



Red Hat OpenStack Platform 13

Red Hat OpenDaylight のインストールおよび設定ガイド

Red Hat OpenStack Platform を使用した OpenDaylight のインストールと設定

Red Hat OpenStack Platform 13 Red Hat OpenDaylight のインストールおよび設定ガイド

Red Hat OpenStack Platform を使用した OpenDaylight のインストールと設定

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2018 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, 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 Software Collections 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

本ガイドでは、Red Hat OpenDaylight のインストールと設定について説明します。

目次

前書き	5
第1章 概要	6
1.1. OPENDAYLIGHT とは	6
1.2. OPENSTACK での OPENDAYLIGHT の機能	6
1.2.1. デフォルトの neutron アーキテクチャー	6
1.2.2. OpenDaylight をベースとするネットワークアーキテクチャー	6
1.3. RED HAT OPENSTACK PLATFORM DIRECTOR およびその設計について	6
1.3.1. Red Hat OpenStack Platform director と OpenDaylight	7
1.3.2. Red Hat OpenStack Platform director でのネットワーク分離	8
1.3.3. ネットワークとファイアウォールの設定	11
第2章 OPENDAYLIGHT を実行するための要件	13
2.1. コンピュートノードの要件	13
2.2. コントローラーノードの要件	13
第3章 オーバークラウドへの OPENDAYLIGHT のインストール	15
3.1. デフォルトの設定と設定値のカスタマイズ	15
3.1.1. デフォルトの環境ファイルについての理解	15
3.1.2. OpenDaylight API サービスの設定	16
3.1.2.1. 設定可能なオプション	16
3.1.3. OpenDaylight OVS サービスの設定	17
3.1.3.1. 設定可能なオプション	17
3.1.4. OpenDaylight での neutron メタデータサービスの使用	19
3.1.5. ネットワーク設定と NIC テンプレートについての理解	19
3.2. OPENDAYLIGHT の基本インストール	20
3.2.1. オーバークラウド用の OpenDaylight 環境ファイルの準備	21
3.2.2. OpenDaylight を実装するオーバークラウドのインストール	22
3.3. カスタムロールでの OPENDAYLIGHT のインストール	22
3.3.1. デフォルトロールをベースとするロールファイルのカスタマイズ	23
3.3.2. OpenDaylight 用のカスタムロールの作成	23
3.3.3. OpenDaylight をカスタムロールで実装するオーバークラウドのインストール	25
3.3.4. カスタムロールでの OpenDaylight インストールの検証	26
3.4. SR-IOV 対応の OPENDAYLIGHT のインストール	27
3.4.1. SR-IOV コンピュートロールの準備	27
3.4.2. SR-IOV エージェントサービスの設定	28
3.4.3. SR-IOV 対応の OpenDaylight のインストール	30
3.5. OVS-DPDK 対応の OPENDAYLIGHT のインストール	31
3.5.1. OVS-DPDK デプロイメントファイルの準備	32
3.5.2. OVS-DPDK デプロイメントの設定	33
3.5.3. OVS-DPDK を実装する OpenDaylight のインストール	34
3.5.4. 例: ODL と VXLAN トンネリングを使用する OVS-DPDK の設定	35
3.5.4.1. ComputeOvsDpdk コンポーザブルロールの生成	35
3.5.4.2. CPU アフィニティー用の tuned 設定	35
3.5.4.3. OVS-DPDK パラメーターの設定	36
3.5.4.4. コントローラーノードの設定	38
3.5.4.5. DPDK インターフェースのコンピュートノードの設定	39
3.5.4.6. オーバークラウドのデプロイ	41
3.6. L2GW 対応の OPENDAYLIGHT のインストール	41
3.6.1. L2GW デプロイメントファイルの準備	41
3.6.2. OpenDaylight L2GW デプロイメントの設定	42
3.6.3. L2GW を実装する OpenDaylight のインストール	43

第4章 デプロイメントのテスト	44
4.1. 基本的なテストの実行	44
4.1.1. テスト用の新規ネットワークの作成	44
4.1.2. ネットワークとテスト環境の設定	45
4.1.3. 接続性のテスト	46
4.1.4. デバイスの作成	46
4.2. 高度なテストの実行	47
4.2.1. オーバークラウドノードへの接続	47
4.2.2. OpenDaylight のテスト	48
4.2.3. Open vSwitch のテスト	49
4.2.4. コンピュートノード上の Open vSwitch の設定の確認	51
4.2.5. neutron の設定の確認	52
第5章 デバッグ	54
5.1. ログの特定	54
5.1.1. OpenDaylight ログへのアクセス	54
5.1.2. OpenStack Networking のログへのアクセス	54
5.2. ネットワークエラーのデバッグ	54
5.2.1. OpenFlow のフローを使用した高度なデバッグ	55
5.2.2. OpenFlow でのパケットのトラバース	56
第6章 デプロイメントの例	58
6.1. テナントネットワークを使用した模範的なインストールシナリオ	58
6.1.1. 物理トポロジー	58
6.1.2. 物理ネットワーク環境のプランニング	58
6.1.3. NIC の接続性のプランニング	59
6.1.4. ネットワーク、VLAN、IP のプランニング	59
6.1.5. このシナリオで使用する OpenDaylight の設定ファイル	61
6.1.5.1. extra_env.yaml ファイル	61
6.1.5.2. undercloud.conf ファイル	61
6.1.5.3. network-environment.yaml ファイル	62
6.1.5.4. controller.yaml ファイル	64
6.1.5.5. compute.yaml ファイル	67
6.1.6. このシナリオで使用する Red Hat OpenStack Platform director の設定ファイル	69
6.1.6.1. neutron.conf ファイル	69
6.1.6.2. ml2_conf.ini ファイル	69
6.2. プロバイダーネットワークを使用する模範的なインストールシナリオ	70
6.2.1. 物理トポロジー	70
6.2.2. 物理ネットワーク環境のプランニング	70
6.2.3. NIC の接続性のプランニング	71
6.2.4. ネットワーク、VLAN、IP のプランニング	71
6.2.5. このシナリオで使用する OpenDaylight の設定ファイル	73
6.2.5.1. extra_env.yaml ファイル	73
6.2.5.2. undercloud.conf ファイル	73
6.2.5.3. network-environment.yaml ファイル	74
6.2.5.4. controller.yaml ファイル	76
6.2.5.5. compute.yaml ファイル	79
6.2.6. このシナリオで使用する Red Hat OpenStack Platform director の設定ファイル	82
6.2.6.1. neutron.conf ファイル	82
6.2.6.2. ml2_conf.ini ファイル	82
第7章 OPENDAYLIGHT での高可用性とクラスタリング	84
7.1. 高可用性とクラスタリング向けの OPENDAYLIGHT の構成	84
7.2. クラスターの動作	85

7.3. クラスターの要件	85
7.4. OPEN VSWITCH の設定	85
7.5. クラスターのモニタリング	85
7.5.1. Jolokia のモニタリング	85
7.6. OPENDAYLIGHT のポートについての理解	86
7.7. OPENDAYLIGHT のフローについての理解	87
7.8. NEUTRON DHCP エージェント HA	88
7.9. NEUTRON メタデータエージェント HA	88
第8章 RED HAT OPENSTACK PLATFORM および OPENDAYLIGHT に関する参考資料	89

前書き

本書では、OpenDaylight のソフトウェア定義ネットワーク (SDN) コントローラーを使用するように Red Hat OpenStack Platform 13 をデプロイする方法について説明します。OpenDaylight コントローラーは、neutron **ML2/OVS** プラグインと、その **L2** エージェントおよび **L3** エージェントと互換性があり、簡単に置き換えることができます。OpenDaylight は、Red Hat OpenStack 環境内でネットワークの仮想化を提供します。

第1章 概要

1.1. OPENDAYLIGHT とは

OpenDaylight プラットフォームは、Java で書かれたプログラミング可能な SDN コントローラーで、OpenStack 環境のネットワーク仮想化に使用することができます。コントローラーのアーキテクチャーは、ノースバウンドとサウスバウンドの別々のインターフェースで構成されます。OpenStack の統合の目的においては、メインのノースバウンドインターフェースは [NeutronNorthbound](#) プロジェクトを使用します。これは、OpenStack Networking サービス (neutron) と通信します。サウスバウンドの OpenDaylight プロジェクト、**OVSDB**、および **OpenFlow** プラグインは **Open vSwitch (OVS)** コントロールおよびデータプレーンと通信します。neutron の設定をネットワーク仮想化に変換するメインの OpenDaylight プロジェクトは、[NetVirt](#) プロジェクトです。

1.2. OPENSTACK での OPENDAYLIGHT の機能

1.2.1. デフォルトの neutron アーキテクチャー

neutron のリファレンスアーキテクチャーでは、一式のエージェントを使用して OpenStack 内のネットワークを管理します。これらのエージェントは、neutron に異なるプラグインとして提供されます。コアプラグインは、**レイヤー 2** のオーバーレイテクノロジーとデータプレーンの種別の管理に使用されます。サービスプラグインは、**レイヤー 3** または OSI モデルのより上位の層でのネットワーク操作 (例: ファイアウォール、DHCP、ルーティング、NAT) の管理に使用されます。

デフォルトでは、Red Hat OpenStack Platform は Modular Layer 2 (**ML2**) のコアプラグインを OVS メカニズムドライバーと共に使用します。これは、各コンピューターノードとコントローラーノードで OVS を設定するためのエージェントを提供します。サービスプラグイン、DHCP エージェント、メタデータエージェントは、**L3** エージェントと共にコントローラー上で実行されます。

1.2.2. OpenDaylight をベースとするネットワークアーキテクチャー

OpenDaylight は、**networking-odl** と呼ばれる独自のドライバーを提供することにより、**ML2** コアプラグインと統合します。これにより、全ノードで OVS エージェントを使用する必要がなくなります。OpenDaylight は、個別のノードにエージェントを使用せずに、環境全体にわたる各 OVS インスタンスを直接プログラミングすることができます。**レイヤー 3** のサービスの場合は、neutron が OpenDaylight **L3** プラグインを使用するように設定されます。この方法を使用すると、データプレーンを直接プログラミングして、OpenDaylight が分散仮想ルーティング機能进行处理できるため、ルーティングやネットワークアドレス変換 (NAT) を処理する複数のノード上のエージェントの数が削減されます。neutron DHCP およびメタデータエージェントは引き続き、DHCP とメタデータ (cloud-init) の要求の管理に使用されます。



注記

OpenDaylight は DHCP サービスを提供できますが、現行の Red Hat OpenStack Platform director のアーキテクチャーをデプロイする場合には、neutron DHCP エージェントを使用すると高可用性 (HA) と仮想マシン (VM) インスタンスのメタデータ (**cloud-init**) のサポートが提供されるため、Red Hat では DHCP サービスの機能は OpenDaylight に依存せずに neutron DHCP エージェントをデプロイすることを推奨しています。

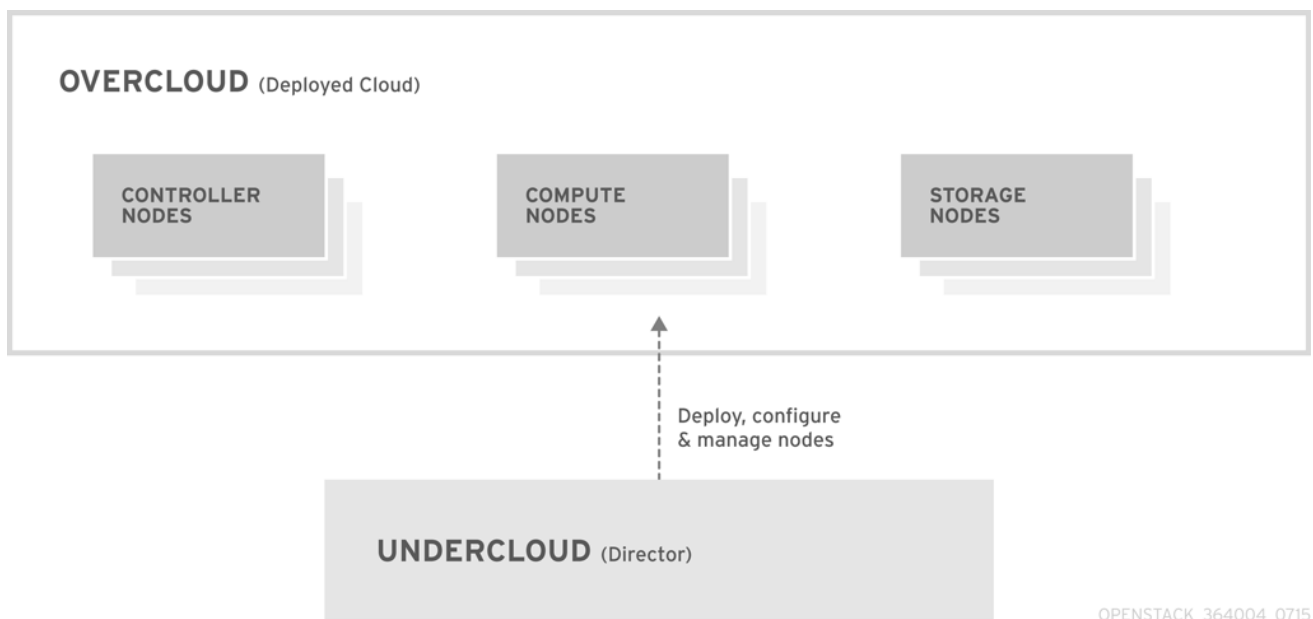
1.3. RED HAT OPENSTACK PLATFORM DIRECTOR およびその設計について

Red Hat OpenStack Platform director は 完全な OpenStack 環境をインストールおよび管理するためのツールセットです。これは、主に OpenStack の [TripleO](#) (OpenStack-On-OpenStack) プロジェクトをベースとしています。

このプロジェクトは OpenStack コンポーネントを使用して、完全に機能する OpenStack 環境をインストールします。また、OpenStack ノードとして稼働するベアメタルシステムのプロビジョニングと制御を行う新たな OpenStack コンポーネントも含まれています。この方法で、リーンで、かつ堅牢性の高い、完全な Red Hat OpenStack Platform 環境をインストールすることができます。

Red Hat OpenStack Platform director では、**アンダークラウド** と **オーバークラウド** の 2 つの主要な概念を採用しています。アンダークラウドは、オーバークラウドのインストールと設定を行います。Red Hat OpenStack Platform director のアーキテクチャーに関する詳しい情報は、『[director のインストールと使用方法](#)』を参照してください。

図1.1 Red Hat OpenStack Platform director: アンダークラウドとオーバークラウド



1.3.1. Red Hat OpenStack Platform director と OpenDaylight

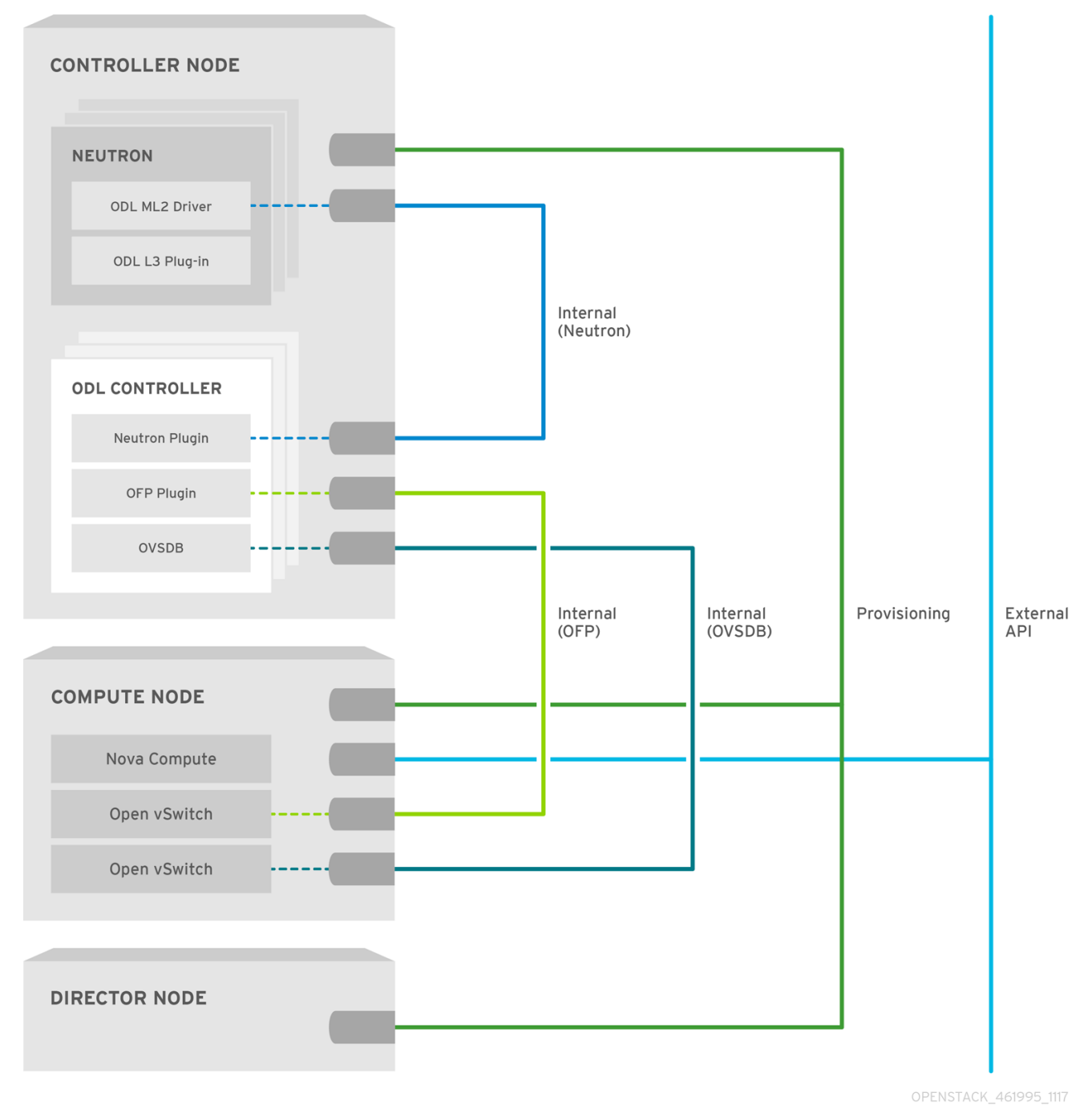
Red Hat OpenStack Platform director では、コンポーザブルサービスとカスタムロールの概念が導入されています。これにより、必要な場合にロールごとに追加/有効化することができる孤立したリソースが形成されます。カスタムロールにより、ユーザーはデフォルトの Controller / Compute ロールに依存しない、独自のロールを作成することができます。ユーザーは、デプロイする OpenStack サービスと、それらのサービスをホストするノードを選択できます。

OpenDaylight を director に統合するために、2 つのサービスが追加されました。

- OpenDaylight **SDN** コントローラーを実行するための **OpenDaylightApi** サービス
- 各ノードで OVS を設定して OpenDaylight と適切に通信するための **OpenDaylightOvs** サービス

デフォルトでは、**OpenDaylightApi** サービスは、コントローラーロール上で実行され、**OpenDaylightOvs** サービスはコントローラーロールとコンピュートロールで実行されます。OpenDaylight は、**OpenDaylightApi** サービスのインスタンスをスケーリングすることによって、**高可用性 (HA)** を提供します。デフォルトでは、コントローラーを 3 つ以上にスケーリングすると、**HA** が自動的に有効化されます。OpenDaylight の **HA** アーキテクチャーに関する詳しい情報は、『[OpenDaylight での高可用性とクラスタリング](#)』を参照してください。

図1.2 OpenDaylight and OpenStack: ベースアーキテクチャー



1.3.2. Red Hat OpenStack Platform director でのネットワーク分離

Red Hat OpenStack Platform director は、個別のサービスを特定の事前定義済みのネットワーク種別に設定することができます。このネットワークトラフィック種別には以下が含まれます。

IPMI	ノードの電源管理ネットワーク。アンダークラウドをインストールする前に、このネットワークを設定する必要があります。
------	--

Provisioning (ctlplane)	director はこのネットワークトラフィック種別を使用して、 DHCP および PXE ブートで新規ノードをデプロイし、オーバークラウドのベアメタルサーバー上で OpenStack Platform のインストールをオーケストレーションします。ネットワークは、アンダークラウドのインストール前に設定する必要があります。または、ironic でオペレーティングシステムのイメージを直接デプロイすることができます。その場合には、 PXE ブートは必要ではありません。
Internal API (internal_api)	Internal API ネットワークは、API 通信を使用した OpenStack サービス間の通信、RPC メッセージ、データベース通信やロードバランサーの背後の内部通信に使用されます。
Tenant (tenant)	neutron は VLAN (各 テナントネットワークがネットワーク VLAN) または オーバーレイトンネルを使用して各テナントに独自のネットワークを提供します。ネットワークは、各テナントネットワーク内で分離されます。トンネリングを使用する場合には、競合は発生することなく、複数のテナントネットワークで同じ IP アドレス範囲を使用することができます。



注記

Generic Routing Encapsulation (GRE) および Virtual eXtensible Local Area Network (VXLAN) は両方ともコードベースで利用可能ですが、OpenDaylight で推奨されるトンネリングプロトコルは **VXLAN** で、**VXLAN** の定義は [RFC 7348](#) に記載されています。本書では、これ以降、トンネリングが使用される場合にはいずれも **VXLAN** を中心とします。

Storage (storage)	Block Storage、NFS、iSCSI など。パフォーマンスを最適化するには、完全に別のスイッチファブリックに分離するのが理想的でしょう。
Storage Management (storage_mgmt)	OpenStack Object Storage (swift) は、このネットワークを使用して、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します。プロキシサービスは、ユーザー要求と背後にあるストレージ層の間の仲介インターフェースとして機能します。プロキシは、受信要求を受け取り、必要なレプリカの位置を特定して要求データを取得します。 Ceph を使用するサービスは、 Ceph と直接対話せずにフロントエンドのサービスを使用するため、ストレージ管理ネットワーク経由で接続を確立します。 RBD ドライバーは例外で、このトラフィックは直接 Ceph に接続する点に注意してください。
External/Public API	この API は、グラフィカルシステム管理用の OpenStack Dashboard (Horizon)、OpenStack サービス用のパブリック API をホストして、インスタンスへの受信トラフィック向けに SNAT を実行します。外部ネットワークがプライベート IP アドレスを使用する場合には (RFC-1918 に準拠)、インターネットからのトラフィックに対して、さらに NAT を実行する必要があります。
Floating IP	テナントネットワーク内のインスタンスに割り当てられた Floating IP アドレスと Fixed IP アドレスとの間の 1 対 1 の IPv4 アドレスマッピングを使用して、受信トラフィックがインスタンスに到達できるようにします。外部と Floating IP ネットワークは、別々に維持管理するのではなく、組み合わせるのが一般的な設定です。

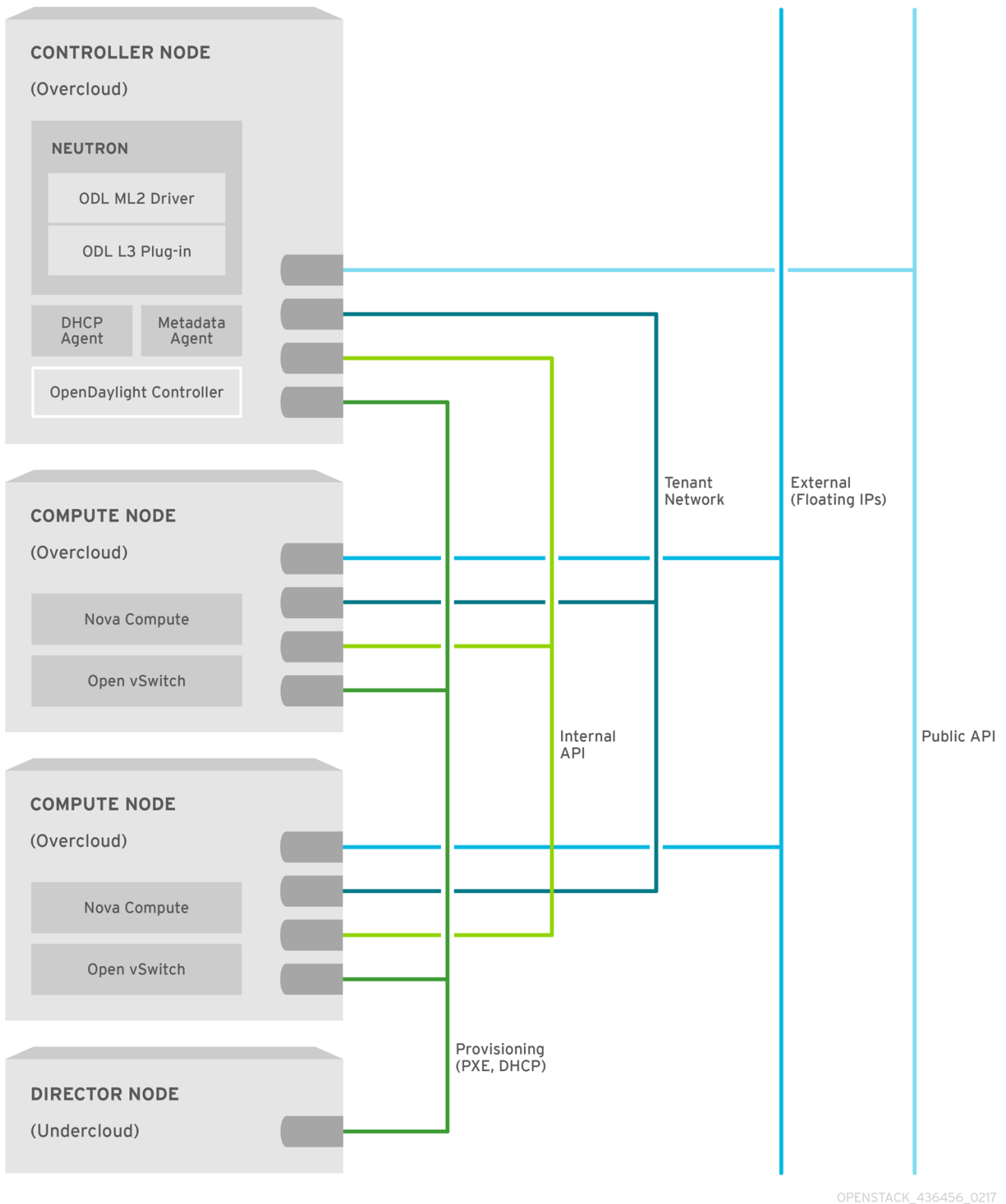
Management	SSH アクセス、DNS トラフィック、NTP トラフィックなどのシステム管理機能を提供します。このネットワークは、コントローラー以外のノード用のゲートウェイとしても機能します。
------------	---

一般的な Red Hat OpenStack Platform のシステム環境では通常、ネットワーク種別の数は物理ネットワークのリンク数を超えます。全ネットワークを正しいホストに接続するには、オーバークラウドは **802.1q VLAN** タグ付けを使用して、1 つのインターフェースに複数のネットワークを提供することができます。ネットワークの多くは、サブネットが分離されていますが、インターネットアクセスまたはインフラストラクチャーにネットワーク接続ができるようにルーティングを提供する **レイヤー 3** のゲートウェイが必要です。

OpenDaylight では、関連するネットワークには **Internal API** および **Tenant** サービスが含まれ、**ServiceNetMap** 内の各ネットワークにマップされます。デフォルトでは **ServiceNetMap** は **OpenDaylightApi** ネットワークを **Internal API** ネットワークにマップします。この設定は、neutron へのノースバウンドトラフィックと、**OVS** へのサウスバウンドトラフィックが **Internal API** ネットワークに分離されることを意味します。

OpenDaylight は分散ルーティングアーキテクチャーを使用するので、各コンピュータノードが **Floating IP** ネットワークに接続されている必要があります。デフォルトでは、Red Hat OpenStack Platform director は **External** ネットワークが OVS ブリッジ **br-ex** にマッピングされた neutron の物理ネットワーク **datacentre** で実行されることを想定します。そのため、コンピュータノードの NIC テンプレートでは、デフォルトの設定に **br-ex** を含める必要があります。

図1.3 OpenDaylight と OpenStack: ネットワーク分離の例



1.3.3. ネットワークとファイアウォールの設定

ファイアウォールが厳しく制約されている場合など、一部のデプロイメントでは、OpenDaylight サービスのトラフィックを有効にするためにファイアウォールを手動で設定する必要がある場合があります。

デフォルトでは、OpenDaylight のノースバウンドは **8080** ポートを使用します。Red Hat OpenStack Platform director でインストールする場合にも、swift サービスとの競合しないようにするために、**8080** ポートを使用して迅速なサービスと競合しないようにするためには、OpenDaylight のポートは **8081** に

設定されています。サウスバウンドは、Red Hat OpenDaylight のソリューションにおけるサウスバウンドは、OVS インスタンスの通常の接続先となるポート **6640** と **6653** でリッスンするように設定されます。

OpenStack では、通常、各サービスに独自の仮想 IP アドレス (VIP) があり、OpenDaylight も同様に動作します。**HAProxy** は **8081** ポートをパブリックに公開し、OpenStack にすでに存在するプレーンの仮想 IP を制御するように設定されます。VIP とポートは、**ML2** プラグインに提供され、neutron はそのポートを介してすべての通信を行います。OVS インスタンスは、OpenDaylight がサウスバウンド用に実行しているノードの物理 IP に直接接続します。

サービス	プロトコル	デフォルトのポート	ネットワーク
OpenStack Neutron API	TCP	9696	Internal API
OpenStack Neutron API (SSL)	TCP	13696	Internal API
OpenDaylight ノースバウンド	TCP	8081	Internal API
OpenDaylight サウスバウンド: OVSDDB	TCP	6640	Internal API
OpenDaylight サウスバウンド: OpenFlow	TCP	6653	Internal API
OpenDaylight 高可用性	TCP	2550	Internal API
VXLAN	UDP	4789	Tenant

表 1: ネットワークとファイアウォールの設定



注記

この項では、OpenDaylight の統合に関連したサービスとプロトコルを中心とする情報を記載していますが、すべては網羅していません。Red Hat OpenStack で実行するサービスに必要なネットワークポートの全一覧は、[『Firewall Rules for Red Hat OpenStack Platform』](#) ガイドを参照してください。

第2章 OPENDAYLIGHT を実行するための要件

以下の項には、OpenDaylight を統合したオーバークラウドのデプロイメント要件に関する情報を記載します。Red Hat OpenDaylight を正しくインストール/実行するために十分なコンピューターリソースが必要です。以下の情報を参照して、最小の要件を理解してください。

2.1. コンピュートノードの要件

コンピュートノードは、仮想マシンインスタンスが起動した後にそれらを稼働させる役割を果たします。全コンピュートノードが、ハードウェアの仮想化をサポートしている必要があります。また、ホストする仮想マシンインスタンスの要件をサポートするのに十分なメモリとディスク容量も必要です。

プロセッサ	Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビットのプロセッサで Intel VT または AMD-V のハードウェア仮想化拡張機能が有効化されていること。このプロセッサには最小でも 4 つのコアが搭載されていることを推奨しています。
メモリ	最小で 6 GB のメモリ。仮想マシンインスタンスに割り当てるメモリ容量に応じて、追加の RAM をこの要件に加算します。
ディスク領域	最小 40 GB の空きディスク領域
ネットワークインターフェースカード	最小 1 枚の 1 Gbps ネットワークインターフェースカード。ただし、実稼働環境では最低でもネットワークインターフェースカード (NIC) を 2 枚使用することを推奨します。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェース向けの場合には追加のネットワークインターフェースを使用します。NIC に関する詳しい情報は、 「Tested NICs」 を参照してください。
電源管理	各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

2.2. コントローラーノードの要件

コントローラーノードは、Red Hat OpenStack Platform 環境の中核となるサービス (例: Horizon Dashboard、バックエンドのデータベースサーバー、Keystone 認証、高可用性サービスなど) をホストする役割を果たします。

プロセッサ	Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビットのプロセッサ
-------	---

メモリー	<p>最小のメモリー容量は 20 GB です。推奨のメモリー容量は、CPU のコア数により異なります。以下の計算を参考にしてください。</p> <p>コントローラーの最小 RAM の計算: 1 コアあたり 1.5 GB のメモリーを使用します。たとえば、コアが 48 個あるマシンには、72 GB の RAM が必要です。</p> <p>コントローラーの推奨 RAM の計算: 1 コアあたり 3 GB のメモリーを使用します。たとえば、コアが 48 個あるマシンには、144 GB の RAM が必要です。必要なメモリーの算出についての詳しい情報は、Red Hat カスタマーポータルで 「Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers」 を参照してください。</p>
ディスク領域	最小 40 GB の空きディスク領域
ネットワークインターフェースカード	最小 2 枚の 1 Gbps ネットワークインターフェースカード。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェース向けの場合には追加のネットワークインターフェースを使用します。
電源管理	各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

第3章 オーバークラウドへの OPENDAYLIGHT のインストール

本ガイドには、OpenDaylight のインストールを中心とした内容のみを記載しています。OpenDaylight をデプロイする前には、アンダークラウド環境が正常に機能しており、オーバークラウドノードが物理ネットワークに接続されていることを確認する必要があります。

アンダークラウドとオーバークラウドのデプロイに必要な手順を記載した『[director のインストールと使用方法](#)』の「[アンダークラウドのインストール](#)」および「[CLI ツールを使用した基本的なオーバークラウドの設定](#)」を参照してください。

Red Hat OpenStack platform で OpenDaylight をインストールするには、いくつかの方法があります。次の章では、OpenDaylight の最も役立つシナリオとインストールの方法を紹介します。

3.1. デフォルトの設定と設定値のカスタマイズ

OpenDaylight のインストールの推奨される方法は、デフォルトの環境ファイル **neutron-opendaylight.yaml** を使用して、そのファイルをアンダークラウドでデプロイメントコマンドに引数として渡す方法です。これにより、OpenDaylight のデフォルトのインストール環境がデプロイされます。

OpenDaylight インストールと設定のその他のシナリオは、このインストールの方法をベースとしています。デプロイメントコマンドに特定の環境ファイルを指定することによって、さまざまな異なるシナリオで OpenDaylight をデプロイすることができます。

3.1.1. デフォルトの環境ファイルについての理解

デフォルトの環境ファイルは **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/** ディレクトリー内の **neutron-opendaylight.yaml** です。この環境ファイルで OpenDaylight がサポートするサービスを有効化/無効化します。また、この環境ファイルは、director がデプロイメント中に設定する必須のパラメーターを定義します。

以下のファイルは、Docker ベースのデプロイメントに使用することができる **neutron-opendaylight.yaml** ファイルの例です。

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

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

Red Hat OpenStack Platform director は **resource_registry** を使用してデプロイメント用のリソースに対応するリソース定義の yaml ファイルにマッピングします。サービスは、マッピング可能なリソース種別の 1 つです。特定のサービスを無効化する場合によっては、**OS::Heat::None** の値を設定します。デフォルトのファイルでは、**OpenDaylightApi** と **OpenDaylightOvs** のサービスが有効化されますが、neutron エージェントは、OpenDaylight がその機能を継承するため、明示的に無効化されます。

Heat パラメーターは、director を使用したデプロイメントの設定値を設定するために使用されます。デフォルト値を上書きするには、環境ファイルの **parameter_defaults** セクションを使用してください。

この例では、OpenDaylight を有効化するために、**NeutronEnableForceMetadata**、**NeutronMechanismDrivers**、**NeutronServicePlugins** のパラメーターが設定されています。



注記

その他のサービスとそれらの設定オプションは、本書の後半に記載しています。

3.1.2. OpenDaylight API サービスの設定

必要に応じて、**/usr/share/openstack-tripleo-heat-templates/puppet/services/.opendaylight-api.yaml** ファイルのデフォルト値を変更することができます。このファイルの設定は、決して直接上書きしないようにしてください。代わりに、このファイルを複製して、元のファイルをバックアップ対策として保持します。複製したファイルのみを編集し、そのファイルをデプロイメントのコマンドで渡します。



注記

後に指定する環境ファイルのパラメーターにより、前の環境ファイルのパラメーターが上書きされます。パラメーターが誤って上書きされるのを防ぐために、環境ファイルの指定順序には注意を払う必要があります。

3.1.2.1. 設定可能なオプション

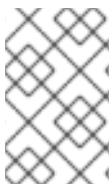
OpenDaylight **API** サービス の設定時には、いくつかのパラメーターを指定することができます。

OpenDaylightPort	ノースバウンドの通信に使用するポートを設定します。デフォルト値は 0 です。このパラメーターは、OSP 13 では非推奨となっています。
OpenDaylightUsername	OpenDaylight のログインユーザー名を設定します。デフォルト値は admin です。
OpenDaylightPassword	OpenDaylight のログインパスワードを設定します。デフォルト値は admin です。
OpenDaylightEnableDHCP	OpenDaylight を有効化して、DHCP サービスとして機能するようにします。デフォルト値は false です。
OpenDaylightFeatures	OpenDaylight で起動する機能のカンマ区切りのリスト。デフォルト値は [odl-netvirt-openstack, odl-jolokia] です。

OpenDaylightConnecti onProtocol	REST のアクセスに使用する L7 プロトコルを設定します。デフォルトは http です。
OpenDaylightManageRe positories	OpenDaylight リポジトリを管理するかどうかを設定します。デフォルトは false です。
OpenDaylightSNATMech anism	OpenDaylight が使用する SNAT メカニズムを設定します。 conntrack と controller から選択することができます。デフォルト値は conntrack です。
OpenDaylightLogMecha nism	OpenDaylight のロギングメカニズムを設定します。 file または console を選択できます。デフォルト値は file です。
OpenDaylightTLSKeyst orePassword	OpenDaylight TLS キーストアのパスワードを設定します。デフォルト値は opendaylight です。パスワードは少なくとも 6 文字でなければなりません。
EnableInternalTLS	内部ネットワークで TLS を有効または無効にします。 true または false を使用できます。デフォルト値は false です。
InternalTLSCAFile	内部ネットワーク内のサービス用に TLS を有効にする場合は、 InternalTLSCAFile パラメーターを使用してデフォルトの CA 証明書を指定する必要があります。デフォルト値は /etc/ipa/ca.crt です。

3.1.3. OpenDaylight OVS サービスの設定

必要に応じて、`/usr/share/openstack-tripleo-heat-templates/puppet/services/opendaylight-ovs.yaml` ファイルのデフォルト値を変更することができます。このファイルの設定は、決して直接上書きしないようにしてください。代わりに、このファイルを複製して、元のファイルをバックアップ対策として保持します。複製したファイルのみを編集し、そのファイルをデプロイメントのコマンドで渡します。



注記

後に指定する環境ファイルのパラメーターにより、前の環境ファイルのパラメーターが上書きされます。パラメーターが誤って上書きされるのを防ぐために、環境ファイルの指定順序には注意を払う必要があります。

3.1.3.1. 設定可能なオプション

OpenDaylight **OVS** サービス の設定時には、いくつかのパラメーターを指定することができます。

OpenDaylightPort	OpenDaylight へのノースバウンドの通信に使用するポートを設定します。デフォルトは 0 です。OVS サービスはノースバウンドで OpenDaylight に対してクエリーを実行し、接続の前に完全に稼動状態であることを確認します。このパラメーターは OSP 13 では非推奨となっています。
-------------------------	---

OpenDaylightConnecti onProtocol	REST アクセスに使用するレイヤー 7 プロトコル。デフォルトは http です。現在 OpenDaylight でサポートされているプロトコルは http のみとなっています。このパラメーターは OSP 13 では非推奨となっています。
OpenDaylightCheckURL	OpenDaylight の検証に使用する URL は、OVS が接続する前に完全に稼働します。デフォルト値は restconf/operational/network-topology:network-topology/topology/netvirt:1 です。
OpenDaylightProvider Mappings	論理ネットワークと物理インターフェースの間のマッピングのコンマ区切りリスト。この設定は、VLAN デプロイメントに必要です。デフォルトは datacentre:br-ex です。
OpenDaylightUsername	OpenDaylight OVS サービスのカスタムユーザー名を設定可能にします。デフォルト値は admin です。
OpenDaylightPassword	OpenDaylight OVS サービスのカスタムパスワードを設定可能にします。デフォルト値は admin です。
HostAllowedNetworkTy pes	この OVS ホストに許可されているネットワーク種別。nova インスタンスとネットワークのスケジュール先となるホストを制限するために、ホストまたはロールによって異なる場合があります。デフォルトは ['local', 'vlan', 'vxlan', 'gre', 'flat'] です。
OvsEnableDpdk	OVS で DPDK を有効化するかどうかを選択します。デフォルト値は false です。
OvsVhostuserMode	vhostuser ポート作成での OVS のモードを指定します。クライアントモードでは、ハイパーバイザーが vhostuser ソケットを作成する役割を果たします。サーバーモードでは、OVS が vhostuser を作成します。デフォルト値は client です。
VhostuserSocketDir	vhostuser ソケットを使用するディレクトリーを指定します。デフォルト値は /var/run/openvswitch です。
OvsHwOffload	OVS ハードウェアオフロードを有効または無効にします。 true または false を使用できます。デフォルト値は false です。このパラメーターは、本リリースではテクノロジープレビューとして提供されています。
EnableInternalTLS	内部ネットワークで TLS を有効または無効にします。 true または false を使用できます。デフォルト値は false です。
InternalTLSCAFile	内部ネットワーク内のサービス用に TLS を有効にする場合は、 InternalTLSCAFile パラメーターを使用してデフォルトの CA 証明書を指定する必要があります。デフォルト値は /etc/ipa/ca.crt です。
ODLUpdateLevel	このパラメーターを使用して、OpenDaylight の更新レベルを指定します。 1 または 2 の値を使用することができます。デフォルト値は 1 です。

VhostuserSocketGroup	vhost-user ソケットディレクトリーのグループを設定します。vhostuser がデフォルトの dpdkvhostuserclient モードの時には、qemu が vhost ソケットを作成します。 VhostuserSocketGroup のデフォルト値は qemu です。
VhostuserSocketUser	vhost-user ソケットディレクトリーのユーザー名を設定します。vhostuser がデフォルトの dpdkvhostuserclient モードの時には、qemu が vhost ソケットを作成します。 VhostuserSocketUser のデフォルト値は qemu です。

3.1.4. OpenDaylight での neutron メタデータサービスの使用

OpenStack Compute サービスにより、特定のアドレス **169.254.169.254** に対して Web 要求を実行することによって、仮想マシンはそれらに関連付けられたメタデータのクエリーを行うことができます。OpenStack Networking は、分離されたネットワークまたは重複する IP アドレスを使用する複数のネットワークから要求が実行された場合でも、そのような **nova-api** に対する要求をプロキシします。

メタデータサービスは、neutron L3 エージェントルーターを使用して、メタデータ要求または DHCP エージェントインスタンスに対応します。レイヤー 3 のルーティングプラグインを有効化して OpenDaylight をデプロイすると、neutron L3 エージェントが無効化されます。このため、テナントネットワークにルーターがある場合でも、メタデータは DHCP インスタンスを通過するように設定する必要があります。この機能は、デフォルトの環境ファイル **neutron-opendaylight.yaml** で有効化されます。無効にするには、**NeutronEnableForceMetadata** を **false** に設定してください。

仮想マシンインスタンスには、**169.254.169.254/32** に DHCP オプション **121** を使用する静的なホストルートがインストールされます。この静的なルートが配置されていると、**169.254.169.254:80** へのメタデータ要求は、DHCP ネットワーク名前空間内のメタデータ名前サーバープロキシに送信されます。名前空間のプロキシは、次に HTTP ヘッダーをインスタンスの IP と共に要求に追加し、Unix ドメインソケットを介して、メタデータエージェントに接続します。メタデータエージェントは、neutron にクエリーを実行し、送信元の IP とネットワーク ID に対応するインスタンス ID を要求し、それを nova メタデータサービスにプロキシします。追加の HTTP ヘッダーは、テナント間の分離を維持し、重複する IP をサポートできるようにするのに必要です。

3.1.5. ネットワーク設定と NIC テンプレートについての理解

Red Hat OpenStack Platform director では、物理的な neutron ネットワークのデータセンターは、デフォルトで **br-ex** という名前の OVS ブリッジにマッピングされます。これは、OpenDaylight の統合と常に同じです。デフォルトの **OpenDaylightProviderMappings** を使用して **flat** または **VLAN_External** ネットワークを作成する予定の場合には、コンピュータード用の NIC テンプレートで OVS br-ex ブリッジを設定する必要があります。レイヤー 3 プラグインは、これらのノードに対して分散ルーティングを使用するので、コントローラーロールの NIC テンプレートでは br-ex を設定する必要はなくなります。

br-ex ブリッジは、ネットワーク分離内の任意のネットワークにマッピングすることができますが、例に示したように、通常は外部ネットワークにマッピングされます。

```
type: ovs_bridge
  name: {get_input: bridge_name}
  use_dhcp: false
  members:
    -
      type: interface
      name: nic3
```

```

        # force the MAC address of the bridge to this interface
        primary: true
    dns_servers: {get_param: DnsServers}
    addresses:
    -
        ip_netmask: {get_param: ExternalIpSubnet}
    routes:
    -
        default: true
        ip_netmask: 0.0.0.0/0
        next_hop: {get_param: ExternalInterfaceDefaultRoute}

```

DPDK では、別の OVS ブリッジを作成する必要があります。これは、通常は **br-phy** という名前で、ovs-dpdk ポートと共に指定します。ブリッジの IP アドレスは、VXLAN オーバーレイネットワークトンネル向けに設定されます。

```

type: ovs_user_bridge
name: br-phy
use_dhcp: false
addresses:
-
    ip_netmask: {get_param: TenantIpSubnet}
members:
-
    type: ovs_dpdk_port
    name: dpdk0
    driver: uio_pci_generic
    members:
    -
        type: interface
        name: nic1
        # force the MAC address of the bridge to
        this interface
        primary: true

```



注記

ネットワーク分離を使用する場合には、コンピュートノード上のこのブリッジには IP アドレスまたはデフォルトのルートを設定する必要はありません。

また、**br-ex** ブリッジを完全に使用することなく、外部ネットワークアクセスを設定することが可能です。このメソッドを使用するには、オーバークラウドのコンピュートノードのインターフェース名を前もって確認しておく必要があります。たとえば、コンピュートノード上の 3 番目のインターフェースの確定的な名前が **eth3** の場合には、コンピュートノード用の NIC テンプレートでインターフェースを指定するのにその名前を使用することができます。

```

-
    type: interface
    name: eth3
    use_dhcp: false

```

3.2. OPENDAYLIGHT の基本インストール

本項では、標準の環境ファイルを使用して OpenDaylight をデプロイする方法を説明します。

3.2.1. オーバークラウド用の OpenDaylight 環境ファイルの準備

作業を開始する前に

- アンダークラウドをインストールします。詳しくは、[「アンダークラウドのインストール」](#)を参照してください。
- オプションとして、オーバークラウドと OpenDaylight のインストール中に使用するコンテナイメージを備えたローカルレジストリーを作成します。詳しくは、[『director のインストールと使用方法』ガイドの「コンテナイメージのソースの設定」](#)を参照してください。

手順

1. アンダークラウドにログインして、admin の認証情報を読み込みます。

```
$ source ~/stackrc
```

2. OpenStack および OpenDaylight のインストールに必要な Docker コンテナイメージへの参照が含まれた Docker レジストリーファイル **odl-images.yaml** を作成します。

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

オーバークラウドをデプロイするための環境の作成が正常に完了し、[「OpenDaylight を実装するオーバークラウドのインストール」](#)に記載のインストールを開始する準備が整いました。

詳細情報

openstack overcloud image prepare コマンドは、オーバークラウドと OpenDaylight のインストール用のコンテナイメージ環境ファイルを準備します。このコマンドでは、以下のオプションを使用します。

-e

OpenDaylight、OVS など、その環境に必要な特定のコンテナイメージを追加するためのサービス環境ファイルを指定します。

--env-file

インストールに使用されるコンテナイメージの一覧を記載した新規コンテナイメージの環境ファイルを作成します。

--pull-source

Docker コンテナレジストリーの場所を設定します。

--namespace

Docker コンテナのバージョンを設定します。

--prefix

イメージ名にプレフィックスを追加します。

--suffix

イメージ名にサフィックスを追加します。

--tag

イメージのリリースを定義します。

3.2.2. OpenDaylight を実装するオーバークラウドのインストール

作業を開始する前に

- 「[オーバークラウド用の OpenDaylight 環境ファイルの準備](#)」の手順に従って、デプロイメントに必要な環境ファイルを作成します。

手順

1. アンダークラウドにログインして、admin の認証情報を読み込みます。

```
$ source ~/stackrc
```

2. 事前に作成した環境ファイルを使用してオーバークラウドをデプロイします。

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



注記

デプロイのコマンドで先に指定する環境ファイルは、そのコマンドで後に指定する環境ファイルによって上書きされます。指定する環境ファイルの順序に注意を払って、パラメーターが誤って上書きされないようにする必要があります。

ヒント

変更するパラメーターのみの設定する最小限の環境ファイルを作成して、デフォルトの環境ファイルと組み合わせることにより、一部のパラメーターを上書きすることができます。

詳細情報

本手順の **openstack overcloud deploy** コマンドでは、以下のオプションを使用しています。

--templates

Heat テンプレートのディレクトリーへのパスを定義します。

-e

環境ファイルを指定します。

3.3. カスタムロールでの **OPENDAYLIGHT** のインストール

カスタムロールで OpenDaylight をインストールすると、**OpenDaylightApi** サービスが分離されて、コントローラーノードとは異なる、指定の OpenDaylight ノードで実行されます。

OpenDaylight 用のカスタムロールを使用する場合には、ノードのレイアウトと機能の設定が含まれたロールファイルを作成する必要があります。

3.3.1. デフォルトロールをベースとするロールファイルのカスタマイズ

ユーザー定義のロール一覧を使用して、OpenStack をデプロイすることができます。各ロールは、ユーザー定義のサービス一覧を実行します。ロールとは、個別のサービスまたは設定が含まれるノードのグループです。たとえば、**nova API** サービスが含まれる **Controller** ロールを作成することができます。ロールのサンプルは、**openstack-tripleo-heat-templates** で参照することができます。

これらのロールを使用して、オーバークラウドノードに必要なロールが記載された **roles_data.yaml** ファイルを作成します。また、ディレクトリー内に個別のファイルを作成し、それらを使用して新しい **roles_data.yaml** を生成することによって、カスタムロールを作成することも可能です。

特定の OpenStack ロールのみをインストールするカスタマイズされた環境ファイルを作成するには、以下の手順を完了します。

手順

- admin の認証情報を読み込みます。

```
$ source ~/stackrc
```

- カスタムの **roles_data.yaml** ファイルを生成するのに使用可能なデフォルトのロールの一覧

```
$ openstack overcloud role list
```

- これらのロールをすべて使用する場合には、以下のコマンドを実行して **roles_data.yaml** ファイルを生成します。

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

- ロールファイルをカスタマイズして、一部のロールのみが含まれるようにするには、前のステップで、コマンドにそのロール名を引数として渡すことができます。たとえば、**Controller**、**Compute**、**Telemetry** ロールのみが含まれる **roles_data.yaml** ファイルを作成するには、以下のコマンドを実行します。

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

3.3.2. OpenDaylight 用のカスタムロールの作成

カスタムロールを作成するには、ロールファイルディレクトリーに新しいロールファイルを作成して、新しい **roles_data.yaml** ファイルを生成します。作成するカスタムロールごとに、新しいロールファイルを作成する必要があります。各カスタムロールファイルは、特定の役割のためのデータが含まれている必要があり、カスタムのロールファイル名はロール名と一致する必要があります。

このファイルでは、少なくとも以下のパラメーターを定義する必要があります。

- **Name:** ロール名を定義します。この名前は、必ず空にせず、一意の文字列する必要があります。

```
- Name: Custom_role
```

- **ServicesDefault:** このロールで使用するサービスをリストします。サービスを使用しない場合には、この変数は空のままにすることができます。以下に書式の例を示します。

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

必須のパラメーター以外に、他の設定も定義することができます。

- **CountDefault:** デフォルトのノード数を定義します。**CountDefault:** が空の場合には、デフォルトでゼロに設定されます。

```
CountDefault: 1
```

- **HostnameFormatDefault:** ホスト名の書式文字列を定義します。この値は任意です。

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

- **Description:** ロールについて説明し、情報を追加します。

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

手順

1. デフォルトのロールファイルを新規ディレクトリーにコピーします。元のファイルは、バックアップとして保管してください。

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

2. `~/roles` の **Controller.yaml** ファイルでデフォルトのコントローラーロールを編集して、そのファイルから **OpenDaylightApi** の行を削除し、コントローラーノード上で **OpenDaylightAPI** サービスを無効にします。

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::OpenDaylightApi #<-- Remove this
    - OS::TripleO::Services::OpenDaylightOvs
```

3. `~/roles` ディレクトリーに新しい **OpenDaylight.yaml** ファイルを作成して、OpenDaylight ロールの説明を追加します。

```
- name: OpenDaylight
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
```

```

- OS::Triple0::Services::Docker
- OS::Triple0::Services::Fluentd
- OS::Triple0::Services::Ipsec
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::LoginDefs
- OS::Triple0::Services::MySQLClient
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::ContainersLogrotateCron
- OS::Triple0::Services::Rhsm
- OS::Triple0::Services::RsyslogSidecar
- OS::Triple0::Services::Securetty
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::Tuned
- OS::Triple0::Services::Ptp
- OS::Triple0::Services::OpenDaylightApi

```

4. ファイルを保存します。

5. OpenDaylight をカスタムロールで実装する OpenStack オーバークラウドのデプロイに使用する新規ロールファイルを生成します。

```

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

```

3.3.3. OpenDaylight をカスタムロールで実装するオーバークラウドのインストール

作業を開始する前に

- アンダークラウドをインストールします (「[アンダークラウドのインストール](#)」を参照)。
- オーバークラウドのコンテナイメージへのリンクを記載した環境ファイルを作成します (「[オーバークラウド用の OpenDaylight 環境ファイルの準備](#)」を参照)。
- カスタムロールで OpenDaylight を設定するためのロールファイルを準備します (「[OpenDaylight 用のカスタムロールの作成](#)」を参照)。

手順

1. 必要な OpenDaylight ノードをフレーバーにマッピングする環境ファイルを作成します。必要な OpenDaylight ノード数。OpenDaylight のコンポーザブルロール用のフレーバーを作成して、そのフレーバーで ironic ノードをタグ付けする必要があります。

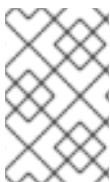
```

- name: odl-composable
  parameter_defaults:
    - OvercloudOpenDaylightFlavor: opendaylight
    - OvercloudOpenDaylightCount: 1
  ServicesDefault:
    - OS::Triple0::Opendaylight::Net::SoftwareConfig:
      /home/stack/templates/nic-configs/odl.yaml

```

2. **-r** 引数を指定してデプロイメントコマンドを実行し、デフォルトのロール定義を上書きします。このオプションは、カスタマイズされたロールが設定されている別の **roles_data.yaml** を使用するようにデプロイメントコマンドに指示します。前のステップで作成した **odl-composable.yaml** 環境ファイルをデプロイメントコマンドに渡します。以下の例では、ironic ノードが合計で 3 つあり、その中の 1 つがカスタムの OpenDaylight ロール用に確保されます。

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



注記

デプロイのコマンドで先に指定する環境ファイルは、そのコマンドで後に指定する環境ファイルによって上書きされます。指定する環境ファイルの順序に注意を払って、パラメーターが誤って上書きされないようにする必要があります。

ヒント

変更するパラメーターのみの設定する最小限の環境ファイルを作成して、デフォルトの環境ファイルと組み合わせることにより、一部のパラメーターを上書きすることができます。

詳細情報

- **-r** オプションは、インストール時にロールの定義を上書きします。

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

- カスタムロールを使用するには、インストール中にそのカスタムロール用に使用する追加の ironic ノードが必要となります。

3.3.4. カスタムロールでの OpenDaylight インストールの検証

作業を開始する前に

- カスタムロールで OpenDaylight を実装するオーバークラウドをインストールします。詳しくは、[「OpenDaylight を実装するオーバークラウドのインストール」](#) を参照してください。

手順

1. 既存のインスタンスを一覧表示します。

```
$ openstack server list
```

2. 結果を確認し、新規 OpenDaylight ロールが 1 つのインスタンスとして専用になっていることを確認します。

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

3.4. SR-IOV 対応の OPENDAYLIGHT のインストール

OpenDaylight は、**Single Root Input/Output Virtualization (SR-IOV)** 対応のコンピュータノードと共にデプロイすることができます。このデプロイメントでは、コンピュータノードは、SR-IOV のみの専用ノードで稼働する必要がある、OVS をベースとする nova インスタンスを混在させてはなりません。単一の OpenDaylight デプロイメント内で OVS と SR-IOV のコンピュータノードの両方をデプロイすることは可能です。

このシナリオでは、カスタムの SR-IOV コンピュートロールを使用してこの種のデプロイメントを行います。

SR-IOV のデプロイメントでは、neutron SR-IOV エージェントを使用して、Virtual Function (VF) を設定する必要があります。これは、デプロイ時に Compute インスタンスに直接渡されて、そこでネットワークポートとして機能します。VF はコンピュータノード上のホスト NIC から派生するので、デプロイメントを開始する前に、ホストのインターフェースに関する情報が必要となります。

3.4.1. SR-IOV コンピュートロールの準備

「[カスタムロールでの OpenDaylight のインストール](#)」に記載したのと同じ方法に従って、SR-IOV コンピュートノード用のカスタムロールを作成し、デフォルトの Compute ロールが OVS ベースの nova インスタンスに対応する一方で、SR-IOV ベースのインスタンスを作成できるようにする必要があります。

作業を開始する前に

- 「[カスタムロールでの OpenDaylight のインストール](#)」の章を参照してください。

手順

1. デフォルトのロールファイルを新規ディレクトリーにコピーします。元のファイルは、バックアップとして保管してください。

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

2. `~/roles` ディレクトリーに新しい **ComputeSriov.yaml** ファイルを作成して、以下のロールの説明を追加します。

```
- name: ComputeSRIOV
  CountDefault: 1
  ServicesDefault:
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::NeutronSriovHostConfig
    - OS::Triple0::Services::NeutronSriovAgent
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::NovaCompute
    - OS::Triple0::Services::NovaLibvirt
    - OS::Triple0::Services::NovaMigrationTarget
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::ComputeNeutronCorePlugin
    - OS::Triple0::Services::Securetty
```

3. ファイルを保存します。

4. **NeutronSriovAgent** および **NeutronSriovHostConfig** のサービスをデフォルトのコンピュートロールから削除して、対応するロールファイルを保存します。

```
- OS::Triple0::Services::NeutronSriovHostConfig
- OS::Triple0::Services::NeutronSriovAgent
```

5. OpenDaylight のコンピュートで SR-IOV をサポートする OpenStack オーバークラウドのデプロイに使用する新規ロールファイルを生成します。

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

3.4.2. SR-IOV エージェントサービスの設定

SR-IOV をサポートする OpenDaylight をデプロイするには、**neutron-opestacklight.yaml** ファイルのデフォルトのパラメーターをオーバーライドする必要があります。`/usr/share/openstack-tripleo-heat-templates` にある標準の SR-IOV 環境ファイルと、**neutron-opestacklight.yaml** 環境ファイルを使用することができます。元のファイルは編集しないのが適切なプラクティスです。代わりに、元の環境ファイルを複製して、その複製したファイル内でパラメーターを変更してください。

また、変更するパラメーターのみを指定する新規環境ファイルを作成して、両方のファイルをデプロイメントに使用することができます。カスタマイズされた OpenDaylight をデプロイするには、デプロイメントコマンドに両方のファイルを渡します。後で指定する環境ファイルが前の設定を上書きするので、これらは正しい順序でデプロイメントコマンドに含める必要があります。ただし順序は、**neutron-opestacklight.yaml** が最初で、**neutron-opestacklight-sriov.yaml** が後になります。

OpenDaylight と SR-IOV をデフォルト設定でデプロイする場合には、Red Hat で提供している **neutron-opestacklight-sriov.yaml** を使用することができます。パラメーターを変更または追加する必要がある場合には、デフォルトの SR-IOV 環境ファイルのコピーを作成して、その新規作成されたファイルを編集してください。

カスタマイズされた **neutron-ovsdaylight-sriov.yaml** ファイルの例を以下に示します。

```
# A Heat environment that can be used to deploy OpenDaylight with SRIOV
resource_registry:
  OS::Triple0::Services::NeutronOvsAgent: OS::Heat::None
  OS::Triple0::Services::ComputeNeutronOvsAgent: OS::Heat::None
  OS::Triple0::Services::ComputeNeutronCorePlugin:
    ../puppet/services/neutron-plugin-ml2.yaml
  OS::Triple0::Services::NeutronCorePlugin: ../puppet/services/neutron-
plugin-ml2-odl.yaml
  OS::Triple0::Services::OpenDaylightApi: ../docker/services/ovsdaylight-
api.yaml
  OS::Triple0::Services::OpenDaylightOvs: ../puppet/services/ovsdaylight-
ovs.yaml
  OS::Triple0::Services::NeutronSriovAgent: ../puppet/services/neutron-
sriov-agent.yaml
  OS::Triple0::Services::NeutronL3Agent: OS::Heat::None

parameter_defaults:
  NeutronEnableForceMetadata: true
  NeutronPluginExtensions: 'port_security'
  NeutronMechanismDrivers: ['sriovnicswitch', 'ovsdaylight_v2']
  NeutronServicePlugins: 'odl-router_v2,trunk'

  # Add PciPassthroughFilter to the scheduler default filters
  #NovaSchedulerDefaultFilters:
  ['RetryFilter', 'AvailabilityZoneFilter', 'RamFilter', 'ComputeFilter', 'Compu
teCapabilitiesFilter',
  'ImagePropertiesFilter', 'ServerGroupAntiAffinityFilter', 'ServerGroupAffini
tyFilter', 'PciPassthroughFilter']
  #NovaSchedulerAvailableFilters:
  ["nova.scheduler.filters.all_filters", "nova.scheduler.filters.pci_passtro
ugh_filter.PciPassthroughFilter"]

  #NeutronPhysicalDevMappings: "datacentre:ens20f2"

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

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

詳細情報

neutron-ovsdaylight-sriov.yaml ファイルでは、以下のオプションを設定することができます。以下の表には、各オプションについての説明と、SR-IOV 機能を有効化するために必要な設定を記載します。

NovaSchedulerDefaultFilters	SR-IOV 用の PCI パススルーを使用できるようにします。このパラメーターは、環境ファイル内でコメント解除して、 PciPassthroughFilter を追加する必要があります。
------------------------------------	---

NovaSchedulerAvailableFilters	Nova のデフォルトフィルターに PCI パススルーフィルターを指定できるようにします。このパラメーターは必須で、 nova.scheduler.filters.all_filters を追加する必要があります。
NeutronPhysicalDevMappings	neutron の論理ネットワークをホストのネットワークインターフェースにマッピングします。neutron が仮想ネットワークを物理ポートにバインドできるようにするには、このパラメーターを指定する必要があります。
NeutronSriovNumVFs	1 つのホストのネットワークインターフェイス用に作成する VF の数。構文: <Interface name>: <number of VFs>
NovaPCIPassthrough	<p>PCI パススルーでの使用を許可する nova の PCI デバイスのホワイトリストを一覧形式で設定します。以下に例を示します。</p> <pre> NovaPCIPassthrough: - vendor_id: "8086" product_id: "154c" address: "0000:05:00.0" physical_network: "datacentre" </pre> <p>特定のハードウェア属性の代わりに単に論理デバイス名を使用することもできます。</p> <pre> NovaPCIPassthrough: - devname: "ens20f2" physical_network: "datacentre" </pre>

3.4.3. SR-IOV 対応の OpenDaylight のインストール

作業を開始する前に

- アンダークラウドをインストールします (「[アンダークラウドのインストール](#)」を参照)。
- オーバークラウドのコンテナイメージへのリンクを記載した環境ファイルを作成します (「[オーバークラウド用の OpenDaylight 環境ファイルの準備](#)」を参照)。
- カスタムロールで SR-IOV 対応の OpenDaylight を設定するためのロールファイルを作成します (「[SR-IOV コンピュートロールの準備](#)」を参照)。

手順

1. `-r` の引数を使用してデプロイメントコマンドを実行し、カスタマイズしたロールファイルと OpenDaylight で SR-IOV 機能を有効化するのに必要な環境ファイルを指定します。

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



注記

デプロイのコマンドで先に指定する環境ファイルは、そのコマンドで後に指定する環境ファイルによって上書きされます。指定する環境ファイルの順序に注意を払って、パラメーターが誤って上書きされないようにする必要があります。

ヒント

変更するパラメーターのみの設定する最小限の環境ファイルを作成して、デフォルトの環境ファイルと組み合わせることにより、一部のパラメーターを上書きすることができます。

詳細情報

- `-r` オプションは、インストール時にロールの定義を上書きします。

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

- カスタムロールを使用するには、インストール中にそのカスタムロール用に使用する追加の ironic ノードが必要となります。

3.5. OVS-DPDK 対応の OPENDAYLIGHT のインストール

OpenDaylight は、director を使用して **Open vSwitch Data Plane Development Kit (DPDK)** アクセラレーションに対応するようにデプロイすることができます。このデプロイメントでは、カーネル空間ではなくユーザー空間でパケットが処理されるために、データプレーンパフォーマンスが向上します。OVS-DPDK 対応のデプロイメントには、潜在的なパフォーマンスを引き出すために、各コンピュータノードのハードウェア物理レイアウトに関する知識が必要です。

特に次の点を考慮すべきです。

- ホスト上のネットワークインターフェースが DPDK をサポートしていること
- コンピュータノードの NUMA ノードのトポロジ (ソケット数、CPU コア、1 ソケットあたりのメモリー容量)
- DPDK NIC PCI バスが各 NUMA ノードに近いこと

- コンピュートノード上で使用可能な RAM 容量
- 『[Network Functions Virtualization Planning and Configuration Guide](#)』を参照してください。

3.5.1. OVS-DPDK デプロイメントファイルの準備

OVS-DPDK をデプロイするには、異なる環境ファイルを使用します。そのファイルは、`/usr/share/openstack-tripleo-heat-templates/environments/services-docker` ディレクトリーにある `neutron-opensdaylight.yaml` 環境ファイルで設定されている一部のパラメーターを上書きします。元の環境ファイルは変更しないでください。代わりに、必要なパラメーターが含まれた新しい環境ファイルを作成します (例: `neutron-opensdaylight-dpdk.yaml`)。

OVS-DPDK を実装する OpenDaylight をデフォルトの設定でデプロイするには、`/usr/share/openstack-tripleo-heat-templates/environments/services-docker` ディレクトリーにあるデフォルトの `neutron-opensdaylight-dpdk.yaml` 環境ファイルを使用します。

デフォルトのファイルには、以下の値が含まれています。

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

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

  ComputeOvsDpdkParameters:
    OvsEnableDpdk: True

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

    ## Attempting to deploy DPDK without appropriate values for the below
parameters may lead to unstable deployments
    ## due to CPU contention of DPDK PMD threads.
    ## It is highly recommended to to enable isolcpus (via KernelArgs) on
compute overcloud nodes and set the following parameters:
    #OvsDpdkSocketMemory: "" # Sets the amount of hugepage memory to
assign per NUMA node.
# It is recommended to use the socket
```

```

closest to the PCIe slot used for the
                                # desired DPDK NIC. Format should be
comma separated per socket string such as:
                                # "<socket 0 mem MB>,<socket 1 mem
MB>", for example: "1024,0".
    #OvsDpdkDriverType: "vfio-pci" # Ensure the Overcloud NIC to be used
for DPDK supports this UIO/PMD driver.
    #OvsPmdCoreList: ""           # List or range of CPU cores for PMD
threads to be pinned to. Note, NIC
                                # location to cores on socket, number
of hyper-threaded logical cores, and
                                # desired number of PMD threads can
all play a role in configuring this setting.
                                # These cores should be on the same
socket where OvsDpdkSocketMemory is assigned.
                                # If using hyperthreading then
specify both logical cores that would equal the
                                # physical core. Also, specifying
more than one core will trigger multiple PMD
                                # threads to be spawned, which may
improve dataplane performance.
    #NovaVcpuPinSet: ""          # Cores to pin Nova instances to. For
maximum performance, select cores
                                # on the same NUMA node(s) selected
for previous settings.

```

3.5.2. OVS-DPDK デプロイメントの設定

`neutron-opendaylight-dpdk.yaml` の値を変更して、OVS-DPDK サービスを設定することができます。

TunedProfileName	IRQ のピンングを有効化して、OVS-DPDK で使用する CPU コアから分離します。デフォルトプロファイル: cpu-partitioning
IsolCpusList	CPU コアの一覧を指定し、カーネルスケジューラーがそれらのコアを使用しないようにして、代わりに OVS-DPDK に割り当てて専用にすることができます。書式は、個々のコアのコンマ区切りリストまたは範囲で指定します (例: 1,2,3,4-8,10-12)。
KernelArgs	<p>ブート時にカーネルに渡す引数をリストします。OVS-DPDK の場合は、IOMMU と Hugepages を有効にする必要があります。以下に例を示します。</p> <pre>---intel_iommu=on iommu=pt default_hugepagesz=1GB hugepagesz=1G hugepages=60 ---</pre> <p>指定されている RAM の容量はヒュージページ用の 60 GB であることに注意してください。この値を設定する場合には、コンピュータノードで利用可能な RAM 容量を考慮することが重要です。</p>

OvsDpdkSocketMemory	<p>各 NUMA ノードに割り当てるヒュージページメモリーの容量を (MB 単位で) 指定します。パフォーマンスを最大限にするには、DPDK NIC に最も近いソケットにメモリーを割り当てます。ソケットあたりのメモリーをリストする形式は以下のようになります。</p> <p>---- "<socket 0 mem MB>,<socket 1 mem MB>" ----</p> <p>例: "1024,0"</p>
OvsDpdkDriverType	<p>PMD スレッドで使用する UIO ドライバー種別を指定します。DPDK NIC は指定したドライバーをサポートしている必要があります。Red Hat OpenStack Platform のデプロイメントでは、vfio-pci のドライバー種別がサポートされています。Red Hat OpenStack Platform のデプロイメントでは、uio_pci_generic や igb_uio などの UIO ドライバーはサポートされていません。</p>
OvsPmdCoreList	<p>PMD スレッドのピンング先となる個々のコアまたは範囲をリストします。ここで指定するコアは、OvsDpdkSocketMemory の設定でメモリーが割り当てられているのと同じ NUMA ノード上にある必要があります。ハイパースレッディングが使用される場合には、ホスト上の物理コアを構成する論理コアを指定する必要があります。</p>
OvsDpdkMemoryChannels	<p>ソケットあたりのメモリーチャンネルの数を指定します。</p>
NovaVcpuPinSet	<p>libvirt で nova インスタンスをピンングするコア。パフォーマンスを最適にするには、OVS PMD コアがピンングされているのと同じソケット上のコアを使用してください。</p>

3.5.3. OVS-DPDK を実装する OpenDaylight のインストール

作業を開始する前に

- アンダークラウドをインストールします (「[アンダークラウドのインストール](#)」を参照)。
- オーバークラウドのコンテナイメージへのリンクを記載した環境ファイルを作成します (「[オーバークラウド用の OpenDaylight 環境ファイルの準備](#)」を参照)。
- カスタムロールで SR-IOV 対応の OpenDaylight を設定するためのロールファイルを作成します (「[OVS-DPDK デプロイメントファイルの準備](#)」を参照)。

手順

1. OpenDaylight で DPDK 機能を有効化するのに必要な環境ファイルを使用してデプロイメントコマンドを実行します。

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

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



注記

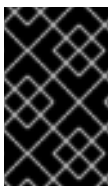
デプロイのコマンドで先に指定する環境ファイルは、そのコマンドで後に指定する環境ファイルによって上書きされます。指定する環境ファイルの順序に注意を払って、パラメーターが誤って上書きされないようにする必要があります。

ヒント

変更するパラメーターのみの設定する最小限の環境ファイルを作成して、デフォルトの環境ファイルと組み合わせることにより、一部のパラメーターを上書きすることができます。

3.5.4. 例: ODL と VXLAN トンネリングを使用する OVS-DPDK の設定

このセクションには、ODL と VXLAN トンネリングを使用する OVS-DPDK の設定例を記載しています。



重要

OVS-DPDK 用の OpenStack を最適化するには、**network-environment.yaml** ファイルに設定する OVS-DPDK パラメーターの最適な値を判断する必要があります。詳しくは、[「Deriving DPDK parameters with workflows」](#) を参照してください。

3.5.4.1. ComputeOvsDpdk コンポーザブルロールの生成

ComputeOvsDpdk ロール用の **roles_data.yaml** を生成します。

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

3.5.4.2. CPU アフィニティー用の tuned 設定

1. **tuned** を設定して CPU フィニティーを有効にします。

```
heat_template_version: 2014-10-16

description: >
  Example extra config for post-deployment

parameters:
  servers:
    type: json
  DeployIdentifier:
    type: string
    default: ''

resources:
```

```

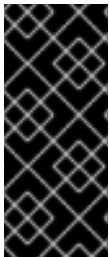
ExtraDeployments:
  type: OS::Heat::StructuredDeployments
  properties:
    servers: {get_param: servers}
    config: {get_resource: ExtraConfig}
    actions: ['CREATE', 'UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}

ExtraConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config: |
      #!/bin/bash
      set -x
      function tuned_service_dependency() {
        tuned_service=/usr/lib/systemd/system/tuned.service
        grep -q "network.target" $tuned_service
        if [ "$?" -eq 0 ]; then
          sed -i '/After=.*s/network.target//g'
$tuned_service
        fi
        grep -q "Before=.*network.target" $tuned_service
        if [ ! "$?" -eq 0 ]; then
          grep -q "Before=.*" $tuned_service
          if [ "$?" -eq 0 ]; then
            sed -i 's/^\(Before=.*\)\/\1 network.target
openvswitch.service/g' $tuned_service
          else
            sed -i '/After/i Before=network.target
openvswitch.service' $tuned_service
          fi
        fi
      }

      if hiera -c /etc/puppet/hiera.yaml service_names | grep -q
neutron_ovs_dpdk_agent; then
        tuned_service_dependency
      fi

```

3.5.4.3. OVS-DPDK パラメーターの設定



重要

OVS-DPDK 向けの OpenStack ネットワークを最適化するには、**network-environment.yaml** で設定する OVS-DPDK パラメーターの最適な値を判断する必要があります。詳しくは、https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_planning_and_configuration#proc_derive-dpdk を参照してください。

1. **resource_registry** の下に OVS-DPDK 用のカスタムリソースを追加します。


```
resource_registry:
  # Specify the relative/absolute path to the config files you
  # want to use for override the default.
  OS::Triple0::ComputeOvsDpdk::Net::SoftwareConfig: nic-
  configs/computeovsdpdk.yaml
  OS::Triple0::Controller::Net::SoftwareConfig: nic-
  configs/controller.yaml
  OS::Triple0::NodeExtraConfigPost: post-install.yaml
```

2. **parameter_defaults** の下で、トンネルの種別とテナントの種別を **vxlan** に設定します。

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

3. **parameters_defaults** の下にブリッジのマッピングを設定します。

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

4. **parameter_defaults** の下に、**ComputeOvsDpdk** ロール向けのロール固有パラメーターを設定します。

```
#####
# OVS DPDK configuration #
#####
ComputeOvsDpdkParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32
  iommu=pt intel_iommu=on isolcpus=2-19,22-39"
  TunedProfileName: "cpu-partitioning"
  IsolCpusList: "2-19,22-39"
  NovaVcpuPinSet: ['4-19,24-39']
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "4096,4096"
  OvsDpdkMemoryChannels: "4"
  OvsDpdkCoreList: "0,20,1,21"
  OvsPmdCoreList: "2,22,3,23"
  OvsEnableDpdk: true
```



注記

ゲストインスタンス作成の失敗を避けるために、DPDK PMD 用の DPDK NIC の有無にかかわらず、各 NUMA ノード上で (シブリングスレッドを使用して) 少なくとも 1 つの CPU を割り当てる必要があります。



注記

本手順に示したとおり、これらのヒュージページは仮想マシンと、**OvsDpdkSocketMemory** パラメーターを使用する OVS-DPDK によって消費されます。仮想マシンが利用可能なヒュージページの数、**boot** パラメーターから **OvsDpdkSocketMemory** を減算した値です。

DPDK インスタンスに関連付けるフレーバーに **hw:mem_page_size=1GB** を追加する必要もあります。



注記

OvsDPDKCoreList と **OvsDpdkMemoryChannels** は、この手順では **必須** の設定です。適切な値を設定せずに DPDK のデプロイメントを試みると、デプロイメントが失敗したり、不安定なデプロイメントになったりします。

3.5.4.4. コントローラーノードの設定

1. 分離ネットワーク用のコントロールプレーンの Linux ボンディングを作成します。

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

2. この Linux ボンディングに VLAN を割り当てます。

```
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bond_api
```

```

addresses:
- ip_netmask:
  get_param: StorageIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageMgmtNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
    get_param: StorageMgmtIpSubnet

- type: vlan
  vlan_id:
    get_param: ExternalNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
    get_param: ExternalIpSubnet
  routes:
  - default: true
    next_hop:
      get_param: ExternalInterfaceDefaultRoute

```

3. クラウドネットワークへの Floating IPアドレスにアクセスするための OVS ブリッジを作成します。

```

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

```

3.5.4.5. DPDK インターフェースのコンピュータノードの設定

デフォルトの **compute.yaml** ファイルから **compute-ovs-dpdk.yaml** ファイルを作成し、次のように変更します。

1. 分離ネットワーク用のコントロールプレーンの Linux ボンディングを作成します。

```

- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:

```

```

    get_param: DnsServers
members:
- type: interface
  name: nic7
  primary: true
- type: interface
  name: nic8

```

2. この Linux ボンディングに VLAN を割り当てます。

```

- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: StorageIpSubnet

```

3. コントローラーにリンクする DPDK ポートを備えたブリッジを設定します。

```

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_:
            get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          members:
            - type: interface
              name: nic3
        - type: ovs_dpdk_port
          name: dpdk1
          members:
            - type: interface
              name: nic4

```



注記

複数の DPDK デバイスを含めるには、追加する DPDK デバイスごとに **type** のコードセクションを繰り返します。



注記

OVS-DPDK を使用する場合には、同じコンピュータノード上の全ブリッジが **ovs_user_bridge** の種別である必要があります。同じノード上で **ovs_bridge** と **ovs_user_bridge** が混在する構成は、director では受け入れ可能ですが、Red Hat OpenStack Platform ではサポートされていません。

3.5.4.6. オーバークラウドのデプロイ

overcloud_deploy.sh スクリプトを実行してオーバークラウドをデプロイします。

```
#!/bin/bash

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

3.6. L2GW 対応の OPENDAYLIGHT のインストール

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト目的のみでご利用いただく機能で、実稼働環境にデプロイすべきではありません。テクノロジープレビューについての詳しい情報は「[対象範囲の詳細](#)」を参照してください。

レイヤー 2 ゲートウェイサービスにより、テナントの仮想ネットワークを物理ネットワークにブリッジングすることができます。この統合により、ルーティングされたレイヤー 3 の接続ではなく、レイヤー 2 のネットワーク接続で物理サーバー上のリソースにアクセスすることができます。これは、L3 や Floating IP を経由する代わりにレイヤー 2 のブロードキャストドメインを拡張することを意味します。

3.6.1. L2GW デプロイメントファイルの準備

L2GW 対応の OpenDaylight をデプロイするには、**/usr/share/openstack-tripleo-heat-**

templates/environments ディレクトリー内の **neutron-l2gw-opensdaylight.yaml** ファイルを使用します。このファイル内の設定を変更する必要がある場合には、既存のファイルを変更する代わりに、この環境ファイルの必要なパラメーターが含まれたコピーを新規作成してください。

OpenDaylight と L2GW をデフォルトの設定でデプロイする場合には、**/usr/share/openstack-tripleo-heat-templates/environments/services-docker** ディレクトリーの **neutron-l2gw-opensdaylight.yaml** を使用することができます。

デフォルトのファイルには、以下の値が含まれています。

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

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

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

3.6.2. OpenDaylight L2GW デプロイメントの設定

neutron-l2gw-opensdaylight.yaml ファイルの値を変更して、サービスを設定することができます。

NeutronServicePlugins	サービスプラグインのエントリーポイントのコンマ区切りリストは、 neutron.service_plugins 名前空間から読み込まれます。デフォルトは router です。
L2gwServiceProvider	このサービスを提供するのに使用する必要のあるプロバイダーを定義します。デフォルトは L2GW:OpenDaylight:networking_odl.l2gateway.driver.OpenDaylightL2gwDriver:default です。
L2gwServiceDefaultInterfaceName	デフォルトのインターフェースの名前を設定します。

L2gwServiceDefaultDeviceName	デフォルトのデバイスの名前を設定します。
L2gwServiceQuotaL2Gateway	L2 ゲートウェイ用のサービスクォータを指定します。デフォルトは 10 です。
L2gwServicePeriodicMonitoringInterval	L2GW サービスのモニタリング間隔を指定します。

3.6.3. L2GW を実装する OpenDaylight のインストール

作業を開始する前に

- アンダークラウドをインストールします (「[アンダークラウドのインストール](#)」を参照)。
- オーバークラウドのコンテナイメージへのリンクを記載した環境ファイルを作成します (「[オーバークラウド用の OpenDaylight 環境ファイルの準備](#)」を参照)。
- SR-IOV 対応のカスタムロールで OpenDaylight を設定するためのロールファイルを作成します (「[L2GW デプロイメントファイルの準備](#)」を参照)。

手順

1. OpenDaylight で L2GW 機能を有効化するのに必要な環境ファイルを使用してデプロイメントコマンドを実行します。

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



注記

デプロイのコマンドで先に指定する環境ファイルは、そのコマンドで後に指定する環境ファイルによって上書きされます。指定する環境ファイルの順序に注意を払って、パラメーターが誤って上書きされないようにする必要があります。

ヒント

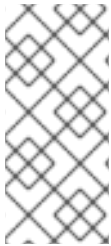
変更するパラメーターのみの設定する最小限の環境ファイルを作成して、デフォルトの環境ファイルと組み合わせることにより、一部のパラメーターを上書きすることができます。

第4章 デプロイメントのテスト

4.1. 基本的なテストの実行

基本的なテストでは、インスタンスがお互いにpingを実行できることを確認します。テストは、**Floating IP SSH** アクセスもチェックします。以下の例では、アンダークラウドからこのテストを実行する方法について説明します。

本手順では、個別のステップを多数実行する必要があるので、便宜を図るために小さいセクションに分けています。ただし、以下の順序に従って全ステップを実行する必要があります。



注記

このステップでは、flat ネットワークを使用して `_External_ network` を作成し、`_VXLAN_` を使用して `_Tenant_ networks` を作成します。デプロイメントに望ましい場合には、`_VLAN External_ networks` と `_VLAN Tenant_ networks` もサポートされています。

4.1.1. テスト用の新規ネットワークの作成

1. オーバークラウドにアクセスするための認証情報を読み込みます。

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

2. オーバークラウドの外部からインスタンスにアクセスするための neutron の外部ネットワークを作成します。

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

3. 新しい外部ネットワーク (前のステップで作成済み) に対応する neutron サブネットを作成します。

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

4. オーバークラウドのインスタンスを作成するのに使用する cirros イメージをダウンロードします。

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

5. オーバークラウドの glance に cirros イメージをアップロードします。

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

6. オーバークラウドインスタンスに使用する **tiny** フレーバーを作成します。

■


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

7. インスタンスをホストするための VXLAN のテナントネットワークを作成します。

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

8. テナントネットワーク (前のステップで作成済み) のサブネットを作成します。

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

9. テナントネットワークの ID を特定して保存します。

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

10. インスタンス **cirros1** を作成して、**net_test** ネットワークと **SSH** セキュリティーグループにアタッチします。

```
$openstack server create --flavor m1.small --image cirros --nic  
net-id=$vlan1 --security-group SSH --key-name RDO_KEY --  
availability-zone nova:overcloud-novacompute-0.localdomain  
net1_host1_vm1 cirros1
```

11. **cirros2** という 2 番目のインスタンスを作成して、そのインスタンスも **net_test** ネットワークと **SSH** セキュリティーグループにアタッチします。

```
$ openstack server create --flavor m1.small --image cirros --nic  
net-id=$vlan1 --security-group SSH --key-name RDO_KEY --  
availability-zone nova:overcloud-novacompute-0.localdomain  
net1_host1_vm1 cirros2
```

4.1.2. ネットワークとテスト環境の設定

1. admin プロジェクトの ID を特定して保存します。

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

2. admin プロジェクトのデフォルトのセキュリティグループを特定して保存します。

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

3. admin のデフォルトセキュリティグループに、ICMP トラフィックの受信を許可するルールを追加します。

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

4. admin のデフォルトセキュリティグループに、ICMP トラフィックの送信を許可するルールを追加します。

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

5. admin のデフォルトセキュリティグループに、SSH トラフィックの受信を許可するルールを追加します。

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

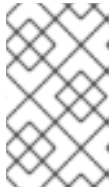
6. admin のデフォルトセキュリティグループにルールを追加して SSH トラフィックの送信を許可します。

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

4.1.3. 接続性のテスト

1. horizon から、インスタンスの **novnc** コンソールにアクセスできるはずです。 **overcloudrc** から取得したパスワードを使用して horizon に **admin** としてログインします。 **cirros** イメージのデフォルトの認証情報はユーザー名が **cirros**、パスワードが **cubswin:)** です。
2. **novnc** コンソールから、DHCP アドレスがインスタンスに割り当てられているかどうかを確認します。

```
$ ip addr show
```



注記

また、アンダークラウドから **nova console-log <instance id>** のコマンドを実行して、インスタンスが DHCP アドレス **nova console-log <instance id>** を受信していることを確認することができます。

3. ステップ 1 と 2 をその他すべてのインスタンスで繰り返します。
4. 1 つのインスタンスから他のインスタンスに ping を試みて、オーバークラウドにおける基本的なテナント ネットワーク接続を検証します。
5. **Floating IP** を使用して他のインスタンスに到達できるかどうかを確認します。

4.1.4. デバイスの作成

1. **cirros1** インスタンスに割り当てる Floating IP を外部ネットワーク上に作成します。

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

2. Floating IP と **cirros1** テナントの IP の間で NAT を処理するルーターを作成します。

```
$ openstack router create test
```

3. ルーターのゲートウェイを外部ネットワークに設定します。

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

4. テナントネットワークに接続するルーターへのインターフェースを追加します。

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

5. ステップ 1 で作成した Floating IP を特定して保存します。

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

6. Floating IP を **cirros1** インスタンスに関連付けます。

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

7. 外部ネットワークにアクセス可能なノードからインスタンスへのログインを試みます。

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

4.2. 高度なテストの実行

OpenDaylight の設定およびデプロイメントのコンポーネントのいくつかは、OpenDaylight をデプロイした後に確認することができます。インストールの特定の部分をテストするには、いくつかの手順に従って操作を実行する必要があります。各手順は別々に記載しています。

この手順は、オーバークラウド ノード上で実行する必要があります。

4.2.1. オーバークラウドノードへの接続

オーバークラウドノードに接続し、それらが正しく動作していることを確認するには、以下の手順を実行します。

手順

1. アンダークラウドにログインします。
2. 以下のコマンドを入力して作業を開始します。

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

3. 全インスタンスを一覧表示します。

```
$ openstack server list
```

4. 必要なインスタンスを選択して、一覧に表示されるインスタンスの IP アドレスを書き留めておきます。
5. 前のステップで取得したリストの IP アドレスを使用してマシンに接続します。

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

6. superuser に切り替えます。

```
$ sudo -i
```

4.2.2. OpenDaylight のテスト

OpenDaylight が正しく動作していることをテストするには、サービスが動作していることと、指定の機能が正しく読み込まれていることを確認する必要があります。

手順

1. OpenDaylight を実行するオーバークラウドノードまたはカスタムロールで実行している OpenDaylight ノードに superuser としてログインします。
2. OpenDaylight コントローラーがすべてのコントローラーノード上で実行されていることを確認します。

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

3. HAProxy がポート 8081 をリッスンするように適切に設定されていることを確認します。

```
# docker exec -it haproxy-bundle-docker-0 grep -A7 opendaylight
/etc/haproxy/haproxy.cfg
listen opendaylight
    bind 172.17.0.10:8081 transparent
    bind 192.168.24.10:8081 transparent
    mode http
    balance source
    server overcloud-controller-0.internalapi.localdomain
172.17.0.22:8081 check fall 5 inter 2000 rise 2
    server overcloud-controller-1.internalapi.localdomain
172.17.0.12:8081 check fall 5 inter 2000 rise 2
    server overcloud-controller-2.internalapi.localdomain
172.17.0.13:8081 check fall 5 inter 2000 rise 2
```

4. HAProxy IPを使用して、karaf のアカウントを接続します。karaf のパスワードは **karaf** です。

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

5. インストール済みの機能を一覧表示します。

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

ステップ 5: この手順で生成されたリストの 3 番目のコラムに **x** がある場合には、その機能は適切にインストールされていることになります。

6. API が稼働中であることを確認します。

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

このAPIエンドポイントは、`/etc/neutron/plugins/ml2/ml2_conf.ini` で設定され、neutron が OpenDaylight と通信するのに使用されます。

7. **VXLAN** トンネルが稼動状態であることを確認します。

```
# vxlan:show
```

8. REST API が正しく応答するかどうかをテストするには、それを使用するモジュールを一覧表示します。

```
# curl -u "admin:admin" http://localhost:8181/restconf/modules
```

以下と同じような出力が表示されます (この例は短く省略されています)。

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

9. ホストの internal_API IP を使用する REST のストリームを一覧表示します。

```
# curl -u "admin:admin" http://localhost:8181/restconf/streams
```

以下と同様の出力が表示されます。

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

10. ホストの internal_API IP で次のコマンドを実行して、NetVirt が動作していることを確認します。

```
# curl -u "admin:admin"
http://localhost:8181/restconf/operational/network-topology:network-topology/topology/topology/netvirt:1
```

NetVirt が動作可能であることを確認するには、以下の出力に注意してください。

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

4.2.3. Open vSwitch のテスト

Open vSwitch を検証するには、コンピュータノードの 1 つに接続し、適切に設定されて OpenDaylight に接続されていることを確認します。

手順

1. オーバークラウド内のコンピュータノードの 1 つに superuser として接続します。
2. Open vSwitch の設定を一覧表示します。

```
# ovs-vsctl show
```

3. 出力に複数のマネージャーがあることに注意してください。この例では、2 行目と 3 行目に複数のマネージャーが表示されています。

```

6b003705-48fc-4534-855f-344327d36f2a
  Manager "ptcp:6639:127.0.0.1"
  Manager "tcp:172.17.1.16:6640"
    is_connected: true
  Bridge br-ex
    fail_mode: standalone
    Port br-ex-int-patch
      Interface br-ex-int-patch
        type: patch
        options: {peer=br-ex-patch}
    Port br-ex
      Interface br-ex
        type: internal
    Port "eth2"
      Interface "eth2"
  Bridge br-isolated
    fail_mode: standalone
    Port "eth1"
      Interface "eth1"
    Port "vlan50"
      tag: 50
      Interface "vlan50"
        type: internal
    Port "vlan30"
      tag: 30
      Interface "vlan30"
        type: internal
    Port br-isolated
      Interface br-isolated
        type: internal
    Port "vlan20"
      tag: 20
      Interface "vlan20"
        type: internal
  Bridge br-int
    Controller "tcp:172.17.1.16:6653"
      is_connected: true
    fail_mode: secure
    Port br-ex-patch
      Interface br-ex-patch
        type: patch
        options: {peer=br-ex-int-patch}
    Port "tun02d236d8248"
      Interface "tun02d236d8248"
        type: vxlan
        options: {key=flow, local_ip="172.17.2.18",
remote_ip="172.17.2.20"}
    Port br-int
      Interface br-int
        type: internal
    Port "tap1712898f-15"
      Interface "tap1712898f-15"
  ovs_version: "2.7.0"

```

4. OpenDaylight が実行されているノードの IP アドレスを **tcp** Manager がポイントしていることを確認します。
5. Manager の下に **is_connected: true** と表示されていることを確認し、OVS から OpenDaylight への接続が確立されて、OVSDB プロトコルを使用していることを確かめます。
6. 各ブリッジ (**br-int** 以外) が存在し、Compute ロールでのデプロイメントに使用した NIC テンプレートと一致していることを確認します。
7. **tcp** 接続が、OpenDaylight サービスが実行されているところの IP に対応していることを確認します。
8. ブリッジ **br-int** に **is_connected: true** が表示され、OpenFlow プロトコルから OpenDaylight への接続が確立されていることを確認します。

詳細情報

- OpenDaylight は、**br-int** のブリッジを自動的に作成します。

4.2.4. コンピュートノード上の **Open vSwitch** の設定の確認

1. コンピュートノードに superuser として接続します。
2. **Open vSwitch** の設定を一覧表示します。

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

3. 出力を確認します。以下の例と同様の内容が表示されます。

```
_uuid          : 4b624d8f-a7af-4f0f-b56a-b8cfabf7635d
bridges        : [11127421-3bcc-4f9a-9040-ff8b88486508,
350135a4-4627-4e1b-8bef-56a1e4249bef]
cur_cfg        : 7
datapath_types : [netdev, system]
db_version     : "7.12.1"
external_ids   : {system-id="b8d16d0b-a40a-47c8-a767-
e118fe22759e"}
iface_types    : [geneve, gre, internal, ipsec_gre, lisp,
patch, stt, system, tap, vxlan]
manager_options : [c66f2e87-4724-448a-b9df-837d56b9f4a9,
defec179-720e-458e-8875-ea763a0d8909]
next_cfg       : 7
other_config   : {local_ip="11.0.0.30",
provider_mappings="datacentre:br-ex"}
ovs_version    : "2.7.0"
ssl            : []
statistics     : {}
system_type    : RedHatEnterpriseServer
system_version : "7.4-Maipo"
```

4. **other_config** オプションの値に、**VXLAN** トンネルを介してテナントネットワークに接続するローカルインタフェースの **local_ip** が設定されていることを確認します。
5. **other_config** のオプション下の **provider_mappings** の値が **OpenDaylightProviderMappings** Heat テンプレートパラメーターで指定されている値と一致

していることを確認します。この設定により、neutron の論理ネットワークが対応する物理インターフェースにマッピングされます。

4.2.5. neutron の設定の確認

手順

1. Controller ロールのノードの 1 つの superuser アカウントに接続します。
2. `/etc/neutron/neutron.conf` ファイルに `service_plugins=odl-router_v2,trunk` が含まれていることを確認します。
3. `/etc/neutron/plugin.ini` ファイルに以下の `ml2` 設定が記載されていることを確認します。

```
[ml2]
mechanism_drivers=opendaylight_v2

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

4. オーバークラウド のコントローラーの 1 つで、neutron エージェントが適切に稼働していることを確認します。

```
# openstack network agent list
```

5. メタデータと DHCP エージェントの両方の `admin_state_up` 値に `True` が設定されていることを確認します。

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

詳細情報

- ステップ 3 の `plugin.ini` の IP は、**InternalAPI** の仮想 IP アドレス (VIP) である必要があります。

- ステップ 5 の出力には、Open vSwitch エージェントまたは L3 エージェントは記載されていません。これらはいずれも OpenDaylight によって管理されるようになったので、出力に含まれていないのは望ましい状態です。

第5章 デバッグ

5.1. ログの特定

5.1.1. OpenDaylight ログへのアクセス

OpenDaylight では、`/var/log/containers/opendaylight` ディレクトリー内のコンテナにログを保管します。最新のログは `karaf.log` という名前です。OpenDaylight は以前のログに連番を付けます (例: `karaf.log.1`、`karaf.log.2`)。

5.1.2. OpenStack Networking のログへのアクセス

OpenStack networking のコマンドが失敗した場合は、最初に neutron のログを確認します。neutron のログは、neutron ノードの `/var/log/containers/neutron` ディレクトリー内の `server.log` ファイルにあります。

`server.log` ファイルには、OpenDaylight との通信に関するエラーも記載されます。neutron エラーが OpenDaylight との対話に起因している場合には、OpenDaylight ログを確認して、エラーの原因を特定する必要もあります。

5.2. ネットワークエラーのデバッグ

ネットワークでエラーが発生したにも拘らず (例: インスタンスの接続できないなど)、OpenStack コマンドを発行してもエラーが報告されなかったり、neutron のログにも記載されない場合には、OVS ノードを検査してネットワークトラフィックと OpenFlow のフローを確認すると役立つ場合があります。

1. ネットワークエラーが発生したノードに **superuser** としてログインします。
2. br-int スイッチに関する情報を表示します。

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

3. 出力を確認します。以下の例と同じような内容が表示されるはずです。

```
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000e4c153bdb306
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS
QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
  1(br-ex-patch): addr:ae:38:01:09:66:5b
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
  2(tap1f0f610c-8e): addr:00:00:00:00:00:00
    config:      PORT_DOWN
    state:       LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
  3(tun1147c81b59c): addr:66:e3:d2:b3:b8:e3
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
LOCAL(br-int): addr:e4:c1:53:bd:b3:06
config:        PORT_DOWN
```

```
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x5): frags=normal
miss_send_len=0
```

4. br-int スイッチの統計を一覧表示します。

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

5. 出力を確認します。以下の例と同じような内容が表示されるはずです。

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

詳細情報

- **ステップ 3** では、この OVS ノードに 3 つのポートがある点に注意してください。1 番目のポートはブリッジ br-ex に送信されるパッチポートで、このシナリオでは、外部ネットワークの接続に使用されます。2 番目のポートは、tap ポートで、DHCP エージェントインスタンスに接続されます (これは、ホストはコントローラーなのでわかります。コントローラーではなく、Compute ロール上の場合には、インスタンスとなります)。3 番目のポートは、テナントトラフィック用に作成された VXLAN トンネルポートです。
- 各ポートの用途を理解したら、ポートの統計を調べて、ポートがトラフィックの送受信を行っていることを確認します (**ステップ 4** を参照)。
- **ステップ 5** の出力から、各ポートがパケットを受信 (rx pkts) および送信 (tx pkts) していることを確認できます。

5.2.1. OpenFlow のフローを使用した高度なデバッグ

OpenFlow に精通している上級ユーザーの場合は、フローを確認して、トラフィックがドロップする場所を検出することができます。

1. フローを一覧表示してヒットしたパケット数を確認するには、以下のコマンドを入力します。

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

2. コマンドの出力を確認して、必要な情報を取得します。

```

OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x80000000, duration=90071.665s, table=0, n_packets=126816,
  n_bytes=8964820, priority=1,in_port=1
  actions=write_metadata:0x200000000001/0xffffffff0000000001,goto_table:1
  7
  cookie=0x80000000, duration=88967.292s, table=0, n_packets=473,
  n_bytes=44448, priority=4,in_port=2
  actions=write_metadata:0x400000000000/0xffffffff0000000001,goto_table:1
  7
  cookie=0x80000001, duration=88954.901s, table=0, n_packets=120636,
  n_bytes=8807869, priority=5,in_port=3
  actions=write_metadata:0x700000000001/0x1ffffff00000000001,goto_table:3
  6
  cookie=0x80000001, duration=90069.534s, table=17, n_packets=126814,
  n_bytes=8964712,
  priority=5,metadata=0x200000000000/0xffffffff0000000000
  actions=write_metadata:0xc0000200000222e0/0xffffffffffffff
  ffe,goto_table:19
  cookie=0x8040000, duration=90069.533s, table=17, n_packets=126813,
  n_bytes=8964658,
  priority=6,metadata=0xc000020000000000/0xffffffff0000000000
  actions=write_metadata:0xe00002138a000000/0xffffffff
  ffffffffe,goto_table:48
  cookie=0x8040000, duration=88932.689s, table=17, n_packets=396,
  n_bytes=36425,
  priority=6,metadata=0xc000040000000000/0xffffffff0000000000
  actions=write_metadata:0xe00004138b000000/0xffffffffffffff
  ffe,goto_table:48

```



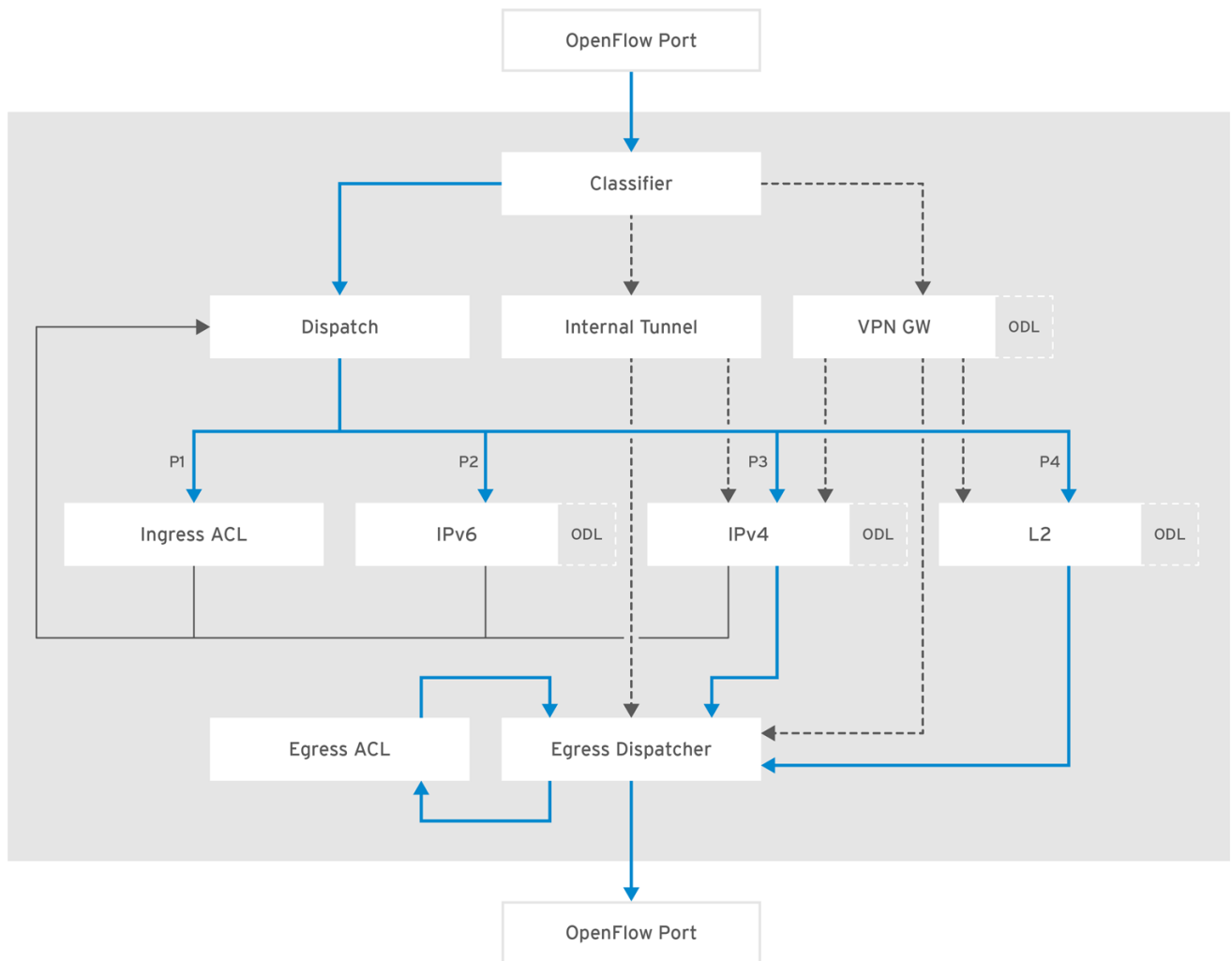
注記

この出力は長いいため途中から省略されています。

5.2.2. OpenFlow でのパケットのトラバース

理解すべき重要な点として、パケットに対して実行されるネットワーク機能は複数の異なる OpenFlow テーブルに分かれ、パケットはそれらのテーブルをゼロから順番に通過していくことが挙げられます。受信パケットはテーブル 0 に着信し、次に **OpenFlow のパイプライン** を経て進み、ポートから送信されて OpenDaylight のコントローラーに到達するか、ドロップされます。パケットは、処理の必要があるネットワーク機能に応じて、1 つまたは複数のテーブルをスキップする場合があります。テーブルの全体図と各テーブルが対応するネットワーク機能を以下に示します。

図5.1 OpenDaylight NetVirt OpenFlow のパイプライン



OPENSTACK_436456_0217

第6章 デプロイメントの例

6.1. テナントネットワークを使用した模範的なインストールシナリオ

本項では、OpenStack を使用した実稼働環境における OpenDaylight のインストールの例を考察します。このシナリオでは、テナントトラフィックの分離にトンネリング (**VXLAN**) が使用されます。

6.1.1. 物理トポロジー

このシナリオのトポロジーは、6 つのノードで構成されます。

- 1 x director アンダークラウドノード
- 3 x OpenStack オーバークラウドコントローラー。OpenStack サービスに加えて OpenDaylight SDN コントローラーがインストール済み。
- 2 x OpenStack オーバークラウドコンピュートノード

6.1.2. 物理ネットワーク環境のプランニング

オーバークラウドのコントローラーノードは、それぞれ、3 つのネットワークインターフェースカード (NIC) を使用します。

名前	目的
nic1	Management ネットワーク (例: SSH を介したノードへのアクセス)
nic2	テナント (VXLAN) キャリア、Provisioning (PXE、DHCP)、 Internal API ネットワーク
nic3	パブリック API のネットワークアクセス

オーバークラウドのコンピュートノードには 3 つの NIC が実装されます。

名前	目的
nic1	Management ネットワーク
nic2	Tenant キャリアー、Provisioning、 Internal API ネットワーク
nic3	External (Floating IP) ネットワーク

アンダークラウドノードには 2 つの NIC が実装されます。

名前	目的
nic1	Management ネットワークに使用

名前	目的
nic2	Provisioning ネットワークに使用

6.1.3. NIC の接続性のプランニング

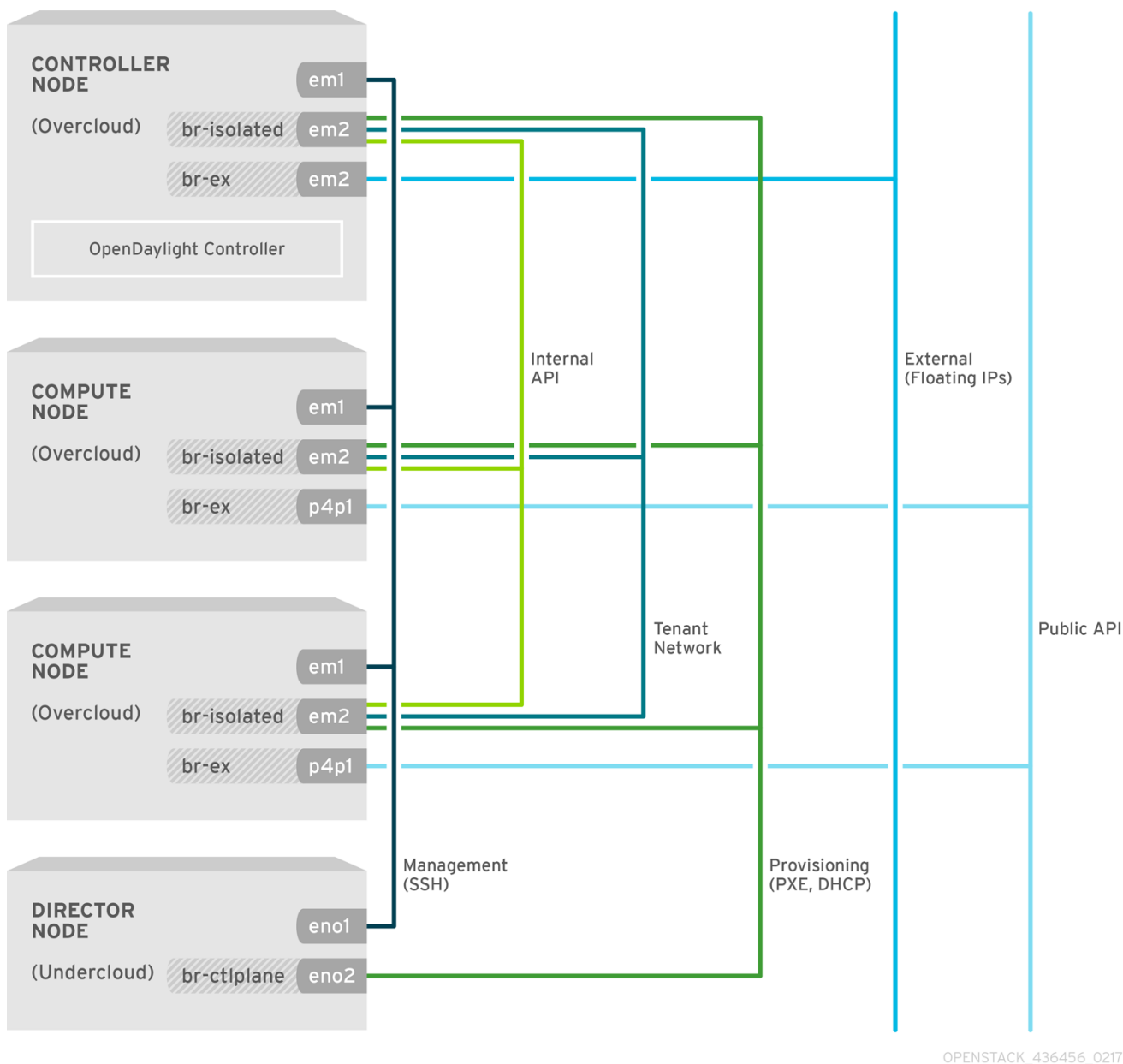
この場合は、環境ファイルは番号付けられた抽象的なインターフェースデバイス名 (**nic1**、**nic2**) を使用し、ホストのオペレーティングシステムで提示されている実際のデバイス名 (例: **eth0**、**eno2** など) は使用しません。同じロールに属するホストに全く同じネットワークインターフェースデバイス名は必要ありません。1 つのホストが **em1** と **em2** のインターフェースを使用して、他のホストが **eno1** と **eno2** を使用しても問題はありません。各 NIC は **nic1** および **nic2** と呼ばれます。

抽象化された NIC スキームでは、稼働中かつ接続済みのインターフェースにのみ依存します。ホストによってインターフェースが異なる場合には、ホストを接続するのに必要な最小限のインターフェース数を使用すれば十分です。たとえば、1 台のホストに物理インターフェースが 4 つあり、他のホストには 6 つある場合、**nic1**、**nic2**、**nic3**、**nic4** のみを使用して、両ホストに 4 本のケーブルを接続します。

6.1.4. ネットワーク、VLAN、IP のプランニング

このシナリオでは、ネットワーク分離を使用して、Management、Provisioning、**Internal API**、Tenant、Public API、Floating IP のネットワークトラフィックを分離します。

図6.1 このシナリオで使用するネットワークトポロジーの詳細

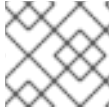


以下の表には、各ネットワークに関連付けられる VLAN ID と IP サブネットをまとめています。

ネットワーク	VLAN ID	IP サブネット
プロビジョニング	ネイティブ	192.0.5.0/24
Internal API	600	172.17.0.0/24
Tenant	603	172.16.0.0/24
パブリック API	411	10.35.184.144/28
Floating IP	412	10.35.186.146/28

OpenStack Platform director は **br-isolated** OVS ブリッジを作成し、ネットワーク設定ファイルの定義に従って各ネットワークの **VLAN** インターフェースを追加します。director は **br-ex** ブリッジも作成して、関連するネットワークインターフェースが接続されます。

ホスト間の接続性を提供する物理ネットワークスイッチが、**VLAN ID** を適用するように適切に設定されていることを確認します。ホストに接続する全スイッチポートは、**VLAN** を使用して「trunk」として設定する必要があります。ここで「trunk」という用語は、複数の **VLAN ID** が同じポートを通過できるポートという意味で使われています。



注記

物理スイッチの設定に関する内容は、本書の対象範囲外です。



注記

network-environment.yaml ファイルの **TenantNetworkVlanID** 属性を使用し、**VXLAN** トンネリングを使用する場合の Tenant ネットワークの VLAN タグを定義することができます (例: **VXLAN** テナントのトラフィックが VLAN タグ付けされた下層のネットワーク上で転送される)。Tenant ネットワークがネイティブ VLAN 上で実行されるようにした方が望ましい場合には、この値を空にすることも可能です。また、VLAN テナント種別のネットワークを使用する場合には、**TenantNetworkVlanID** に指定されている値以外の VLAN タグを使用することができる点にも注意してください。

6.1.5. このシナリオで使用する OpenDaylight の設定ファイル

このシナリオにおける OpenStack と OpenDaylight のデプロイには、アンダークラウドで以下のデプロイメントコマンドを実行しています。

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

また、本ガイドでは、このシナリオに使用した設定ファイルとその内容を記載し、使用した設定についても説明しています。

6.1.5.1. extra_env.yaml ファイル

このファイルにはパラメーターが 1 つしかありません。

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

これらは、OpenDaylight によって制御される各ノードのマッピングです。物理ネットワーク **datacenter** は、**br-ex** OVS ブリッジにマッピングされ、Tenant ネットワークのトラフィックは **br-isolated** OVS ブリッジにマッピングされます。

6.1.5.2. undercloud.conf ファイル

このファイルは、`/home/stack/baremetal-vlan/` ディレクトリーにあります。



注記

ファイルのパスは、カスタマイズされたバージョンの設定ファイルをポイントしています。

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

上記の例では、Provisioning ネットワーク用の 192.0.5.0/24 サブネットが使用されています。物理インターフェース **eno2** はアンダークラウドノードでプロビジョニング用に使用されます。

6.1.5.3. network-environment.yaml ファイル

これは、ネットワークを設定するメインのファイルで、`/home/stack/baremetal-vlan/` ディレクトリーにあります。以下のファイルでは、VLAN ID と IP サブネットが異なるネットワークとプロバイダーマッピングに指定されます。nic-configs ディレクトリー内の **controller.yaml** および **compute.yaml** は、コントローラーノードとコンピュータードのネットワーク設定を指定するのに使用されます。

この例では、コントローラーノードの数 (3) とコンピュータードの数 (2) が指定されています。

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to use
  # for
  # override the default.
  OS1::Triple0::Compute::Net::SoftwareConfig: nic-configs/compute.yaml
  OS::Triple0::Controller::Net::SoftwareConfig: nic-
  configs/controller.yaml

  # Network isolation configuration
  # Service section
  # If some service should be disable, use the following example
  # OS::Triple0::Network::Management: OS::Heat::None
  OS::Triple0::Network::External: /usr/share/openstack-tripleo-heat-
  templates/network/external.yaml
  OS::Triple0::Network::InternalApi: /usr/share/openstack-tripleo-heat-
  templates/network/internal_api.yaml
  OS::Triple0::Network::Tenant: /usr/share/openstack-tripleo-heat-
  templates/network/tenant.yaml
  OS::Triple0::Network::Management: OS::Heat::None
  OS::Triple0::Network::StorageMgmt: OS::Heat::None
  OS::Triple0::Network::Storage: OS::Heat::None
```

```

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

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

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

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

parameter_defaults:
# Customize all these values to match the local environment
InternalApiNetCidr: 172.17.0.0/24
TenantNetCidr: 172.16.0.0/24
ExternalNetCidr: 10.35.184.144/28
# CIDR subnet mask length for provisioning network
ControlPlaneSubnetCidr: '24'
InternalApiAllocationPools: [{'start': '172.17.0.10', 'end':
'172.17.0.200'}]
TenantAllocationPools: [{'start': '172.16.0.100', 'end':
'172.16.0.200'}]

```

```

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

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

```

6.1.5.4. controller.yaml ファイル

このファイルは、`/home/stack/baremetal-vlan/nic-configs/` ディレクトリー内にあります。この例では、**br-isolated** と **br-ex** の 2 つのスイッチを定義しています。**nic2** は **br-isolated** の下、**nic3** は **br-ex** の下です。

```

heat_template_version: pike

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

parameters:

```

```

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

```

```

that will be added to resolv.conf.
  type: comma_delimited_list
  EC2MetadataIp: # Override this with parameter_defaults
  description: The IP address of the EC2 metadata server.
  type: string

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

```

```

        name: nic3
        # force the MAC address of the bridge to this interface
        -
          type: vlan
          vlan_id: {get_param: ExternalNetworkVlanID}
          addresses:
            -
              ip_netmask: {get_param: ExternalIpSubnet}
          routes:
            -
              default: true
              next_hop: {get_param:
ExternalInterfaceDefaultRoute}

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

```

6.1.5.5. compute.yaml ファイル

このファイルは `/home/stack/baremetal-vlan/nic-configs/` ディレクトリーにあります。コンピュートのオプションの大半はコントローラーのオプションと同じです。この例では、**nic3** は、外部接続 (Floating IP ネットワーク) に使用される **br-ex** の下にあります。

```

heat_template_version: pike

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

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  InternalApiNetworkVlanID:
    default: ''

```

```

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

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'

```



```

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

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

```

6.1.6. このシナリオで使用する **Red Hat OpenStack Platform director** の設定ファイル

6.1.6.1. neutron.conf ファイル

このファイルは、`/etc/neutron/` ディレクトリーにあり、以下の情報が含まれています。

```

[DEFAULT]
service_plugins=odl-router_v2,trunk

```

6.1.6.2. ml2_conf.ini ファイル

このファイルは、`/etc/neutron/plugins/ml2/` ディレクトリーにあり、以下の情報が含まれています。

■

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

[ml2_type_vlan]
network_vlan_ranges = datacentre:412:412

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

1. [ml2] セクションの下には、ネットワーク種別として VXLAN が使用されており、opendaylight_v2 メカニズムドライバーが指定されている点に注意してください。
2. [ml2_type_vlan] の下には、network-environment.yaml ファイルで設定されているのと同じマッピングを指定する必要があります。
3. [ml2_odl] の下には、OpenDaylightController にアクセスするための設定が記載されているはずです。

この情報を使用して、OpenDaylight コントローラーへのアクセスを確認することができます。

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

6.2. プロバイダーネットワークを使用する模範的なインストールシナリオ

このインストールシナリオでは、テナントネットワークではなく、プロバイダーネットワークを使用した OpenStack と OpenDaylight の例を示します。外部の neutron プロバイダーネットワークは、レイヤー 3 (L3) およびその他のネットワークサービスを提供する物理ネットワークインフラストラクチャーに仮想マシンインスタンスをブリッジングします。大半の場合は、プロバイダーネットワークは、VLAN ID を使用してレイヤー 2 (L2) セグメンテーションを実装します。プロバイダーネットワークは、プロバイダーネットワーク上で仮想マシンインスタンスの起動をサポートする各コンピュータノードでプロバイダーブリッジにマッピングします。

6.2.1. 物理トポロジー

このシナリオのトポロジーは、6 つのノードで構成されます。

- 1 x director アンダークラウドノード
- 3 x OpenStack オーバークラウドコントローラー。OpenStack サービスに加えて OpenDaylight SDN コントローラーがインストール済み。
- 2 x OpenStack オーバークラウドコンピュータノード

6.2.2. 物理ネットワーク環境のプランニング

オーバークラウドコントローラーノードはそれぞれ、4 つのネットワークインターフェースカード (NIC) を使用します。

名前	目的
nic1	Management ネットワーク (例: SSH を介したノードへのアクセス)
nic2	Provisioning (PXE、DHCP)、 Internal API ネットワーク
nic3	Tenant ネットワーク
nic4	Public API ネットワーク、Floating IP ネットワーク

オーバークラウドのコンピュータノードには、4 つの NIC が実装されます。

名前	目的
nic1	Management ネットワーク
nic2	Provisioning および Internal API ネットワーク
nic3	Tenant ネットワーク
nic4	Floating IP ネットワーク

アンダークラウドノードには 2 つの NIC が実装されます。

名前	目的
nic1	Management ネットワークに使用
nic2	Provisioning ネットワークに使用

6.2.3. NIC の接続性のプランニング

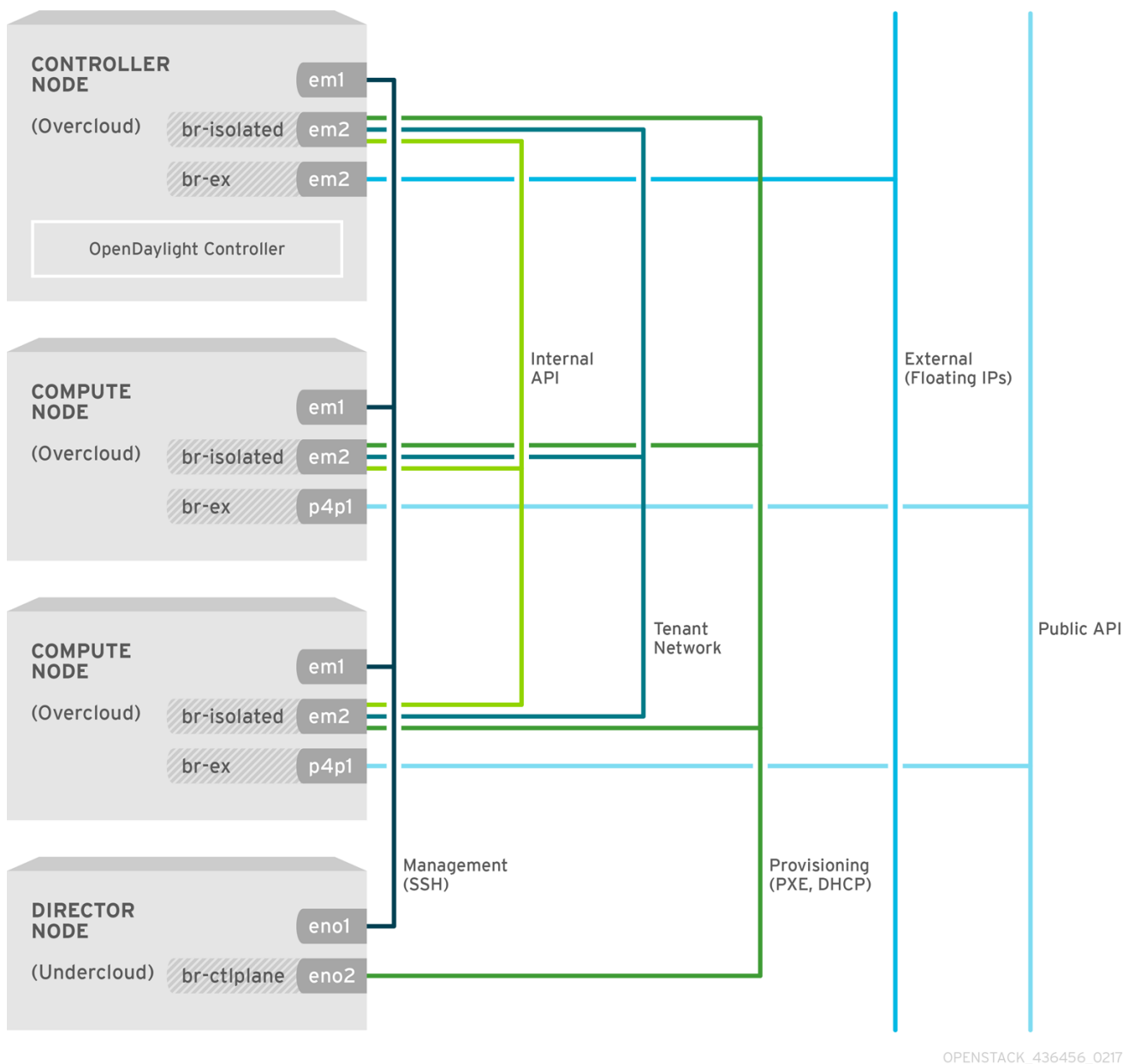
この場合は、環境ファイルは番号付けられた抽象的なインターフェースデバイス名 (**nic1**、**nic2**) を使用し、ホストのオペレーティングシステムで提示されている実際のデバイス名 (例: **eth0**、**eno2** など) は使用しません。同じロールに属するホストに全く同じネットワークインターフェースデバイス名は必要ありません。1 つのホストが **em1** と **em2** のインターフェースを使用して、他のホストが **eno1** と **eno2** を使用しても問題はありません。各 NIC は **nic1** および **nic2** と呼ばれます。

抽象化された NIC スキームでは、稼働中かつ接続済みのインターフェースにのみ依存します。ホストによってインターフェースが異なる場合には、ホストを接続するのに必要な最小限のインターフェース数を使用すれば十分です。たとえば、1 台のホストに物理インターフェースが 4 つあり、他のホストには 6 つある場合、**nic1**、**nic2**、**nic3**、**nic4** のみを使用して、両ホストに 4 本のケーブルを接続します。

6.2.4. ネットワーク、VLAN、IP のプランニング

このシナリオでは、ネットワーク分離を使用して、Management、Provisioning、**Internal API**、Tenant、Public API、Floating IP のネットワークトラフィックを分離します。

図6.2 このシナリオで使用するネットワークトポロジーの詳細



以下の表には、各ネットワークに関連付けられる VLAN ID と IP サブネットをまとめています。

ネットワーク	VLAN ID	IP サブネット
プロビジョニング	ネイティブ	192.0.5.0/24
Internal API	600	172.17.0.0/24
Tenant	554,555-601	172.16.0.0/24
パブリック API	552	192.168.210.0/24
Floating IP	553	10.35.186.146/28

OpenStack Platform director は **br-isolated** OVS ブリッジを作成し、ネットワーク設定ファイルの定義に従って各ネットワークの **VLAN** インターフェースを追加します。**br-ex** ブリッジも director によっ

て自動的に作成され、関連するネットワークインターフェースが接続されます。

ホスト間の接続性を提供する物理ネットワークスイッチが、**VLAN ID** を適用するように適切に設定されていることを確認します。ホストに接続する全スイッチポートは、**VLAN** を使用して **trunk** として設定する必要があります。ここで「trunk」という用語は、複数の **VLAN ID** が同じポートを通過できるポートという意味で使用しています。



注記

物理スイッチの設定に関する内容は、本書の対象範囲外です。



注記

network-environment.yaml の **TenantNetworkVlanID** で、**VXLAN** トンネリングを使用する場合の Tenant ネットワークの VLAN タグを定義することができます (**VXLAN** テナントのトラフィックが VLAN タグ付けされた下層のネットワーク上で転送される)。Tenant ネットワークがネイティブ VLAN 上で実行されるようにした方が望ましい場合には、この値を空にすることも可能です。また、VLAN テナント種別のネットワークを使用する場合には、**TenantNetworkVlanID** に指定されている値以外の VLAN タグを使用することができる点にも注意してください。

6.2.5. このシナリオで使用する OpenDaylight の設定ファイル

このシナリオにおける OpenStack と OpenDaylight のデプロイには、アンダークラウドで以下のデプロイメントコマンドを実行しています。

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

本ガイドには、このシナリオの設定ファイル、設定ファイルの内容、設定に関する説明も記載しています。

6.2.5.1. extra_env.yaml ファイル

このファイルにはパラメーターが 1 つしかありません。

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

これらは、OpenDaylight によって制御される各ノードのマッピングです。物理ネットワーク **datacenter** は、**br-ex** OVS ブリッジにマッピングされ、Tenant ネットワークのトラフィックは **br-vlan** OVS ブリッジにマッピングされます。

6.2.5.2. undercloud.conf ファイル

このファイルは **/home/stack/** ディレクトリー内にあります。



注記

ファイルのパスは、カスタマイズされたバージョンの設定ファイルをポイントしていません。

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

上記の例では、Provisioning ネットワーク用の 192.0.5.0/24 サブネットが使用されています。物理インターフェース **eno2** はアンダークラウドノードでプロビジョニング用に使用されます。

6.2.5.3. network-environment.yaml ファイル

これは、ネットワークを設定する主要なファイルで、**/home/stack/baremetal-vlan/** ディレクトリにあります。以下のファイルでは、VLAN ID と IP サブネットが異なるネットワークとプロバイダーマッピングに指定されます。nic-configs ディレクトリ内の **controller.yaml** および **compute.yaml** ファイルは、コントローラーノードとコンピューターノードのネットワーク設定を指定するのに使用されます。

この例では、コントローラーノードの数 (3) とコンピューターノードの数 (2) が指定されています。

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

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

    # Port assignments for the VIPs
    OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
    tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
```

```

tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml

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

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

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

parameter_defaults:
  # Customize all these values to match the local environment
  InternalApiNetCidr: 172.17.0.0/24
  TenantNetCidr: 172.16.0.0/24
  ExternalNetCidr: 192.168.210.0/24
  # CIDR subnet mask length for provisioning network
  ControlPlaneSubnetCidr: '24'
  InternalApiAllocationPools: [{'start': '172.17.0.10', 'end':
'172.17.0.200'}]
  TenantAllocationPools: [{'start': '172.16.0.100', 'end':
'172.16.0.200'}]
  # Use an External allocation pool which will leave room for floating IPs
  ExternalAllocationPools: [{'start': '192.168.210.2', 'end':
'192.168.210.12'}]

```

```
# Set to the router gateway on the external network
ExternalInterfaceDefaultRoute: 192.168.210.1
# Gateway router for the provisioning network (or Undercloud IP)
ControlPlaneDefaultRoute: 192.0.5.1
# Generally the IP of the Undercloud
EC2MetadataIp: 192.0.5.1
InternalApiNetworkVlanID: 600
TenantNetworkVlanID: 554
ExternalNetworkVlanID: 552
# Define the DNS servers (maximum 2) for the overcloud nodes
DnsServers: ["10.35.28.28", "8.8.8.8"]
# May set to br-ex if using floating IPs only on native VLAN on bridge
br-ex
NeutronExternalNetworkBridge: ""
# The tunnel type for the tenant network (vxlan or gre). Set to '' to
disable tunneling.
NeutronTunnelTypes: ''
# The tenant network type for Neutron (vlan or vxlan).
NeutronNetworkType: 'vlan'
# The OVS logical->physical bridge mappings to use.
# NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-isolated'
# The Neutron ML2 and OpenVSwitch vlan mapping range to support.
NeutronNetworkVLANRanges: 'datacentre:552:553,tenant:555:601'
# Nova flavor to use.
OvercloudControlFlavor: baremetal
OvercloudComputeFlavor: baremetal
# Number of nodes to deploy.
ControllerCount: 3
ComputeCount: 2

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

6.2.5.4. controller.yaml ファイル

このファイルは、`/home/stack/baremetal-vlan/nic-configs/` ディレクトリーにあります。この例では、**br-isolated**、**br-vlan**、**br-ex** のスイッチを定義しています。**nic2** は **br-isolated** の下、**nic3** は **br-ex** の下です。

```
heat_template_version: pike

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

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
```



```

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

```

```

    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: interface
              name: nic1
              use_dhcp: false
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                -
                  ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
          members:
            -
              type: interface
              name: nic2
              # force the MAC address of the bridge to this interface
              primary: true
            -
              type: vlan
              vlan_id: {get_param: InternalApiNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: InternalApiIpSubnet}
            -
              type: ovs_bridge
              name: br-ex
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              members:
                -
                  type: interface
                  name: nic4
                  # force the MAC address of the bridge to this interface
                -
                  type: vlan
                  vlan_id: {get_param: ExternalNetworkVlanID}
                  addresses:

```

```

        -
          ip_netmask: {get_param: ExternalIpSubnet}
        routes:
          -
            default: true
            next_hop: {get_param:
ExternalInterfaceDefaultRoute}
        -
          type: ovs_bridge
          name: br-vlan
          use_dhcp: false
          dns_servers: {get_param: DnsServers}
          members:
            -
              type: interface
              name: nic3
            -
              type: vlan
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}

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

```

6.2.5.5. compute.yaml ファイル

このファイルは `/home/stack/baremetal-vlan/nic-configs/` ディレクトリーにあります。コンピュートのオプションの大半はコントローラーのオプションと同じです。この例では、**nic4** は、外部接続 (Floating IP ネットワーク) に使用される **br-ex** の下に 있습니다。

```

heat_template_version: pike

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

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

```

```

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

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

```

```

    type: interface
    name: nic1
    use_dhcp: false
  -
    type: ovs_bridge
    name: br-isolated
    use_dhcp: false
    dns_servers: {get_param: DnsServers}
    addresses:
      -
        ip_netmask:
          list_join:
            - '/'
            - - {get_param: ControlPlaneIp}
              - {get_param: ControlPlaneSubnetCidr}
    routes:
      -
        ip_netmask: 169.254.169.254/32
        next_hop: {get_param: EC2MetadataIp}
      -
        next_hop: {get_param: ControlPlaneDefaultRoute}
        default: true
    members:
      -
        type: interface
        name: nic2
        # force the MAC address of the bridge to this interface
        primary: true
      -
        type: vlan
        vlan_id: {get_param: InternalApiNetworkVlanID}
        addresses:
          -
            ip_netmask: {get_param: InternalApiIpSubnet}
  -
    type: ovs_bridge
    name: br-ex
    use_dhcp: false
    members:
      -
        type: interface
        name: nic4
  -
    type: ovs_bridge
    name: br-vlan
    use_dhcp: false
    dns_servers: {get_param: DnsServers}
    members:
      -
        type: interface
        name: nic3
      -
        type: vlan
        vlan_id: {get_param: TenantNetworkVlanID}
        addresses:
          -

```

```

        ip_netmask: {get_param: TenantIpSubnet}

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

```

6.2.6. このシナリオで使用する Red Hat OpenStack Platform director の設定ファイル

6.2.6.1. neutron.conf ファイル

このファイルは、`/etc/neutron/` ディレクトリーにあり、以下の情報が含まれています。

```

[DEFAULT]
service_plugins=odl-router_v2,trunk

```

6.2.6.2. ml2_conf.ini ファイル

このファイルは、`/etc/neutron/plugins/ml2/` ディレクトリーにあり、以下の情報が含まれています。

```

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

[ml2_type_flat]
flat_networks = datacentre

[ml2_type_geneve]
[ml2_type_gre]
tunnel_id_ranges = 1:4094

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

[ml2_type_vxlan]
vni_ranges = 1:4094
vxlan_group = 224.0.0.1

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

```

1. [ml2] セクションの下には、ネットワーク種別として VXLAN が使用されており、**opendaylight_v2** メカニズムドライバーが指定されている点に注意してください。

2. [ml2_type_vlan] の下で、**network-environment.yaml1** ファイルと同じマッピングを設定します。
3. [ml2_odl] の下には、OpenDaylightController にアクセスするための設定が記載されているはずです。

この情報を使用して、OpenDaylight コントローラーへのアクセスを確認することができます。

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

第7章 OPENDAYLIGHT での高可用性とクラスタリング

Red Hat OpenStack Platform 13 は、neutron と OpenDaylight コントローラーの両方で高可用性クラスタリングをサポートしています。以下の表には、高可用性クラスターを実行する場合の推奨アーキテクチャーをまとめています。

ノード種別	ノード数	ノードのモード
Neutron	3	active/active/active
OpenDaylight	3	active/active/active
コンピュータード (nova または OVS)	任意	

OpenDaylight のロールは、コンポーザブルなので、neutron と同じノードまたは別のノードにデプロイすることができます。設定はすべてアクティブモードです。全ノードが要求を処理できます。要求を受信したノードが処理できない場合には、そのノードが別の適切なノードに転送します。全ノードが相互に同期します。Open_vSwitch Database (OVSDB) スキーマのサウスバウンドでは、利用可能なコントローラーノードが Open vSwitch を共有し、各スイッチはクラスター内の特定のノードによって処理されます。

7.1. 高可用性とクラスタリング向けの OPENDAYLIGHT の構成

Red Hat OpenStack Platform director は OpenDaylight コントローラーノードをデプロイするので、OpenDaylight のクラスタリングを設定するために必要な全情報を持っています。OpenDaylight の各ノードには、ノードの **ロール** (クラスター内での名前) を特定して、クラスター内の他のノード (**シードノード**) を少なくともリストした **akka.conf** 設定ファイルが必要です。ノードには、クラスター内のデータの複製方法を記述した **module-shards.conf** ファイルも必要です。Red Hat OpenStack Platform director は、選択したデプロイメントの構成に基づいて適切に設定を行います。**akka.conf** ファイルはノードに依存する一方、**module-shards.conf** ファイルはノードとインストールされているデータストア (ならびにそのインストールされた機能。その大部分を制御します) に依存します。

akka.conf のサンプルファイル:

```
odl-cluster-data {
  akka {
    remote {
      netty.tcp {
        hostname = "192.0.2.1"
      }
    },
    cluster {
      seed-nodes = [
        "akka.tcp://opendaylight-cluster-data@192.0.2.1:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.2:2550",
        "akka.tcp://opendaylight-cluster-data@192.0.2.3:2550"],
      roles = [ "member-1" ]
    }
  }
}
```


上記のノードの例はシードノードです。これらは、現在のクラスター設定全体を反映する必要はありません。シードノードのリストを使用して現在のクラスター内の実ノードが到達可能な限り、起動するノードはクラスターに参加することができます。設定ファイルでは、名前の代わりに IP アドレスを使用することができます。

7.2. クラスターの動作

クラスターは、動的に定義されないで、自動調整は行いません。新しいノードを起動して、その新しいノードのみを設定し、既存のクラスターに接続することはできません。クラスターは、クラスター管理の **RPC** を使用して、ノードの追加と削除について通知する必要があります。

クラスターはリーダー/フォロワーモデルです。アクティブなノードの 1 つがリーダーとして選択され、残りのアクティブなノードがフォロワーになります。クラスターは、Raft のコンセンサスをベースとするモデルに従って永続化を処理します。この原則に従ってクラスター内のノードの過半数が同意すれば、1 つのトランザクションのみがコミットされます。

OpenDaylight では、ノードがクラスターとの接続を失った場合には、ローカルのトランザクションは先に進められなくなります。最終的には、タイムアウトして (デフォルトでは 10 分)、フロントエンドのアクターが停止します。これらはすべてシャードごとに適用されるので、シャードによって異なるリーダーを持つことができます。この動作により、以下のいずれかの状況となります。

- 通信がない状態が 10 分未満の場合には、マイノリティーノードがマジョリティーリーダーに再接続します。全トランザクションがロールバックされ、大半のトランザクションは再生されます。
- 通信がない状態が 10 分以上の場合には、マイノリティーノードが稼働停止し、ログに情報を記録します。読み取り専用の要求はまだ完了するはずですが、変更は永続されず、ノードは自律的にクラスターに再度参加できません。

これは、クラスター外のノードをユーザーがモニタリングする必要があることを意味します。可用性とクラスターの同期をチェックして、同期されていない時間が長すぎる場合には、それらのノードを再起動します。ノードをモニタリングするには、ユーザーは Jolokia REST サービスを使用します (詳しくは、「[Jolokia を使用したモニタリング](#)」を参照)。

7.3. クラスターの要件

ボンディングや MTU など、クラスターをサポートするために特定のネットワーク要件はありません。クラスターの通信は、高レイテンシーをサポートしていませんが、データセンターレベルのレイテンシーは受容可能です。

7.4. OPEN VSWITCH の設定

各スイッチは、Red Hat OpenStack Platform director によりすべてのコントローラーで自動設定されます。OVSDB は、クラスターノード内のスイッチ共有をサポートして、一定レベルのロードバランシングを可能にします。ただし、各スイッチはクラスター内の全ノードに連絡して、最初に応答があったノードを選択し、そのノードをデフォルトでマスタースイッチにします。この動作により、コントローラーに割り当てられたノードの **クラスタリング** が行われ、最も応答が早いノードが大半のスイッチを処理します。

7.5. クラスターのモニタリング

7.5.1. Jolokia のモニタリング

クラスターのステータスをモニタリングするには、OpenDaylight で **Jolokia** のサポートを有効化する

必要があります。Jolokia のアドレス

(<http://192.0.2.1:8181/jolokia/read/org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-config,type=DistributedConfigDatastore>) からデータストアクラスターのレポートを取得することができます。このレポートは JSON 形式の文書です。



注記

環境に応じて **IP address** と **member-1** の値を変更する必要があります。どのノードが応答してもよい場合には、IP アドレスは仮想 IP をポイントすることができます。ただし、特定のコントローラーのアドレスを指定した方が、より適切な結果が得られます。

この説明には、各ノードで同じリーダーを指定する必要があります。



注記

アップストリームの OpenDaylight チームが開発した **Cluster Monitor Tool** でクラスターをモニタリングすることも可能です。これは、OpenDaylight の Github リポジトリにあります。

このツールは、Red Hat OpenStack Platform 13 の一部ではないため、Red Hat からのサポートは提供していません。

7.6. OPENDAYLIGHT のポートについての理解

OpenDaylight の正式な全ポートの一覧は、OpenDaylight の Wiki ページに掲載されています。このシナリオに関連するポートは以下のとおりです。

ポート番号	用途
2550	クラスタリング
6653	OpenFlow
6640、6641	OVSDB
8087	neutron
8081	RESTCONF、Jolokia

コントローラーでこれらのポートへのトラフィックをブロックすると、以下のような影響があります。

クラスタリング

クラスター化されたノードは、通信できなくなります。クラスター化モードで実行する場合には、各ノードにピアが少なくとも 1 つ必要です。全トラフィックがブロックされた場合には、コントローラーが停止します。

OpenFlow

スイッチがコントローラーに到達できなくなります。

OVSDB

Open vSwitch は、コントローラーに到達できなくなります。コントローラーは、アクティブな OVS 接続を開始できますが、スイッチからコントローラーへの ping は失敗して、他のコントローラーにフェイルオーバーします。

neutron

Neutron がコントローラーに到達できなくなります。

RESTCONF

REST エンドポイントを使用する外部ツールは、コントローラーに到達できなくなります。このシナリオで影響を受けるのは、モニタリングツールのみのはずです。

OpenDaylight 側では、ログに表示されるのはクラスターのブロックされているトラフィックのみです。これは、他のポートが ODL コントローラーとの通信に使用されるのが理由です。現在、Red Hat OpenStack Platform director によって開放されるポートの一覧は、<https://github.com/openstack/tripleo-heat-templates/blob/master/puppet/services/.opendaylight-api.yaml#L114> に記載されています。

ターゲットデバイスでこれらのポートへのトラフィックをブロックすると、以下のような影響があります。

クラスタリング

クラスター化されたノードは、通信できなくなります。クラスター化モードで実行する場合には、各ノードにピアが少なくとも 1 つ必要です。全トラフィックがブロックされた場合には、コントローラーが停止します。

OpenFlow

コントローラーはフローをプッシュできなくなります。

OVSDB

コントローラーはスイッチに到達できなくなります (コントローラーは OVS のパッシブ接続に応答可能となります)。

2 番目の状況では、いずれの場合も、OpenDaylight は設定と稼働状態は別々に維持管理するので、設定は引き続き到達不可能なデバイスをポイントし、コントローラーはそれらのデバイスへの接続を試み続けます。

7.7. OPENDAYLIGHT のフローについての理解

フロー	説明
Neutron → ODL	Neutron、HA プロキシ、ODL の順序です。Pacemaker は仮想 IP (物理 IP が 3 つでバックアップ) を管理します。ドライバーは TCP セッションを開いたままに維持しようと試みます。これにより影響を受ける場合があります (https://review.openstack.org/#/c/440866/)。
ODL → Neutron	ODL により開始される通信はありません。
ODL → ODL	ODL ノードはポート 2550 (設定可能) で相互に通信します。

フロー	説明
ODL → OVS	ODL は、OVSDDB (ポート 6640 および 6641) と OpenFlow (ポート 6633) を使用してスイッチと通信します。仮想 IP は関与せず、ODL は全スイッチの IP アドレスを認識しており、各 ODL ノードは全スイッチについて認識しています。
OVS → ODL	ODL は、OVSDDB (ポート 6640 および 6641) と OpenFlow (ポート 6633) を使用してスイッチと通信します。仮想 IP は関与せず、ODL が全スイッチを設定して、全コントローラーを認識するようにします。スイッチからコントローラーへの通知は全ノードに送信されます。

7.8. NEUTRON DHCP エージェント HA

デフォルトの設定では、DHCP エージェントが OVS エージェントと共に全 neutron ノードで実行されます。ロールはコンポーザブルなので、エージェントはコントローラーから分離することができます。DHCP エージェントは、ポートを立ち上げる段階とリースの更新時の高可用性のみに重要です。ポートの作成時には、neutron が IP と MAC アドレスを割り当てて、ポートが立ち上がる前に全 DHCP エージェントを適切に設定します。この段階では、受信する DHCP 要求は、全 DHCP エージェントが応答します。

DHCP エージェントに問題が発生した場合にデータプレーンの可用性を最大限にするために、リース期間が長く設定されており、ノードには更新のための遅延が短時間設定されています。このため、DHCP エージェントはほとんど必要ありませんが、必要とされる場合には、利用できない DHCP エージェントに対して要求を実行するノードは即時にフェイルオーバーし、ブロードキャスト要求を発行して、残りの DHCP エージェントのいずれかを自動的に選択します。

エージェントには、独自のプロセスモニターがあります。**systemd** によりエージェントが起動され、それらの名前空間が作成されて、その内部でプロセスが開始します。エージェントが機能しなくなった場合には、名前空間は稼働状態のままで、**systemd** は、他のプロセスを再起動したり終了したりせずにエージェントを再起動します (systemd はそれらのプロセスを所有しません)。次にエージェントは名前空間を再度接続して、実行中の全プロセスと共に再利用します。

7.9. NEUTRON メタデータエージェント HA

リファレンス実装では、メタデータサービスは、対応する DHCP エージェントと同じ名前空間内で、ネットワークノードと統合されたコントローラー上で実行されます。メタデータプロキシは、ポート 80 をリッスンし、既知のメタデータアドレスを使用して静的ルートでトラフィックを仮想マシンからプロキシにリダイレクトします。プロキシは Unix ソケットを使用して同じノード上でメタデータサービスと通信し、メタデータサービスは nova と通信します。Unix ソケットでは、プロキシとサービス間で IP をルーティング可能にする必要はないので、メタデータサービスは、ノードがルーティングされなくても利用可能です。HA は keepalive と VRRP エレクションを使用して処理されます。フェイルオーバー時間は 2-5 秒です。エージェントは、DHCP エージェントと同じように処理されます (systemd および名前空間)。

Red Hat OpenStack Platform 11 ではメタデータサービスは、カスタムの Python スクリプトですが、Red Hat OpenStack Platform 13 では HAProxy で、メモリー使用量が 30 パーセント低くなります。多くのユーザーはルーターごとに 1 プロキシを使用し、コントローラーあたりのルーター数が数百もしくは数千にも及ぶため、これは特に重要となります。

第8章 RED HAT OPENSTACK PLATFORM および OPENDAYLIGHT に関する参考資料

コンポーネント	参考資料
OpenDaylight	本書に記載されていない詳しい情報については、OpenDaylight Carbon の ドキュメント を参照してください。
Red Hat OpenDaylight Product Guide	Red Hat OpenDaylight についてと、Red Hat OpenStack Platform との関連についての情報は、『 Red Hat OpenDaylight Product Guide 』を参照してください。
Red Hat Enterprise Linux	Red Hat OpenStack Platform は、Red Hat Enterprise Linux 7.4 でサポートされています。Red Hat Enterprise Linux のインストールについては、『 Red Hat Enterprise Linux インストールガイド 』で対応するインストールガイドを参照してください。
Red Hat OpenStack Platform	<p>OpenStack のコンポーネントとそれらの依存関係をインストールするには、Red Hat OpenStack Platform director を使用します。director は基本的な OpenStack アンダークラウドとして使用して、OpenStack ノードを最終的なオーバークラウド内にプロビジョニングして管理します。デプロイしたオーバークラウドに必要な環境に加えて、アンダークラウドをインストールするための追加のホストマシンが 1 台必要となる点に注意してください。詳しい手順は、『Red Hat OpenStack Platform director のインストールと使用方法』を参照してください。</p> <p>ネットワーク分離、ストレージ設定、SSL 通信、一般的な設定方法など、director を使用した Red Hat OpenStack Platform のエンタープライズ環境向けの高度な機能の設定に関する情報は、『オーバークラウドの高度なカスタマイズ』を参照してください。</p>
NFV のドキュメント	NFV 対応の Red Hat OpenStack Platform のデプロイメントに関する詳しい情報は、『 Network Functions Virtualization Planning and Configuration Guide 』を参照してください。