



Red Hat OpenStack Platform 16.1

ネットワークガイド

Red Hat OpenStack Platform Networking の詳細ガイド

Red Hat OpenStack Platform 16.1 ネットワークガイド

Red Hat OpenStack Platform Networking の詳細ガイド

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

一般的な OpenStack Networking タスクのガイドです。

目次

はじめに	6
多様性を受け入れるオープンソースの強化	7
RED HAT ドキュメントへのフィードバック (英語のみ)	8
第1章 OPENSTACK ネットワークの概要	9
1.1. RHOSP ネットワークの管理	9
1.2. NETWORKING サービスコンポーネント	11
1.3. MODULAR LAYER 2 (ML2) ネットワーク	11
1.4. ML2 ネットワーク種別	12
1.5. MODULAR LAYER 2 (ML2) メカニズムドライバー	13
1.6. OPEN VSWITCH	14
1.7. OPEN VIRTUAL NETWORK (OVN)	14
1.8. MODULAR LAYER 2 (ML2) タイプおよびメカニズムドライバーの互換性	15
1.9. RHOSP NETWORKING サービス用のメカニズムドライバードライバー	15
第2章 ML2/OVN の操作	16
2.1. RHOSP OVN アーキテクチャーのコンポーネント一覧	16
2.2. ML2/OVN DATABASES	18
2.3. コンピュートノード上の OVN-CONTROLLER サービス	19
2.4. コンピュートノードの OVN メタデータエージェント	19
2.5. OVN コンポーザブルサービス	19
2.6. OVN でのレイヤー 3 高可用性	20
2.7. ML2/OVN メカニズムドライバーの制約	21
2.8. ML2/OVN を使用したセキュアではないポートの制限	22
2.9. 新規 RHOSP 16.1 デプロイメントでのデフォルトの ML2/OVN に代わる ML2/OVS の使用	22
2.10. DEPLOYING A CUSTOM ROLE WITH ML2/OVN	23
2.11. ML2/OVN デプロイメントにおける SR-IOV とネイティブ OVN DHCP の組み合わせ	26
第3章 プロジェクトネットワークの管理	28
3.1. VLAN のプランニング	28
3.2. ネットワークトラフィックの種別	28
3.3. IP アドレスの消費	30
3.4. 仮想ネットワーク	30
3.5. ネットワークルーティングの追加	30
3.6. ネットワークプランの例	31
3.7. ネットワークの作成	31
3.8. サブネットの使用	34
3.9. サブネットの作成	34
3.10. ルーターの追加	36
3.11. すべてのリソースのパージおよびプロジェクトの削除	37
3.12. ルーターの削除	37
3.13. サブネットの削除	37
3.14. ネットワークの削除	38
第4章 物理ネットワークへの仮想マシンインスタンスの接続	39
4.1. OPENSTACK NETWORKING トポロジーの概要	39
4.2. OPENSTACK NETWORKING サービスの配置	39
4.3. フラットプロバイダーネットワークの設定	40
4.4. 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	42
4.5. 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	46

4.6. VLAN プロバイダーネットワークの設定	49
4.7. 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	51
4.8. 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	55
4.9. ML2/OVS デプロイメントでのプロバイダーネットワーク用マルチキャストスヌーピングの有効化	57
4.10. ML2/OVN デプロイメントでのマルチキャストの有効化	59
4.11. コンピュートのメタデータアクセスの有効化	61
4.12. FLOATING IP アドレス	61
第5章 FLOATING IP アドレスの管理	62
5.1. FLOATING IP アドレスプールの作成	62
5.2. 特定の FLOATING IP アドレスの割り当て	63
5.3. 高度なネットワークの作成	64
5.4. FLOATING IP アドレスの無作為な割り当て	65
5.5. 複数の FLOATING IP アドレスプールの作成	67
5.6. 物理ネットワークのブリッジ	67
5.7. インターフェイスの追加	68
5.8. インターフェイスの削除	69
第6章 ネットワークのトラブルシューティング	70
6.1. 基本的な PING 送信テスト	70
6.2. ポートの現在のステータスの表示	72
6.3. VLAN プロバイダーネットワークへの接続に関するトラブルシューティング	73
6.4. VLAN 設定とログファイルの確認	74
6.5. ML2/OVN 名前空間内での基本的な ICMP テストの実行	75
6.6. プロジェクトネットワーク (ML2/OVS) 内からのトラブルシューティング	76
6.7. 名前空間内での高度な ICMP テストの実行 (ML2/OVS)	77
6.8. OVN トラブルシューティングコマンドのエイリアスの作成	78
6.9. OVN の論理フローのモニターリング	80
6.10. OPENFLOWS のモニターリング	83
6.11. ML2/OVN デプロイメントの検証	84
6.12. ML2/OVN のロギングモードの設定	86
6.13. エッジサイトへの登録に失敗する OVN コントローラーの修正	87
6.14. ML2/OVN ログファイル	89
第7章 OPENSTACK NETWORKING での物理スイッチの設定	90
7.1. 物理ネットワーク環境のプランニング	90
7.2. CISCO CATALYST スイッチの設定	90
7.3. CISCO NEXUS スイッチの設定	97
7.4. CUMULUS LINUX スイッチの設定	101
7.5. EXTREME EXOS スイッチの設定	103
7.6. JUNIPER EX シリーズスイッチの設定	107
第8章 最大伝送単位 (MTU) 設定の定義	113
8.1. MTU の概要	113
8.2. DIRECTOR での MTU 設定の定義	114
8.3. MTU 計算結果の確認	114
第9章 QUALITY OF SERVICE (QOS) ポリシーを使用したデータトラフィックの管理	115
9.1. QOS ルール	115
9.2. QOS ポリシーのネットワークサービスの設定	117
9.3. QOS ポリシーを使用した最小帯域幅の制御	121
9.4. QOS ポリシーを使用したネットワークトラフィックの制限	128
9.5. DSCP マーキング QOS ポリシーを使用したネットワークトラフィックの優先順位付け	132

9.6. ネットワークサービス RBAC を使用したプロジェクトへの QOS ポリシーの適用	135
第10章 ブリッジマッピングの設定	136
10.1. ブリッジマッピングの概要	136
10.2. トラフィックの流れ	136
10.3. ブリッジマッピングの設定	137
10.4. OVS ブリッジマッピングのメンテナンス	138
第11章 VLAN 対応のインスタンス	141
11.1. VLAN トランクおよび VLAN 透過ネットワーク	141
11.2. ML2/OVN デプロイメントでの VLAN 透過性の有効化	141
11.3. トランクプラグインのレビュー	143
11.4. トランク接続の作成	143
11.5. トランクへのサブポートの追加	145
11.6. トランクを使用するためのインスタンスの設定	146
11.7. NETWORKING サービスの RPC タイムアウトの設定	148
11.8. トランクの状態について	150
第12章 RBAC ポリシーの設定	151
12.1. RBAC ポリシーの概要	151
12.2. RBAC ポリシーの作成	151
12.3. RBAC ポリシーの確認	152
12.4. RBAC ポリシーの削除	152
12.5. 外部ネットワークへの RBAC ポリシーアクセスの付与	153
第13章 分散仮想ルーター (DVR) の設定	154
13.1. 分散仮想ルーター (DVR) について	154
13.2. DVR の概要	155
13.3. DVR に関する既知の問題および注意	155
13.4. サポートされているルーティングアーキテクチャー	156
13.5. ML2 OVS を使用した DVR のデプロイ	157
13.6. 集中ルーティングから分散ルーティングへの移行	158
13.7. 分散仮想ルーター (DVR) が無効になっている ML2/OVN OPENSTACK のデプロイ	159
第14章 IPV6 を使用したプロジェクトネットワーク	160
14.1. IPV6 サブネットのオプション	160
14.2. ステートフル DHCPV6 を使用した IPV6 サブネットの作成	161
第15章 プロジェクトクォータの管理	164
15.1. プロジェクトクォータの設定	164
15.2. L3 のクォータオプション	164
15.3. ファイアウォールのクォータオプション	164
15.4. セキュリティーグループのクォータオプション	164
15.5. 管理用のクォータオプション	165
第16章 ルーティング対応プロバイダーネットワークのデプロイ	166
16.1. ルーティング対応プロバイダーネットワークのメリット	166
16.2. ルーティング対応プロバイダーネットワークの概要	166
16.3. ルーティング対応プロバイダーネットワークの制限	167
16.4. ルーティング対応プロバイダーネットワークの準備	167
16.5. ルーティング対応プロバイダーネットワークの作成	170
16.6. ルーティング非対応からルーティング対応プロバイダーネットワークへの移行	176
第17章 許可するアドレスペアの設定	179
17.1. 許可するアドレスペアの概要	179

17.2. ポートの作成および1つのアドレスペアの許可	179
17.3. 許可するアドレスペアの追加	180
第18章 一般的なネットワーク管理タスク	181
18.1. L2 POPULATION ドライバーの設定	181
18.2. VRRP パケットロスを避けるための KEEPALIVED の調整	181
18.3. DNS がポートに割り当てる名前の指定	183
18.4. DHCP 属性のポートへの割り当て	186
18.5. カーネルモジュールの読み込み	188
18.6. 共有セキュリティグループの設定	189
第19章 レイヤー 3 高可用性 (HA) の設定	192
19.1. 高可用性 (HA) なしの RHOSP NETWORKING サービス	192
19.2. レイヤー 3 高可用性 (HA) の概要	192
19.3. レイヤー 3 高可用性 (HA) のフェイルオーバー条件	193
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	193
19.5. RHOSP NETWORKING サービスに対する高可用性 (HA) の変更	193
19.6. RHOSP NETWORKING サービスノードでのレイヤー 3 高可用性 (HA) の有効化	194
19.7. 高可用性 (HA) RHOSP NETWORKING サービスノード設定の確認	195
第20章 NETWORKER ノードの交換	197
20.1. ネットワークノードの交換の準備	197
20.2. NETWORKER ノードの交換	198
20.3. ノードの再スケジュールと NETWORKING サービスのクリーンアップ	201
第21章 タグを使用した仮想デバイスの識別	204
21.1. 仮想デバイスのタグ付け	204

はじめに



注記

インスタンスの作成中に、ロールベースのアクセス制御 (RBAC) 共有セキュリティーグループをインスタンスに直接適用することはできません。RBAC 共有セキュリティーグループをインスタンスに適用するには、最初にポートを作成し、共有セキュリティーグループをそのポートに適用してから、そのポートをインスタンスに割り当てる必要があります。[セキュリティーグループのポートへの追加](#) を参照してください。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#) をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. オプション: ドキュメントチームが問題の詳細を確認する際に使用できるメールアドレスを記入してください。
7. **Submit** をクリックします。

第1章 OPENSTACK ネットワークの概要

Networking サービス (neutron) は、Red Hat OpenStack Platform (RHOSP) のソフトウェア定義ネットワーク (SDN) のコンポーネントです。RHOSP Networking サービスは、仮想マシンインスタンスとの間の内部および外部トラフィックを管理し、ルーティング、セグメンテーション、DHCP、メタデータなどのコアサービスを提供します。仮想ネットワーク機能とスイッチ、ルーター、ポート、ファイアウォールの管理のための API を提供します。

1.1. RHOSP ネットワークの管理

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) を使用すると、サイトのネットワークの目標を効果的に満たすことができます。これにより、以下が可能になります。

- **プロジェクト内の VM インスタンスへの接続を提供する。**

プロジェクトネットワークは、主に一般的な (非特権) プロジェクトが、管理者を介さずにネットワークを管理できるようにするものです。これらのネットワークは完全に仮想化されており、他のプロジェクトネットワークやインターネットなどの外部ネットワークとやり取りするために仮想ルーターが必要です。また、プロジェクトネットワークは通常、インスタンスに対して DHCP やメタデータサービスを提供します。RHOSP は、フラット、VLAN、VXLAN、GRE、および GENEVE のプロジェクトネットワークタイプをサポートします。

詳しくは、[Managing project networks](#) を参照してください。

- **VM インスタンスをプロジェクト外のネットワークに接続する。**

プロバイダーネットワークは、プロジェクトネットワークのような接続性を提供します。ただし、これらのネットワークは物理ネットワークインフラストラクチャーとインターフェイスするため、管理 (特権付き) ユーザーのみがこれらのネットワークを管理できます。RHOSP は、フラットと VLAN というプロバイダーネットワークのタイプをサポートします。

プロジェクトネットワークの内部では、Floating IP アドレスのプールまたは単一の Floating IP アドレスを使用して、受信トラフィックを VM インスタンスに転送できます。ブリッジマッピングを使用すると、OVS または OVN で作成したブリッジに物理ネットワーク名 (インターフェイスラベル) を関連付け、プロバイダーネットワークトラフィックが物理ネットワークに到達できるようにすることができます。

詳しくは、[Connecting VM instances to physical networks](#) を参照してください。

- **エッジに最適化されたネットワークを構築する。**

オペレーターは、エッジデプロイメントで通常使用されるルーティング対応プロバイダーネットワーク (RPN) を作成し、セグメントが1つしかない従来のネットワークではなく、複数のレイヤー 2 ネットワークセグメントに依存することができます。

エンドユーザー向けには1つのネットワークしか表示されないの、ルーティング対応プロバイダーネットワーク (RPN) によりクラウドが単純化されます。クラウドオペレーターの場合、ルーティング対応プロバイダーネットワークはスカルルティーとフォールトトレランスを提供します。たとえば、重大なエラーが発生した場合でも、1つのセグメントしか影響を受けず、ネットワーク全体で障害が発生することはありません。

詳しくは、[Deploying routed provider networks](#) を参照してください。

- **ネットワークリソースを高可用性にする。**

アベイラビリティゾーン (AZ) や Virtual Router Redundancy Protocol (VRRP) を利用して、ネットワークリソースの可用性を高く保つことができます。オペレーターは、さまざまな AZ のさまざまな電源に接続されているネットワークノードをグループ化します。次に、オペレーターは、DHCP、L3、FW などの重要なサービスを別々の AZ で実行するようにスケジュールします。

RHOSP は VRRP を使用して、プロジェクトルーターと Floating IP アドレスの高可用性を実現します。集中型ルーティングの代わりに、分散仮想ルーティング (DVR) は、VRRP に基づく代替ルーティング設計を提供します。これは、L3 エージェントをデプロイし、すべてのコンピューティングノードにルーターをスケジュールします。

詳しくは、[Using availability zones to make network resources highly available](#) を参照してください。

- **ポートレベルでネットワークを保護する。**

セキュリティグループは、受信 (インスタンスへのインバウンド) と送信 (インスタンスからのアウトバウンド) のネットワークトラフィックをポートレベルで制御する仮想ファイアウォールルールのコンテナを提供します。セキュリティグループは、デフォルトの拒否ポリシーを使用し、特定のトラフィックを許可するルールのみを含んでいます。各ポートは、1つ以上のセキュリティグループを追加的に参照できます。ファイアウォールドライバーは、セキュリティグループのルールを、iptables などの基礎となるパケットフィルタリングテクノロジーの設定に変換します。

詳しくは、[Configuring shared security groups](#) を参照してください。

- **ポートトラフィックを管理する。**

許可されたアドレスペアを使用して、特定の MAC アドレス、IP アドレス、またはその両方を識別し、サブネットに関係なくネットワークトラフィックがポートを通過できるようにします。許可するアドレスペアを定義すると、VRRP (仮想ルーター冗長プロトコル) 等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つの仮想マシンインスタンス間で 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。

詳しくは、[Configuring allowed address pairs](#) を参照してください。

- **大規模なオーバーレイネットワークを最適化する。**

L2 Population ドライバーを使用すると、ブロードキャスト、マルチキャスト、およびユニキャストトラフィックを有効にして、大規模なオーバーレイネットワークでスケールアウトできます。

詳しくは、[Configuring the L2 population driver](#) を参照してください。

- **VM インスタンス上のトラフィックの受信および送信の制限を設定する。**

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。

詳しくは、[Configuring Quality of Service \(QoS\) policies](#) を参照してください。

- **RHOSP プロジェクトが作成できるネットワークリソースの量を管理する。**

Networking サービスの quota オプションを使用すると、プロジェクトユーザーが作成できるネットワークリソースの量に制限を設けることができます。これには、ポート、サブネット、ネットワークなどのリソースが含まれます。

詳細は、[Managing project quotas](#) を参照してください。

- **ネットワーク機能仮想化 (NFV) 用に VM インスタンスを最適化する。**

インスタンスは、単一の仮想 NIC を使用して、VLAN タグ付けされ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 タグ付けされ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ます。

VLAN 透過ネットワークでは、仮想マシンインスタンスで VLAN タグ付けを設定します。VLAN タグはネットワークを通じて転送され、同じ VLAN の仮想マシンインスタンスにより消費され、他のインスタンスやデバイスでは無視されます。VLAN トランクは、複数の VLAN を 1 つのトランクポートに結び付けて、VLAN 対応のインスタンスをサポートします。

詳しくは、[VLAN-aware instances](#) を参照してください。

- **共有ネットワークにインスタンスを接続できるプロジェクトを制御する。**

RHOSP Networking サービスの role-based access control (RBAC) ポリシーを使用すると、クラウド管理者は、一部のプロジェクトがネットワークを作成する機能を削除し、代わりにプロジェクトに対応する既存のネットワークに接続することを許可することができます。

詳細は、[Configuring RBAC policies](#) を参照してください。

1.2. NETWORKING サービスコンポーネント

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) には、以下のコンポーネントが含まれています。

- **API サーバー**

RHOSP のネットワーク API は、レイヤ 2 ネットワークと IP アドレス管理 (IPAM) のサポート、およびレイヤ 2 ネットワークと外部ネットワークへのゲートウェイ間のルーティングを可能にするレイヤ 3 ルーター設定のエクステンションを備えています。RHOSP ネットワーキングには、ルーター、スイッチ、仮想スイッチ、ソフトウェア定義ネットワーク (SDN) コントローラーなど、さまざまな商用およびオープンソースネットワークテクノロジーとの相互運用性を可能にするプラグインのリストが増えています。

- **Modular Layer 2 (ML2) プラグインとエージェント**

ML2 は、ポートをプラグおよびアンプラグし、ネットワークやサブネットを作成して、IP アドレス指定を提供します。

- **メッセージングキュー**

API 操作を完了するために、エージェント間で RPC リクエストを受け付け、ルーティングします。メッセージキューは、各ハイパーバイザー上で動作する neutron サーバーと neutron エージェント間の RPC 用 ML2 プラグイン、Open vSwitch 用 ML2 メカニズムドライバー、Linux ブリッジで使用されています。

1.3. MODULAR LAYER 2 (ML2) ネットワーク

Modular Layer 2 (ML2) は Red Hat OpenStack Platform (RHOSP) のネットワーキングのコアとなるプラグインです。ML2 モジュラー設計により、メカニズムドライバーを介した混合ネットワークテクノロジーの同時操作が可能になります。Open Virtual Network (OVN) は、ML2 で使用されるデフォルトのメカニズムドライバーです。

ML2 フレームワークは、設定可能な 2 種類のドライバーを区別します。

タイプドライバー

RHOSP ネットワークが技術的にどのように実現されるかを定義します。

使用可能な各ネットワークタイプは ML2 タイプのドライバーによって管理され、必要なタイプ固有のネットワーク状態を維持します。プロバイダーネットワークのタイプ固有の情報を検証するタイプドライバーは、プロジェクトネットワークでの空きセグメントの割り当てを行います。タイプドライバーの例としては、GENEVE、GRE、VXLAN などがあります。

メカニズムドライバー

特定のタイプの RHOSP ネットワークにアクセスするためのメカニズムを定義します。メカニズムドライバーは、タイプドライバーによって確立された情報を取得し、それを有効になっているネットワークメカニズムに適用します。メカニズムドライバーの例としては、Open Virtual Networking (OVN) や Open vSwitch (OVS) などがあります。

メカニズムドライバーは L2 エージェントを使用でき、RPC を使用して外部デバイスまたはコントローラーと直接対話します。同じ仮想ネットワークの異なるポートにアクセスするために、複数のメカニズムとタイプドライバーを同時に使用することができます。

関連情報

- [「Modular Layer 2 \(ML2\) タイプおよびメカニズムドライバーの互換性」](#)

1.4. ML2 ネットワーク種別

複数のネットワークセグメントを同時に操作できます。ML2 は、複数のネットワークセグメントの使用と相互接続をサポートします。ML2 はポートを接続のあるセグメントにバインドするため、ポートをネットワーク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。

- フラット
- VLAN
- GENEVE トンネル
- VXLAN トンネルと GRE トンネル

フラット

すべての仮想マシン (VM) インスタンスは、同じネットワーク上に存在し、ホストと共有することもできます。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

RHOSP ネットワーキングを使用すると、ユーザーは、物理ネットワークに存在する VLAN に対応する VLAN ID (802.1Q タグ付き) を使用して、複数のプロバイダーまたはプロジェクトネットワークを作成できます。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 VLAN 上にある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。

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。通信する必要がある場合、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効です。

GENEVE トンネル

GENEVE は、ネットワーク仮想化におけるさまざまなデバイスの変化する機能とニーズを認識し、それに対応します。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

まず、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。GENEVE 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。

VXLAN トンネルと GRE トンネル

VXLAN と 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通信をサポートします。RHOSP ネットワーキングルーターは、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) に接続するためにも必要とされ、このルーターは、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。

関連情報

- [「Modular Layer 2 \(ML2\) タイプおよびメカニズムドライバーの互換性」](#)

1.5. MODULAR LAYER 2 (ML2) メカニズムドライバー

Modular Layer 2 (ML2) プラグインは、共通のコードベースを持つメカニズムとして実装されます。このアプローチにより、コードの再利用が可能になる上、コードのメンテナンスとテストにおける複雑性が大幅に軽減されます。

Orchestration サービス (heat) パラメーター **NeutronMechanismDrivers** を使用して、メカニズムドライバーを有効にします。heat カスタム環境ファイルの例を以下に示します。

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

メカニズムドライバーを指定する順番が重要です。上記の例で、ベアメタルメカニズムドライバーを使用してポートをバインドする場合は、**ansible** の前に **baremetal** を指定する必要があります。この順番で指定しないと、ansible ドライバーがポートをバインドします。**NeutronMechanismDrivers** の値の一覧では、ansible が **baremetal** に優先するためです。

RHOSP 15 以降のすべての新規デプロイメントについて、Red Hat では ML2/OVN をデフォルトのメカニズムドライバーとして選択しました。これは、今日のほとんどのお客様にとって ML2/OVS メカニズムドライバー以上のメリットが即座に得られるためです。継続して ML2/OVN 機能セットの拡張および改善を行っているため、これらのメリットはリリースと共に拡大します。

非推奨の ML2/OVS メカニズムドライバーは、RHOSP 17 リリースでサポートされます。この間、ML2/OVS ドライバーはメンテナンスモードのままで、バグ修正と通常のサポートを受け、ほとんどの新機能開発は ML2/OVN メカニズムドライバーで行われます。

RHOSP 18.0 では、Red Hat は ML2/OVS メカニズムドライバーを完全に削除し、サポートを停止する予定です。

既存の Red Hat OpenStack Platform (RHOSP) デプロイメントで ML2/OVS メカニズムドライバーを使用している場合は、今すぐメカニズムドライバーへの移行計画の評価を開始してください。移行は RHOSP 16.2 でサポートされており、RHOSP 17.1 でもサポートされる予定です。移行ツールの使用は、RHOSP 17.0 でのテスト目的に限定されています。

ML2/OVS から ML2/OVN への移行を試みる前に、プロアクティブケースを作成する必要があります。プロアクティブケースを作成しない場合、Red Hat では移行をサポートしません。[How to open a proactive case for a planned activity on Red Hat OpenStack Platform?](#) を参照してください。

関連情報

- **Component, Plug-In, and Driver Support in Red Hat OpenStack Platform** の [Neutron](#)
- **Advanced Overcloud Customization** の [Environment files](#)
- **Advanced Overcloud Customization** ガイドの [Including environment files in overcloud creation](#)

1.6. 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イスの結合の詳細については、**Advanced Overcloud Customization** ガイドの [Network Interface Bonding](#) 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イ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イ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。

1.7. OPEN VIRTUAL NETWORK (OVN)

Open Virtual Network (OVN) は、仮想マシンやコンテナ環境において、論理的なネットワーク抽象化をサポートするためのシステムです。Open vSwitch のオープンソース仮想ネットワークングと呼ばれることもある OVN は、OVS の既存の機能を補完し、論理 L2 および L3 オーバーレイ、セキュリティグループ、DHCP などのサービスなど、論理ネットワーク抽象化のネイティブサポートを追加します。

物理ネットワークは、物理ワイヤー、スイッチ、およびルーターで設定されます。仮想ネットワークは、物理ネットワークをハイパーバイザーやコンテナプラットフォームに拡張し、VM やコンテナを物理ネットワークにブリッジします。OVN 論理ネットワークとは、トンネルなどのカプセル化によって物理ネットワークから分離されたソフトウェアに実装されたネットワークです。これにより、論理ネットワークで使用される IP などのアドレス空間と物理ネットワークで使用されるアドレス空間が競合を起こすことなくオーバーラップするようになります。論理ネットワークトポロジは、それらが実行されている物理ネットワークのトポロジに関係なく配置できます。このため、論理ネットワークの一部である VM は、ネットワークを遮断することなく、ある物理マシンから別の物理マシンへ移行することができます。

カプセル化レイヤーは、論理ネットワークに接続された VM やコンテナが、物理ネットワーク上のノードと通信することを阻止します。VM とコンテナをクラスターリングする場合、これは許容範囲内であり、望ましい場合もあります。しかし、多くの場合、VM およびコンテナは物理ネットワークへの接続整を必要とします。OVN では、この目的のために複数の形式のゲートウェイを提供します。OVN のデプロイメントは、いくつかのコンポーネントで設定されています。

Cloud Management System (CMS)

OVN 論理ネットワーク要素を管理し、OVN 論理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要素に接続することにより、OVN を物理ネットワークに統合します。いくつかの例には、OpenStack および OpenShift が含まれます。

OVN データベース

OVN の論理ネットワークと物理ネットワークを表すデータを格納します。

Hypervisors

Open vSwitch を実行し、OVN 論理ネットワークを物理マシンまたは仮想マシン上の OpenFlow に変換します。

ゲートウェイ

トンネルと物理ネットワークインフラストラクチャー間でパケットを転送することにより、トンネルベースの OVN 論理ネットワークを物理ネットワークに拡張します。

1.8. MODULAR LAYER 2 (ML2) タイプおよびメカニズムドライバーの互換性

Red Hat OpenStack Platform のデータネットワークを計画する際には、以下の表を参照して、各 Modular Layer 2 (ML2) メカニズムドライバーがサポートするネットワークタイプを決定してください。

表1.1 ML2 メカニズムドライバーがサポートするネットワークタイプ

メカニズムドライバー	これらのタイプのドライバーをサポートします。				
	フラット	GRE	VLAN	VXLAN	GENEVE
Open Virtual Network (OVN)	○	いいえ	はい	いいえ	○
Open vSwitch (OVS)	○	○	○	○	いいえ

1.9. RHOSP NETWORKING サービス用のメカニズムドライバードライバー

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) は、拡張可能です。エクステンションには 2 つの目的があります。バージョンを変更せずに API に新機能を導入できるようにすることと、ベンダー固有のニッチ機能を導入できるようにすることです。アプリケーションは、`/extensions` URI で GET を実行することにより、使用可能なエクステンションをプログラムで一覧表示することができます。これはバージョン管理されたリクエストであることに注意してください。つまり、ある API バージョンで使用可能なエクステンションは、別のバージョンでは使用できない場合があります。

ML2 プラグインは、他のプラグイン可能なドライバーがネットワークオブジェクトの ML2 プラグインに実装されているコアリソースを拡張できるようにするエクステンションドライバーもサポートします。エクステンションドライバーの例としては、QoS やポートセキュリティなどのサポートが挙げられます。

第2章 ML2/OVN の操作

Red Hat OpenStack Platform (RHOSP) のネットワークは、Networking サービス (neutron) によって管理されます。Networking サービスの中核は Modular Layer 2 (ML2) プラグインで、RHOSP ML2 プラグインのデフォルトメカニズムドライバーは Open Virtual Networking (OVN) メカニズムドライバーです。

以前の RHOSP バージョンでは、デフォルトで Open vSwitch (OVS) メカニズムドライバーが使用されていました。RHOSP 16.0 以降のすべての新規デプロイメントについて、Red Hat では ML2/OVN をデフォルトのメカニズムドライバーとして選択しました。これは、今日のほとんどのお客様にとって ML2/OVS メカニズムドライバー以上のメリットが即座に得られるためです。Red Hat およびコミュニティが継続して ML2/OVN 機能セットの拡張および改善を行っているため、これらのメリットはリリースと共に拡大します。

既存の Red Hat OpenStack Platform (RHOSP) デプロイメントで ML2/OVS メカニズムドライバーが使用されている場合、OVS ドライバーを ML2/OVN メカニズムドライバーに置き換えるメリットおよび現実性の評価をする必要があります。Red Hat は、RHOSP 16.1 の ML2/OVN への直接移行をサポートしていません。ML2/OVN メカニズムドライバーに移行する前に、最新の RHOSP 16.2 バージョンにアップグレードする必要があります。

2.1. RHOSP OVN アーキテクチャーのコンポーネント一覧

RHOSP OVN アーキテクチャーでは、Networking API をサポートするために OVS Modular Layer 2 (ML2) メカニズムドライバーが OVN ML2 メカニズムドライバーに置き換えられます。OVN は、Red Hat OpenStack Platform のネットワークサービスを提供します。

図 2.1 に示すように、OVN アーキテクチャーは次のコンポーネントとサービスで設定されます。

OVN メカニズムドライバーを備えた ML2 プラグイン

ML2 プラグインは、OpenStack 固有のネットワーク設定を、プラットフォーム非依存の OVN 論理ネットワーク設定に変換します。通常、コントローラーノード上で実行されます。

OVN northbound (NB) データベース (ovn-nb)

このデータベースは、OVN ML2 プラグインからの論理 OVN ネットワーク設定を保管します。通常コントローラーノードで実行され、TCP ポート **6641** をリスンします。

OVN northbound サービス (ovn-northd)

このサービスは OVN NB データベースからの論理ネットワーク設定を論理データパスフローに変換して、それらを OVN Southbound データベースに投入します。通常、コントローラーノード上で実行されます。

OVN southbound (SB) データベース (ovn-sb)

このデータベースは、変換された論理データパスフローを保管します。通常コントローラーノードで実行され、TCP ポート **6642** をリスンします。

OVN コントローラー (ovn-controller)

このコントローラーは OVN SB データベースに接続して Open vSwitch コントローラーとして機能し、ネットワークトラフィックの制御とモニターリングを行います。これにより、**OS::TripleO::Services::OVNController** が定義されているすべてのコンピュートおよびゲートウェイノードで実行されます。

OVN メタデータエージェント (ovn-metadata-agent)

このエージェントは、OVS インターフェイス、ネットワーク名前空間、メタデータ API 要求のプロキシに使用される HAProxy プロセスを管理するための **haproxy** インスタンスを作成します。このエージェントは、**OS::TripleO::Services::OVNMetadataAgent** が定義されているすべてのコンピュートおよびゲートウェイノードで実行されます。

OVS データベースサーバー (OVSDb)

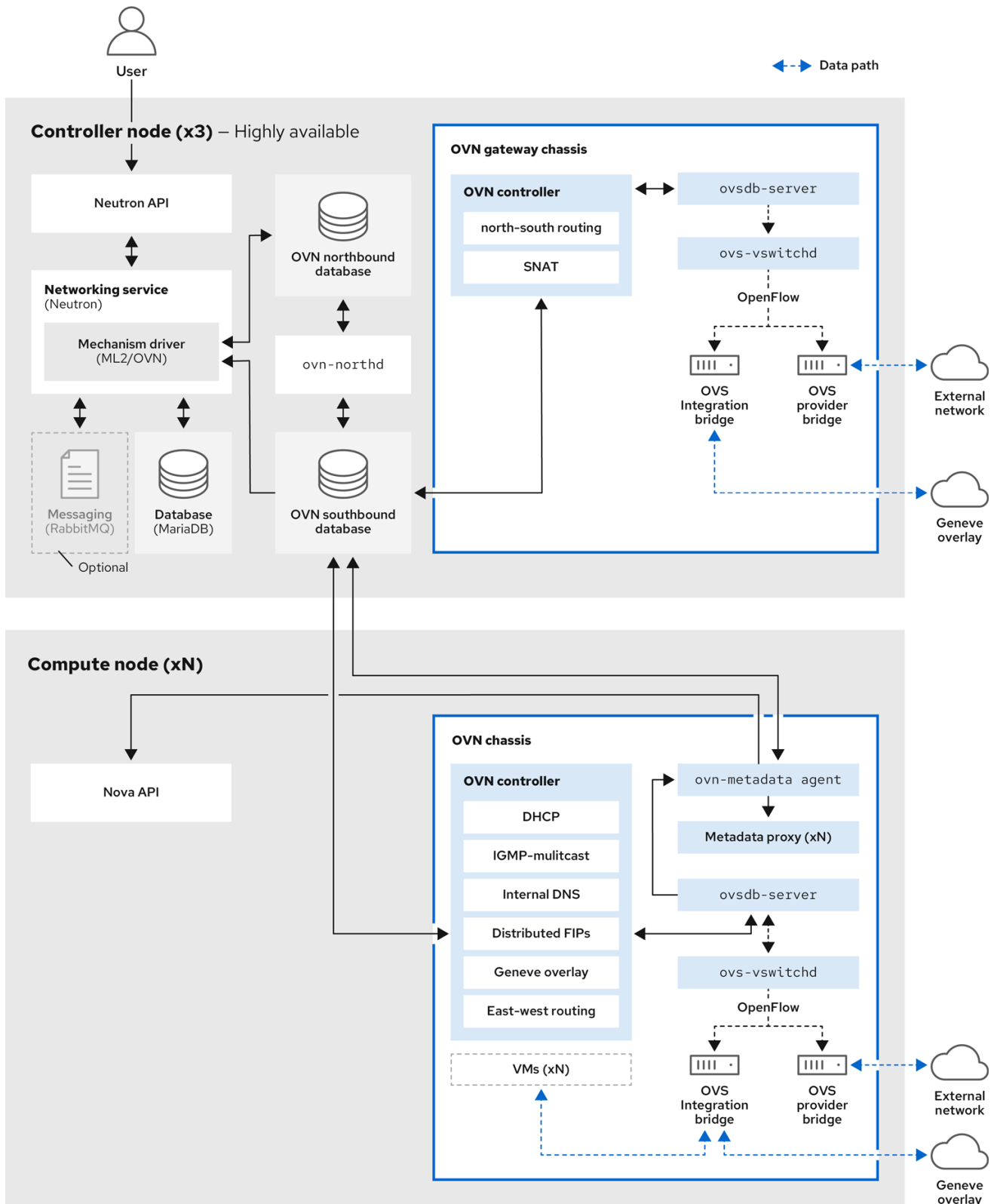
OVN の Northbound および Southbound データベースをホストします。また、**ovs-vswitchd** と連携して OVS データベース **conf.db** をホストします。



注記

NB データベースのスキーマファイルは **/usr/share/ovn/ovn-nb.ovsschema** にあり、SB データベースのスキーマファイルは **/usr/share/ovn/ovn-sb.ovsschema** にあります。

図2.1 RHOSP 環境の OVN アーキテクチャー



329_OpenStack_0623

2.2. ML2/OVN DATABASES

Red Hat OpenStack Platform ML2/OVN デプロイメントでは、ネットワーク設定情報は共有分散データベースを介してプロセス間で受け渡されます。これらのデータベースを調べて、ネットワークのステータスを確認し、問題を特定できます。

OVN ノースバウンドデータベース

ノースバウンドデータベース (**OVN_Northbound**) は、OVN と Red Hat OpenStack Platform (RHOSP) などのクラウド管理システムの間インターフェイスとして機能します。RHOSP は、ノースバウンドデータベースのコンテンツを生成します。

ノースバウンドデータベースには、ネットワークの現在の望ましい状態が含まれており、論理ポート、論理スイッチ、論理ルーターなどのコレクションとして提示されます。すべての RHOSP Networking サービス (neutron) オブジェクトは、ノースバウンドデータベースのテーブルに表示されます。

OVN サウスバウンドデータベース

サウスバウンドデータベース (**OVN_Southbound**) は、OVN システムが仮想ネットワークの抽象化をサポートするための論理的および物理的な設定状態を保持します。**ovn-controller** は、このデータベース内の情報を使用して、ネットワークサービス (neutron) の要件を満たすように OVS を設定します。

2.3. コンピュートノード上の OVN-CONTROLLER サービス

ovn-controller サービスは各コンピュータノードで実行され、OVN Southbound (SB) データベースサーバーに接続して論理フローを取得します。次に **ovn-controller** はその論理フローを OpenFlow の物理フローに変換して、OVS ブリッジ (**br-int**) に追加します。**ovs-vswitchd** と通信して OpenFlow フローをインストールするために、**ovn-controller** は **ovn-controller** の起動時に渡された UNIX ソケットパス (例: **unix:/var/run/openvswitch/db.sock**) を使用して、(**conf.db** をホストする) ローカルの **ovsdb-server** に接続します。

ovn-controller サービスは、**Open_vSwitch** テーブルの **external_ids** コラムに特定のキーと値のペアがあることを想定します。**puppet-ovn** は **puppet-vswitch** を使用して、これらのフィールドにデータを読み込みます。**puppet-vswitch** が **external_ids** コラムに設定するキーと値のペアは以下のとおりです。

```
hostname=<HOST NAME>
ovn-encap-ip=<IP OF THE NODE>
ovn-encap-type=geneve
ovn-remote=tcp:OVN_DBS_VIP:6642
```

2.4. コンピュートノードの OVN メタデータエージェント

OVN メタデータエージェントは **tripleo-heat-templates/deployment/ovn/ovn-metadata-container-puppet.yaml** ファイルで設定され、**OS::TripleO::Services::OVNMetadataAgent** でデフォルトのコンピュータロールに含まれます。そのため、デフォルトのパラメーターを使用する OVN メタデータエージェントは、OVN のデプロイメントの一環としてデプロイされます。

OpenStack のゲストインスタンスは、169.254.169.254 のリンクローカル IP アドレスで利用可能なネットワークのメタデータサービスにアクセスします。**neutron-ovn-metadata-agent** は、コンピュータのメタデータ API があるホストネットワークへのアクセスが可能です。各 HAProxy は、適切なホストネットワークに到達できないネットワーク名前空間内にあります。HaProxy は、メタデータ API の要求に必要なヘッダーを追加してから、UNIX ドメインソケット上でその要求を **neutron-ovn-metadata-agent** に転送します。

OVN のネットワークサービスは、メタデータサービスを有効化する各仮想ネットワークに独自のネットワーク名前空間を作成します。コンピュータノード上のインスタンスによってアクセスされる各ネットワークには、対応するメタデータ名前空間 (**ovnmeta-<datapath_uuid>**) があります。

2.5. OVN コンポーザブルサービス

通常 Red Hat OpenStack Platform は、事前定義済みロールのノード (Controller ロール、Compute ロール、さまざまなストレージロール種別のノードなど) で設定されます。これらのデフォルトロールには、それぞれコアの heat テンプレートコレクションで定義されるサービスのセットが含まれます。

デフォルトの OSP 16.1 デプロイメントでは、ML2/OVN コンポーザブルサービスはコントローラーノード上で実行されます。オプションとして、カスタムの Networker ロールを作成し、専用のネットワークカーノードで OVN コンポーザブルサービスを実行することができます。

OVN コンポーザブルサービス **ovn-dbs** は、**ovn-dbs-bundle** というコンテナにデプロイされます。デフォルトのインストールでは、**ovn-dbs** は Controller ロールに含まれ、コントローラーノードで実行されます。サービスはコンポーザブルなので、Networker ロール等の別のロールに割り当てることができます。

OVN コンポーザブルサービスを別のロールに割り当てる場合には、サービスが Pacemaker サービスと同じノード上に共存し、OVN データベースコンテナを制御するようにします。

関連情報

- [ML2/OVN でのカスタムロールのデプロイ](#)
- [ML2/OVN デプロイメントにおける SR-IOV とネイティブ OVN DHCP の組み合わせ](#)

2.6. OVN でのレイヤー 3 高可用性

OVN は、特別な設定なしでレイヤー 3 の高可用性 (L3 HA) をサポートします。OVN は、指定した外部ネットワークで L3 ゲートウェイとして機能することが可能なすべての利用可能なゲートウェイノードに対して、ルーターポートを自動的にスケジューリングします。OVN L3 HA は OVN

Logical_Router_Port テーブルの **gateway_chassis** コラムを使用します。大半の機能は、バンドルされた **active_passive** の出力を使用する OpenFlow ルールによって管理されます。**ovn-controller** は Address Resolution Protocol (ARP) リスポンダーとルーターの有効化/無効化を処理します。FIP 用の Gratuitous ARP およびルーターの外部アドレスも **ovn-controller** によって定期的に送信されます。



注記

L3HA は OVN を使用してルーターのバランスを取り、元のゲートウェイノードに戻して、ノードがボトルネックとなるのを防ぎます。

BFD モニターリング

OVN は双方向フォワーディング検出 (BFD) プロトコルを使用してゲートウェイノードの可用性をモニターリングします。このプロトコルは、ノード間で確立される Geneve トンネル上でカプセル化されます。

各ゲートウェイノードは、デプロイメント内のスタートポロジを設定するその他すべてのゲートウェイノードをモニターリングします。ゲートウェイノードは、コンピュータノードもモニターリングして、パケットのルーティングの有効化/無効化および ARP の応答とアナウンスメントを行います。

各コンピュータノードは BFD を使用して、各ゲートウェイノードをモニターリングし、特定のルーターのアクティブなゲートウェイノードを介して送信元および宛先のネットワークアドレス変換 (SNAT および DNAT) などの外部のトラフィックを自動的に誘導します。コンピュータノードは他のコンピュータノードをモニターリングする必要はありません。



注記

ML2-OVS 設定で検出されるような外部ネットワークのエラーは検出されません。

OVN 向けの L3 HA では、以下の障害モードがサポートされています。

- ゲートウェイノードがネットワーク (トンネリングインターフェイス) から切断された場合。
- **ovs-vswitchd** が停止した場合 (**ovs-switchd** が BFD のシグナリングを行うロールを果たします)。
- **ovn-controller** が停止した場合 (**ovn-controller** は登録済みノードとして、それ自身を削除します)。



注記

この BFD モニタリングメカニズムは、リンクのエラーのみで機能し、ルーティングのエラーには機能しません。

2.7. ML2/OVN メカニズムドライバーの制約

ML2/OVS メカニズムドライバーで利用可能な機能の一部は、ML2/OVN メカニズムドライバーではまだサポートされていません。

2.7.1. ML2/OVN でサポートされていない ML2/OVS 機能

機能	備考	本機能の経緯
VLAN プロジェクト (テナント) ネットワーク上での分散仮想ルーター (DVR) と OVN の組み合わせ	<p>FIP トラフィックは、ML2/OVN および DVR を使用する VLAN テナントネットワークに渡されません。</p> <p>DVR は、新しい ML2/OVN デプロイメントではデフォルトで有効化されています。VLAN テナントネットワークで OVN を使用する必要がある場合は、DVR を無効にすることができます。DVR を無効にするには、環境ファイルに以下の行を追加します。</p> <pre>parameter_defaults: NeutronEnableDVR: false</pre>	Bug 1704596 Bug 1766930
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。	Bug 1622154

2.7.2. OVN に関する主な制約

外部ポートは論理ルーターのゲートウェイポートと同じ場所に配置されていないため、VLAN テナントネットワークでは、VF (直接) ポートでの North-South ルーティングは SR-IOV では機能しません。[Bug #1875852](#) を参照してください。

2.8. ML2/OVN を使用したセキュアではないポートの制限

デフォルトの ML2/OVN メカニズムドライバーと多数のポートを使用した Red Hat Open Stack Platform (RHOSP) デプロイメントでは、ポートセキュリティープラグイン拡張を無効にすると、ポートに到達できなくなる可能性があります。

一部の大規模な ML2/OVN RHOSP デプロイメントでは、ML2/OVN 内のフローチェーン制限により、セキュリティープラグインが無効になっているポートを対象とする ARP 要求がドロップされます。

ML2/OVN がサポートできる実際の論理スイッチポートの最大数は文書化されていませんが、約 4000 ポートに制限されます。

概略の制限に影響する属性は、ML2/OVN が生成する OpenFlow パイプラインでの再送信数と、論理トポロジー全体への変更です。

2.9. 新規 RHOSP 16.1 デプロイメントでのデフォルトの ML2/OVN に代わる ML2/OVS の使用

Red Hat OpenStack Platform (RHOSP) 16.0 以降のデプロイメントでは、Modular Layer 2 プラグインと Open Virtual Network の組み合わせ (ML2/OVN) が RHOSP Networking サービスのデフォルトメカニズムドライバーです。アプリケーションが ML2/OVS メカニズムドライバーを必要とする場合は、この設定を変更することができます。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. テンプレートファイル `/home/stack/templates/containers-prepare-parameter.yaml` で、**neutron_driver** パラメーターの値として **ovn** の代わりに **ovs** を使用します。

```
parameter_defaults:
  ContainerImagePrepare:
    - set:
      ...
      neutron_driver: ovs
```

3. 環境ファイル `/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml` で、**NeutronNetworkType** パラメーターに **geneve** ではなく **vxlan** または **gre** が含まれるようにします。

例

```
parameter_defaults:
  ...
  NeutronNetworkType: 'vxlan'
```

4. コア heat テンプレート、環境ファイル、および変更したファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovs.yaml \
-e /home/stack/templates/containers-prepare-parameter.yaml \
```

関連情報

- [Advanced Overcloud Customization](#) の [Environment files](#)
- [Advanced Overcloud Customization](#) ガイドの [Including environment files in overcloud creation](#)

2.10. DEPLOYING A CUSTOM ROLE WITH ML2/OVN

デフォルトの OSP 16.1 デプロイメントでは、ML2/OVN コンポーザブルサービスはコントローラーノード上で実行されます。以下の例のように、サポートされているカスタムロールをオプションで使用できます。

Networker

専用のネットワークカーノードで OVN コンポーザブルサービスを実行します。

Networker と SR-IOV の組み合わせ

SR-IOV と共に専用のネットワークノードで OVN コンポーザブルサービスを実行します。

Controller と SR-IOV の組み合わせ

SR-IOV 対応のコントローラーノードで OVN コンポーザブルサービスを実行します。

独自のカスタムロールを生成することもできます。

制限

本リリースでは、ML2/OVN デプロイメントで SR-IOV とネイティブ OVN DHCP の組み合わせを使用する場合、以下の制限が適用されます。

- すべてのポートに対して HA シャーシグループが1つしかないため、すべての外部ポートは単一のゲートウェイノード上でスケジュールされる。
- 外部ポートは論理ルーターのゲートウェイポートと共存しないため、VLAN テナントネットワークでは、VF (直接) ポートでの North-South ルーティングは SR-IOV では機能しない。[Bug #1875852](#) を参照してください。

前提条件

- カスタムロールのデプロイ方法を理解している。詳細は、[オーバークラウドの高度なカスタマイズ](#) ガイドの [コンポーザブルサービスとカスタムロール](#) を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインし、source コマンドで **stackrc** ファイルを読み込みます。

```
$ source stackrc
```

2. デプロイメントに適したカスタムロールファイルを選択します。そのままでご自分のニーズに適する場合には、直接デプロイコマンドで使します。あるいは、他のカスタムロールファイルを組み合わせる独自のカスタムロールファイルを生成することもできます。

デプロイメント	ロール	ロールファイル
Networker ロール	Networker	Networker.yaml
Networker ロールと SR-IOV の組み合わせ	NetworkerSriov	NetworkerSriov.yaml
共存する control および networker と SR-IOV の組み合わせ	ControllerSriov	ControllerSriov.yaml

3. (オプション) これらのカスタムロールファイルの1つと他のカスタムロールファイルを組み合わせる新しいカスタムロールデータファイルを生成します。**オーバークラウドの高度なカスタマイズ**の [roles_data ファイルの作成](#) に記載の手順に従います。デプロイメントに応じて、適切なソース出力ファイルを含めます。
4. (オプション) ロール用の特定のノードを特定するには、特定のハードウェアフレーバーを作成して特定のノードにフレーバーを割り当てることができます。次に、環境ファイルを使用してロールのフレーバーを定義し、ノード数を指定します。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 [新規ロールの作成](#) の例を参照してください。
5. デプロイメントに適した環境ファイルを作成します。

デプロイメント	環境ファイルのサンプル
Networker ロール	neutron-ovn-dvr-ha.yaml
Networker ロールと SR-IOV の組み合わせ	ovn-sriov.yaml

6. デプロイメントに適するように、以下の設定を含めます。

デプロイメント	設定
Networker ロール	<pre> ControllerParameters: OVNCMSOptions: "" ControllerSriovParameters: OVNCMSOptions: "" NetworkerParameters: OVNCMSOptions: "enable-chassis-as-gw" NetworkerSriovParameters: OVNCMSOptions: "" </pre>

デプロイメント	設定
Networker ロールと SR-IOV の組み合わせ	<pre>OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None ControllerParameters: OVNCMSOptions: "" ControllerSriovParameters: OVNCMSOptions: "" NetworkerParameters: OVNCMSOptions: "" NetworkerSriovParameters: OVNCMSOptions: "enable-chassis-as-gw"</pre>
共存する control および networker と SR-IOV の組み合わせ	<pre>OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None ControllerParameters: OVNCMSOptions: "" ControllerSriovParameters: OVNCMSOptions: "enable-chassis-as-gw" NetworkerParameters: OVNCMSOptions: "" NetworkerSriovParameters: OVNCMSOptions: ""</pre>

7. オーバークラウドをデプロイします。 **-e** オプションを使用して、環境ファイルをデプロイメントコマンドに追加します。 **-r** オプションを使用して、カスタムロールデータファイルをデプロイメントコマンドに追加します。(例: **-r Networker.yaml** または **-r mycustomrolesfile.yaml**)。

検証手順 - OVN のデプロイ

- 1. オーバークラウドの SSH ユーザー (デフォルトでは **heat-admin**) としてコントローラーまたは Networker ノードにログインします。

例

```
ssh heat-admin@controller-0
```

- 2. **ovn_metadata_agent** がコントローラーノードおよびネットワークカーノードで実行されていることを確認します。

```
$ sudo podman ps | grep ovn_metadata
```

出力例

```
a65125d9588d undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp16-openstack-
neutron-metadata-agent-ovn:16.1_20200813.1 kolla_start      23 hours ago Up 21 hours
ago      ovn_metadata_agent
```

3. OVN サービスが設定されたコントローラーノードまたは専用のネットワークカーノードが OVS のゲートウェイとして設定されていることを確認します。

```
$ sudo ovs-vsctl get Open_Vswitch . external_ids:ovn-cms-options
```

出力例

```
...
enable-chassis-as-gw
...
```

検証手順 - SR-IOV のデプロイメント

1. デフォルトでは **heat-admin** であるオーバークラウド SSH ユーザーとしてコンピュータノードにログインします。

例

```
ssh heat-admin@compute-0
```

2. **neutron_sriov_agent** がコンピュータノードで実行されていることを確認します。

```
$ sudo podman ps | grep neutron_sriov_agent
```

出力例

```
f54cbbf4523a undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp16-openstack-neutron-
sriov-agent:16.2_20200813.1
kolla_start 23 hours ago Up 21 hours ago      neutron_sriov_agent
```

3. ネットワークに接続された SR-IOV NIC が正常に検出されていることを確認します。

```
$ sudo podman exec -uroot galera-bundle-podman-0 mysql nova -e 'select
hypervisor_hostname,pci_stats from compute_nodes;'
```

出力例

```
computesriov-1.localdomain {... {"dev_type": "type-PF", "physical_network": "datacentre",
"trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF", "physical_network": "datacentre",
"trusted": "true", "parent_ifname": "enp7s0f3"}, "count": 5}, ...}
computesriov-0.localdomain {... {"dev_type": "type-PF", "physical_network": "datacentre",
"trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF", "physical_network": "datacentre",
"trusted": "true", "parent_ifname": "enp7s0f3"}, "count": 5}, ...}
```

関連情報

- Advanced Overcloud Customization ガイドの [Composable services and custom roles](#)

2.11. ML2/OVN デプロイメントにおける SR-IOV とネイティブ OVN DHCP の組み合わせ

カスタムロールをデプロイして、ML2/OVN デプロイメントにおいて SR-IOV とネイティブ OVN DHCP の組み合わせを使用することができます。「[Deploying a custom role with ML2/OVN](#)」を参照してください。

制限

本リリースでは、ML2/OVN デプロイメントで SR-IOV とネイティブ OVN DHCP の組み合わせを使用する場合、以下の制限が適用されます。

- すべてのポートに対して HA シャーシグループが1つしかないため、すべての外部ポートは単一のゲートウェイノード上でスケジュールされる。
- 外部ポートは論理ルーターのゲートウェイポートと共存しないため、VLAN テナントネットワークでは、VF (直接) ポートでの North-South ルーティングは SR-IOV では機能しない。[Bug #1875852](#) を参照してください。

関連情報

- **Advanced Overcloud Customization** ガイドの [Composable services and custom roles](#)

第3章 プロジェクトネットワークの管理

プロジェクトネットワークは、クラウドコンピューティングのネットワークトラフィックを分離するのに役立ちます。プロジェクトネットワークの作成手順には、ネットワークのプランニングと作成、サブネットおよびルーターの追加が含まれます。

3.1. 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。

3.2. 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

すべての OpenStack デプロイメントで、このセクションのすべての分離 VLAN が必要なものではありません。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、コンピュータ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の通信に内部 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イスとして機能します。プロキシは、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。

3.3. 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。

3.4. 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イスには、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イスに 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。

3.5. 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イ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、**+インターフェイスの追加** 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。インターフェイスの 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。

3.6. 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3.1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)	10.10.3.10 - 10.10.3.250	241	255.255.252.0

3.7. 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。外部ネットワークの接続に関する詳細は、**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こと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ロール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、このサブネットが HTTP Web サーバーをホストし、VLAN タグが 122 の場合には webserver_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。
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。

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。

フィールド	説明
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イスの 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あればなりません。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、192.168.22.100,192.168.22.150 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。



重要

DNS 等の重要なサービスの場合、クラウド上ではホストしないことがベストプラクティスです。たとえば、クラウドで DNS をホストしている場合にクラウドが正常に動作しなくなると、DNS は利用できず、クラウドコンポーネントは互いに検索することができなくなります。

- **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。

3.8. 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が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。

3.9. 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

フィールド	説明
ネットワークアドレス	IP アドレス範囲とサブネットマスクが1つの値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イスの 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あればなりません。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、192.168.22.100,192.168.22.150 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、その後ルーター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。

3.10. 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イ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イスが必要です。

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イスを作成します。

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。

3.11. すべてのリソースのページおよびプロジェクトの削除

openstack project purge コマンドを使用して、特定のプロジェクトに属するすべてのリソースを削除することや、プロジェクトを削除することもできます。

たとえば、**test-project** プロジェクトのリソースをページし、続いてプロジェクトを削除するには、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       |
+-----+-----+

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

3.12. ルーターの削除

インターフェイ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。

インターフェイ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** のインターフェイスを選択し、**インターフェイスの削除** 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。

3.13. 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。

3.14. 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イ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イ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イスをすべて削除します。

インターフェイスを削除するには、**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イスを特定します。
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。インターフェイスのネットワーク 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イスの **インターフェイスの削除** 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。

第4章 物理ネットワークへの仮想マシンインスタンスの接続

フラットプロバイダーネットワークおよび VLAN プロバイダーネットワークを使用して、仮想マシンインスタンスを外部ネットワークに直接接続することができます。

4.1. 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。

関連情報

- [「OpenStack Networking サービスの配置」](#)

4.2. OPENSTACK NETWORKING 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ロール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。

関連情報

- [「OpenStack Networking トポロジーの概要」](#)

4.3. フラットプロバイダーネットワークの設定

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およびそれぞれ別の物理インターフェイスがあり、各コンピュータ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場合に役立ちます。

前提条件

- 複数の物理ネットワークがある。
以下の例では、**physnet1** および **physnet2** という物理ネットワークをそれぞれ使用します。
- 独立した物理インターフェイスがある。
この例では、それぞれ別の物理インターフェイス **eth0** と **eth1** を使用します。

手順

1. アンダークラウドホストに stack ユーザーとしてログインして、カスタム YAML 環境ファイルを作成します。

例

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

ヒント

Red Hat OpenStack Platform Orchestration サービス (heat) は、**テンプレート** と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル** を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、orchestration テンプレートをカスタマイズするための特別な種別のテンプレートです。

2. YAML 環境ファイルの **parameter_defaults** セクションで、**NeutronBridgeMappings** を使用して外部ネットワークへのアクセスに使用する OVS ブリッジを指定します。

例

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

3. コントローラーノードおよびコンピュータノードのカスタム NIC 設定テンプレートで、インターフェイスがアタッチされたブリッジを設定します。

例

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

```

        primary: true
    - type: ovs_bridge
      name: br-net2
      mtu: 1500
      use_dhcp: false
      members:
    - type: interface
      name: eth1
      mtu: 1500
      use_dhcp: false
      primary: true

```

...

4. **openstack overcloud deploy** コマンドを実行し、変更したカスタム NIC テンプレートおよび新しい環境ファイルを含む、テンプレートおよび環境ファイルを追加します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

```

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

```

検証

1. フラットネットワークとして外部ネットワーク (**public1**) を作成し、設定済みの物理ネットワーク (**physnet1**) 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仮想マシンインスタンスを作成できるようにします。

例

```

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

```

2. **openstack subnet create** コマンドを使用して、サブネット (**public_subnet**) を作成します。

例

```

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

```

3. 仮想マシンインスタンスを作成し、それを新たに作成した外部ネットワークに直接接続します。

例

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

関連情報

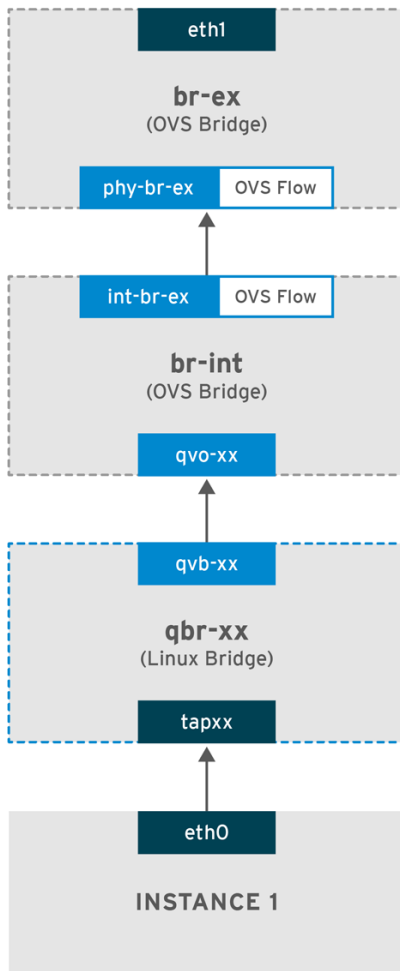
- Advanced Overcloud Customization ガイドの [Custom network interface templates](#)
- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)
- コマンドラインインターフェイスリファレンス の [network create](#)
- コマンドラインインターフェイスリファレンス の [subnet create](#)
- コマンドラインインターフェイスリファレンス の [server create](#)

4.4. 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、物理インターフェイスをブリッジに追加してインスタンスをコンピュータノードに作成すると、得られるインターフェイスとブリッジの設定は、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** インターフェイスから送信され、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. インターフェイス **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、物理インターフェイスに転送されます。

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イスに転送されます。

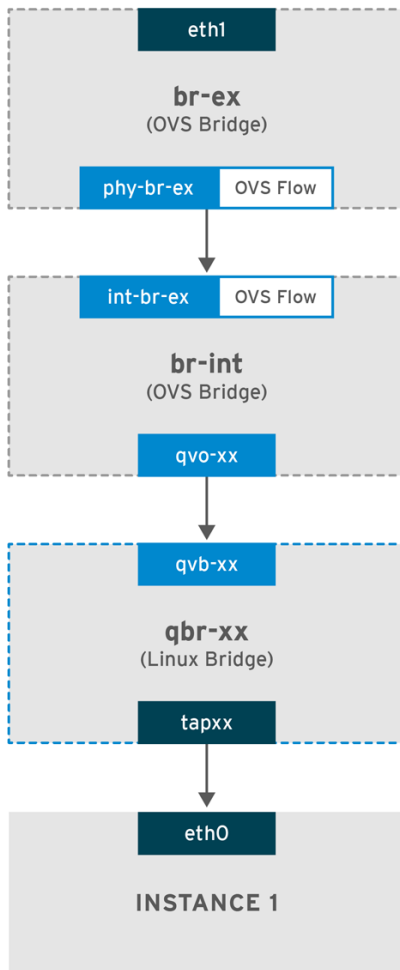
```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=4703.491s, table=0, n_packets=3620, n_bytes=333744, idle_age=0, priority=1
```

```
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
```

物理インターフェイスが別の VLAN タグ付けされたインターフェイスの場合、その物理インターフェイ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イスに到達するまでのフローについて説明します。



OPENSTACK_450456_0617

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**)、**qvovxxx** に転送されます。
3. **qvovxxx** は、VLAN タグを削除した後に、パケットを受け入れて **qvboxx** 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タノードで使ったサンプルの 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タ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。

関連情報

- [「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」](#)

4.5. 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。

手順

1. **bridge_mappings** を確認します。
使用する物理ネットワーク名が **bridge_mapping** 設定の内容と一致していることを確認してください。

例

この例では、物理ネットワーク名は **physnet1** です。

```
$ openstack network show provider-flat
```

出力例

```
...
| provider:physical_network | physnet1
...
```

例

この例では、**bridge_mapping** 設定の内容も **physnet1** です。

```
$ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
```

出力例

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

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。

例

この例では、ネットワーク **provider-flat** に関する詳細が照会されます。

```
$ openstack network show provider-flat
```

出力例

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

3. パッチピアを確認します。
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
```

出力例

```
Bridge br-int
```

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

出力例

br-ex でのパッチピアの設定:

```
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。

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、特定のコンピュータ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。

- a. 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。
- b. 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。

出力例

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"
```

例

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イ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
```

関連情報

- [「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」](#)
- [ブリッジマッピングの設定](#)

4.6. VLAN プロバイダーネットワークの設定

単一の NIC 上の VLAN タグ付けされた複数のインターフェイスを複数のプロバイダーネットワークに接続する場合、これらの新しい VLAN プロバイダーネットワークは仮想マシンインスタンスを直接外部ネットワークに接続できます。

前提条件

- VLAN 範囲を持つ物理ネットワークがある。
以下の例では、VLAN 範囲が **171-172** の **physnet1** と呼ばれる物理ネットワークを使用します。
- ネットワークノードとコンピュータノードが、物理インターフェイスを使用して物理ネットワークに接続されている。
以下の例では、物理インターフェイス **eth1** を使用して物理ネットワーク **physnet1** に接続されているネットワークノードとコンピュータノードを使用します。
- これらのインターフェイ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。

手順

1. アンダークラウドホストに stack ユーザーとしてログインして、カスタム YAML 環境ファイルを作成します。

例

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

ヒント

Red Hat OpenStack Platform Orchestration サービス (heat) は、**テンプレート** と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル** を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、orchestration テンプレートをカスタマイズするための特別な種別のテンプレートです。

2. YAML 環境ファイルの **parameter_defaults** セクションで、**NeutronTypeDrivers** を使用してネットワーク種別ドライバーを指定します。

例

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

3. **NeutronNetworkVLANRanges** 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。

例

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

4. 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。

例

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

5. コアテンプレートおよびこの新しい環境ファイルを含む環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

検証

1. 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。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。

例

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

$ openstack network create \
  --provider-network-type vlan \
```

```
--provider-physical-network physnet1 \
--provider-segment 172 \
provider-vlan172
```

- 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。

例

```
$ 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
```

関連情報

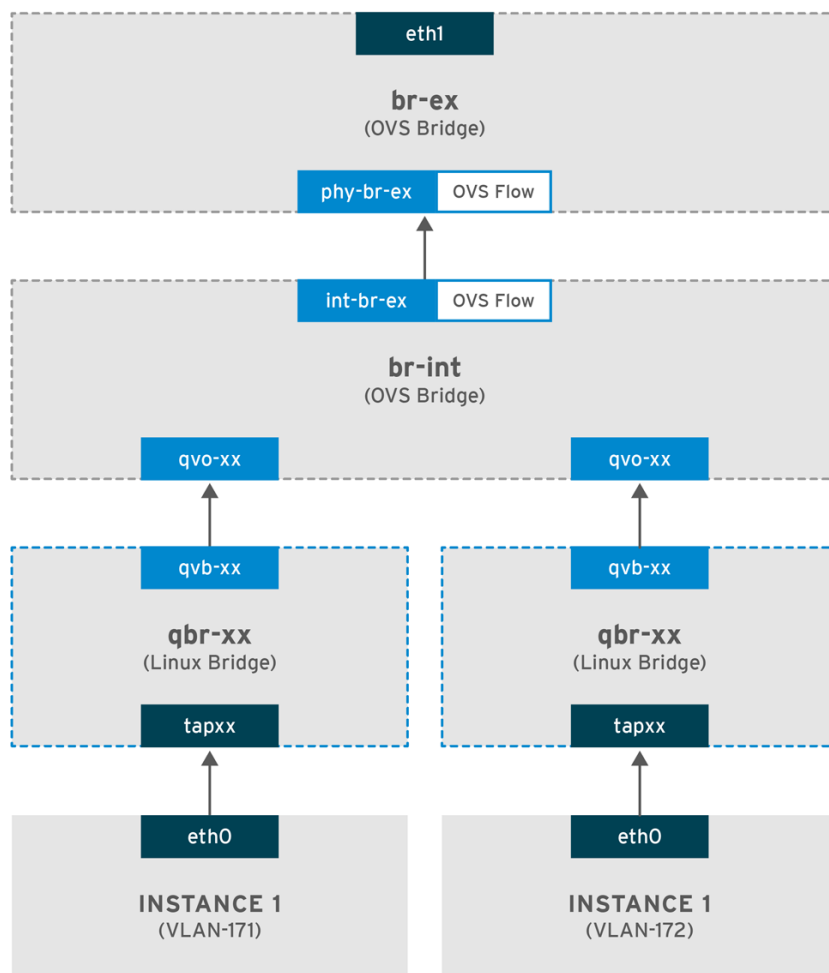
- **Advanced Overcloud Customization** ガイドの [Custom network interface templates](#)
- **Advanced Overcloud Customization** の [Environment files](#)
- **Advanced Overcloud Customization** ガイドの [Including environment files in overcloud creation](#)
- コマンドラインインターフェイスリファレンス の [network create](#)
- コマンドラインインターフェイスリファレンス の [subnet create](#)

4.7. 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、物理インターフェイスをブリッジに追加してインスタンスをコンピュータノードに作成すると、得られるインターフェイスとブリッジの設定は、以下の図のようになります。



OPENSTACK_450456_0617

1. インスタンスの **eth0** インターフェイ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** ↔ **qvoxxx** 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イスに転送します。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イスに転送します。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イス **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、コンピュータ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タ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。

関連情報

- [「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」](#)

4.8. 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。

手順

1. **bridge_mapping** 設定で使用する物理ネットワーク名が物理ネットワーク名と一致することを確認します。

例

```
$ openstack network show provider-vlan171
```

出力例

```
...
| provider:physical_network | physnet1
...
```

例

```
$ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
```

出力例

この出力例では、物理ネットワーク名 **physnet1** が **bridge_mapping** 設定で使用されている名前と一致しています。

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

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。

例

```
$ openstack network show provider-vlan171
```

出力例

```
...
| provider:network_type    | vlan
| provider:physical_network | physnet1
| provider:segmentation_id | 171
...
```

3. パッチピアを確認します。

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
```

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. ネットワークフローを確認します。

- a. 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。
- b. 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。
- c. 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。
- d. 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。

例

```
$ ovs-vsctl show
```

この出力例では、**eth1** は **br-ex** のポートです。

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

例

```
$ ip a
```

出力例

この出力例では、**eth1** がポートとして追加されており、カーネルがすべてのパケットをインターフェイスから OVS ブリッジ **br-ex** に移動するように設定されています。これは、エントリー **master ovs-system** で確認できます。

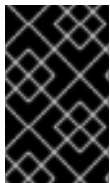
```
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP qlen 1000
```

関連情報

- 「[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](#)」

4.9. ML2/OVS デプロイメントでのプロバイダーネットワーク用マルチキャストスヌーピングの有効化

マルチキャストパケットが Red Hat OpenStack Platform (RHOSP) プロバイダーネットワーク内の全ポートにあふれるのを防ぐには、マルチキャストスヌーピングを有効にする必要があります。Modular Layer 2 プラグインと Open vSwitch メカニズムドライバーの組み合わせ (ML2/OVS) を使用する RHOSP デプロイメントでは、YAML 形式の環境ファイルで RHOSP Orchestration (heat) **NeutronEnableIcmpSnooping** パラメーターを宣言してこれを行います。



重要

マルチキャストスヌーピングの設定を実稼働環境に適用する前に、設定を綿密にテストして理解する必要があります。設定が間違っていると、マルチキャストが中断したり、誤ったネットワーク動作を引き起こしたりする可能性があります。

前提条件

- ML2/OVS プロバイダーネットワークのみを使用する設定にする必要があります。
- 物理ルーターでも IGMP スヌーピングを有効にする必要があります。
つまり、OVS (および物理ネットワーク用) でスヌーピングキャッシュを維持するために、物理ルーターはプロバイダーネットワーク上で IGMP クエリーパケットを送信して、マルチキャストグループメンバーからの通常の IGMP レポートを要求する必要があります。
- 仮想マシンインスタンスへの受信 IGMP を許可する (あるいは、ポートセキュリティを無効にする) には、RHOSP Networking サービスのセキュリティグループルールを設定する必要があります。
以下の例では、**ping_ssh** セキュリティグループに対してルールが作成されます。

例

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

手順

- アンダークラウドホストに stack ユーザーとしてログインして、カスタム YAML 環境ファイルを作成します。

例

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

ヒント

Orchestration サービス (heat) は、テンプレートと呼ばれるプランのセットを使用して環境をインストールおよび設定します。カスタム環境ファイルを使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、heat テンプレートをカスタマイズするための特別な種別のテンプレートです。

2. YAML 環境ファイルの **parameter_defaults** セクションで、**NeutronEnableIcmpSnooping** を **true** に設定します。

```
parameter_defaults:
  NeutronEnableIcmpSnooping: true
  ...
```



重要

コロン (:) と **true** の間に空白文字を追加するようにしてください。

3. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

検証

- マルチキャストスヌーピングが有効であることを確認します。

例

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

出力例

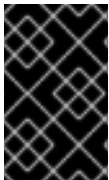
```
...
mcast_snooping_enable: true
...
other_config: {mac-table-size="50000", mcast-snooping-disable-flood-unregistered=True}
...
```

関連情報

- [Component, Plug-In, and Driver Support in Red Hat OpenStack Platform](#)の [Neutron](#)
- [Advanced Overcloud Customization](#) の [Environment files](#)
- [Advanced Overcloud Customization](#) ガイドの [Including environment files in overcloud creation](#)
- [オーバークラウドのパラメーター](#)の [Networking \(neutron\) パラメーター](#)
- [インスタンスの作成と管理](#) ガイドの [セキュリティグループの作成](#)

4.10. ML2/OVN デプロイメントでのマルチキャストの有効化

マルチキャストトラフィックをサポートするには、マルチキャストトラフィックがマルチキャストグループ内の仮想マシン (VM) インスタンスに到達できるように、デプロイメントのセキュリティ設定を変更します。マルチキャストトラフィックのフラッドिंगを防ぐには、IGMP スヌーピングを有効にします。



重要

マルチキャストスヌーピングの設定をテストして十分に理解してから、設定を実稼働環境に適用してください。設定が間違っていると、マルチキャストが中断したり、誤ったネットワーク動作を引き起こしたりする可能性があります。

前提条件

- ML2/OVN メカニズムドライバーを使用する OpenStack デプロイメント

手順

1. セキュリティーを設定し、適切な仮想マシンインスタンスへのマルチキャストトラフィックを許可します。たとえば、セキュリティグループルールのペアを作成し、IGMP クエリーアーからの IGMP トラフィックが仮想マシンインスタンスに到達/発進するのを許可し、3 番目のルールでマルチキャストトラフィックを許可します。

例

セキュリティグループ **mySG** により、IGMP トラフィックが仮想マシンインスタンスに到達/発進するのを許可します。

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

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

別のルールにより、マルチキャストトラフィックが仮想マシンインスタンスに到達するのを許可します。

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

セキュリティグループルールを設定する代わりに、オペレーターはネットワーク上のポートセキュリティを選択的に無効にすることもできます。ポートセキュリティを無効にする場合は、関連するセキュリティリスクについて検討し、対応を計画してください。

2. アンダークラウドノードの環境ファイルで heat パラメーター **NeutronEnableIcmpSnooping: True** を設定します。たとえば、以下の行を `ovn-extras.yaml` に追加します。

例

```
parameter_defaults:
    NeutronEnableIcmpSnooping: True
```

3. この環境ファイルをご自分の環境に該当するその他の環境ファイルと共に **openstack overcloud deploy** コマンドに追加して、オーバークラウドをデプロイします。

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

-e ovn-extras.yaml \
...
```

<other_overcloud_environment_files> を既存のデプロイメントに含まれる環境ファイルの一覧に置き換えます。

検証

1. マルチキャストスヌーピングが有効であることを確認します。ノースバウンドデータベースの `Logical_Switch` テーブルを一覧表示します。

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

出力例

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

Networking サービス (neutron) の `igmp_snooping_enable` 設定は、OVN ノースバウンドデータベースの `Logical_Switch` テーブルの `other_config` 列で設定される `mcast_snoop` オプションに変換されます。`mcast_flood_unregistered` は常に `false` である点に注意してください。

2. IGMP グループを表示します。

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

出力例

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

関連情報

- [Component, Plug-In, and Driver Support in Red Hat OpenStack Platform](#) の [Neutron](#)
- [Advanced Overcloud Customization](#) の [Environment files](#)
- [Advanced Overcloud Customization ガイド](#) の [Including environment files in overcloud creation](#)

4.11. 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
```

4.12. 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5章 FLOATING IP アドレスの管理

プライベートの Fixed IP アドレスを持つことに加え、仮想マシンインスタンスには、他のネットワークと通信するためのパブリックまたは Floating IP アドレスを持たせることができます。本項では、Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) で Floating IP を作成し、管理する方法を説明します。

5.1. 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 形式) 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 を使用することができます。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。

手順

- 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。

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

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。

例

```
$ 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
```

検証

- インスタンスにランダムな Floating IP を割り当てること、プールが適切に設定されていることを確認できます。(下記のリンクを参照してください。)

関連情報

- コマンドラインインターフェイスリファレンスの [subnet create](#)
- [Floating IP アドレスの無作為な割り当て](#)

5.2. 特定の FLOATING IP アドレスの割り当て

特定の Floating IP アドレスを仮想マシンインスタンスに割り当てることが可能です。

手順

- **openstack server add floating ip** コマンドを使用して、Floating IP アドレスをインスタンスに確保します。

例

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

検証手順

- **openstack server show** コマンドを使用して、Floating IP がインスタンスに割り当てられていることを確認します。

例

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

出力例

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

```

| updated          | 2021-08-11T14:45:37Z          |
| user_id          | 4b7e19a0d723310fd92911eb2fe59743a3a5cd32 |
|                  | 45f76ffced91096196f646b5      |
| volumes_attached |                               |
+-----+-----+

```

関連情報

- コマンドラインインターフェイスリファレンス の [server add floating ip](#)
- コマンドラインインターフェイスリファレンス の [server show](#)
- [Floating IP アドレスの無作為な割り当て](#)

5.3. 高度なネットワークの作成

管理者は、**管理** の画面から Dashboard で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。

手順

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。

関連情報

- [特定の Floating IP アドレスの割り当て](#)

- Floating IP アドレスの無作為な割り当て

5.4. FLOATING IP アドレスの無作為な割り当て

外部 IP アドレスのプールから、仮想マシンインスタンスに Floating IP アドレスを動的に確保することができます。

前提条件

- ルーティング可能な外部 IP アドレスのプール
詳細は、「[Floating IP アドレスプールの作成](#)」を参照してください。

手順

1. 以下のコマンドを入力して、プールから Floating IP アドレスを確保します。以下の例では、ネットワークは **public** という名前です。

例

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

出力例

以下の例では、新たに確保された Floating IP は **192.0.2.200** です。これをインスタンスに割り当てることができます。

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

2. 以下のコマンドを入力して、インスタンスを探します。

```
$ openstack server list
```

出力例

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

3. インスタンス名または ID を Floating IP に関連付けます。

例

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

検証手順

- 以下のコマンドを入力して、Floating IP がインスタンスに割り当てられていることを確認します。

例

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

出力例

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

関連情報

- コマンドラインインターフェイスリファレンス の [floating ip create](#)
- コマンドラインインターフェイスリファレンス の [server add floating ip](#)
- コマンドラインインターフェイスリファレンス の [server show](#)
- [Floating IP アドレスプールの作成](#)

5.5. 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。

手順

- `/var/lib/config-data/puppet-generated/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ようになります。

関連情報

- [Floating IP アドレスプールの作成](#)
- [Floating IP アドレスの無作為な割り当て](#)

5.6. 物理ネットワークのブリッジ

仮想インスタン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イス **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。

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。

手順

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)
以下に例を示します。

```
# 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。

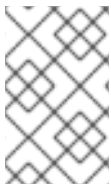
関連情報

- [ブリッジマッピングの設定](#)

5.7. インターフェイスの追加

ルーターとサブネットを橋渡しするインターフェイ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イスを追加して、新しいインターフェイ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、**+インターフェイスの追加** 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。インターフェイスに対して 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イス接続が反映されます。

5.8. インターフェイスの削除

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イスを削除することができます。

インターフェイスを削除するには、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イ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. インターフェイス種別 (**内部インタフェース**) を選択し、**インターフェイスの削除** をクリックします。

第6章 ネットワークのトラブルシューティング

Red Hat OpenStack Platform 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。ML2/OVS ネットワークとデフォルトの ML2/OVN ネットワークのトラブルシューティングには、いくつかの違いがあります。

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。

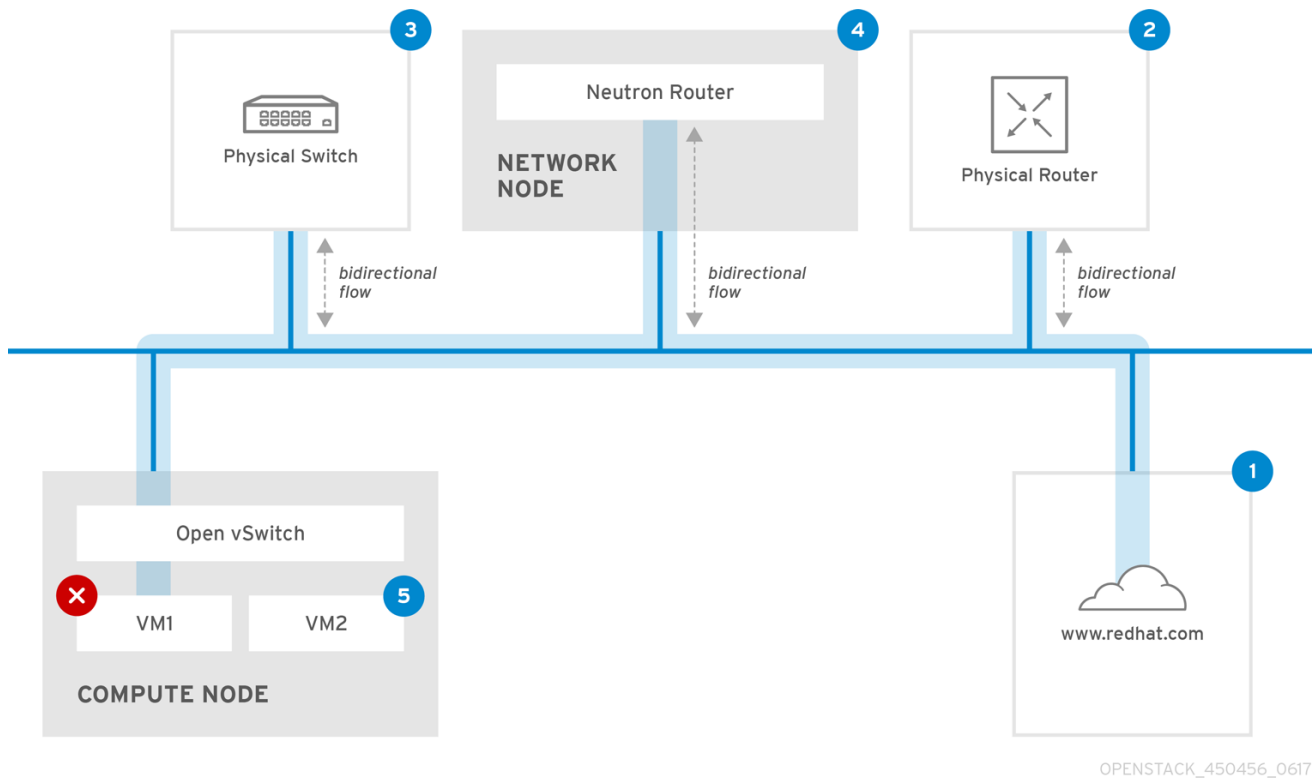
```
$ ping www.example.com
```

```
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.example.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イスです。
 - **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:** 同じコンピュータノード上にある、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 のログファイルを確認します。

6.2. ポートの現在のステータスの表示

基本的なトラブルシューティングタスクでは、ルーターに接続されたすべてのポートのインベントリを作成し、ポートのステータス (**DOWN** または **ACTIVE**) を判断します。

手順

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。

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

出力例

```
+-----+-----+-----+-----+
| id                | name | mac_address    | fixed_ips |
|                   |      |                 |            |
+-----+-----+-----+-----+
| b58d26f0-cc03-43c1-ab23-ccdb1018252a |      | fa:16:3e:94:a7:df | {"subnet_id": "a592fdbabab-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
```

出力例

```

+-----+-----+
+
| 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:vnictype | normal                                 |
| device_id       | 49c6ebdc-0e62-49ad-a9ca-58cea464472f   |
| device_owner    | network:router_interface               |
| extra_dhcp_opts |                                         |
| fixed_ips       | {"subnet_id": "a592fdbb-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、ステータスを判断します。

6.3. VLAN プロバイダーネットワークへの接続に関するトラブルシューティング

OpenStack Networking 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。

手順

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イスの 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
```

6.4. VLAN 設定とログファイルの確認

デプロイメントの検証またはトラブルシューティングに役立つように、以下を実行できます。

- Red Hat Openstack Platform (RHOSP) Networking サービス (neutron) エージェントの登録およびステータスを確認する。
- VLAN 範囲などのネットワーク設定値を検証する。

手順

1. `openstack network agent list` コマンドを使用して、RHOSP Networking サービスのエージェントが稼働し、正しいホスト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            |
+-----+-----+-----+-----+-----+
```

2. `/var/log/containers/neutron/openvswitch-agent.log` を確認します。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を確認します。
3. `/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。

4. `/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。
5. **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。

6.5. ML2/OVN 名前空間内での基本的な ICMP テストの実行

基本的なトラブルシューティングステップとして、同じレイヤー 2 ネットワーク上にある OVN メタデータインターフェイスからインスタンスに ping 送信を試みることができます。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVN を使用する RHOSP デプロイメント。

手順

1. Red Hat OpenStack Platform の認証情報を使用してオーバークラウドにログインします。
2. **openstack server list** コマンドを実行して、仮想マシンインスタンスの名前を取得します。
3. **openstack server show** コマンドを実行して、インスタンスを実行しているコンピューターノードを確認します。

例

```
$ openstack server show my_instance -c OS-EXT-SRV-ATTR:host \
-c addresses
```

出力例

```
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| OS-EXT-SRV-ATTR:host | compute0.overcloud.example.com         |
| addresses         | finance-network1=192.0.2.2; provider-  |
|                   | storage=198.51.100.13                  |
+-----+-----+
```

4. コンピューターノードホストにログインします。

例

```
$ ssh heat-admin@compute0.example.com
```

5. **ip netns list** コマンドを実行して、OVN メタデータ名前空間を表示します。

出力例

```
ovnmeta-07384836-6ab1-4539-b23a-c581cf072011 (id: 1)
ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa (id: 0)
```

- メタデータ名前空間を使用して、**ip netns exec** コマンドを実行して、関連付けられたネットワークに ping を実行します。

例

```
$ sudo ip netns exec ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa \
ping 192.0.2.2
```

出力例

```
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.470 ms
64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.183 ms
64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.296 ms
64 bytes from 192.0.2.2: icmp_seq=5 ttl=64 time=0.307 ms
^C
--- 192.0.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 122ms
rtt min/avg/max/mdev = 0.183/0.347/0.483/0.116 ms
```

関連情報

- コマンドラインインターフェイスリファレンス の [server show](#)

6.6. プロジェクトネットワーク (ML2/OVS) 内からのトラブルシューティング

Red Hat Openstack Platform (RHOSP) ML2/OVS ネットワークでは、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。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVS を使用する RHOSP デプロイメント。

手順

- 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て、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。

```
$ openstack network list
```

この出力では、**web-servers** ネットワークの ID (**9cb32fe0-d7fb-432c-b116-f483c6497b08**) に注意してください。このコマンドは、ネットワーク ID をネットワーク名前空間に追加します。これにより、次のステップで名前空間を特定できます。

出力例

```
+-----+-----+-----+
| 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 |
+-----+-----+-----+
```

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
```

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。

以下の例では、名前空間は **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08** です。

出力例

```
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
```

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。
以下の例では、**route -n** コマンドを使用しています。

例

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

出力例

```
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
```

6.7. 名前空間内での高度な ICMP テストの実行 (ML2/OVS)

tcpdump と **ping** コマンドの組み合わせを使用して、Red Hat Openstack Platform (RHOSP) ML2/OVS ネットワークのトラブルシューティングを行うことができます。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVS を使用する RHOSP デプロイメント。

手順

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

例

```
# 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。

例

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

出力例

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



注記

トラフィックの **tcpdump** 分析を実行すると、仮想マシンインスタンスではなくルーターインターフェイス方向の応答パケットが表示されます。**qrouter** によりリターンパケットで Destination Network Address Translation (DNAT) が実行されるので、これは想定どおりの動作です。

6.8. OVN トラブルシューティングコマンドのエイリアスの作成

ovn_controller コンテナで **ovn-nbctl show** などの OVN コマンドを実行します。コンテナはコントローラーノードおよびコンピューターノードで実行します。コマンドへのアクセスを簡素化するには、エイリアスを定義するスクリプトを作成し **source** コマンドでスクリプトファイルを読み込みます。

前提条件

- デフォルトのメカニズムドライバーとして ML2/OVN を使用する Red Hat OpenStack Platform のデプロイメント

手順

1. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコントローラーホストにログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. 実行する **ovn** コマンドが含まれるシェルスクリプトファイルを作成します。

```
ov
```

例

```
vi ~/bin/ovn-alias.sh
```

3. **ovn** コマンドを追加して、スクリプトファイルを保存します。

例

この例では、**ovn-sbctl** コマンド、**ovn-nbctl** コマンド、および **ovn-trace** コマンドがエイリアスファイルに追加されています。

```
EXTERNAL_ID=\
$(sudo ovs-vsctl get open . external_ids:ovn-remote | awk -F: '{print $2}')
export NBDB=tcp:${EXTERNAL_ID}:6641
export SBDB=tcp:${EXTERNAL_ID}:6642

alias ovn-sbctl="sudo podman exec ovn_controller ovn-sbctl --db=$SBDB"
alias ovn-nbctl="sudo podman exec ovn_controller ovn-nbctl --db=$NBDB"
alias ovn-trace="sudo podman exec ovn_controller ovn-trace --db=$SBDB"
```

4. コンピュートホストで、この手順のステップを繰り返します。

検証

1. **source** コマンドでスクリプトファイルを読み込みます。

例

```
# source ovn-alias.sh
```

2. コマンドを実行して、スクリプトファイルが正しく動作することを確認します。

例

```
# ovn-nbctl show
```

出力例

```
switch 26ce22db-1795-41bd-b561-9827cbd81778 (neutron-f8e79863-6c58-43d0-8f7d-
8ec4a423e13b) (aka internal_network)
port 1913c3ae-8475-4b60-a479-df7bcce8d9c8
  addresses: ["fa:16:3e:33:c1:fc 192.168.254.76"]
port 1aabaee3-b944-4da2-bf0a-573215d3f3d9
  addresses: ["fa:16:3e:16:cb:ce 192.168.254.74"]
port 7e000980-59f9-4a0f-b76a-4fdf4e86f27b
  type: localport
  addresses: ["fa:16:3e:c9:30:ed 192.168.254.2"]
```

関連情報

- **ovn-nbctl --help** コマンド
- **ovn-sbctl --help** コマンド

- **ovn-trace --help** コマンド

6.9. OVN の論理フローのモニターリング

OVN は論理フローを使用します。これは、優先度、マッチング、アクションで設定されるフローのテーブルです。これらの論理フローは、各 Red Hat Openstack Platform (RHOSP) コンピュートノード上で実行される **ovn-controller** に分散されます。コントローラーノード上で **ovn-sbctl lflow-list** コマンドを使用して、論理フローの完全なセットを表示します。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVN を使用する RHOSP デプロイメント。
- OVN データベースコマンドのエイリアスファイルを作成します。
「[OVN トラブルシューティングコマンドのエイリアスの作成](#)」を参照してください。

手順

1. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコントローラーホストにログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. OVN データベースコマンドのエイリアスファイルを入手します。
詳細は、「[OVN トラブルシューティングコマンドのエイリアスの作成](#)」を参照してください。

例

```
source ~/ovn-alias.sh
```

3. 論理フローを表示します。

```
$ ovn-sbctl lflow-list
```

4. 出力を確認します。

出力例

```
Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10) Pipeline: ingress
table=0 (ls_in_port_sec_l2 ), priority=100 , match=(eth.src[40]), action=(drop;)
table=0 (ls_in_port_sec_l2 ), priority=100 , match=(vlan.present), action=(drop;)
table=0 (ls_in_port_sec_l2 ), priority=50 , match=(inport == "sw0-port1" && eth.src ==
{00:00:00:00:00:01}), action=(next;)
table=0 (ls_in_port_sec_l2 ), priority=50 , match=(inport == "sw0-port2" && eth.src ==
{00:00:00:00:00:02}), action=(next;)
table=1 (ls_in_port_sec_ip ), priority=0 , match=(1), action=(next;)
table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && arp.sha == 00:00:00:00:00:01), action=(next;)
table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port1" && eth.src ==
```

```

00:00:00:00:00:01 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
00:00:00:00:00:01) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:01))))), action=
(next;)
    table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && arp.sha == 00:00:00:00:00:02), action=(next;)
    table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
00:00:00:00:00:02) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:02))))), action=
(next;)
    table=2 (ls_in_port_sec_nd ), priority=80 , match=(inport == "sw0-port1" && (arp || nd)),
action=(drop;)
    table=2 (ls_in_port_sec_nd ), priority=80 , match=(inport == "sw0-port2" && (arp || nd)),
action=(drop;)
    table=2 (ls_in_port_sec_nd ), priority=0 , match=(1), action=(next;)
    table=3 (ls_in_pre_acl ), priority=0, match=(1), action=(next;)
    table=4 (ls_in_pre_lb ), priority=0 , match=(1), action=(next;)
    table=5 (ls_in_pre_stateful ), priority=100 , match=(reg0[0] == 1), action=(ct_next;)
    table=5 (ls_in_pre_stateful ), priority=0 , match=(1), action=(next;)
    table=6 (ls_in_acl ), priority=0 , match=(1), action=(next;)
    table=7 (ls_in_qos_mark ), priority=0 , match=(1), action=(next;)
    table=8 (ls_in_lb ), priority=0 , match=(1), action=(next;)
    table=9 (ls_in_stateful ), priority=100 , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
    table=9 (ls_in_stateful ), priority=100 , match=(reg0[2] == 1), action=(ct_lb;)
    table=9 (ls_in_stateful ), priority=0 , match=(1), action=(next;)
    table=10 (ls_in_arp_rsp ), priority=0 , match=(1), action=(next;)
    table=11 (ls_in_dhcp_options ), priority=0 , match=(1), action=(next;)
    table=12 (ls_in_dhcp_response), priority=0 , match=(1), action=(next;)
    table=13 (ls_in_l2_lkup ), priority=100 , match=(eth.mcast), action=(output =
"_MC_flood"; output;)
    table=13 (ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:01), action=
(output = "sw0-port1"; output;)
    table=13 (ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:02), action=
(output = "sw0-port2"; output;)
Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10) Pipeline: egress
    table=0 (ls_out_pre_lb ), priority=0 , match=(1), action=(next;)
    table=1 (ls_out_pre_acl ), priority=0 , match=(1), action=(next;)
    table=2 (ls_out_pre_stateful), priority=100 , match=(reg0[0] == 1), action=(ct_next;)
    table=2 (ls_out_pre_stateful), priority=0 , match=(1), action=(next;)
    table=3 (ls_out_lb ), priority=0 , match=(1), action=(next;)
    table=4 (ls_out_acl ), priority=0 , match=(1), action=(next;)
    table=5 (ls_out_qos_mark ), priority=0 , match=(1), action=(next;)
    table=6 (ls_out_stateful ), priority=100 , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
    table=6 (ls_out_stateful ), priority=100 , match=(reg0[2] == 1), action=(ct_lb;)
    table=6 (ls_out_stateful ), priority=0 , match=(1), action=(next;)
    table=7 (ls_out_port_sec_ip ), priority=0 , match=(1), action=(next;)
    table=8 (ls_out_port_sec_l2 ), priority=100 , match=(eth.mcast), action=(output;)
    table=8 (ls_out_port_sec_l2 ), priority=50 , match=(output == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
    table=8 (ls_out_port_sec_l2 ), priority=50 , match=(output == "sw0-port2" && eth.dst ==
{00:00:00:00:00:02}), action=(output;)

```

OVN と OpenFlow には、主に以下のような相違点があります。

- OVN ポートは、ネットワーク内にある論理エンティティで、単一のスイッチ上にある物理ポートではありません。
- OVN により、パイプライン内の各テーブルには番号に加えて名前が付けられます。名前は、パイプライン内のそのステージの目的を示します。
- OVN の match 構文は、複雑なブール表現をサポートしています。
- OVN の論理フローでは、OpenFlow よりも幅広いアクションをサポートしています。OVN の論理フローの構文で DHCP などの高度な機能を実装することができます。

5. OVN トレースを実行します。

ovn-trace コマンドを使用して、パケットが OVN の論理フローをどのように通過するかシミュレーションしたり、パケットがドロップする原因を特定するのに役立てたりすることができます。**ovn-trace** コマンドには、以下のパラメーターを指定して実行してください。

DATAPATH

シミュレーションされるパケットの送信が開始される場所の論理スイッチまたは論理ルーター。

MICROFLOW

シミュレーションされるパケット。**ovn-sb** データベースで使用される構文で指定します。

例

この例では、シミュレーションされるパケットに **--minimal** の出力オプションが示されており、そのパケットが宛先に到達したことを表しています。

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

出力例

```
#
reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x
0000
    output("sw0-port2");
```

例

さらに詳しい情報を表示するには、シミュレーションされる同じパケットの **--summary** 出力に完全な実行パイプラインが表示されます。

```
$ ovn-trace --summary sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

出力例

出力例は次のとおりです。

- パケットは **sw0-port1** ポートから **sw0** ネットワークに入り、受信のパイプラインを通過します。
- **output** 変数が **sw0-port2** に設定されているのは、このパケットの宛先が **sw0-port2** に指定されていることを意味します。

- パケットは受信のパイプラインから出力されます。このパイプラインは、**output** 変数が **sw0-port2** に設定された **sw0** の送信パイプラインにパケットを送ります。
- 出力のアクションは、送信のパイプラインで実行されます。このパイプラインでは、パケットが **output** 変数の現在の値である **sw0-port2** に出力されます。

```
#
reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type
=0x0000
ingress(dp="sw0", inport="sw0-port1") {
    output = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", output="sw0-port2") {
        output;
        /* output to "sw0-port2", type "" */;
    };
};
```

関連情報

- [「OVN トラブルシューティングコマンドのエイリアスの作成」](#)
- **ovn-sbctl --help** コマンド
- **ovn-trace --help** コマンド

6.10. OPENFLOWS のモニターリング

ovs-ofctl dump-flows コマンドを使用して、Red Hat Openstack Platform (RHOSP) ネットワーク内の論理スイッチ上の OpenFlow のフローをモニターリングすることができます。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVN を使用する RHOSP デプロイメント。

手順

1. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコントローラーホストにログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. **ovs-ofctl dump-flows** コマンドを実行します。

例

```
$ sudo ovs-ofctl dump-flows br-int
```

3. 出力を確認します。以下の様な出力になるはずです。

出力例

```
$ ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=72.132s, table=0, n_packets=0, n_bytes=0, idle_age=72,
  priority=10,in_port=1,dl_src=00:00:00:00:00:01 actions=resubmit(,1)
  cookie=0x0, duration=60.565s, table=0, n_packets=0, n_bytes=0, idle_age=60,
  priority=10,in_port=2,dl_src=00:00:00:00:00:02 actions=resubmit(,1)
  cookie=0x0, duration=28.127s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0
  actions=drop
  cookie=0x0, duration=13.887s, table=1, n_packets=0, n_bytes=0, idle_age=13,
  priority=0,in_port=1 actions=output:2
  cookie=0x0, duration=4.023s, table=1, n_packets=0, n_bytes=0, idle_age=4,
  priority=0,in_port=2 actions=output:1
```

関連情報

- **ovs-ofctl --help** コマンド

6.11. ML2/OVN デプロイメントの検証

Red Hat OpenStack Platform (RHOSP) デプロイメントの ML2/OVN ネットワークを検証するには、テストネットワークとサブネットを作成し、特定のコンテナが実行中であることを検証するなどの診断タスクを実行します。

前提条件

- Networking サービス (neutron) のデフォルトメカニズムドライバーとして ML2/OVN を使用する RHOSP の新規デプロイメント
- OVN データベースコマンドのエイリアスファイルを作成します。
[「OVN トラブルシューティングコマンドのエイリアスの作成」](#) を参照してください。

手順

1. テストネットワークおよびサブネットを作成します。

```
NETWORK_ID=\
$(openstack network create internal_network | awk '/ id/ {print $4}')

openstack subnet create internal_subnet \
--network $NETWORK_ID \
--dns-nameserver 8.8.8.8 \
--subnet-range 192.168.254.0/24
```

エラーが発生した場合は、以下の手順を実行します。

2. 関連するコンテナがコントローラーホストで実行していることを確認します。
 - a. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコントローラーホストにログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

- b. 以下のコマンドを入力します。

```
sudo podman ps -a --format="{{.Names}}"|grep ovn
```

次のサンプルに示すように、出力には OVN コンテナが一覧表示されます。

出力例

```
ovn-dbs-bundle-podman-0
ovn_dbs_init_bundle
ovn_controller
```

3. 関連するコンテナがコンピュータホストで実行していることを確認します。

- a. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコンピュータホストにログインします。

例

```
$ ssh heat-admin@compute-0.ctlplane
```

- b. 以下のコマンドを入力します。

```
$ sudo podman ps -a --format="{{.Names}}"|grep ovn
```

次のサンプルに示すように、出力には OVN コンテナが一覧表示されます。

出力例

```
ovn_controller
ovn_metadata_agent
neutron-haproxy-ovnm-meta-26f493a7-1141-416a-9288-f08ff45fccac
neutron-haproxy-ovnm-meta-b81bd1f1-0ff4-4142-8706-0f295db3c3af
```

4. ログファイルでエラーメッセージの有無を確認します。

```
grep -r ERR /var/log/containers/openvswitch/ /var/log/containers/neutron/
```

5. エイリアスファイルを読み込んで、OVN データベースコマンドを実行します。
詳細は、「[OVN トラブルシューティングコマンドのエイリアスの作成](#)」を参照してください。

例

```
$ source ~/ovn-alias.sh
```

6. ノースバウンドデータベースおよびサウスバウンドデータベースをクエリーして応答するかどうを確認します。

例

```
# ovn-nbctl show
# ovn-sbctl show
```

7. 同じレイヤー 2 ネットワーク上にある OVN メタデータインターフェイスからインスタンスに ping 送信を試みます。
詳細は、「[ML2/OVN 名前空間内での基本的な ICMP テストの実行](#)」を参照してください。
8. サポートのために Red Hat に問い合わせる必要がある場合は、Red Hat ソリューション [How to collect all required logs for Red Hat Support to investigate an OpenStack issue](#) に記載されている手順を実施します。

関連情報

- コマンドラインインターフェイスリファレンスの [network create](#)
- コマンドラインインターフェイスリファレンスの [subnet create](#)
- 「[OVN トラブルシューティングコマンドのエイリアスの作成](#)」
- `ovn-nbctl --help` コマンド
- `ovn-sbctl --help` コマンド

6.12. ML2/OVN のロギングモードの設定

追加のトラブルシューティング情報を取得するために、ML2/OVN ロギングを debug モードに設定します。追加のデバッグ情報が必要ない場合は、ロギングを info モードに戻して、ディスク領域の使用量を減らします。

前提条件

- デフォルトのメカニズムドライバーとして ML2/OVN を使用する Red Hat OpenStack Platform のデプロイメント

手順

1. ロギングモードを設定するコントローラーまたはコンピューターノードに、OVN コンテナにアクセスするために必要な権限を持つユーザーとしてログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. ML2/OVN ロギングモードを設定します。

デバッグロギングモード

```
$ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set dbg
```

情報ロギングモード

```
$ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set info
```

検証

- **ovn-controller** コンテナログにデバッグメッセージが含まれていることを確認します。

```
$ sudo grep DBG /var/log/containers/openvswitch/ovn-controller.log
```

出力例

文字列 **|DBG|** を含む最近のログメッセージが表示されるはずです。

```
2022-09-29T20:52:54.638Z|00170|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-
int.mgmt: received: OFPT_ECHO_REQUEST (OF1.5) (xid=0x0): 0 bytes of payload
2022-09-29T20:52:54.638Z|00171|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-
int.mgmt: sent (Success): OFPT_ECHO_REPLY (OF1.5) (xid=0x0): 0 bytes of payload
```

- **ovn-controller** コンテナログに次のような文字列が含まれていることを確認します。

```
...received request vlog/set["info"], id=0
```

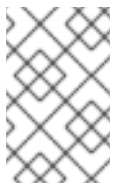
関連情報

- [「ML2/OVN ログファイル」](#)

6.13. エッジサイトへの登録に失敗する OVN コントローラーの修正

問題

Red Hat OpenStack Platform (RHOSP) エッジサイトの OVN コントローラーが登録に失敗します。



注記

このエラーは、**以前の** RHOSP バージョン (RHOSP 16.1.7 以前または RHOSP 16.2.0) から更新された RHOSP 16.1 ML2/OVN デプロイメントで発生する可能性があります。

サンプルエラー

発生したエラーは次のようなものです。

```
2021-04-12T09:14:48.994Z|04754|ovsdb_idl|WARN|transaction error: {"details": "Transaction
causes multiple rows in \"Encap\" table to have identical values (geneve and \"10.14.2.7\") for
index on columns \"type\" and \"ip\". First row, with UUID 3973cad5-eb8a-4f29-85c3-
c105d861c0e0, was inserted by this transaction. Second row, with UUID f06b71a8-4162-475b-
8542-d27db3a9097a, existed in the database before this transaction and was not modified by the
transaction.", "error": "constraint violation"}
```

原因

ovn-controller プロセスがホスト名を置き換える場合は、別の encap エントリーを含む別のシャーシエントリーを登録します。詳細は、[BZ#1948472](#) を参照してください。

解決方法

問題を解決するには、次の手順に従います。

1. まだ作成していない場合は、この手順で、後で使用する必要な OVN データベースコマンドのエイリアスを作成します。
詳細については、[OVN トラブルシューティングコマンドのエイリアスの作成](#) を参照してください。
2. OVN コンテナにアクセスするために必要な権限を持つユーザーとしてコントローラーホストにログインします。

例

```
$ ssh heat-admin@controller-0.ctlplane
```

3. `/var/log/containers/openvswitch/ovn-controller.log` から IP アドレスを取得します。
4. IP アドレスが正しいことを確認します。

```
ovn-sbctl list encap |grep -a3 <IP address from ovn-controller.log>
```

5. IP アドレスを含むシャーシを削除します。

```
ovn-sbctl chassis-del <chassis-id>
```

6. **Chassis_Private** テーブルをチェックして、シャーシが削除されたことを確認します。

```
ovn-sbctl find Chassis_private chassis="[]"
```

7. エントリーが報告された場合は、次のコマンドでそれらを削除します。

```
$ ovn-sbctl destroy Chassis_Private <listed_id>
```

8. 次のコンテナを再起動します。

- **tripleo_ovn_controller**
- **tripleo_ovn_metadata_agent**

```
$ sudo systemctl restart tripleo_ovn_controller
$ sudo systemctl restart tripleo_ovn_metadata_agent
```

検証

- OVN エージェントが実行していることを確認します。

```
$ openstack network agent list -c "Agent Type" -c State -c Binary
```

出力例

```
+-----+-----+-----+
| Agent Type          | State | Binary          |
+-----+-----+-----+
| OVN Controller Gateway agent | UP    | ovn-controller  |
| OVN Controller Gateway agent | UP    | ovn-controller  |
```

OVN Controller agent	UP	ovn-controller	
OVN Metadata agent	UP	neutron-ovn-metadata-agent	
OVN Controller Gateway agent	UP	ovn-controller	
+-----+-----+-----+			

6.14. ML2/OVN ログファイル

ログファイルは、ML2/OVN メカニズムドライバーのデプロイメントと操作に関連するイベントを追跡します。

表6.1 ノードごとの ML2/OVN ログファイル

ノード	Log	パス /var/log/containers/openvswi tch...
コントローラー、コンピュー ト、ネットワーク	OVS ノースバウンドデータベ ースサーバー	.../ovn-controller.log
Controller	OVS ノースバウンドデータベ ースサーバー	.../ovsdb-server-nb.log
Controller	OVS サウスバウンドデータベ ースサーバー	.../ovsdb-server-sb.log
Controller	OVN ノースバウンドデータベ ースサーバー	.../ovn-northd.log

第7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。特定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。

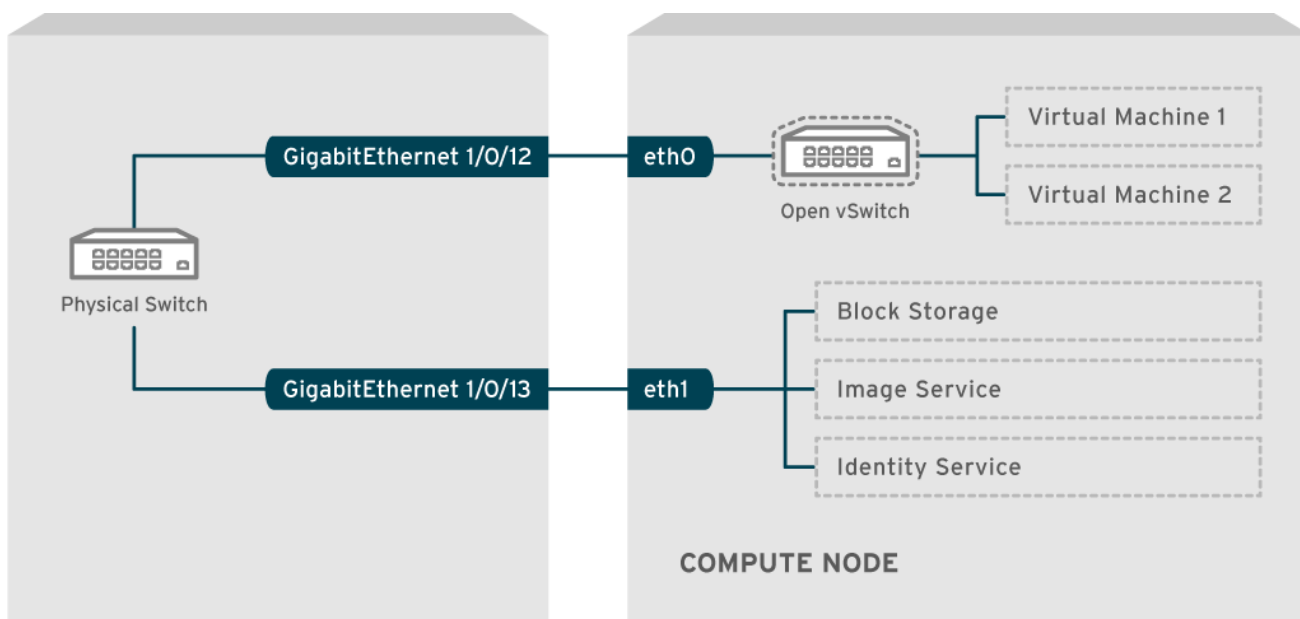
7.1. 物理ネットワーク環境のプランニング

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、コンピュータ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タ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。

図7.1 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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.2. CISCO CATALYST スイッチの設定

7.2.1. 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することができます。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タ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。

7.2.2. 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 (インターフェイス 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	このインターフェイスを識別するのに使用する一意の説明的な値。
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。

フィールド	説明
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。

7.2.3. アクセスポートについて

コンピュータノード上の全 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kstレーズ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。

7.2.4. Cisco Catalyst スイッチでのアクセスポートの設定

- 図7.1「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 (インターフェイス 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。

フィールド	説明
description Access port for Compute Node	このインターフェイ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。コンピュータノードには、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。

7.2.5. LACP ポートアグリゲーションについて

リンクアグリゲーション制御プロトコル (LACP) を使用して、複数の物理 NIC をバンドルし、単一の論理チャネルを形成できます。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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.2.6. 物理 NIC 上での LACP の設定

物理 NIC でリンクアグリゲーション制御プロトコル (LACP) を設定できます。

手順

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"
```

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.2.7. 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。

手順

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

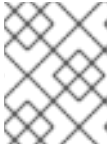
interface GigabitEthernet1/0/13
    switchport access vlan 100
    switchport mode access
    speed 1000
    duplex full
    channel-group 10 mode active
    channel-protocol lacp
```

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。

7.2.8. 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。

関連情報

- [最大伝送単位 \(MTU\) 設定の定義](#)

7.2.9. 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イスごとではなく、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イスごとの MTU 設定をサポートしていれば、この機能を使用の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.

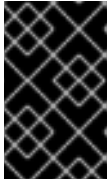
sw01(config)# system mtu jumbo 9000
Changes to the system jumbo MTU will not take effect until the next reload is done
```



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み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イスに適用され、**Jumbo MTU** は全ギガビット対応インターフェイ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
```

7.2.10. 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。

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

手順

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。

7.3. CISCO NEXUS スイッチの設定

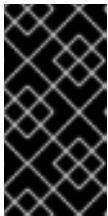
7.3.1. トランクポートについて

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することができます。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タ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。

7.3.2. 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 (インターフェイス **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
```

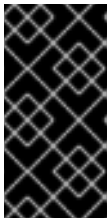
7.3.3. アクセスポートについて

コンピュータノード上の全 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。

7.3.4. Cisco Nexus スイッチでのアクセスポートの設定

手順

- [図7.1「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 (インターフェイス **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
```

7.3.5. LACP ポートアグリゲーションについて

リンクアグリゲーション制御プロトコル (LACP) を使用して、複数の物理 NIC をバンドルし、単一の論理チャネルを形成できます。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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.3.6. 物理 NIC 上での LACP の設定

物理 NIC でリンクアグリゲーション制御プロトコル (LACP) を設定できます。

手順

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 を使用するよう設定します。

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

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.3.7. Cisco Nexus スイッチでの LACP の設定

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

手順

1. コンピュータノードの 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イスの代わりにトランクインターフェイ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イスなので、アクセスポートだけを設定の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イスには 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
```

注記

PXE を使用して Cisco スイッチ上のノードをプロビジョニングする場合、ポートを立ち上げてサーバーを起動するために、オプション **no lacp graceful-convergence** および **no lacp suspend-individual** を設定する必要がある場合があります。詳細は、Cisco スイッチのドキュメントを参照してください。

7.3.8. 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。

関連情報

- [最大伝送単位 \(MTU\) 設定の定義](#)

7.3.9. 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イスに、MTU の設定を適用します。

手順

- 以下のコマンドを実行して、インターフェイス 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
```

7.3.10. 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。

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

手順

- Cisco Nexus 7000 シリーズスイッチ上の個別のインターフェイスに対して、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。

7.4. CUMULUS LINUX スイッチの設定

7.4.1. 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することができます。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タ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。

7.4.2. Cumulus Linux スイッチでのトランクポートの設定

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。

手順

- 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。

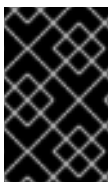
```
auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports glob swp1-2
    bridge-vids 100 200
```

7.4.3. アクセスポートについて

コンピュータノード上の全 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kstレーズ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。

7.4.4. Cumulus Linux スイッチでのアクセスポートの設定

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イ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、管理インターフェイスに **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。

手順

- [図7.1「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。

```
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
```

7.4.5. LACP ポートアグリゲーションについて

リンクアグリゲーション制御プロトコル (LACP) を使用して、複数の物理 NIC をバンドルし、単一の論理チャネルを形成できます。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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.4.6. 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。

関連情報

- [最大伝送単位 \(MTU\) 設定の定義](#)

7.4.7. Cumulus Linux スイッチでの MTU の設定

手順

- 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。

7.4.8. 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。

7.4.9. 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。

手順

- 全ポート/インターフェイスの 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
```

7.5. EXTREME EXOS スイッチの設定

7.5.1. 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タ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。

7.5.2. 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。



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。

手順

- 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 (インターフェイス 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。

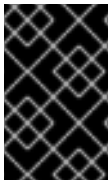
```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

7.5.3. アクセスポートについて

コンピュータノード上の全 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。

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

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 (インターフェイス 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。

手順

- 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。

```
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
```

7.5.5. LACP ポートアグリゲーションについて

リンクアグリゲーション制御プロトコル (LACP) を使用して、複数の物理 NIC をバンドルし、単一の論理チャネルを形成できます。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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.5.6. 物理 NIC 上での LACP の設定

物理 NIC でリンクアグリゲーション制御プロトコル (LACP) を設定できます。

手順

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"
```

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

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

手順

- 以下の例では、コンピュータノードに 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。

7.5.8. 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。

関連情報

- [最大伝送単位 \(MTU\) 設定の定義](#)

7.5.9. Extreme Networks EXOS スイッチでの MTU の設定**手順**

- 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
```

例

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

7.5.10. 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。

7.5.11. Extreme Networks EXOS スイッチでの LLDP の設定**手順**

- 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
```

7.6. JUNIPER EX シリーズスイッチの設定

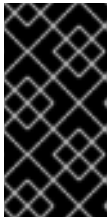
7.6.1. トランクポートについて

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することができます。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タ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。

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

手順

- 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 (インターフェイス 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;
    }
  }
}
```

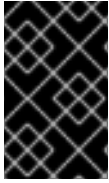
7.6.3. アクセスポートについて

コンピュータノード上の全 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。

7.6.4. 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。

+



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。

手順

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 (インターフェイス **ge-1/0/13**) に接続されていることを前提としています。

+

```
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;
    }
  }
}
```

7.6.5. LACP ポートアグリゲーションについて

リンクアグリゲーション制御プロトコル (LACP) を使用して、複数の物理 NIC をバンドルし、単一の論理チャネルを形成できます。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。

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.6.6. 物理 NIC 上での LACP の設定

物理 NIC でリンクアグリゲーション制御プロトコル (LACP) を設定できます。

手順

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"
```

関連情報

Advanced Overcloud Customization ガイドの [Network Interface Bonding](#) を参照してください。

7.6.7. Juniper EX シリーズスイッチでの LACP の設定

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

手順

1. コンピュータノードの 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。

7.6.8. 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。

関連情報

- [最大伝送単位 \(MTU\) 設定の定義](#)

7.6.9. 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。

手順

1. Juniper EX シリーズスイッチの場合は、インターフェイスごとに 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
```

7.6.10. 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。

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

LLDP は、全インターフェイスでグローバルに有効にすることや、特定のインターフェイスでのみ有効にすることができます。

手順

- 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イス **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。

第8章 最大伝送単位 (MTU) 設定の定義

8.1. MTU の概要

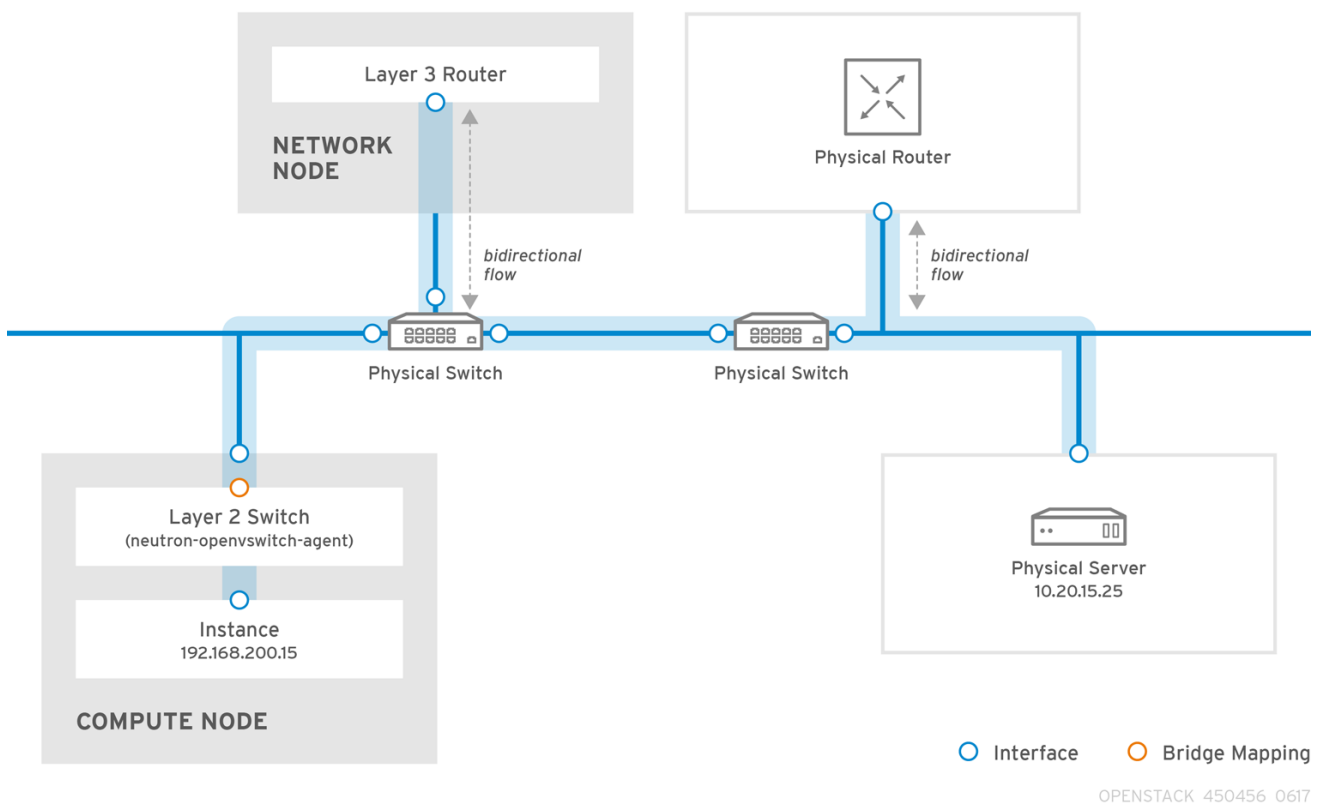
OpenStack Networking は、インスタンスに安全に適用できる最大伝送単位 (MTU) サイズの最大値を計算することができます。MTU の値は、単一のネットワークパケットで転送できる最大データ量を指定します。この数は、アプリケーションに最も適したサイズによって変わります。たとえば、NFS 共有で必要な MTU サイズは VoIP アプリケーションに必要なサイズとは異なる場合があります。

注記

openstack network show <network_name> 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ります。

8.2. 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)、インターフェイス、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}
```

8.3. 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イスを設定します。

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

第9章 QUALITY OF SERVICE (QOS) ポリシーを使用したデータトラフィックの管理

サービスの品質 (QoS) ポリシーを使用して、Red Hat OpenStack Platform (RHOSP) ネットワークの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、VM インスタンスにさまざまなサービスレベルを提供できます。

個々のポートに QoS ポリシーを適用することも、特定のポリシーがアタッチされていないポートがポリシーを継承するプロジェクトネットワークに QoS ポリシーを適用することもできます。



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。

9.1. QOS ルール

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) でサービス品質 (QoS) ポリシーを定義するために、以下のルールタイプを設定することができます。

最小帯域幅 (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。

帯域幅の制限 (bandwidth_limit)

ネットワーク、ポート、フローティング IP、およびルーターゲートウェイ IP の帯域幅制限を提供します。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。

DSCP マーキング (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。

QoS ポリシーは、仮想マシンインスタンスの配置、Floating IP の割り当て、ゲートウェイ IP の割り当てなど、さまざまなコンテキストで実施することができます。

実施する際のコンテキストと使用するメカニズムドライバーに応じて、QoS ルールは、送信トラフィック (インスタンスからのアップロード)、受信トラフィック (インスタンスへのダウンロード)、またはその両方に影響します。

表9.1 ドライバーでサポートされているトラフィックの方向 (すべての QoS ルールタイプ)

ルール	メカニズムドライバーでサポートされているトラフィックの方向		
	ML2/OVS	ML2/SR-IOV	ML2/OVN
最小帯域幅	送信のみ [4][5]	送信のみ	現在、サポートなし [6]

帯域幅の制限	送信 [1][2] および受信	送信のみ [3]	送信および受信
DSCP マーキング	送信のみ	該当なし	送信のみ [7]

[1] OVS の送信帯域幅制限は TAP インターフェイスで行われ、トラフィックポリシングであり、トラフィックシェーピングではありません。

[2] RHOSP 16.2.2 以降では、**ip link** コマンドを使用してネットワークインターフェイスに QoS ポリシーを適用することにより、ハードウェアオフロードポートで OVS の送信帯域幅制限に対応します。

[3] メカニズムドライバーは **max-burst-kbits** パラメーターをサポートしていないため、これを無視します。

[4] トンネル化されていないネットワーク (フラットと VLAN) にのみ適用されるルールです。

[5] **ip link** コマンドを使用してネットワークインターフェイスに QoS ポリシーを適用することにより、ハードウェアオフロードポートで OVS の送信最小帯域幅制限に対応します。

[6] https://bugzilla.redhat.com/show_bug.cgi?id=2060310

7 ML2/OVN は、トンネリングされたプロトコルでの DSCP マーキングをサポートしていません。

表9.2 配置報告やスケジューリング用にドライバーでサポートされているトラフィックの方向 (最小帯域幅のみ)

適用タイプ	方向メカニズムドライバーでサポートされているトラフィック		
	ML2/OVS	ML2/SR-IOV	ML2/OVN
Placement	送信および受信	送信および受信	現在、対応なし

表9.3 適用タイプのドライバーでサポートされているトラフィックの方向 (帯域幅制限のみ)

適用タイプ	メカニズムドライバーでサポートされているトラフィックの方向	
	ML2/OVS	ML2/OVN
Floating IP	送信および受信	送信および受信
ゲートウェイ IP	送信および受信	現在、サポートなし [1]

[1] https://bugzilla.redhat.com/show_bug.cgi?id=2064185

関連情報

- [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](#)

9.2. QOS ポリシーのネットワークサービスの設定

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) のサービス品質機能は、**qos** サービスプラグインを通じて提供されます。ML2/OVS および ML2/OVN メカニズムドライバーでは、デフォルトで **qos** が読み込まれます。ただし、これは ML2/SR-IOV には当てはまりません。

ML2/OVS および ML2/SR-IOV メカニズムドライバーで **qos** サービスプラグインを使用する場合は、それぞれのエージェントに **qos** 拡張機能もロードする必要があります。

次のリストは、ネットワークサービスを QoS 用に設定するために実行する必要があるタスクをまとめたものです。タスクの詳細は、次のリストの後に続きます。

- すべてのタイプの QoS ポリシーの場合:
 - **qos** サービスプラグインを追加します。
 - エージェントの **qos** 拡張機能を追加します (OVS および SR-IOV のみ)。
- 最小帯域幅ポリシーのみを使用して VM インスタンスをスケジュールするための追加タスク:
 - Compute サービス (nova) が使用する名前と異なる場合は、ハイパーバイザー名を指定します。
 - 各コンピュータードで、関連するエージェントのリソースプロバイダーの入力帯域幅と出力帯域幅を設定します。
 - (オプション) **vnic_types** をサポート対象外としてマークします。
- トンネリングのみで ML/OVS を使用するシステムでの DSCP マーキングポリシーの追加タスク:
 - **dscp_inherit** を **true** に設定します。

前提条件

- RHOSP アンダークラウドにアクセスできる **stack** ユーザーである必要がある。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. `source` コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. **qos** サービスプラグインがまだロードされていないことを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされていない場合、**ResourceNotFound** エラーが発生します。エラーが発生しない場合は、プラグインがロードされており、このトピックの手順を実行する必要はありません。

4. YAML カスタム環境ファイルを作成します。

例

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

5. 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** の下の新しい行で、**NeutronServicePlugins** パラメーターに **qos** を追加します。

```
parameter_defaults:
  NeutronServicePlugins: "qos"
```

6. ML2/OVS および ML2/SR-IOV メカニズムドライバーを使用する場合は、それぞれ **NeutronAgentExtensions** または **NeutronSriovAgentExtensions** 変数を使用して、エージェントに **qos** 拡張機能もロードする必要があります。

- ML2/OVS

```
parameter_defaults:
  NeutronServicePlugins: "qos"
  NeutronAgentExtensions: "qos"
```

- ML2/SR-IOV

```
parameter_defaults:
  NeutronServicePlugins: "qos"
  NeutronSriovAgentExtensions: "qos"
```

7. 最小帯域幅 QoS ポリシーを使用して VM インスタンスをスケジュールする場合は、次のことも行う必要があります。

- a. プラグインのリストに **placement** を追加し、リストに **qos** も含まれていることを確認します。

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
```

- b. ハイパーバイザー名が Compute サービス (nova) で使用される標準的なハイパーバイザー名と一致する場合は、ステップ 7.iii に進みます。
ハイパーバイザー名が Compute サービスで使用される標準的なハイパーバイザー名と一致しない場合は、**resource_provider_default_hypervisor** を使用して代替のハイパーバイザー名を指定します。

- ML2/OVS

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
    Neutron::agents::ml2::ovs::resource_provider_default_hypervisor: %
    {hiera("fqdn_canonical")}
```

- ML2/SR-IOV

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
```

```
Neutron::agents::ml2::sriov::resource_provider_default_hypervisor: %
{hiera('fqdn_canonical')}
```

重要

代替ハイパーバイザー名を設定する別の方法は、**resource_provider_hypervisor** を使用することです。

◦ ML2/OVS

```
parameter_defaults:
  ExtraConfig:
```

```
Neutron::agents::ml2::ovs::resource_provider_hypervisors:"ens5:%
{hiera('fqdn_canonical')},ens6:%{hiera('fqdn_canonical')}"
```

◦ ML2/SR-IOV

```
parameter_defaults:
  ExtraConfig:
```

```
Neutron::agents::ml2::sriov::resource_provider_hypervisors:
  "ens5:%{hiera('fqdn_canonical')},ens6:%
  {hiera('fqdn_canonical')}"
```

- c. 最小帯域幅を提供する必要がある各コンピュータ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。
次の形式を使用して、送信、受信、またはその両方を設定できます。

- 送信帯域幅だけを設定する場合 (kbps 単位):

```
NeutronOvsResourceProviderBandwidths: <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):

```
NeutronOvsResourceProviderBandwidths: <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):

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<egress_kbps>:
<ingress_kbps>,<bridge1>:<egress_kbps>:<ingress_kbps>,...,<bridgeN>:
<egress_kbps>:<ingress_kbps>
```

例 - OVS エージェント

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、ネットワーク環境ファイルに以下の設定を追加します。

```
parameter_defaults:
  ...
  NeutronBridgeMappings: physnet0:br-physnet0
  NeutronOvsResourceProviderBandwidths: br-physnet0:10000000:10000000
```

例 - SRIOV エージェント

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、ネットワーク環境ファイルに以下の設定を追加します。

```
parameter_defaults:
...
NeutronML2PhysicalNetworkMtus: physnet0:1500,physnet1:1500
NeutronSriovResourceProviderBandwidths:
ens5:40000000:40000000,ens6:40000000:40000000
```

- d. オプション: **vnic_types** をサポート対象外として識別するには (複数の ML2 メカニズムドライバがデフォルトでサポートし、複数のエージェントが Placement サービスで追跡されている場合)、環境ファイルに以下の設定を追加します。

例 - OVS エージェント

```
parameter_defaults:
...
NeutronOvsVnicTypeBlacklist: direct
```

例 - SRIOV エージェント

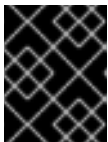
```
parameter_defaults:
...
NeutronSriovVnicTypeBlacklist: direct
```

8. DSCP マーキングポリシーを作成し、トンネリングプロトコル (VXLAN または GRE) で ML2/OVS を使用する場合は、**NeutronAgentExtensions** の下に次の行を追加します。

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

dscp_inherit が **true** の場合、Networking サービスは内部ヘッダーの DSCP 値を外部ヘッダーにコピーします。

9. コア heat テンプレート、その他の環境ファイル、およびこの新しいカスタム環境ファイルを指定して、deployment コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

```
$ openstack overcloud deploy --templates \
-e <other_environment_files> \
-e /home/stack/templates/my-neutron-environment.yaml
```

検証

- **qos** サービスプラグインがロードされていることを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされている場合、**ResourceNotFound** エラーは発生しません。

関連情報

- [RHOSP Networking サービス用のメカニズムドライバードライバー](#)
- [Director インストールと使用方法 ガイドの 環境ファイル](#)
- [Director インストールと使用方法 ガイドの オーバークラウドの作成時の環境ファイルの追加](#)
- [「Networking サービスのバックエンドの実行を使用した最小帯域幅の適用」](#)
- [「最小帯域幅の QoS ポリシーを使用したインスタンスのスケジューリング」](#)
- [「QoS ポリシーを使用したネットワークトラフィックの制限」](#)
- [「DSCP マーキング QoS ポリシーを使用したネットワークトラフィックの優先順位付け」](#)

9.3. QOS ポリシーを使用した最小帯域幅の制御

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) の場合、ネットワークサービスのバックエンドの適用とリソース割り当てのスケジューリングの2つの異なるコンテキストで保証された最小帯域幅の QoS ルールを適用できます。

ネットワークバックエンド、ML2/OVS または ML2/SR-IOV は、ルールが適用される各ポートが指定されたネットワーク帯域幅以上であることを保証しようとします。

リソース割り当てスケジューリング帯域幅強制を使用すると、Compute サービス (nova) は最小帯域幅をサポートするホストにのみ VM インスタンスを配置します。

ネットワークサービスバックエンドの適用、リソース割り当てスケジューリングの適用、またはその両方を使用して、QoS 最小帯域幅ルールを適用できます。

次の表は、最小帯域幅の QoS ポリシーをサポートする Modular Layer2(ML2) メカニズムドライバを示しています。

表9.4 最小帯域幅の QoS をサポートする ML2 メカニズムドライバ

ML2 メカニズムドライバ	エージェント	VNIC タイプ
ML2/SR-IOV	sriovnicswitch	直接的な
ML2/OVS	openvswitch	normal

関連情報

- [「Networking サービスのバックエンドの実行を使用した最小帯域幅の適用」](#)

- [「最小帯域幅の QoS ポリシーを使用したインスタンスのスケジューリング」](#)

9.3.1. Networking サービスのバックエンドの実行を使用した最小帯域幅の適用

Red Hat OpenStack Platform (RHOSP) のサービス品質 (QoS) ポリシーをポートに適用することで、ポートのネットワークトラフィックの最小帯域幅を保証できます。これらのポートは、フラットまたは VLAN 物理ネットワークによってサポートされている必要があります。



注記

現在、Open Virtual Network メカニズムドライバー (ML2/OVN) を備えた Modular Layer 2 プラグインは、最小帯域幅の QoS ルールをサポートしていません。

前提条件

- RHOSP Networking サービス (neutron) には、**qos** サービスプラグインがロードされている必要があります。(これがデフォルトです)。
- 同じ物理インターフェイス上で、帯域幅が保証されているポートと保証されていないポートを混在させないでください。これにより、保証のないポートで必要なリソースが拒否される (枯渇する) 可能性があります。

ヒント

ホストアグリゲートを作成して、帯域幅が保証されているポートと帯域幅が保証されていないポートを分離します。

手順

1. Source コマンドで認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

2. **qos** サービスプラグインが Networking サービスにロードされていることを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされていない場合は、**ResourceNotFound** エラーが発生します。続行するには **qos** サービスプラグインをロードする必要があります。詳細は、[「QoS ポリシーのネットワークサービスの設定」](#) を参照してください。

3. QoS ポリシーを作成するプロジェクトの ID を特定します。

```
$ openstack project list
```

出力例

```
+-----+-----+
| ID                | Name  |
+-----+-----+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
```

```
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+-----+-----+
```

4. 前の手順のプロジェクト ID を使用して、プロジェクトの QoS ポリシーを作成します。

例

この例では、**guaranteed_min_bw** という名前の QoS ポリシーが **admin** プロジェクト用に作成されています。

```
$ openstack network qos policy create --share \
--project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. ポリシーのルールを設定します。

例

この例では、**guaranteed_min_bw** という名前のポリシーに対して、最小帯域幅が **40000000** kbps の入力および出力の QoS ルールが作成されます。

```
$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--ingress guaranteed_min_bw

$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--egress guaranteed_min_bw
```

6. ポリシーを適用するポートを設定します。

例

この例では、**guaranteed_min_bw** ポリシーがポート **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12** に適用されます。

```
$ openstack port set --qos-policy guaranteed_min_bw \
56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

検証

- **ML2/SR-IOV**
ルートアクセスを使用して、コンピュータノードにログインし、物理関数で保持されている仮想関数の詳細を表示します。

例

```
# ip -details link show enp4s0f1
```

出力例

```
50: enp4s0f1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 9000 qdisc mq
master mx-bond state UP mode DEFAULT group default qlen 1000
```

```

link/ether 98:03:9b:9d:73:74 brd ff:ff:ff:ff:ff:ff permaddr 98:03:9b:9d:73:75 promiscuity 0
minmtu 68 maxmtu 9978
bond_slave state BACKUP mii_status UP link_failure_count 0 perm_hwaddr
98:03:9b:9d:73:75 queue_id 0 addrngenmode eui64 numtxqueues 320 numrxqueues 40
gso_max_size 65536 gso_max_segs 65535 portname p1 switchid 74739d00039b0398
vf 0 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 1 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 2 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 3 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 4 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 5 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 6 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 7 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
vf 8 link/ether fa:16:3e:2a:d2:7f brd ff:ff:ff:ff:ff:ff, tx rate 999 (Mbps), max_tx_rate
999Mbps, spoof checking off, link-state disable, trust off, query_rss off
vf 9 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off

```

- **ML2/OVS**

ルートアクセスを使用して、計算ノードにログインし、物理ブリッジインターフェイスに **tc** ルールとクラスを表示します。

例

```
# tc class show dev mx-bond
```

出力例

```

class htb 1:11 parent 1:ffff prio 0 rate 4Gbit ceil 34359Mbit burst 9000b cburst 8589b
class htb 1:1 parent 1:ffff prio 0 rate 72Kbit ceil 34359Mbit burst 9063b cburst 8589b
class htb 1:ffff root rate 34359Mbit ceil 34359Mbit burst 8589b cburst 8589b

```

関連情報

- コマンドラインインターフェイスリファレンスの [network qos policy create](#)
- コマンドラインインターフェイスリファレンスの [network qos rule create](#)
- コマンドラインインターフェイスリファレンスの [port set](#)

9.3.2. 最小帯域幅の QoS ポリシーを使用したインスタンスのスケジューリング

最小帯域幅の QoS ポリシーをポートに適用して、Red Hat OpenStack Platform (RHOSP) VM インスタンスが生成されるホストに最小のネットワーク帯域幅があることを保証できます。

前提条件

- RHOSP Networking サービス (neutron) には、**qos** および **placement** サービスのプラグインがロードされている必要があります。**qos** サービスプラグインはデフォルトでロードされます。
- ネットワークサービスは、次の 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**
- ML2/OVS または ML2/SR-IOV メカニズムドライバーを使用する必要がある。
- 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ルがない場合に限る。ポートがバインドされている場合、ネットワークサービスは配置 API の使用情報を更新できない。
- 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 サービス (nova) 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。

手順

1. Source コマンドで認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

2. **qos** サービスプラグインが Networking サービスにロードされていることを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされていない場合は、**ResourceNotFound** エラーが発生します。続行するには **qos** サービスプラグインをロードする必要があります。詳細は、[「QoS ポリシーのネットワークサービスの設定」](#) を参照してください。

3. QoS ポリシーを作成するプロジェクトの ID を特定します。

```
$ openstack project list
```

出力例

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

4. 前の手順のプロジェクト ID を使用して、プロジェクトの QoS ポリシーを作成します。

例

この例では、**guaranteed_min_bw** という名前の QoS ポリシーが **admin** プロジェクト用に作成されています。

```
$ openstack network qos policy create --share \
  --project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. ポリシーのルールを設定します。

例

この例では、**guaranteed_min_bw** という名前のポリシーに対して、最小帯域幅が **40000000** kbps の入力および出力の QoS ルールが作成されます。

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

6. ポリシーを適用するポートを設定します。

例

この例では、**guaranteed_min_bw** ポリシーがポート **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12** に適用されます。

```
$ openstack port set --qos-policy guaranteed_min_bw \
  56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

検証

1. アンダークラウドホストに stack ユーザーとしてログインします。
2. source コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. 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。

```
$ 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 |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
```

4. 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>
```

例

この例では、リソースプロバイダー UUID、**e518d381-d590-5767-8f34-c20def34b252** を使用して、ホスト **dell-r730-014** のインターフェイス **enp6s0f1** によって提供される帯域幅がチェックされます。

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 \
resource provider inventory list e518d381-d590-5767-8f34-c20def34b252
```

出力例

```
+-----+-----+-----+-----+-----+-----+
| 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 |
+-----+-----+-----+-----+-----+-----+
```

5. 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>
```

例

この例では、リソースプロバイダーに対するクレームは、リソースプロバイダー UUID **e518d381-d590-5767-8f34-c20def34b252** を使用して、ホスト **dell-r730-014** でチェックされます。

```
[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
```

出力例

```
{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}}}
```

関連情報

- コマンドラインインターフェイスリファレンスの [network qos policy create](#)
- コマンドラインインターフェイスリファレンスの [network qos rule create](#)
- コマンドラインインターフェイスリファレンスの [port set](#)

9.4. QOS ポリシーを使用したネットワークトラフィックの制限

Red Hat OpenStack Platform (RHOSP) ネットワークサービス (neutron) のサービス品質 (QoS) ポリシーを作成して、RHOSP ネットワーク、ポート、または Floating IP の帯域幅を制限し、指定のレートを超えるトラフィックをドロップできます。

前提条件

- Networking サービスには **qos** サービスプラグインがロードされている必要があります (プラグインはデフォルトでロードされます)。

手順

1. Source コマンドで認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

2. **qos** サービスプラグインが Networking サービスにロードされていることを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされていない場合は、**ResourceNotFound** エラーが発生します。続行するには **qos** サービスプラグインをロードする必要があります。詳細は、「[QoS ポリシーのネットワークサービスの設定](#)」を参照してください。

3. QoS ポリシーを作成するプロジェクトの ID を特定します。

```
$ openstack project list
```

出力例

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

4. 前の手順のプロジェクト ID を使用して、プロジェクトの QoS ポリシーを作成します。

例

この例では、**admin** プロジェクト用に **bw-limiter** という名前の QoS ポリシーが作成されます。

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

5. ポリシーのルールを設定します。



注記

各ルールのタイプまたは方向が異なる限り、複数のルールをポリシーに追加できます。たとえば、1つを送信方向、もう1つを受信方向に設定して2つの bandwidth-limit ルールを指定できます。

例

この例では、**50000** kbps の帯域幅制限と **50000** kbps の最大バーストサイズで、**bw-limiter** という名前のポリシーに対して QoS Ingress ルールと egress ルールが作成されます。

```
$ openstack network qos rule create --type bandwidth-limit \
--max-kbps 50000 --max-burst-kbits 50000 --ingress bw-limiter

$ openstack network qos rule create --type bandwidth-limit \
--max-kbps 50000 --max-burst-kbits 50000 --egress bw-limiter
```

6. ポリシーがアタッチされたポートを作成するか、既存のポートにポリシーをアタッチできます。

例 - ポリシーがアタッチされたポートの作成

この例では、ポリシー **bw-limiter** がポート **port2** に関連付けられています。

```
$ openstack port create --qos-policy bw-limiter --network private port2
```

出力例

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	2022-07-04T19:20:24Z
data_plane_status	None
description	
device_id	
device_owner	
dns_assignment	None
dns_name	None
extra_dhcp_opts	
fixed_ips	ip_address='192.0.2.210', subnet_id='292f8c-...'
id	f51562ee-da8d-42de-9578-f6f5cb248226
ip_address	None
mac_address	fa:16:3e:d9:f2:ba
name	port2
network_id	55dc2f70-0f92-4002-b343-ca34277b0234
option_name	None
option_value	None
port_security_enabled	False
project_id	98a2f53c20ce4d50a40dac4a38016c69
qos_policy_id	8491547e-add1-4c6c-a50e-42121237256c
revision_number	6
security_group_ids	0531cc1a-19d1-4cc7-ada5-49f8b08245be
status	DOWN
subnet_id	None
tags	[]
trunk_details	None
updated_at	2022-07-04T19:23:00Z

例 - ポリシーの既存のポートへのアタッチ

この例では、ポリシー **bw-limiter** が **port1** に関連付けられています。

```
$ openstack port set --qos-policy bw-limiter port1
```

検証

- ポートに帯域制限ポリシーが適用されていることを確認します。
 - ポリシー ID を取得します。

例

この例では、QoS ポリシー **bw-limiter** が照会されます。

```
$ openstack network qos policy show bw-limiter
```

出力例

```

+-----+-----+
| Field          | Value                                                                 |
+-----+-----+
| description     |                                                                       |
| id              | 8491547e-add1-4c6c-a50e-42121237256c                                |
| is_default      | False                                                                |
| name            | bw-limiter                                                           |
| project_id      | 98a2f53c20ce4d50a40dac4a38016c69                                    |
| revision_number | 4                                                                    |
| rules           | [{u'max_kbps': 50000, u'direction': u'egress',                      |
|                   | u'type': u'bandwidth_limit',                                         |
|                   | u'id': u'0db48906-a762-4d32-8694-3f65214c34a6',                     |
|                   | u'max_burst_kbps': 50000,                                             |
|                   | u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'},        |
|                   | {u'max_kbps': 50000, u'direction': u'ingress',                      |
|                   | u'type': u'bandwidth_limit',                                         |
|                   | u'id': u'faabef24-e23a-4fdf-8e92-f8cb66998834',                     |
|                   | u'max_burst_kbps': 50000,                                             |
|                   | u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'}]        |
| shared          | False                                                                |
+-----+-----+

```

- ポートを照会し、そのポリシー ID が前の手順で取得したものと一致することを確認します。

例

この例では、**port1** が照会されます。

```
$ openstack port show port1
```

出力例

```

+-----+-----+
| Field          | Value                                                                 |
+-----+-----+
| admin_state_up  | UP                                                                    |
| allowed_address_pairs | ip_address='192.0.2.128', mac_address='fa:16:3e:e1:eb:73'          |
| binding_host_id  | compute-2.redhat.local                                              |
| binding_profile  |                                                                       |
| binding_vif_details | port_filter='True'                                                  |
| binding_vif_type  | ovs                                                                  |
| binding_vnic_type | normal                                                                |
| created_at       | 2022-07-04T19:07:56                                                 |
| data_plane_status | None                                                                  |
| description      |                                                                       |
| device_id        | 53abd2c4-955d-4b44-b6ad-f106e3f15df0                                |
| device_owner     | compute:nova                                                         |
| dns_assignment   | fqdn='host-192-0-2-213.openstacklocal.', hostname='my-host3',      |
|                   | ip_address='192.0.2.213'                                             |
| dns_domain       | None                                                                  |
+-----+-----+

```

```

| dns_name          |          |
| extra_dhcp_opts   |          |
| fixed_ips         | ip_address='192.0.2..213', subnet_id='641d1db2-3b40-437b-b87b-63 |
|                   | 079a7063ca' |
|                   | ip_address='2001:db8:0:f868:f816:3eff:fee1:eb73', subnet_id='c7ed0 |
|                   | 70a-d2ee-4380-baab-6978932a7dcc' |
| id                | 56x9aiw1-2v74-144x-c2q8-ed8w423a6s12 |
| location          | cloud=", project.domain_id=, project.domain_name=, project.id='7c |
|                   | b99d752fdb4944a2208ec9ee019226', project.name=, |
| region_name       | 'regio |
|                   | nOne', zone= |
| mac_address       | fa:16:3e:e1:eb:73 |
| name              | port2 |
| network_id        | 55dc2f70-0f92-4002-b343-ca34277b0234 |
| port_security_enabled | True |
| project_id        | 98a2f53c20ce4d50a40dac4a38016c69 |
| propagate_uplink_status | None |
| qos_policy_id     | 8491547e-add1-4c6c-a50e-42121237256c |
| resource_request  | None |
| revision_number   | 6 |
| security_group_ids | 4cdeb836-b5fd-441e-bd01-498d758704fd |
| status            | ACTIVE |
| tags              | |
| trunk_details     | None |
| updated_at        | 2022-07-04T19:11:41Z |
+-----+-----+

```

関連情報

- コマンドラインインターフェイスリファレンス の [network qos rule create](#)
- コマンドラインインターフェイスリファレンス の [network qos rule set](#)
- コマンドラインインターフェイスリファレンス の [network qos rule delete](#)
- コマンドラインインターフェイスリファレンス の [network qos rule list](#)

9.5. DSCP マーキング QOS ポリシーを使用したネットワークトラフィックの優先順位付け

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、Red Hat OpenStack Platform (RHOSP) 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。RHOSP Networking service (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だけを管理することができます。

前提条件

- Networking サービスには、**qos** サービスプラグインがロードされている必要があります。(これがデフォルトです)。
- ML2/OVS または ML2/OVN メカニズムドライバーを使用する必要があります。

手順

1. Source コマンドで認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

2. **qos** サービスプラグインが Networking サービスにロードされていることを確認します。

```
$ openstack network qos policy list
```

qos サービスプラグインがロードされていない場合は、**ResourceNotFound** エラーが発生します。続行する前に Networking サービスを設定する必要があります。詳細は、「[QoS ポリシーのネットワークサービスの設定](#)」を参照してください。

3. QoS ポリシーを作成するプロジェクトの ID を特定します。

```
$ openstack project list
```

出力例

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

4. 前の手順のプロジェクト ID を使用して、プロジェクトの QoS ポリシーを作成します。

例

この例では、**admin** プロジェクトに **qos-web-servers** という名前の QoS ポリシーが作成されます。

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

5. DSCP ルールを作成し、それをポリシーに適用します。

例

この例では、DSCP ルールは DSCP マーク **18** を使用して作成され、**qos-web-servers** ポリシーに適用されます。

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

出力例

```
Created a new dscp_marking_rule:
+-----+-----+
| Field  | Value                |
+-----+-----+
```

```

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

```

6. ルールに割り当てられている DSCP 値を変更できます。

例

この例では、**qos-web-servers** ポリシーのルール **d7f976ec-7fab-4e60-af70-f59bf88198e6** の DSCP マーク値が 22 に変更されます。

```

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

```

7. DSCP ルールを削除できます。

例

この例では、**qos-web-servers** ポリシーの DSCP ルール **d7f976ec-7fab-4e60-af70-f59bf88198e6** が削除されます。

```

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

```

検証

- QoS ポリシーに DSCP ルールが適用されていることを確認します。

例

DSCP ルール (**d7f976ec-7fab-4e60-af70-f59bf88198e6**) が QoS ポリシー (**qos-web-servers**) に適用されていることを確認します。

```

$ openstack network qos rule list qos-web-servers

```

出力例

```

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

```

関連情報

- コマンドラインインターフェイスリファレンス の [network qos rule create](#)
- コマンドラインインターフェイスリファレンス の [network qos rule set](#)
- コマンドラインインターフェイスリファレンス の [network qos rule delete](#)
- コマンドラインインターフェイスリファレンス の [network qos rule list](#)

9.6. ネットワークサービス RBAC を使用したプロジェクトへの QoS ポリシーの適用

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) を使用すると、サービス品質 (QoS) ポリシー用のロールベースのアクセス制御 (RBAC) を追加できます。その結果、個々のプロジェクトに QoS ポリシーを適用できます。

前提条件

- 1つ以上の QoS ポリシーを使用できる。

手順

- 特定の QoS ポリシーに関連付けられた RHOSP Networking サービス RBAC ポリシーを作成し、特定のプロジェクトに割り当てます。

```
$ openstack network rbac create --type qos_policy --target-project <project_name | project_ID> --action access_as_shared <QoS_policy_name | QoS_policy_ID>
```

例

たとえば、**bw-limiter** という名前の優先度の低いネットワークトラフィックを許可する QoS ポリシーがあるとします。RHOSP Networking サービスの RBAC ポリシーを使用して、特定のプロジェクトに QoS ポリシーを適用できます。

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

関連情報

- コマンドラインインターフェイスリファレンス の [network rbac create](#)
- 「Networking サービスのバックエンドの実行を使用した最小帯域幅の適用」
- 「最小帯域幅の QoS ポリシーを使用したインスタンスのスケジューリング」
- 「QoS ポリシーを使用したネットワークトラフィックの制限」
- 「DSCP マーキング QoS ポリシーを使用したネットワークトラフィックの優先順位付け」

第10章 ブリッジマッピングの設定

Red Hat OpenStack Platform (RHOSP) では、ブリッジマッピングは物理ネットワーク名 (インターフェイスラベル) を Modular Layer 2 プラグインメカニズムドライバーの Open vSwitch (OVS) または Open Virtual Network (OVN) で作成されたブリッジに関連付けます。RHOSP Networking サービス (neutron) は、ブリッジマッピングを使用して、プロバイダーのネットワークトラフィックが物理ネットワークに到達できるようにします。

このセクションに含まれるトピックは次のとおりです。

- [「ブリッジマッピングの概要」](#)
- [「トラフィックの流れ」](#)
- [「ブリッジマッピングの設定」](#)
- [「OVS ブリッジマッピングのメンテナンス」](#)
- [「OVS パッチポートの手動クリーンアップ」](#)
- [「OVS パッチポートの自動クリーンアップ」](#)

10.1. ブリッジマッピングの概要

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) では、ブリッジマッピングを使用して、プロバイダーのネットワークトラフィックが物理ネットワークに到達できるようにします。トラフィックはルーターの **qg-xxx** インターフェイスからプロバイダーネットワークを離れ、中間ブリッジ (**br-int**) に到着します。

データパスの次の部分は、展開で使用するメカニズムドライバーによって異なります。

- ML2/OVS: **br-int** と **br-ex** の間のパッチポートにより、トラフィックはプロバイダーネットワークのブリッジを通過し、物理ネットワークに出ます。
- ML2/OVN: ネットワーキングサービスは、ハイパーバイザーにバインドされた VM があり、VM がポートを必要とする場合にのみ、ハイパーバイザーにパッチポートを作成します。

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。



注記

Networking サービスは、各物理ネットワークに対して1つのブリッジのみをサポートします。同じブリッジに複数の物理ネットワークをマッピングしないでください。

10.2. 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イス、その後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** に移動するとき、パケットの外部 VLAN ID は **br-int** の内部 VLAN タグに置き換えられ、これにより **qg-xxx** はパケットを受け入れることができます。

出力パケットの場合、パケットの内部 VLAN タグは、**br-ex** (または **NeutronNetworkVLANRanges** パラメーターで定義された外部ブリッジ) で外部 VLAN タグに置き換えられます。

10.3. ブリッジマッピングの設定

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) がプロバイダーのネットワークトラフィックを物理ネットワークに接続するために使用するブリッジマッピングを変更するには、必要な heat パラメーターを変更して、オーバークラウドを再デプロイします。

前提条件

- の下にあるホストに **stack** ユーザーとしてアクセスできる必要があります。
- 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する必要があります。
- また、コンピュートノードのブリッジマッピングを設定する必要もあります。

手順

1. アンダークラウドホストに stack ユーザーとしてログインします。
2. source コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. カスタム YAML 環境ファイルを作成します。

例

```
$ vi /home/stack/templates/my_bridge_mappings.yaml
```

4. 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** キーワードの後に、サイトに適した値を持つ **NeutronBridgeMappings** heat パラメーターを追加します。

例

この例では、**NeutronBridgeMappings** パラメーターは、物理名 **datacentre** と **tenant**、ブリッジ **br-ex** と **br-tenant** をそれぞれ関連付けます。

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



注記

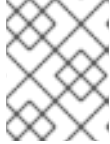
NeutronBridgeMappings パラメーターを使用しないと、デフォルトではホストの外部ブリッジ (br-ex) を物理名 (datacentre) にマッピングします。

5. フラットネットワークを使用している場合は、**NeutronFlatNetworks** パラメーターを使用してその名前を追加します。

例

この例では、パラメーターは物理名 **datacentre** をブリッジ **br-ex** に関連付け、物理名 **tenant** をブリッジ **br-tenant** に関連付けます。

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



注記

NeutronFlatNetworks パラメーターが使用されていない場合、デフォルトは **datacentre** です。

6. VLAN ネットワークを使用している場合は、**NeutronNetworkVLANRanges** パラメーターを使用して、アクセスする VLAN の範囲とともにネットワーク名を指定します。

例

この例では、**NeutronNetworkVLANRanges** パラメーターは、**tenant** ネットワークの VLAN 範囲を **1 - 1000** で指定します。

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
  NeutronNetworkVLANRanges: "tenant:1:1000"
```

7. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、deployment コマンドを実行します。

```
$ openstack overcloud deploy --templates \
  -e <your_environment_files> \
  -e /home/stack/templates/my_bridge_mappings.yaml
```

8. 以下の手順を実行します。
 - 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イスを使用して、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

- [Director のインストールと使用 ガイド](#)で [ネットワーク設定ファイルの形式を更新する](#)
- [Director インストールと使用方法 ガイド](#)の [オーバークラウドの作成時の環境ファイルの追加](#)

10.4. 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。

10.4.1. 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
```

10.4.2. 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。

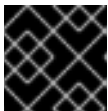
前提条件

- 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](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

第11章 VLAN 対応のインスタンス

11.1. VLAN トランクおよび VLAN 透過ネットワーク

VM インスタンスは、単一の仮想 NIC を使用して、VLAN タグ付けされ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 タグ付けされ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。

ML2/OVN のデプロイメントでは、VLAN 透過ネットワークを使用して VLAN 対応のインスタンスをサポートすることができます。これとは別に、ML2/OVN または ML2/OVS のデプロイメントでは、トランクを使用して VLAN 対応のインスタンスをサポートすることができます。

VLAN 透過ネットワークでは、仮想マシンインスタンスで VLAN タグ付けを設定します。VLAN タグはネットワークを通じて転送され、同じ VLAN のインスタンスにより消費され、他のインスタンスやデバイスでは無視されます。VLAN 透過ネットワークでは、VLAN はインスタンスで管理されます。

OpenStack Networking サービス (neutron) で VLAN を設定する必要はありません。

VLAN トランクは、複数の VLAN を 1 つのトランクポートに結び付けて、VLAN 対応のインスタンスをサポートします。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、または Geneve) 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。ネットワークパケットは、ネットワーク全体でタグ付けされる必要はなく、インスタンスに注入される直前にタグ付けされます。

特定の機能について、VLAN 透過ネットワークと VLAN トランクの比較を以下の表にまとめます。

	透過	トランク
メカニズムドライバーのサポート	ML2/OVN	ML2/OVN、ML2/OVS
VLAN 設定の管理	仮想マシンインスタンスによる	OpenStack Networking サービス (neutron) による
IP 割り当て	仮想マシンインスタンスでの設定	DHCP による割り当て
VLAN ID	柔軟。インスタンスで VLAN ID を設定することが可能。	固定。インスタンスは、トランクで設定した VLAN ID を使用する必要がある。

11.2. ML2/OVN デプロイメントでの VLAN 透過性の有効化

仮想マシン (VM) インスタンス間で VLAN タグ付けされたトラフィックを送信する必要がある場合は、VLAN の透過性を有効にします。VLAN 透過ネットワークでは、neutron で VLAN を設定するのではなく、直接仮想マシンで設定することができます。

前提条件

- メカニズムドライバーとして ML2/OVN を使用する Red Hat OpenStack Platform 16.1 以降のデプロイメント
- 種別 VLAN または Geneve のプロバイダーネットワーク。フラット種別のプロバイダーネットワークのデプロイメントでは、VLAN の透過性を使用しないでください。

- 両方の VLAN において、外部スイッチが ethertype 0x8100 を使用する 802.1q VLAN スタックをサポートするようにします。OVN VLAN の透過性は、0x88A8 または 0x9100 に設定された外部プロバイダー VLAN ethertype を使用する 802.1ad QinQ をサポートしません。

手順

1. アンダークラウドノードの環境ファイルで、**EnableVLANTransparency** パラメーターを **true** に設定します。たとえば、次の行を **ovn-extras.yaml** に追加します。

```
parameter_defaults:
  EnableVLANTransparency: true
```

2. この環境ファイルをご自分の環境に該当するその他の環境ファイルと共に **openstack overcloud deploy** コマンドに追加して、オーバークラウドをデプロイします。

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

-e ovn-extras.yaml \
...
```

<other_overcloud_environment_files> を既存のデプロイメントに含まれる環境ファイルの一覧に置き換えます。

3. **--transparent-vlan** 引数を使用してネットワークを作成します。

例

```
$ openstack network create network-name --transparent-vlan
```

4. 参加するそれぞれの仮想マシンに VLAN インターフェイスを設定します。
VLAN の透過性に必要な追加のタグ付けに対応するために、インターフェイスの MTU を下層のネットワークの MTU より 4 バイト少ない値に設定します。たとえば、下層のネットワークの MTU が 1500 の場合は、インターフェイスの MTU を 1496 に設定します。

次のコマンド例では、MTU が 1496 の VLAN インターフェイスを **eth0** に追加します。VLAN は 50 で、インターフェイス名は **vlan50** です。

例

```
$ ip link add link eth0 name vlan50 type vlan id 50 mtu 1496
$ ip link set vlan50 up
$ ip addr add 192.128.111.3/24 dev vlan50
```

5. 仮想マシンポートに **--allowed-address** を設定します。
許可されたアドレスを、手順 4 で VM 内の VLAN インターフェイス上に作成した IP アドレスに設定します。オプションで、VLAN インターフェイスの MAC アドレスを設定することもできます。

例

次の例では、ポート **fv82gwk3-qq2e-yu93-go31-56w7sf476mm0** で、IP アドレスを **192.128.111.3** に設定し、オプションの MAC アドレス **00:40:96:a8:45:c4** を 設定します。

```
$ openstack port set --allowed-address ip-address=192.128.111.3,mac-
address=00:40:96:a8:45:c4 fv82gwk3-qq2e-yu93-go31-56w7sf476mm0
```

検証

1. vlan50 の IP アドレスを使用して VLAN 上の 2 つの仮想マシン間で ping を送信します。
2. **eth0** で **tcpdump** を使用して、VLAN タグが付いたままパケットが到達するかどうかを確認します。

関連情報

- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

11.3. 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/puppet-generated/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
```

11.4. トランク接続の作成

VLAN タグ付けされたトラフィック用のトランクを実装するには、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イスも作成する必要があります。

1. トランキングされた VLAN へのアクセスを必要とするインスタンスが含まれるネットワークを特定します。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-b93adbdc197 |
| 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 |
+-----+-----+
```

11.5. 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

```

11.6. トランクを使用するためのインスタンスの設定

Red Hat OpenStack Platform (RHOSP) Networking サービス (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、仮想マシン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。

前提条件

- コンピュートノードのライブマイグレーションを実行している場合は、RHOSP Networking サービスの RPC 応答タイムアウトが RHOSP デプロイメントに対して適切に設定されていることを確認します。RPC のレスポンスタイムアウト値はサイトごとに異なり、システムの速度によって異なります。一般的な推奨値は、100 トランクポートごとに 120 秒以上です。ベストプラクティスは、RHOSP デプロイメントのトランクポートバインドプロセスの時間を測定してから、RHOSP Networking サービスの RPC 応答タイムアウトを適切に設定することです。RPC のレスポンスタイムアウト値を低く維持してみてください。ただし、RHOSP Networking サービスが RPC の応答を受け取る時間を十分に取ってください。詳細は、[「Networking サービスの RPC タイムアウトの設定」](#) を参照してください。

手順

1. **network trunk** コマンドを使用して、ネットワークトランクの設定を確認します。

例

```
$ openstack network trunk list
```

出力例

```

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

```

例

```
$ openstack network trunk show parent-trunk
```

出力例

```

+-----+-----+-----+
| Field      | Value                                     |
+-----+-----+-----+
| admin_state_up | UP                                       |
| created_at   | 2021-10-20T02:05:17Z                   |
| description   |                                           |
| id           | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88   |
| name         | parent-trunk                           |
| port_id      | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39   |
| revision_number | 2                                       |
| status       | DOWN                                    |
| sub_ports    | port_id='479d742e-dd00-4c24-8dd6-b7297fab3ee9', segm |
|              | entation_id='55', segmentation_type='vlan'      |
| tenant_id    | 745d33000ac74d30a77539f8920555e7       |
| updated_at   | 2021-08-20T02:10:06Z                   |
+-----+-----+-----+

```

2. 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。

例

```
openstack server create --image cirros --flavor m1.tiny --security-group default --key-name sshaccess --nic port-id=20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 testInstance
```

出力例

```

+-----+-----+-----+
| Property          | Value                                     |
+-----+-----+-----+
| OS-DCF:diskConfig | MANUAL                                   |
| OS-EXT-AZ:availability_zone |                                           |
| OS-EXT-SRV-ATTR:host | -                                       |
| OS-EXT-SRV-ATTR:hostname | testinstance                           |
| OS-EXT-SRV-ATTR:hypervisor_hostname | -                                       |
| OS-EXT-SRV-ATTR:instance_name |                                           |
| OS-EXT-SRV-ATTR:kernel_id |                                           |
| OS-EXT-SRV-ATTR:launch_index | 0                                       |
| OS-EXT-SRV-ATTR:ramdisk_id |                                           |
| OS-EXT-SRV-ATTR:reservation_id | r-juqco0el                             |
| OS-EXT-SRV-ATTR:root_device_name | -                                       |
+-----+-----+-----+

```

```

| OS-EXT-SRV-ATTR:user_data      | - |
| OS-EXT-STS:power_state        | 0 |
| OS-EXT-STS:task_state         | scheduling |
| OS-EXT-STS:vm_state           | building |
| OS-SRV-USG:launched_at        | - |
| OS-SRV-USG:terminated_at      | - |
| accessIPv4                    | |
| accessIPv6                    | |
| adminPass                     | uMyL8PnZRBwQ |
| config_drive                  | |
| created                       | 2021-08-20T03:02:51Z |
| description                   | - |
| flavor                        | m1.tiny (1) |
| hostId                        | |
| host_status                   | |
| id                            | 88b7aede-1305-4d91-a180-67e7eac |
|                               | 8b70d |
| image                         | cirros (568372f7-15df-4e61-a05f |
|                               | -10954f79a3c4) |
| key_name                      | sshaccess |
| locked                        | False |
| metadata                     | {} |
| name                          | testInstance |
| os-extended-volumes:volumes_attached | [] |
| progress                     | 0 |
| security_groups               | default |
| status                        | BUILD |
| tags                          | [] |
| tenant_id                    | 745d33000ac74d30a77539f8920555e |
|                               | 7 |
| updated                      | 2021-08-20T03:02:51Z |
| user_id                      | 8c4aea738d774967b4ef388eb41fef5 |
|                               | e |
+-----+-----+

```

関連情報

- [Networking サービスの RPC タイムアウトの設定](#)

11.7. NETWORKING サービスの RPC タイムアウトの設定

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) の RPC 応答タイムアウトを変更する必要がある場合もあります。たとえば、タイムアウト値が低すぎると、トランクポートを使用するコンピュータノードのライブマイグレーションが失敗する可能性があります。

RPC のレスポンスタイムアウト値はサイトごとに異なり、システムの数によって異なります。一般的な推奨値は、100 トランクポートごとに 120 秒以上です。

お使いのサイトでトランクポートを使用している場合、ベストプラクティスは、RHOSP デプロイメントのトランクポートバインドプロセスの時間を測定してから、RHOSP Networking サービスの RPC 応答タイムアウトを適切に設定することです。RPC のレスポンスタイムアウト値を低く維持してみてください。ただし、RHOSP Networking サービスが RPC の応答を受け取る時間を十分に取ってください。

手動の hieradata オーバーライド **rpc_response_timeout** を使用して、RHOSP Networking サービスの RPC 応答タイムアウト値を設定することができます。

手順

1. アンダークラウドホストに stack ユーザーとしてログインして、カスタム YAML 環境ファイルを作成します。

例

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

ヒント

RHOSP Orchestration サービス (heat) は、**テンプレート**と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル**を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、heat テンプレートをカスタマイズするための特別な種別のテンプレートです。

2. **ExtraConfig** 下の YAML 環境ファイルで、**rpc_response_timeout** に適切な値 (秒単位) を設定します。(デフォルト値は 60 秒です。)

例

```
parameter_defaults:
  ExtraConfig:
    neutron::rpc_response_timeout: 120
```



注記

RHOSP Orchestration サービス (heat) は、カスタムの環境ファイルで設定した値ですべての RHOSP ノードを更新しますが、この値は RHOSP Networking コンポーネントにのみ影響します。

3. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

関連情報

- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

11.8. 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12章 RBAC ポリシーの設定

12.1. 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。

12.2. 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。

12.3. 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 |
+-----+-----+
```

12.4. 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
```

12.5. 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イ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13章 分散仮想ルーター (DVR) の設定

13.1. 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。

新たなデフォルトの RHOSP デプロイメントでは、DVR および Modular Layer 2 プラグインと Open Virtual Network メカニズムドライバの組み合わせ (ML2/OVN) が使用されます。

ML2/OVS のデプロイメントでは、DVR はデフォルトで無効になっています。

13.1.1. レイヤー 3 ルーティングの概要

Red Hat OpenStack Platform Networking サービス (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仮想マシンインスタンスは、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)。

13.1.2. フローのルーティング

Red Hat OpenStack Platform (RHOSP) 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 (インターネット上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):** Networking サービスは、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 接頭辞が割り当てられた Networking サービスのネットワークでは、外部にアクセスするために Networking サービス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。

13.1.3. 集中ルーティング

Networking サービス (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デプロイ、導入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イスはコントローラーノードでのみ利用できるため、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イ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ルリングを改善するには、Networking サービス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。

13.2. 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サービスは、すべての分散ルーター上でホストされます。

13.3. DVR に関する既知の問題および注意

- 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。

- ML2/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。
- ML2/OVS メカニズムドライバーネットワークバックエンドおよび DVR を使用すると、仮想 IP を作成することができます。ただし、**allowed_address_pairs** を使用するバインドポートに割り当てられる IP アドレスは、仮想ポートの IP アドレス (/32) と一致する必要があります。バインドポート **allowed_address_pairs** に CIDR 形式の IP アドレスを使用する場合には、ポート転送はバックエンドで設定されず、バインドされた IP ポートに到達できる必要のある CIDR の IP でトラフィックが失敗します。
- 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。
- ML2/OVS デプロイメントでは、DVR が有効な場合でも IPv6 トラフィックは分散されません。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通過します。ML2/OVS と共に IPv6 ルーティングを広範囲に渡って使用する場合は、DVR を使用しないでください。ML2/OVN デプロイメントでは、すべての East/West トラフィックは常に分散され、North/South トラフィックは DVR が設定されている場合に分散される点に注意してください。
- ML2/OVS デプロイメントでは、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 16.1 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設定でコントローラーノードにデプロイされます。
- コンピュートノードには、外部ブリッジに接続された外部ネットワーク上のインターフェイスが必要です。このインターフェイスを使用して、外部ルーターゲートウェイの VLAN またはフラットネットワークに接続し、Floating IP をホストし、Floating IP を使用する VM の SNAT を実行します。
- ML2/OVS のデプロイメントでは、各コンピュートノードに追加の IP アドレスが1つ必要です。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。

13.4. サポートされているルーティングアーキテクチャー

Red Hat OpenStack Platform (RHOSP) は、以下にリストされている RHOSP バージョンで、集中型の高可用性 (HA) ルーティングと分散仮想ルーター (DVR) の両方をサポートします。

- RHOSP 集中型 HA ルーティングサポートは、RHOSP 8 で開始されました。
- RHOSP 分散ルーティングのサポートは RHOSP12 で開始されました。

13.5. ML2 OVS を使用した DVR のデプロイ

ML2/OVS デプロイメントにおいて分散仮想ルーター (DVR) をデプロイおよび管理するには、heat テンプレートおよび環境ファイルで設定を行います。

heat テンプレート設定を使用して、ホストのネットワーク設定をプロビジョニングします。

- 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イスを、コンピュータードとコントローラーノードの両方で設定する。
- コンピュータ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イスを設定する。

また、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 Networking サービス (neutron) を設定し、トラフィックがブリッジを使用できるようにします。

デフォルト設定はガイドラインとしてのみ提供されます。ネットワークの分離、専用の NIC、またはその他の変動要因のためにカスタマイズが必要となる実稼働環境またはテスト環境で機能することは想定されていません。環境を設定する際には、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を正しく設定する必要があります。

以下の手順の例は、典型的なデフォルト値を使用して概念実証用の環境を設定する方法を示しています。

手順

1. ファイル `overcloud-resource-registry.yaml` またはデプロイメントコマンドに含まれる環境ファイルで、`OS::TripleO::Compute::Net::SoftwareConfig` の値が `OS::TripleO::Controller::Net::SoftwareConfig` の値と一致していることを確認します。
この値で、`net_config_bridge.yaml` 等のファイル名を指定します。指定したファイルで、外部ネットワーク用コンピュータード L2 エージェントの Neutron ブリッジマッピングを設定します。ブリッジは、DVR デプロイメントのコンピュータードがホストする Floating IP アドレスのトラフィックをルーティングします。通常、このファイル名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。



注記

コンピュータードのネットワーク設定をカスタマイズする場合には、代わりにカスタム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. コンピュータードに外部ブリッジが設定されていることを確認します。
 - a. `openstack-tripleo-heat-templates` ディレクトリーのローカルコピーを作成します。
 - b. `$ cd <local_copy_of_templates_directory>`
 - c. `process-templates` スクリプトを実行して、テンプレートを一時的な出力ディレクトリーにレンダリングします。

```
$ ./tools/process-templates.py -r <roles_data.yaml> \
  -n <network_data.yaml> -o <temporary_output_directory>
```

- d. `<temporary_output_directory>/network/config` でロールファイルを確認します。

3. 必要な場合には、Compute テンプレートをカスタマイズして、コントローラーノードに一致する外部ブリッジを追加し、環境ファイルの **OS::TripleO::Compute::Net::SoftwareConfig** のカスタムファイルパスに名前を付けます。
4. オーバークラウドのデプロイ時に、**environments/services/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/services/neutron-ovs-dvr.yaml
```

5. L3 HA が無効になっていることを確認します。



注記

L3 エージェントの外部ブリッジ設定は Red Hat OpenStack Platform 13 で非推奨になり、Red Hat OpenStack Platform 15 で廃止されました。

13.6. 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。

手順

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。
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。

例

```
$ openstack router set --disable router1
```

- b. 高可用性の設定を無効にします。

例

```
$ openstack router set --no-ha router1
```

- c. ルーターが DVR を使用するよう設定します。

例

```
$ openstack router set --distributed router1
```

- d. ルーターを有効にします。

例

```
$ openstack router set --enable router1
```

- e. 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。

関連情報

- [ML2 OVS を使用した DVR のデプロイ](#)

13.7. 分散仮想ルーター (DVR) が無効になっている ML2/OVN OPENSTACK のデプロイ

Open Virtual Network メカニズムドライバー (ML2/OVN) および DVR を使用する neutron Modular Layer 2 プラグインに、新たな Red Hat OpenStack Platform (RHOSP) デプロイメントのデフォルト。

DVR トポロジーでは、Floating IP アドレスを持つコンピューターノードは、仮想マシンインスタンスとルーターに外部接続 (north-south トラフィック) を提供するネットワーク間のトラフィックをルーティングします。インスタンス間のトラフィック (east-west トラフィック) も分散されます。

必要に応じて、DVR を無効にしてデプロイできます。これにより、north-south DVR が無効になり、north-south トラフィックでコントローラーまたはネットワークノードを通過する必要があります。DVR が無効であっても、east-west ルーティングは常に ML2/OVN デプロイメントで分散されます。

前提条件

- RHOSP 16.1 ディストリビューションでカスタマイズとデプロイメントの準備が整った。

手順

1. 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:
  NeutronEnableDVR: false
```

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。以下に例を示します。

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

13.7.1. 関連情報

- ネットワークガイドの [分散仮想ルーター \(DVR\) について](#)

第14章 IPV6 を使用したプロジェクトネットワーク


14.1. IPV6 サブネットのオプション

Red Hat OpenStack Platform (RHOSP) プロジェクトネットワークに IPv6 サブネットを作成する場合、アドレスモードおよびルーター広告モードを指定して、以下の表に示す特定の結果を得ることができます。



注記

ML2/OVN デプロイメントでは、RHOSP は IPv6 接頭辞の委譲をサポートしません。グローバルユニキャストアドレスの接頭辞を手動で設定する必要があります。

RA モード	アドレスモード	結果
ipv6_ra_mode=not set	ipv6-address-mode=slaac	<div>インスタンスは、ステートレスアドレス自動設定 (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。</div> <div><div><p>注記</p><p>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 <code>address_assign_type</code> を使用してサブネットを作成することはできません。</p></div></div>
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	<div>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。</div>

RA モード	アドレスモード	結果
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。
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。

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

Red Hat OpenStack Platform (RHOSP) プロジェクトネットワークに IPv6 サブネットを作成することができます。

たとえば、QA という名前のプロジェクトの database-servers という名前のネットワークに、ステートフル DHCPv6 を使用して IPv6 サブネットを作成することができます。

手順

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** コマンドで、プロジェクト ID、ネットワーク名、および ipv6 アドレスモードを指定します。

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

検証手順

1. 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 |
+-----+-----+-----+-----+
-----+
```

結果

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 |
+-----+-----+-----+-----+-----+-----+
-----+
```

関連情報

IPv6 サブネットで特定の結果を得るためのルーター広告モードとアドレスモードの組み合わせを探すには、[ネットワークガイドの IPv6 サブネットのオプション](#) を参照してください。

第15章 プロジェクトクォータの管理

15.1. 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。

手順

- `/var/lib/config-data/puppet-generated/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。

15.2. 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

15.3. 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数

15.4. セキュリティーグループのクォータオプション

Networking サービスクォータエンジンは、セキュリティーグループおよびセキュリティーグループルールを管理し、デフォルトのセキュリティーグループ (および IPv4 および IPv6 のすべての送信トラフィックを許可する 2 つのデフォルトのセキュリティーグループルール) を作成する前にすべての

クォータをゼロに設定することはできません。新規プロジェクトの作成時に、ネットワークまたはポートが作成されるまで、またはセキュリティーグループもしくはセキュリティーグループルールを一覧表示するまで、Networking サービスはデフォルトのセキュリティーグループを作成しません。

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

15.5. 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マニターの数
ヘルスマニターはリソースを消費しませんが、OpenStack Networking はヘルスマニターをリソースのコンシューマーとみなすため、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コンシューマーとみなすため、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16章 ルーティング対応プロバイダーネットワークのデプロイ

16.1. ルーティング対応プロバイダーネットワークのメリット

Red Hat OpenStack Platform (RHOSP) では、オペレーターはルーティング対応プロバイダーネットワーク (RPN) を作成することができます。RPN は通常エッジデプロイメントで使用され、1つのセグメントしか持たない従来のネットワークとは異なり、複数のレイヤー 2 ネットワークセグメントに基づきます。

エンドユーザー向けには1つのネットワークしか表示されないの、ルーティング対応プロバイダーネットワーク (RPN) によりクラウドが単純化されます。クラウドオペレーターの場合、ルーティング対応プロバイダーネットワークはスカルルティーとフォールトトレランスを提供します。たとえば、重大なエラーが発生した場合でも、1つのセグメントしか影響を受けず、ネットワーク全体で障害が発生することはありません。

ルーティング対応プロバイダーネットワーク (RPN) 以前は、オペレーターは、通常以下のアーキテクチャーのいずれかを選択する必要がありました。

- 単一の大規模レイヤー 2 ネットワーク
- 複数の小規模レイヤー 2 ネットワーク

単一の大規模レイヤー 2 ネットワークの場合、スケーリングによりネットワークが複雑になり、耐障害性が低下します (障害ドメインの数が増えます)。

複数の小規模レイヤー 2 ネットワークの場合、スケーリングへの対応は良好で障害ドメインの数は減りますが、エンドユーザーにとっては複雑になる可能性があります。

Red Hat OpenStack Platform 16.1.1 以降では、ML2/OVS または SR-IOV メカニズムドライバーを使用してルーティング対応プロバイダーネットワークをデプロイすることができます。

関連情報

- [「ルーティング対応プロバイダーネットワークの概要」](#)

16.2. ルーティング対応プロバイダーネットワークの概要

ネットワークサブネットとセグメント間の1対1の関連付けが原因で、ルーティング対応プロバイダーネットワーク (RPN) は他の種別のネットワークとは異なります。以前のバージョンでは、Red Hat OpenStack (RHOSP) Networking サービスは RPN をサポートしていませんでした。Networking サービスでは、すべてのサブネットが同じセグメントに属するか、あるいはいずれのセグメントにも属さない必要があったためです。

ルーティング対応プロバイダーネットワークでは、仮想マシン (VM) インスタンスで利用可能な IP アドレスは、特定のコンピュートノードで利用可能なネットワークのセグメントによって異なります。Networking サービスのポートは、1つのネットワークセグメントにのみ関連付けることができます。

従来のネットワーク設定と同様に、レイヤー 2 (スイッチング) は同じネットワークセグメント上のポート間のトラフィックの移動を処理し、レイヤー 3 (ルーティング) はセグメント間のトラフィックの移動を処理します。

Networking サービスは、セグメント間のレイヤー 3 サービスを提供しません。その代わりに、サブネットをルーティングするのに物理ネットワークインフラストラクチャーに依存します。したがって、従来のプロバイダーネットワークと同様に、Networking サービスおよび物理ネットワークインフラ

トラクチャーには、ルーティング対応プロバイダーネットワークの設定が含まれている必要があります。

Compute サービス (nova) スケジューラーはネットワークセグメントを認識しないため、RPN をデプロイする際に、それぞれのリーフ、ラックセグメント、または DCN エッジサイトを Compute サービスのホストアグリゲートまたはアベイラビリティゾーンにマッピングする必要があります。

DHCP-metadata サービスが必要な場合は、それぞれのエッジサイトまたはネットワークセグメントにアベイラビリティゾーンを定義し、ローカル DHCP エージェントがデプロイされるようにする必要があります。

関連情報

- [「ルーティング対応プロバイダーネットワークのメリット」](#)

16.3. ルーティング対応プロバイダーネットワークの制限

ルーティング対応プロバイダーネットワーク (RPN) はすべてのメカニズムドライバでサポートされてはいないので、Compute サービススケジューラーおよびその他のソフトウェアには、以下の一覧で説明する制約があります。

- ルーティング対応プロバイダーネットワークは、ML2/OVS および SR-IOV メカニズムドライバでのみサポートされます。
Open Virtual Network (OVN) には対応していません。
- リモートデプロイメントおよびエッジデプロイメント向けの OVS-DPDK(DHCP を使用しない) のサポートは、Red Hat OpenStack Platform 16.1.4 以降ではテクノロジープレビューとなっています。
- 中央 SNAT または Floating IP を使用した North-south ルーティングはサポートされません。
- SR-IOV または PCI パススルーを使用する場合、物理ネットワーク (physnet) の名前は中央サイトおよびリモートサイトまたはセグメントで同一でなければなりません。セグメント ID を再利用することはできません。
- Compute サービス (nova) スケジューラーは、セグメントを認識しません。(それぞれのセグメントまたはエッジサイトを Compute のホストアグリゲートまたはアベイラビリティゾーンにマッピングする必要があります)。現在、利用することのできる仮想マシンインスタンスのブートオプションは2つだけです。
 - **port-id** を使用し IP アドレスは使用せず Compute のアベイラビリティゾーン (セグメントまたはエッジサイト) を指定するブート
 - **network-id** を使用し Compute のアベイラビリティゾーン (セグメントまたはエッジサイト) を指定するブート
- コールドマイグレーションまたはライブマイグレーションは、移行先 Compute アベイラビリティゾーン (セグメントまたはエッジサイト) を指定する場合にのみ機能します。

16.4. ルーティング対応プロバイダーネットワークの準備

Red Hat OpenStack Platform (RHOSP) でルーティング対応プロバイダーネットワークを作成するには、さまざまなタスクを実施する必要があります。

手順

1. ネットワーク内では、それぞれのセグメントに固有の物理ネットワーク名を使用します。これにより、サブネット間で同じセグメンテーション情報を再利用することができます。
たとえば、特定のプロバイダーネットワークのすべてのセグメントで同じ VLAN ID を使用します。
2. セグメント間のルーティングを実装します。
セグメント上の各サブネットには、その特定のサブネット上のルーターインターフェイスのゲートウェイアドレスが含まれている必要があります。

表16.1 ルーティングのセグメント例

セグメント	バージョン	アドレス	ゲートウェイ
segment1	4	203.0.113.0/24	203.0.113.1
segment1	6	fd00:203:0:113::/64	fd00:203:0:113::1
segment2	4	198.51.100.0/24	198.51.100.1
segment2	6	fd00:198:51:100::/64	fd00:198:51:100::1

3. セグメントをコンピュータードにマッピングします。
ルーティング対応プロバイダーネットワークでは、コンピュータードが異なるセグメントに存在することになります。ルーティング対応プロバイダーネットワークのすべてのコンピュータードホストが、そのセグメントのいずれかに直接接続されているようにします。

表16.2 セグメントとコンピュータードのマッピング例

ホスト	ラック	物理ネットワーク
compute0001	rack 1	segment 1
compute0002	rack 1	segment 1
...
compute0101	rack 2	segment 2
compute0102	rack 2	segment 2
compute0102	rack 2	segment 2
...

4. セグメントごとに少なくとも1つの DHCP エージェントをデプロイします。
従来のプロバイダーネットワークとは異なり、DHCP エージェントはネットワーク内で複数のセグメントをサポートすることができません。ノード数を減らすために、ネットワークノードではなくセグメントが含まれるコンピュータードに DHCP エージェントをデプロイします。

表16.3 セグメントごとの DHCP エージェントのマッピング例

ホスト	ラック	物理ネットワーク
network0001	rack 1	segment 1
network0002	rack 1	segment 1
...

カスタムロールファイルを使用して、DHCP エージェントおよび Networking サービスのメタデータエージェントをコンピュータノードにデプロイします。

以下に例を示します。

```
#####
####
# Role: ComputeSriov                                     #
#####
####
- name: ComputeSriov
  description: |
    Compute SR-IOV Role
  CountDefault: 1
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  RoleParametersDefault:
    TunedProfileName: "cpu-partitioning"
  update_serial: 25
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    ...
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronMetadataAgent
    ...
```

カスタム環境ファイルに以下のキーと値のペアを追加します。

```
parameter_defaults:
  ....
  NeutronEnableIsolatedMetadata: 'True'
  ....
```

5. RHOSP Placement サービス **python3-osc-placement** パッケージがアンダークラウドにインストールされていることを確認します。

このパッケージは、RHOSP 16.1.6 以降のアンダークラウドで利用できます。RHOSP の以前のバージョンでは、パッケージを手動でインストールする必要があります。実行している RHOSP のバージョンを確認するには、アンダークラウドで以下のコマンドを入力します。

```
$ cat /etc/rhosp-release
Red Hat OpenStack Platform release 16.1.5 GA (Train)
```

Placement サービスをインストールするには、アンダークラウドに root としてログインし、以下のコマンドを実行します。

```
# yum install python3-osc-placement
```

関連情報

- [「ルーティング対応プロバイダーネットワークの作成」](#)
- [オーバークラウドの高度なカスタマイズの コンポーザブルサービスとカスタムロール](#)

16.5. ルーティング対応プロバイダーネットワークの作成

エンドユーザー向けには1つのネットワークしか表示されないのが、ルーティング対応プロバイダーネットワーク (RPN) により Red Hat OpenStack Platform (RHOSP) クラウドが単純化されます。クラウドオペレーターの場合、ルーティング対応プロバイダーネットワークはスカルラティーとフォールトトレランスを提供します。

以下の手順を実施すると、2つのネットワークセグメントを持つルーティング対応プロバイダーネットワークが作成されます。それぞれのセグメントには、1つのIPv4 サブネットおよび1つのIPv6 サブネットが含まれます。

前提条件

- xref:prepare-routed-prov-network_deploy-routed-prov-networks の手順を完了する。

手順

1. デフォルトのセグメントが含まれる 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 プロバイダーネットワークは **multisegment1** という名前で、**provider1** という名前の物理ネットワークおよび ID が **128** の VLAN を使用します。

例

```
$ openstack network create --share --provider-physical-network provider1 \
--provider-network-type vlan --provider-segment 128 multisegment1
```

出力例

```
+-----+-----+
| Field          | Value                               |
+-----+-----+
| admin_state_up | UP                                  |
| id             | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
```

```

| ipv4_address_scope | None |
| ipv6_address_scope | None |
| l2_adjacency       | True |
| mtu                 | 1500 |
| name                | multisegment1 |
| port_security_enabled | True |
| provider:network_type | vlan |
| provider:physical_network | provider1 |
| provider:segmentation_id | 128 |
| revision_number     | 1 |
| router:external     | Internal |
| shared              | True |
| status              | ACTIVE |
| subnets            | |
| tags                | [] |
+-----+

```

2. デフォルトのネットワークセグメントの名前を **segment1** に変更します。

a. セグメント ID を取得します。

```
$ openstack network segment list --network multisegment1
```

出力例

```

+-----+-----+-----+-----+
+-----+
| ID | Name | Network | Network Type |
Segment |
+-----+-----+-----+-----+
+-----+
| 43e16869-ad31-48e4-87ce-acf756709e18 | None | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan | 128 |
+-----+-----+-----+-----+
+-----+

```

b. セグメント ID を使用して、ネットワークセグメントの名前を **segment1** に変更します。

```
$ openstack network segment set --name segment1 43e16869-ad31-48e4-87ce-acf756709e18
```

3. プロバイダーネットワーク上に 2 番目のセグメントを作成します。

以下の例では、ネットワークセグメントは **provider2** という名前の物理ネットワークおよび ID が **129** の VLAN を使用します。

例

```
$ openstack network segment create --physical-network provider2 \
--network-type vlan --segment 129 --network multisegment1 segment2
```

出力例

```

+-----+-----+
| Field | Value |
+-----+-----+

```

```

+-----+-----+
| description | None |
| headers    |      |
| id         | 053b7925-9a89-4489-9992-e164c8cc8763 |
| name       | segment2 |
| network_id | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| network_type | vlan |
| physical_network | provider2 |
| revision_number | 1 |
| segmentation_id | 129 |
| tags       | [] |
+-----+-----+

```

4. ネットワークに **segment1** および **segment2** のセグメントが含まれていることを確認します。

```
$ openstack network segment list --network multisegment1
```

出力例

```

+-----+-----+-----+-----+-----+
----+
| ID | Name | Network | Network Type | Segment |
+-----+-----+-----+-----+-----+
----+
| 053b7925-9a89-4489-9992-e164c8cc8763 | segment2 | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan | 129 |
| 43e16869-ad31-48e4-87ce-acf756709e18 | segment1 | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan | 128 |
+-----+-----+-----+-----+-----+
----+

```

5. **segment1** セグメント上に、IPv4 サブネットおよび IPv6 サブネットをそれぞれ 1 つ作成します。
以下の例では、IPv4 サブネットは **203.0.113.0/24** を使用します。

例

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 4 --subnet-range 203.0.113.0/24 \
  multisegment1-segment1-v4
```

出力例

```

+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 203.0.113.2-203.0.113.254 |
| cidr | 203.0.113.0/24 |
| enable_dhcp | True |
| gateway_ip | 203.0.113.1 |
| id | c428797a-6f8e-4cb1-b394-c404318a2762 |
| ip_version | 4 |
| name | multisegment1-segment1-v4 |
+-----+-----+

```

```
| network_id      | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1                                     |
| segment_id     | 43e16869-ad31-48e4-87ce-acf756709e18 |
| tags           | []                                   |
+-----+-----+
```

以下の例では、IPv6 サブネットは **fd00:203:0:113::/64** を使用します。

例

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 6 --subnet-range fd00:203:0:113::/64 \
  --ipv6-address-mode slaac multisegment1-segment1-v6
```

出力例

```
+-----+-----+
| Field      | Value                                     |
+-----+-----+
| allocation_pools | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr         | fd00:203:0:113::/64                     |
| enable_dhcp   | True                                     |
| gateway_ip    | fd00:203:0:113::1                       |
| id           | e41cb069-9902-4c01-9e1c-268c8252256a    |
| ip_version    | 6                                         |
| ipv6_address_mode | slaac                                   |
| ipv6_ra_mode   | None                                     |
| name          | multisegment1-segment1-v6               |
| network_id     | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9    |
| revision_number | 1                                         |
| segment_id     | 43e16869-ad31-48e4-87ce-acf756709e18    |
| tags          | []                                       |
+-----+-----+
```



注記

デフォルトでは、プロバイダーネットワーク上の IPv6 サブネットは、ステートレスアドレス自動設定 (SLAAC) およびルーター広告の物理ネットワークインフラストラクチャーに基づきます。

6. **segment2** セグメント上に、IPv4 サブネットおよび IPv6 サブネットをそれぞれ1つ作成します。

以下の例では、IPv4 サブネットは **198.51.100.0/24** を使用します。

例

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 4 --subnet-range 198.51.100.0/24 \
  multisegment1-segment2-v4
```

出力例

```

+-----+
| Field      | Value                                |
+-----+
| allocation_pools | 198.51.100.2-198.51.100.254      |
| cidr        | 198.51.100.0/24                    |
| enable_dhcp  | True                               |
| gateway_ip   | 198.51.100.1                       |
| id           | 242755c2-f5fd-4e7d-bd7a-342ca95e50b2 |
| ip_version   | 4                                  |
| name         | multisegment1-segment2-v4         |
| network_id   | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1                                  |
| segment_id   | 053b7925-9a89-4489-9992-e164c8cc8763 |
| tags         | []                                  |
+-----+

```

以下の例では、IPv6 サブネットは **fd00:198:51:100::/64** を使用します。

例

```

$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 6 --subnet-range fd00:198:51:100::/64 \
  --ipv6-address-mode slaac multisegment1-segment2-v6

```

出力例

```

+-----+
| Field      | Value                                |
+-----+
| allocation_pools | fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff |
| cidr        | fd00:198:51:100::/64                    |
| enable_dhcp  | True                               |
| gateway_ip   | fd00:198:51:100::1                       |
| id           | b884c40e-9cfe-4d1b-a085-0a15488e9441      |
| ip_version   | 6                                  |
| ipv6_address_mode | slaac                               |
| ipv6_ra_mode  | None                                |
| name         | multisegment1-segment2-v6         |
| network_id   | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9      |
| revision_number | 1                                  |
| segment_id   | 053b7925-9a89-4489-9992-e164c8cc8763      |
| tags         | []                                  |
+-----+

```

検証

1. それぞれの IPv4 サブネットが少なくとも 1 つの DHCP エージェントと関連付けられていることを確認します。

```

$ openstack network agent list --agent-type dhcp --network multisegment1

```

出力例

-

```

+-----+-----+-----+-----+-----+-----+
| ID | Agent Type | Host | Availability Zone | Alive | State | Binary |
+-----+-----+-----+-----+-----+-----+
| c904ed10-922c-4c1a-84fd-d928abaf8f55 | DHCP agent | compute0001 | nova | :- )
| UP | neutron-dhcp-agent |
| e0b22cc0-d2a6-4f1c-b17c-27558e20b454 | DHCP agent | compute0101 | nova | :- )
| UP | neutron-dhcp-agent |
+-----+-----+-----+-----+-----+-----+

```

2. Compute サービスの配置 API の各セグメントの IPv4 サブネットにインベントリーが作成されたことを確認します。
すべてのセグメント ID に対して、以下のコマンドを実行します。

```

$ SEGMENT_ID=053b7925-9a89-4489-9992-e164c8cc8763
$ openstack resource provider inventory list $SEGMENT_ID

```

出力例

この出力例では、セグメントのうち1つだけが表示されています。

```

+-----+-----+-----+-----+-----+-----+
| resource_class | allocation_ratio | max_unit | reserved | step_size | min_unit | total |
+-----+-----+-----+-----+-----+-----+
| IPV4_ADDRESS | 1.0 | 1 | 2 | 1 | 1 | 30 |
+-----+-----+-----+-----+-----+-----+

```

3. Compute サービスの各セグメントにホストアグリゲートが作成されたことを確認します。

```

$ openstack aggregate list

```

出力例

以下の例では、1つのセグメントだけが示されています。

```

+-----+-----+-----+-----+-----+-----+
| Id | Name | Availability Zone |
+-----+-----+-----+-----+-----+-----+
| 10 | Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763 | None |
+-----+-----+-----+-----+-----+-----+

```

4. 1つまたは複数のインスタンスを起動します。それぞれのインスタンスは、特定のコンピュータノードで使用するセグメントに従って IP アドレスを取得します。

注記

ユーザーがポート作成要求で Fixed IP を指定すると、その特定の IP が直ちにポートに割り当てられます。ただし、ポートを作成してインスタンスに渡した際の動作は、従来のネットワークとは異なります。ポート作成要求で Fixed IP が指定されていない場合、特定のコンピュートノードが明確になるまで Networking サービスは IP アドレスをポートに割り当てません。たとえば、以下のコマンドを実行した場合:

```
$ openstack port create --network multisegment1 port1
```

出力例

```
+-----+
| Field          | Value                                |
+-----+
| admin_state_up | UP                                  |
| binding_vnic_type | normal                              |
| id              | 6181fb47-7a74-4add-9b6b-f9837c1c90c4 |
| ip_allocation   | deferred                            |
| mac_address     | fa:16:3e:34:de:9b                  |
| name            | port1                               |
| network_id      | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| port_security_enabled | True                                |
| revision_number | 1                                    |
| security_groups | e4fcef0d-e2c5-40c3-a385-9c33ac9289c5 |
| status          | DOWN                                |
| tags            | []                                   |
+-----+
```

関連情報

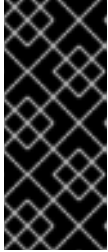
- [「ルーティング対応プロバイダーネットワークの準備」](#)
- コマンドラインインターフェイスリファレンス の [network create](#)
- コマンドラインインターフェイスリファレンス の [network segment create](#)
- コマンドラインインターフェイスリファレンス の [subnet create](#)
- Command Line Interface Reference の [port create](#)

16.6. ルーティング非対応からルーティング対応プロバイダーネットワークへの移行

ネットワークのサブネットをネットワークセグメントの ID に関連付けることにより、ルーティングに対応しないネットワークをルーティング対応プロバイダーネットワーク (RPN) に移行することができます。

前提条件

- 移行するルーティング非対応のネットワークには、セグメントおよびサブネットがそれぞれ 1 つ だけ 含まれている必要があります。



重要

複数のサブネットまたはネットワークセグメントが含まれるルーティング非対応プロバイダーネットワークの場合、ルーティング対応プロバイダーネットワークに安全に移行することはできません。ルーティング非対応のネットワークでは、サブネット割り当てプールからのアドレスは、ポートがバインドされるネットワークセグメントを考慮せずにポートに割り当てられます。

手順

1. 移行されるネットワークについて、現在のネットワークセグメントの ID を取得します。

例

```
$ openstack network segment list --network my_network
```

出力例

```
+-----+-----+-----+-----+
+
| ID                               | Name | Network                               | Network Type | Segment |
+-----+-----+-----+-----+
+
| 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 | None | 45e84575-2918-471c-95c0-018b961a2984 | flat        | None    |
+-----+-----+-----+-----+
+
```

2. 移行されるネットワークについて、現在のサブネットの ID を取得します。

例

```
$ openstack network segment list --network my_network
```

出力例

```
+-----+-----+-----+-----+
+
| ID                               | Name   | Network                               | Subnet      |
+-----+-----+-----+-----+
| 71d931d2-0328-46ae-93bc-126caf794307 | my_subnet | 45e84575-2918-471c-95c0-018b961a2984 | 172.24.4.0/24 |
+-----+-----+-----+-----+
+
```

3. サブネットの現在の **segment_id** の値が **None** であることを確認します。

例

```
$ openstack subnet show my_subnet --c segment_id
```

出力例

```
+-----+-----+
| Field   | Value |
+-----+-----+
```

```
+-----+-----+
| segment_id | None |
+-----+-----+
```

4. サブネットの **segment_id** の値をネットワークセグメント ID に変更します。
以下に例を示します。

```
$ openstack subnet set --network-segment 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7
my_subnet
```

検証

- サブネットが希望のネットワークセグメントに関連付けられていることを確認します。

例

```
$ openstack subnet show my_subnet --c segment_id
```

出力例

```
+-----+-----+
| Field   | Value                                     |
+-----+-----+
| segment_id | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 |
+-----+-----+
```

関連情報

- コマンドラインインターフェイスリファレンス の [subnet show](#)
- コマンドラインインターフェイスリファレンス の [subnet set](#)

第17章 許可するアドレスペアの設定

17.1. 許可するアドレスペアの概要

許可するアドレスペアは、特定の MAC アドレス、IP アドレス、またはその両方を指定して、サブネットに関係なくネットワークトラフィックがポートを通過できるようにする際に設定します。許可するアドレスペアを定義すると、VRRP (仮想ルーター冗長プロトコル) 等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つの仮想マシンインスタンス間で 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。

Red Hat OpenStack Platform コマンドラインクライアントの **openstack port** コマンドを使用して、許可するアドレスペアを定義します。

重要

許可するアドレスペアでは、より広い IP アドレス範囲を持つデフォルトのセキュリティグループを使用しないように注意してください。そうしないと、1つのポートが同じネットワーク内の他のすべてのポートのセキュリティグループをバイパスできてしまいます。

たとえば、以下のコマンドはネットワーク内のすべてのポートに影響を与え、すべてのセキュリティグループをバイパスします。

```
# openstack port set --allowed-address mac-address=3e:37:09:4b,ip-address=0.0.0.0/0 9e67d44eab334f07bf82fa1b17d824b6
```

注記

ML2/OVN メカニズムドライバーネットワークバックエンドを使用すると、仮想 IP を作成することができます。ただし、**allowed_address_pairs** を使用するバインドポートに割り当てられる IP アドレスは、仮想ポートの IP アドレス (/32) と一致する必要があります。

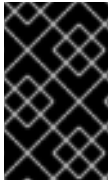
バインドポート **allowed_address_pairs** に CIDR 形式の IP アドレスを使用する場合には、ポート転送はバックエンドで設定されず、バインドされた IP ポートに到達できる必要のある CIDR の IP でトラフィックが失敗します。

関連情報

- コマンドラインインターフェイスリファレンスの [port](#)
- 「ポートの作成および1つのアドレスペアの許可」
- 「許可するアドレスペアの追加」

17.2. ポートの作成および1つのアドレスペアの許可

ポートを作成して許可するアドレスペアを設定すると、ネットワークトラフィックはサブネットに関係なくポートを通過して流れることができます。



重要

許可するアドレスペアでは、より広い IP アドレス範囲を持つデフォルトのセキュリティグループを使用しないでください。そうしないと、1つのポートが同じネットワーク内の他のすべてのポートのセキュリティグループをバイパスできてしまいます。

手順

- 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。

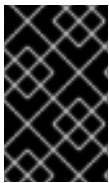
```
$ openstack port create --network <network> --allowed-address mac-address=
<mac_address>,ip-address=<ip_cidr> <port_name>
```

関連情報

- コマンドラインインターフェイスリファレンスの [port](#)

17.3. 許可するアドレスペアの追加

ポートに許可するアドレスペアを設定して、ネットワークトラフィックがサブネットに関係なくポートを通過して流れるのを許可することができます。



重要

許可するアドレスペアでは、より広い IP アドレス範囲を持つデフォルトのセキュリティグループを使用しないでください。そうしないと、1つのポートが同じネットワーク内の他のすべてのポートのセキュリティグループをバイパスできてしまいます。

手順

- 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 --allowed-address mac-address=<mac_address>,ip-address=<ip_cidr>
<port>
```



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。

関連情報

- コマンドラインインターフェイスリファレンスの [port](#)

第18章 一般的なネットワーク管理タスク

Red Hat OpenStack Platform Networking サービス (neutron) で、レイヤー 2 Population ドライバーの設定や内部 DNS によってポートに割り当てられた名前の指定など、管理タスクを実施しなければならない場合があります。

18.1. 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
```

18.2. VRRP パケットロスを避けるための KEEPALIVED の調整

1つのホスト上の高可用性 (HA) ルーターの数が多い場合、HA ルーターのフェイルオーバーが発生す

ると、仮想ルーター冗長プロトコル (VRRP) メッセージで IRQ キューがオーバーフローする可能性があります。このオーバーフローにより、Open vSwitch (OVS) がそれらの VRRP メッセージに応答して転送することができなくなります。

VRRP パケットのオーバーロードを避けるには、Controller ロールの **ExtraConfig** セクションの **ha_vrrp_advert_int** パラメーターを使用して、VRRP 広告の間隔を増やす必要があります。

手順

1. アンダークラウドに stack ユーザーとしてログインし、source コマンドで **stackrc** ファイルを読み込み、director コマンドラインツールを有効にします。

例

```
$ source ~/stackrc
```

2. カスタム YAML 環境ファイルを作成します。

例

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

ヒント

Red Hat OpenStack Platform Orchestration サービス (heat) は、**テンプレート** と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル** を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、heat テンプレートをカスタマイズするための特別な種別のテンプレートです。

3. YAML 環境ファイルで、実際のサイトに固有の値と共に **ha_vrrp_advert_int** 引数を使用して VRRP 広告の間隔を増やします。(デフォルトは **2** 秒です)。
Gratuitous ARP メッセージに値を設定することもできます。

ha_vrrp_garp_master_repeat

マスター状態への移行後に一度に送信する Gratuitous ARP メッセージの数。(デフォルトのメッセージ数は 5 です。)

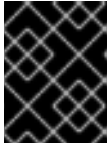
ha_vrrp_garp_master_delay

優先順位の低い広告がマスター状態で受信された後、2 番目の Gratuitous ARP メッセージセットの遅延。(デフォルトは 5 秒です。)

例

```
parameter_defaults:
  ControllerExtraConfig:
    neutron::agents::l3::ha_vrrp_advert_int: 7
    neutron::config::l3_agent_config:
      DEFAULT/ha_vrrp_garp_master_repeat:
        value: 5
      DEFAULT/ha_vrrp_garp_master_delay:
        value: 5
```

4. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

関連情報

- RFC 4541 の [2.1.2 項 Data Forwarding Rules](#) サブセクション 2
- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

18.3. DNS がポートに割り当てる名前の指定

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) のポートエクステンション用 `dns_domain` (**`dns_domain_ports`**) を有効にすると、内部 DNS によりポートに割り当てられる名前を指定することができます。

YAML 形式の環境ファイルで RHOSP Orchestration (heat) **`NeutronPluginExtensions`** パラメーターを宣言して、ポートエクステンション用 `dns_domain` を有効にします。対応するパラメーター **`NeutronDnsDomain`** を使用して、デフォルト値 **`openstacklocal`** をオーバーライドするドメイン名を指定します。オーバークラウドの再デプロイ後に、OpenStack Client ポートコマンド **`port set`** または **`port create`** で **`--dns-name`** を指定して、ポート名を割り当てることができます。

また、ポートエクステンション用 `dns_domain` を有効にすると、仮想マシンインスタンスのブート中に、Compute サービスが **`dns_name`** 属性にインスタンスの **`hostname`** 属性を自動的に設定します。ブートプロセスの最後に、`dnsmasq` はインスタンスのホスト名で割り当てられたポートを認識します。

手順

1. アンダークラウドに stack ユーザーとしてログインし、source コマンドで **`stackrc`** ファイルを読み込み、director コマンドラインツールを有効にします。

例

```
$ source ~/stackrc
```

2. カスタム YAML 環境ファイル (**`my-neutron-environment.yaml`**) を作成します。



注記

丸カッコ内の値は、この手順のコマンド例で使用するサンプルの値です。これらのサンプル値を、実際のサイトに適した値に置き換えてください。

例

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

ヒント

アンダークラウドには、オーバークラウドの作成プランを形作るさまざまな Orchestration サービスのテンプレートが含まれます。YAML フォーマットの環境ファイルを使って、オーバークラウドの特性をカスタマイズすることができます。このファイルで、Orchestration サービスのコアテンプレートコレクションのパラメーターおよびリソースを上書きします。必要に応じていくつでも環境ファイルを追加することができます。

- 環境ファイルに **parameter_defaults** セクションを追加します。このセクションで、ポートエクステンション用 dns_domain **dns_domain_ports** を追加します。

例

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



注記

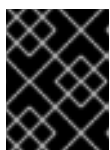
dns_domain_ports を設定する場合は、デプロイメントで DNS Integration エクステンション **dns_domain** も使用しないようにしてください。これらのエクステンションは互換性がなく、両方のエクステンションを同時に定義することはできません。

- また、**parameter_defaults** セクションで、**NeutronDnsDomain** パラメーターを使用してドメイン名 (**example.com**) を追加します。

例

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

- コア Orchestration テンプレート、環境ファイル、およびこの新しい環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

検証

1. オーバークラウドにログインし、ネットワーク (**public**) に新しいポート (**new_port**) を作成します。DNS 名 (**my_port**) をポートに割り当てます。

例

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

2. ポート (**new_port**) の詳細を表示します。

例

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

出力

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

dns_assignment セクションにおいて、ポートの完全修飾ドメイン名 (**fqdn**) 値には、DNS 名 (**my_port**) と、前のステップで **NeutronDnsDomain** で設定したドメイン名 (**example.com**) の連結が含まれています。

3. 作成したポート (**new_port**) を使用して、新しい仮想マシンインスタンス (**my_vm**) を作成します。

例

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

4. ポート (**new_port**) の詳細を表示します。

例

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

出力

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

Compute サービスは、**dns_name** 属性を元の値 (**my_port**) からポートが関連付けられたインスタンスの名前 (**my_vm**) に変更します。

関連情報

- [Advanced Overcloud Customization](#) の [Environment files](#)
- [Advanced Overcloud Customization](#) ガイドの [Including environment files in overcloud creation](#)
- [コマンドラインインターフェイスリファレンス](#) の [port](#)
- [コマンドラインインターフェイスリファレンス](#) の [server create](#)

18.4. DHCP 属性のポートへの割り当て

Red Hat Openstack Platform (RHOSP) Networking サービス (neutron) エクステンションを使用して、ネットワーク機能を追加できます。追加の DHCP オプションエクステンション (**extra_dhcp_opt**) を使用して、DHCP 属性を持つ DHCP クライアントのポートを設定できます。たとえば、**tftp-server**、**server-ip-address**、**bootfile-name** などの PXE ブートオプションを DHCP クライアントポートに追加できます。

extra_dhcp_opt 属性の値は、DHCP オプションオブジェクトの配列であり、各オブジェクトには **opt_name** と **opt_value** が含まれています。IPv4 がデフォルトのバージョンですが、3 番目のオプションである **ip-version=6** を含めることで、これを IPv6 に変更できます。

VM インスタンスが起動すると、RHOSP Networking サービスは DHCP プロトコルを使用してインスタンスにポート情報を提供します。実行中のインスタンスにすでに接続されているポートに DHCP 情報を追加した場合、インスタンスの再起動時に、新しい DHCP ポート情報のみを使用します。

より一般的な DHCP ポート属性には、**bootfile-name**、**dns-server**、**domain-name**、**mtu**、**server-ip-address**、および **tftp-server** があります。**opt_name** の許容値の完全なセットについては、DHCP の仕様を参照してください。

前提条件

- RHOSP 管理者権限が必要です。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. `source` コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. カスタム YAML 環境ファイルを作成します。

例

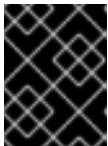
```
$ vi /home/stack/templates/my-octavia-environment.yaml
```

4. 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。これらのキーワードの下に、追加の DHCP オプションエクステンション **extra_dhcp_opt** を追加します。

例

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

5. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、deployment コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

検証

1. Source コマンドで認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

2. ネットワーク (**public**) に新しいポート (**new_port**) を作成します。DHCP 仕様から新しいポートに有効な属性を割り当てます。

例

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

3. ポート (**new_port**) の詳細を表示します。

例

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

出力例

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

関連情報

- [OVN supported DHCP options](#)
- [Dynamic Host Configuration Protocol \(DHCP\) and Bootstrap Protocol \(BOOTP\) Parameters](#)
- [Advanced Overcloud Customization](#) の [Environment files](#)
- [Advanced Overcloud Customization](#) ガイドの [Including environment files in overcloud creation](#)
- [Command Line Interface Reference](#) の [port create](#)
- [コマンドラインインターフェイスリファレンス](#) の [port show](#)

18.5. カーネルモジュールの読み込み

一部の Red Hat OpenStack Platform (RHOSP) 機能には、特定のカーネルモジュールを読み込む必要があります。たとえば、OVS ファイアウォールドライバーの場合、2つの仮想マシンインスタンス間の GRE トンネリングをサポートするには、**nf_conntrack_proto_gre** カーネルモジュールを読み込む必要があります。

特別な Orchestration サービス (heat) パラメーター **ExtraKernelModules** を使用することで、GRE トンネリング等の機能に必要なカーネルモジュールについての設定情報が heat に保存されるようになります。この後、通常のモジュール管理時に、これらの必要なカーネルモジュールが読み込まれます。

手順

1. アンダークラウドホストに stack ユーザーとしてログインして、カスタム YAML 環境ファイルを作成します。

例

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

ヒント

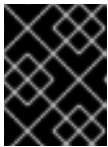
heat は、**テンプレート** と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル** を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、heat テンプレートをカスタマイズするための特別な種別のテンプレートです。

2. YAML 環境ファイルの **parameter_defaults** セクションで、**ExtraKernelModules** を読み込むモジュールの名前に設定します。

例

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

3. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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

検証

- heat がモジュールを正しく読み込んでいれば、コンピュータノードで **lsmod** コマンドを実行すると、出力が表示されるはずです。

例

```
sudo lsmod | grep nf_conntrack_proto_gre
```

関連情報

- [Advanced Overcloud Customization の Environment files](#)
- [Advanced Overcloud Customization ガイドの Including environment files in overcloud creation](#)

18.6. 共有セキュリティーグループの設定

1つ以上の Red Hat OpenStack Platform (RHOSP) プロジェクトがデータを共有できるようにするには、RHOSP Networking サービス (neutron)RBAC ポリシー機能を使用してセキュリティーグループを共有できます。OpenStack クライアントを使用して、セキュリティーグループと Networking サービスのロールベースアクセス制御 (RBAC) ポリシーを作成します。

インスタンスの作成時にセキュリティーグループをインスタンスに直接適用することや、実行中のインスタンスのポートに適用することができます。



注記

インスタンスの作成中に、ロールベースのアクセス制御 (RBAC) 共有セキュリティーグループを直接インスタンスに適用することはできません。RBAC 共有セキュリティーグループをインスタンスに適用するには、最初にポートを作成し、共有セキュリティーグループをそのポートに適用してから、そのポートをインスタンスに割り当てる必要があります。[セキュリティーグループのポートへの追加](#) を参照してください。

前提条件

- 共有する RHOSP プロジェクトが少なくとも 2 つある。
- プロジェクトの 1 つ **現在のプロジェクト** で、別のプロジェクト **ターゲットプロジェクト** と共有するセキュリティーグループを作成している。
以下の例では、**ping_ssh** セキュリティーグループが作成されます。

例

```
$ openstack security group create ping_ssh
```

手順

- セキュリティーグループが含まれる現在のプロジェクトのオーバークラウドにログインします。
- ターゲットプロジェクトの名前または ID を取得します。

```
$ openstack project list
```

- RHOSP プロジェクト間で共有するセキュリティーグループの名前または ID を取得します。

```
$ openstack security group list
```

- 前のステップの識別子を使用して、**openstack network rbac create** コマンドを使用して RBAC ポリシーを作成します。
以下の例では、ターゲットプロジェクトの ID は **32016615de5d43bb88de99e7f2e26a1e** です。セキュリティーグループの ID は **5ba835b7-22b0-4be6-bdbe-e0722d1b5f24** です。

例

```
$ openstack network rbac create --target-project \
32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
--type security_group 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24
```

--target-project

セキュリティーグループへのアクセスを必要とするプロジェクトを指定します。

ヒント

--target-project <target-project> 引数の代わりに **--target-all-projects** を使用して、すべてのプロジェクト間でデータを共有できます。デフォルトでは、admin ユーザーのみがこの特権を持ちます。

--action access_as_shared

プロジェクトを実行できるものを指定します。

--type

ターゲットオブジェクトがセキュリティーグループであることを示します。

5ba835b7-22b0-4be6-bdbe-e0722d1b5f24

は、アクセスが許可される特定のセキュリティーグループの ID です。

そのポートにバインドできるだけでなく、OpenStack クライアントの **security group** コマンドを実行すると、ターゲットプロジェクトはセキュリティーグループにアクセスできます。他のユーザー (管理者および所有者以外) はセキュリティーグループにアクセスすることはできません。

ヒント

ターゲットプロジェクトのアクセス権を削除するには、**openstack network rbac delete** コマンドを使用して、アクセスを許可する RBAC ポリシーを削除します。

関連情報

- インスタンスの作成と管理 ガイドの [セキュリティーグループの作成](#)
- コマンドラインインターフェイスリファレンス の [security group create](#)
- コマンドラインインターフェイスリファレンス の [network rbac create](#)

第19章 レイヤー 3 高可用性 (HA) の設定

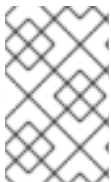
19.1. 高可用性 (HA) なしの RHOSP NETWORKING サービス

高可用性 (HA) 機能が有効化されていない Red Hat OpenStack Platform (RHOSP) Networking サービスのデプロイメントは、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 Networking サービス Layer 3 (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

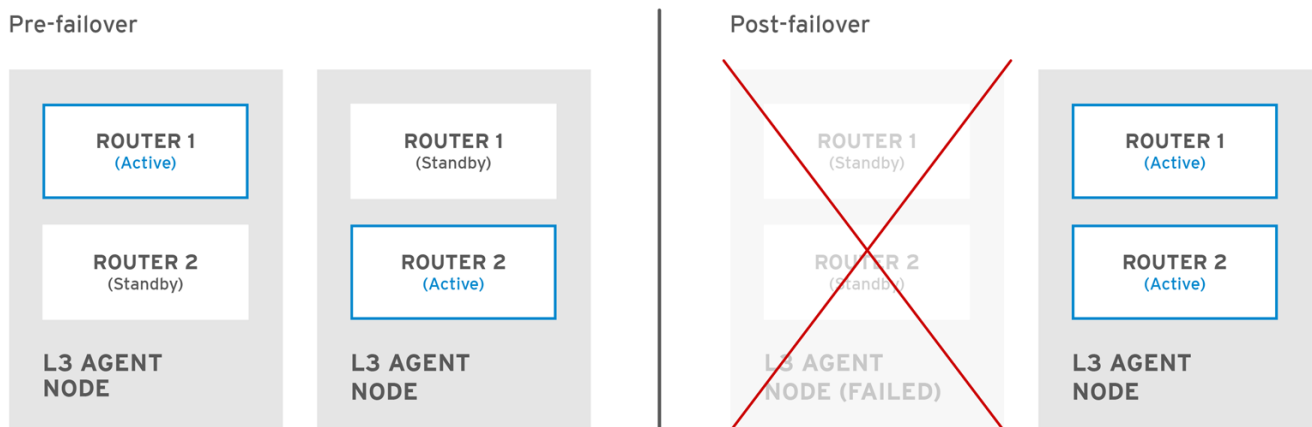
この active/passive の高可用性 (HA) 設定は、業界標準の VRRP (RFC 3768 で定義) を使用してプロジェクトルーターと Floating IP アドレスを保護します。ノードの1つを **active** ルーター、残りを **standby** ロールとして機能するように指定することで、仮想ルーターは複数の Red Hat OpenStack Platform (RHOSP) Networking サービスノードの間で無作為にスケジュールされます。



注記

レイヤー 3 (L3) HA をデプロイするには、冗長系の Networking サービスノードにおいて、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 Networking サービスエージェントノード上で稼働しています。L3 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、L3 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。



OPENSTACK_450456_0617

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。

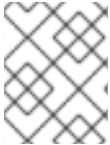
関連情報

- [Virtual Router Redundancy Protocol \(VRRP\)](#)

19.3. レイヤー 3 高可用性 (HA) のフェイルオーバー条件

Red Hat OpenStack Platform (RHOSP) Networking サービスのレイヤー 3 (L3) 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。

- Networking サービス 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

Red Hat OpenStack Platform (RHOSP) Networking サービスレイヤー 3 (L3) 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、L3 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。

- L3 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。
- 高可用性 (HA) ルーターを ML2/OVS に実装する場合、それぞれの L3 エージェントは **haproxy** および **neutron-keepalived-state-change-monitor** プロセスをルーターごとに生成します。各プロセスは約 20 MB のメモリーを消費します。デフォルトでは、各 HA ルーターは 3 つの L3 エージェントにあり、各ノードのリソースを消費します。したがって、RHOSP ネットワークのサイジング時に、実装予定の HA ルーターの数をサポートするのに十分なメモリーが割り当てられているようにしてください。

19.5. RHOSP NETWORKING サービスに対する高可用性 (HA) の変更

Red Hat OpenStack Platform (RHOSP) Networking サービス (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/puppet-generated/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** に加えられる高可用性 (HA) の変更:
 - Networking サービスで使用するスケジューラー (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 (L3) 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。
 - L3 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。
- Networking サービスの L3 エージェントに加えられる 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. RHOSP NETWORKING サービスノードでのレイヤー 3 高可用性 (HA) の有効化

インストール時に、Red Hat OpenStack Platform (RHOSP) director は、RHOSP コントローラーが少なくとも 2 つあり、分散仮想ルーター (DVR) を使用しない場合に、デフォルトで仮想ルーターの高可用性 (HA) を有効化します。RHOSP Orchestration サービス (heat) パラメーター **max_l3_agents_per_router** を使用して、HA ルーターがスケジュールされる RHOSP Networking サービスレイヤー 3 (L3) エージェントの最大数を設定できます。

前提条件

- RHOSP デプロイメントで DVR が使用されていない。
- 2 つ以上の RHOSP コントローラーがデプロイされている。

手順

1. アンダークラウドに stack ユーザーとしてログインし、source コマンドで **stackrc** ファイルを読み込み、director コマンドラインツールを有効にします。

例

```
$ source ~/stackrc
```

2. カスタム YAML 環境ファイルを作成します。

例

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

ヒント

Orchestration サービス (heat) は、**テンプレート**と呼ばれるプランのセットを使用して環境をインストールおよび設定します。**カスタム環境ファイル**を使用して、オーバークラウドの要素をカスタマイズすることができます。このファイルは、heat テンプレートをカスタマイズするための特別な種別のテンプレートです。

3. YAML 環境ファイルで、**NeutronL3HA** パラメーターを **true** に設定します。これにより、director がデフォルトで設定しなかった場合でも HA が有効になります。

```
parameter_defaults:
  NeutronL3HA: 'true'
```

4. HA ルーターがスケジュールされる L3 エージェントの最大数を設定します。
max_l3_agents_per_router パラメーターを、デプロイメント内にあるネットワークノードの最小数と合計数の間の値に設定します。(ゼロの値は、ルーターがすべてのエージェントでスケジュールされることを意味します。)

例

```
parameter_defaults:
  NeutronL3HA: 'true'
  ControllerExtraConfig:
    neutron::server::max_l3_agents_per_router: 2
```

この例では、4 つの Networking サービスノードをデプロイする場合、2 つの L3 エージェントのみが各 HA 仮想ルーターを保護します (1 つはアクティブ、もう 1 つはスタンバイ)。

max_l3_agents_per_router の値を利用可能なネットワークノードの数よりも大きく設定している場合は、新しい L3 エージェントを追加してスタンバイルーターの数をスケールアウトすることができます。デプロイする新しい L3 エージェントノードごとに、Networking サービスは **max_l3_agents_per_router** の上限に達するまで、仮想ルーターをスタンバイバージョンを追加でスケジュールします。

5. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、**openstack overcloud deploy** コマンドを実行します。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

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



注記

NeutronL3HA を **true** に設定すると、作成されるすべての仮想ルーターが HA ルーターにデフォルト設定されます。ルーターの作成時に、**openstack router create** コマンドに **--no-ha** オプションを追加して HA オプションを上書きできます。

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

関連情報

- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

19.7. 高可用性 (HA) RHOSP NETWORKING サービスノード設定の確認

手順

- 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-** の接頭辞が付けられて高可用性 (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 NETWORKER ノードの交換

特定の状況下で、高可用性クラスター内の Networker プロファイルを持つ Red Hat OpenStack Platform (RHOSP) ノードが故障する場合があります。詳細は、[Director インストールと使用方法 ガイドの ノードのプロファイルへのタグ付け](#) を参照してください)。このような場合、ノードをクラスターから削除し、Networking サービス (neutron) エージェントを実行する新しい Networker ノードと交換する必要があります。

このセクションのトピックは、次のとおりです。

- [「ネットワークノードの交換の準備」](#)
- [「Networker ノードの交換」](#)
- [「ノードの再スケジュールと Networking サービスのクリーンアップ」](#)

20.1. ネットワークノードの交換の準備

Red Hat OpenStack Platform (RHOSP) のオーバークラウド上の Networker ノードを交換する場合、いくつかの準備ステップを実行する必要があります。必要な準備手順をすべて完了することで、Networker ノードの交換プロセス中の複雑な状況を回避することができます。

前提条件

- RHOSP デプロイメントは、3 台以上の Networker ノードで高可用性を実現します。

手順

1. アンダークラウドに stack ユーザーとしてログインします。
2. source コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. アンダークラウドで、overcloud スタックの現在のステータスをチェックします。

```
$ openstack stack list --nested
```

overcloud スタックと後続の子スタックは、**CREATE_COMPLETE** または **UPDATE_COMPLETE** のステータスである必要があります。

4. Relax-and-Recover ツールを実行して、アンダークラウドノードの最近のバックアップイメージがあることを確認します。
詳細は、[Backing up and restoring the undercloud and control plane nodes](#) ガイドを参照してください。
5. コントローラーノードに root でログインします。
6. コンテナ上でインタラクティブな bash シェルを開き、Galera クラスターの状態を確認します。

```
# pcs status
```

コントローラーノードが **Master** モードになっていることを確認します。

UUID	Name	Instance UUID
36404147-7c8a-41e6-8c72-6af1e339da2a	None	7bee57cf-4a58-4eaf-b851-f3203f6e5e05
91eb9ac5-7d52-453c-a017-0f2fb289c3cd	None	None
75b25e9a-948d-424a-9b3b-0f2fb289c3cd	None	None
038727da-6a5c-425f-bd45-16aa2bc4ba91	None	763bfec2-9354-466a-ae65-1fdf45d35c61
dc2292e6-4056-46e0-8848-165d06fcc948	None	2017b481-706f-44e1-852a-57fb03ecef11
c7eadcea-e377-4392-9fc3-716f1bd57527	None	5f73c7d7-4826-49a5-b6be-0a95c6bdd2f8
da3a8d19-8a59-4e9d-923a-29254d688f6d	None	cfefaf60-8311-4bc3-9416-46852e2cb83f
807cb6ce-6b94-4cd1-9969-d390650854c7	None	c07c13e6-a845-4791-9628-c8514585fe27
0c245daa-7817-4ae9-a883-fed2e9c68d6c	None	844c9a88-713a-4ff1-8737-30858d724593
e6499ef7-3db2-4ab4-bfa7-feb44c6591c6	None	aef7c27a-f0b4-4814-b0ff-d3f792321212
7545385c-bc49-4eb9-b13c-201368ce1c62	None	c2e40164-c659-4849-a28f-a7b270ed2970

4. **baremetal node maintenance set** コマンドを使用して、ノードをメンテナンスモードに設定します。

例

```
$ openstack baremetal node maintenance set e6499ef7-3db2-4ab4-bfa7-ef59539bf972
```

5. RHOSP ディレクターを含むノードプールに新しいノードを追加するための JSON ファイルを作成します。

例

```
{
  "nodes":[
    {
      "mac":[
        "dd:dd:dd:dd:dd:dd"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"ipmi",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.168.24.207"
    }
  ]
}
```

詳細は、**Director インストールと使用方法 ガイド**の [オーバークラウドへのノードの追加](#) を参照してください。

6. **openstack overcloud node import** コマンドを実行し、新しいノードを登録します。

例

```
$ openstack overcloud node import newnode.json
```

7. 新しいノードを登録した後、以下のコマンドを使用して、イントロスペクションプロセスを起動します。

```
$ openstack baremetal node manage <node>
$ openstack overcloud node introspect <node> --provide
```

8. **openstack baremetal node set** コマンドを使用して、新しいノードに Networker プロファイルのタグを付けます。

例

```
$ openstack baremetal node set --property \
  capabilities='profile:networker,boot_option:local' \
  91eb9ac5-7d52-453c-a017-c0e3d823efd0
```

9. 削除するノードのインデックスを定義した `~/templates/remove-networker.yaml` 環境ファイルを作成します。

例

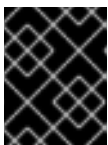
```
parameters:
  NetworkerRemovalPolicies:
    [{'resource_list': ['1']}]
```

10. `~/templates/node-count-networker.yaml` 環境ファイルを作成し、そのファイルに Networker ノードの総数を設定します。

例

```
parameter_defaults:
  OvercloudNetworkerFlavor: networker
  NetworkerCount: 3
```

11. **openstack overcloud deploy** コマンドを実行し、コアヒートテンプレート、環境ファイル、および変更した環境ファイルを含めます。



重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

```
$ openstack overcloud deploy --templates \
  -e <your_environment_files> \
  -e /home/stack/templates/node-count-networker.yaml \
  -e /home/stack/templates/remove-networker.yaml
```

RHOSP director は、古い Networker ノードを削除し、新しいノードを作成して、オーバークラウドスタックを更新します。

検証

1. オーバークラウドスタックのステータスを確認します。

```
$ openstack stack list --nested
```

2. 新しい Networker ノードが一覧表示され、古いノードが削除されていることを確認します。

```
$ openstack server list -c ID -c Name -c Status
```

出力例

```
+-----+-----+-----+
| ID                | Name                | Status |
+-----+-----+-----+
| 861408be-4027-4f53-87a6-cd3cf206ba7a | overcloud-compute-0 | ACTIVE |
| 0966e9ae-f553-447a-9929-c4232432f718 | overcloud-compute-1 | ACTIVE |
| 9c08fa65-b38c-4b2e-bd47-33870bff06c7 | overcloud-compute-2 | ACTIVE |
| a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af | overcloud-controller-0 | ACTIVE |
| cfefaf60-8311-4bc3-9416-6a824a40a9ae | overcloud-controller-1 | ACTIVE |
| 97a055d4-ae4d-481c-82b7-4a5f384036d2 | overcloud-controller-2 | ACTIVE |
| 844c9a88-713a-4ff1-8737-6410bf551d4f | overcloud-networker-0 | ACTIVE |
| c2e40164-c659-4849-a28f-507eb7edb79f | overcloud-networker-2 | ACTIVE |
| 425a0828-b42f-43b0-940c-7fb02522753a | overcloud-networker-3 | ACTIVE |
+-----+-----+-----+
```

関連情報

- Director インストールと使用方法 ガイドの [オーバークラウドへのノードの追加](#)
- Director インストールと使用方法 ガイドの [オーバークラウドのノードの登録](#)
- コマンドラインインターフェイスリファレンス の [baremetal node manage](#)
- Command Line Interface Reference の [overcloud node introspect](#)
- Advanced Overcloud Customization の [Environment files](#)
- Advanced Overcloud Customization ガイドの [Including environment files in overcloud creation](#)

20.3. ノードの再スケジュールと NETWORKING サービスのクリーンアップ

Red Hat OpenStack Platform (RHOSP) の Networker ノードの交換の一環として、削除したノード上のすべての Networking サービスエージェントをデータベースから削除してください。こうすることで、ネットワークサービスがエージェントを out-of-service ("dead") として認識しないようにします。ML2/OVS ユーザーの場合、削除されたノードからエージェントを削除すると、DHCP リソースが他の Networker ノードに自動的に再スケジュールされるようになります。

前提条件

- RHOSP デプロイメントは、3 台以上の Networker ノードで高可用性を実現します。

手順

1. アンダークラウドに stack ユーザーとしてログインします。
2. source コマンドでオーバークラウドの認証情報ファイルを読み込みます。

例

```
$ source ~/overcloudrc
```

3. RHOSP Networking サービスプロセスが存在し、**overcloud-networker-1** に対して out-of-service (**xxx**) とマークされていることを確認します。

```
$ openstack network agent list -c ID -c Binary -c Host -c Alive | grep overcloud-networker-1
```

ML2/OVN の出力例

```
+-----+-----+-----+-----+
| ID                | Host                | Alive | Binary                |
+-----+-----+-----+-----+
| 26316f47-4a30-4baf-ba00-d33c9a9e0844 | overcloud-networker-1 | xxx   | ovn-controller        |
+-----+-----+-----+-----+
```

ML2/OVS の出力例

```
+-----+-----+-----+-----+
| ID                | Host                | Alive | Binary                |
+-----+-----+-----+-----+
| 8377-66d75323e466c-b838-1149e10441ee | overcloud-networker-1 | xxx   | neutron-
metadata-agent |
| b55d-797668c336707-a2cf-cba875eeda21 | overcloud-networker-1 | xxx   | neutron-l3-agent    |
| 9dcb-00a9e32ecde42-9458-01cfa9742862 | overcloud-networker-1 | xxx   | neutron-ovs-
agent          |
| be83-e4d9329846540-9ae6-1540947b2ffd | overcloud-networker-1 | xxx   | neutron-dhcp-
agent          |
+-----+-----+-----+-----+
```

4. **overcloud-networker-1** に登録されているエージェントの UUID を取得します。

```
$ AGENT_UUIDS=$(openstack network agent list -c ID -c Host -c Alive -c Binary -f value |
grep overcloud-networker-1 | cut -d\ -f1)
```

5. データベースから残りの **overcloud-networker-1** エージェントを削除します。

```
$ for agent in $AGENT_UUIDS; do neutron agent-delete $agent ; done
```

出力例

Deleted agent(s): 26316f47-4a30-4baf-ba00-d33c9a9e0844

関連情報

- Command Line Interface Reference の [network agent list](#)

第21章 タグを使用した仮想デバイスの識別

21.1. 仮想デバイスのタグ付け

Red Hat OpenStack Platform では、複数のネットワークインターフェイスまたはブロックデバイスを使用して VM インスタンスを起動する場合、デバイスのタグ付けを使用して、各デバイスの目的のロールをインスタンスのオペレーティングシステムに伝えることができます。インスタンスの起動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。

手順

- 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。

例

```
$ 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。

設定ドライブが有効で、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。

