 が、**provider-172** の場合は VLAN タグ 3 がパケットに追加されます。

```
# 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
```

2 番目のルールでは、VLAN タグ 172 (**dl_vlan=172**) が付いた int-br-ex (**in_port=18**) に到達するパケットは VLAN タグが 3 (**actions=mod_vlan_vid:3,NORMAL**) に置き換えられ、次に進むように記載されています。次に 3 番目のルールは、VLAN タグ 171 (**dl_vlan=171**) が付いた int-br-ex (**in_port=18**) に到達するパケットは VLAN タグが 2 (**actions=mod_vlan_vid:2,NORMAL**) に置き換えられ、次に進むように記載されています。

- in-br-ex から内部 VLAN タグがパケットに追加されると、qvoxxx はそのパケットを受け入れ、VLAN タグを削除してから qvbxx に転送します。パケットは、その後にインスタンスに到達します。

VLAN タグ 2 および 3 は、テスト用のコンピュートノードで、VLAN プロバイダーネットワーク (provider-171 および provider-172) に使用した一例である点に注意してください。お使いの VLAN プロバイダーネットワークに必要な設定は異なる場合があります。また、同じネットワークを使用する 2 つの異なるコンピュートノードで VLAN タグが異なる場合もあります。

7.2.3. トラブルシューティング

VLAN プロバイダーネットワークの接続についてトラブルシューティングを行う場合は、本項に記載のパケットフローを参照してください。さらに、以下の設定オプションを確認してください。

1. 一貫して物理ネットワーク名が使用されていることを確認してください。以下の例では、ネットワークの作成時と、**bridge_mapping** の設定において、一貫して **physnet1** が使用されています。

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

# neutron net-show provider-vlan171
...
| provider:physical_network | physnet1
...
```

2. ネットワークが **external** として **vlan** の種別で作成され、正しい **segmentation_id** の値が使用されていることを確認します。

```
# neutron net-show provider-vlan171
...
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 171                                  |
...
```

3. **ovs-vsctl show** を実行して、**br-int** および **br-ex** がパッチピア **int-br-ex <→ phy-br-ex** を使用して接続されていることを確認します。
この接続は、**/etc/neutron/plugins/ml2/openvswitch_agent.ini** で **bridge_mapping** が正しく設定されていることを前提として、**neutron-openvswitch-agent** の再起動の後に作成されます。
サービスを再起動してもこの接続が作成されない場合には **bridge_mapping** の設定を再確認してください。

4. 送信パケットのフローを確認するには、**ovs-ofctl dump-flows br-ex** および **ovs-ofctl dump-flows br-int** を実行して、このフローにより VLAN ID が外部 VLAN id (**segmentation_id**) にマッピングされていることを確認します。受信パケットには、外部 VLAN ID が内部 VLAN ID にマッピングされます。
このフローは、このネットワークに初めてインスタンスを作成した場合に neutron OVS エージェントにより追加されます。インスタンスの起動後にネットワークが作成されていない場合は、ネットワークが **external** で **vlan** として作成されていて、**physical_network** の名前が正しいことを確認します。また、**bridge_mapping** の設定を再確認してください。

5. 最後に、**ifcfg-br-ex** と **ifcfg-ethx** の設定を確認します。**ethX** が **br-ex** の中にポートとして追加されており、いずれも **ip a** のコマンド出力において **UP** フラグがついていることを確認します。
たとえば、以下の出力例では、**eth1** は **br-ex** 内のポートとなっています。

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

以下のコマンドでは、**eth1** がポートとして追加され、カーネルがこのインターフェースから OVS ブリッジ **br-ex** にすべてのパケットを移動することを認識していることが分かります。これは、エントリー **master ovs-system** で確認できます。

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

7.3. コンピュートのメタデータアクセスの有効化

この方法で接続されたインスタンスは、プロバイダーの外部ネットワークに直接アタッチされ、OpenStack Networking (neutron) ルーターではなく、外部ルーターがデフォルトゲートウェイとして設定されます。これは、**neutron** ルーターはインスタンスから **nova-metadata** サーバーへのメタデータ要求をプロキシー化するために使用することができないため、**cloud-init** の実行中にエラーが発生する可能性があることを意味しますが、この問題は dhcp エージェントがメタデータ要求をプロキシー化するように設定することによって解決することができます。この機能は、**/etc/neutron/dhcp_agent.ini** で有効にすることができます。以下に例を示します。

```
enable_isolated_metadata = True
```

7.4. FLOATING IP アドレス

同時にプライベートネットワークに追加された場合でも、同じネットワークを利用して、Floating IP アドレスをインスタンスに割り当てることができる点に注意してください。このネットワークから Floating IP として割り当てられたアドレスは、ネットワークノードの **qrouter-xxx** の名前空間にバインドされ、関連付けられたプライベート IP アドレスに **DNAT-SNAT** を実行します。反対に、直接外部ネットワークにアクセスできるように割り当てられた IP アドレスはインスタンス内に直接バインドされ、インスタンスが外部ネットワークと直接通信できるようになります。

第8章 OPENSTACK NETWORKING の物理スイッチの設定

本章では、OpenStack Networking に必要な一般的な物理スイッチの設定手順を説明します。以下のスイッチに関するベンダー固有の設定を記載しています。

- Cisco Catalyst
- Cisco Nexus
- Cumulus Linux
- Extreme Networks EXOS
- Juniper EX Series

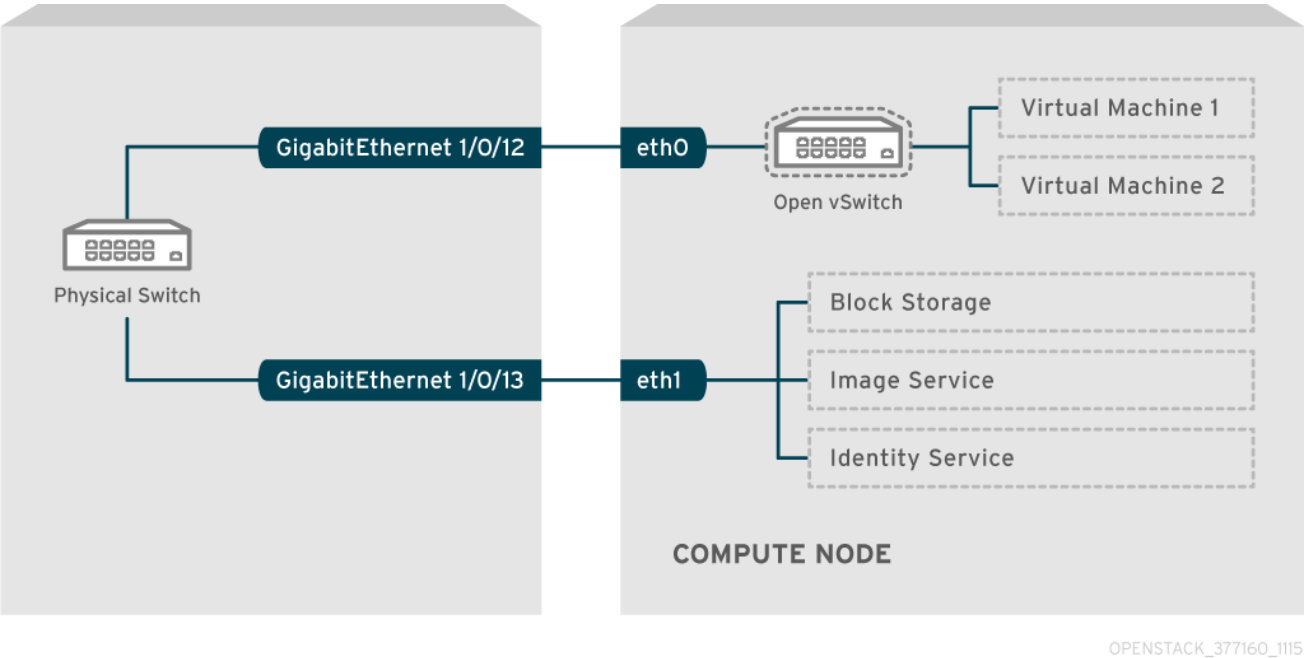
8.1. 物理ネットワーク環境のプランニング

OpenStack ノード内の物理ネットワークアダプターは、異なる種別のネットワークトラフィックを伝送することが想定される場合があります。これには、インスタンストラフィック、ストレージデータ、認証要求が含まれます。NIC が伝送するトラフィックの種別によって、物理スイッチ上でのポートの設定方法が異なります。

最初のステップでは、コンピュートノード上の物理 NIC で伝送するトラフィックの種別を決定する必要があります。次に、NIC が物理スイッチポートに接続されると、そのスイッチポートはトラックトラフィックまたは一般のトラフィックを許可するように特別に設定する必要があります。

たとえば、以下の図は、eth0 と eth1 の 2 つの NIC を搭載したコンピュートノードを示しています。各 NIC は、物理スイッチ上のギガビットイーサネットポートに接続され、eth0 がインスタンストラフィックを伝送し、eth1 が OpenStack サービスの接続性を提供します。

ネットワークレイアウト例



注記

この図には、耐障害性に必要な追加の冗長 NIC は含まれていません。

8.2. CISCO CATALYST スイッチの設定

8.2.1. トランクポートの設定

OpenStack Networking により、インスタンスは物理ネットワーク上にすでに存在する VLAN に接続することができます。**トランク** という用語は、単一のポートで複数の VLAN が通過を許可することを意味します。トランクにより、VLAN は、仮想スイッチを含む複数のスイッチを橋渡しすることができます。たとえば、物理ネットワークで **VLAN110** としてタグされたトラフィックが、コンピュートノードに到達すると、8021q モジュールによってタグ付けされたトラフィックが vSwitch 上の適切な VLAN にダイレクトされます。

8.2.1.1. Cisco Catalyst スイッチのトランクポートの設定

Cisco IOS を実行する Cisco Catalyst スイッチを使用する場合には、以下の設定構文を使用して、VLAN 110 と 111 のトラフィックがインスタンスに到達できるよう設定することが可能です。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **GigabitEthernet1/0/12** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

```
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
```

これらの設定についての説明を以下に記載します。

フィールド	説明
interface GigabitEthernet1/0/12	ノードの NIC が結線されるスイッチポートです。これは単なる一例なので、最初に正しいポートをここで設定するように確認することが重要です。show interface コマンドでポートの一覧を表示することができます。
description Trunk to Compute Node	show interface コマンドを使用して全インターフェースを一覧表示する際に表示される説明。どのシステムがこのポートに結線されていて、その接続の目的とする機能がわかるのに十分な説明である必要があります。
spanning-tree portfast trunk	環境で STP を使用することを前提として、Port Fast に対してこのポートがトランクトラフィックに使用されることを伝えます。

フィールド	説明
switchport trunk encapsulation dot1q	802.1q のトランク標準 (ISL ではなく) を有効化します。これは、スイッチが何をサポートするかによって異なります。
switchport mode trunk	このポートは、アクセスポートではなく、トランクポートとして設定します。これで VLAN トラフィックが仮想スイッチに到達できるようになります。
switchport trunk native vlan 2	ネイティブ VLAN を設定して、タグの付いていない (VLAN 以外の) トラフィックの送信先をスイッチに指示します。
switchport trunk allowed vlan 2,110,111	トランクを通過できる VLAN を定義します。

8.2.2. アクセスポートの設定

コンピュートノード上の全 NIC がインスタンスのトラフィックを伝送するわけではないので、複数の VLAN が通過できるように設定する必要はありません。このようなポートに設定する必要があるのは VLAN 1 つのみで、管理トラフィックやブロックストレージデータの転送などの他の運用上の要件を満たす必要がある可能性があります。これらのポートはアクセスポートとして一般的に知られており、必要な設定は通常、トランクポートよりも簡単です。

8.2.2.1. Cisco Catalyst スイッチのアクセスポートの設定

「ネットワークレイアウト例」の図に示した例を使用して、GigabitEthernet1/0/13 (Cisco Catalyst スイッチ上) を eth1 のアクセスポイントとして設定します。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **GigabitEthernet1/0/12** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

これらの設定についての説明を以下に記載します。

フィールド	説明

フィールド	説明
interface GigabitEthernet1/0/13	ノードの NIC が結線されるスイッチポートです。インターフェースの値は単なる一例なので、最初に正しいポートをここで設定するように確認することが重要です。ポートの一覧を表示するには、show interface コマンドを使用できます。
description Access port for Compute Node	show interface コマンドを使用して全インターフェースを一覧表示する際に表示される説明。どのシステムがこのポートに結線されていて、その接続の目的とする機能がわかるのに十分な説明である必要があります。
switchport mode access	このポートは、トランクポートとしてではなく、アクセスポートとして設定します。
switchport access vlan 200	VLAN 200 上でトラフィックを許可するポートを設定します。コンピュートノードには、この VLAN からの IP アドレスを設定すべきです。
spanning-tree portfast	STP を使用する場合には、この設定は、STP がこのポートをトランクとして初期化を試みないように指示して、初回接続時 (例: サーバーの再起動時など) のポートハンドシェイクをより迅速に行うことができます。

8.2.3. LACP ポートアグリゲーションの設定

LACP により、複数の物理 NIC をバンドルして単一の論理チャネルを形成することができます。LACP は、802.3ad (または、Linux ではボンディングモード 4) としても知られており、負荷分散と耐障害性のための動的なボンディングを作成します。LACP は、物理 NIC と物理スイッチポートの両方の物理エンドで設定する必要があります。

8.2.3.1. 物理 NIC 上での LACP の設定

1. **/home/stack/network-environment.yaml** ファイルを編集します。

```
- 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. Open vSwitch ブリッジが **LACP** を使用するように設定します。

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

ネットワークボンディングの設定方法についての説明は、『オーバークラウドの高度なカスタマイズ』を参照してください。

8.2.3.2. Cisco Catalyst スイッチ上での LACP の設定

以下の例では、コンピュートノードに VLAN 100 を使用する NIC が 2 つあります。

1. コンピュートノードの 2 つの NIC をスイッチ (例: ポート 12 と 13) に物理的に接続します。

2. LACP ポートチャネルを作成します。

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

3. スイッチポート 12 (Gi1/0/12) および 13 (Gi1/0/13) を設定します。

```
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. 新しいポートチャネルを確認します。出力には、新規ポートチャネル **Po1** と、メンバーポートの **Gi1/0/12** および **Gi1/0/13** が表示されます。

```
sw01# show etherchannel summary
<snip>

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

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

注記

「**copy running-config startup-config**」のコマンドを実行して、running-config を startup-config にコピーして、変更を適用することを忘れないようにしてください。

8.2.4. MTU の設定

特定の種別のネットワークトラフィックには、MTU サイズの調整が必要な場合があります。たとえば、特定の NFS または iSCSI のトラフィックには、ジャンボフレーム (9000 バイト) が推奨される場合があります。

注記

MTU の設定は、エンドツーエンド (トラフィックが通過すると想定されている全ホップ) で変更する必要があります。これには、仮想スイッチが含まれます。OpenStack 環境における MTU の変更についての説明は、「9章*MTU の設定*」を参照してください。

8.2.4.1. Cisco Catalyst スイッチ上での MTU の設定

以下の例では、Cisco Catalyst 3750 スイッチでジャンボフレームを有効にします。

1. 現在の MTU 設定を確認します。

```
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. 3750 のスイッチでは、MTU 設定はインターフェースごとではなく、スイッチ全体で変更されます。以下のコマンドは、スイッチが 9000 バイトのジャンボフレームを使用するように設定します。お使いのスイッチがサポートしている場合には、インターフェースごとに MTU を設定した方がよいかもしれません。

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

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

注記

「**copy running-config startup-config**」のコマンドを実行して、running-config を startup-config にコピーして、変更を保存することを忘れないようにしてください。

3. 可能な場合には、スイッチを再読み込みして変更を適用してください。この操作を実行すると、そのスイッチに依存しているデバイスでネットワークが停止することになります。

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

4. スイッチが再読み込みの操作を完了したら、新しいジャンボ MTU のサイズを確認します。スイッチ

のモデルによって実際の出力は異なる場合があります。たとえば、**System MTU** がギガビット非対応のインターフェースに適用され、**Jumbo MTU** は全ギガビット対応インターフェースを記述する可能性があります。

```
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
```

8.2.5. LLDP ディスカバリーの設定

ironic-python-agent サービスは、接続されたスイッチから LLDP パケットをリッスンします。収集される情報には、スイッチ名、ポートの詳細、利用可能な VLAN を含めることができます。Cisco Discovery Protocol (CDP) と同様に、LLDP は、director の **イントロスペクション** プロセス中の物理ハードウェアの検出を補助します。

8.2.5.1. Cisco Catalyst スイッチ上での LLDP の設定

1. **lldp run**を使用して、Cisco Catalyst スイッチで LLDP をグローバルに有効にします。

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

sw01(config)# lldp run
```

2. 隣接する LLDP 対応デバイスを表示します。

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

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

Total entries displayed: 1
```

注記

「**copy running-config startup-config**」のコマンドを実行して、running-config を startup-config にコピーして、変更を保存することを忘れないようにしてください。

8.3. CISCO NEXUS スイッチの設定

8.3.1. トランクポートの設定

OpenStack Networking により、インスタンスは物理ネットワーク上にすでに存在する VLAN に接続することができます。**トランク** という用語は、単一のポートで複数の VLAN が通過を許可することを意味します。トランクにより、VLAN は、仮想スイッチを含む複数のスイッチを橋渡しすることができま

す。たとえば、物理ネットワークで **VLAN110** としてタグされたトラフィックが、コンピュートノードに到達すると、8021q モジュールによってタグ付けされたトラフィックが vSwitch 上の適切な VLAN にダイレクトされます。

8.3.1.1. Cisco Nexus スイッチのトランクポートの設定

Cisco Nexus を使用する場合には、以下の設定構文を使用して、VLAN 110 と 111 のトラフィックがインスタンスに到達できるように設定することが可能です。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **Ethernet1/12** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

8.3.2. アクセスポートの設定

コンピュートノード上の全 NIC がインスタンスのトラフィックを伝送するわけではないので、複数の VLAN が通過できるように設定する必要はありません。このようなポートに設定する必要があるのは VLAN 1 つのみで、管理トラフィックやブロックストレージデータの転送などの他の運用上の要件を満たす必要がある可能性があります。これらのポートはアクセスポートとして一般的に知られており、必要な設定は通常、トランクポートよりも簡単です。

8.3.2.1. Cisco Nexus スイッチのアクセスポートの設定

「ネットワークレイアウト例」の図に示した例を使用して、Ethernet1/13 (Cisco Nexus スイッチ上) を eth1 のアクセスポイントとして設定します。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **Ethernet1/13** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

8.3.3. LACP ポートアグリゲーションの設定

LACP により、複数の物理 NIC をバンドルして単一の論理チャネルを形成することができます。LACP は、802.3ad (または、Linux ではボンディングモード 4) としても知られており、負荷分散と耐障害性のための動的なボンディングを作成します。LACP は、物理 NIC と物理スイッチポートの両方の物理エンドで設定する必要があります。

8.3.3.1. 物理 NIC 上での LACP の設定

1. **/home/stack/network-environment.yaml** ファイルを編集します。

```
- 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. Open vSwitch ブリッジが **LACP** を使用するように設定します。

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

ネットワークボンディングの設定方法についての説明は、『オーバークラウドの高度なカスタマイズ』を参照してください。

8.3.3.2. Cisco Nexus スイッチ上での LACP の設定

以下の例では、コンピュートノードに VLAN 100 を使用する NIC が 2 つあります。

1. コンピュートノードの 2 つの NIC をスイッチ (例: ポート 12 と 13) に物理的に接続します。

2. LACP が有効化されているかどうかを確認します。

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

3. ポート 1/12 と 1/13 をアクセスポートおよびチャネルグループのメンバーとして設定します。デプロイメントによっては、アクセスインターフェースの代わりにトランクインターフェースを使用するようにデプロイすることができます。たとえば、Cisco UCI の場合には、NIC は仮想インターフェースなので、全アクセスポートを設定した方がよいでしょう。また、インターフェースで VLAN タグ付けが設定されている可能性が高くなります。

```
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
```

8.3.4. MTU の設定

特定の種別のネットワークトラフィックには、MTU サイズの調整が必要な場合があります。たとえば、特定の NFS または iSCSI のトラフィックには、ジャンボフレーム (9000 バイト) が推奨される場合があります。

注記

MTU の設定は、エンドツーエンド (トラフィックが通過すると想定されている全ホップ) で変更する必要があります。これには、仮想スイッチが含まれます。OpenStack 環境における MTU の変更についての説明は、「9章*MTU の設定*」を参照してください。

8.3.4.1. Cisco Nexus 7000 スイッチ上での MTU の設定

MTU の設定は、7000 シリーズのスイッチ上の単一のインターフェースに適用することができます。以下のコマンドは、インターフェース 1/12 が 9000 バイトのジャンボフレームを使用するように設定します。

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

8.3.5. LLDP ディスカバリーの設定

ironic-python-agent サービスは、接続されたスイッチから LLDP パケットをリッスンします。収集される情報には、スイッチ名、ポートの詳細、利用可能な VLAN を含めることができます。Cisco Discovery Protocol (CDP) と同様に、LLDP は、director の **イントロスペクション** プロセス中の物理ハードウェアの検出を補助します。

8.3.5.1. Cisco Nexus 7000 スイッチ上での LLDP の設定

LLDP は、Cisco Nexus 7000 シリーズスイッチ上の個別のインターフェースに対して有効化することができます。

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

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

注記

「**copy running-config startup-config**」のコマンドを実行して、running-config を startup-config にコピーして、変更を保存することを忘れないようにしてください。

8.4. CUMULUS LINUX スイッチの設定

8.4.1. トランクポートの設定

OpenStack Networking により、インスタンスは物理ネットワーク上にすでに存在する VLAN に接続することができます。**トランク** という用語は、単一のポートで複数の VLAN が通過を許可することを意味します。トランクにより、VLAN は、仮想スイッチを含む複数のスイッチを橋渡しすることができます。たとえば、物理ネットワークで **VLAN110** としてタグされたトラフィックが、コンピュートノードに到達すると、8021q モジュールによってタグ付けされたトラフィックが vSwitch 上の適切な VLAN にダイレクトされます。

8.4.1.1. Cumulus Linux スイッチのトランクポートの設定

Cumulus Linux スイッチを使用する場合には、以下のような設定構文を使用して、VLAN 100 と 200 のトラフィックがインスタンスに到達できるように設定することが可能です。この設定は、物理ノードでトランシーバーが物理スイッチ上のスイッチポート **swp1** および **swp2** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

8.4.2. アクセスポートの設定

コンピュートノード上の全 NIC がインスタンスのトラフィックを伝送するわけではないので、複数の VLAN が通過できるように設定する必要はありません。このようなポートに設定する必要があるのは VLAN 1 つのみで、管理トラフィックやブロックストレージデータの転送などの他の運用上の要件を満たす必要がある可能性があります。これらのポートはアクセスポートとして一般的に知られており、必要な設定は通常、トランクポートよりも簡単です。

8.4.2.1. Cumulus Linux スイッチのアクセスポートの設定

「ネットワークレイアウト例」の図に示した例を使用して、**swp1** (Cumulus Linux スイッチ上) をアクセスポートとして設定します。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェースに接続されていることを前提としています。Cumulus Linux スイッチは、管理インターフェースに **eth** を、アクセス/トランクポートに **swp** を使用します。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

```
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
```

8.4.3. LACP ポートアグリゲーションの設定

LACP により、複数の物理 NIC をバンドルして単一の論理チャネルを形成することができます。LACP は、802.3ad (または、Linux ではボンディングモード 4) としても知られており、負荷分散と耐障害性のための動的なボンディングを作成します。LACP は、物理 NIC と物理スイッチポートの両方の物理エンドで設定する必要があります。

8.4.3.1. 物理 NIC 上での LACP の設定

Cumulus Linux では物理 NIC を設定する必要はありません。

8.4.3.2. Cumulus Linux スイッチでの LACP の設定

ボンディングの設定には、**/etc/network/interfaces** を編集して **bond0** の内容を追加します。

```
auto bond0
iface bond0
   address 10.0.0.1/30
   bond-slaves swp1 swp2 swp3 swp4
```

注記

更新した設定を **sudo ifreload -a** を実行して再読み込みし、変更を適用することを忘れないでください。

8.4.4. MTU の設定

特定の種別のネットワークトラフィックには、MTU サイズの調整が必要な場合があります。たとえば、特定の NFS または iSCSI のトラフィックには、ジャンボフレーム (9000 バイト) が推奨される場合があります。

注記

MTU の設定は、エンドツーエンド (トラフィックが通過すると想定されている全ホップ) で変更する必要があります。これには、仮想スイッチが含まれます。OpenStack 環境における MTU の変更についての説明は、「9章*MTU の設定*」を参照してください。

8.4.4.1. Cumulus Linux スイッチでの MTU の設定

以下の例では、Cumulus Linux スイッチでジャンボフレームを有効にします。

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

更新した設定を **sudo ifreload -a** を実行して再読み込みし、変更を適用することを忘れないでください。

8.4.5. LLDP ディスカバリーの設定

デフォルトでは、LLDP サービスはデーモン (lldpd) として実行され、スイッチのブート時に起動します。

全ポート/インターフェースの LLDP

```
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
```

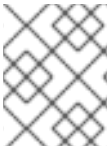
8.5. EXTREME NETWORKS EXOS スイッチの設定

8.5.1. トランクポートの設定

OpenStack Networking により、インスタンスは物理ネットワーク上にすでに存在する VLAN に接続することができます。**トランク** という用語は、単一のポートで複数の VLAN が通過を許可することを意味します。トランクにより、VLAN は、仮想スイッチを含む複数のスイッチを橋渡しすることができます。たとえば、物理ネットワークで **VLAN110** としてタグされたトラフィックが、コンピュートノードに到達すると、8021q モジュールによってタグ付けされたトラフィックが vSwitch 上の適切な VLAN にダイレクトされます。

8.5.1.1. Extreme Networks EXOS スイッチでのトランクポートの設定

X-670 シリーズのスイッチを使用する場合には、以下の例を参考にして、VLAN 110 と 111 のトラフィックがインスタンスに到達できるようにすることが可能です。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **24** に接続されていることを前提としています。この例では、**DATA** と **MNGT** が VLAN 名です。



注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

8.5.2. アクセスポートの設定

コンピュートノード上の全 NIC がインスタンスのトラフィックを伝送するわけではないので、複数の VLAN が通過できるように設定する必要はありません。このようなポートに設定する必要があるのは VLAN 1 つのみで、管理トラフィックやブロックストレージデータの転送などの他の運用上の要件を満たす必要がある可能性があります。これらのポートはアクセスポートとして一般的に知られており、必要な設定は通常、トランクポートよりも簡単です。

8.5.2.1. Extreme Networks EXOS スイッチのアクセスポートの設定

上記の図の例を続行するには、以下の例で (Extreme Networks X-670 シリーズスイッチ上の) 10 を eth1 のアクセスポイントとして設定します。次の設定を使用して、VLAN **110** と **111** のトラフィックがインスタンスに到達できるようにすることが可能です。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **10** に接続されていることを前提としています。

注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

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

例:

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

8.5.3. LACP ポートアグリゲーションの設定

LACP により、複数の物理 NIC をバンドルして単一の論理チャネルを形成することができます。LACP は、802.3ad (または、Linux ではボンディングモード 4) としても知られており、負荷分散と耐障害性のための動的なボンディングを作成します。LACP は、物理 NIC と物理スイッチポートの両方の物理エンドで設定する必要があります。

8.5.3.1. 物理 NIC 上での LACP の設定

1. **/home/stack/network-environment.yaml** ファイルを編集します。

```
- 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. Open vSwitch ブリッジが **LACP** を使用するように設定します。

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

ネットワークボンディングの設定方法についての説明は、『オーバークラウドの高度なカスタマイズ』を参照してください。

8.5.3.2. Extreme Networks EXOS スイッチでの LACP の設定

以下の例では、コンピュートノードに VLAN 100 を使用する NIC が 2 つあります。

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

例:

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

注記

LACP ネゴシエーションスクリプトでタイムアウトの期間を修正する必要がある場合があります。詳しくは、「LACP configured ports interfere with PXE/DHCP on servers」の記事を参照してください。

8.5.4. MTU の設定

特定の種別のネットワークトラフィックには、MTU サイズの調整が必要な場合があります。たとえば、特定の NFS または iSCSI のトラフィックには、ジャンボフレーム (9000 バイト) が推奨される場合があります。

注記

MTU の設定は、エンドツーエンド (トラフィックが通過すると想定されている全ホップ) で変更する必要があります。これには、仮想スイッチが含まれます。OpenStack 環境における MTU の変更についての説明は、「9章*MTU の設定*」を参照してください。

8.5.4.1. Extreme Networks EXOS スイッチでの MTU の設定

この例では、任意の Extreme Networks EXOS スイッチでジャンボフレームを有効にし、9000 バイトでの IP パケットの転送をサポートします。

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

例:

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

8.5.5. LLDP ディスカバリーの設定

ironic-python-agent サービスは、接続されたスイッチから LLDP パケットをリッスンします。収集される情報には、スイッチ名、ポートの詳細、利用可能な VLAN を含めることができます。Cisco Discovery Protocol (CDP) と同様に、LLDP は、director の **イントロスペクション** プロセス中の物理ハードウェアの検出を補助します。

8.5.5.1. Extreme Networks EXOS スイッチでの LLDP の設定

以下の例は、任意の Extreme Networks EXOS スイッチで LLDP を設定できるようにします。この例では、**11** がポートの文字列を示しています。

```
enable lldp ports 11
```

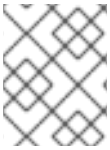
8.6. JUNIPER EX シリーズスイッチの設定

8.6.1. トランクポートの設定

OpenStack Networking により、インスタンスは物理ネットワーク上にすでに存在する VLAN に接続することができます。**トランク** という用語は、単一のポートで複数の VLAN が通過を許可することを意味します。トランクにより、VLAN は、仮想スイッチを含む複数のスイッチを橋渡しすることができます。たとえば、物理ネットワークで **VLAN110** としてタグされたトラフィックが、コンピュートノードに到達すると、8021q モジュールによってタグ付けされたトラフィックが vSwitch 上の適切な VLAN にダイレクトされます。

8.6.1.1. Juniper EX シリーズスイッチでのトランクポートの設定

Juniper JunOS を実行する Juniper EX シリーズのスイッチを使用する場合には、以下の設定を使用して、VLAN **110** と **111** のトラフィックがインスタンスに到達できるようにことが可能です。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **ge-1/0/12** に接続されていることを前提としています。



注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

```
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;
                }
            }
}
```

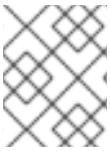
8.6.2. アクセスポートの設定

コンピュートノード上の全 NIC がインスタンスのトラフィックを伝送するわけではないので、複数の VLAN が通過できるように設定する必要はありません。このようなポートに設定する必要があるのは

VLAN 1 つのみで、管理トラフィックやブロックストレージデータの転送などの他の運用上の要件を満たす必要がある可能性があります。これらのポートはアクセスポートとして一般的に知られており、必要な設定は通常、トランクポートよりも簡単です。

8.6.2.1. Juniper EX シリーズスイッチのアクセスポートの設定

上記の図の例を続行するには、以下の例で (Juniper EX シリーズスイッチ上の) ge-1/0/13 を eth1 のアクセスポイントとして設定します。この設定は、物理ノードで、イーサネットケーブルが物理スイッチ上のインターフェース **ge-1/0/13** に接続されていることを前提としています。



注記

これらの値は、単なる例に過ぎません。変更せずにそのままコピーしてスイッチの設定に貼り付けると、機能が予期せず停止してしまう可能性があります。

```
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;
                }
            }
}
```

8.6.3. LACP ポートアグリゲーションの設定

LACP により、複数の物理 NIC をバンドルして単一の論理チャネルを形成することができます。LACP は、802.3ad (または、Linux ではボンディングモード 4) としても知られており、負荷分散と耐障害性のための動的なボンディングを作成します。LACP は、物理 NIC と物理スイッチポートの両方の物理エンドで設定する必要があります。

8.6.3.1. 物理 NIC 上での LACP の設定

1. **/home/stack/network-environment.yaml** ファイルを編集します。

```
- 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. Open vSwitch ブリッジが **LACP** を使用するように設定します。

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

ネットワークボンディングの設定方法についての説明は、『オーバークラウドの高度なカスタマイズ』を参照してください。

8.6.3.2. Juniper EX シリーズスイッチでの LACP の設定

以下の例では、コンピュートノードに VLAN 100 を使用する NIC が 2 つあります。

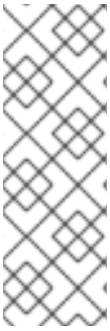
1. コンピュートノードの 2 つの NIC をスイッチ (例: ポート 12 と 13) に物理的に接続します。

2. ポートアグリゲートを作成します。

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

3. スイッチポート 12 (ge-1/0/12) と 13 (ge-1/0/13) を設定して、ポートアグリゲート **ae1** のメンバーに入れます。

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



注記

Red Hat OpenStack Platform director を使用したデプロイメントの場合には、ボンディングから PXE ブートするには、ボンディングのメンバーの 1 つを **lacp force-up** として設定する必要があります。これにより、イントロスペクションと初回ブート時には 1 つのボンディングメンバーのみが稼働状態になります。**lacp force-up** で設定されているボンディングメンバーは、**instackenv.json** に MAC アドレスが記載されているのと同じボンディングメンバーである必要があります (**ironic** に認識される MAC アドレスが **force-up** で設定されている MAC アドレスと同じである必要があります)。

4. ポートアグリゲート **ae1** で LACP を有効にします。

```
interfaces {
    ae1 {
        aggregated-ether-options {
```

```
            lacp {
                active;
            }
        }
    }
}
```

5. アグリゲート **ae1** を VLAN 100 に追加します。

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

6. 新しいポートチャネルを確認します。出力には、新規ポートアグリゲート **ae1** と、メンバーポートの **ge-1/0/12** および **ge-1/0/13** が表示されます。

```
> show lacp statistics interfaces ae1

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

注記

commit コマンドを実行して変更を適用することを忘れないようにしてください。

8.6.4. MTU の設定

特定の種別のネットワークトラフィックには、MTU サイズの調整が必要な場合があります。たとえば、特定の NFS または iSCSI のトラフィックには、ジャンボフレーム (9000 バイト) が推奨される場合があります。

注記

MTU の設定は、エンドツーエンド (トラフィックが通過すると想定されている全ホップ) で変更する必要があります。これには、仮想スイッチが含まれます。OpenStack 環境における MTU の変更についての説明は、「9章*MTU の設定*」を参照してください。

8.6.4.1. Juniper EX シリーズスイッチでの MTU の設定

以下の例では、Juniper EX4200 スイッチでジャンボフレームを有効にします。

注記

MTU 値の計算は、Juniper と Cisco のどちらのデバイスを使用しているかによって異なります。たとえば、Juniper の **9216** は、Cisco の **9202** に相当します。追加のバイトが L2 ヘッダーに使用され、Cisco はこれを指定された MTU 値に自動的に追加しますが、Juniper を使用する場合には、使用可能な MTU は指定値よりも 14 バイト少なくなります。従って、VLAN で MTU 値 **9000** をサポートするには、Juniper で MTU 値を **9014** に設定する必要があります。

1. Juniper EX シリーズスイッチの場合は、インターフェースごとに MTU の設定を実行します。以下のコマンドは、ge-1/0/14 および ge-1/0/15 ポート上のジャンボフレームを設定します。

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

注記

commit コマンドを実行して変更を保存することを忘れないようにしてください。

2. LACP アグリゲートを使用する場合には、メンバーの NIC ではなく、そのアグリゲートで MTU サイズを設定する必要があります。たとえば、以下のコマンドを実行すると、**ae1** アグリゲートの MTU サイズが設定されます。

```
set interfaces ae1 mtu 9216
```

8.6.5. LLDP ディスカバリーの設定

ironic-python-agent サービスは、接続されたスイッチから LLDP パケットをリッスンします。収集される情報には、スイッチ名、ポートの詳細、利用可能な VLAN を含めることができます。Cisco Discovery Protocol (CDP) と同様に、LLDP は、director の **イントロスペクション** プロセス中の物理ハードウェアの検出を補助します。

8.6.5.1. Juniper EX シリーズスイッチでの LLDP の設定

LLDP は、全インターフェースまたは個別のインターフェースのみでグローバルに有効にすることができます。

1. たとえば、LLDP を Juniper EX 4200 スイッチでグローバルに有効にします。

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

2. または、LLDP を単一のインターフェース **ge-1/0/14** のみで有効にします。

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

```
  }
  }
}
```

注記

commit コマンドを実行して変更を適用することを忘れないようにしてください。

パート II. 高度な設定

高度な OpenStack Networking の機能に関するクックブック形式のシナリオが含まれています。

第9章 MTU の設定

9.1. MTU の概要

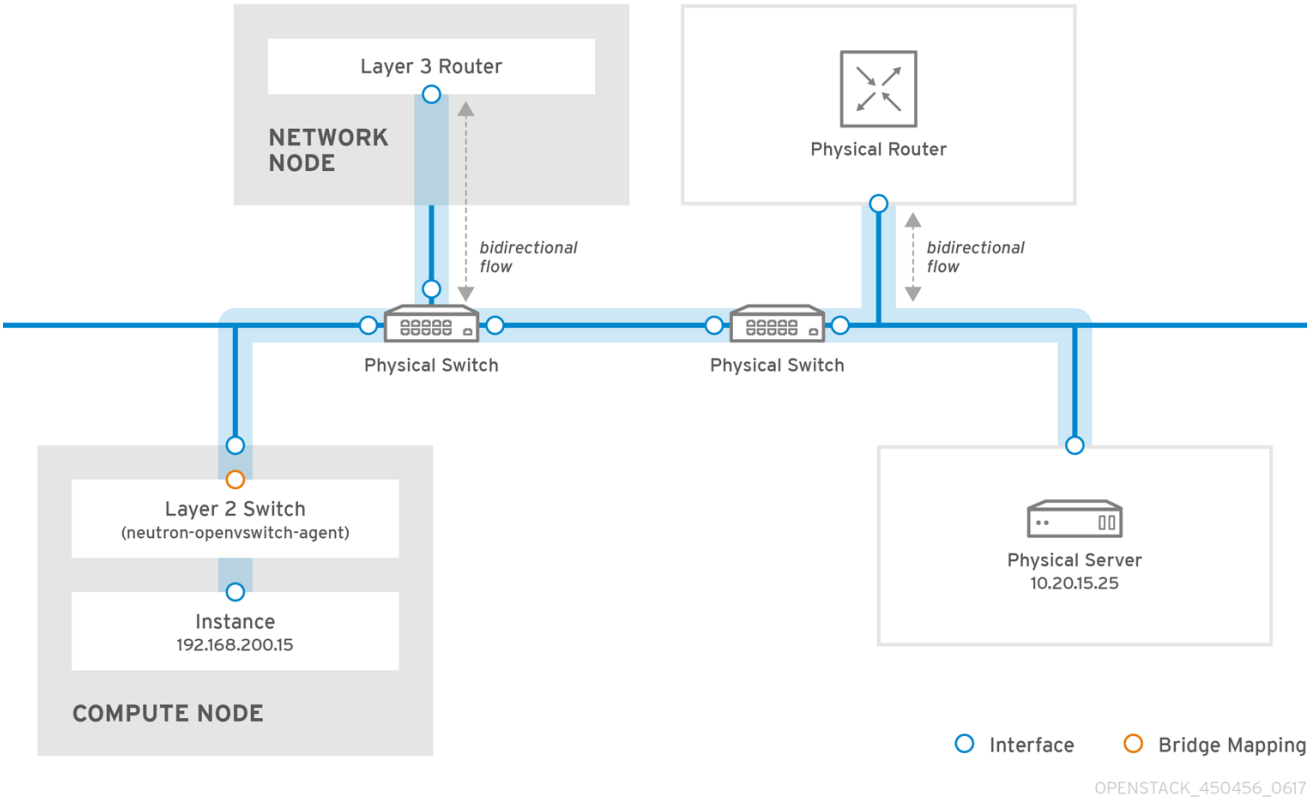
OpenStack Networking は、インスタンスに安全に適用することができる許容最大 MTU サイズの計算が可能です。MTU の値は、単一のネットワークパケットで転送できる最大データ量を指定します。この数は、アプリケーションに最も適したサイズによって変わります。たとえば、NFS 共有で必要な MTU サイズは VoIP アプリケーションで必要なサイズとは異なる場合があります。

注記

OpenStack Networking では、許容最大 MTU 値を計算して、**neutron net-show** コマンドで表示することができます。**net-mtu** は neutron API の拡張機能なので、一部の実装には含まれていない可能性があります。インスタンスがサポートしている場合には、必要な MTU 値を DHCPv4 クライアントに広告して自動設定を行うことが可能です。また、ルーター広告 (RA) パケットを使用して IPv6 クライアントに広告することも可能です。ルーター広告を送信するには、ネットワークがルーターに接続されている必要がある点に注意してください。

MTU 設定を正しく機能させるにはエンドツーエンドで一貫して設定する必要があります。つまり、MTU 設定は、仮想マシン自体、仮想ネットワークのインフラストラクチャー、物理ネットワーク、宛先のサーバー自体など、パケットが通過するすべてのポイントで同じサイズに指定する必要があります。

たとえば、以下の図の丸印は、インスタンスと物理サーバーの間のトラフィックに合わせて MTU 値を調節する必要があるポイントを示しています。ネットワークトラフィックを処理するインターフェースの MTU 値はすべて、特定の MTU サイズのパケットに対応するように変更する必要があります。トラフィックが **192.168.200.15** から物理サーバー **10.20.15.25** に伝送されることが想定される場合には、この変更が必要です。



MTU 値に一貫性がないと、ネットワークに複数の問題が発生します。最も一般的な問題は、ランダム

なパケットロスにより接続が中断して、ネットワークのパフォーマンスが低下することです。このような問題は、可能なすべてのネットワークポイントを特定してから、正しい MTU 値が設定されていることを検証する必要があるため、トラブルシューティングが困難です。

9.1.1. MTU 広告の設定

MTU 広告により、自動化された DHCP 設定や IPv6 RA 設定のレルムに MTU 設定が移動されるため、MTU 設定プロセスが簡素化されます。その結果、DHCPv4 または IPv6 RA を使用してインスタンスに最適な MTU サイズが広告されます。

MTU 広告は **/etc/neutron/neutron.conf** で有効にします。

```
advertise_mtu = True
```

このオプションは、Mitaka ではデフォルトで有効化されていますが、Newton では非推奨となり、今後のリリースでは削除される見込みです。このオプションを設定した場合には、テナントネットワークの設定済み MTU オプションが DHCPv4 および IPv6 RA を使用してインスタンスに広告されます。

注記

すべての DHCPv4 クライアントが MTU 値の自動設定をサポートしているわけではありません。

9.1.2. director での MTU の設定

以下の例では、NIC 設定テンプレートを使用した MTU の設定方法について説明します。MTU は、ブリッジ、ボンディング (該当する場合)、インターフェース、VLAN で設定する必要があります。

```
-
  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}
```

9.1.3. MTU 計算の確認

計算された MTU 値を確認します。これは、インスタンスが使用可能な MTU の最大許容値を計算した結果です。次に、ネットワークトラフィックの経路で関連する全インターフェースにこの値を設定して先に進むことができます。

```
# neutron net-show <network>
```

第10章 QUALITY-OF-SERVICE (QOS) の設定

Red Hat OpenStack Platform 11 ではネットワークの quality-of-service (QoS) ポリシーがサポートされるようになりました。これらのポリシーにより、OpenStack の管理者は、インスタンスの送信トラフィックに速度制限を適用して、さまざまなサービスレベルを提供することができます。QoS ポリシーを実装すると、指定した速度を超過するトラフィックはドロップされるようになります。

10.1. QOS ポリシースコープ

QoS ポリシーは個々のポートまたは特定のテナントネットワークに適用されます。テナントネットワークに適用される場合、そのポリシーは、ポリシーが適用されていないポートに継承されます。

10.2. QOS ポリシー管理

QoS ポリシーは、動的に適用、変更、削除することができます。以下の例では、帯域幅を制限するルールを手動で作成して、1 つのポートに適用します。

1. テナントの一覧を確認して、QoS ポリシーを作成するテナントの ID を決定します。

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

2. **admin** テナントに **bw-limiter** という名前の QoS ポリシーを作成します。

```
# neutron qos-policy-create 'bw-limiter' --tenant-id 98a2f53c20ce4d50a40dac4a38016c69
```

3. **bw-limiter** の規制ルールを設定します。

```
# neutron qos-bandwidth-limit-rule-create bw-limiter --max_kbps 3000 --max_burst_kbps
3000
```

4. **bw-limiter** ポリシーを適用する neutron ポートを設定します

```
# neutron port-update <port id> --qos-policy bw-limiter
```

5. QoS ルールを確認します。以下に例を示します。

```
# neutron qos-rule-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                               |
+-------------------+-----------------------------------+
```

以下の値により、規制のアルゴリズムを適宜に設定することができます。

- **max_kbps**: インスタンスが送信可能な最大速度 (Kbps 単位)
- **max_burst_kbps**: トークンのバッファーが満杯であった場合にそのポートが一度に送信することができたデータの最大量 (キロビット単位)。トークンのバッファーは「max_kbps」の速度で補充されます。

10.3. 送信トラフィックの DSCP マーキング

Differentiated Services Code Point (DSCP) では、IP ヘッダーに関連の値を埋め込むことで、ネットワーク上に QoS を実装することができます。OpenStack Networking (neutron) QoS ポリシーは、DSCP マーキングを使用して、neutron ポートとネットワーク上で送信トラフィックを管理することができるようになりました。現在、DSCP は Open vSwitch (OVS) を使用する VLAN とフラットプロバイダーネットワークのみに利用できます。今後、VXLAN もサポートされる予定です。

この実装では、最初にポリシーが作成され、DSCP ルールが定義されてポリシーに適用されます。これらのルールは、**--dscp-mark** パラメーターを使用して、DSCP マークに 10 進数の値を指定します。以下に例を示します。

1. 新規 QoS ポリシーを作成します。

```
neutron qos-policy-create qos-web-servers --tenant-id 98a2f53c20ce4d50a40dac4a38016c69
```

2. DSCP マーク **18** を使用して、DSCP ルールを作成し、**qos-web-servers** ポリシーに適用します。

```
neutron qos-dscp-marking-rule-create qos-web-servers --dscp-mark 18
Created a new dscp_marking_rule:
+-----------+--------------------------------------+
| Field     | Value                                |
+-----------+--------------------------------------+
| dscp_mark | 18                                   |
| id        | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----------+--------------------------------------+
```

3. QoS ポリシー **qos-web-servers** の DSCP ルールを表示します。

```
neutron qos-dscp-marking-rule-list qos-web-servers
+-----------+--------------------------------------+
| dscp_mark | id                                   |
+-----------+--------------------------------------+
|        18 | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----------+--------------------------------------+
```

4. **qos-web-servers** ポリシーに割り当てられた DSCP ルールの詳細を表示します。

```
neutron qos-dscp-marking-rule-show d7f976ec-7fab-4e60-af70-f59bf88198e6 qos-web-servers
+-----------+--------------------------------------+
| Field     | Value                                |
+-----------+--------------------------------------+
```

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

5. ルールに割り当てられた DSCP 値を変更します。

```
neutron qos-dscp-marking-rule-update d7f976ec-7fab-4e60-af70-f59bf88198e6 qos-web-servers --
dscp-mark 22
Updated dscp_marking_rule: d7f976ec-7fab-4e60-af70-f59bf88198e6
```

6. DSCP ルールを削除します。

```
neutron qos-dscp-marking-rule-delete d7f976ec-7fab-4e60-af70-f59bf88198e6 qos-web-servers
Deleted dscp_marking_rule(s): d7f976ec-7fab-4e60-af70-f59bf88198e6
```

10.4. QOS ポリシーの RBAC

Red Hat OpenStack Platform 11 では、QoS ポリシーのロールベースアクセス制御 (RBAC) が追加され、QoS ポリシーを特定のプロジェクトに適用できるようになりました。

たとえば、優先順位が低いネットワークトラフィックを許可する QoS ポリシーを作成して、特定のプロジェクトにのみ適用することができます。以下のコマンドでは、以前に作成した **bw-limiter** ポリシーが **demo** テナントに割り当てられます。

```
# neutron rbac-create 'bw-limiter' --type qos-policy --target-tenant
80bf5732752a41128e612fe615c886c6 --action access_as_shared
```

第11章 ブリッジマッピングの設定

本章では、Red Hat OpenStack Platform の **ブリッジマッピング** の設定方法を説明します。

11.1. ブリッジマッピングを使用する目的

ブリッジマッピングにより、プロバイダーネットワークのトラフィックは、物理ネットワークに到達することが可能となります。トラフィックは、ルーターの **qg-xxx** インターフェースからプロバイダーネットワークの外部に送出されて、**br-int** に達します。次に **br-int** と **br-ex** の間のパッチポートにより、トラフィックはプロバイダーネットワークのブリッジを通過して物理ネットワークまで到達することができます。

11.1.1. ブリッジマッピングの設定

以下は、**br-int** と **br-ex** のパッチピアの例です。

```
int-br-ex <-> phy-br-ex
```

この接続は、bridge_mappings で設定します。以下に例を示します。

```
bridge_mappings = physnet1:br-ex,physnet2:br-ex2
```

注記

bridge_mapping エントリーが見つからない場合には、ネットワーク接続が存在しないか、物理ネットワークへの通信が機能していません。

この設定の最初のエントリーは、パッチピアケーブルを使用して **br-int** と **br-ex** の間の接続を作成します。2 番目のエントリーは、**br-ex2** のパッチピアを作成します。

11.1.2. コントローラーノードの設定

bridge_mappings の設定は、コントローラーノードの **network_vlan_ranges** オプションと相関する必要があります。たとえば、上記の例では、コントローラーノードを以下のように設定します。

```
network_vlan_ranges = physnet1:10:20,physnet2:21:25
```

これらの値により、対応する外部ネットワークを表すプロバイダーが作成されてから、外部ネットワークはルーターのインターフェースを介してテナントのネットワークに接続されます。このため、ルーターがスケジュールされているネットワークノード上で **bridge_mappings** を設定する必要があります。これは、プロバイダーネットワークによって表されているルーターのトラフィックが正しい物理ネットワーク (例: **physnet1**) を使用して送信できることを意味します。

11.1.3. トラフィックの流れ

この設定は、接続の作成以外に、**br-int** と **br-ex** で OVS フローを設定し、外部ネットワークとの間でネットワークトラフィックの送受信が可能になります。外部ネットワークはそれぞれ、内部 VLAN id で表され、ルーターの **qg-xxx** ポートにタグ付けされます。パケットが **phy-br-ex** に到達すると、**br-ex** ポートは VLAN タグをストリップ化して、このパケットを物理インターフェース、その後に外部ネットワークに移動します。外部ネットワークからのリターンパケットは **br-ex** に到達して、**phy-br-ex <→**

int-br-ex を使用することで **br-int** に移動されます。パケットが **int-br-ex** に到達すると、**br-int** の別のフローがパケットに内部 VLAN タグを追加します。これにより、パケットは **qg-xxx** により受理可能になります。

11.2. ブリッジマッピングのメンテナンス

マッピングを削除した後には、patch port のクリーンアップを実行する必要があります。この操作により、ブリッジ設定から誤ったエントリーが確実に消去されます。このタスクを実行するには 2 つのオプションを利用できます。

- 手動ポートクリーンアップ: 不要なポートを慎重に削除する必要があります。ネットワーク接続を停止する必要はありません。
- **neutron-ovs-cleanup** を使用した自動ポートクリーンアップ: クリーンアップが自動で実行されますが、ネットワーク接続を停止する必要があります。また、必要なマッピングを再度追加する必要があります。このオプションは、ネットワーク接続を停止しても支障がない場合に選択してください。

各オプションの例を以下に示します。

11.2.1. 手動ポートクリーンアップ

手動ポートクリーンアッププロセスは、不要なポートを削除します。ネットワーク接続を停止する必要がありません。これらのポートは命名規則を使用して特定することができます。**br-$external_bridge** では、"phy-"**$external_bridge、br-int では "int-"**$external_bridge という名前が指定されます。

以下の例に記載する手順では、bridge_mappings からブリッジを削除し、対応するポートをクリーンアップします。**1. openvswitch_agent.ini** を編集して、**bridge_mappings** から **physnet2:br-ex2** のエントリーを削除します。

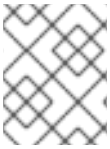
```
bridge_mappings = physnet1:br-ex,physnet2:br-ex2
```

physnet2:br-ex2 のエントリーを削除します。削除した後の **bridge_mappings** は以下のようになります。

```
bridge_mappings = physnet1:br-ex
```

2. ovs-vsctl を使用して、削除した **physnet2:br-ex2** エントリーに関連付けられた patch port を削除します。

```
# ovs-vsctl del-port br-ex2 phy-br-ex2
# ovs-vsctl del-port br-int int-br-ex2
```



注記

bridge_mappings エントリー全体が削除またはコメントアウトされた場合には、各エントリーに対してクリーンアップコマンドを実行する必要があります。

3. neutron-openvswitch-agent を再起動します。

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

11.2.2. 「neutron-ovs-cleanup」を使用した自動ポートクリーンアップ

この操作は **neutron-ovs-cleanup** コマンドに **--ovs_all_ports** フラグを指定して実行します。**neutron** サービスまたはノード全体を再起動して、ブリッジを通常の稼動状態に戻します。このプロセスでは、ネットワーク接続を完全に停止する必要があります。

neutron-ovs-cleanup コマンドは、すべての OVS ブリッジから全ポート (インスタンス、qdhcp/qrouter など) を抜線します。**--ovs_all_ports** フラグを使用すると **br-int** から全ポートが削除されて、**br-tun** からトンネルエンドが、またブリッジ間からは patch port がクリーンアップされます。また、物理インターフェース (例: eth0、eth1) はブリッジ (例: br-ex、br-ex2) から削除されます。これにより、**ovs-vsctl** を使用してポートを手動で再度追加するまでの間は、インスタンスへの接続はできなくなります。

```
# ovs-vsctl add-port br-ex eth1
```

11.2.2.1. neutron-ovs-cleanup の使用例

1. **openvswitch_agent.ini**> に記載されている bridge_mapping のエントリーをバックアップします。

2. **neutron-ovs-cleanup** コマンドに **--ovs_all_ports** フラグを指定して実行します。このステップにより、ネットワーク接続が完全に停止される点に注意してください。

```
# /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
```

3. OpenStack Networking サービスを再起動します。

```
# systemctl restart neutron-openvswitch-agent
# systemctl restart neutron-l3-agent.service
# systemctl restart neutron-dhcp-agent.service
```

4. **openvswitch_agent.ini** に **bridge_mapping** エントリーを再度追加して、接続をリストアします。

5. **neutron-openvswitch-agent** サービスを再起動します。

```
# systemctl restart neutron-openvswitch-agent
```

注記

OVS エージェントを再起動する際には、**bridge_mappings** に設定されていない接続には影響はありません。**br-int** を **br-ex2** に接続していて、かつ **br-ex2** にフローがある場合には、**bridge_mappings** 設定から削除 (または完全にコメントアウト) してしまうと、サービスやノードを再起動するなど、どのような操作を行っても、2 つのブリッジは切断されません。

第12章 VLAN 対応のインスタンス

12.1. 概要

インスタンスは、単一の仮想 NIC を使用して、VLAN のタグが付いたトラフィックを送受信できるようになりました。この機能は、単一の仮想 NIC で複数の顧客/サービスに対応できるので、特に VLAN のタグが付いたトラフィックを想定する NFV アプリケーション (VNF) に役立ちます。

たとえば、テナントのデータネットワークは VLAN またはトンネリング (VXLAN/GRE) の分割を使用できますが、インスタンスからは VLAN ID がタグ付けされたトラフィックが見えるので、ネットワークパケットはネットワーク全体でタグ付けが必要なわけではなく、インスタンスに注入される直前にタグ付けされます。

親ポートを作成して、既存の neutron ネットワークにアタッチすることで実装を開始します。これにより、作成した親ポートにトランクの接続が追加されます。次にサブポートを作成します。このサブポートは、VLAN とインスタンスを接続するポートで、このポートを使用することでトランクに接続を確立できます。インスタンスのオペレーティングシステム内で、サブポートが関連付けられた VLAN のトラフィックをタグ付けするサブインターフェースを作成する必要があります。

12.2. トランクプラグインのレビュー

director ベースのデプロイメントでは、トランクプラグインはデフォルトでオンになっています。コントローラーノードで設定をレビューすることができます。

1. コントローラーノード上で、**/etc/neutron/neutron.conf** の **trunk** プラグインが有効化されていることを確認します。以下に例を示します。

```
service_plugins=router,metering,qos,trunk
```

12.3. トランク接続の作成

1. トランクポートの接続を必要とするネットワークを特定します。これは、トランクされた VLAN にアクセスを必要とするインタンスが含まれるネットワークのことです。以下の例では、このネットワークは **パブリック** ネットワークとなっています。

```
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. 親のトランクポートを作成して、インスタンスの接続先のネットワークにアタッチします。この例では、**parent-trunk-port** という名前の neutron ポートが **パブリック** ネットワーク上に作成されます。このトランクは、**サブポート** の作成に使用できるので、**親** ポートと見なされます。

```
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. 前のステップで作成したポートを使用して、トランクを作成します。この例では、トランクは **parent-trunk** という名前です。

```
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. トランクの接続を確認します。

```
openstack network trunk list
+--------------------------------------+--------------+--------------------------------------+-------------+
| ID                                   | Name         | Parent Port                          | Description |
+--------------------------------------+--------------+--------------------------------------+-------------+
| 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 | parent-trunk | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
|
+--------------------------------------+--------------+--------------------------------------+-------------+
```

- トランク接続の詳細を表示します。

```
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. トランクへのサブポートの追加

1. neutron ポートを作成します。このポートは、トランクへのサブポート接続として使用されます。また、親ポートに割り当てられた MAC アドレスを指定する必要があります。

```
openstack port create --network private --mac-address fa:16:3e:33:c4:75 subport-trunk-port
+-----------------------+-------------------------------------------------------------------------+
| Field                 | Value                                                                   |
+-----------------------+-------------------------------------------------------------------------+
| admin_state_up        | UP                                                                      |
| allowed_address_pairs |                                                                         |
| binding_host_id       |                                                                         |
| binding_profile       |                                                                         |
| binding_vif_details   |                                                                         |
| binding_vif_type      | unbound                                                                 |
| binding_vnic_type     | normal                                                                  |
| created_at            | 2016-10-20T02:08:14Z                                                    |
| description           |                                                                         |
| device_id             |                                                                         |
| device_owner          |                                                                         |
| extra_dhcp_opts       |                                                                         |
| fixed_ips             | ip_address='10.0.0.11', subnet_id='1a299780-56df-4c0b-a4c0-c5a612cef2e8' |
| headers               |                                                                         |
| id                    | 479d742e-dd00-4c24-8dd6-b7297fab3ee9                                    |
| mac_address           | fa:16:3e:33:c4:75                                                       |
| name                  | subport-trunk-port                                                      |
| network_id            | 3fe6b758-8613-4b17-901e-9ba30a7c4b51                                    |
| project_id            | 745d33000ac74d30a77539f8920555e7                                        |
| project_id            | 745d33000ac74d30a77539f8920555e7                                        |
| revision_number       | 4                                                                       |
| security_groups       | 59e2af18-93c6-4201-861b-19a8a8b79b23                                    |
| status                | DOWN                                                                    |
| updated_at            | 2016-10-20T02:08:15Z                                                    |
+-----------------------+-------------------------------------------------------------------------+
```

注記

HttpException: Conflict のエラーが発生した場合には、親のトランクポートのあるネットワークとは異なるネットワークで、サブポートを作成していることを確認してください。この例では、親トランクポートに **public** ネットワークを、サブポートには **private** を使用します。

2. トランク (**parent-trunk**) とポートを関連付けて、VLAN ID (**55**) を指定します。

```
openstack network trunk set --subport port=subport-trunk-port,segmentation-
type=vlan,segmentation-id=55 parent-trunk
```

12.5. トランクを使用するためのインスタンスの設定

neutron がサブポートに割り当てた MAC アドレスを使用するには、インスタンスのオペレーティングシステムを設定する必要があります。サブポート作成のステップで、固有の MAC アドレスを使用するようにサブポートも設定してください。

1. ネットワークトランクの設定を確認します。

```
$ openstack network trunk list
+--------------------------------------+-------------+--------------------------------------+-------------+
| ID                                   | Name        | Parent Port                          | Description |
+--------------------------------------+-------------+--------------------------------------+-------------+
| 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 | parent-trunk | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
|
+--------------------------------------+-------------+--------------------------------------+-------------+


$ 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 | 2                                                                                               |
| status          | DOWN                                                                                            |
| sub_ports       | port_id='479d742e-dd00-4c24-8dd6-b7297fab3ee9', segmentation_id='55',
segmentation_type='vlan' |
| tenant_id       | 745d33000ac74d30a77539f8920555e7                                                                |
| updated_at      | 2016-10-20T02:10:06Z                                                                            |
+-----------------+-------------------------------------------------------------------------------------------------+
```

2. 親ポートの **id** を仮想 NIC として使用するインスタンスを作成します。

```
nova boot --image cirros --flavor m1.tiny testInstance --security-groups default --key-name sshaccess
--nic port-id=20b6fdf8-0d43-475a-a0f1-ec8f757a4a39


+--------------------------------------+-----------------------------------------------+
```

```
| 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                             | 2016-10-20T03:02:51Z                         |
| description                         | -                                            |
| flavor                              | m1.tiny (1)                                  |
| hostId                              |                                              |
| host_status                         |                                              |
| id                                  | 88b7aede-1305-4d91-a180-67e7eac8b70d         |
| 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                           | 745d33000ac74d30a77539f8920555e7             |
| updated                             | 2016-10-20T03:02:51Z                         |
| user_id                             | 8c4aea738d774967b4ef388eb41fef5e             |
+-------------------------------------+----------------------------------------------+
```

12.6. トランクの状態

- **ACTIVE**: トランクは想定通りに機能しており、現在要求はありません。
- **DOWN**: トランクの仮想/物理リソースが同期されていません。これは、ネゴシエーション中の一時的な状態である場合があります。
- **BUILD**: 要求があり、リソースがプロビジョニングされています。操作が正常に完了すると、トランクは **ACTIVE** に戻ります。

- **DEGRADED**: プロビジョニング要求が完了しなかったため、トランクは一部のみプロビジョニングされました。サブポートを削除して操作を再試行することを推奨します。

- **ERROR**: プロビジョニング要求は成功しませんでした。エラーの原因となったリソースを削除すると、トランクは正常な状態に戻ります。**ERROR** 状態の間には、それ以上サブポートを追加しないでください。問題がさらに発生する原因となる可能性があります。

第13章 RBAC の設定

OpenStack Networking の Role-based Access Control (RBAC) により、**neutron** 共有ネットワークに対する、より粒度の高い制御が可能となります。以前のリリースでは、ネットワークは全テナントで共有するか、全く共有しないかのいずれかでした。本リリースでは、OpenStack Networking は RBAC テーブルを使用してテナント間における **neutron** ネットワークの共有を制御するようになりました。これにより、管理者は、ネットワークにインスタンスを接続するパーミッションをどのテナントに付与するかをコントロールすることができます。
その結果、クラウド管理者は、一部のテナントがネットワークを作成できないようにして、そのプロジェクトに対応した既存のネットワークに接続できるようにすることが可能です。

13.1. 新規 RBAC ポリシーの作成

以下の手順では、RBAC ポリシーを使用してテナントに共有ネットワークへのアクセスを許可する方法の実例を紹介します。

1. 利用可能なネットワークの一覧を表示します。

```
# neutron net-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. テナントの一覧を表示します。

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

3. **auditors** テナント (**4b0b98f8c6c040f38ba4f7146e8680f5**) へのアクセスを許可する **web-servers** ネットワークの RBAC エントリーを作成します。

```
# neutron rbac-create fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 --type network --target-tenant
4b0b98f8c6c040f38ba4f7146e8680f5 --action access_as_shared
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_tenant | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| tenant_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

これにより、**auditors** プロジェクトでインスタンスを **web-servers** ネットワークに接続できるようになります。

13.2. 設定したRBAC ポリシーの確認

1. 既存の RBAC ポリシーの ID を取得するには、**neutron rbac-list** オプションを使用してください。

```
# neutron 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. **neutron rbac-show** を使用して特定の RBAC エントリーの詳細を表示します。

```
# neutron 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_tenant | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| tenant_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

13.3. RBAC ポリシーの削除

1. 既存の RBAC の ID を取得するには、**neutron rbac-list** オプションを使用してください。

```
# neutron 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. **neutron rbac-delete** コマンドで対象の ID 値を使用して RBAC を削除します。

```
# neutron rbac-delete 314004d0-2261-4d5e-bda7-0181fcf40709
Deleted rbac_policy: 314004d0-2261-4d5e-bda7-0181fcf40709
```

13.4. 外部ネットワークの RBAC

--action access_as_external パラメーターを使用して、外部ネットワーク (ゲートウェイインターフェースがアタッチされているネットワーク) への RBAC アクセスを許可することができます。

たとえば、以下の手順では **web-servers** ネットワークの RBAC を作成して、engineering テナント (**c717f263785d4679b16a122516247deb**) へのアクセスを許可します。

1. **--action access_as_external** を使用して新しい RBAC ポリシーを作成します。

```
# neutron rbac-create 6e437ff0-d20f-4483-b627-c3749399bdca --type network --target-tenant
c717f263785d4679b16a122516247deb --action access_as_external
 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_tenant | c717f263785d4679b16a122516247deb     |
| tenant_id     | c717f263785d4679b16a122516247deb     |
+---------------+--------------------------------------+
```

2. 上記のコマンドを実行した結果、Engineering テナントのユーザーは、ネットワークの表示やそのネットワークへのインスタンスの接続が可能になります。

```
$ neutron net-list
+--------------------------------------+-------------+-----------------------------------------------------+
| id                                   | name        | subnets                                             |
+--------------------------------------+-------------+-----------------------------------------------------+
| 6e437ff0-d20f-4483-b627-c3749399bdca | web-servers | fa273245-1eff-4830-b40c-57eaeac9b904
192.168.10.0/24 |
+--------------------------------------+-------------+-----------------------------------------------------+
```

第14章 分散仮想ルーター (DVR) の設定

Red Hat OpenStack Platform 13 をデプロイする場合には、そのデプロイメントに対して集中ルーティングモデルまたは DVR のオプションから選択することができます。DVR は完全にサポートされており、設定オプションとして利用できますが、Red Hat OpenStack Platform 13 director のデフォルトの設定は、集中ルーティングのままとなっています。

集中ルーティングも、DVR も有効なルーティングモデルで、それぞれに短所、長所がある点に留意することが重要です。本書を使用して、集中ルーティングと DVR のどちらがニーズにより適しているかを慎重に計画することを推奨します。

14.1. レイヤー 3 ルーティングの概要

OpenStack Networking (neutron) は、プロジェクトネットワークにルーティングサービスを提供します。ルーターがない場合には、プロジェクトネットワーク内のインスタンスは、共有 L2 ブロードキャストドメインでのみ相互通信が可能となります。ルーターを作成して、テナントネットワークに割り当てると、そのネットワークのインスタンスが他のプロジェクトネットワークやアップストリームと通信することができます (外部ゲートウェイがルーターに定義されている場合)。

14.1.1. ルーティングのフロー

OpenStack のルーティングサービスは主に、3 つのフローに分類できます。

- **East-West ルーティング**: 同じテナント内の異なるネットワーク間のトラフィックのルーティング。このトラフィックは OpenStack デプロイメント外には出ません。この定義は、IPv4 と IPv6 のサブネット両方に提供されます。

- **Floating IP を使用した North-South ルーティング**: Floating IP のアドレス指定とはインスタンス間をフロートする変更可能な 1 対 1 の NAT であるという説明が最も適しています。Floating IP は、Floating IP と neutron ポートの間での 1 対 1 の関連付けとしてモデル化されていますが、Floating IP は NAT の変換を実行する neutron ルーターとの関連付けで実装されています。Floating IP 自体は、外部と接続可能なルーターを提供するアップリンクネットワークから取得しているため、インスタンスと (インターネットのエンドポイントなど) 外部のリソースとの間の通信が可能です。Floating IP は、IPv4 の概念で、IPv6 には適用されません。プロジェクトが使用する IPv6 のアドレス指定は、プロジェクト全体で重複のないグローバルユニキャストアドレス (GUA) を使用することが前提であるため、NAT なしにルーティングが可能です。

- **Floating IP なしの North-South ルーティング** (別名: **SNAT**): Neutron は、Floating IP が割り当てられていないインスタンスに、デフォルトのポートアドレス変換 (PAT) サービスを提供します。このサービスを使用すると、インスタンスはルーター経由で外部のエンドポイントと通信ができますが、外部のエンドポイントからはインスタンスへは通信ができません。たとえば、インターネット上の Web サイトをブラウズできますが、外部の Web ブラウザーはこのインスタンス内でホストされている Web サイトをブラウスすることはできません。SNAT は、IPv4 トラフィックのみに適用されます。さらに、GUA プレフィックスが割り当てられた neutron プロジェクトネットワークでは、外部にアクセスするために neutron ルーターの外部ゲートウェイポート上に NAT は必要ありません。

14.1.2. 集中ルーティング

neutron は当初、集中ルーティングモデルで設計されました。このモデルでは、neutron L3 エージェントで管理されるプロジェクトの仮想ルーターはすべて専用のノードまたはノードのクラスター (ネットワークノードまたはコントローラーノード) にデプロイされるので、ルーティングの機能が必要となる度に (East/West、Floating IP または SNAT)、トラフィックはトポロジー内の専用のノードを経由します。そのため、複数の課題が発生し、トラフィックフローは最適な状態ではありませんでした。以下に例を示します。

- コントローラーノード経由で伝送されるインスタンス間のトラフィック: L3 を使用して 2 つのインスタンス間で通信する必要がある場合に、トラフィックはコントローラーノードを経由する必要があります。同じコンピュートノードでインスタンスがそれぞれスケジューリングされている場合でも、トラフィックはコンピュートノードを離れてからコントローラーを通過して、コンピュートノードに戻ってくる必要があるため、パフォーマンスに悪影響を与えます。

- コントローラーノード経由でパケットを送受信するインスタンス (Floating IP を使用): 外部ネットワークのゲートウェイインターフェースはコントローラーノードのみで利用できるので、トラフィックはインスタンスから開始される場合でも、外部ネットワークにあるインスタンスを宛先とする場合でも、トラフィックはコントローラーノードを経由する必要があります。その結果、大規模な環境では、コントローラーノードにかかるトラフィックの負荷が高くなり、パフォーマンスやスケーラビリティーに影響を及ぼします。また、外部ネットワークのゲートウェイインターフェースで十分な帯域幅を確保できるように慎重に計画する必要があります。SNAT トラフィックにも同じ要件が適用されます。

L3 エージェントのスケーリングを改善するには、neutron で複数のノードに仮想ルーターを分散する L3 HA の機能を使用することができます。コントローラーノードが失われた場合には、HA ルーターは別のノードのスタンバイにフェイルオーバーして、HA ルーターのフェイルオーバーが完了するまではパケットが失われます。この機能は、Red Hat Enterprise Linux OpenStack Platform 6 から導入されており、デフォルトで有効になっています。

14.2. DVR の概要

分散仮想ルーティング (DVR) では、別のルーティング設計も提供されており、Red Hat OpenStack Platform 11 以降のバージョンで完全にサポートされています。これは、コントローラーノードの障害のあるドメインを分離して、L3 エージェントをデプロイしてネットワークトラフィックを最適化し、全コンピュートノードにルーターをスケジューリングすることが目的です。DVR の使用時には以下が可能です。

- East-West トラフィックは分散されて、コンピュートノード上で直接ルーティングされます。

- Floating IP を持つインスタンスの North-South トラフィックは、分散されて、コンピュートノードにルーティングされます。これには、外部ネットワークを全コンピュートノードに接続する必要があります。

- Floating IP を持たないインスタンスの North-South トラフィックは分散されず、依然として専用のコントローラーノードが必要です。

  - ノードコントローラーノード上の L3 エージェントを新しい **dvr_snat** モードで設定して、ノードが SNAT トラフィックのみにサービスを提供するようにします。

- neutron のメタデータエージェントは分散され、全コンピュートノード上にデプロイされます。このメタデータのプロキシーサービスは分散された全ルーター上でホストされます。

14.3. 既知の問題および警告

注記

Red Hat OpenStack Platform 14 では、カーネルバージョンが **kernel-3.10.0-514.1.1.el7** 以降でない場合には、DVR は使用しないでください。

- DVR のサポートは、ML2 のコアプラグイン、Open vSwitch (OVS) のメカニズムドライバー、および OVN に制限されます。他のバックエンドはサポートされません。

- DVR が有効化されている場合でも、SNAT トラフィックは分散されません。SNAT は機能しますが、送信/受信トラフィックは中央のコントローラーノードを経由する必要があります。

- DVR が有効化されている場合でも、IPv6 トラフィックは分散されません。IPv6 ルーティングは機能しますが、送信/受信トラフィックはすべて中央のコントローラーノードを経由する必要があります。IPv6 ルーティングを広範囲にわたり、使用している場合には、今回は DVR を使用しないことを推奨します。

- DVR は、L3 HA を使用する場合にはサポートされません。Red Hat OpenStack Platform 14 director では、DVR を使用する場合は、L3 HA はオフになります。つまり、ルーターはこれまでどおりネットワークノードでスケジューリングされ (また L3 エージェントの間で負荷が共有され) ますが、エージェントが機能しなくなると、このエージェントがホストするルーターすべてが機能しなくなります。影響があるのは、SNAT トラフィックのみです。**allow_automatic_l3agent_failover** 機能を使用すると、1 つのネットワークノードが失敗してもルーターは別のノードで再スケジュールされるので、このような場合に推奨されます。

- neutron DHCP エージェントが管理する DHCP サーバーは分散されず、コントローラーノードにデプロイされます。Red Hat OpenStack Platform では、ルーティング設計 (集中型または DVR) にかかわらず、DHCP エージェントは高可用性構成でコントローラーノードにデプロイされます。

- Floating IP の場合は、各コンピュートノードに **外部** ネットワーク上のインターフェースが 1 つ必要です。また、外部ゲートウェイポートの実装と Floating IP ネットワークの名前空間が原因で、各コンピュートノードに追加の IP アドレスが 1 つ必要となりました。

- プロジェクトデータの分離において、VLAN、GRE、VXLAN のすべてがサポートされます。GRE または VXLAN を使用する場合は、L2 Population 機能はオンにする必要があります。Red Hat OpenStack Platform director では、インストール時にこれは強制的に有効になります。

14.4. サポートされているルーティングアーキテクチャー

- 集中 HA ルーティング: Red Hat Enterprise Linux OpenStack Platform 8 から Red Hat OpenStack Platform 14

- 分散ルーティング: Red Hat OpenStack Platform 12 以降

- 集中 HA ルーティングを実行する Red Hat OpenStack Platform 8 以降のデプロイメントから分散ルーティングのみを使用する Red Hat OpenStack Platform 10 以降のデプロイメントへのアップグレード

14.5. ML2 OVS を使用した DVR のデプロイ

neutron-ovs-dvr.yaml の環境ファイルは、必須の DVR 固有のパラメーターを設定します。任意のデプロイメント構成の DVR を設定するには他にも考慮する事項があります。要件は以下のとおりです。

(a) 外部ネットワークトラフィック用の物理ネットワークに接続されたインターフェースは、コンピュートノードとコントローラーノードの両方で設定すること。

(b) コンピュートノードおよびコントローラーノードでブリッジを作成して、外部ネットワークトラフィック用のインターフェースを設定すること。

(c) Neutron がこのブリッジを使用できるように設定すること。

ホストのネットワーク設定 (**a** および **b**) は Heat テンプレートが制御しており、これらのテンプレートにより、**os-net-config** プロセスで使用できるように、Heat が管理するノードに設定が渡されます。これは基本的には、ホストのネットワークのプロビジョニングを自動化しています。プロビジョニングし

たネットワーク環境と一致するように、Neutron も設定する必要があります (**c**)。デフォルトの設定は、実稼動環境で機能するように想定されていません。たとえば、通常のデフォルト設定を使用する概念実証用の環境は、以下のようになります。

1. **environments/neutron-ovs-dvr.yaml** の **OS::TripleO::Compute::Net::SoftwareConfig** の値が使用中の **OS::TripleO::Controller::Net::SoftwareConfig** の値と同じであることを確認します。これは通常、**environments/net-multiple-nics.yaml** など、オーバークラウドのデプロイ時に使用するネットワーク環境ファイルに含まれます。これにより、コンピュートノードの L3 エージェントに適した外部のネットワークブリッジが作成されます。

注記

コンピュートノードのネットワーク設定をカスタマイズした場合には、これらのファイルに適切な設定を追加する必要がある場合があります。

2. **OS::TripleO::Compute::Ports::ExternalPort: ../network/ports/external.yaml** など、**OS::TripleO::Compute::Ports::ExternalPort** を適切な値に変更して、外部ネットワークにあるコンピュートノードの neutron ポートを設定します。

3. オーバークラウドのデプロイ時に環境ファイルとして **environments/neutron-ovs-dvr.yaml** を追加します。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/neutron-ovs-dvr.yaml
```

4. L3 HA が無効になっていることを確認します。

実稼動環境 (または、ネットワークの分離、専用の NIC など、特別にカスタマイズする必要があるテスト環境) の場合には、環境の例は指針として利用してください。L2 エージェント (例: OVS) が使用するブリッジマッピングタイプのパラメーターや、他のエージェント (例: L3 エージェント) の外部向けのブリッジへの参照には、細心の注意を払う必要があります。

注記

L3 エージェントの外部ブリッジの設定は現在も提供されていますが非推奨となっており、今後廃止される予定です。

14.6. 集中ルーティングから分散ルーティングへの移行

本項では、L3 HA 集中ルーティングを使用する Red Hat OpenStack Platform デプロイメントの分散ルーティングへのアップグレードパスについて説明します。

1. デプロイメントをアップグレードして、正しく機能していることを確認します。

2. 「ML2 OVS を使用した DVR のデプロイ」 に記載の手順に従って、director のスタック更新を実行して DVR を設定します。

3. 既存のルーターでルーティングがまだ機能していることを確認します。

4. L3 HA ルーターを直接 **分散型** に移行することはできません。代わりに、各ルーターで L3 HA オプションを無効にしてから分散型のオプションを有効にします。

4a. ルーターの **admin** 状態を無効にします。

```
$ neutron router-update --admin-state-up=False
```

4b. ルーターを **レガシー** タイプに変換します。

```
$ neutron router-update --ha=False
```

4c. ルーターが DVR を使用するように設定します。

```
$ neutron router-update --distributed=True
```

4d. ルーターを **admin** 状態に切り替えます。

```
$ neutron router-update --admin-state-up=True
```

4e. ルーティングがまだ機能しており、分散型になったことを確認します。

第15章 NEUTRON LBAASV2 API を使用した LOAD BALANCING-AS-A-SERVICE の設定

Load Balancing-as-a-Service (LBaaS) は、OpenStack Networking が指定のインスタンス間で受信要求を均等に分散できるようにします。このステップバイステップのガイドでは、OpenStack Networking が Open vSwitch (OVS) プラグインで LBaas を使用するように設定します。

Red Hat OpenStack Platform 5 で導入された Load Balancing-as-a-Service (LBaaS) は、OpenStack Networking が指定のインスタンス間で受信要求を均等に分散できるようにします。これにより、インスタンス間でのワークロードが予測可能な方法で共有されるようになり、システムリソースのより効率的な使用が可能となります。受信要求は、以下の負荷分散メソッドの 1 つを使用して分散されます。

- **ラウンドロビン**: 複数のインスタンス間で要求を均等にローテーションします。
- **送信元 IP アドレス**: 同じ送信元 IP アドレスからの要求は常に一定のインスタンスへ送信されます。
- **最小コネクション**: アクティブな接続が最も少ないインスタンスに要求が割り当てられます。

表15.1 LBaaS の機能

機能	説明
監視機能	LBaaS は、PING、TCP、HTTP、HTTPS GET メソッドを使用した可用性モニタリング機能を提供します。モニターは、要求の処理にプールメンバーが利用可能かどうかを判断するために実装されます。
管理	LBaaS は、さまざまなツールセットを使用して管理されます。REST API は、プログラムベースの管理およびスクリプト作成に使用できます。ユーザーは、CLI (neutron) または OpenStack Dashboard のいずれかを使用して、ロードバランサーの管理タスクを行います。
接続制限	接続制限を使用して、受信トラフィックのシェーピングを行うことができます。この機能は、ワークロードを制御することも可能で、DoS (Denial of Service) 攻撃の緩和にも有効です。
セッションの永続化	LBaaS は、複数のインスタンスで構成されるプール内で同じインスタンスに受信要求がルーティングされるようにすることで、セッションの永続化をサポートします。LBaaS は、クッキーや送信元 IP アドレスに基づいたルーティングの決定をサポートします。

注記

LBaaS は現在 IPv4 アドレスのみサポートします。

注記

LBaaSv1 は Red Hat OpenStack Platform 11 (Ocata) で削除され、LBaaSv2 に置き換えられました。

15.1. OPENSTACK NETWORKING および LBAAS トポロジー

OpenStack Networking (neutron) サービスは、大きく 2 つのカテゴリーに分類することができます。

1. **Neutron API server**: このサービスは、エンドユーザーとサービスが OpenStack Networking と対話できるように、主に API を提供する OpenStack Networking API サーバーを実行します。またこのサービスは、基盤のデータベースと対話して、テナントネットワーク、ルーター、ロードバランサーの詳細などを保存する役割も果たします。

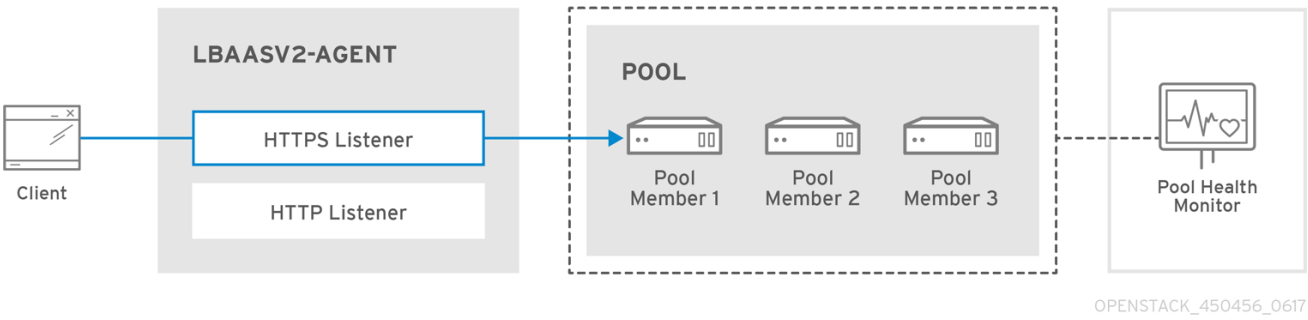
2. **Neutron エージェント**: OpenStack Networking のさまざまな機能を提供するサービスです。

- **neutron-dhcp-agent**: テナントプライベートネットワークの DHCP IP アドレスを管理します。
- **neutron-l3-agent**: テナントプライベートネットワーク、外部ネットワークなどの間のレイヤー 3 ルーティングを容易化します。
- **neutron-lbaasv2-agent**: テナントにより作成された LBaaS ルーターをプロビジョニングします。

注記

neutron-lbassv2-agent (HAProxy を使用) は非推奨となりました。Octavia を使用したロードバランシングの推奨されるリファレンス実装については、「*16章Octavia を使用した Load Balancing-as-a-Service (LBaaS)*」を参照してください。

以下の図には、HTTPS トラフィックからプールメンバーへのフローを示しています。



15.1.1. LBaaS のサポートステータス

- LBaaS v1 API は、バージョン 10 で削除されました。
- LBaaS v2 API は Red Hat OpenStack Platform 7 で導入されました。これは、バージョン 10 で提供されている唯一の LBaaS API です。
- LBaaS デプロイメントは現在 Red Hat OpenStack Platform director ではサポートされていません。

注記

neutron-lbassv2-agent (HAProxy を使用) は非推奨となり、サポートされなくなりました。Octavia を使用したロードバランシングの推奨されるリファレンス実装については、「16章*Octavia を使用した Load Balancing-as-a-Service (LBaaS)*」を参照してください。neutron-lbaas RPM は、サードパーティーのプラグインに対応するための API をサポートするために、現在も提供されています。

15.1.2. サービスの配置

OpenStack Networking Service は、同じ物理サーバーまたは別の専用サーバーで実行することができます。

注記

Red Hat OpenStack Platform 11 では、コンポーザブルロールのサポートが追加され、ネットワークサービスをカスタムロール別に分類することができます。ただし、本ガイドでは、内容をわかりやすくするために、デプロイメントにはデフォルトのコントローラーノードを使用することを前提とします。

API サーバーを実行するサーバーは通常、**コントローラーノード** と呼ばれ、OpenStack Networking エージェントを実行するサーバーは **Network node** と呼ばれます。理想的な実稼動環境では、パフォーマンスやスケーラビリティーの理由により、コンポーネントを専用のノードに分類しますが、テストまたは PoC デプロイメントではすべてのコンポーネントを 1 つの同じノードで実行します。本章では、どちらのシナリオにも対応しますが、コントローラーノードの設定のセクションは API サーバーで、ネットワークノードのセクションは LBaaS エージェントを実行するサーバーで行う必要があります。

注記

コントローラーおよびネットワークロールの両方が同じ物理ノードに存在する場合には、この物理ノード (サーバー) で手順を実行する必要があります。

第16章 OCTAVIA を使用した LOAD BALANCING-AS-A-SERVICE (LBAAS)

OpenStack Load-balancing サービス (Octavia) は、Red Hat OpenStack platform director ベースのインストール環境で、Load Balancing-as-a-Service (LBaaS) バージョン 2 の実装を提供します。本項では、Octavia を有効化する方法について説明します。Octavia サービスは、Networking API サーバーと同じノードでホストされていることを前提とします。デフォルトでは、ロードバランシングサービスはコントローラーノード上で実行されます。

注記

Octavia を使用した LBaaSv2 では、現在プラグインをサポートしていません。商用の OpenStack ロードバランシングソリューションを利用の場合には、引き続き LBaaSv2 API を使用する必要があります。詳しくは、「*15章Neutron LBaaSv2 API を使用した Load Balancing-as-a-Service の設定*」を参照してください。

16.1. OCTAVIA の概要

Octavia は、**amphora** と呼ばれるコンピュートノード上の一式のインスタンスを使用します。amphora との通信は、ロードバランシングの管理ネットワーク (**lb-mgmt-net**) 上で行われます。

Octavia には以下が含まれています。

API コントローラー (octavia_api container)

設定更新および amphora インスタンスのデプロイ、モニタリング、削除でコントローラーワーカーと通信します。

コントローラーワーカー (octavia_worker container)

LB ネットワーク上で設定および設定の更新を amphora に送信します。

ヘルスマネージャー

個々の amphora の正常性をモニタリングし、amphora で予期せずにエラーが発生した場合にフェイルオーバーイベントを処理します。

ハウスキーピングマネージャー

古い (削除済みの) データベースレコードのクリーンアップ、spares プールの管理、amphora の証明書のローテーションの管理を行います。

ロードバランサー

ロードバランシングエンティティーを表す最上位の API オブジェクト。仮想 IP アドレスは、ロードバランサーの作成時に確保されます。ロードバランサーの作成時には、amphora インスタンスがコンピュートノードで起動されます。

Amphora

ロードバランシングを行うインスタンス。Amphora は通常、コンピュートノード上で実行されるインスタンスで、リスナー、プール、ヘルスモニター、L7 ポリシー、メンバーで設定されます。Amphora はヘルスマネージャーに定期的なハートビートを送信します。

リスナー

ロードバランシングされるサービスのリスニングエンドポイント (例: HTTP)。リスナーは、複数のプールを参照する場合があります (レイヤー 7 ルールを使用してそれらのプールを切り替えます)。

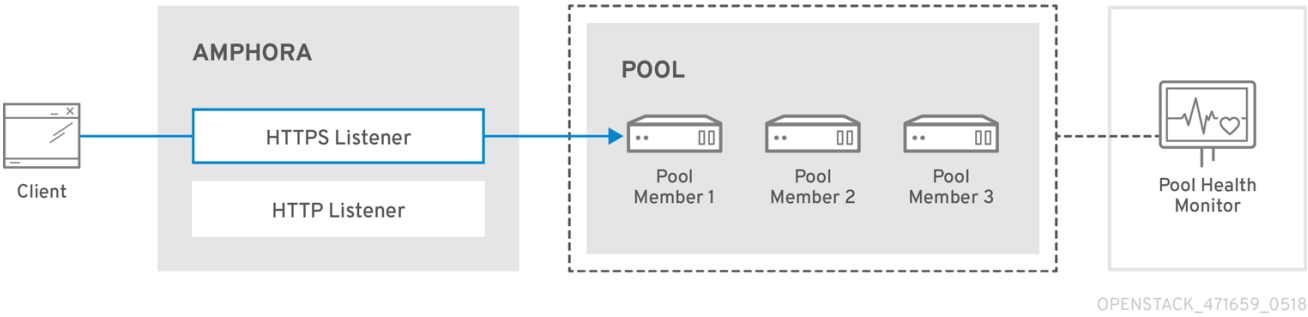
プール

ロードバランサー (amphora) からのクライアント要求を処理するメンバーのグループ。1 つのプールは 1 つのリスナーにのみ関連付けられます。

メンバー

プール内のロードバランサー (amphora) の背後で、トラフィックを処理する Compute インスタンス

以下の図には、HTTPS トラフィックからプールメンバーへのフローを示しています。



16.2. ソフトウェア要件

Octavia には、以下の OpenStack コアコンポーネントを設定する必要があります。

- Compute (nova)
- Networking (**allowed_address_pairs** を有効化)
- Image (glance)
- Key Manager (barbican) TLS オフロード機能が有効化されている場合
- Identity (keystone)
- RabbitMQ
- MySQL

16.3. アンダークラウドの前提条件

本項では、アンダークラウドがインストール済みで、Octavia を有効化してオーバークラウドをデプロイする準備が整っていることを前提としています。コンテナーベースのデプロイメントのみがサポートされています。Octavia はコントローラーノードで実行されます。

注記

既存のオーバークラウドデプロイメントで Octavia サービスを有効にする場合には、アンダークラウドを準備する必要があります。準備を行わないと、Octavia が動作していない状態でオーバークラウドのインストールは成功と報告されます。アンダークラウドを準備するには、『Transitioning to Containerized Services』を参照してください。

16.4. OCTAVIA デプロイメントのプランニング

Red Hat OpenStack Platform は、ロードバランシングサービスのデプロイ後のステップを簡素化するためのワークフロータスクを提供しています。**tripleo-common/workbooks/octavia_post.yaml** ワークブックは、**tripleo-heat-templates/docker/services/octavia/octavia-deployment-config.yaml** ファイルから設定されます。

この Octavia ワークフローは、Ansible Playbook のセットを実行して、オーバークラウドのデプロイの最終段階として、以下のデプロイ後のステップを提供します。

- 証明書とキーを設定します。
- amphora、Octavia コントローラーワーカー、ヘルスマネージャーの間のロードバランシング管理ネットワークを設定します。

> **注記**
>
> OpenStack Heat テンプレートは直接編集しないでください。カスタムの環境ファイル (例: **octavia-environment.yaml**) を作成して、デフォルトのパラメーター値をオーバーライドしてください。

Amphora イメージ

director はデフォルトの amphora イメージを自動的にダウンロードして、オーバークラウドの Image サービスにアップロードし、Octavia がその amphora イメージを使用するように設定します。director は、スタックの更新またはアップグレード中に、このイメージを最新の amphora イメージに更新します。

> **注記**
>
> カスタムの amphora イメージはサポートされていません。

16.4.1. Octavia の証明書とキーの設定

Octavia のコンテナーでは、ロードバランサーとの通信、および相互の通信をセキュアに実行する必要があります。独自の証明書とキーを指定することができます。適切なパラメーターをカスタムの環境ファイル (例: **octavia-environment.yaml**) に追加します。

ユーザーが提供する証明書とキーの設定

OctaviaGenerateCerts パラメーターを **false** に設定して、独自の証明書とキーを Octavia に指定することができます。

以下の例は、証明書とキーを自分で指定するためのパラメーターの設定値を示しています。

```
parameter_defaults:
  OctaviaCaCert: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    [snip]
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----

  OctaviaCaKey: |
    -----BEGIN RSA PRIVATE KEY-----
    Proc-Type: 4,ENCRYPTED
    [snip]
    -----END RSA PRIVATE KEY-----[

  OctaviaClient: |
    Certificate:
      Data:
        Version: 3 (0x2)
        Serial Number: 3 (0x3)
    [snip]
```

```
    -----END PRIVATE KEY-----

  OctaviaCaKeyPassphrase:
    b28c519a-5880-4e5e-89bf-c042fc75225d

  OctaviaGenerateCerts: false
[rest of file snipped]
```

注記

証明書とキーは、複数行の値で、すべての行を同じレベルにインデントする必要があります。

16.5. OCTAVIA のデプロイ

Octavia をデプロイするには、Octavia のパラメーターの適切な値を判断して、オーバーライドする値をカスタムの環境ファイル (例: **octavia-environment.yaml**) で設定します。

16.5.1. ロードバランシングネットワークの設定

以下のデフォルト値を使用してロードバランシングネットワークを設定することができます。

```
OctaviaControlNetwork: 'lb-mgmt-net'
OctaviaControlSubnet: 'lb-mgmt-subnet'
OctaviaControlSecurityGroup: 'lb-mgmt-sec-group'
OctaviaControlSubnetCidr: '172.24.0.0/16'
OctaviaControlSubnetGateway: '172.24.0.1'
OctaviaControlSubnetPoolStart: '172.24.0.2'
OctaviaControlSubnetPoolEnd: '172.24.255.254'
```

16.5.2. 管理ポートの設定

この手順はオプションです。デフォルトの設定値を使用しない場合には、カスタムの環境ファイル (例: **octavia-environment.yaml**) を使用して以下のパラメーターをオーバーライドします。

```
OctaviaMgmtPortDevName: "o-hm0"
```

16.5.3. director を使用した Octavia のデプロイ

お使いの環境が Octavia のイメージにアクセスできることを確認してください。イメージの登録方法に関する詳しい情報は、『director のインストールと使用方法』の「コンテナー化」を参照してください。

オーバークラウドで Octavia をデプロイする手順

```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml
```

注記

director は、スタックの更新またはアップグレード中に amphora イメージを最新の amphora イメージに更新します。

16.6. HTTP ロードバランサーの設定

簡単な HTTP ロードバランサーを設定する手順

1. サブネット上にロードバランサーを作成します。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id private-subnet
```

ロードバランサーが作成されるまで待ちます。オプションで **openstack loadbalancer show lb1** コマンドを使用して、ロードバランスのステータスが **Active** および **ONLINE** になるのを確認することができます。仮想 IP のポート ID は、後のステップで使用します。

2. リスナーを作成します。

```
$ openstack loadbalancer listener create --name listener1 --protocol HTTP --protocol-port 80
lb1
```

3. リスナーのデフォルトプールを作成します。

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
listener1 --protocol HTTP
```

4. プール上にヘルスモニターを作成して、"/healthcheck" パスをテストします。

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
HTTP --url-path /healthcheck pool1
```

5. プールにロードバランサーのメンバーを追加します。

```
$ openstack loadbalancer member create --subnet-id private-subnet --address 192.0.2.10 --
protocol-port 80 pool1
$ openstack loadbalancer member create --subnet-id private-subnet --address 192.0.2.11 --
protocol-port 80 pool1
```

6. パブリックのサブネットに Floating IP アドレスを作成します。

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

7. この Floating IP をロードバランサーの仮想 IP ポートに割り当てます。

```
$ openstack floating ip set --port `_LOAD_BALANCER_VIP_PORT_` `_FLOATING_IP_`
```

16.7. ロードバランサーの検証

ロードバランサーを検証する手順

1. ロードバランサーの設定を確認するには、**openstack loadbalancer show** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer show lb1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
```

```
| admin_state_up      | True                                 |
| created_at          | 2018-04-18T12:28:34                  |
| description         |                                      |
| flavor              |                                      |
| id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | octavia                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2018-04-18T14:03:09                  |
| vip_address         | 192.168.0.11                         |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. ロードバランサー lb1 に関連付けられた amphora の UUID を確認するには、**amphora list** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer amphora list | grep <UUID of loadbalancer lb1>
```

3. amphora の情報を確認するには、amphora の UUID と共に **amphora show** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer amphora show 62e41d30-1484-4e50-851c-7ab6e16b88d0
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| id              | 62e41d30-1484-4e50-851c-7ab6e16b88d0 |
| loadbalancer_id | 53a497b3-267d-4abc-968f-94237829f78f |
| compute_id      | 364efdb9-679c-4af4-a80c-bfcb74fc0563 |
| lb_network_ip   | 192.168.0.13                         |
| vrrp_ip         | 10.0.0.11                            |
| ha_ip           | 10.0.0.10                            |
| vrrp_port_id    | 74a5c1b4-a414-46b8-9263-6328d34994d4 |
| ha_port_id      | 3223e987-5dd6-4ec8-9fb8-ee34e63eef3c |
| cert_expiration | 2020-07-16T12:26:07                  |
| cert_busy       | False                                |
| role            | BACKUP                               |
| status          | ALLOCATED                            |
| vrrp_interface  | eth1                                 |
| vrrp_id         | 1                                    |
| vrrp_priority   | 90                                   |
| cached_zone     | nova                                 |
| created_at      | 2018-07-17T12:26:07                  |
| updated_at      | 2018-07-17T12:30:36                  |
+-----------------+--------------------------------------+
```

4. リスナーの情報を確認するには、**openstack loadbalancer listener show** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer listener show listener1
+---------------------------+--------------------------------------------------------------------------+
| Field                     | Value                                                                    |
+---------------------------+--------------------------------------------------------------------------+
| admin_state_up            | True                                                                     |
| connection_limit          | -1                                                                       |
| created_at                | 2018-04-18T12:51:25                                                      |
| default_pool_id           | 627842b3-eed8-4f5f-9f4a-01a738e64d6a                                     |
| default_tls_container_ref | http://10.0.0.101:9311/v1/secrets/7eafeabb-b4a1-4bc4-8098-
b6281736bfe2 |
| description               |                                                                          |
| id                        | 09f28053-fde8-4c78-88b9-0f191d84120e                                     |
| insert_headers            | None                                                                     |
| l7policies                |                                                                          |
| loadbalancers             | 788fe121-3dec-4e1b-8360-4020642238b0                                     |
| name                      | listener1                                                                |
| operating_status          | ONLINE                                                                   |
| project_id                | dda678ca5b1241e7ad7bf7eb211a2fd7                                         |
| protocol                  | TERMINATED_HTTPS                                                         |
| protocol_port             | 443                                                                      |
| provisioning_status       | ACTIVE                                                                   |
| sni_container_refs        | []                                                                       |
| updated_at                | 2018-04-18T14:03:09                                                      |
+---------------------------+--------------------------------------------------------------------------+
```

5. プールとロードバランサーのメンバーを確認するには、**openstack loadbalancer pool show** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer pool show pool1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2018-04-18T12:53:49                  |
| description         |                                      |
| healthmonitor_id    |                                      |
| id                  | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| lb_algorithm        | ROUND_ROBIN                          |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| loadbalancers       | 788fe121-3dec-4e1b-8360-4020642238b0 |
| members             | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
|                     | 40db746d-063e-4620-96ee-943dcd351b37 |
| name                | pool1                                |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol            | HTTP                                 |
| provisioning_status | ACTIVE                               |
| session_persistence | None                                 |
| updated_at          | 2018-04-18T14:03:09                  |
+---------------------+--------------------------------------+
```

6. Floating IP アドレスを確認するには、**openstack floating ip list** コマンドを使用します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack floating ip list
+--------------------------------------+---------------------+------------------+------------------------------
--------+-------------------------------------+----------------------------------+
| ID                                   | Floating IP Address | Fixed IP Address | Port                         |
Floating Network                     | Project                          |
+--------------------------------------+---------------------+------------------+------------------------------
--------+-------------------------------------+----------------------------------+
| 89661971-fa65-4fa6-b639-563967a383e7 | 10.0.0.213          | 192.168.0.11     | 69a85edd-
5b1c-458f-96f2-b4552b15b8e6 | fe0f3854-fcdc-4433-bc57-3e4568e4d944 |
dda678ca5b1241e7ad7bf7eb211a2fd7 |
+--------------------------------------+---------------------+------------------+------------------------------
--------+-------------------------------------+----------------------------------+
```

7. HTTPS トラフィックがロードバランサー全体に流れることを確認します。

```
(overcloud) [stack@undercloud-0 ~]$ curl -v https://10.0.0.213 --insecure
* About to connect() to 10.0.0.213 port 443 (#0)
*   Trying 10.0.0.213...
* Connected to 10.0.0.213 (10.0.0.213) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
*  subject: CN=www.example.com,O=Dis,L=Springfield,ST=Denial,C=US
*  start date: Apr 18 09:21:45 2018 GMT
*  expire date: Apr 18 09:21:45 2019 GMT
*  common name: www.example.com
*  issuer: CN=www.example.com,O=Dis,L=Springfield,ST=Denial,C=US
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 10.0.0.213
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
* Connection #0 to host 10.0.0.213 left intact
```

第17章 IPV6 を使用したテナントネットワーク

本章では、テナントネットワークに IPv6 サブネットを実装する方法について説明します。director 7.3 では、テナントネットワークに加えて、IPv6 ネイティブのデプロイメントをオーバークラウドノード用に設定することが可能です。

Red Hat OpenStack Platform 6 から、テナントネットワークに IPv6 のサポートが追加されました。IPv6 サブネットは、既存のテナントネットワーク内で作成され、Stateless Address Autoconfiguration (SLAAC)、**Stateful DHCPv6**、**Stateless DHCPv6** の複数のアドレス割り当てモードをサポートします。本章では、IPv6 サブネット作成のオプションについて説明し、それらのステップを実行する手順の例を記載します。

17.1. IPV6 サブネットのオプション

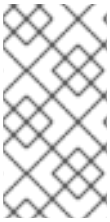
IPv6 サブネットは、**neutron subnet-create** コマンドを使用して作成します。また、オプションとして、アドレスモードとルーター広告モードを指定することができます。これらのオプションの設定可能な組み合わせは以下のとおりです。

RA モード	アドレスモード	結果
ipv6_ra_mode=not set	ipv6-address-mode=slaac	インスタンスは、**SLAAC** を使用して外部ルーター (OpenStack Networking で管理されていないルーター) から IPv6 アドレスを受信します。
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	インスタンスは、**DHCPv6 stateful** を使用して、OpenStack Networking (dnsmasq) から IPv6 アドレスとオプションの情報を受信します。
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateless	インスタンスは、SLAAC を使用して外部ルーターから IPv6 アドレスを受信し、**DHCPv6 stateless** を使用して OpenStack Networking (dnsmasq) からオプションの情報を受信します。
ipv6_ra_mode=slaac	ipv6-address-mode=not-set	インスタンスは、SLAAC を使用して OpenStack Networking (**radvd**) から IPv6 アドレスを受信します。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=not-set	インスタンスは、**DHCPv6 stateful** を使用して、外部の DHCPv6 サーバーから IPv6 アドレスとオプションの情報を受信します。

RA モード	アドレスモード	結果
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=not-set	インスタンスは、SLAAC を使用して OpenStack Networking (**radvd**) から IPv6 アドレスを受信し、**DHCPv6 stateless** を使用して外部 DHCPv6 サーバーからオプションの情報を受信します。
ipv6_ra_mode=slaac	ipv6-address-mode=slaac	インスタンスは、**SLAAC** を使用して OpenStack Networking (**radvd**) から IPv6 アドレスを受信します。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=dhcpv6-stateful	インスタンスは、**DHCPv6 stateful** を使用して OpenStack Networking (**dnsmasq**) から IPv6 アドレスを、**DHCPv6 stateful** を使用して OpenStack Networking (**dnsmasq**) から任意の情報を取得します。
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	インスタンスは、**SLAAC** を使用して OpenStack Networking (**dnsmasq**) から IPv6 アドレスを、**DHCPv6 stateless** を使用して OpenStack Networking (**dnsmasq**) から任意の情報を取得します。

17.1.1. Stateful DHCPv6 を使用した IPv6 サブネットの作成

以下の手順では、上記で説明した設定を使用してテナントネットワークに IPv6 サブネットを作成します。最初のステップでは、テナントとネットワークに関する必要な情報を収集し、次にその情報を使用してサブネットを作成するコマンドを構築します。



注記

OpenStack Networking は、SLAAC には EUI-64 IPv6 アドレスの割り当てのみをサポートします。これにより、ホストは Base 64 ビットと MAC アドレスに基づいて自らアドレスを割り当てるため、IPv6 ネットワークが簡素化されます。SLAAC の異なるネットマスクと **address_assign_type** を使用してサブネットの作成を試みると失敗します。

1. IPv6 サブネットを作成するプロジェクトのテナント ID を取得します。これらの値は、OpenStack デプロイメント固有なので、実際に使用する値は異なります。以下の例では、**QA** テナントが IPv6 サブネットを受信します。

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
```

```
| 25837c567ed5458fbb441d39862e1399 |    QA    |
| f59f631a77264a8eb0defc898cb836af |  admin   |
| 4e2e1951e70643b5af7ed52f3ff36539 |   demo   |
| 8561dff8310e4cd8be4b6fd03dc8acf5 | services |
+----------------------------------+----------+
```

2. OpenStack Networking (neutron) 内の全ネットワークの一覧を取得します。IPv6 サブネットをホストするネットワークの名前を書き留めておきます。以下の例では、**database-servers** を使用します。

```
# neutron net-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. 上記のステップから **QA tenant-id** (**25837c567ed5458fbb441d39862e1399**) を使用してネットワークを作成するコマンドを構築します。もう 1 つの要件は、IPv6 サブネットをホストする宛先ネットワークの名前です。以下の例では、**database-servers** ネットワークを使用しています。

```
# neutron subnet-create --ip-version 6 --ipv6_address_mode=dhcpv6-stateful --tenant-id
25837c567ed5458fbb441d39862e1399 database-servers fdf8:f53b:82e4::53/125

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

4. ネットワークの一覧をチェックして、設定を確認します。**database-servers** のエントリーには新規作成された IPv6 サブネットが反映されている点に注意してください。

```
# neutron net-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        |
+--------------------------------------+------------------+-------------------------------------------------------------+
```

この設定により、**QA** テナントで作成されたインスタンスが **database-servers** サブネットに追加されると、DHCP IPv6 アドレスを受信できるようになります。

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

第18章 テナントクォータの管理

本章では、OpenStack Networking コンポーネントのテナント/プロジェクトクォータ管理について説明します。

OpenStack Networking (neutron) は、テナント/プロジェクトが作成するリソースの数を制限するクォータの使用をサポートします。たとえば、以下のように、**neutron.conf** ファイルの **quota_router** の値を変更することで、テナントが作成可能なルーター数を制限することができます。

```
quota_router = 10
```

この設定では、テナントごとにルーターを 10 個まで設定できるように制限します。

さまざまなネットワークコンポーネントに、さらなるクォータ設定が提供されています。

18.1. L3 クォータオプション

L3 ネットワークで利用可能なクォータオプション: **quota_floatingip** (テナント 1 つにつき許容される Floating IP の数)、**quota_network** (テナント 1 つにつき許容されるネットワークの数)、**quota_port** (テナント 1 つにつき許容されるポートの数)、**quota_router** (テナント 1 つにつき許容されるルーターの数)、**quota_subnet** (テナント 1 つにつき許容されるサブネットの数)、**quota_vip** (テナント 1 つにつき許容される仮想 IP の数)

18.2. ファイアウォールのクォータオプション

ファイアウォール管理を統括するクォータオプション: **quota_firewall** (テナント 1 つにつき許容されるファイアウォール数)、**quota_firewall_policy** (テナント 1 つにつき許容されるファイアウォールのポリシー数)、**quota_firewall_rule** (テナント 1 つにつき許容されるファイアウォールのルール数)

18.3. セキュリティーグループのクォータオプション

セキュリティーグループの許容数を管理するクォータオプション: **quota_security_group** (テナント 1 つにつき許容されるセキュリティーグループ数)、**quota_security_group_rule** (テナント 1 つにつき許容されるセキュリティーグループルール数)

18.4. 管理クォータオプション

管理者が検討するクォータオプション: **default_quota** (テナント 1 つにつき許容されるデフォルトのリソース数)、**quota_health_monitor** (テナント 1 つにつき許容されるヘルスモニターの数。ヘルスモニターはリソースを消費しませんが、OpenStack Networking のバックエンドではメンバーをリソースのコンシューマーとして処理するため、クォータオプションの利用が可能)、**quota_member** (テナント 1 つにつき許容されるメンバーの数。メンバーはリソースを消費しませんが、OpenStack Networking のバックエンドがメンバーをリソースのコンシューマーとして処理するため、クォータオプションの利用が可能)、**quota_pool** (テナント 1 つにつき許容されるプール数)

第19章 FIREWALL-AS-A-SERVICE (FWAAS) の設定

Firewall-as-a-Service (FWaaS) プラグインは、OpenStack Networking (neutron) に境界ファイアウォール管理を追加します。FWaaS は iptables を使用して、プロジェクト内の全 Networking ルーターにファイアウォールポリシーを適用し、プロジェクトごとにファイアウォールポリシーと論理ファイアウォールインスタンス 1 つをサポートします。

FWaaS は、OpenStack Networking (neutron) ルーターでトラフィックをフィルタリングして、境界で動作します。これは、セキュリティーグループとは異なり、インスタンスレベルで動作します。

注記

FWaaS は現在テクノロジープレビューとなっており、テストされていない操作は推奨されません。

以下のサンプル図では、**VM2** インスタンスの送信/受信トラフィックのフローを示しています。

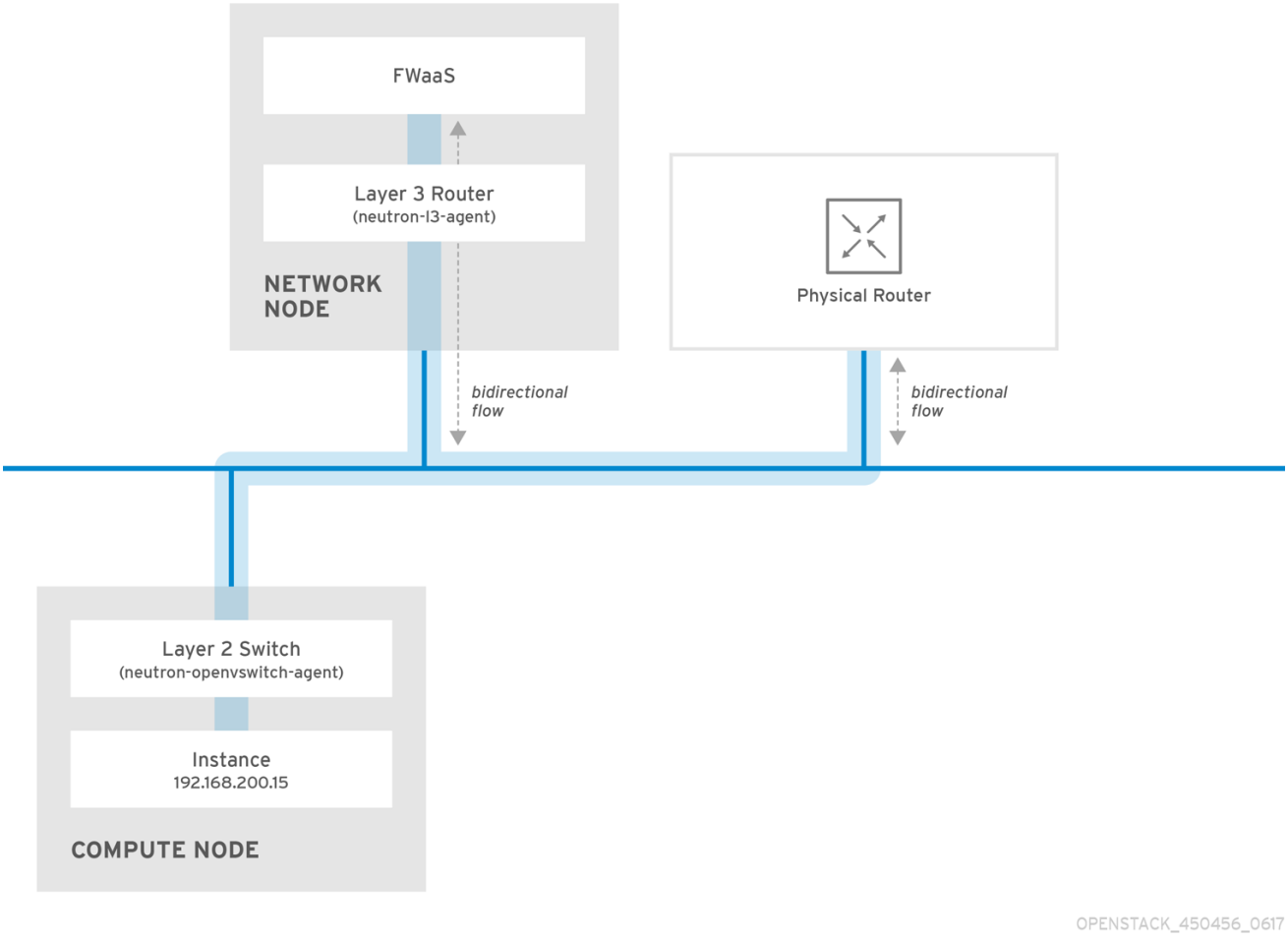


図 1. FWaaS アーキテクチャー

19.1. FWAAS の有効化

1. FWaaS パッケージをインストールします。

```
# yum install openstack-neutron-fwaas python-neutron-fwaas
```

2. **neutron.conf** ファイルで FWaaS プラグインを有効化します。

```
service_plugins = neutron.services.firewall.fwaas_plugin.FirewallPlugin
```

3. **fwaas_driver.ini** ファイルで FWaaS を設定します。

```
[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. FWaaS 管理オプションは、OpenStack Dashboard で利用できます。通常、コントローラーノードに設置されている **local_settings.py** ファイルで、これらのオプションを有効化します。

```
/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py
'enable_firewall' = True
```

5. **neutron-server** を再起動して変更を適用します。

```
# systemctl restart neutron-server
```

19.2. FWAAS の設定

まず、ファイアウォールルールを作成して、これらのルールを含めるポリシーを作成します。次に、ファイアウォールを作成して、ポリシーを適用します。

1. ファイアウォールルールを作成します。

```
$ neutron firewall-rule-create --protocol <tcp|udp|icmp|any> --destination-port <port-range> --action <allow|deny>
```

CLI では、プロトコルの値が必要です。ルールがプロトコルに依存しない場合は、**any** の値を使用することができます。

2. ファイアウォールポリシーを作成します。

```
$ neutron firewall-policy-create --firewall-rules "<firewall-rule IDs or names separated by space>" myfirewallpolicy
```

上記で指定したルールの順番は重要です。ルールなしで、ファイアウォールポリシーを作成して、update 操作 (複数のルールの追加時) または insert-rule 操作 (単一ルールの追加時) のいずれかを使用してから、後でルールを追加することができます。

注記: FWaaS は常に、各ポリシーの最も低い優先順位で、「default deny all」ルールを追加します。そのため、デフォルトでは、ルールを持たないファイアウォールポリシーは全トラフィックをブロックします。

19.3. ファイアウォールの作成

```
$ neutron firewall-create <firewall-policy-uuid>
```

ファイアウォールは、OpenStack Networking のルーターが作成され、インターフェースが接続されるまで、PENDING_CREATE の状態に留まります。

第20章 ALLOWED-ADDRESS-PAIRS の設定

Allowed-address-pairs では、サブネットに関わらず、ポートを通過する mac_address/ip_address (CIDR) ペアを指定することができます。これにより、2 つのインスタンス間をフロートして、データプレーンによるフェイルオーバーを加速化することが可能な、VRRP などのプロトコルを使用することができます。

注記

allowed-address-pairs 拡張は、現在 ML2 および Open vSwitch のプラグインのみでサポートされています。

allowed-address-pairs の基本操作

ポートを作成して、1 つのアドレスペアを許可します。

```
# neutron port-create net1 --allowed-address-pairs type=dict list=true mac_address=
<mac_address>,ip_address=<ip_cidr>
```

allowed-address-pairs の追加

```
# neutron port-update <port-uuid> --allowed-address-pairs type=dict list=true mac_address=
<mac_address>,ip_address=<ip_cidr>
```

注記

OpenStack Networking では、ポートの **mac_address** と **ip_address** が一致する allowed-address-pair の設定ができません。その理由は、**mac_address** と **ip_address** が一致するトラフィックはすでにポートを通過できるので、このような設定をしても効果がないためです。

第21章 レイヤー 3 高可用性の設定

本章では、OpenStack Networking のデプロイメントにおけるレイヤー 3 高可用性 (L3 HA) の役割について説明し、ネットワークの仮想ルーターを保護する実装手順を記載します。

21.1. 高可用性なしの OPENSTACK NETWORKING

高可用性機能が有効化されていない OpenStack Networking デプロイメントは、物理ノードの障害からの影響を受けやすくなります。

一般的なデプロイメントでは、テナントが仮想ルーターを作成します。この仮想ルーターは、物理 L3 エージェントノードで実行されるようにスケジューリングされます。L3 エージェントノードがなくなると、そのノードに依存していた仮想マシンは外部ネットワークと接続できなくなり、Floating IP アドレスも利用できなくなります。また、そのルーターがホストするネットワーク間の接続も失われます。

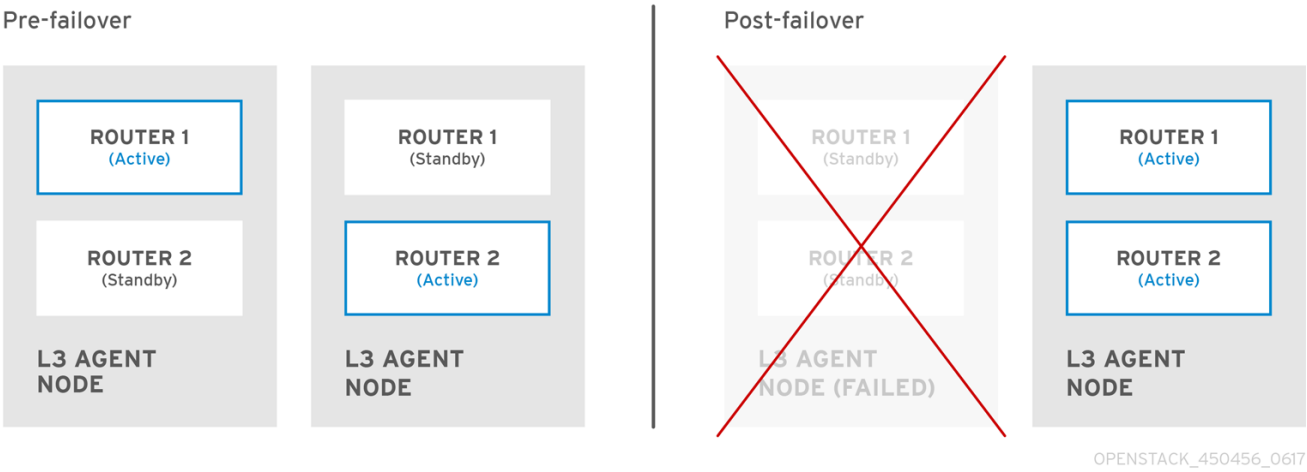
21.2. レイヤー 3 高可用性の概要

この active/passive の高可用性設定は、業界標準の VRRP (RFC 3768 で定義) を使用してテナントルーターと Floating IP アドレスを保護します。ノードの 1 つを **active**、残りを **standby** ロールとして機能するように指定することで、仮想ルーターは複数の OpenStack Networking ノードの間で無作為にスケジュールされます。

注記

レイヤー 3 高可用性を正常にデプロイメントするには、冗長化された OpenStack Networking ノードにおいて、Floating IP 範囲や外部ネットワークへのアクセスなど、同様の設定を維持する必要があります。

以下の図では、アクティブなルーター 1 とルーター 2 が別個の物理 L3 エージェントノード上で稼働しています。レイヤー 3 高可用性には、対応するノードの仮想ルーターのバックアップがスケジュールされており、物理ノードに障害が発生した場合にサービスを再開する準備が整っています。L3 エージェントノードが失敗すると、レイヤー 3 高可用性は影響を受けた仮想ルーターと Floating IP アドレスを稼働中のノードに再スケジュールします。



フェイルオーバーのイベント時には、Floating IP 経由のインスタンスの TCP セッションは影響を受けず、中断なしで新しい L3 ノードに移行されます。SNAT トラフィックのみがフェイルオーバーイベントからの影響を受けます。

active/active HA モードの場合には、L3 エージェント自体はさらに保護されます。

21.2.1. フェイルオーバーの状況

レイヤー 3 高可用性は、以下のイベントにおいて保護リソースのリスケジュールを自動的に行います。

- ハードウェアの障害が原因で L3 エージェントノードがシャットダウンするか、停電した場合
- L3 エージェントノードは、物理ネットワークから分離され、接続が切断された場合

注記

L3 エージェントサービスを手動で停止しても、フェイルオーバーのイベントが開始されるわけではありません。

21.3. テナントに関する留意事項

レイヤー 3 高可用性設定はバックエンドで行われており、テナントには表示されません。通常通り、仮想ルーターを継続して作成/管理することができますが、レイヤー 3 の高可用性の実装を設計する場合に認識しておく必要のある制限事項があります。

- レイヤー 3 高可用性は、テナントごとに仮想ルーター 255 個までサポートします。
- 内部の VRRP メッセージは、個別の内部ネットワーク内でトランスポートされ、プロジェクトごとに自動的にこれらのメッセージが作成されます。このプロセスは、ユーザーに透過的に行われます。

21.4. 背景の変更

ルーターの作成時に、管理者が **--ha=True/False** を設定できるように、Neutron API が更新されました。これは、neutron.conf の l3_ha の設定 (デフォルト) を上書きします。必要な設定手順については、次の項を参照してください。

21.4.1. neutron-server への変更

- レイヤー 3 高可用性は、OpenStack Networking で使用されるスケジューラー (無作為または leastrouter のスケジューラー) に関わらず、無作為にアクティブなロールを割り当てます。
- 仮想ルーターへの仮想 IP の確保を処理するために、データベーススキーマが変更されます。
- 上記のようにレイヤー 3 高可用性トラフィックを転送するために、トラフィックネットワークが作成されます。

21.4.2. L3 エージェントへの変更

- 新しい keepalived のマネージャーが追加され、負荷分散と HA 機能が提供されるようになりました。
- IP アドレスが仮想 IP に変換されます。

21.5. 設定手順

以下の手順では、OpenStack Networking と L3 エージェントノード上でレイヤー 3 高可用性を有効化します。

21.5.1. OpenStack Networking ノードの設定

1. neutron.conf ファイルで L3 HA を有効化し、各仮想ルーターを保護する L3 エージェントノード数を定義して、レイヤー 3 高可用性を設定します。

```
l3_ha = True
max_l3_agents_per_router = 2
min_l3_agents_per_router = 2
```

これらの設定については、以下で説明します。

- **l3_ha**: True に設定されると、これ以降に作成された仮想ルーターはすべて、(レガシーではなく) HA にデフォルト設定されます。管理者は、以下を使用して各ルーターの値を上書きすることができます。

```
# neutron router-create --ha=<True | False> routerName
```

- **max_l3_agents_per_router**: このオプションは、デプロイメント内にあるネットワークノードの合計数と最小数の間の値に設定します。たとえば、OpenStack Networking ノードを 4 つデプロイして、最大値を 2 に設定した場合には、L3 エージェント 2 つのみが各 HA 仮想ルーター (1 つは active、もう 1 つは standby) を保護することになります。さらに、新規の L3 エージェントノードがデプロイされるたびに、max_l3_agents_per_router の上限に達するまで、standby バージョンの仮想ルーターが追加でスケジュールされます。これにより、新規 L3 エージェントを追加することで、standby ルーターの数をスケールアウトすることができます。

- **min_l3_agents_per_router**: 最小値を設定することで、HA ルールが強制された状態に保つことができます。この設定は仮想ルーターの作成時に検証され、HA を提供するのに十分な数の L3 エージェントノードが利用できるようにします。HA ルーターの作成時には少なくともこの最小値で指定した数のアクティブな L3 エージェントが必要であるため、たとえば、ネットワークノードが 2 つあり、1 つが利用できなくなった場合、その間は新しいルーターを作成できません。

2. neutron-server サービスを再起動して変更を適用します。

```
# systemctl restart neutron-server.service
```

21.5.2. 設定の確認

仮想ルーターの名前空間内で ip address のコマンドを実行すると、プレフィックスとして **ha-** が指定された HA デバイスが結果内に返されます。

```
# 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
```

レイヤー 3 高可用性が有効化され、個別のノードで障害が発生した場合に、仮想ルーターと Floating IP アドレスが保護されます。

第22章 仮想デバイスの識別に対するタグの使用

複数のネットワークインターフェースまたはブロックデバイスを使用してインスタンスを起動している場合には、デバイスのタグ付け機能を使用して各デバイスの目的のロールとインスタンスのオペレーティングシステムを通信させることができます。インスタンスの起動時に、デバイスにタグが割り当てられ、メタデータ API とコンフィグドライブ (有効な場合) を使用してインスタンスのオペレーティングシステムに公開されます。

タグは、以下のパラメーターを使用して設定されます。

- **--block-device tag=device metadata**
- **--nic tag=device metadata**

たとえば、以下のコマンドは、**--block-device** および **--nic** のタグパラメーターを使用してインスタンスを作成します。

```
$ 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
```

割り当てらるタグが既存のインスタンスのメタデータに追加され、メタデータ API とコンフィグドライブ上の両方に公開されます。上記の例では、以下の **devices** セクションがメタデータに表示されます。

- **meta_data.json** の例:

```
  {
 "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"]
  }
 ]
}
```

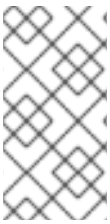
デバイスタグのメタデータは、メタデータ API から **GET /openstack/latest/meta_data.json** として確認できます。コンフィグドライブが有効で、インスタンスのオペレーティングシステムの **/configdrive** にマウントされている場合には、このメタデータは **/configdrive/openstack/latest/meta_data.json** にも表示されます。

第23章 レイヤー 2 ゲートウェイの使用

23.1. 概要

本章では、Red Hat OpenStack Platform を OpenDaylight と共に使用して、SDN オーバーレイソリューションを作成する方法について説明します。このソリューションにより、デプロイメントはベアメタルワークロード、レガシー VLAN ドメイン、SR-IOV 対応ワークロード、仮想化ワークロードの間でシームレスに接続することができます。

この接続は、物理 ToR (top-of-rack) スイッチ (**コア** スイッチではなく、**アクセス** スイッチとしても知られる) 上に VXLAN ゲートウェイをデプロイすることによって提供されます。ToR スイッチは、VXLAN カプセル化と HWVTEP スキーマをサポートしている必要があります。ネットワークの切り替えは、neutron のマルチセグメントネットワークと L2GW サービスプラグインを使用して実行されます。Southbound の設定は、OVSDB プロトコルと **hardware_vtep** スキーマを実装する HWVTEP プラグインを使用して実装されます。



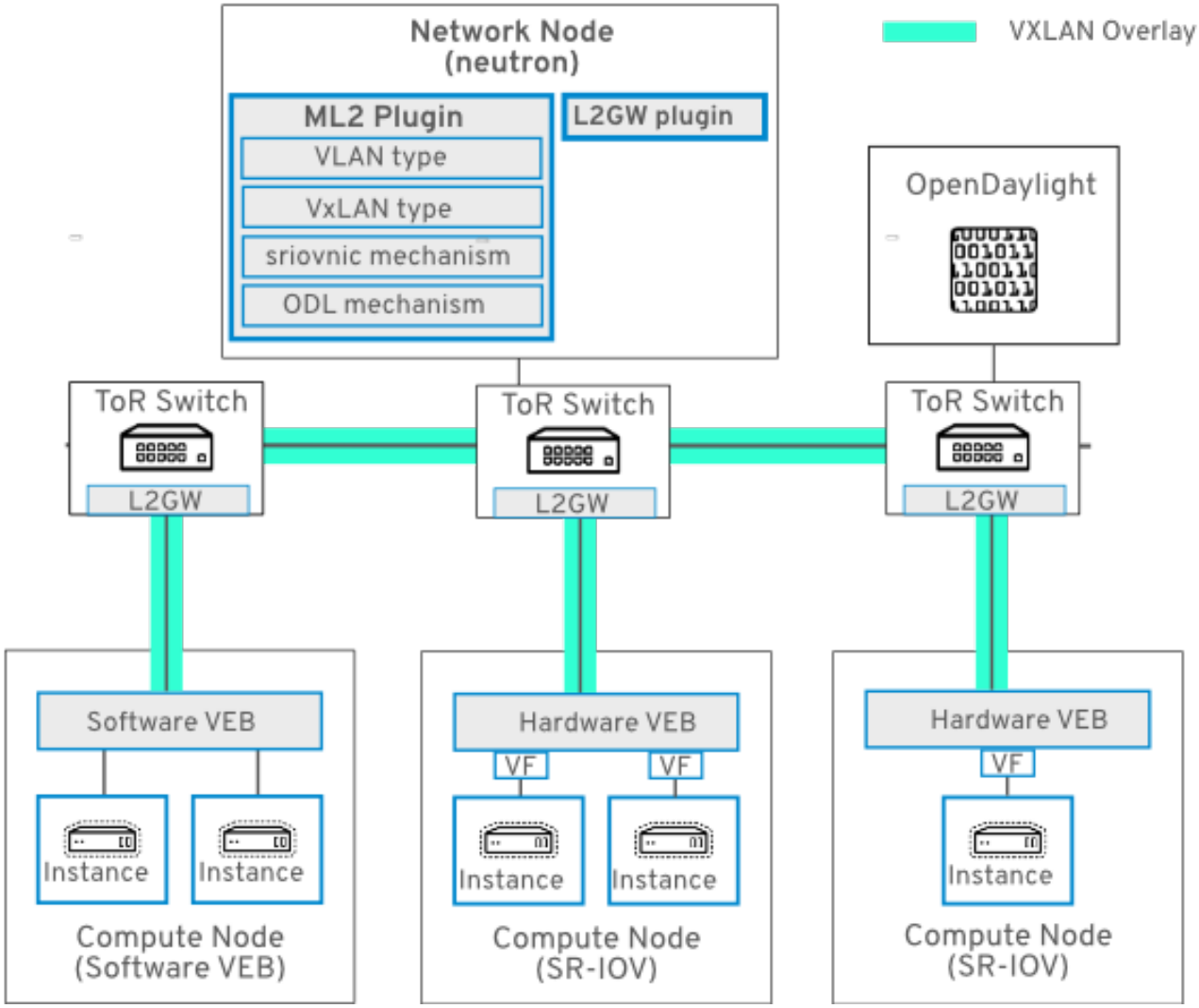
注記

マルチセグメントのネットワークとは、複数のセグメント ID をトランキングするように設定されたネットワークです。このネットワークは、1 つは VLAN、もう 1 は VXLAN の複数のセグメントがあると見なされます。これにより、カプセル化された VXLAN トラフィックはコンピュートノードと ToR スイッチの間で切り替えられます。

本ガイドでは、一連の L2GW ユースケースについて説明し、コンピュートノードで SR-IOV 設定は必須となっています。

23.1.1. ネットワークトポロジー

以下の図は、SDN オーバーレイにおける SR-IOV 接続を示しています。



23.1.2. 要件

- director ベースのデプロイには、OpenDaylight ロールを有効にする必要があります。『Red Hat OpenDaylight Installation and Configuration Guide』を参照してください。
- ユースケース 1 から 6 では、コンピュートノードで SR-IOV を有効化する必要があります。director ベースのデプロイメントの場合は、『Network Functions Virtualization Planning and Configuration Guide』の「Planning an SR-IOV deployment」を参照してください。

23.2. ユースケース 1: 1 台の SR-IOV コンピュートノード上に 2 つのインスタンスがある場合

このユースケースは、SR-IOV を有効化した同じコンピュートノード上に 2 つのインスタンスがあります。このシナリオでは、VXLAN トンネルが ToR からコントローラーノードに通ります。

1. nova がインスタンスを起動する時には、その仮想マシンの DHCP リクエストは SR-IOV NIC を出て、ToR 上の eth2 で入ります。
2. ToR スイッチ (**hardware_vtep** スキーマで OVSDB を実行) は、DHCP パケットを VXLAN トンネル内にカプセル化します。
3. DHCP パケットは、TOR とコントローラーノード間で VXLAN トンネルを介して送信されます。

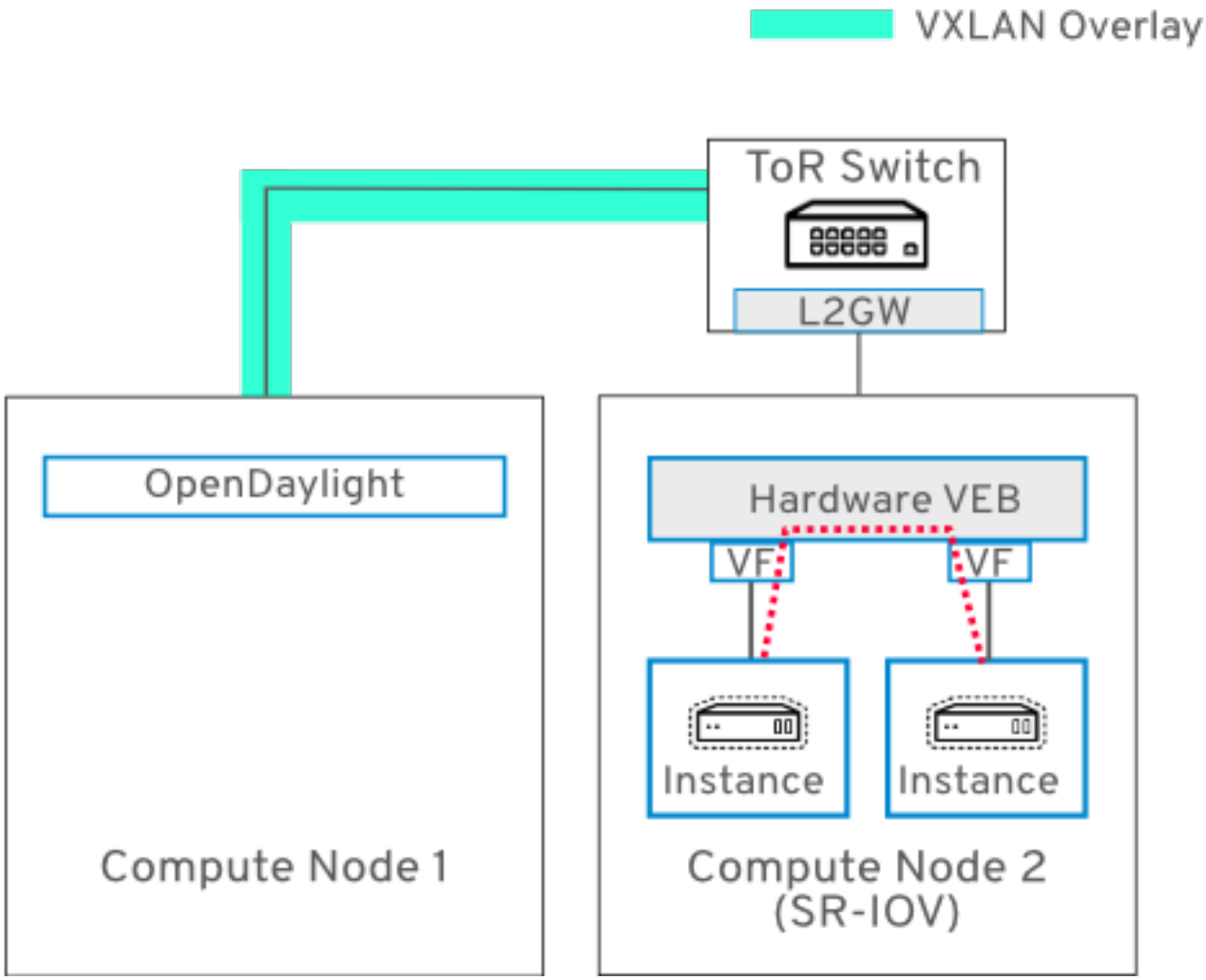
4. DHCP パケットはコントローラーノード上の VXLAN トンネルを出て、VXLAN ヘッダーが削除された後に、ネイティブの DHCP パケットが処理されて、コントローラーノードから DHCP 応答が送信されます。

5. VXLAN ヘッダーが DHCP 応答に追加され、VXLAN トンネルを介してコントローラーノードから ToR スイッチに送られます。

6. ToR では、VXLAN ヘッダーが削除され、VLAN タグが DHCP パケットに追加されます。

7. DHCP パケット (VLAN タグ付き) は ToR を介して送信され、コンピュートノード 2 で受信されます。ここで、DHCP 応答が受信され、インスタンスによって処理されます。

これで各インスタンスに IP アドレスが割り当てられ、コンピュートノード上の **VM1** と **VM2** の間の IP パケットは Intel **82599ES** NIC によりローカルでスイッチされます。この例で使用している NIC は、ローカルスイッチングをサポートしている Intel 82599ES なので、VM1 と VM2 の間の IP パケットは NIC から外に送信されません。ローカルスイッチングをサポートしていないベンダーの NIC の場合は、VM1 から VM2 へのパケットは ToR によってスイッチされます。

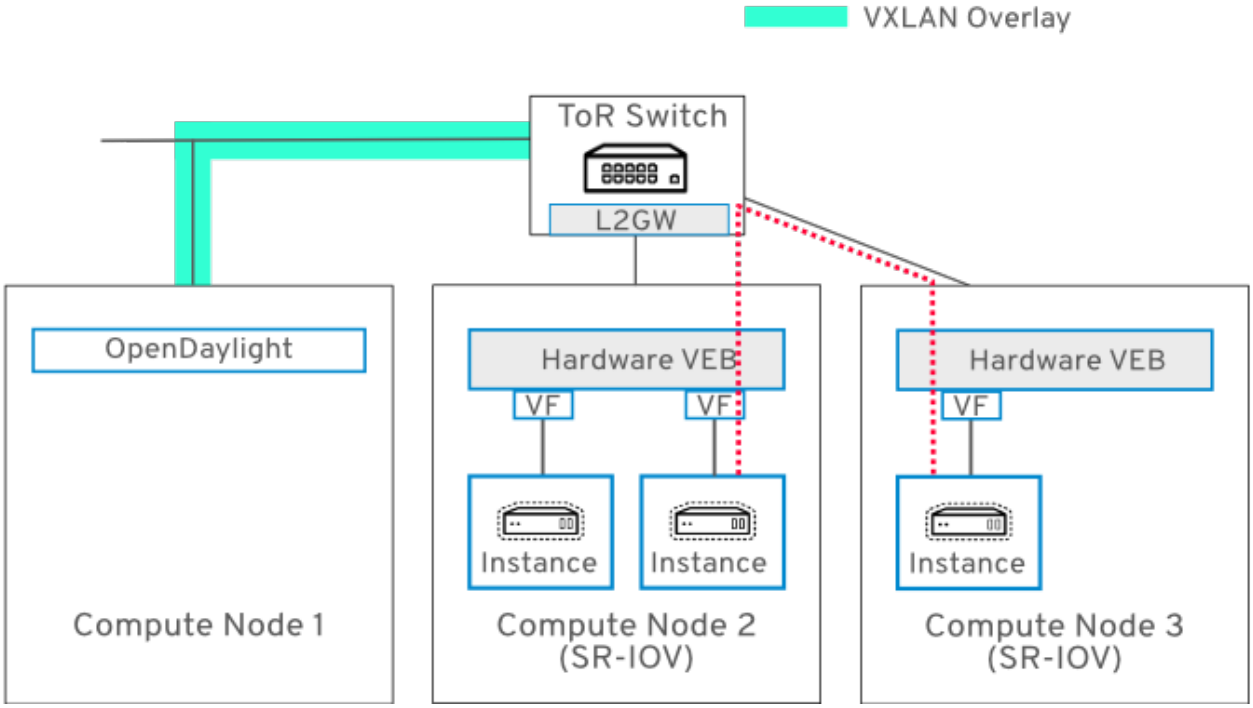


- **ノード 1**:
  - Red Hat Enterprise Linux 7
  - OpenStack コントローラーノード
  - OpenStack コンピュートノード
  - OpenDaylight ロール

- ノード 2:
  - Red Hat Enterprise Linux 7
  - OpenStack コンピュートノード
  - SR-IOV が NIC **em1** で有効化
  - ネットワークタグ: **2901**

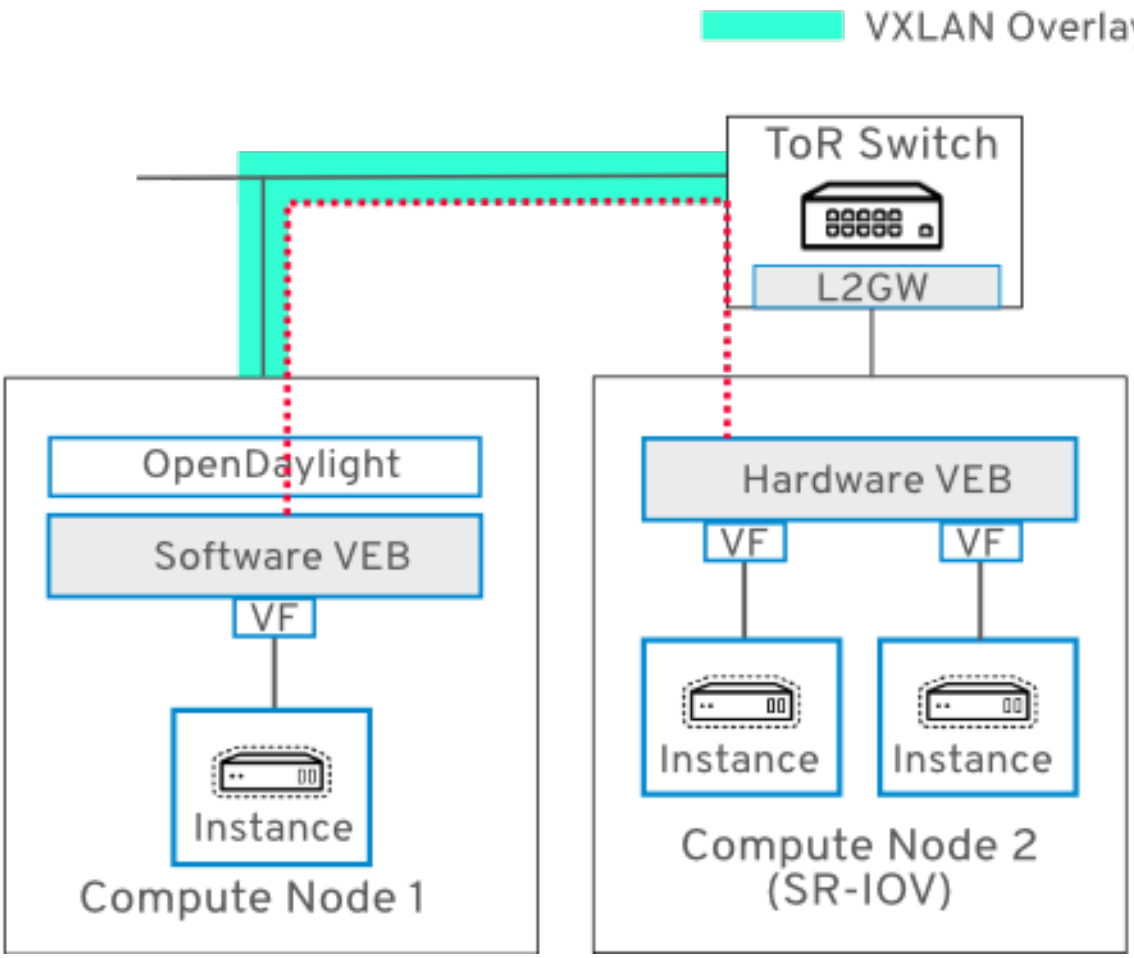
23.3. ユースケース 2: 別のコンピュートノード (複数) 上にインスタンスがある場合

このユースケースは **ユースケース 1** と似ており、DHCP は同じように機能します。異なるのは、**VM1** (コンピュートノード 1 上) と **VM2** (コンピュートノード 2 上) の間の IP 転送が ToR スイッチによって実行される点です。これは、赤の破線で示しています。



23.4. ユースケース 3: ハードウェア VEB 上のインスタンスにソフトウェア VEB 上のインスタンスが接続する場合

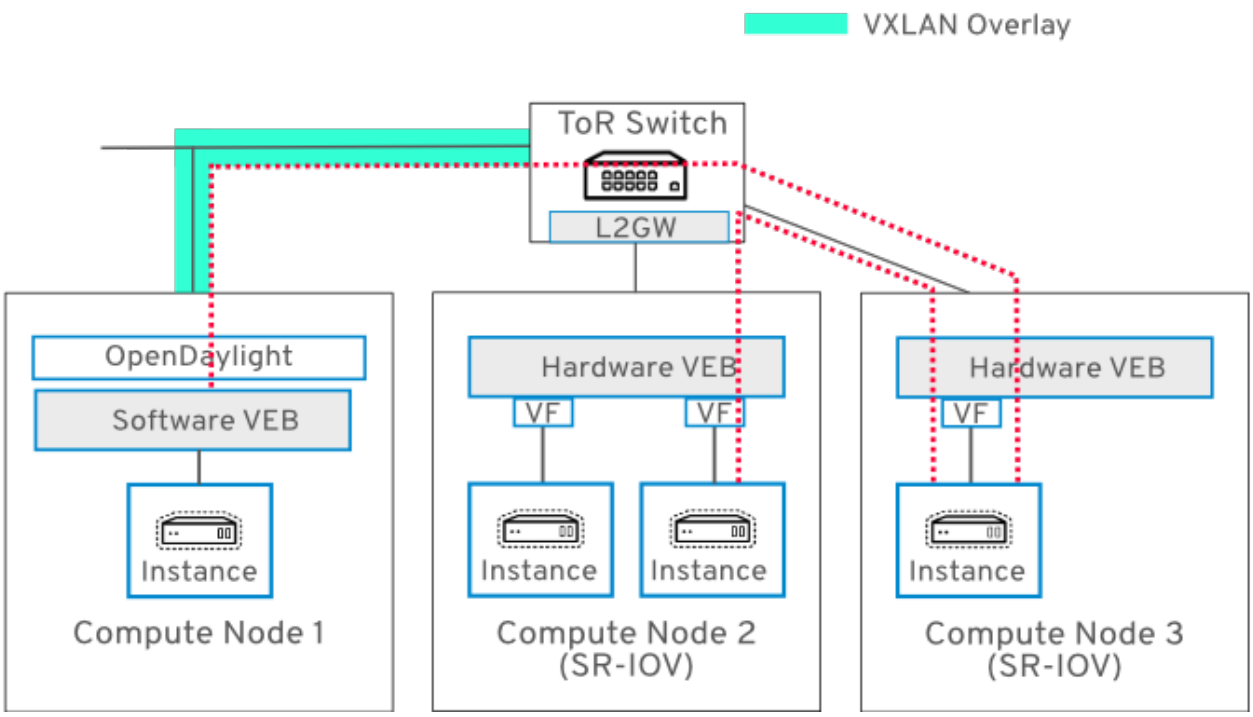
このユースケースでは、ソフトウェア VEB (Virtual Ethernet Bridge) とハードウェア VEB にアタッチされたインスタンス間で接続を確立しします。



- ノード 1:
  - Red Hat Enterprise Linux 7
  - OpenStack コントローラーノード
  - OpenStack コンピュートノード
  - OpenDaylight ロール
- ノード 2:
  - Red Hat Enterprise Linux 7
  - OpenStack コンピュートノード
  - SR-IOV が NIC **em1** で有効化
  - ネットワークタグ: **2900**

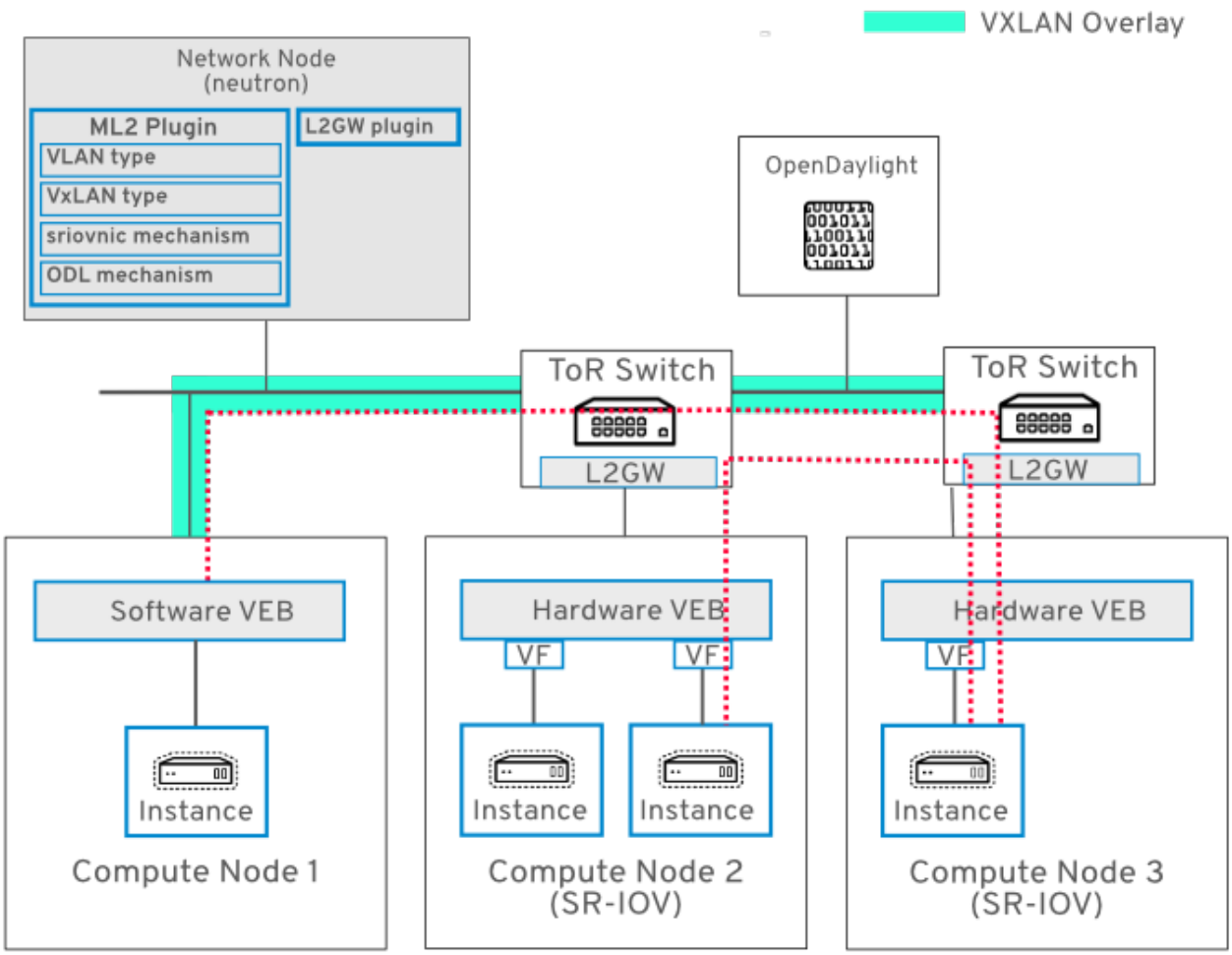
23.5. ユースケース 4: SR-IOV PHYSICAL FUNCTION (PF) に接続されたインスタンスの場合

このユースケースでは、**VM2** (コンピュートノード 2 上) の受信および送信トラフィックは、ToR スイッチを介して、**VM2** にアタッチされた SR-IOV PF をトラバースします。



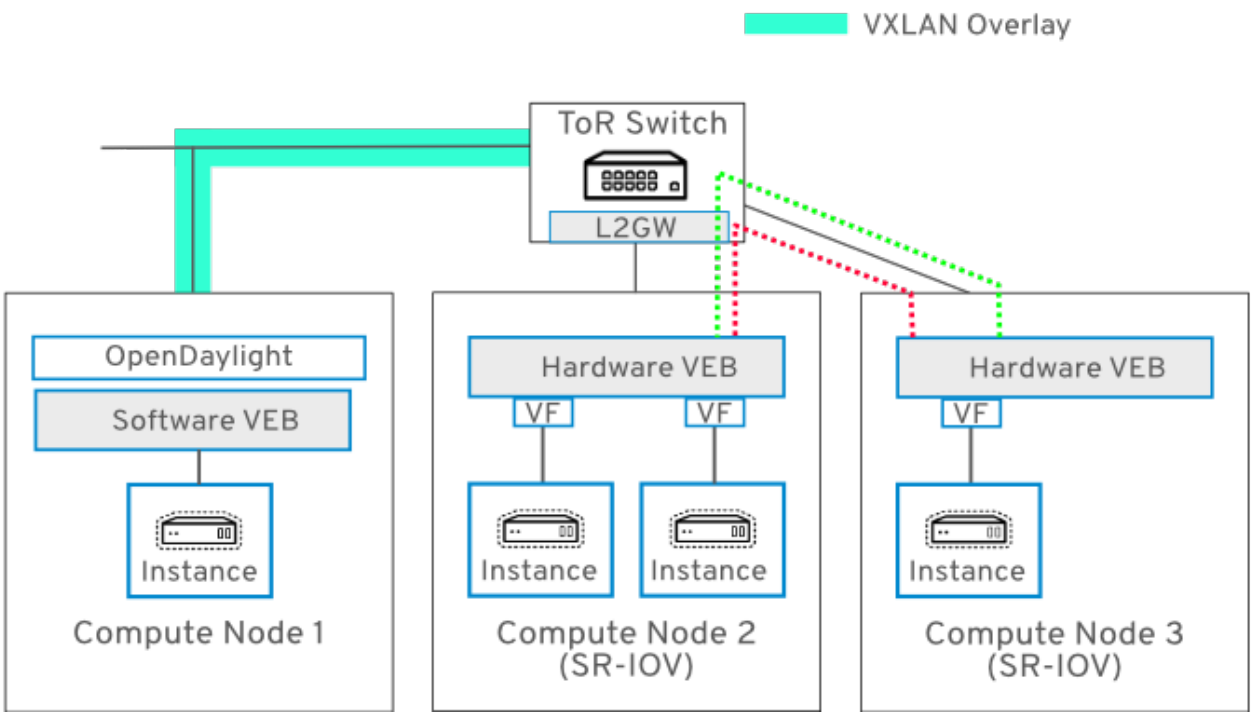
23.6. ユースケース 5: TOR スイッチが 2 つある場合

このユースケースでは、3 台のコンピュートノードが 2 つの異なる ToR スイッチに接続されています。**multinet** という名前の単一の neutron ネットワークが 3 台のノードをすべて結んでおり、インスタンスは同じ論理レイヤー 2 ネットワーク上で相互に接続することができます。



23.7. ユースケース 6 : 同じインターフェースを共有する異なる複数のネットワークに接続されたインスタンスの場合

このユースケースでは、**VM1** および **VM2** は別々のコンピュートノード上にあり、同じ neutron ネットワークを共有しています。**VM3** と **VM4** も別々のノード上にあり、同じ neutron ネットワークを共有しています。両方の neutron ネットワークのトラフィックは各ノード上の同じ物理 NIC を通過します。



- VM1 と VM2:
  - 接続されている neutron ネットワーク: **multinet**
  - VXLAN VNI: **1500**
  - VLAN タグ: **2901**
- VM3 と VM4:
  - 接続されている neutron ネットワーク: **multinet1**
  - VXLAN VNI: **1501**
  - VLAN タグ: **2902**

23.8. SDN トポロジーの構築

L2GW VXLAN トンネルのスイッチの設定および準備のための設定手順は、各 ToR ベンダースイッチ固有である可能性が高いので、以下に記載するステップの設定コマンドは、スイッチベンダーのマニュアルを参照して確認してください。

各 ToR スイッチには以下が必要となります:

- OVSDB の有効化と設定
- データネットワーク上の ODL コントローラーへの IP 接続。これは、VXLAN トンネルのトランスポートに必要です。
- 管理ネットワーク上の ODL コントローラーへの IP 接続。これは、OVSDB の制御メッセージに必要です。

23.9. OPENDAYLIGHT の設定

- director ベースのデプロイには、OpenDaylight ロールを有効にする必要があります。『Red Hat OpenDaylight Installation and Configuration Guide』を参照してください。

OpenDaylight (ODL) のデプロイが完了したら、必要なトランスポートゾーンを作成する必要があります。

23.9.1. トランスポートゾーンの設定

本項では OpenDaylight 用のトランスポートゾーンを作成します。以下の例で使用している IP アドレスは、お使いのデプロイメントに適した IP アドレスに変更する必要があります。

1. トランスポートゾーンを作成します。以下に例を示します。

```
URL:
 http://${ODL-IP}:8181/restconf/config/itm:transport-zones/
JSON:
{
    "transport-zone": [
        {
            "zone-name": "TZA",
            "subnets": [
                {
                    "prefix": "192.168.254.0/24",
                    "vlan-id": 0,
                    "vteps": [
                        {
                            "dpn-id": 95311090836804,
                            "portname": "eth1",
                            "ip-address": "192.168.254.31"
                        }
                    ],
                    "gateway-ip": "0.0.0.0"
                }
            ],
            "tunnel-type": "odl-interface:tunnel-type-vxlan"
        }
    ]
}
```

2. VXLAN トンネルネットワーク (VTEP)の一部となるコントローラーノードとコンピュートノードをトランスポートゾーンに追加します。そのためには、デバイスの **dpnid** が必要です。これは、以下のように取得することができます。

```
$ curl -s -u admin:admin -X GET http://10.8.125.240:8181/restconf/operational/odl-interface-meta:bridge-ref-info/
{
  "bridge-ref-info": {
    "bridge-ref-entry": [
      {
        "bridge-reference": "/network-topology:network-topology/network-topology:topology[network-topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-id='ovsdb://uuid/2647ac59-281f-4968-beb1-7a7d64990f19/bridge/br-int']",
        "dpid": 181136841129213
      }
```

```
        ]
    }
}
```

a. **sudo ovs-vsctl show** を実行して **ovsdb** UUID を取得します。これは、上記の **curl** コマンドでデバイスの **dpnid** を特定するのに必要です。

```
$ sudo ovs-vsctl show
2647ac59-281f-4968-beb1-7a7d64990f19
  Manager "tcp:10.8.125.240:6640"
    is_connected: true
  Bridge br-int
```

b. bridge-ref **curl** コマンドを使用して取得した **dpn-id** を指定して、デバイスをトランスポートゾーンに追加します。これには、**Postman** を使用することができます。もう 1 つの方法として、代わりに **curl** を使用することができます。この例では、IP アドレスが 192.168.254.31 のリモート VTEP を追加します。リモート VTEP のトンネルインターフェースは **eth1** で、リモート VTEP の **dpn-id** は以下に示したとおりです。OpenDaylight が VLAN ベースのネットワーク上でリッスンしている場合には、その **vlan-id** を指定する必要があります。

```
$ curl -X POST -H "Authorization: Basic YWRtaW46YWRtaW4=" -H "Content-Type:
application/json" -H "Cache-Control: no-cache" -H "Postman-Token: ae2ebaee-8813-
79df-7114-0bc786287793" -d '{
      "transport-zone": [
         {
            "zone-name": "zone1",
            "subnets": [
               {
                  "prefix": "192.168.254.0/24",
                  "vlan-id": 0,
                  "vteps": [
                     {
                        "dpn-id": 181136841129213,
                        "portname": "eth1",
                        "ip-address": "192.168.254.31"
                     }
                  ],
                  "gateway-ip": "0.0.0.0"
               }
            ],
            "tunnel-type": "odl-interface:tunnel-type-vxlan"
         }
      ]
}
' "http://10.8.125.240:8181/restconf/config/itm:transport-zones/"
```

その結果、出力される一覧の各 VTEP には、以下のような属性が表示されます。

- **ip-address**: リモートの VTEP IP アドレス
- **portname**: トンネルインターフェースのポート名
- **dpn-id**: 上記のステップで取得した値

3. この例の **curl** コマンドは、**zone1** という名前のトランスポートゾーンを作成します。

```
curl -X POST -H "Authorization: Basic YWRtaW46YWRtaW4=" -H "Content-Type:
application/json" -H "Cache-Control: no-cache" -H "Postman-Token: ae2ebaee-8813-79df-
7114-0bc786287793" -d '{
      "transport-zone": [
         {
            "zone-name": "zone1",
            "subnets": [
               {
                  "prefix": "192.0.2.0/24",
                  "vlan-id": 0,
                  "vteps": [
                     {
                        "dpn-id": 92455225558023,
                        "portname": "eth3",
                        "ip-address": "192.0.2.6"
                     }
                  ],
                  "gateway-ip": "0.0.0.0"
               }
            ],
            "tunnel-type": "odl-interface:tunnel-type-vxlan"
         }
      ]
}
' "http://10.8.125.240:8181/restconf/config/itm:transport-zones/"
```

4. ToR スイッチが OpenDaylight ノードを VTEP マネージャーとして使用するように設定します。設定コマンドと構文については、ToR スイッチベンダーのマニュアルを参照してください。

23.10. OPENSTACK ネットワークの作成

1. 2 つのネットワークセグメントで構成される **multinet** という名前のネットワークを作成します。1 番目のセグメントは VLAN タイプで、仮想マシンにアタッチされる SRIOV Virtual Function の VLAN タグの設定に使用されます。2 番目のセグメントは VXLAN タイプで、ToR スイッチから ODL コントローラーノードおよびコンピュートノードへの VXLAN トンネルに使用されます。

```
$ neutron net-create multinet --segments type=dict list=true
provider:physical_network=",provider:segmentation_id=1500,provider:network_type=vxlan
provider:physical_network=physnet_sriov,provider:segmentation_id=2201,provider:network_typ
e=vlan
```

2. **multinet** ネットワーク用のサブネットを作成します。

```
$ neutron subnet-create multinet --allocation-pool start=10.100.5.2,end=10.100.5.254 --
name mn-subnet --dns-nameserver 8.8.8.8 10.100.5.0/24
```

3. 適切な SR-IOV ドライバー (例: Intel **ixgbe** ドライバー) を含む Red Hat Enterprise Linux 7 イメージを作成します。完了したら、イメージを glance にインポートします。

```
$ glance image-create --name rhel7 --disk-format qcow2 --container-format bare --visibility
public --file /opt/images/rhel7.qcow2
```

4. L2 ゲートウェイを作成します。以下のコマンドは、**gw1** という名前の L2GW を作成し、アクセスポートは **eth2** と **eth3** です。

> 注記
>
> ポート更新の API には、現在いくつかの制約があるため、このコマンドに全ゲートウェイポートを含めることを考慮してください。

```
$ neutron l2-gateway-create gw1 --tenant_id $(openstack project list | grep '\sadmin' | awk
'{print $2}')   --device name=hwvtep,interface_names="eth1;eth2"
```

- **name**: ToR スイッチの設定に一致する必要があります。これは、**hardware_vtep** スキーマの **Physical_Switch** テーブル内で **name** として定義されています。ToR スイッチベンダーのマニュアルを参照してください。

5. L2 ゲートウェイの接続を作成します。これにより、ToR スイッチと、トランスポートゾーンで定義されている VTEP との間のトンネルが確立します。

```
$ neutron l2-gateway-connection-create gw1 multinet --default-segmentation-id 2201
$ neutron l2-gateway-connection-create gw1 multinet1 --default-segmentation-id 2203
```

   a. 以下に示す L2GW 作成の代替方法に注意してください。この例は、VLAN タグをサポートしない SR-IOV PF を設定する場合に役立ちます。この例では、**eth1** にはタグがなく、**eth2** には **2201** タグが付いています。

```
$ neutron l2-gateway-create gw1 --tenant_id $(openstack project list | grep '\sadmin' |
awk '{print $2}')   --device name=hwvtep,interface_names="eth1|0;eth2|2201"
$ neutron l2-gateway-connection-create gw1 multinet
```

6. コンピュートノード上に SR-IOV VF ポートを作成します。

```
$ port1_id=$(neutron port-create multinet --name sriov_port1 --binding:vnic_type direct --
device-owner nova-compute | grep "\ id\ " | awk '{ print $4 }')

$ port2_id=$(neutron port-create multinet --name sriov_port2 --binding:vnic_type direct --
device-owner nova-compute | grep "\ id\ " | awk '{ print $4 }')

$ port3_id=$(neutron port-create multinet --name sriov_port3 --binding:vnic_type direct --
device-owner nova-compute | grep "\ id\ " | awk '{ print $4 }')
```

7. コンピュートノード上に **direct-physical** SR-IOV PF ポートを作成します。

```
$ port2_id=$(neutron port-create multinet --name sriov_port2 --binding:vnic_type direct-
physical --device-owner nova-compute | grep "\ id\ " | awk '{ print $4 }')
```

8. コンピュートノード上にインスタンスを作成します。これらは、前のステップで作成した SR-IOV ポートに接続されます。

注記

イメージ内のゲスト OS が Network Manager を使用している場合には、DHCP は、SR-IOV VF インターフェースでのみ機能します。Red Hat Enterprise Linux の場合は、イメージに対して **virt-customize** を使用して、**ifcfg-eth0** をシステムが使用するインターフェースの名前 (例: **ens4**) に変更する必要があります。

```
$ nova boot --poll --flavor m1.small --image rhel7 --nic port-id=$port1_id --key-name
admin_key sriov-vm1

$ nova boot --poll --flavor m1.small --image rhel7 --nic port-id=$port3_id --key-name
admin_key sriov-vm3

$ nova boot --poll --flavor m1.small --image rhel7 --nic port-id=$port2_id --key-name
admin_key sriov-vm2

$ nova boot --poll --flavor m1.small --image rhel7 --nic net-id=$(neutron net-list | grep -w
multinet | awk '{print $2}') --key-name admin_key vmvx01
```

注記

本項では、SR-IOV インターフェースを使用してインスタンスを相互接続する方法について説明しますが、L2GW neutron ドライバーを引き続き使用して基本的な接続を確立できる点に注意してください。これには、VXLAN ベースのプロジェクトネットワークの L2 ブロードキャストドメインを、外部の VLAN ベースのプロバイダーネットワークに拡張する必要があります。このユースケースでは、複数のプロバイダーネットワークを作成する必要はなく、標準の VXLAN ネットワークを使用します。これにより、外部のアプライアンスを使用して通常のインスタンスを接続することができます。

23.11. レビュー

上記のステップに従って作業を完了すると、インスタンスの 1 つに SSH 接続してから、他のコンピュートノード上でホストされているインスタンスに ping できるようになるはずです。その結果 ICMP トラフィックは OVS と ToR スイッチを、カプセル化されたレイヤー 2 トンネルとしてトラバースするようになります。