Red Hat OpenStack Platform 16.0

ネットワークガイド

Red Hat OpenStack Platform Networking の詳細ガイド

Last Updated: 2022-09-13

Red Hat OpenStack Platform 16.0 ネットワークガイド

Red Hat OpenStack Platform Networking の詳細ガイド

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

法律上の通知

概要

一般的な OpenStack Networking タスクのガイドです。

目次

前書き

OpenStack Networking サービス (コード名: **neutron**) は、Red Hat OpenStack Platform 16.0 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。

ソフトウェア定義ネットワーク (SDN)

ソフトウェア定義ネットワーク (SDN) を使用することで、ネットワーク管理者は下層の機能の抽象化によりネットワークサービスを管理することができます。サーバーのワークロードを仮想環境に移行しても、データの送受信のためにそれらのサーバーが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、接続先のサーバーと同様に、これらの物理ネットワークの概念が仮想化されていると考えるのに無理はないでしょう。

本ガイドの構成

- **前書き**: ソフトウェア定義ネットワーク (SDN) の定義について簡単に説明します。
- **パート I**: 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
  - プロジェクトネットワークのトラブルシューティング
- **パート II**: 高度な Red Hat OpenStack Platform 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。これには以下の項目が含まれます。
  - 仮想ルーターのレイヤー 3 高可用性の設定
  - DVR およびその他のネットワーク機能の設定

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。OSI ネットワークモデルでは、ケーブルがレイヤー 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。エンタープライズスイッチには複数のイーサネットポートがあり、追加のマシンを接続することができます。複数のマシンで構成されるネットワークは、ローカルエリアネットワーク (LAN) と呼ばれます。

複雑さが増すので、スイッチは OSI モデルの新たなレイヤー (レイヤー 2) に相当します。各 NIC には、ハードウェアごとに一意な MAC アドレス番号が割り当てられ、この番号を使用することにより、同じスイッチに接続された複数のマシンはお互いを認識することができます。スイッチは、どの MAC アドレスがどのポートに結線されているかのリストを管理するので、コンピューター間でデータ送信を試みる際に、スイッチはそれら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監視する CAM (Content Addressable Memory) のエントリーを調整します。

1.1.1. VLAN

VLAN を使用して、同じスイッチ上で動作してい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。つまり、それぞれ別のネットワークのメンバーとなるように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。この場合、それぞれのネットワークは、セキュリティー上の理由からトラフィックを分割するのに使用できる、小規模な LAN ということになります。

たとえば、スイッチに合計 24 個のポートがある場合に、ポート 1 - 6 を VLAN200 に、ポート 7 - 18 を VLAN201 に、それぞれ割り当てることができます。その結果、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離され、直接通信することはできなくなります。通信する必要があれば、スイッチの VLAN200 部分と VLAN201 部分が 2 つの別個の物理スイッチであったかのように、トラフィックはルーターを通過する必要があります。相互に通信が可能な VLAN の組み合わせを制御するには、ファイアウォールも有用です。

1.2. 2 つの LAN の接続

2 つの LAN がそれぞれ別個のスイッチ上で稼働している状況で、LAN 間で情報を共有させたいとします。このような通信を可能にする設定としては、以下の 2 つのオプションがあります。

- **802.1Q VLAN タグ付けを使用して、両方の物理スイッチにまたがる単一の VLAN を設定する。**
  ネットワークケーブルの一方の端を 1 つのスイッチのポートに接続し、他の端を別のスイッチのポートに接続し、続いてこれらのポートを 802.1Q タグ付けポート (トランクポートとも呼ばれる) として設定する必要があります。これら 2 つのスイッチが 1 つの大きな論理スイッチとして機能し、接続されているコンピューターが互いを認識することができます。

  このオプションの難点はスケーラビリティーです。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することのできるスイッチの数は限られています。

- **ルーターを用意し、ケーブルを使用して各スイッチに接続する。**
  ルーターは、両方のスイッチに設定されたネットワークを認識します。スイッチに結線した各ケーブル端には、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は、トラフィックの送付先マシンが送付元マシンと同じ

LAN 上にないことが明らかな場合の送付先を定義します。デフォルトゲートウェイを設置することで、送付先に関する具体的な情報が無くても、各コンピューターは他のコンピューターにトラフィックを送付することができます。それぞれのコンピューターはデフォルトゲートウェイにトラフィックを送付し、ルーターはトラフィックを受け取る送付先コンピューターを決定します。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。

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、全ネットワーク間を移動するトラフィックを制御します。そのために、ファイアウォールは、ネットワークを通過することのできるトラフィックを規定する事前定義済みのルールセットを参照します。これらのルールは粒度を細か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、暗号化された Web トラフィック (HTTPS) を一方向に送付している場合にのみ、木曜の午後に限り、**VLAN201** のコンピューターとだけ通信できるものとする」といった設定が可能です。

これらのルールを確実に適用するために、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、正当なパケット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に見せかけて、密かにデータを抜き出すことができます。DPI はこのような脅威を軽減する手段の 1 つです。

1.3. OPENSTACK NETWORKING (NEUTRON) の使用

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用にネットワークインフラストラクチャーを物理ネットワークに橋渡しすることができます。

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。

- **一般的な方法:**
  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。
  - 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:**
  サブネットマスクは、アクティブビットの合計に短縮されます。

192.168.100.0/24 を例に取ると、**/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の合計数を指します。

また、CIDR 形式は、**ifcfg-xxx** スクリプトにおいて **NETMASK** 値の代わりに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、インスタンスを発着する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

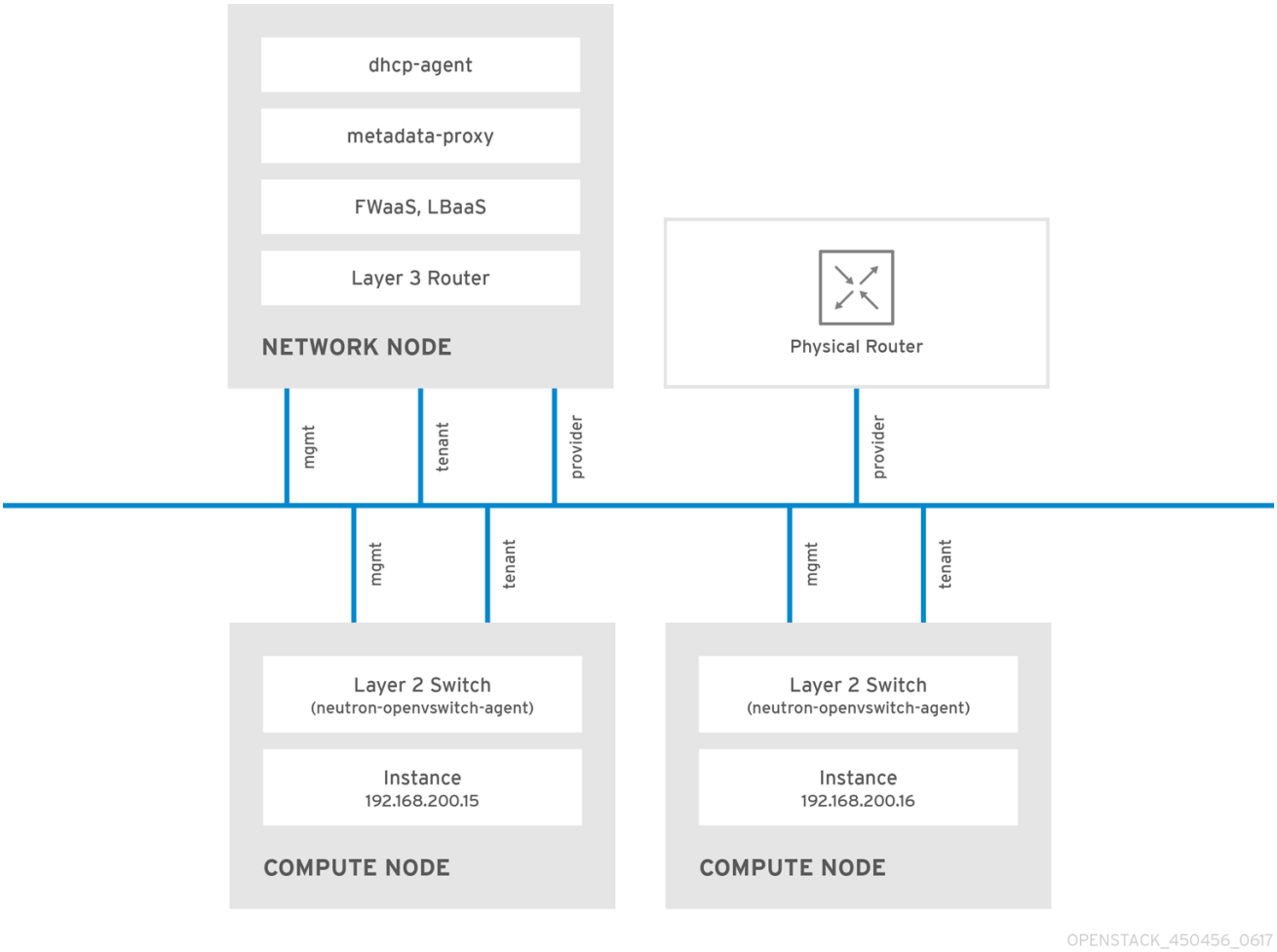
Red Hat OpenStack Platform 11 では、コンポーザブルロールのサポートが追加され、ネットワークサービスをカスタムロール別に分類することができます。ただし、本ガイドでは、内容をわかりやすくするために、デプロイメントにはデフォルトの Controller ロールを使用することを前提とします。

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

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。director のデプロイメントに関する詳細な情報は、『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 ノードがレイヤー 3 ルーティングと DHCP の機能を果たし、Firewall-as-a-Service (FWaaS) および Load Balancing-as-a-Service (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、それぞれにプロジェク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、neutron ポートが送受信す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です。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。

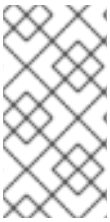
Floating IP アドレス、OpenStack Networking LBaaS の仮想 IP、およびインスタンスのために作成されたポートは、セキュリティーグループに割り当てられます。セキュリティーグループを指定しない場合には、ポートは default のセキュリティーグループに割り当てられます。デフォルトでは、このグループは全受信トラフィックをドロップし、全送信トラフィックを許可します。ただし、デフォルトのセキュリティーグループのメンバーであるインスタンス間ではトラフィックが流れます。グループはそれ自体をポイントするリモートグループ ID を持つためです。

デフォルトセキュリティーグループのフィルターの動作を変更するには、グループにセキュリティールールを追加するか、まったく新しいセキュリティーグループを作成します。

2.4. OPEN VSWITCH

Open vSwitch (OVS) 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 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。OVS と物理スイッチの統合には、STP、LACP、802.1Q VLAN タグ付け等のレイヤー 2 (L2) 機能が必要です。Open vSwitch のバージョン 1.11.0-1.el6 以降は、VXLAN および GRE を使用したトンネリングもサポートします。

ネットワークインターフェースのボンディングに関する詳細は、『オーバークラウドの高度なカスタマイズ』の「ネットワークインターフェースボンディング」の章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、OVS 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) によるネットワーク

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 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、ML2 メカニズムドライバーにより実装されるようになりました。

注記

ML2 はデフォルトの OpenStack Networking プラグインで、OVN 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 (OVS) プラグインを実行するデプロイは、OVS プラグインだけを使用する必要がありました。モノリシックなプラグインでは、linuxbridge 等の別のプラグインを同時に実行することはサポートされませんでした。この制約により、複数の異なる要件を伴う環境では、ニーズを満たすことが困難となっていました。

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に対する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応じて、ML2 は、以下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
- Geneve

ml2_conf.ini ファイルの ML2 セクションで、タイプドライバーを有効にします。以下に例を示します。

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

2.5.3. ML2 メカニズムドライバー

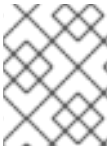
共通のコードベースを使用するメカニズムとして、プラグインが実装されるようになりました。このアプローチにより、コードの再利用が可能になる上、コードのメンテナンスとテストにおける複雑性が大幅に軽減されます。

注記

サポート対象のメカニズムドライバーの一覧は、『リリースノート』 を参照してください。

デフォルトのメカニズムドライバーは OVN です。**ml2_conf.ini** ファイルの ML2 セクションで、メカニズムドライバーを有効にします。以下に例を示します。

```
[ml2]
mechanism_drivers = ovn
```



注記

これらの設定は、Red Hat OpenStack Platform director により管理されます。手動で変更しないでください。

2.6. ML2 タイプドライバーとメカニズムドライバーの互換性

メカニズムドライバー	タイプドライバー				
	flat	gre	vlan	vxlan	geneve
ovn	互換	いいえ	はい	いいえ	互換
openvswitch	互換	はい	はい	はい	非互換

2.7. ML2/OVN メカニズムドライバーの制約

2.7.1. 本リリースでは、サポートされる ML2/OVS から ML2/OVN への移行方法はありません。

Red Hat OpenStack Platform (RHOSP) の本リリースでは、ML2/OVS メカニズムドライバーから ML2/OVN メカニズムドライバーへの移行はサポートされません。RHOSP の本リリースでは、OpenStack コミュニティーの移行ストラテジーはサポートされません。移行サポートは、RHOSP の今後のリリースで予定されています。

移行サポートの進捗を追跡するには、https://bugzilla.redhat.com/show_bug.cgi?id=1862888 を参照してください。

2.7.2. ML2/OVN ではまだサポートされていない ML2/OVS 機能

機能	備考	本機能の経緯
断片化 / ジャンボフレーム	OVN では、まだ ICMP「fragmentation needed」パケットの送信はサポートされません。断片化が必要な大きな ICMP/UDPパケットは、ML2/OVN では ML2/OVS ドライバー実装のようには機能しません。TCP トラフィックは、最大セグメントサイズ (MSS) クランプにより処理されます。	Bug 1547074 (ovn-network) Bug 1702331 (Core ovn)
ポート転送	OVN ではポート転送はサポートされません。	Bug 1654608 Port Forwarding API
セキュリティーグループプロギング API	ML2/OVN では、セキュリティーグループイベント (インスタンスが制限された操作の実行やリモートサーバーの制限されたポートへのアクセスを試みるケース) を記録するログファイルを利用することはできません。	Bug 1619266
マルチキャスト	統合ブリッジとして ML2/OVN を使用する場合には、マルチキャストトラフィックはブロードキャストトラフィックとして扱われます。 統合ブリッジは FLOW モードで動作します。したがって、IGMP スヌーピングを利用することはできません。この機能をサポートするためには、コア OVN が IGMP スヌーピングをサポートしている必要があります。	Bug 1672278
SR-IOV	現状、SR-IOV は neutron DHCP エージェントがデプロイされている場合に限り機能します。	Bug 1666684
OVN と DHCP の組み合わせでのベアメタルマシンのプロビジョニング	OVN 上の組み込み型 DHCP サーバーは、現状ベアメタルノードをプロビジョニングすることができません。プロビジョニングネットワーク用に、DHCP を提供することができません。iPXE のチェーンブートにはタグ付け (dnsmasq の **--dhcp-match**) が必要ですが、OVN DHCP サーバーではサポートされていません。	https://bugzilla.redhat.com/show_bug.cgi?id=1622154
OVS_DPDK	OVS_DPDK は、現在 OVN ではサポートされていません。	

2.8. デフォルトの ML2/OVN ドライバーに代わる ML2/OVS メカニズムドライバーの使用

お使いのアプリケーションに ML2/OVS メカニズムドライバーが必要な場合、環境ファイル **neutron-ovs.yaml** を使用してオーバークラウドをデプロイすることができます。これにより、デフォルトの ML2/OVN メカニズムドライバーが無効になり ML2/OVS が有効化されます。

2.8.1. 新規 RHOSP 16.0 デプロイメントでの ML2/OVS の使用

オーバークラウドのデプロイメントコマンドに、以下の例に示すように環境ファイル **neutron-ovs.yaml** を追加します。

```
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml
```

環境ファイルの使用に関する詳しい情報は、『 オーバークラウドの **高度なカスタマイズ**』 ガイドの「オーバークラウド作成時 の環境ファイルの追加」を参照してください。

2.8.2. 以前の RHOSP の ML2/OVS から RHOSP 16.0 の ML2/OVS へのアップグレード

ML2/OVS を使用する以前のバージョンの RHOSP からのアップグレード後も ML2/OVS を使用し続けるには、文書化されている Red Hat のアップグレード手順に従い、ML2/OVS から ML2/OVN への移行は実施しないでください。

アップグレード手順には、オーバークラウドデプロイメントコマンドへの **-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml** の追加が含まれます。

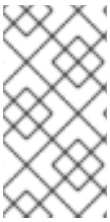
2.9. L2 POPULATION ドライバーの設定

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。

L2 Population ドライバーを有効にするには、以下の手順を実施します。

1.L2 Population ドライバーを有効にするには、メカニズムドライバーの一覧に追加します。また、少なくとも 1 つのトンネリングドライバーも有効にする必要があります (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,geneve
mechanism_drivers = l2population
```

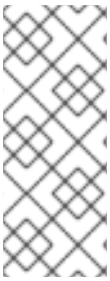


注記

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。一般的な用途の場合には、Red Hat では OpenStack Platform director のデフォルトである Open vSwitch (OVS) 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。その場合には、L2 エージェントが含まれる各ノードで有効にします。

```
[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** フラグを設定します。

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

2.10. 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。

2.10.1. 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。ネットワークの名前空間を使用して、各プロジェクトに独自の分離されたレイヤー 3 ルーターを提供します。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。neutron は、これらの IP アドレスを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ます。選択した IP 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、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。

2.10.2. DHCP エージェント

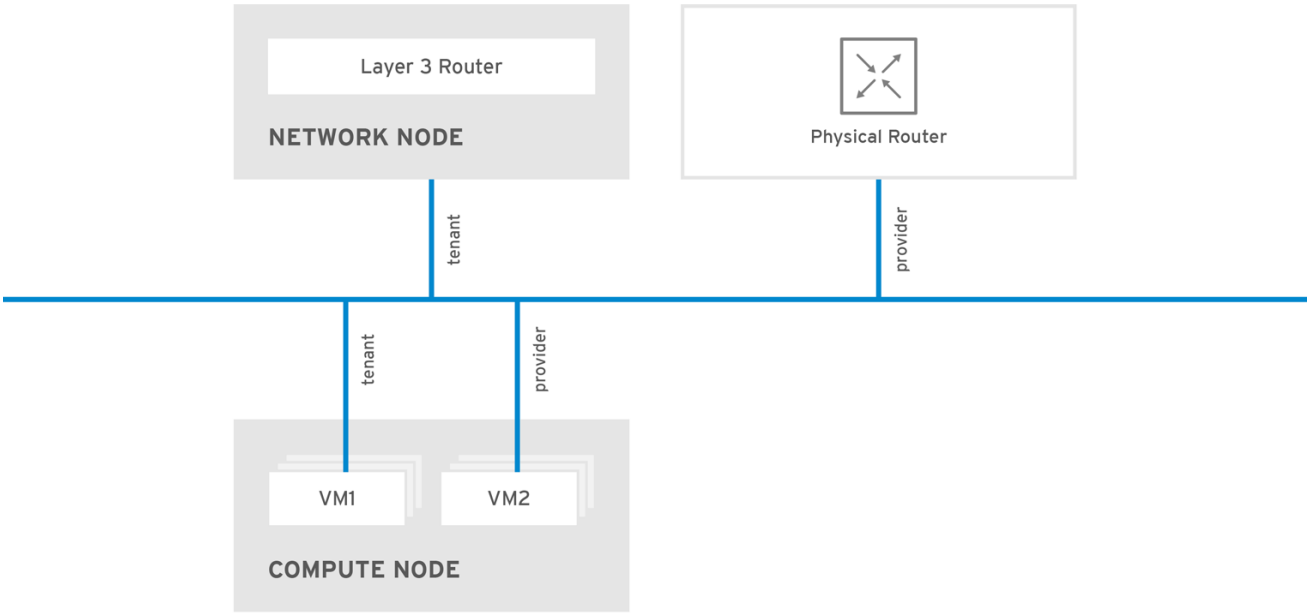
OpenStack Networking DHCP エージェントは、各プロジェクトのサブネットが DHCP サーバーとして機能するために作成されるネットワークの名前空間を管理します。各名前空間は、ネットワーク上の仮想マシンへの IP アドレス確保が可能な dnsmasq プロセスを実行します。サブネットの作成時にこのエージェントが有効化されて稼働している場合には、そのサブネットにはデフォルトで DHCP が有効化されます。

2.10.3. 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。

2.11. プロジェクトネットワークとプロバイダーネットワーク

以下の図には、プロジェクトネットワークおよびプロバイダーネットワーク種別の概要と、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。



2.11.1. プロジェクトネットワーク

ユーザーは、プロジェクト内の接続のためにプロジェクトネットワークを作成します。デフォルトではプロジェクトネットワークは完全に分離され、他のプロジェクトとは共有されません。OpenStack Networking は、さまざまな種別のプロジェク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プロジェク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 プロジェク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、直接接続されたプロジェク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。VXLAN および GRE タイプドライバーは、ML2/OVS メカニズムドライバーと互換性があります。

- **Geneve のトンネル**: GENEVE は、ネットワーク仮想化における各種デバイスの変更機能を認識し、そのニーズに対応します。システム全体を規定するのではなく、トンネリングのフレームワークを提供します。GENEVE は、カプセル化中に追加されるメタデータの内容を柔軟に定義し、さまざまな仮想化シナリオへの対応を試みます。UDP をトランスポートプロトコルとして使用し、拡張可能なオプションヘッダーを使用してサイズを動的に変動させます。GNEVE はユニキャスト、マルチキャスト、およびブロードキャストをサポートします。GENEVE タイプドライバーは、ML2/OVN メカニズムドライバーと互換性があります。

注記

プロジェクトネットワークの QoS ポリシーを設定することが可能です。詳細は、「10 章*Quality of Service (QoS) ポリシーの設定*」を参照してください。

2.11.2. プロバイダーネットワーク

OpenStack の管理者は、プロバイダーネットワークを作成します。プロバイダーネットワークは、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で、プロジェクト間でプロバイダーネットワークを共有することもできます。

2.11.2.1. 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場合に便利です。複数のプロバイダーネットワークに接続するために、単一のインターフェース上で 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。

2.11.2.2. コントローラーノード用ネットワークの設定

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、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し (**--share** を使用)、他のユーザーが外部ネットワークに直接接続されたインスタンスを作成できるようにします。

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

3. **openstack subnet create** コマンドまたは Dashboard 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。

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

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

```
systemctl restart tripleo_neutron_api
```

2.11.2.3. ネットワークおよびコンピュートノード用ネットワークの設定

ノードを外部ネットワークに接続し、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するには、ネットワークノードおよびコンピュートノードで以下の手順を実施します。

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
```

2.11.2.4. ネットワークノードの設定

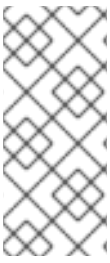
1. /etc/neutron/l3_agent.ini で **external_network_bridge =** に空の値を設定します。

external_network_bridge = に空の値を設定すると、複数の外部ネットワークブリッジを使用することができます。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。

```
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
```



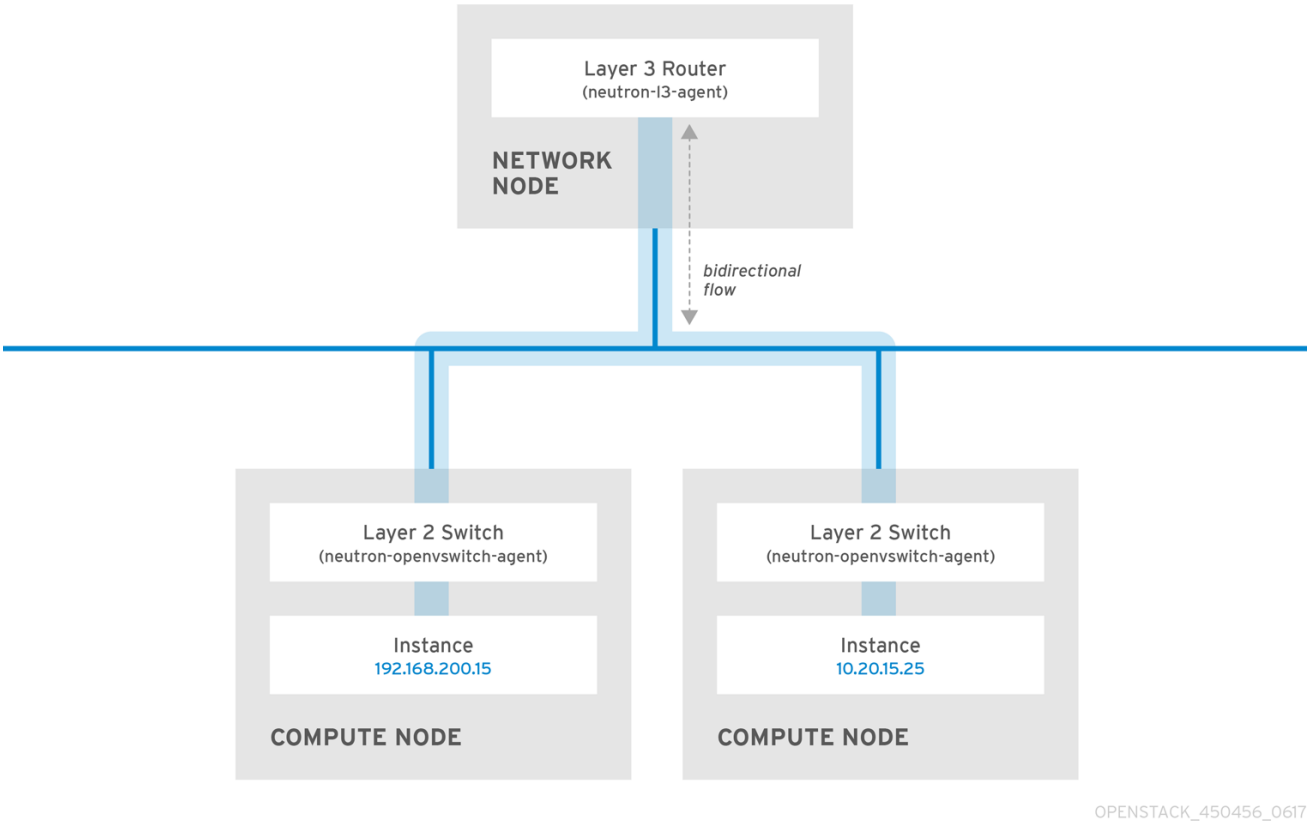
注記

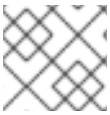
複数のフラットプロバイダーネットワークが存在する場合には、ネットワークごとに独立した物理インターフェースおよびブリッジを使用して外部ネットワークに接続する必要があります。**ifcfg-*** スクリプトを適切に設定し、**bridge_mappings** オプションで各ネットワークのコンマ区切りリストによりマッピングを指定します。ブリッジマッピングについての詳しい情報は、「11章 *ブリッジマッピングの設定*」を参照してください。

2.12. 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で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、複数の論理ネットワーク間よりも同じ論理ネットワーク内の方が早く移動します。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はルーターを通過する必要があり、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。



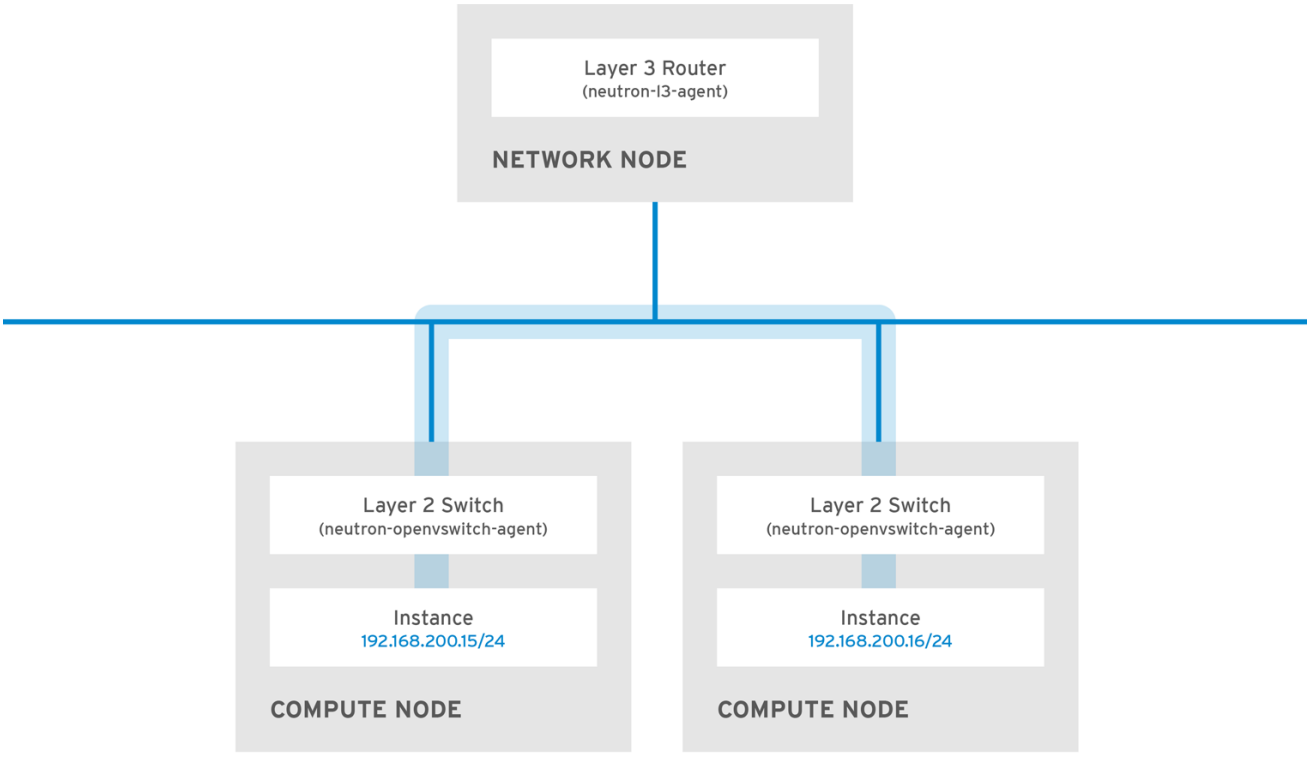


注記

高性能なハードウェアルーターでも、この構成では追加のレイテンシーが発生します。

2.12.1. 可能な範囲でのスイッチの使用

スイッチングは、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高速に機能することが可能です。頻繁に通信するシステム間のホップ数ができる限り少なくなるように設計してください。たとえば、以下の図は 2 つの物理ノードにまたがるスイッチ付きのネットワークを示しています。この場合、ナビゲーション用のルーターを使用せずに 2 つのインスタンスは直接通信することができます。これらのインスタンスが同じサブネットを共有するようになり、同じ論理ネットワーク上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ためには、VXLAN または GRE などのカプセル化トンネルを使用します。Red Hat では、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 MTU サイズを調節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」のアーティクル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 一般的なネットワーク管理タスク

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し DHCP を使用して IP アドレスを受け取ることができるように、ネットワークを作成します。ネットワークを Red Hat OpenStack Platform デプロイメント内の外部ネットワークと統合することや、他の物理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。外部ネットワーク接続に関する詳しい情報は、**「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、まったく異なるシステムを同じネットワークでホストし、それらのシステムを分離する必要があ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だけ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する必要があります。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、ルーティング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、以下の値を指定します。

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、ネットワークを外部ネットワークと統合しない場合には、**internal-only** とします。
管理状態有効	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 Down の状態で作成するには、このフィールドを使用します。その場合、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** タブで以下の値を指定します。

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 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ルーティングの次のホップとなり、ネットワークアドレス フィールドで指定した範囲内でなければなりません。たとえば、CIDR 形式のネットワークアドレスが 192.168.122.0/24 の場合には、通常デフォルトのゲートウェイは 192.168.122.1 となります。
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 を使用して、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 アドレス設定モード**: IPv6 ネットワークを作成する際の設定モード。IPv6 ネットワークを作成する場合には、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する必要があります。

     - **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、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。OpenStack Networking ルーターオプションまたは外部ルーターオプションを選択しま

す。ra_mode が slaac に、address_mode が slaac に設定された OpenStack Networking サブネットを作成するには、この設定を使用します。

- **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。ra_mode が dhcpv6-stateful に、address_mode が dhcpv6-stateful に設定されたサブネットを作成するには、この設定を使用します。

- **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。ra_mode が dhcpv6-stateless に、address_mode が dhcpv6-stateless に設定されたサブネットを作成するには、この設定を使用します。

- **IP アドレス割り当てプール**: 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 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。まず CIDR 形式で宛先の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次の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、**ネットワークの編集** をクリックしてオプションを変更することもできます。インスタンスの作成時には、そのサブネットを使用するように設定できるようになりました。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。プロジェクトを指定し使用するネットワーク種別を定義するには、これらのオプションを使用します。

高度なネットワークを作成するには、以下の手順を実施します。

1. Dashboard で、**管理 > ネットワーク > 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。

2. **プロジェクト** ドロップダウンリストを使用して、新規ネットワークをホストするプロジェクトを選択します。

3. **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。このオプションを選択すると、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できます。

- **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。

4. **Create Network** をクリックします。
   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選択し、**+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、新規サブネットの名前を選択します。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. **送信** をクリックします。
   ネットワーク上のインスタンスが、サブネット外部のシステムと通信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が使用され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。

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には、以下の手順を実施します。

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** 一覧で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。このネットワークに関連付けられたすべてのサブネットの **ネットワーク ID** フィールドには、この値が使用されます。

2. **プロジェクト > ネットワーク > ルーター**に移動し、**ルーター** 一覧で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、削除するサブネットと他のサブネットを区別することができます。インターフェースのネットワーク ID が以前のステップで書き留めた ID と一致しているかどうかを確認することで、さらに確実に識別することができます。

3. 削除する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。

4. **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。

5. 削除するサブネットの **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。

6. **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。
7. **ネットワークの削除** をクリックします。

3.5. プロジェクトのネットワークの削除

neutron purge コマンドを使用して、特定のプロジェクトに割り当てられた neutron リソースをすべて削除します。

たとえば、削除前に **test-project** プロジェクトの neutron リソースを削除するには、以下のコマンドを実行します。

```
# 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. サブネットの使用

サブネットを使用して、インスタンスにネットワーク接続を付与します。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。

既存のネットワークに対してのみ、サブネットを作成することができます。OpenStack Networking のプロジェク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、まったく異なるシステムを同じネットワークでホストし、それらのシステムを分離する必要があ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だけ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する必要があります。したがって、互いに大量のトラフィックを送受信する必要があるシステムを同じサブネットにグルーピングすることで、ネットワークレイテンシーおよび負荷を軽減することができます。

3.6.1. サブネットの作成

サブネットを作成するには、以下の手順に従います。

1. Dashboard で **プロジェクト > ネットワーク > ネットワーク**を選択して、**ネットワーク** ビューで対象のネットワークの名前をクリックします。

2. **+サブネットの作成** をクリックして、以下の値を指定します。

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
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 形式のアドレス。CIDR 形式のアドレスを決定するには、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 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バージョン (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 アドレスの範囲は、選択したプロトコル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ルーティングの次のホップとなり、ネットワークアドレス フィールドで指定した範囲内でなければなりません。たとえば、CIDR 形式のネットワークアドレスが 192.168.122.0/24 の場合には、通常デフォルトのゲートウェイは 192.168.122.1 となります。
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ます。

3. **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 を使用して、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 アドレス設定モード**: IPv6 ネットワークを作成する際の設定モード。IPv6 ネットワークを作成する場合には、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する必要があります。

     - **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、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。OpenStack Networking ルーターオプションまたは外部ルーターオプションを選択しま

す。ra_mode が slaac に、address_mode が slaac に設定された OpenStack Networking サブネットを作成するには、この設定を使用します。

- **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。ra_mode が dhcpv6-stateful に、address_mode が dhcpv6-stateful に設定されたサブネットを作成するには、この設定を使用します。

- **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。ra_mode が dhcpv6-stateless に、address_mode が dhcpv6-stateless に設定されたサブネットを作成するには、この設定を使用します。

- **IP アドレス割り当てプール**: 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 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。まず CIDR 形式で宛先の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次の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。

4. **作成** をクリックします。
   サブネットは、**サブネット** の一覧に表示されます。必要に応じて、**ネットワークの編集** をクリックしてオプションを変更することもできます。インスタンスの作成時には、そのサブネットを使用するように設定できるようになりました。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。

ネットワーク内の特定サブネットを削除するには、以下の手順を実施します。

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。
2. 対象のネットワークの名前をクリックします。
3. 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次の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。

ルーターを作成するには、以下の手順を実施します。

1. Dashboard で **プロジェクト > ネットワーク > ルーター**を選択し、**+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. **Set Gateway** をクリックします。
ルーターを追加したら、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送信するように設定しなければなりません。そのためには、サブネットとルーター間のインターフェースを作成します。

重要

サブネットのデフォルトルートを上書きすることはできません。サブネットのデフォルトルートが削除されると、L3 エージェントは自動的に対応するルーター名前空間のルートも削除するので、関連付けられたサブネットとの間でネットワークトラフィックの送受信ができなくなります。既存のルーター名前空間のルートが削除された場合にこの問題を解決するには、以下の手順を実施します。

1. サブネット上の全 Floating IP の割り当てを解除します。
2. ルーターをサブネットから切断します。
3. ルーターをサブネットに再接続します。
4. すべての Floating IP を再接続します。

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。

インターフェースの接続を解除しルーターを削除するには、以下の手順を実施します。

1. Dashboard で **プロジェクト > ネットワーク > ルーター**を選択し、削除するルーターの名前をクリックします。
2. 種別が **内部インタフェース** 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. 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

ルーターとサブネットを橋渡しするインターフェースを使用することにより、ルーターはインスタンスが送信したトラフィックを中継サブネット外部の宛先に転送することができます。

ルーターのインターフェースを追加して、新しいインターフェースをサブネットに接続するには、以下の手順を実施します。

注記

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。この機能を使用することで、ネットワーク管理タスクを実施する際に、全仮想ルーターおよびネットワークをグラフィカルに表した図を表示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その上にカーソルを移動し、**+インターフェースの追加** をクリックします。
3. ルーターに接続するサブネットを指定します。
   IP アドレスを指定することもできます。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が確認できるので、このアドレスを指定しておくとテストやトラブルシューティングに役立ちます。
4. **Add interface** 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

ルーターがサブネットのトラフィックを転送する必要がなくなった場合には、サブネットへのインターフェースを削除することができます。

インターフェースを削除するには、以下の手順を実施します。

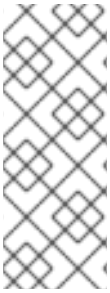
1. Dashboard で **プロジェクト > ネットワーク > ルーター**を選択します。
2. 削除するインターフェースをホストしているルーターの名前をクリックします。
3. インターフェース種別 (**内部インタフェース**) を選択し、**インターフェースの削除** をクリックします。

3.12. IP アドレスの設定

OpenStack Networking における IP アドレスの割り当てを管理するには、本項の手順に従います。

3.12.1. Floating IP アドレスプールの作成

Floating IP アドレスを使用して、ネットワークの受信ネットワークトラフィックを OpenStack インスタンスに転送することができます。まず適切にルーティング可能な外部 IP アドレスのプールを定義する必要があります。その後、それらの IP アドレスをインスタンスに動的に割り当てることができます。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 アドレスを割り当てたインスタンスにルーティングします。



注記

OpenStack Networking は、同じ IP 範囲/CIDR から全プロジェクト (テナント) に Floating IP アドレスを確保します。これにより、すべてのプロジェクトが全 Floating IP サブネットからの Floating IP を使用できることができます。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を 10 に設定する一方、**ProjectC** のクォータを 0 に設定することができます。

外部サブネットを作成する際に、Floating IP 確保用プールを定義することもできます。サブネットが Floating IP アドレスのみをホストする場合には、**openstack subnet create** コマンドで **--no-dhcp** オプションを指定して、DHCP による割り当てを無効にすることを検討してください。

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

以下に例を示します。

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

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 アドレスをインスタンスに動的に確保するには、以下の手順を実施します。

1. 以下の **openstack** コマンドを入力します。

   ```
   # openstack floating ip create
   ```

   以下の例では、特定の IP アドレスを選択する代わりに、OpenStack Networking にプールから Floating IP アドレスを確保するよう要求します。

   ```
   # openstack floating ip 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 アドレスを確保したら、特定のインスタンスに割り当てることができます。

2. 以下のコマンドを入力し、インスタンスに関連付けられたポート ID を特定します。

   ```
   # openstack port list
   ```

(ポート ID とインスタンスに割り当てられた固定 IP アドレスのマッピングが表示されます。)

```
# openstack 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"}    |
+--------+------+-------------+-----------------------------------------------------------+
```

3. インスタンス ID をインスタンスのポート ID に関連付けます。
openstack server add floating ip INSTANCE_NAME_OR_ID FLOATING_IP_ADDRESS

   以下に例を示します。

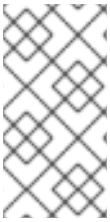
```
# openstack server add floating ip VM1 172.24.4.225
```

4. MAC アドレス (3 列目) がインスタンスのポートと一致していることを確認して、インスタンスの正しいポート ID を使用したことを確認します。

```
# openstack 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"}|
+--------+------+-------------+-----------------------------------------------------------+
```

3.13. 複数の FLOATING IP アドレスプールの作成

OpenStack Networking は、それぞれの L3 エージェントごとに 1 つの Floating IP プールをサポートします。したがって、追加の Floating IP プールを作成するには、L3 エージェントをスケールアウトする必要があります。



注記

/var/lib/config-data/neutron/etc/neutron/neutron.conf で、属性 **handle_internal_only_routers** の値が環境内の 1 つの L3 エージェントに対してのみ True に設定されていることを確認します。このオプションにより、L3 エージェントは外部ルーター以外だけ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

仮想インスタンスの送受信接続を可能にするには、仮想ネットワークと物理ネットワーク間をブリッジングします。

以下の手順では、例として示した物理インターフェース **eth0** はブリッジ **br-ex** にマッピングされます。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間を中継する機能を果たします。

これにより、**eth0** を通過するすべてのトラフィックは、設定した Open vSwitch を使用してインスタンスに到達します。

詳細は、「11章 *ブリッジマッピングの設定*」を参照してください。

物理 NIC を仮想 Open vSwitch ブリッジにマッピングするには、以下の手順を実施します。

1. テキストエディターで **/etc/sysconfig/network-scripts/ifcfg-eth0** を開き、以下のパラメーターをご自分のサイトのネットワークに適した値で更新します。

   - IPADDR
   - NETMASK GATEWAY
   - DNS1 (ネームサーバー)
     以下は例です。

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

2. テキストエディターで **/etc/sysconfig/network-scripts/ifcfg-br-ex** を開き、前のステップで eth0 に確保した IP アドレスの値で仮想ブリッジのパラメーターを更新します。

   ```
   # 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。本項では、必要なアドレスの数を適切に予測する方法、およびそのアドレスが環境のどこで使用されるかについて説明します。

4.1. VLAN のプランニング

Red Hat OpenStack Platform のデプロイメントを計画する際は、個々の IP アドレスの確保元となるサブネットの数を把握することから始めます。複数のサブネットを使用する場合、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、管理または API トラフィックは、Web トラフィックに対応するシステムと同じネットワーク上に置かないことが理想的です。VLAN 間のトラフィックはルーターを通過するので、ファイアウォールを実装してトラフィックフローを管理することができます。

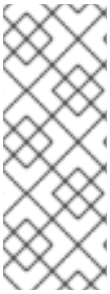
VLAN は、全体計画 (トラフィックの分離、高可用性、およびデプロイメント内のさまざまな種類の仮想ネットワークリソースに対する IP アドレスの使用状況などが含まれます) の一部としてプランニングする必要があります。

注記

1 つのネットワーク、あるいはネットワークノードの 1 つの OVS エージェントに設定できる VLAN の最大数は 4094 です。最大数を超える VLAN が必要な場合は、複数のプロバイダーネットワーク (VXLAN ネットワーク) および複数のネットワークノードを作成することができます。それぞれのノードには、最大で 4094 のプライベートネットワークを設定することができます。

4.2. ネットワークトラフィックの種別

異種のネットワークトラフィックをホストする場合は、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 を指定することができます。外部ネットワークだけは、外部の物理ネットワークへのルーティングを可能にする必要があります。本リリースでは、director により DHCP サービスが提供されます。



注記

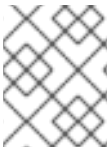
本項で説明するすべての分離 VLAN が、すべての OpenStack デプロイメントで必要な訳ではありません。たとえば、クラウドユーザーがアドホックの仮想ネットワークをオンデマンドで作成しない場合には、プロジェクトネットワークは必要ない可能性があります。各仮想マシンを他の物理システムと同じスイッチに直接接続する場合には、コンピュートノードを直接プロバイダーネットワークに接続し、インスタンスが直接そのプロバイダーネットワークを使用するように設定します。

- **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、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**: OpenStack のサービスは、API 通信、RPC メッセージ、データベース通信などに内部 API ネットワークを使用します。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、以下のサービスごとに IP アドレスの割当てを計画する必要があります。

- 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、Pacemaker により仮想 IP アドレスが物理ノード間で移動します。

- **ストレージ**: Block Storage、NFS、iSCSI、およびその他のストレージサービス。パフォーマンス上の理由から、このネットワークを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ー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、入着要求を受け取り、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、Storage Management 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。

- **プロジェクトネットワーク**: Neutron は、VLAN 分離 (各プロジェクトネットワークがネットワーク VLAN) または VXLAN か GRE によるトンネリングを使用した独自のネットワークを各プロジェクトに提供します。ネットワークトラフィックは、プロジェクトのネットワークごとに分離されます。それぞれのプロジェクトネットワークには IP サブネットが割り当てられ、複数のプロジェク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。このネットワークを SNAT に使用することもできます。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**: 既存のネットワークインフラストラクチャーにインスタンスをアタッチするには、プロバイダーネットワークを使用します。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 トラフィックを、それぞれ別の物理 NIC に割り当てます (冗長化の目的で、異なるスイッチに接続された複数の NIC が使用される場合があります)。

- **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 つの仮想 IP を割り当てる計画としてください。

4.4. 仮想ネットワーク

以下に示す仮想リソースは、OpenStack Networking の IP アドレスを消費します。これらのリソース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。

- **プロジェクトネットワーク**: 各プロジェクトネットワークには、サブネットが必要です。このサブネットを使用して、IP アドレスをインスタンスに割り当てることができます。

- **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 を使用する場合は、各ルーターのインターフェースに 2 つの IP アドレスが必要です。

- **インスタンス**: 各インスタンスには、インスタンスをホストするプロジェクト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インスタンスに Floating IP アドレスを確保する必要があります。

- **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。すべてのサービスが、少数の仮想 IP アドレスを共有します。API、RPC、およびデータベースサービスは、内部 API の仮想 IP アドレスで通信します。

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
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)	172.16.4.10 - 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 ルーターポートの確認

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

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、ポートの現在の状態 (DOWN または ACTIVE) を取得するには、以下の手順を実施します。

1. **r1** という名前のルーターに接続されたすべてのポートを表示するには、以下のコマンドを実行します。

```
#  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. 各ポートの詳細を表示するには、以下のコマンドを実行します。表示するポートのポート ID を指定します。コマンドの出力にはポートのステータスが含まれており、以下の例では状態が **ACTIVE** であることが分かります。

```
# openstack 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                                     |
+-----------------------+----------------------------------------------------------------------------------
+
```

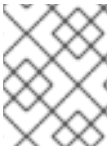
3. ポートごとにステップ 2 を実施して、ステータスを取得します。

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、(デプロイメントが) 複雑化しているように感じられる場合があります。しかし、OpenStack Networking のネットワークの接続性をトラブルシューティング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。

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訳ではありません。ping コマンドは、特定の宛先にトラフィックを送信し、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 プロトコルを使用した操作です。**ping** を使用するには、ICMP トラフィックが中間ファイアウォールを通過するのを許可する必要があります。

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。

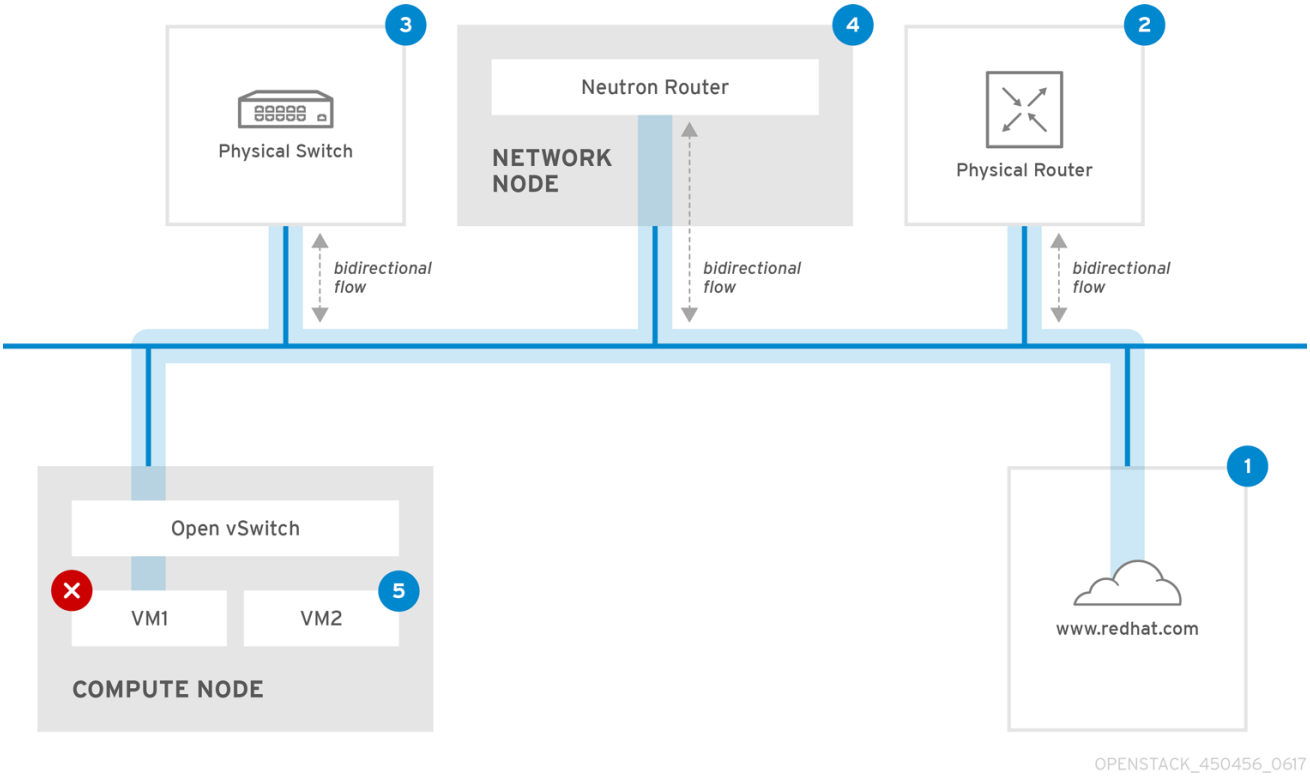
たとえば、以下に示す ping テストコマンドが成功するためには、複数のネットワークインフラストラクチャー層が検証されます。つまり、名前の解決、IP ルーティング、およびネットワークスイッチのすべてが正常に機能していなければなりません。

```
$ 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。パケットロスがゼロパーセントであれば、接続が安定し、タイムアウトが発生しなかったことを示しています。

```
--- 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
```

ping テストの送信先によっては、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。ping 送信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。



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 テストを送信することです。

- **成功**: このテストは、マシンとインターネットの間にあるさまざまなネットワークポイントすべてが正常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、問題は概ねローカルマシンの設定にあると考えられます。

2.**物理ルーター**: これは、外部の送信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する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。正常に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、このセグメントが正常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が設定されていることを確認してください。

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、仮想スイッチの設定を検証し、Open vSwitch (または Linux ブリッジ) 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 タグ付け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、以下の手順を実施します。

1. **ping <gateway-IP-address>** コマンドを使用して、ゲートウェイに ping 送信を行います。
   以下のコマンドで作成されたネットワークを例にして説明します。

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

   上記の例では、ゲートウェイの IP アドレスは **192.168.120.254** です。

   ```
   $ ping 192.168.120.254
   ```

2. ping 送信に失敗する場合は、以下の項目を確認します。

   a. 関連付けられた VLAN へのネットワークフローがあることを確認する。
   VLAN ID が設定されていない可能性があります。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。(例のステップ 1 の **--provider:segmentation_id=120** を参照してください。)

   b. コマンド **ovs-ofctl dump-flows <bridge-name>** を使用して、ブリッジインターフェースの VLAN フローを確認する。
   以下の例では、ブリッジは **br-ex** という名前です。

   ```
   # 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 network agent list** コマンドを使用して、すべてのエージェントが稼働し、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。**external_network_bridge** パラメーターにハードコードされた値がある場合、L3 エージェントと共にプロバイダーネットワークを使用することができず、必要なフローを作成することはできません。**external_network_bridge** の値は、external_network_bridge = "" の形式でなければなりません。

- **/etc/neutron/plugin.ini** ファイルで **network_vlan_ranges** の値を確認します。プロバイダーネットワークの場合には、数字の VLAN ID を指定しないでください。VLAN を分離したプロジェクトネットワークを使用している場合に限り、ID を指定します。

- 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. プロジェク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、プロジェクトが互いに干渉を生じさせることなくネットワークを設定できるように、すべてのプロジェクトトラフィックはネットワークの名前空間に含まれます。たとえば、ネットワークの名前空間を使用することで、異なるプロジェクトが 192.168.1.1/24 の同じサブネット範囲を指定しても、テナント間で干渉は生じません。

プロジェク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、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 network list** コマンドを使用して、すべてのプロジェク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  |
+--------------------------------------+-------------+--------------------------------------------------------+
```

上記の例では、**web-servers** ネットワークを確認します。web-servers 行の id の値 (**9cb32fe0-d7fb-432c-b116-f483c6497b08**) を書き留めてください。この値がネットワークの名前空間に追加されているので、次のステップで名前空間の特定が容易になります。

2. **ip netns list** コマンドを使用して、ネットワークの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
```

出力に、**web-servers** のネットワーク ID と一致する名前空間が表示されます。上記の例では、名前空間は **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08** です。

3. 名前空間内でトラブルシューティングのコマンド **ip netns exec <namespace>** を実行し、**web-servers** ネットワークの設定を検証します。

```
# 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
```

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** セッションを実行中のターミナルで、ping テストの結果の詳細を確認します。

```
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 解析を実行する際には、インスタンスではなくルータ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。qrouter によりリターンパケットで DNAT が実行されるので、これは想定どおりの動作です。

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について説明します。

7.1. OPENSTACK NETWORKING トポロジーの概要

OpenStack Networking (neutron) には、さまざまな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、プロジェクトネットワーク、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**: プロジェク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**: プロジェク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。
- **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。

7.1.1. 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 (ロールによって名前が付けられる) 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

本章の以下の手順は、これらの 3 つのノード種別が含まれる環境に適用されます。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、3 つすべてのノードで、コントローラーノードおよびネットワークノードに該当するセクションの手順を実行する必要があります。

7.2. フラットプロバイダーネットワークの使用

本項の手順では、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。

7.2.1. 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、既存の値リストに **flat** を追加し、**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。他のユーザーがインスタンスを直接接続できるように、このネットワークを共有ネットワークとして作成します。

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

3. **openstack subnet create** コマンドまたは OpenStack Dashboard を使用して、この外部ネットワーク内にサブネットを作成します。

```
# openstack subnet create --dhcp --allocation-pool start=192.168.100.20,end=192.168.100.100 --
gateway 192.168.100.1 --network public01 public_subnet
```

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
```

7.2.2. 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およびコンピュートノードを外部ネットワークに接続し、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、これらのノードで以下の手順を実施します。

1. Open vSwitch ブリッジとポートを作成します。以下のコマンドを実行し、外部ネットワークのブリッジ (br-ex) を作成して、対応するポート (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で **neutron-openvswitch-agent** サービスを再起動して、これらの変更を適用します。

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

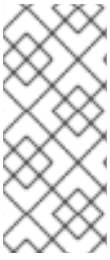
7.2.3. ネットワークノードの設定

1. **/etc/neutron/l3_agent.ini** の **external_network_bridge =** パラメーターに空の値を設定し、外部プロバイダーネットワークの使用を有効にします。

```
# 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、これらの変更を適用します。

```
# 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、ネットワークごとに独立した物理インターフェースおよびブリッジを使用して外部ネットワークに接続するようにします。**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ます。**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。

7.2.4. 外部ネットワークへのインスタンスの接続

外部ネットワークを作成したら、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。

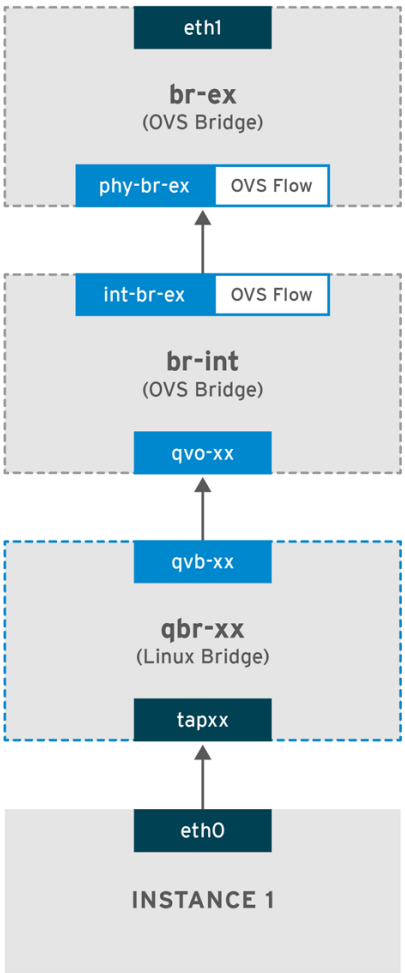
2. Dashboard の **ネットワーク** タブから、新たに作成した外部ネットワークに新規インスタンスを直接追加します。

7.2.5. フラットプロバイダーネットワークのパケットフローが機能する仕組み

本項では、フラットプロバイダーネットワークが設定された状況で、インスタンスに対するトラフィックがどのように送付されるかについて詳しく説明します。

フラットプロバイダーネットワークでの送信トラフィックのフロー

以下の図で、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について説明します。**br-ex** 外部ブリッジを設定した後に、物理インターフェースをブリッジに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、得られるインターフェースとブリッジの構成は、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. パケットはインスタンスの **eth0** インターフェースから送信され、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** に接続されます。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。

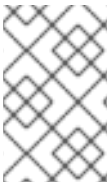
3. インターフェース **qvb-xx** は **qbr-xx** linux ブリッジに、**qvo-xx** は **br-int** Open vSwitch (OVS) ブリッジに接続されています。

`qbr-xx`Linux ブリッジの設定例:

```
 # 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** に到達する際に、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を以下に示します。

```
 # 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
```

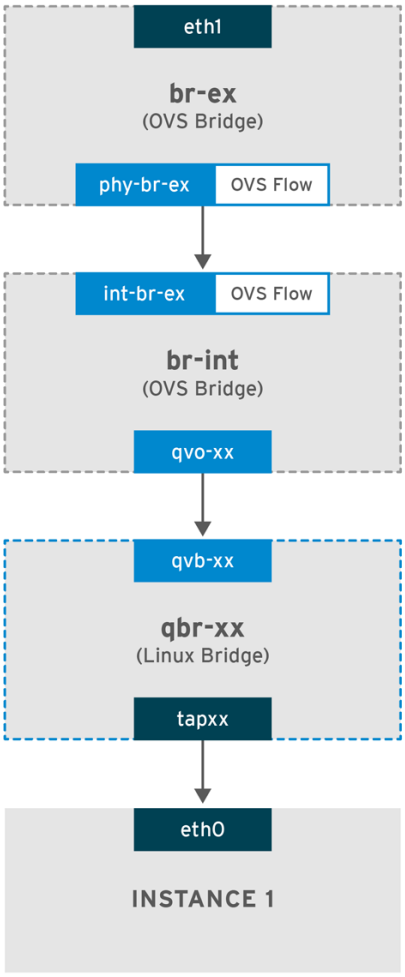
以下の出力例では、VLAN タグが **5** (**dl_vlan=5**) の **phy-br-ex** (**in_port=2**) に到達するパケットを示しています。また、br-ex の OVS フローにより VLAN タグが取り除かれ、パケットが物理インターフェースに転送されます。

```
# 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
```

物理インターフェースが別の VLAN タグ付けされたインターフェースの場合、その物理インターフェースはパケットにタグを追加します。

フラットプロバイダーネットワークでの受信トラフィックのフロー

本項では、外部ネットワークからの受信トラフィックがインスタンスのインターフェースに到達するまでのフローについて説明します。



1. 受信トラフィックは、物理ノードの **eth1** に到達します。

2. パケットは **br-ex** ブリッジに移動します。

3. このパケットは、パッチピア **phy-br-ex <--> int-br-ex** を通じて **br-int** に移動します。

以下の例では、**int-br-ex** はポート番号 **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 タグのない (vlan_tci=0x0000) int-br-ex (in_port=15) に到達するパケットを管理します。このルールにより、パケットに VLAN タグ 5 が追加され (**actions=mod_vlan_vid:5,NORMAL**)、**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。この値は、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 つの異なる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。

7.2.6. フラットプロバイダーネットワーク上での、インスタンス/物理ネットワーク間の接続のトラブルシューティング

「フラットプロバイダーネットワークのパケットフローが機能する仕組み」で提供される出力で、フラットプロバイダーネットワークで問題が発生した場合にトラブルシューティングを行うのに十分なデバッグ情報が得られます。以下の手順で、トラブルシューティングのプロセスについてさらに詳しく説明します。

1. bridge_mappings を確認します。

以下の例に示すように、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。

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

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

2.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。

```
 # openstack network 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 コマンドを実行し、パッチピア **int-br-ex <--> phy-br-ex** を使用して **br-int** と **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
```

/etc/neutron/plugins/ml2/openvswitch_agent.ini の **bridge_mapping** が正しく設定されていれば、この接続は **neutron-openvswitch-agent** サービスを再起動する際に作成されます。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。

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** として作成されていて、**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** の設定を確認してください。
- 最後に **ifcfg-br-ex** と **ifcfg-ethx** の設定を確認します。**ethX** が **br-ex** 内のポートとして追加されていること、および **ip a** の出力で **ifcfg-br-ex** および **ifcfg-ethx** に **UP** フラグが表示されることを確認します。

以下の出力では、**eth1** が **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いて、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
```

7.3. VLAN プロバイダーネットワークの使用

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には、本項の手順を実施して、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る VLAN プロバイダーネットワークを作成します。以下の例では、VLAN 範囲が **171 - 172** の **physnet1** という名前の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、物理インターフェース **eth1** を使用して物理ネットワークに接続されます。これらのインターフェースを接続するスイッチポートは、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。サンプルの VLAN ID と名前を使用して VLAN プロバイダーネットワークを設定するには、以下の手順を実施します。

7.3.1. 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 **vlan** メカニズムドライバーを有効にし、**vlan** を既存の値リストに追加します。

```
[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。

```
[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種別 **vlan** として作成して、設定済みの **physical_network** に関連付けます。他のユーザーが直接インスタンスを接続できるように、外部ネットワークを作成する際に **--shared** を使用します。以下のサンプルコマンドを実行して、2 つのネットワーク (VLAN 171 用および VLAN 172 用) を作成します。

```
openstack network create
  --provider-network-type vlan
  --external
  --provider-physical-network physnet1
  --segment 171
```

```
  --share

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

5. 複数のサブネットを作成して、外部ネットワークを使用するように設定します。**openstack subnet create** または Dashboard のどちらかを使用して、これらのサブネットを作成することができます。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 に関連付けられるようにします。以下の例では、VLAN 171 はサブネット **10.65.217.0/24** を、VLAN 172 は **10.65.218.0/24** を、それぞれ使用しています。

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

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

7.3.2. ネットワークノードとコンピュートノードの設定

ノードを外部ネットワークに接続し、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するには、ネットワークノードおよびコンピュートノードで以下の手順を実施します。

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

- 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. ノードをリブートするか、**network** サービスを再起動して、ネットワーク設定の変更を適用します。

```
# 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、変更を適用します。

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

7.3.3. ネットワークノードの設定

1. **/etc/neutron/l3_agent.ini** の **external_network_bridge =** パラメーターに空の値を設定します。これにより、ブリッジベースの外部ネットワーク (**external_network_bridge = br-ex** と設定する) ではなく、プロバイダー外部ネットワークを使用することができます。

```
# 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
```

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

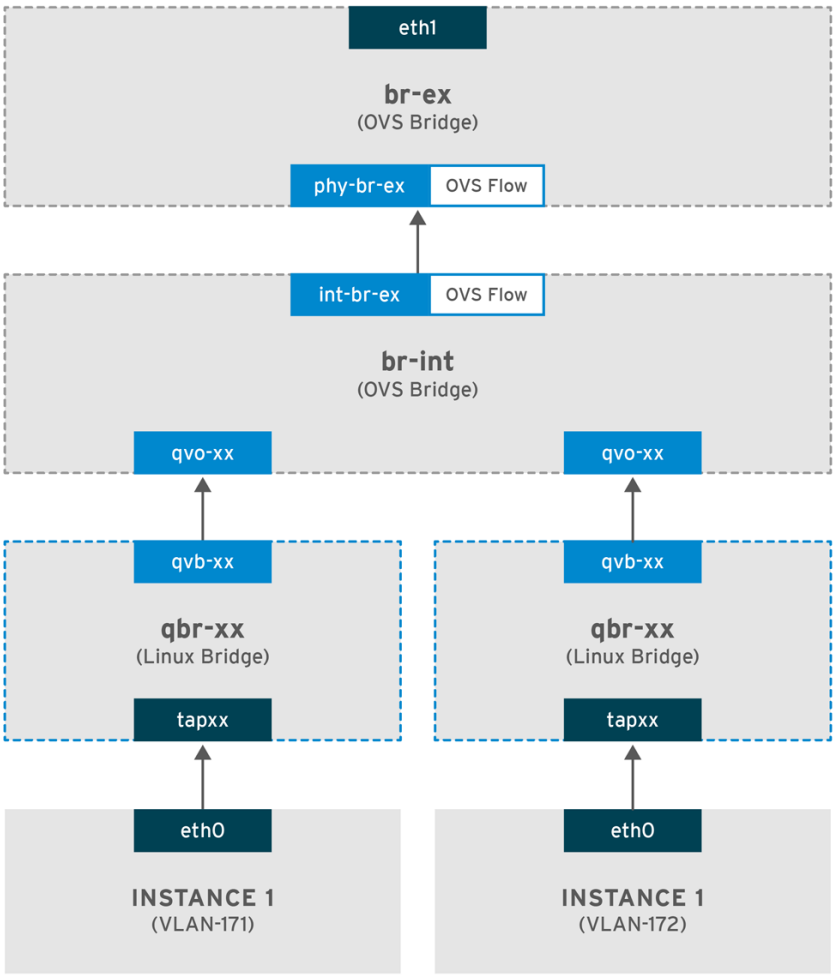
7.3.4. VLAN プロバイダーネットワークのパケットフローが機能する仕組み

本項では、VLAN プロバイダーネットワークが設定された状況で、インスタンスに対するトラフィックがどのように送付されるかについて詳しく説明します。

VLAN プロバイダーネットワークでの送信トラフィックのフロー

以下の図で、インスタンスから送信され直接 VLAN プロバイダー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** を設定した後に、物理インターフェー

スをブリッジに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、得られるインターフェースとブリッジの構成は、以下の図のようになります。



OPENSTACK_450456_0617

1. インスタンスの **eth0** インターフェースから送信されたパケットは、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、veth ペア **qvbxx <→ qvoxx** を使用して **br-int** に接続されます。

3. **qvbxx** は linux ブリッジ **qbr-xx** に、**qvoxx** は Open vSwitch ブリッジ **br-int** に接続されています。

Linux ブリッジ上の qvb-xx の設定例

以下の例には、2 つのインスタンスおよびこれに対応する 2 つの linux ブリッジが表示されています。

```
# 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。この例では、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、この VLAN タグがパケットのヘッダーに追加されます。

- 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**) の付いた phy-br-ex (**in_port=4**) に到達するパケットが表示されます。Open vSwitch は VLAN タグを 171 に置き換え (**actions=mod_vlan_vid:171,NORMAL**)、パケットを物理インターフェースに転送します。このコマンドでは、VLAN タグ 3 (**dl_vlan=3**) の付いた phy-br-ex (**in_port=4**) に到達するパケットも表示されます。Open vSwitch は VLAN タグを 172 に置き換え (**actions=mod_vlan_vid:172,NORMAL**)、パケットを物理インターフェースに転送します。neutron-openvswitch-agent は、これらのルールを追加します。

```
# 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。

VLAN プロバイダーネットワークでの受信トラフィックのフロー

以下のフローは、プロバイダーネットワーク provider-171 に VLAN タグ 2 を使用し、プロバイダーネットワーク provider-172 に VLAN タグ 3 を使用して、コンピュートノードでテストを行った際の例です。フローは、統合ブリッジ br-int のポート 18 を使用します。

実際の VLAN プロバイダーネットワークでは、異なる設定が必要な場合があります。また、ネットワークの設定要件は、2 つの別個のコンピュートノード間で異なる場合があります。

以下のコマンドの出力には、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
```

以下のコマンドの出力には、br-int のフローのルールが表示されます。

```
# 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
```

受信フローの例

ここでは、以下の br-int OVS フローの例を示しています。

```
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
```

- VLAN タグ 172 の付いた外部ネットワークからのパケットが、物理ノード上の **eth1** から **br-ex** ブリッジに到達します。
- このパケットは、パッチピア **phy-br-ex <-> int-br-ex** を通じて **br-int** に移動します。
- パケットは、フローの条件 (**in_port=18,dl_vlan=172**) を満たします。
- フローのアクション (**actions=mod_vlan_vid:3,NORMAL**) は VLAN タグ 172 を内部 VLAN タグ 3 に置き換え、通常のレイヤー 2 処理でパケットをインスタンスに転送します。

7.3.5. VLAN プロバイダーネットワーク上での、インスタンス/物理ネットワーク間の接続の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、「VLAN プロバイダーネットワークのパケットフローが機能する仕組み」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. 一貫した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

# openstack network 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。

```
# openstack network 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 **vlan** として作成されていて、**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** の設定を再確認してください。

5.最後に、ifcfg-br-ex と ifcfg-ethx の設定を再確認します。br-ex にポート **ethX** が含まれていること、および **ip a** コマンドの出力で ifcfg-br-ex と ifcfg- ethx の両方に **UP** フラグが表示されることを

確認します。
たとえば、以下の出力では **eth1** が **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"
```

以下のコマンドでは、**eth1** がポートとして追加されていること、およびカーネルがこのインターフェースからのパケットをすべて OVS ブリッジ **br-ex** に移動するように設定され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
```

7.4. コンピュートのメタデータアクセスの有効化

本章で説明する方法で接続されたインスタンスは、プロバイダー外部ネットワークに直接アタッチされ、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。OpenStack Networking (neutron) ルーターは使用されません。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。ただし、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
```

7.5. FLOATING IP アドレス

Floating IP がすでにプライベートネットワークに割り当てられている場合でも、同じネットワークを使用して Floating IP アドレスをインスタンスに確保することができます。このネットワークから Floating IP として確保するアドレスは、ネットワークノードの **qrouter-xxx** の名前空間にバインドされ、割り当て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確保する 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 で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 シリーズ

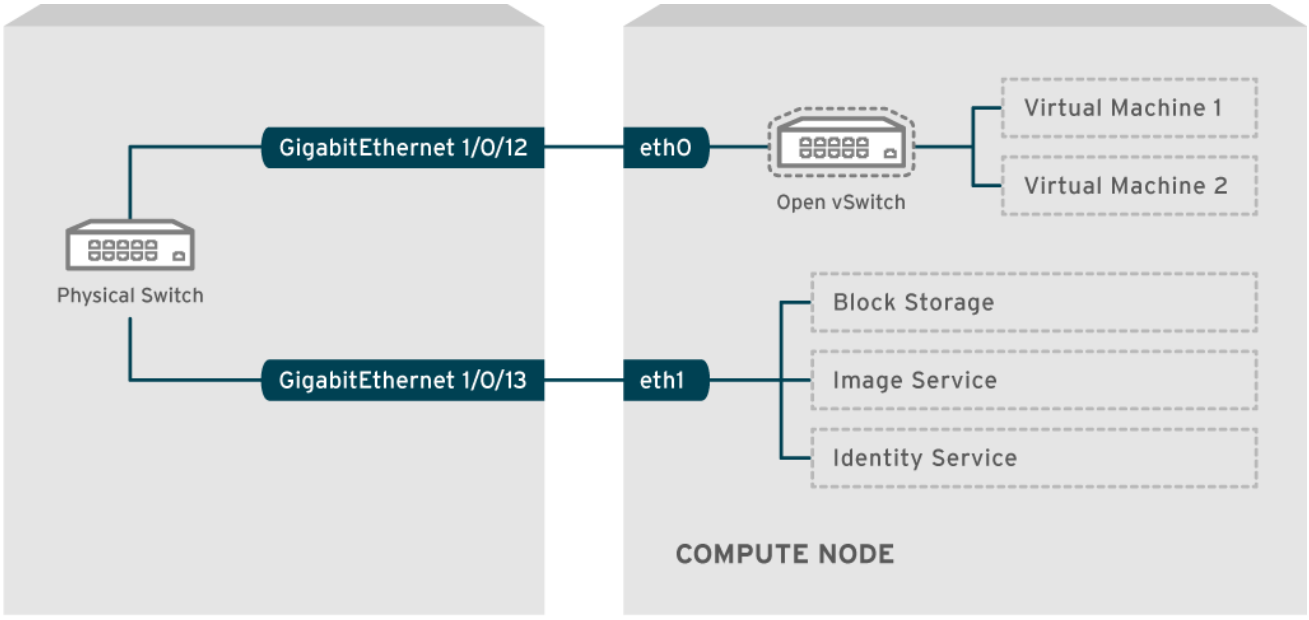
8.1. 物理ネットワーク環境のプランニング

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します。これには、インスタンストラフィック、ストレージデータ、および認証要求が含まれます。これらの NIC が伝送するトラフィックの種別によって、物理スイッチ上のポートの設定方法が異なります。

まず、コンピュートノード上のどの物理 NIC でどのトラフィック種別を伝送するか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際に、そのスイッチポートがトラン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



OPENSTACK_377160_1115

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。

ネットワークインターフェースのボンディングに関する詳細は、『オーバークラウドの高度なカスタマイズ』の「ネットワークインターフェースボンディング」の章を参照してください。

8.2. CISCO CATALYST スイッチの設定

8.2.1. トランクポートについて

OpenStack Networking により、インスタンスを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 VLAN の通過を許可することを意味します。これらのポートを使用することで、VLAN は仮想スイッチを含む複数のスイッチ間にまたがることができます。たとえば、物理ネットワークで VLAN110 のタグが付いたトラフィックがコンピュートノードに到達すると、タグの付いたトラフィックが 8021q モジュールによって vSwitch 上の適切な VLAN に転送されます。

8.2.2. Cisco Catalyst スイッチで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に設定することが可能です。
  この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。

重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

```
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
```

以下の一覧を使用して、上記のパラメーターについて説明します。

フィールド	説明
interface GigabitEthernet1/0/12	X ノードの NIC の接続先となるスイッチポート。**GigabitEthernet1/0/12** の値を、実際の環境の正しいポートの値で置き換えるようにしてください。ポートの一覧を表示するには、show interface コマンドを使用します。
description Trunk to Compute Node	このインターフェースを識別するのに使用する一意の説明的な値。

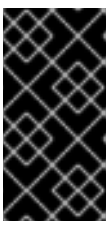
フィールド	説明
spanning-tree portfast trunk	環境で STP が使用される場合には、この値を設定して Port Fast に対してこのポートがトランクトラフィックに使用されることを指示します。
switchport trunk encapsulation dot1q	802.1q のトランク標準 (ISL ではなく) を有効にします。この値は、スイッチがサポートする設定によって異なります。
switchport mode trunk	このポートは、アクセスポートではなく、トランクポートとして設定します。これで VLAN トラフィックが仮想スイッチに到達できるようになります。
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。

8.2.3. アクセスポートについて

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訳ではないので、すべての NIC 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。アクセスポートに必要なのは 1 つの VLAN だけで、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可能性があります。これらのポートは一般的にアクセスポートと呼ばれ、必要な設定は通常、トランクポートよりも簡単です。

8.2.4. Cisco Catalyst スイッチで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** のアクセスポートとして設定します。
この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (インターフェース GigabitEthernet1/0/12) に接続されています。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

```
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
interface GigabitEthernet1/0/13	X ノードの NIC の接続先となるスイッチポート。**GigabitEthernet1/0/12** の値を、実際の環境の正しいポートの値で置き換えるようにしてください。ポートの一覧を表示するには、show interface コマンドを使用します。
description Access port for Compute Node	このインターフェースを識別するのに使用する一意の説明的な値。
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 アドレスを設定する必要があります。
spanning-tree portfast	STP を使用する場合には、この値を設定し、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ます。これにより、初回接続時 (例: サーバーのリブート時など) 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。

8.2.5. LACP ポートアグリゲーションについて

LACP を使用して、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。

8.2.6. 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』の「ネットワークインターフェースボンディング」の章を参照してください。

8.2.7. Cisco Catalyst 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 NIC を共に物理的にスイッチ (例: ポート 12 と 13) 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 コマンドを実行して running-config を startup-config にコピーし、変更を適用するのを忘れないようにしてください。

8.2.8. MTU 設定について

特定のネットワークトラフィック種別に対して、MTU サイズを調整する必要があります。たとえば、特定の NFS または iSCSI のトラフィックでは、ジャンボフレーム (9000 バイト) が必要になります。

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 *最大伝送単位 (MTU) 設定の定義*」を参照してください。

8.2.9. Cisco Catalyst スイッチでの MTU の設定

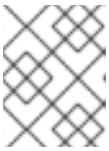
Cisco Catalyst 3750 スイッチでジャンボフレームを有効にするには、以下の例に示す手順を実施します。

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。以下のコマンドを実行して、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インターフェースごとの MTU 設定をサポートしていれば、この機能を使用する方が望ましい場合があります。

```
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 コマンドを実行して running-config を startup-config にコピーし、変更を保存するのを忘れないようにしてください。

3. スイッチを再読み込みして変更を適用します。

重要

スイッチを再読み込みすると、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。したがって、計画的なメンテナンス期間中にのみスイッチの再読込みを行ってください。

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

4. スイッチが再読み込みされたら、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
```

8.2.10. LLDP ディスカバリーについて

ironic-python-agent サービスは、接続されたスイッチからの 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検出を補助します。

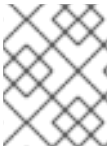
8.2.11. Cisco Catalyst スイッチでの LLDP の設定

1. **lldp run** コマンドを実行して、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 コマンドを実行して running-config を startup-config にコピーし、変更を保存するのを忘れないようにしてください。

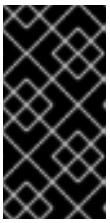
8.3. CISCO NEXUS スイッチの設定

8.3.1. トランクポートについて

OpenStack Networking により、インスタンスを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 VLAN の通過を許可することを意味します。これらのポートを使用することで、VLAN は仮想スイッチを含む複数のスイッチ間にまたがることができます。たとえば、物理ネットワークで VLAN110 のタグが付いたトラフィックがコンピュートノードに到達すると、タグの付いたトラフィックが 8021q モジュールによって vSwitch 上の適切な VLAN に転送されます。

8.3.2. Cisco Nexus スイッチで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。
  この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

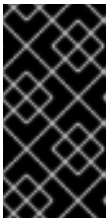
```
interface Ethernet1/12
  description Trunk to Compute Node
  switchport mode trunk
  switchport trunk allowed vlan 2,110,111
  switchport trunk native vlan 2
end
```

8.3.3. アクセスポートについて

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訳ではないので、すべての NIC 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。アクセスポートに必要なのは 1 つの VLAN だけで、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可能性があります。これらのポートは一般的にアクセスポートと呼ばれ、必要な設定は通常、トランクポートよりも簡単です。

8.3.4. Cisco Nexus スイッチで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** のアクセスポートとして設定します。この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

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

8.3.5. LACP ポートアグリゲーションについて

LACP を使用して、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。

8.3.6. 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』の「ネットワークインターフェースボンディング」の章を参照してください。

8.3.7. Cisco Nexus 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 NIC を物理的にスイッチ (例: ポート 12 と 13) に接続します。
2. LACP が有効なことを確認します。

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

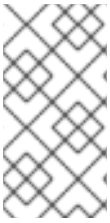
3. ポート 1/12 と 1/13 をアクセスポートおよびチャネルグループのメンバーとして設定します。デプロイメントによっては、アクセスインターフェースの代わりにトランクインターフェースをデプロイすることができます。

   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、アクセスポートだけを設定する方が望ましい場合があります。多くの場合、これらのインターフェースには VLAN タグ付けが設定されています。

```
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
```

8.3.8. MTU 設定について

特定のネットワークトラフィック種別に対して、MTU サイズを調整する必要があります。たとえば、特定の NFS または iSCSI のトラフィックでは、ジャンボフレーム (9000 バイト) が必要になります。



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 *最大伝送単位 (MTU) 設定の定義*」を参照してください。

8.3.9. Cisco Nexus 7000 スイッチでの MTU の設定

7000 シリーズのスイッチ上の単一のインターフェースに、MTU の設定を適用します。

- 以下のコマンドを実行して、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
```

8.3.10. LLDP ディスカバリーについて

ironic-python-agent サービスは、接続されたスイッチからの 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検出を補助します。

8.3.11. Cisco Nexus 7000 スイッチでの LLDP の設定

- 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、LLDP を有効に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 コマンドを実行して running-config を startup-config にコピーし、変更を保存するの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. トランクポートについて

OpenStack Networking により、インスタンスを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 VLAN の通過を許可することを意味します。これらのポートを使用することで、VLAN は仮想スイッチを含む複数のスイッチ間にまたがることができます。たとえば、物理ネットワークで VLAN110 のタグが付いたトラフィックがコンピュートノードに到達すると、タグの付いたトラフィックが 8021q モジュールによって vSwitch 上の適切な VLAN に転送されます。

8.4.2. Cumulus Linux スイッチでのトランクポートの設定

- 以下の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します。
  この設定では、物理ノードのトランシーバーが物理スイッチポート (swp1 および swp2) に接続されていることを前提としています。

重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

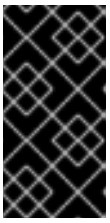
```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200
```

8.4.3. アクセスポートについて

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訳ではないので、すべての NIC 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。アクセスポートに必要なのは 1 つの VLAN だけで、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可能性があります。これらのポートは一般的にアクセスポートと呼ばれ、必要な設定は通常、トランクポートよりも簡単です。

8.4.4. Cumulus Linux スイッチで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。
  この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチ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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

```
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
```

8.4.5. LACP ポートアグリゲーションについて

LACP を使用して、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。

8.4.6. MTU 設定について

特定のネットワークトラフィック種別に対して、MTU サイズを調整する必要があります。たとえば、特定の NFS または iSCSI のトラフィックでは、ジャンボフレーム (9000 バイト) が必要になります。

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 *最大伝送単位 (MTU) 設定の定義*」を参照してください。

8.4.7. 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
```

注記

sudo ifreload -a コマンドを実行して更新した設定を 再読み込みし、変更を適用するのを忘れないようにしてください。

8.4.8. LLDP ディスカバリーについて

ironic-python-agent サービスは、接続されたスイッチからの 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検出を補助します。

8.4.9. Cumulus Linux スイッチでの LLDP 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 隣接デバイスをすべて表示するには、以下のコマンドを実行します。

```
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 EXOS スイッチの設定

8.5.1. トランクポートについて

OpenStack Networking により、インスタンスを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 VLAN の通過を許可することを意味します。これらのポートを使用することで、VLAN は仮想スイッチを含む複数のスイッチ間にまたがることができます。たとえば、物理ネットワークで VLAN110 のタグが付いたトラフィックがコンピュートノードに到達すると、タグの付いたトラフィックが 8021q モジュールによって vSwitch 上の適切な VLAN に転送されます。

8.5.2. 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設定します。
この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。

重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

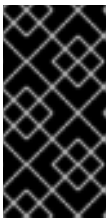
```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

8.5.3. アクセスポートについて

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訳ではないので、すべての NIC 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。アクセスポートに必要なのは 1 つの VLAN だけで、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可能性があります。これらのポートは一般的にアクセスポートと呼ばれ、必要な設定は通常、トランクポートよりも簡単です。

8.5.4. Extreme Networks EXOS スイッチでのアクセスポートの設定

- 以下の設定例では、Extreme Networks X-670 シリーズでは、**eth1** のアクセスポートとして **10** が使用されています。
この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

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

以下に例を示します。

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

8.5.5. LACP ポートアグリゲーションについて

LACP を使用して、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。

8.5.6. 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』の「ネットワークインターフェースボンディング」の章を参照してください。

8.5.7. 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
```

以下に例を示します。

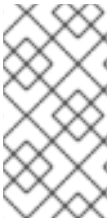
```
#enable sharing 11 grouping 11,12 lacp
#configure vlan DATA add port 11 untagged
```

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」を参照してください。

8.5.8. MTU 設定について

特定のネットワークトラフィック種別に対して、MTU サイズを調整する必要があります。たとえば、特定の NFS または iSCSI のトラフィックでは、ジャンボフレーム (9000 バイト) が必要になります。



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 *最大伝送単位 (MTU) 設定の定義*」を参照してください。

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

- 以下の例に示すコマンドを実行して、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 パケット転送のサポートを設定します。

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

以下に例を示します。

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

8.5.10. LLDP ディスカバリーについて

ironic-python-agent サービスは、接続されたスイッチからの 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検出を補助します。

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

- 以下の例では、Extreme Networks EXOS スイッチで LLDP を有効にします。**11** はポート文字列を表します。

```
enable lldp ports 11
```

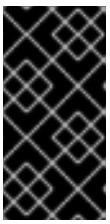
8.6. JUNIPER EX シリーズスイッチの設定

8.6.1. トランクポートについて

OpenStack Networking により、インスタンスを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 VLAN の通過を許可することを意味します。これらのポートを使用することで、VLAN は仮想スイッチを含む複数のスイッチ間にまたがることができます。たとえば、物理ネットワークで VLAN110 のタグが付いたトラフィックがコンピュートノードに到達すると、タグの付いたトラフィックが 8021q モジュールによって vSwitch 上の適切な VLAN に転送されます。

8.6.2. 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します。
  この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

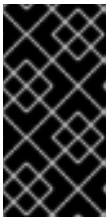
```
ge-1/0/12 {
        description Trunk to Compute Node;
            unit 0 {
                family ethernet-switching {
                    port-mode trunk;
                    vlan {
                        members [110 111];
                        }
                    native-vlan-id 2;
                }
            }
}
```

8.6.3. アクセスポートについて

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訳ではないので、すべての NIC 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。アクセスポートに必要なのは 1 つの VLAN だけで、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可能性があります。これらのポートは一般的にアクセスポートと呼ばれ、必要な設定は通常、トランクポートよりも簡単です。

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

- Juniper EX シリーズに関する以下の例では、**ge-1/0/13** が **eth1** のアクセスポートとして示されています。
  この設定では、物理ノードの NIC がイーサネットケーブルにより物理スイッチポート (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。



重要

以下に示す値は、例として提示しています。この例で使用している値を、実際の環境に合わせて変更する必要があります。これらの値を調整せずにコピーしてご自分のスイッチ設定に貼り付けると、予期せぬ機能停止を招く可能性があります。

```
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;
            }
        }
}
```

8.6.5. LACP ポートアグリゲーションについて

LACP を使用して、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。

8.6.6. 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』の「ネットワークインターフェースボンディング」の章を参照してください。

8.6.7. 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 を物理的にスイッチ (例: ポート 12 と 13) 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、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いずれかを 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 を設定するボンディングメンバーは、**instackenv.json** で指定する MAC アドレスを持つボンディングメンバーでなければなりません (ironic に認識される MAC アドレスは、force-up が設定される MAC アドレスと同じでなければなりません)。

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のを忘れないようにしてください。

8.6.8. MTU 設定について

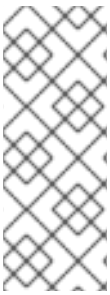
特定のネットワークトラフィック種別に対して、MTU サイズを調整する必要があります。たとえば、特定の NFS または iSCSI のトラフィックでは、ジャンボフレーム (9000 バイト) が必要になります。

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 *最大伝送単位 (MTU) 設定の定義*」を参照してください。

8.6.9. 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。

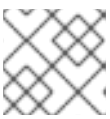


注記

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。したがって、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の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
```

8.6.10. LLDP ディスカバリーについて

ironic-python-agent サービスは、接続されたスイッチからの 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検出を補助します。

8.6.11. 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でローバルに有効にすることや、特定のインターフェースでのみ有効にすることができます。

- LLDP を Juniper EX 4200 スイッチでグローバルに有効にするには、以下の設定を使用します。

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

- LLDP を単一のインターフェース **ge-1/0/14** で有効にするには、以下の設定を使用します。

```
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のを忘れないようにしてください。

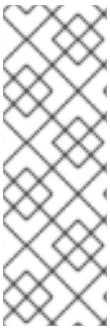
パート II. 高度な設定

高度な Red Hat OpenStack Platform Networking 機能について、クックブック形式のシナリオを用いて説明します。

第9章 最大伝送単位 (MTU) 設定の定義

9.1. MTU の概要

OpenStack Networking は、インスタンスに安全に適用できる最大伝送単位 (MTU) サイズの最大値を計算することができます。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。

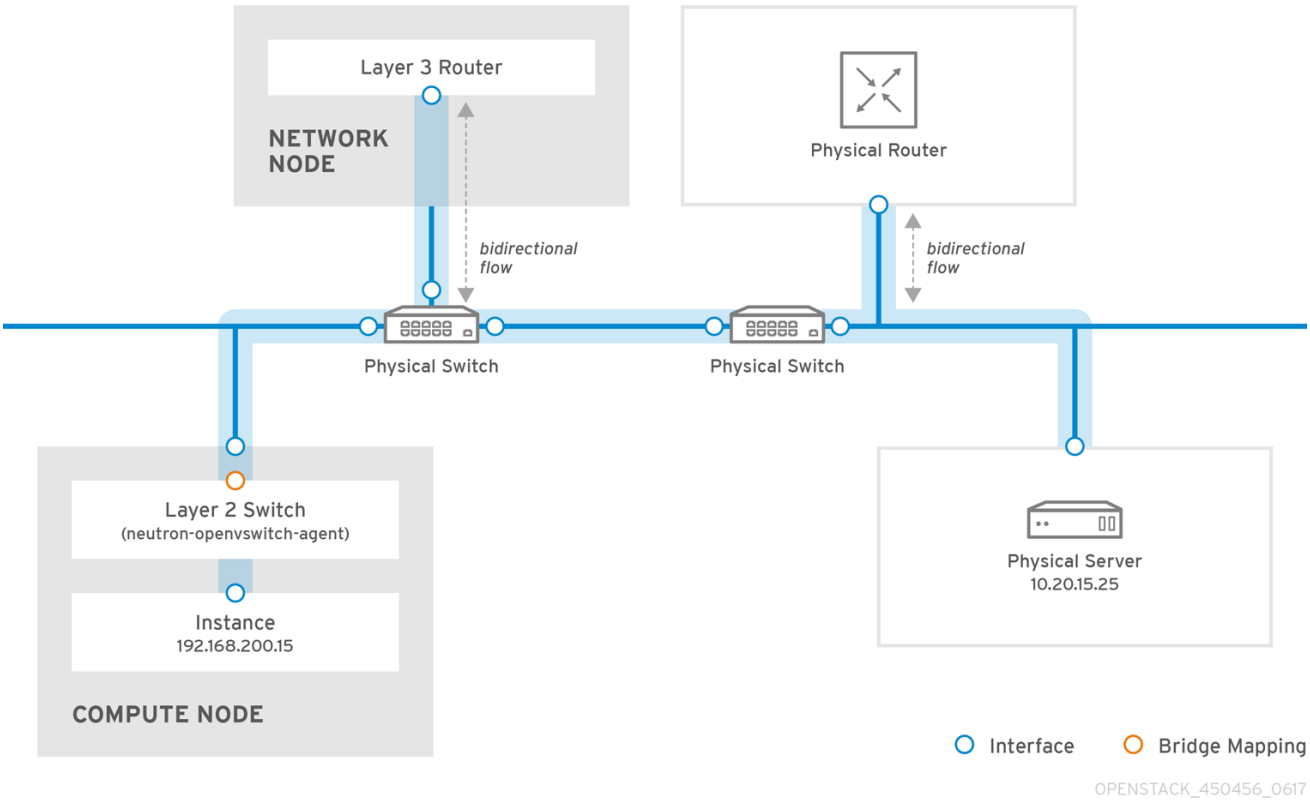


注記

neutron net-show コマンドを使用して、OpenStack Networking が計算する MTU の最大値を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 クライアントに通知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 クライアントに通知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ります。

MTU 設定は、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、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、送信先のサーバーなど、パケットが通過するすべてのポイントで同じでなければなりません。

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。特定の MTU サイズのパケットに対応するように、ネットワークトラフィックを処理する全インターフェースの MTU 値を変更する必要があります。トラフィックがインスタンス 192.168.200.15 から物理サーバー 10.20.15.25 に伝送される場合には、この変更が必要です。



MTU 値に一貫性がないと、ネットワークにさまざまな問題が発生します。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、トラブルシューティングが困難です。正しい MTU 値が間違いなく設定されるように、考え得るすべてのネットワークポイントを特定して確認する必要があるためです。

9.2. DIRECTOR での MTU 設定の定義

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。ブリッジ、ボンディング (該当する場合)、インターフェース、および VLAN で MTU を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}
```

9.3. MTU 計算結果の確認

インスタンスが使用可能な MTU の最大値として計算された MTU 値を確認することができます。計算されたこの MTU 値を使用して、ネットワークトラフィックのパスとなる全インターフェースを設定します。

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

第10章 QUALITY OF SERVICE (QOS) ポリシーの設定

Quality of Service (QoS) ポリシーを使用して送信および受信トラフィックにレート制限を適用することで、さまざまなインスタンスのサービスレベルを提供することができます。

個別のポートに QoS ポリシーを適用することができます。QoS ポリシーをプロジェクトネットワークに適用することもできます。この場合、特定のポリシーが設定されていないポートは、ネットワークのポリシーを継承します。

注記

DHCP や内部ルーターポート等の内部ネットワークが所有するポートは、ネットワークポリシーの適用から除外されます。

QoS ポリシーは、動的に適用、変更、削除することができます。ただし、最小帯域幅を確保する QoS ポリシーについては、ポリシーが割り当てられたポートを使用するインスタンスがない場合に限り、変更を適用することができます。

10.1. QOS ルール

以下のルール種別で、特定の Quality of Service (QoS) ポリシーの制限を定義します。

- **bandwidth_limit**: ネットワーク、ポート、または Floating IP での帯域幅を制限します。このルール種別を実装すると、指定したレートを超過するトラフィックはすべてドロップされます。
- **minimum_bandwidth**: 特定のトラフィック種別での最小帯域幅の制約を提供します。このルール種別を実装すると、ルールが適用される各ポートに指定した最小帯域幅を提供するための最大限の努力が行われます。
- **dscp_marking**: ネットワークトラフィックに Differentiated Services Code Point (DSCP) 値をマーキングします。

関連する手順

以下の手順で、各ルール種別で QoS ポリシーを作成する方法を説明します。

- 帯域幅を制限する QoS ポリシーおよびルールの作成と適用
- 最小帯域幅を確保する QoS ポリシーおよびルールの作成と適用
- 送信トラフィックの DSCP マーキング

10.2. QOS ポリシーおよびルールの作成と適用

Quality of Service (QoS) ポリシーおよびルールを作成してポリシーをポートに適用するには、以下の手順を実施します。

手順

1. QoS ポリシーを作成するプロジェクトの ID を特定します。

```
(overcloud) $ openstack project list
```

```
----------------------------------------+
| ID                                | Name    |
----------------------------------------+
| 8c409e909fb34d69bc896ab358317d60 | admin   |
| 92b6c16c7c7244378a062be0bfd55fa0 | service |
----------------------------------------+
```

2. 新規 QoS ポリシーを作成します。

```
(overcloud) $ openstack network qos policy create --share --project <project_ID>
<policy_name>
```

3. QoS ポリシーの新規ルールを作成します。

```
(overcloud) $ openstack network qos rule create --type <rule-type> [rule properties]
<policy_name>
```

表10.1 ルールの属性

属性	説明
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のできるデータの最大量 (キロビット単位)。トークンバッファーは「max_kbps」の速度で補充されます。 TCP トラフィックのバースト値は、必要な帯域幅の制限値の 80% に設定することができます。たとえば、帯域幅の制限が 1000 kbps に設定されている場合には、800 kbps のバースト値で十分です。 **注記** ● バースト値を低く設定しすぎると、帯域幅の制限値が適切であっても帯域幅の使用量にスロットリングが適用されるため、帯域幅が想定よりも低くなります。 ● バースト値の設定が高すぎると、ほとんどのパケットに制限が適用されず、帯域幅の制限が予想よりも高くなります。
min-kbps	インスタンスに確保される最小帯域幅 (Kbps 単位)

属性	説明
ingress/egress	ルールが適用されるトラフィックの方向。クラウドサーバーの視点からは、受信はダウンロードを意味し、送信はアップロードを意味します。
dscp-mark	DSCP マークの 10 進数値を指定します。

4. ポリシーを適用するポートまたはネットワークを設定します。既存のポートまたはネットワークを更新することも、ポリシーを適用する新規ポートまたはネットワークを作成することもできます。

   - 既存のポートにポリシーを適用する場合:

     ```
     (overcloud) $ openstack port set --qos-policy <policy_name> <port_name|port_ID>
     ```

   - 新規ポートを作成する場合:

     ```
     (overcloud) $ openstack port create --qos-policy <policy_name> --network
     <network_name|network_ID> <port_name|port_ID>
     ```

   - 既存のネットワークにポリシーを適用する場合:

     ```
     (overcloud) $ openstack network set --qos-policy <policy_name>
     <network_name|network_ID>
     ```

   - 新規ネットワークを作成する場合:

     ```
     (overcloud) $ openstack network create --qos-policy <policy_name> <network_name>
     ```

10.2.1. 帯域幅を制限する QoS ポリシーおよびルールの作成と適用

ネットワーク、ポート、または Floating IP の帯域幅を制限する QoS ポリシーを作成して、指定したレートを超えるトラフィックをすべてドロップすることができます。帯域幅を制限する QoS ポリシーおよびルールを作成して適用するには、以下の手順を実施します。

手順

1. **/etc/neutron/plugins/ml2/<agent_name>_agent.ini** で OpenStack Networking に対する **qos** 拡張機能がまだ有効にされていない場合は、以下の手順を実施します。

   a. カスタム環境ファイルを作成して、以下の設定を追加します。

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

   b. この設定を適用するには、その他の環境ファイルと共にカスタム環境ファイルをスタックに追加して、オーバークラウドをデプロイします。

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

詳しい情報は、**『director のインストールと使用方法』**の「オーバークラウド環境の変更」を参照してください。

2. QoS ポリシーを作成するプロジェクトの ID を特定します。

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

3. **admin** プロジェクトに「bw-limiter」という名前の QoS ポリシーを作成します。

```
(overcloud) $ openstack network qos policy create --share --project
98a2f53c20ce4d50a40dac4a38016c69 bw-limiter
```

4. 「bw-limiter」ポリシーのルールを設定します。

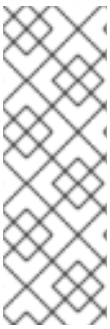
```
(overcloud) $ openstack network qos rule create --type bandwidth-limit --max_kbps 3000 --
max_burst_kbps 300 bw-limiter
```

5. 「bw-limiter」ポリシーを適用するポートを設定します。

```
(overcloud) $ openstack port set --qos-policy bw-limiter <port_name|port_ID>
```

10.2.2. 最小帯域幅を確保する QoS ポリシーおよびルールの作成と適用

物理ネットワーク (physnet) がサポートする **segmentation_type=flat** または **segmentation_type=vlan** が設定されたネットワークのポートに対して、帯域幅の確保を要求することができます。



注記

- 同じ物理インターフェース上で、帯域幅を確保するポートと確保しないポートを混在させないでください。確保しないポートの帯域幅が不足する可能性があるためです。ホストアグリゲートを作成して、帯域幅を確保するポートを確保しないポートから分離します。
- 最小帯域幅を確保する QoS ポリシーを変更できるのは、ポリシーが割り当てられたポートを使用するインスタンスがない場合に限ります。

サポートされているドライバーおよびエージェント

- SR-IOV (sriovnicswitch) vnic_types: direct、macvtap

- ML2/OVS (openvswitch) vnic_types: normal、direct

注記

ML2/OVN は最小帯域幅をサポートしません。

前提条件

- Placement サービスは、マイクロバージョン 1.29 をサポートする必要があります。
- Compute (nova) サービスは、マイクロバージョン 2.72 をサポートする必要があります。
- Networking (neutron) サービスは、以下の API 拡張機能をサポートする必要があります。
  - **agent-resources-synced**
  - **port-resource-request**
  - **qos-bw-minimum-ingress**
- OpenStack CLI を使用して配置情報を照会するには、アンダークラウドに Placement サービスパッケージ **python3-osc-placement** をインストールします。

手順

1. Openstack Networking で Placement サービスプラグインがまだ設定されていない場合には、以下の手順を実施します。
   a. カスタム環境ファイルに **NeutronServicePlugins** がすでに設定されている場合には、パラメーターを更新して「placement」を含めます。そうでなければ、カスタム環境ファイルを作成して以下の設定を追加します。

```
parameter_defaults:
 NeutronServicePlugins: 'router,qos,segments,trunk,placement'
```

   b. この設定を適用するには、その他の環境ファイルと共にカスタム環境ファイルをスタックに追加して、オーバークラウドをデプロイします。

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

   詳しい情報は、**『director のインストールと使用方法』**の「オーバークラウド環境の変更」を参照してください。

2. (オプション) **vnic_types** をブラックリストに登録するには (複数の ML2 メカニズムドライバーがデフォルトでサポートし、複数のエージェントが Placement により追跡されている場合)、**/etc/neutron/plugins/ml2/ml2_conf.ini** に **vnic_type_blacklist** を追加して、エージェントを再起動します。

```
[ovs_driver]
vnic_type_blacklist = direct
[sriov_driver]
#vnic_type_blacklist = direct
```

3. 最小帯域幅を提供する必要のある各コンピュートノードの該当するエージェントに対して、リソースプロバイダーの受信および送信帯域幅を設定します。以下の形式を使用して、受信もしくは送信のみ、またはその両方を設定することができます。

   - 送信帯域幅だけを設定する場合 (kbps 単位):

     ```
     resource_provider_bandwidths = <bridge0>:<egress_kbps>:,<bridge1>:
     <egress_kbps>:,...,<bridgeN>:<egress_kbps>:
     ```

   - 受信帯域幅だけを設定する場合 (kbps 単位):

     ```
     resource_provider_bandwidths = <bridge0>::<ingress_kbps>,<bridge1>::
     <ingress_kbps>,...,<bridgeN>::<ingress_kbps>
     ```

   - 送信および受信帯域幅の両方を設定する場合 (kbps 単位):

     ```
     resource_provider_bandwidths = <bridge0>:<egress_kbps>:<ingress_kbps>,<bridge1>:
     <egress_kbps>:<ingress_kbps>,...,<bridgeN>:<egress_kbps>:<ingress_kbps>
     ```

     以下に例を示します。

     - OVS エージェント用にリソースプロバイダーの受信および送信帯域幅を設定するには、**/etc/neutron/plugins/ml2/openvswitch_agent.ini** に **resource_provider_bandwidths** を追加します。

       ```
       [ovs]
       bridge_mappings = physnet0:br-physnet0
       resource_provider_bandwidths = br-physnet0:10000000:10000000
       ```

     - SRIOV エージェント用にリソースプロバイダーの受信および送信帯域幅を設定するには、**/etc/neutron/plugins/ml2/sriov_agent.ini** に **resource_provider_bandwidths** を追加します。

       ```
       [sriov_nic]
       physical_device_mappings = physnet0:ens5,physnet0:ens6
       resource_provider_bandwidths =
       ens5:40000000:40000000,ens6:40000000:40000000
       ```

   リソースプロバイダーの帯域幅を実装するには、設定したエージェントを再起動します。

4. QoS ポリシーを作成するプロジェクトの ID を特定します。

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

5. **admin** プロジェクトに「guaranteed_min_bw」という名前の QoS ポリシーを作成します。

```
(overcloud) $ openstack network qos policy create --share --project
98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

6. 「guaranteed_min_bw」ポリシーのルールを設定します。

```
(overcloud) $ openstack network qos rule create --type minimum-bandwidth --min-kbps
40000000 --ingress guaranteed_min_bw
(overcloud) $ openstack network qos rule create --type minimum-bandwidth --min-kbps
40000000 --egress guaranteed_min_bw
```

7. 「guaranteed_min_bw」ポリシーを適用するポートを設定します。

```
(overcloud) $ openstack port set --qos-policy guaranteed_min_bw <port_name|port_ID>
```

検証

1. 利用可能なすべてのリソースプロバイダーを一覧表示します。

```
(undercloud) $ openstack --os-placement-api-version 1.17 resource provider list
```

出力例:

```
+--------------------------------------+---------------------------------------------------------+------------+
--------------------------------------+--------------------------------------+
| uuid                                 | name                                                    | generation |
root_provider_uuid                   | parent_provider_uuid                 |
+--------------------------------------+---------------------------------------------------------+------------+
--------------------------------------+--------------------------------------+
| 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | dell-r730-014.localdomain                               |
28 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | None                                 |
| 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | dell-r730-063.localdomain                               |
18 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | None                                 |
| e2f5082a-c965-55db-acb3-8daf9857c721 | dell-r730-063.localdomain:NIC Switch agent
|          0 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | 6b15ddce-13cf-4c85-a58f-
baec5b57ab52 |
| d2fb0ef4-2f45-53a8-88be-113b3e64ba1b | dell-r730-014.localdomain:NIC Switch agent
|          0 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | 31d3d88b-bc3a-41cd-9dc0-
fda54028a882 |
| f1ca35e2-47ad-53a0-9058-390ade93b73e | dell-r730-063.localdomain:NIC Switch
agent:enp6s0f1 |         13 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | e2f5082a-c965-55db-
acb3-8daf9857c721 |
| e518d381-d590-5767-8f34-c20def34b252 | dell-r730-014.localdomain:NIC Switch
agent:enp6s0f1 |         19 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | d2fb0ef4-2f45-53a8-
88be-113b3e64ba1b |
+--------------------------------------+---------------------------------------------------------+------------+
--------------------------------------+--------------------------------------+
```

2. 特定のリソースが提供する帯域幅を確認します。

```
(undercloud) $ openstack --os-placement-api-version 1.17 resource provider inventory list
<rp_uuid>
```

以下の出力例には、dell-r730-014 のインターフェース enp6s0f1 によって提供される帯域幅が示されています。

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 resource provider inventory list e518d381-d590-5767-8f34-c20def34b252
---------------------------------------------------------------------------------------------------
| resource_class              | allocation_ratio | min_unit |    max_unit | reserved | step_size |
total |
---------------------------------------------------------------------------------------------------
| NET_BW_EGR_KILOBIT_PER_SEC |              1.0 |        1 | 2147483647 |        0 |         1 |
10000000 |
| NET_BW_IGR_KILOBIT_PER_SEC |              1.0 |        1 | 2147483647 |        0 |         1 |
10000000 |
---------------------------------------------------------------------------------------------------
```

3. インスタンス実行時のリソースプロバイダーに対する要求を確認するには、以下のコマンドを実行します。

```
(undercloud) $ openstack --os-placement-api-version 1.17 resource provider show --
allocations  <rp_uuid>
```

出力例:

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 resource provider
show --allocations  e518d381-d590-5767-8f34-c20def34b252 -f value -c allocations
{3cbb9e07-90a8-4154-8acd-b6ec2f894a83: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 8848b88b-4464-443f-bf33-5d4e49fd6204: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 9a29e946-698b-4731-bc28-89368073be1a: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, a6c83b86-9139-4e98-9341-dc76065136cc: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}, da60e33f-156e-47be-a632-870172ec5483: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, eb582a0e-8274-4f21-9890-9a0d55114663: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}}
```

10.2.3. 送信トラフィックの DSCP マーキング

differentiated services code point (DSCP) を使用すると、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 quality-of-service (QoS) ポリシー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ます。現在、DSCP は Open vSwitch (OVS) を使用する VLAN およびフラットプロバイダーネットワークでのみ利用することができます。

ポリシーを作成し、DSCP ルールを定義し、そのルールをポリシーに適用するには、以下の例に示す手順を実施します。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。

1. 新規 QoS ポリシーを作成します。

```
openstack network qos policy create --project 98a2f53c20ce4d50a40dac4a38016c69
qos_policy_name
```

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

```
openstack network qos rule create --type dscp-marking --dscp-mark 18 qos_policy_name
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。

```
openstack network qos 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。

```
openstack network qos rule show qos-web-servers d7f976ec-7fab-4e60-af70-f59bf88198e6
+-----------+--------------------------------------+
| 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。

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

6. DSCP ルールを削除します。

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

10.2.4. QoS ポリシーおよびルール適用の確認方法

以下のコマンドを使用して、Quality of Service (QoS) ポリシーおよびルールの作成および適用を確認します。

アクション	コマンド
利用可能な QoS ポリシーを一覧表示する	**$ openstack network qos policy list**

アクション	コマンド
特定の QoS ポリシーの詳細を表示する	**$ openstack network qos policy show <policy_name>**
利用可能な QoS ルールを一覧表示する	**$ openstack network qos rule type list**
特定の QoS ポリシーのルールを一覧表示する	**$ openstack network qos rule list <policy_name>**
特定のルールの詳細を表示する	**$ openstack network qos rule type show <rule_id>**
利用可能なポートを一覧表示する	**$ openstack port list**
特定のポートの詳細を表示する	**$ openstack port show <port_ID/port_name>**

10.3. QOS ポリシーの RBAC

quality-of-service (QoS) ポリシーにロールベースのアクセス制御 (RBAC) を追加することができます。これにより、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。**bw-limiter** ポリシーをプロジェクト **demo** に割り当てるには、以下のコマンドを実行します。

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

第11章 ブリッジマッピングの設定

本章では、Red Hat OpenStack Platform でのブリッジマッピングの設定について説明します。

11.1. ブリッジマッピングの概要

ブリッジマッピングは、物理ネットワーク名 (インターフェースラベル) を OVS または OVN で作成したブリッジに関連付けます。以下の例では、物理名 (datacentre) が外部ブリッジ (br-ex) にマッピングされます。

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

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 OVS の場合、**br-int** と **br-ex** 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。OVN の場合は、ポートを必要とするハイパーバイザーに仮想マシンがバインドされている場合に限り、ハイパーバイザーにパッチポートが作成されます。

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に、ブリッジマッピングを設定します。ルータートラフィックは、正しい物理ネットワーク (プロバイダーネットワーク) を使用して外部に送信されます。

11.2. トラフィックの流れ

それぞれの外部ネットワークは内部 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 タグを取り除き、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、**phy-br-ex <-> int-br-ex** を使用して **br-int** に移動します。パケットが **br-ex** から **br-int** に移動する際に、パケットの外部 vlan ID は **br-int** で内部 vlan タグに置き換えられます。これにより、**qg- xxx** がパケットを受け入れることができます。

送信パケットの場合は、パケットの内部 vlan タグは br-ex (または **network_vlan_ranges** パラメーターで定義される外部ブリッジ) で外部 vlan タグに置き換えられます。

11.3. ブリッジマッピングの設定

Red Hat OpenStack Platform (RHOSP) director は、事前定義された NIC テンプレートを使用して、オーバークラウドをインストールし初期ネットワーク設定を定義します。

カスタマイズした環境ファイルの **NeutronBridgeMappings** パラメーターを使用して、ブリッジマッピング等の初期ネットワーク設定の項目をカスタマイズすることができます。この環境ファイルを **openstack overcloud deploy** コマンドで指定します。

前提条件

- 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に、ブリッジマッピングを設定する必要があります。
- ML2/OVS および ML2/OVN DVR 構成の両方について、コンピュートノードにもブリッジマッピングを設定する必要があります。

手順

1. カスタム環境ファイルを作成し、実際の環境に適した値で **NeutronBridgeMappings** heat パラメーターを追加します。

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

NeutronBridgeMappings heat パラメーターは、物理名 (**datacentre**) をブリッジ (**br-ex**) に関連付けます。

注記

NeutronBridgeMappings パラメーターを使用しないと、デフォルトではホストの外部ブリッジ (br-ex) を物理名 (datacentre) にマッピングします。

2. この設定を適用するには、その他の環境ファイルと共にカスタム環境ファイルをスタックに追加して、オーバークラウドをデプロイします。

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

3. 続いて、以下の手順を実施します。

   a. ネットワークの VLAN 範囲を使用して、対応する外部ネットワークを表すプロバイダーネットワークを作成します。(neutron プロバイダーネットワークまたは Floating IP ネットワークを作成する際には、物理名を使用します。)

   b. ルーターインターフェースを使用して、外部ネットワークをプロジェクトネットワークに接続します。

関連情報

- 『オーバークラウドの **高度なカスタマイズ**』 の「 ネットワーク環境パラメーター 」
- 『 オーバークラウドの **高度なカスタマイズ**』 の「オーバークラウド作成時の環境ファイル の追加」

11.4. OVS ブリッジマッピングのメンテナンス

OVS ブリッジマッピングを削除したら、引き続きクリーンアップ操作を行い、ブリッジ設定から関連付けられたパッチポートのエントリーが消去されている状態にする必要があります。この操作は以下の手順により実施することができます。

- 手動ポートクリーンアップ: 不要なパッチ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。
- 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、必要なブリッジマッピングを再度追加する必要があります。ネットワーク接続の停止を許容できる場合には、計画的なメンテナンス期間中にこのオプションを選択します。

注記

OVN ブリッジマッピングが削除されると、OVN コントローラーは自動的に関連付けられたパッチポートのクリーンアップを行います。

11.4.1. OVS パッチポートの手動クリーンアップ

OVS ブリッジマッピングを削除したら、関連付けられたパッチポートも削除する必要があります。

前提条件

- クリーンアップを行うパッチポートは、Open Virtual Switch (OVS) ポートでなければなりません。
- パッチポートの手動クリーンアップを行うのに、システムを停止する必要は **ありません**。
- クリーンアップを行うパッチポートは、命名規則により特定することができます。
  - **br-$external_bridge** では、パッチポートは **phy-<external bridge name>** と命名されます (例: phy-br-ex2)。
  - **br-int** では、パッチポートは **int-<external bridge name>** と命名されます (例: **int-br-ex2**)。

手順

1. **ovs-vsctl** を使用して、削除したブリッジマッピングのエントリーに関連付けられた OVS パッチポートを削除します。

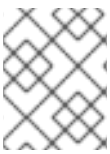
```
# ovs-vsctl del-port br-ex2 datacentre
# ovs-vsctl del-port br-tenant tenant
```

2. **neutron-openvswitch-agent** を再起動します。

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

11.4.2. OVS パッチポートの自動クリーンアップ

OVS ブリッジマッピングを削除したら、関連付けられたパッチポートも削除する必要があります。



注記

OVN ブリッジマッピングが削除されると、OVN コントローラーは自動的に関連付けられたパッチポートのクリーンアップを行います。

前提条件

- クリーンアップを行うパッチポートは、Open Virtual Switch (OVS) ポートでなければなりません。
- **neutron-ovs-cleanup** コマンドでパッチポートの自動クリーンアップを行うと、ネットワーク接続が停止します。したがって、この操作は計画的なメンテナンス期間中にのみ実施する必要があります。

- **--ovs_all_ports** フラグを使用して **br-int** から全パッチポートを削除すると、**br-tun** からトンネルエンドが、またブリッジ間からはパッチポートがクリーンアップされます。
- **neutron-ovs-cleanup** コマンドは、すべての OVS ブリッジから全パッチポート (インスタンス、qdhcp/qrouter 等) を抜線します。

手順

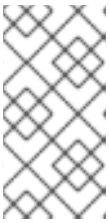
1. **--ovs_all_ports** フラグを指定して **neutron-ovs-cleanup** コマンドを実行します。

重要

このステップを実施すると、ネットワーク接続が完全に停止されます。

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

2. オーバークラウドを再デプロイして接続を回復します。
openstack overcloud deploy コマンドを再実行すると、ブリッジマッピングの値が再適用されます。



注記

再起動後、OVS エージェントは bridge_mappings に存在しない接続に干渉しません。したがって、**br-int** が **br-ex2** に接続され、**br-ex2** にフローがある場合、bridge_mappings 設定から **br-int** を削除しても、OVS エージェントまたはノードの再起動時に 2 つのブリッジが切断されることはありません。

関連情報

- 『オーバークラウドの **高度なカスタマイズ**』 の「 ネットワーク環境パラメーター 」
- 『 オーバークラウドの **高度なカスタマイズ**』 の「オーバークラウド作成時の環境ファイル の追加」

第12章 VLAN 対応のインスタンス

12.1. VLAN 対応インスタンスの概要

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することができます。このことは、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。単一の仮想 NIC で複数の顧客/サービスに対応することができるためです。

たとえば、プロジェク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ます。したがって、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訳ではなく、インスタンスに注入される直前にタグ付けされます。

VLAN のタグが付いたトラフィックを実装するには、親ポートを作成して、新しいポートを既存の neutron ネットワークにアタッチします。新しいポートをアタッチすると、OpenStack Networking は作成した親ポートにトランク接続を追加します。次にサブポートを作成します。これらのサブポートは VLAN とインスタンスを接続し、トランクへの接続を確立することができます。インスタンスのオペレーティングシステム内で、サブポートに関連付けられた 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も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

Red Hat OpenStack のデプロイメント時に、トランクプラグインがデフォルトで有効になっています。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。

- コントローラーノード上で、/var/lib/config-data/neutron/etc/neutron/neutron.conf ファイルでトランクプラグインが有効であることを確認します。

```
service_plugins=router,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、トランキングされた VLAN への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 **public** ネットワークです。

```
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ネットワークにアタッチします。以下の例では、**public** ネットワーク上に parent-trunk-port という名前の neutron ポートを作成します。このトランクは、**サブポート** の作成に使用することができるので、**親** ポートです。

```
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. ステップ 2 で作成したポートを使用してトランクを作成します。以下の例では、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 |              |
+--------------------------------------+-------------+--------------------------------------+-------------+
```

5. トランク接続の詳細を表示します。

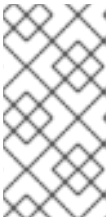
```
openstack network trunk show parent-trunk
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | UP                                   |
| created_at      | 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です。親ポートに割り当てた MAC アドレスも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パブリックネットワークを、サブポートにはプライベートネットワークを使用しています。

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。サブポートの作成ステップ中に、特定の MAC アドレスを使用するようにサブポートを設定することもできます。

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 **port-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. トランクの状態について

- **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。プロビジョニング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して、トランクを正常な状態に戻します。**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 ポリシーの設定

13.1. RBAC ポリシーの概要

OpenStack Networking のロールベースアクセス制御 (RBAC) ポリシーにより、細かな粒度で **neutron** 共有ネットワークを制御することができます。OpenStack Networking は RBAC テーブルを使用してプロジェクト間における **neutron** ネットワークの共有を制御します。これにより、管理者はインスタンスをネットワークにアタッチする権限が付与されるプロジェクトを管理することができます。

その結果、クラウド管理者は、一部のプロジェクトからネットワーク作成機能を削除することや、逆にそのプロジェクトに対応した既存ネットワークへの接続を許可することが可能です。

13.2. RBAC ポリシーの作成

以下の手順では、ロールベースのアクセス制御 (RBAC) ポリシーを使用して、プロジェク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。

```
# openstack network list
+--------------------------------------+-------------+--------------------------------------------------------+
| id                                   | name        | subnets                                                |
+--------------------------------------+-------------+--------------------------------------------------------+
| fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 | web-servers | 20512ffe-ad56-4bb4-b064-2cb18fecc923 192.168.200.0/24 |
| bcc16b34-e33e-445b-9fde-dd491817a48a | private     | 7fe4a05a-4b81-4a59-8c47-82c965b0e050 10.0.0.0/24      |
| 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 | public      | 2318dc3b-cff0-43fc-9489-7d4cf48aaab9 172.24.4.224/28  |
+--------------------------------------+-------------+--------------------------------------------------------+
```

2. プロジェクトの一覧を表示します。

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

3. **web-servers** ネットワークの RBAC エントリーを作成し、 **auditors** プロジェクト (**4b0b98f8c6c040f38ba4f7146e8680f5**) にアクセスを許可します。

```
# openstack network rbac create --type network --target-project 4b0b98f8c6c040f38ba4f7146e8680f5 --action access_as_shared web-servers
Created a new rbac_policy:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_shared                     |
| id            | 314004d0-2261-4d5e-bda7-0181fcf40709 |
```

```
| object_id      | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type    | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| project_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+----------------+--------------------------------------+
```

これにより、**auditors** プロジェクトのユーザーは、インスタンスを **web-servers** ネットワークに接続することができます。

13.3. RBAC ポリシーの確認

1. **openstack network rbac list** コマンドを実行して、既存のロールベースアクセス制御 (RBAC) ポリシーの ID を取得します。

```
# openstack network rbac list
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. **openstack network rbac-show** コマンドを実行して、特定の RBAC エントリーの詳細を表示します。

```
# openstack network rbac show 314004d0-2261-4d5e-bda7-0181fcf40709
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| action         | access_as_shared                     |
| id             | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id      | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type    | network                              |
| target_project | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| project_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+----------------+--------------------------------------+
```

13.4. RBAC ポリシーの削除

1. **openstack network rbac list** コマンドを実行して、既存のロールベースアクセス制御 (RBAC) ポリシーの ID を取得します。

```
# openstack network rbac list
+--------------------------------------+-------------+--------------------------------------+
| id                                   | object_type | object_id                            |
+--------------------------------------+-------------+--------------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-b81a-4572-9c7f-
7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-fee8-43da-bb99-
032e4aaf3f85 |
+--------------------------------------+-------------+--------------------------------------+
```

2. 削除する RBAC の ID を指定して **openstack network rbac delete** コマンドを実行し、RBAC を削除します。

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

13.5. 外部ネットワークへの RBAC ポリシーアクセスの付与

--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) へのロールベースアクセス制御 (RBAC) ポリシーによるアクセスを許可することができます。

web-servers ネットワークの RBAC を作成し、エンジニアリングプロジェクト (c717f263785d4679b16a122516247deb) にアクセスを許可するには、以下の手順例のステップを実行します。

- **--action access_as_external** オプションを使用して、新しい RBAC ポリシーを作成します。

```
# openstack network rbac create --type network --target-project
c717f263785d4679b16a122516247deb --action access_as_external web-servers
 Created a new rbac_policy:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| action         | access_as_external                   |
| id             | ddef112a-c092-4ac1-8914-c714a3d3ba08 |
| object_id      | 6e437ff0-d20f-4483-b627-c3749399bdca |
| object_type    | network                              |
| target_project | c717f263785d4679b16a122516247deb     |
| project_id     | c717f263785d4679b16a122516247deb     |
+----------------+--------------------------------------+
```

上記のコマンドを実行した結果、エンジニアリングプロジェクトのユーザーは、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。

```
$ openstack network 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

14.1. 分散仮想ルーター (DVR) について

Red Hat OpenStack Platform をデプロイする場合、集中ルーティングモデルまたは DVR のどちらかを選択することができます。

それぞれのモデルには短所と長所があります。本項を使用して、集中ルーティングと DVR のどちらがよりニーズに適しているかを慎重に検討してください。

新規の ML2/OVN デプロイメントではデフォルトで DVR が有効化され、新規の ML2/OVS デプロイメントではデフォルトで無効化されています。OpenStack Networking (neutron) API 用の Heat テンプレート (**deployment/neutron/neutron-api-container-puppet.yaml**) には、分散仮想ルーティング (DVR) を有効化/無効化するためのパラメーターが含まれています。DVR を無効にするには、環境ファイルで以下のように設定します。

```
parameter_defaults:
  NeutronEnableDVR: false
```

14.1.1. 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を通じて他のインスタンスと通信することができます。ルーターを作成して、プロジェク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)。

14.1.2. フローのルーティング

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**: 同じプロジェク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適用されます。

- **Floating IP を使用した North-South ルーティング**: Floating IP のアドレス指定は 1 対 1 の NAT で、変更およびインスタンス間の移動が可能です。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、ルーターに外部接続を提供するアップリンクネットワークから取得されます。したがって、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、インスタンスはインターネット上の Web サイトにアクセスすることができますが、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 サイトにアクセスすることができません。SNAT は、IPv4 トラフィックにのみ適用されます。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。

14.1.3. 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) にデプロイされます。したがって、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ります。このことが、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でのみ利用できるので、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ます。そのため、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。

14.2. DVR の概要

分散仮想ルーティング (DVR) は、集中ルーティングとは別のルーティング設計を提供します。DVR 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します。DVR には以下の特徴があります。

- 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。そのためには、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。

- ノードが SNAT トラフィックだけに対応するように、コントローラーノード上の L3 エージェントは **dvr_snat** モードを使用します。

- 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、すべての分散ルーター上でホストされます。

14.3. DVR に関する既知の問題および注意

注記

Red Hat OpenStack Platform 16.0 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 のコアプラグインと Open vSwitch (OVS) メカニズムドライバーの組み合わせまたは ML2/OVN メカニズムドライバーに制限されます。他のバックエンドはサポートされません。
- OVS DVR のデプロイメントでは、Red Hat OpenStack Platform Load-balancing サービス (octavia) のネットワークトラフィックは、コンピュートノードではなくコントローラーノードおよびネットワークノードを通過します。
- DVR が有効であっても、SNAT (送信元ネットワークアドレス変換) トラフィックは分散されません。SNAT は機能しますが、すべての送信/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、すべての送信/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。広範囲に渡って IPv6 ルーティングを使用する場合、DVR を使用しないでください。
- 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 16.0 director で DVR を使用すると、L3 HA は無効になります。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 エージェント間で負荷が共有され) ますが、エージェントの 1 つが機能しなくなると、このエージェントがホストするすべてのルーターも機能しなくなります。この影響を受けるのは SNAT トラフィックだけです。このような場合には、1 つのネットワークノードに障害が発生してもルーターが別のノードに再スケジュールされるように、**allow_automatic_l3agent_failover** 機能を使用することが推奨されます。
- 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。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 を使用する場合は、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、各コンピュートノードに追加の IP アドレスを 1 つ設定する必要があります。これは、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す。
- 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 機能を有効にする必要があります。Red Hat OpenStack Platform director は、インストール時に L2 Population を強制的に有効にします。

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 15
- 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 環境ファイルには、必要な DVR 固有のパラメーターの設定が含まれます。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を、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。

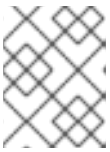
B. コンピュートノードおよびコントローラーノードでブリッジを作成して、外部ネットワークトラフィック用のインターフェースを設定すること。

C. X がこのブリッジを使用するのを許可するように Neutron を設定すること。

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 (C.) も設定する必要があります。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。

通常のデフォルト設定を使用する概念実証用の環境は、以下の例のようになります。

1. **environments/neutron-ovs-dvr.yaml** ファイルで **OS::TripleO::Compute::Net::SoftwareConfig** の値が現在の **OS::TripleO::Controller::Net::SoftwareConfig** の値と同じであることを確認します。
通常、この値は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 (例: **environments/net-multiple-nics.yaml**) に含まれます。この値により、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。



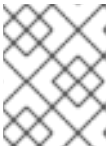
注記

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する場合には、これらのファイルに適切な設定を追加しなければならない場合があります。

2. オーバークラウドのデプロイ時に、**environments/neutron-ovs-dvr.yaml** ファイルをデプロイコマンドに追加します。

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

3. 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、特別なカスタマイズが必要なテスト環境) では、環境の例を指針として利用することができます。L2 エージェントが使用するブリッジマッピング種別のパラメーターや、他のエージェント (例: L3 エージェント) の外部向けブリッジへの参照が正しいことを確認します。



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. director のスタック更新を実行して DVR を設定します。
   詳細は、「ML2 OVS を使用した DVR のデプロイ」を参照してください。
3. 既存のルーターでルーティングが正常に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。

a. ルーターを無効にします。

```
$ openstack router set --disable
```

b. 高可用性の設定を無効にします。

```
$ openstack router set --no-ha
```

c. ルーターが DVR を使用するように設定します。

```
$ openstack router set --distributed
```

a. ルーターを有効にします。

```
$ openstack router set --enable
```

b. 分散ルーティングが正常に機能していることを確認します。

第15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 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、負荷分散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

Red Hat では、Neutron-LBaaS から Octavia への移行パスをサポートしていません。ただし、サポート対象外のオープンソースツールを利用することが可能です。詳細は、「nlbaas2octavia-lb-replicator」を参照してください。

15.1. 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、負荷分散の管理ネットワーク (**lb-mgmt-net**) 上で行われます。

Octavia には以下のサービスが含まれています。

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。

重要

PING タイプのヘルスモニターは、メンバーが到達可能で ICMP エコーリクエストに応答するかどうかを確認します。**PING** は、そのインスタンスで実行中のアプリケーションが正常かどうかを検出しません。**PING** は、ICMP エコーリクエストが有効なヘルスチェックである特定のケースでのみ使用してください。

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

負荷分散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します。

Amphora

負荷分散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、メンバーの設定に応じた負荷分散パラメーターにより設定されます。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

負荷が分散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し、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る場合があります。

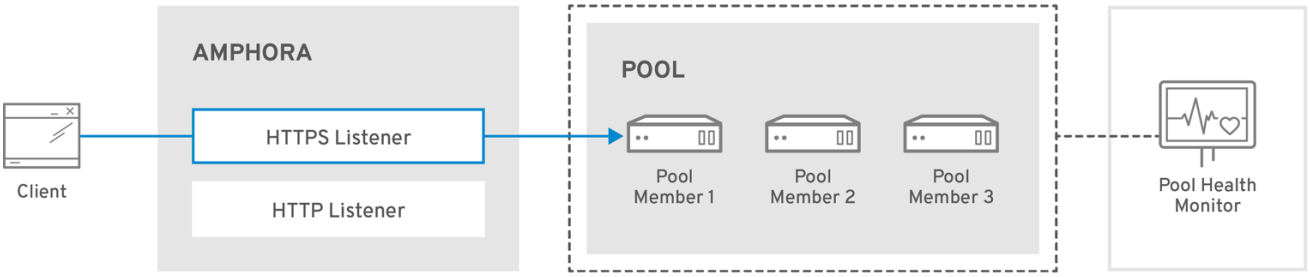
プール

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コンピュートインスタンス

以下の図には、プールメンバーへの HTTPS トラフィックフローを示しています。



15.2. OCTAVIA に関するソフトウェア要件

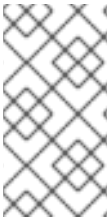
Octavia には、以下の OpenStack コアコンポーネントの設定が必要です。

- Compute (nova)
- Networking (**allowed_address_pairs** を有効化)
- Image (glance)
- Identity (keystone)
- RabbitMQ
- MySQL

15.3. アンダークラウドの前提条件

本項の前提は以下のとおりです。

- 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。
- 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る。
- Octavia はコントローラーノードで実行される。



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、『**コンテナー化されたサービスへの移行**』を参照してください。

15.3.1. Octavia 機能のサポートマトリックス

表15.1 Octavia 機能のサポートマトリックス

機能	RHOSP 16.0 でのサポートレベル
ML2/OVS L3 HA	フルサポート
ML2/OVS DVR	フルサポート
ML2/OVS L3 HA とコンポーザブルネットワークノードの組み合わせ [1]	フルサポート
ML2/OVS DVR とコンポーザブルネットワークノードの組み合わせ [1]	フルサポート
ML2/OVN L3 HA	フルサポート
ML2/OVN DVR	フルサポート
amphora アクティブ/スタンバイ	**テクノロジープレビューとしてのみ提供**: 13.0 および **すべての** メンテナンスリリース
HTTPS 終端ロードバランサー	フルサポート
amphora 予備プール	テクノロジープレビューとしてのみ提供
UDP	テクノロジープレビューとしてのみ提供
バックアップメンバー	テクノロジープレビューとしてのみ提供
プロバイダーフレームワーク	テクノロジープレビューとしてのみ提供
TLS クライアント認証	テクノロジープレビューとしてのみ提供
TLS バックエンド暗号化	テクノロジープレビューとしてのみ提供
Octavia フレーバー	フルサポート
オブジェクトタグ	テクノロジープレビューとしてのみ提供
リスナー API タイムアウト	フルサポート
ログのオフロード	テクノロジープレビューとしてのみ提供
仮想 IP アクセス制御リスト	フルサポート
ボリュームベースの amphora	サポートなし

[1] ネットワークノードと OVS、メタデータ、DHCP、L3、および Octavia (ワーカー、ヘルスモニター、ハウスキーピング) の組み合わせ

15.4. OCTAVIA デプロイメントのプランニング

Red Hat OpenStack Platform は、Load-balancing 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。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。

- 証明書と鍵を設定します。
- 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負荷分散管理ネットワークを設定します。

注記

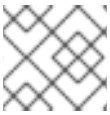
OpenStack Heat テンプレートは直接編集しないでください。カスタムの環境ファイル (例: **octavia-environment.yaml**) を作成して、デフォルトのパラメーター値をオーバーライドしてください。

Amphora イメージ

事前にプロビジョニングされたサーバーでは、octavia をデプロイする前に、アンダークラウドに amphora イメージをインストールする必要があります。

```
$ sudo dnf install octavia-amphora-image-x86_64.noarch
```

事前にプロビジョニングされていないサーバーでは、Red Hat OpenStack Platform 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。スタックの更新またはアップグレード中に、director は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。

15.4.1. Octavia の証明書と鍵の設定

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。独自の証明書および鍵を指定するか、Red Hat OpenStack Platform director による自動生成を許可することができます。director が必要なプライベート認証局を自動的に作成し、必要な証明書を発行するのを許可することを推奨します。

独自の証明書および鍵を使用する必要がある場合は、以下の手順を実施します。

1. **openstack overcloud deploy** コマンドを実行するマシンで、カスタム YAML 環境ファイルを作成します。

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

2. YAML 環境ファイルに以下のパラメーターを追加し、ご自分のサイトに適した値を設定します。

- **OctaviaCaCert**:
  Octavia が証明書を生成するのに使用する CA の証明書

- **OctaviaCaKey**:
  生成された証明書の署名に使用するプライベート CA 鍵

- **OctaviaClientCert**:
  コントローラー用に Octavia CA が発行するクライアント証明書および暗号化されていない鍵

- **OctaviaCaKeyPassphrase**:
  上記のプライベート CA 鍵で使用するパスフレーズ

- **OctaviaGenerateCerts**:
  証明書および鍵の自動生成の有効 (true) または無効 (false) を director に指示するブール値

以下に例を示します。

注記

証明書と鍵の値は複数行にまたがるので、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。

```
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-----[

  OctaviaClientCert: |
    -----BEGIN CERTIFICATE-----

MIIDmjCCAoKgAwIBAgIBATANBgkqhkiG9w0BAQsFADBcMQswCQYDVQQGEwJVUzEP

    [snip]
    270l5ILSnfejLxDH+vI=
    -----END CERTIFICATE-----
    -----BEGIN PRIVATE KEY-----

MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDU771O8MTQV8RY

    [snip]
    KfrjE3UqTF+ZaaIQaz3yayXW
    -----END PRIVATE KEY-----

  OctaviaCaKeyPassphrase:
```

```
  b28c519a-5880-4e5e-89bf-c042fc75225d

OctaviaGenerateCerts: false
[rest of file snipped]
```

警告

director が生成するデフォルトの証明書を使用し、**OctaviaGenerateCerts** パラメーターを **false** に設定した場合、証明書は有効期限が切れても自動的に更新されません。

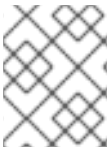
15.5. OCTAVIA のデプロイ

Red Hat OpenStack Platform (RHOSP) director を使用して、Octavia をデプロイします。director は、heat テンプレートを使用して Octavia (およびその他の RHOSP コンポーネント) をデプロイします。

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。イメージレジストリー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 」セクション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
```

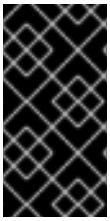


注記

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。

15.6. OCTAVIA のデフォルト設定の変更

Octavia をデプロイする際に director が使用するデフォルトパラメーターをオーバーライドするには、1 つまたは複数の YAML 形式のカスタム環境ファイルで専用の値を指定することができます (例: **octavia-environment.yaml**)。



重要

ベストプラクティスとしては、必ず Octavia の設定変更を適切な Heat テンプレートに加え、Red Hat OpenStack Platform director を再実行します。個々のファイルを手動で変更し、director を使用しない場合、アドホックの設定変更が失われるリスクがあります。

director が Octavia のデプロイおよび設定に使用するパラメーターは、非常に明示的です。以下にいくつか例を示します。

- **OctaviaControlNetwork**
  amphora 管理ネットワークに使用する neutron ネットワークの名前

- **OctaviaControlSubnetCidr**
  amphora 管理サブネット用のサブネット (CIDR 形式)

- **OctaviaMgmtPortDevName**
  octavia ワーカー/ヘルスマネージャーと amphora マシン間の通信に使用される octavia 管理ネットワークインターフェースの名前

- **OctaviaConnectionLogging**
  負荷分散インスタンス (amphora) の接続フローのロギングを有効 (true) および無効にするブール値。amphora ではログローテーションが設定されているため、ログでディスクが埋め尽くされることはほとんどありません。無効にすると、パフォーマンスに若干の影響が出ます。

director の使用する Octavia パラメーターの一覧は、アンダークラウドの以下のファイルを参照してください。

```
/usr/share/openstack-tripleo-heat-templates/deployment/octavia/octavia-deployment-config.j2.yaml
```

ご自分の環境ファイルには、**parameter_defaults:** というキーワードを含める必要があります。**parameter_defaults:** のキーワードの後に、ご自分のパラメーター/値ペアを定義します。以下は例です。

```
parameter_defaults:
  OctaviaMgmtPortDevName: "o-hm0"
  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'
```

ヒント

YAML ファイルでは、ファイル内でパラメーターがどこに置かれるかが非常に重要です。必ず **parameter_defaults:** は 1 列目から書き始め (先頭にスペースを置かない)、パラメーター/値のペアは 5 列目から書き始めてください (各パラメーターの前にスペースを 4 つ置く)。

15.7. アクセス制御リストを使用したロードバランサーの保護

Octavia API を使用してアクセス制御リスト (ACL) を作成し、リスナーへの受信トラフィックを、許可されたソース IP アドレスのセットに制限することができます。それ意外の受信トラフィックは、すべて拒否されます。

前提条件

本項では、Octavia ロードバランサーの保護方法を説明する上で、以下のことを前提としています。

- バックエンドサーバー 192.0.2.10 および 192.0.2.11 は **private-subnet** という名前のサブネット上にあり、TCP ポート 80 にカスタムアプリケーションが設定されている。

- サブネット **public-subnet** はクラウドオペレーターが作成した共有外部サブネットで、インターネットからアクセスすることができる。

- ロードバランサーはインターネットからアクセス可能な基本的なロードバランサーで、リクエストをバックエンドサーバーに分散する。

- TCP ポート 80 のアプリケーションには、限られたソース IP アドレス (192.0.2.0/24 および 198.51.100/24) からしかアクセスすることができない。

手順

1. サブネット (**public-subnet**) にロードバランサー (**lb1**) を作成します。

注記

丸かっこ () 内の名前を、実際に使用される名前に置き換えてください。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
```

2. ロードバランサー (**lb1**) のステータス表示が **ACTIVE** および **ONLINE** になるまで、以下のコマンドを再実行します。

```
$ openstack loadbalancer show lb1
```

3. 許可される CIDR でリスナー (**listener1**) を作成します。

```
$ openstack loadbalancer listener create --name listener1 --protocol TCP --protocol-port 80 --allowed-cidr 192.0.2.0/24 --allowed-cidr 198.51.100/24 lb1
```

4. リスナー (**listener1**) のデフォルトプール (**pool1**) を作成します。

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener listener1 --protocol TCP
```

5. サブネット (**private-subnet**) 上のメンバー 192.0.2.10 および 192.0.2.11 を、作成したプール (**pool1**) に追加します。

```
$ 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
```

検証手順

1. 以下のコマンドを入力します。

```
$ openstack loadbalancer listener show listener1
```

以下のような出力が表示されるはずです。

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| connection_limit          | -1                                   |
| created_at                | 2019-12-09T11:38:05                  |
| default_pool_id           | None                                 |
```

```
| default_tls_container_ref   | None                                 |
| description                 |                                      |
| id                          | d26ba156-03c3-4051-86e8-f8997a202d8e |
| insert_headers              | None                                 |
| l7policies                  |                                      |
| loadbalancers               | 2281487a-54b9-4c2a-8d95-37262ec679d6 |
| name                        | listener1                            |
| operating_status            | ONLINE                               |
| project_id                  | 308ca9f600064f2a8b3be2d57227ef8f     |
| protocol                    | TCP                                  |
| protocol_port               | 80                                   |
| provisioning_status         | ACTIVE                               |
| sni_container_refs          | []                                   |
| timeout_client_data         | 50000                                |
| timeout_member_connect      | 5000                                 |
| timeout_member_data         | 50000                                |
| timeout_tcp_inspect         | 0                                    |
| updated_at                  | 2019-12-09T11:38:14                  |
| client_ca_tls_container_ref | None                                 |
| client_authentication       | NONE                                 |
| client_crl_container_ref    | None                                 |
| allowed_cidrs               | 192.0.2.0/24                         |
|                             | 198.51.100/24                        |
+-----------------------------+--------------------------------------+
```

パラメーター **allowed_cidrs** は、192.0.2.0/24 および 198.51.100/24 からのトラフィックだけを許可するように設定します。

2. ロードバランサーが保護されていることを確認するには、**allowed_cidrs** の一覧に記載されていない CIDR のクライアントからリスナーにリクエストの実施を試みます。リクエストは成功しないはずです。以下のような出力が表示されるはずです。

```
curl: (7) Failed to connect to 10.0.0.226 port 80: Connection timed out
curl: (7) Failed to connect to 10.0.0.226 port 80: Connection timed out
curl: (7) Failed to connect to 10.0.0.226 port 80: Connection timed out
curl: (7) Failed to connect to 10.0.0.226 port 80: Connection timed out
```

15.8. 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には、以下の手順を実施します。

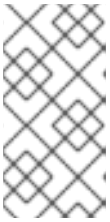
1. サブネット上にロードバランサーを作成します。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id private-subnet
```

2. ロードバランサーの状態を監視します。

```
$ openstack loadbalancer show lb1
```

ACTIVE および ONLINE のステータスが表示されていれば、ロードバランサーが作成され稼働していることを意味しているので、次のステップに進むことができます。



注記

Compute サービス (nova) からロードバランサーのステータスを確認するには、**openstack server list --all | grep amphora** コマンドを使用します。ロードバランサーはコンテナーではなく仮想マシン (VM) なので、その作成プロセスには時間がかかります (ステータスは **PENDING** と表示されます)。

3. リスナーを作成します。

```
$ openstack loadbalancer listener create --name listener1 --protocol HTTP --protocol-port 80
lb1
```

4. 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
```

5. 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
```

6. 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
```

7. public サブネットに Floating IP アドレスを作成します。

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

8. 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_`
```

ヒント

LOAD_BALANCER_VIP_PORT を特定するには、**openstack loadbalancer show lb1** コマンドを実行します。

15.9. ロードバランサーの検証

ロードバランサーを検証するには、以下の手順を実施します。

1. ロードバランサーの設定を確認するには、**openstack loadbalancer show** コマンドを実行します。

```
(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** コマンドを実行します。

```
(overcloud) [stack@undercloud-0 ~]$ openstack loadbalancer amphora list | grep <UUID of loadbalancer lb1>
```

3. amphora の情報を表示するには、amphora の UUID と共に **amphora show** コマンドを実行します。

```
(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                  |
| image_id        | a3f9f3e4-92b6-4a27-91c8-ddc69714da8f |
+-----------------+--------------------------------------+
```

4. リスナーの情報を表示するには、**openstack loadbalancer listener show** コマンドを実行します。

```
(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表示するには、**openstack loadbalancer pool show** コマンドを実行します。

```
(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** コマンドを実行します。

```
(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
```

15.10. TLS 終端 HTTPS ロードバランサーの概要

TLS 終端 HTTPS ロードバランサーが実装されると、Web クライアントは Transport Layer Security (TLS) プロトコルを介してロードバランサーと通信します。ロードバランサーは TLS セッションを終端し、復号化されたリクエストをバックエンドサーバーに転送します。

ロードバランサーで TLS セッションを終端することで、CPU 負荷の高い暗号化操作をロードバランサーにオフロードし、これによりロードバランサーはレイヤー 7 インスペクション等の高度な機能を使用することができます。

15.11. TLS 終端 HTTPS ロードバランサーの作成

以下の手順で、Transport Layer Security (TLS) を介してインターネットからアクセス可能な TLS 終端 HTTPS ロードバランサーを作成し、暗号化されていない HTTP プロトコルを通じてリクエストをバックエンドサーバーに分散する方法を説明します。

前提条件

- TCP ポート 80 でセキュアではない HTTP アプリケーションをホストするバックエンドサーバーが含まれるプライベートサブネット
- インターネットからアクセス可能な共有外部 (パブリック) サブネット
- TLS 公開鍵の暗号化が以下のように設定されている。
  - ロードバランサーの仮想 IP アドレス (例: www.example.com) に割り当てられた DNS 名用に、TLS 証明書、鍵、および中間証明書チェーンが外部認証局 (CA) から取得される。
  - 証明書、鍵、および中間証明書チェーンが、現在のディレクトリー内の個別ファイルに保存される。
  - 鍵および証明書は PEM 形式でエンコードされる。
  - 鍵はパスフレーズで暗号化されない。
  - 中間証明書チェーンには PEM 形式でエンコードされた複数の証明書が含まれ、チェーンを形成する。
- Key Manager サービス (barbican) を使用するように Load-balancing サービス (octavia) が設定されている。詳しい情報は、『Manage Secrets with OpenStack Key Manager』 を参照してください。

手順

1. 鍵 (**server.key**)、証明書 (**server.crt**)、および中間証明書チェーン (**ca-chain.crt**) を 1 つの PKCS12 ファイル (**server.p12**) に組み合わせます。

注記

丸かっこ内の値は例として示しています。これらの例を実際のサイトに適した値に置き換えてください。

```
$ openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.crt -passout pass: -out server.p12
```

2. Key Manager サービスを使用して、PKCS12 ファイルのシークレットリソース (**tls_secret1**) を作成します。

```
$ openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e 'base64' --payload="$(base64 < server.p12)"
```

3. パブリックサブネット (**public-subnet**) にロードバランサー (**lb1**) を作成します。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
```

4. 次のステップに進むためには、作成したロードバランサー (**lb1**) がアクティブでオンライン状態でなければなりません。
ロードバランサーの応答が **ACTIVE** および **ONLINE** のステータスになるまで、**openstack loadbalancer show** コマンドを実行します。(複数回このコマンドを実行しなければならない場合があります。)

```
$ openstack loadbalancer show lb1
```

5. TERMINATED_HTTPS リスナー (**listener1**) を作成し、リスナーのデフォルト TLS コンテナーとしてシークレットリソースを参照します。

```
$ openstack loadbalancer listener create --protocol-port 443 --protocol
TERMINATED_HTTPS --name listener1 --default-tls-container=$(openstack secret list | awk
'/ tls_secret1 / {print $2}') lb1
```

6. プール (**pool1**) を作成し、リスナーのデフォルトプールに設定します。

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
listener1 --protocol HTTP
```

7. プライベートサブネット (**private-subnet**) 上のセキュアではない HTTP バックエンドサーバー (**192.0.2.10** および **192.0.2.11**）をプールに追加します。

```
$ 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
```

15.12. SNI を使用した TLS 終端 HTTPS ロードバランサーの作成

以下の手順で、Transport Layer Security (TLS) を介してインターネットからアクセス可能な TLS 終端 HTTPS ロードバランサーを作成し、暗号化されていない HTTP プロトコルを通じてリクエストをバックエンドサーバーに分散する方法を説明します。この構成には、複数の TLS 証明書が含まれ Server Name Indication (SNI) テクノロジーを実装するリスナーが 1 つあります。

前提条件

- TCP ポート 80 でセキュアではない HTTP アプリケーションをホストするバックエンドサーバーが含まれるプライベートサブネット
- インターネットからアクセス可能な共有外部 (パブリック) サブネット
- TLS 公開鍵の暗号化が以下のように設定されている。
  - ロードバランサーの仮想 IP アドレス (例: www.example.com および www2.example.com) に割り当てられた DNS 名用に、複数の TLS 証明書、鍵、および中間証明書チェーンが外部認証局 (CA) から取得される。
  - 証明書、鍵、および中間証明書チェーンが、現在のディレクトリー内の個別ファイルに保存される。
  - 鍵および証明書は PEM 形式でエンコードされる。
  - 鍵はパスフレーズで暗号化されない。
  - 中間証明書チェーンには PEM 形式でエンコードされた複数の証明書が含まれ、チェーンを形成する。

- Key Manager サービス (barbican) を使用するように Load-balancing サービス (octavia) が設定されている。詳しい情報は、『Manage Secrets with OpenStack Key Manager』 を参照してください。

手順

1. SNI 一覧の TLS 証明書ごとに、鍵 (**server.key**)、証明書 (**server.crt**)、および中間証明書チェーン (**ca-chain.crt**) を 1 つの PKCS12 ファイル (**server.p12**) に組み合わせます。
以下の例では、それぞれの証明書 (**www.example.com** および **www2.example.com**) 用に、2 つの PKCS12 ファイル (**server.p12** および **server2.p12**) を作成します。

注記

丸かっこ内の値は例として示しています。これらの例を実際のサイトに適した値に置き換えてください。

```
$ openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.crt -passout pass:
-out server.p12

$ openssl pkcs12 -export -inkey server2.key -in server2.crt -certfile ca-chain2.crt -passout
pass: -out server2.p12
```

2. Key Manager サービスを使用して、PKCS12 ファイルのシークレットリソース (**tls_secret1** および **tls_secret2**) を作成します。

```
$ openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e 'base64' --
payload="$(base64 < server.p12)"
$ openstack secret store --name='tls_secret2' -t 'application/octet-stream' -e 'base64' --
payload="$(base64 < server2.p12)"
```

3. パブリックサブネット (**public-subnet**) にロードバランサー (**lb1**) を作成します。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
```

4. 次のステップに進むためには、作成したロードバランサー (**lb1**) がアクティブでオンライン状態でなければなりません。
ロードバランサーの応答が **ACTIVE** および **ONLINE** のステータスになるまで、**openstack loadbalancer show** コマンドを実行します。(複数回このコマンドを実行しなければならない場合があります。)

```
$ openstack loadbalancer show lb1
```

5. TERMINATED_HTTPS リスナー (**listener1**) を作成し、SNI を使用して両方のシークレットリソースを参照します。
リスナーのデフォルト TLS コンテナーとして **tls_secret1** を参照します。)

```
$ openstack loadbalancer listener create --protocol-port 443 \
--protocol TERMINATED_HTTPS --name listener1 \
--default-tls-container=$(openstack secret list | awk '/ tls_secret1 / {print $2}') \
--sni-container-refs $(openstack secret list | awk '/ tls_secret1 / {print $2}') \
$(openstack secret list | awk '/ tls_secret2 / {print $2}') -- lb1
```

6. プール (**pool1**) を作成し、リスナーのデフォルトプールに設定します。

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
listener1 --protocol HTTP
```

7. プライベートサブネット (**private-subnet**) 上のセキュアではない HTTP バックエンドサーバー (**192.0.2.10** および **192.0.2.11**) をプールに追加します。

```
$ 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
```

15.13. 同じバックエンド上での HTTP および TLS 終端 HTTPS ロードバランサーの作成

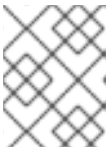
以下の手順で、セキュアではないリスナーと TLS 終端 HTTPS リスナーを同じロードバランサー (同じ IP アドレス) 上に設定する方法を説明します。クライアントがセキュアなプロトコルまたはセキュアではない HTTP プロトコルで接続されているかどうかにかかわらず、まったく同じコンテンツで Web クライアントに応答する場合に、この構成を使用します。

前提条件

- TCP ポート 80 でセキュアではない HTTP アプリケーションをホストするバックエンドサーバーが含まれるプライベートサブネット
- インターネットからアクセス可能な共有外部 (パブリック) サブネット
- TLS 公開鍵の暗号化が以下のように設定されている。
  - ロードバランサーの仮想 IP アドレス (例: www.example.com) に割り当てられた DNS 名用に、TLS 証明書、鍵、およびオプションの中間証明書チェーンが外部認証局 (CA) から取得される。
  - 証明書、鍵、および中間証明書チェーンが、現在のディレクトリー内の個別ファイルに保存される。
  - 鍵および証明書は PEM 形式でエンコードされる。
  - 鍵はパスフレーズで暗号化されない。
  - 中間証明書チェーンには PEM 形式でエンコードされた複数の証明書が含まれ、チェーンを形成する。
- Key Manager サービス (barbican) を使用するように Load-balancing サービス (octavia) が設定されている。詳しい情報は、『Manage Secrets with OpenStack Key Manager』 を参照してください。
- セキュアではない HTTP リスナーが、HTTPS TLS 終端ロードバランサーと同じプールで設定されている。

手順

1. 鍵 (**server.key**)、証明書 (**server.crt**)、および中間証明書チェーン (**ca-chain.crt**) を 1 つの PKCS12 ファイル (**server.p12**) に組み合わせます。



注記

丸かっこ内の値は例として示しています。これらの例を実際のサイトに適した値に置き換えてください。

```
$ openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.crt -passout pass: -out server.p12
```

2. Key Manager サービスを使用して、PKCS12 ファイルのシークレットリソース (**tls_secret1**) を作成します。

```
$ openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e 'base64' --payload="$(base64 < server.p12)"
```

3. パブリックサブネット (**public-subnet**) にロードバランサー (**lb1**) を作成します。

```
$ openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
```

4. 次のステップに進むためには、作成したロードバランサー (**lb1**) がアクティブでオンライン状態でなければなりません。
ロードバランサーの応答が **ACTIVE** および **ONLINE** のステータスになるまで、**openstack loadbalancer show** コマンドを実行します。(複数回このコマンドを実行しなければならない場合があります。)

```
$ openstack loadbalancer show lb1
```

5. TERMINATED_HTTPS リスナー (**listener1**) を作成し、リスナーのデフォルト TLS コンテナーとしてシークレットリソースを参照します。

```
$ openstack loadbalancer listener create --protocol-port 443 --protocol TERMINATED_HTTPS --name listener1 --default-tls-container=$(openstack secret list | awk '/ tls_secret1 / {print $2}') lb1
```

6. プール (**pool1**) を作成し、リスナーのデフォルトプールに設定します。

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
```

7. プライベートサブネット (**private-subnet**) 上のセキュアではない HTTP バックエンドサーバー (**192.0.2.10** および **192.0.2.11**) をプールに追加します。

```
$ 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
```

8. セキュアではない HTTP リスナー (**listener2**) を作成し、そのデフォルトのプールをセキュアなリスナーと同じプールに設定します。

```
$ openstack loadbalancer listener create --protocol-port 80 --protocol HTTP --name listener2 --default-pool pool1 lb1
```

15.14. AMPHORA ログへのアクセス

Amphora は負荷分散を実施するインスタンスです。Amphora のログ情報は、systemd ジャーナルで確認することができます。

1. ssh-agent を開始し、ユーザー ID キーをエージェントに追加します。

```
[stack@undercloud-0] $ eval `ssh-agent -s`
[stack@undercloud-0] $ ssh-add
```

2. SSH を使用して Amphora インスタンスに接続します。

```
[stack@undercloud-0] $ ssh -A -t heat-admin@<controller node IP address> ssh cloud-user@<IP address of Amphora in load-balancing management network>
```

3. systemd ジャーナルを表示します。

```
[cloud-user@amphora-f60af64d-570f-4461-b80a-0f1f8ab0c422 ~] $ sudo journalctl
```

ジャーナル出力の絞り込みに関する情報は、journalctl の man ページを参照してください。

4. ジャーナルの表示を終了し、Amphora インスタンスおよびコントローラーノードへの接続を終了したら、必ず SSH エージェントを停止してください。

```
[stack@undercloud-0] $ exit
```

15.15. 実行中の AMPHORA インスタンスの更新

15.15.1. 概要

定期的に、実行中の負荷分散インスタンス (amphora) をより新しいイメージで更新する必要があります。たとえば、以下のイベント時に amphora インスタンスを更新する必要があります。

- Red Hat OpenStack Platform の更新またはアップグレード
- システムへのセキュリティーアップデート
- ベースとなる仮想マシンのフレーバー変更

amphora イメージを更新するには、ロードバランサーをフェイルオーバーし、続いてロードバランサーが再びアクティブな状態になるのを待つ必要があります。ロードバランサーがアクティブな状態に戻れば、新しいイメージを実行しています。

15.15.2. 前提条件

amphora の新しいイメージは、OpenStack の更新またはアップグレード時に利用することができます。

15.15.3. 新しいイメージでの amphora インスタンスの更新

OpenStack の更新またはアップグレード時に、director は自動的にデフォルトの amphora イメージをダウンロードし、それをオーバークラウドの Image サービス (glance) にアップロードし、Octavia が新しいイメージを使用するように設定します。ロードバランサーをフェイルオーバーすると、Octavia で

は強制的に新しい amphora イメージが開始されます。

1. amphora の更新を開始する前に、必ず前提条件を確認してください。
2. 更新するすべてのロードバランサーの ID を一覧表示します。

```
$ openstack loadbalancer list -c id -f value
```

3. それぞれのロードバランサーをフェイルオーバーします。

```
$ openstack loadbalancer failover <loadbalancer_id>
```

注記

ロードバランサーのフェイルオーバーを開始したら、システムの使用状況を監視し、必要に応じてフェイルオーバーを実行する速度を調整します。ロードバランサーのフェイルオーバーにより、新規仮想マシンおよびポートが作成されます。これにより、一時的に OpenStack Networking の負荷が高まる場合があります。

4. ロードバランサーのフェイルオーバーの状態を監視します。

```
$ openstack loadbalancer show <loadbalancer_id>
```

ロードバランサーのステータスが **ACTIVE** になれば、更新は完了です。

第16章 IPV6 を使用したテナントネットワーク

16.1. プロジェクトネットワークの概要

本章では、Red Hat OpenStack Platform (RHOSP) プロジェクトネットワークに IPv6 サブネットを実装する方法について説明します。プロジェクトネットワークに加えて、RHOSP director ではオーバークラウドノードの IPv6 ネイティブデプロイメントを設定することができます。

RHOSP では、プロジェクトネットワークで IPv6 がサポートされます。IPv6 サブネットは、既存のプロジェクトネットワーク内で作成され、**ステートレスアドレス自動設定 (SLAAC)**、**ステートフル DHCPv6**、および **ステートレス DHCPv6** の複数のアドレス割り当てモードをサポートします。

16.2. IPV6 サブネットのオプション

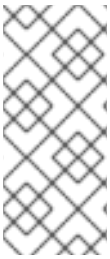
openstack subnet create コマンドを使用して、IPv6 サブネットを作成します。アドレスモードおよびルーター広告モードを指定することもできます。以下の一覧を使用して、**openstack subnet create** コマンドで指定することのできるオプションの組み合わせについて説明します。

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 アドレスを受信し、**DHCPv6 stateful** 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=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	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 OpenStack Networking (**dnsmasq**) からオプションの情報を受信します。

16.3. ステートフル DHCPV6 を使用した IPV6 サブネットの作成

セクション 17.1 に記載のオプションを使用してプロジェクトネットワークに IPv6 サブネットを作成するには、以下の手順を実施します。まず、プロジェクトとネットワークに関する必要な情報を収集し、続いて **openstack subnet create** コマンドにその情報を追加します。



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。異なるネットマスクおよび SLAAC の **address_assign_type** を使用してサブネットを作成することはできません。

1. IPv6 サブネットを作成するプロジェクトのプロジェクト 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、実際の値はこの例の値とは異なります。

```
# 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、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。

```
# openstack network list
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| id                                   | name            | subnets                                               |
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private         | c17f74c4-db41-4538-af40-
48670069af70 10.0.0.0/24             |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public          | 303ced03-6019-4e79-a21c-
1942a460b920 172.24.4.224/28         |
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers |
|
+--------------------------------------+-----------------+-------------------------------------------------------
---------+
```

3. **openstack subnet create** コマンドにプロジェクト ID およびネットワーク名を追加します。

```
# openstack subnet create --ip-version 6 --ipv6-address-mode dhcpv6-stateful --project
25837c567ed5458fbb441d39862e1399 --network database-servers --subnet-range
fdf8:f53b:82e4::53/125 subnet_name

Created a new subnet:
+-------------------+-------------------------------------------------------------------+
| Field             | Value                                                             |
+-------------------+-------------------------------------------------------------------+
| allocation_pools  | {"start": "fdf8:f53b:82e4::52", "end": "fdf8:f53b:82e4::56"} |
| cidr              | fdf8:f53b:82e4::53/125                                            |
| dns_nameservers   |                                                                   |
| enable_dhcp       | True                                                              |
| gateway_ip        | fdf8:f53b:82e4::51                                                |
| host_routes       |                                                                   |
| id                | cdfc3398-997b-46eb-9db1-ebbd88f7de05                              |
| ip_version        | 6                                                                 |
| ipv6_address_mode | dhcpv6-stateful                                                   |
| ipv6_ra_mode      |                                                                   |
| name              |                                                                   |
| network_id        | 6aff6826-4278-4a35-b74d-b0ca0cbba340                              |
| tenant_id         | 25837c567ed5458fbb441d39862e1399                                  |
+-------------------+-------------------------------------------------------------------+
```

4. ネットワークの一覧を確認して、ここでの設定を検証します。**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。

```
# openstack network list
```

```
+--------------------------------------+----------------+-------------------------------------------------
---------+
| id                                   | name           | subnets                                         |
+--------------------------------------+----------------+-------------------------------------------------
---------+
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers | cdfc3398-997b-46eb-9db1-
ebbd88f7de05 fdf8:f53b:82e4::50/125 |
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-db41-4538-af40-
48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-6019-4e79-a21c-
1942a460b920 172.24.4.224/28        |
+--------------------------------------+----------------+-------------------------------------------------
---------+
```

この設定により、QA プロジェクトの作成するインスタンスが database-servers サブネットに追加されると、DHCP IPv6 アドレスを取得できるようになります。

```
# openstack server list
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
| ID                                   | Name       | Status | Task State | Power State | Networks
|
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
| fad04b7a-75b5-4f96-aed9-b40654b56e03 | corp-vm-01 | ACTIVE | -          | Running     |
database-servers=fdf8:f53b:82e4::52 |
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
```

第17章 プロジェクトクォータの管理

17.1. プロジェクトクォータの設定

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。

- /var/lib/config-data/neutron/etc/neutron/neutron.conf ファイルで、さまざまなネットワークコンポーネントのプロジェクトクォータを設定することができます。
  たとえば、プロジェクトが作成することのできるルーターの数を制限するには、**quota_router** の値を変更します。

  ```
  quota_router = 10
  ```

  この例では、各プロジェクトのルーター数は最大 10 に制限されます。

クォータ設定の一覧は、すぐ後のセクションを参照してください。

17.2. L3 のクォータオプション

レイヤー 3 (L3) ネットワークで使用できるクォータオプションを以下に示します。

- **quota_floatingip**: プロジェクトで利用可能な Floating IP の数
- **quota_network**: プロジェクトで利用可能なネットワークの数
- **quota_port**: プロジェクトで利用可能なポートの数
- **quota_router**: プロジェクトで利用可能なルーターの数
- **quota_subnet**: プロジェクトで利用可能なサブネットの数
- **quota_vip**: プロジェクトで利用可能な仮想 IP アドレスの数

17.3. ファイアウォールのクォータオプション

プロジェクトファイアウォールの管理に使用できるクォータオプションを以下に示します。

- **quota_firewall**: プロジェクトで利用可能なファイアウォールの数
- **quota_firewall_policy**: プロジェクトで利用可能なファイアウォールポリシーの数
- **quota_firewall_rule**: プロジェクトで利用可能なファイアウォールルールの数

17.4. セキュリティーグループのクォータオプション

プロジェクトが作成することのできるセキュリティーグループ数の管理に使用できるクォータオプションを以下に示します。

- **quota_security_group**: プロジェクトで利用可能なセキュリティーグループの数
- **quota_security_group_rule**: プロジェクトで利用可能なセキュリティーグループルールの数

17.5. 管理用のクォータオプション

管理者がプロジェクトのクォータを管理する際に使用できる追加のオプションを以下に示します。

- **default_quota***: プロジェクトで利用可能なデフォルトのリソース数
- **quota_health_monitor***: プロジェクトで利用可能な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 はヘルスモニターをリソースの消費者とみなすため、クォータオプションが利用可能です。
- **quota_member**: プロジェクトで利用可能なプールメンバーの数
  プールメンバーはリソースを消費しませんが、OpenStack Networking はプールメンバーをリソースの消費者とみなすため、クォータオプションが利用可能です。
- **quota_pool**: プロジェクトで利用可能なプールの数

第18章 ALLOWED-ADDRESS-PAIRS の設定

18.1. ALLOWED-ADDRESS-PAIRS の概要

allowed-address-pairs を使用して、サブネットに関わらずポートを通過する mac_address/ip_address (CIDR) ペアを指定します。これにより、VRRP などのプロトコルを使用することができます。このプロトコルでは、2 つのインスタンス間で IP アドレスを移動して、迅速なデータプレーンのフェイルオーバーが可能です。

注記

allowed-address-pairs 拡張は、現在 ML2 および Open vSwitch のプラグインでのみサポートされています。

18.2. ポートの作成および 1 つのアドレスペアの許可

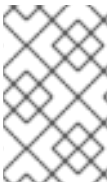
- 以下のコマンドを使用して、ポートを作成して 1 つのアドレスペアを許可します。

```
# openstack port create --network net1 --allowed-address mac_address=
<mac_address>,ip_address=<ip_cidr> PORT_NAME
```

18.3. ALLOWED-ADDRESS-PAIRS の追加

- 以下のコマンドを使用して、許可するアドレスペアを追加します。

```
# openstack port set <port-uuid> --allowed-address mac_address=
<mac_address>,ip_address=<ip_cidr>
```



注記

ポートの **mac_address** と **ip_address** が一致する allowed-address-pair を設定することはできません。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19章 レイヤー 3 高可用性 (HA) の設定

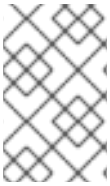
19.1. 高可用性 (HA) なしの OPENSTACK NETWORKING

高可用性 (HA) 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、プロジェク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ます。したがって、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が失われます。

19.2. レイヤー 3 高可用性 (HA) の概要

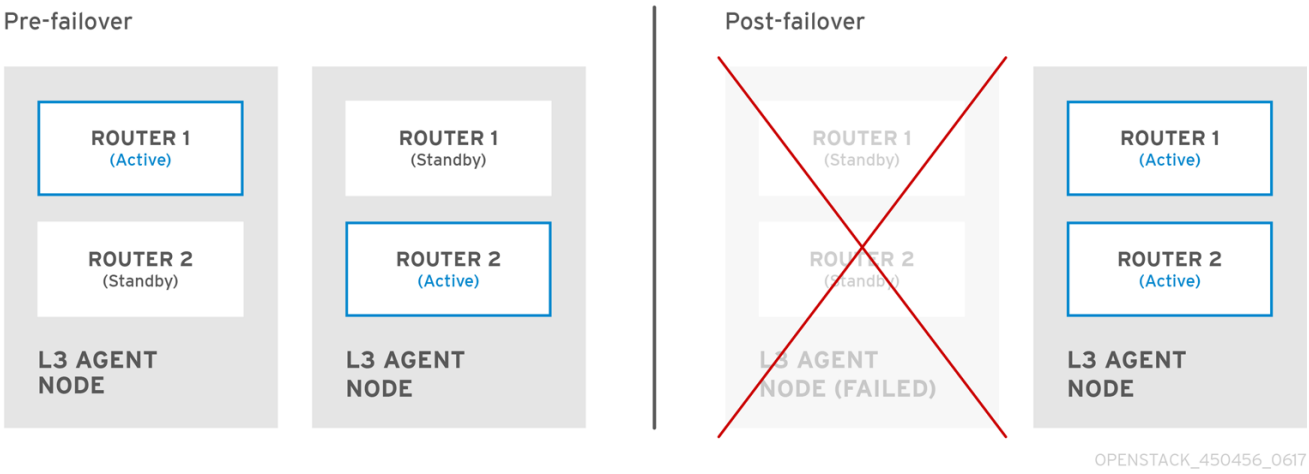
この active/passive の高可用性 (HA) 設定は、業界標準の VRRP (RFC 3768 で定義) を使用してプロジェクトルーターと Floating IP アドレスを保護します。ノードの 1 つを **active** ルーター、残りを **standby** ロールとして機能するように指定することで、仮想ルーターは複数の OpenStack Networking ノードの間で無作為にスケジュールされます。



注記

レイヤー 3 HA をデプロイするには、冗長系の 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 Router1 ルーターと Router2 ルーターが別個の物理 L3 エージェントノード上で稼働しています。レイヤー 3 HA は対応するノードに仮想ルーターのバックアップをスケジュールし、物理ノードに障害が発生した場合のサービス再開に備えます。L3 エージェントノードに障害が発生すると、レイヤー 3 HA 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。



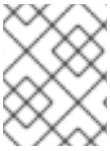
フェイルオーバーのイベント時には、Floating IP 経由のインスタンスの TCP セッションは影響を受けず、中断なしで新しい L3 ノードに移行されます。SNAT トラフィックのみがフェイルオーバーイベントの影響を受けます。

active/active HA モードの場合には、L3 エージェントはさらに保護されます。

19.3. レイヤー 3 高可用性 (HA) のフェイルオーバー条件

レイヤー 3 高可用性 (HA) は、以下のイベントにおいて保護するリソースを自動的に再スケジュールします。

- L3 エージェントノードがシャットダウンしたか、ハードウェアの障害により電力の供給を失った場合
- L3 エージェントノードが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訳ではありません。

19.4. レイヤー 3 高可用性 (HA) におけるプロジェクトの留意事項

レイヤー 3 高可用性 (HA) 設定はバックエンドで行われており、プロジェクトがそれを認識することはありません。通常通り、プロジェクトは仮想ルーターの作成/管理を続けることができます。ただし、レイヤー 3 HA の実装を設計する場合に留意すべき制限事項があります。

- レイヤー 3 HA がサポートする仮想ルーターの数は、プロジェクトごとに最大で 255 個です。
- 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、ユーザーが意識すること無く行われます。

19.5. OPENSTACK NETWORKING に加えられる高可用性 (HA) の変更

Neutron API の更新により、管理者はルーターの作成時に **--ha=True/False** フラグを設定できるようになりました。この設定は、/var/lib/config-data/neutron/etc/neutron/neutron.conf の **l3_ha** のデフォルト設定を上書きします。

- **neutron-server に加えられる HA の変更:**
  - 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、レイヤー 3 HA は無作為にアクティブなロールを割り当てます。
  - データベーススキーマが変更され、仮想ルーターへの仮想 IP アドレス (VIP) の確保を処理します。
  - レイヤー 3 HA トラフィックを転送するために、トランスポートネットワークが作成されます。
- **L3 エージェントに加えられる高可用性 (HA) 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 アドレスに変換されます。

19.6. OPENSTACK NETWORKING ノードでのレイヤー 3 高可用性 (HA) の有効化

OpenStack Networking ノードおよび L3 エージェントノードでレイヤー 3 高可用性 (HA) を有効にするには、以下の手順を実施します。

1. /var/lib/config-data/neutron/etc/neutron/neutron.conf ファイルで L3 HA を有効にし、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 HA を設定します。

```
l3_ha = True
max_l3_agents_per_router = 2
min_l3_agents_per_router = 2
```

L3 HA パラメーター:

- **l3_ha**: True に設定されると、これ以降に作成される仮想ルーターは、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、**openstack router create** コマンドで以下のオプションを使用して、各ルーターの値を上書きすることができます。

```
# openstack router create --ha
```

または

```
# openstack router create --no-ha
```

- **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、このパラメーターを 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) を保護します。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。したがって、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。

  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。したがって、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
```

19.7. 高可用性 (HA) ノード設定の確認

- 仮想ルーターの名前空間内で **ip address** コマンドを実行すると、出力では **ha-** のプレフィックスが付けられて HA デバイスが返されます。

```
# 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 HA 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20章 タグを使用した仮想デバイスの識別

20.1. 仮想デバイスのタグ付けの概要

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。インスタンスのブート時にタグがデバイスに割り当てられ、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**

20.2. 仮想デバイスのタグ付け

- 仮想デバイスをタグ付けするには、インスタンスの作成時にタグパラメーター **--block-device** および **--nic**を使用します。
  以下は例です。

```
$ 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
```

  割り当てられた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。

  この例では、以下の devices セクションにメタデータが反映されます。

  meta_data.json ファイルの内容の例:

```
  {
 "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** を使用して確認することができます。

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** にも保管されます。