



# Red Hat OpenStack Platform 13

## ネットワーク機能仮想化 (NFV) のプランニング および設定ガイド

ネットワーク機能仮想化 (NFV) の OpenStack デプロイメントのプランニングと設定



# Red Hat OpenStack Platform 13 ネットワーク機能仮想化 (NFV) のプランニングおよび設定ガイド

---

ネットワーク機能仮想化 (NFV) の OpenStack デプロイメントのプランニングと設定

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、プランニングに関する重要な情報を記載し、Red Hat OpenStack Platform の NFV デプロイメントで SR-IOV と OVS-DPDK を設定する手順を説明します。

## 目次

前書き .....	4
第1章 概要 .....	5
第2章 ハードウェア要件 .....	6
2.1. テスト済みの NIC .....	6
2.2. NUMA ノードのトポロジーについての理解 .....	6
2.3. BIOS 設定の確認 .....	10
第3章 ソフトウェア要件 .....	11
3.1. NFV デプロイメントでサポートされている構成 .....	11
3.2. サポートされているドライバー .....	11
3.3. サードパーティー製のソフトウェアとの互換性 .....	11
第4章 ネットワークの考慮事項 .....	12
第5章 SR-IOV デプロイメントのプランニング .....	13
5.1. SR-IOV デプロイメント向けのハードウェアの分割 .....	13
5.2. NFV SR-IOV デプロイメントのトポロジー .....	13
5.2.1. HCI を使用しない NFV SR-IOV .....	14
第6章 SR-IOV デプロイメントの設定 .....	16
6.1. SR-IOV 設定パラメーターについての概要 .....	16
6.2. OVS ハードウェアオフロード .....	17
6.2.1. VLAN を使用した OVS ハードウェアオフロード対応の SR-IOV の設定 .....	18
6.2.2. VXLAN を使用した OVS ハードウェアオフロード対応の SR-IOV の設定 .....	19
6.3. SR-IOV 用のフレーバーの作成とインスタンスのデプロイ .....	21
第7章 OVS-DPDK デプロイメントのプランニング .....	23
7.1. CPU 分割と NUMA トポロジーを使用する OVS-DPDK .....	23
7.2. ワークフローと派生パラメーターについての概要 .....	23
7.3. OVS-DPDK の派生パラメーター .....	24
7.4. 手動で計算した OVS-DPDK のパラメーターについての概要 .....	25
7.4.1. CPU パラメーター .....	25
7.4.2. メモリーパラメーター .....	26
7.4.3. ネットワークパラメーター .....	28
7.4.4. その他のパラメーター .....	29
7.5. 2 NUMA ノード構成の OVS-DPDK デプロイメントの例 .....	29
7.6. NFV OVS-DPDK デプロイメントのトポロジー .....	31
第8章 OVS-DPDK デプロイメントの設定 .....	34
8.1. ワークフローを使用した DPDK パラメーターの算出 .....	34
8.2. OVS-DPDK のトポロジー .....	37
8.3. OVS-DPDK インターフェースの MTU 値の設定 .....	39
8.4. セキュリティーグループの設定 .....	40
8.5. OVS-DPDK インターフェース向けのマルチキューの設定 .....	41
8.6. 既知の制限事項 .....	41
8.7. OVS-DPDK 用のフレーバーの作成とインスタンスのデプロイ .....	42
8.8. 設定のトラブルシューティング .....	44
第9章 NFV ワークロードに向けた RT-KVM の有効化 .....	46
9.1. RT-KVM コンピュートノードのプランニング .....	46
9.2. RT-KVM 対応の OVS-DPDK の設定 .....	48
9.2.1. ComputeOvsDpdk コンポーザブルロールの生成 .....	48

9.2.2. CPU アフィニティー向けの tuned の設定	48
9.2.3. OVS-DPDK パラメーターの設定	50
9.2.4. コンテナイメージの準備	51
9.2.5. オーバークラウドのデプロイ	51
9.3. RT-KVM インスタンスの起動	51
<b>第10章 例: ODL および VXLAN トンネリングを使用する OVS-DPDK の設定</b>	<b>53</b>
10.1. COMPUTEOVSDPDK コンポーザブルロールの生成	53
10.2. CPU アフィニティー向けの TUNED の設定	53
10.3. OVS-DPDK パラメーターの設定	54
10.4. コントローラーノードの設定	55
10.5. DPDK インターフェイス用のコンピュートノードの設定	57
10.6. オーバークラウドのデプロイ	58
<b>第11章 NFV を実装した RED HAT OPENSTACK PLATFORM のアップグレード</b>	<b>60</b>
<b>第12章 パフォーマンス</b>	<b>61</b>
<b>第13章 その他の参考資料</b>	<b>62</b>
<b>付録A ODL DPDK YAML ファイルのサンプル</b>	<b>63</b>
A.1. VXLAN DPDK ODL YAML ファイルのサンプル	63
A.1.1. post-install.yaml	63
A.1.2. network.environment.yaml	64
A.1.3. controller.yaml	65
A.1.4. compute-ovs-dpdk.yaml	69
A.1.5. overcloud_deploy.sh	72
<b>付録B 改訂履歴</b>	<b>73</b>



## 前書き

Red Hat OpenStack Platform は、Red Hat Enterprise Linux 上にプライベートまたはパブリッククラウドを構築するための基盤を提供します。これにより、スケーラビリティが極めて高く、耐障害性に優れたプラットフォームをクラウド対応のワークロード開発にご利用いただくことができます。

本ガイドでは、Red Hat OpenStack Platform director を使用して、NFV デプロイメント向けに SR-IOV および DPDK データパス対応の Open vSwitch (OVS-DPDK) のプランニングおよび設定を行う方法について説明します。



## 第1章 概要

ネットワーク機能仮想化 (NFV) とは、汎用のクラウドベースのインフラストラクチャー上でネットワーク機能を仮想化するソフトウェアソリューションです。NFV により、通信事業者 (CSP) は従来のハードウェアから離れることができます。

NFV の概念に関する俯瞰的な情報は、『[ネットワーク機能仮想化 \(NFV\) の製品ガイド](#)』を参照してください。



### 注記

OVS-DPDK および SR-IOV の設定は、ハードウェアとトポロジに依存します。本ガイドでは、CPU の割り当て、メモリの確保、NIC の設定の例を紹介します。これらは、トポロジとユースケースによって異なる場合があります。

Red Hat OpenStack Platform director を使用すると、オーバークラウドのネットワークを分離することができます。この機能では、特定のネットワーク種別 (例: 外部、テナント、内部 API など) を分離ネットワークに分けることができます。ネットワークは、単一ネットワークインターフェース上または複数のネットワークインターフェースに分散してデプロイすることが可能です。Open vSwitch では、複数のインターフェースを単一のブリッジに割り当ててボンディングを作成することができます。Red Hat OpenStack Platform のインストールでは、ネットワークの分離はテンプレートファイルを使用して設定されます。テンプレートファイルを指定しない場合には、サービスネットワークはすべてプロビジョニングネットワーク上にデプロイされます。テンプレートの設定ファイルは 3 種類あります。

- **network-environment.yaml**: このファイルには、オーバークラウドノードのネットワーク設定で使用するサブネット、IP アドレス範囲などのネットワークの情報が含まれます。さらに、このファイルには、さまざまなシナリオでできるように、デフォルトのパラメーターの値を上書きする異なる設定も含まれます。
- ホストネットワークのテンプレート (例: **compute.yaml**、**controller.yaml**): オーバークラウドノードのネットワークインターフェース設定を定義します。ネットワーク情報の値は、**network-environment.yaml** ファイルによって提供されます。
- インストール後の設定用のファイル (**post-install.yaml**): インストール後に実行するさまざまな設定のステップを提供します。以下に例を示します。
  - Tuned のインストールと設定。tuned パッケージには、システムコンポーネントの使用状況をモニタリングして、そのモニタリング情報に基づいてシステムの設定を動的にチューニングする **tuned** デーモンが含まれています。OVS-DPDK および SR-IOV のデプロイメントで適切な CPU アフィニティーの設定を指定するには、**tuned cpu-partitioning** プロファイルを使用すべきです。このパッケージに関する詳しい情報は、『[パフォーマンスチューニングガイド](#)』を参照してください。

これらの Heat テンプレートファイルは、アンダークラウドノードの **/usr/share/openstack-tripleo-heat-templates/** にあります。

以下の項では、Red Hat OpenStack Platform director を使用した NFV 用 Heat テンプレートのプランニングおよび設定の方法について説明します。



### 注記

NFV の設定には、YAML ファイルを使用します。YAML ファイル形式に関する基礎的な説明は、『[YAML in a Nutshell](#)』を参照してください。

## 第2章 ハードウェア要件

本項では、NFV に必要なハードウェアの詳細情報を記載します。

「[Red Hat Technologies Ecosystem](#)」を使用し、カテゴリを選んでから製品バージョンを選択して、認定済みハードウェア、ソフトウェア、クラウドプロバイダー、コンポーネントの一覧を確認してください。

Red Hat OpenStack Platform の認定済みハードウェアの完全一覧については「[Red Hat OpenStack Platform certified hardware](#)」を参照してください。

### 2.1. テスト済みの NIC

NFV 向けのテスト済み NIC の一覧は、「[Network Adapter Support](#)」を参照してください (カスタマーポータルログインが必要です)。

### 2.2. NUMA ノードのトポロジーについての理解

デプロイメントを計画する際には、コンピュートノードの NUMA トポロジーを理解した上で CPU とメモリーのリソースを分割し、パフォーマンスを最適化する必要があります。NUMA 情報の特定は、以下の手順で行うことができます。

- ベアメタルノードからこの情報を取得するには、ハードウェアイントロスペクションを有効にします。
- 各ベアメタルノードにログオンして、手動で情報を収集します。



#### 注記

ハードウェアイントロスペクションで NUMA 情報を取得するには、アンダークラウドのインストールと設定が完了している必要があります。詳しくは、『[director のインストールと使用方法](#)』を参照してください。

#### ハードウェアイントロスペクション情報の取得

Bare Metal サービスでは、ハードウェア検査時に追加のハードウェア情報を取得するためのパラメーター (**inspection\_extras**) がデフォルトで有効になっています。これらのハードウェア情報を使って、オーバークラウドを設定することができます。**undercloud.conf** ファイルの **inspection\_extras** パラメーターに関する詳細は、「[director の設定](#)」を参照してください。

たとえば、**numa\_topology** コレクターは、このハードウェア inspection\_extras の一部で、各 NUMA ノードに関する以下の情報が含まれます。

- RAM (キロバイト単位)
- 物理 CPU コアおよびそのシブリングスレッド
- NUMA ノードに関連付けられた NIC

この情報を取得するには、ベアメタルノードの **UUID** を指定して、**openstack baremetal introspection data save \_UUID\_ | jq .numa\_topology** コマンドを実行します。

取得されるベアメタルノードの NUMA 情報の例を、以下に示します。

```
{
```

```
"cpus": [  
  {  
    "cpu": 1,  
    "thread_siblings": [  
      1,  
      17  
    ],  
    "numa_node": 0  
  },  
  {  
    "cpu": 2,  
    "thread_siblings": [  
      10,  
      26  
    ],  
    "numa_node": 1  
  },  
  {  
    "cpu": 0,  
    "thread_siblings": [  
      0,  
      16  
    ],  
    "numa_node": 0  
  },  
  {  
    "cpu": 5,  
    "thread_siblings": [  
      13,  
      29  
    ],  
    "numa_node": 1  
  },  
  {  
    "cpu": 7,  
    "thread_siblings": [  
      15,  
      31  
    ],  
    "numa_node": 1  
  },  
  {  
    "cpu": 7,  
    "thread_siblings": [  
      7,  
      23  
    ],  
    "numa_node": 0  
  },  
  {  
    "cpu": 1,  
    "thread_siblings": [  
      9,  
      25  
    ],  
    "numa_node": 1  
  }  
]
```

```
},
{
  "cpu": 6,
  "thread_siblings": [
    6,
    22
  ],
  "numa_node": 0
},
{
  "cpu": 3,
  "thread_siblings": [
    11,
    27
  ],
  "numa_node": 1
},
{
  "cpu": 5,
  "thread_siblings": [
    5,
    21
  ],
  "numa_node": 0
},
{
  "cpu": 4,
  "thread_siblings": [
    12,
    28
  ],
  "numa_node": 1
},
{
  "cpu": 4,
  "thread_siblings": [
    4,
    20
  ],
  "numa_node": 0
},
{
  "cpu": 0,
  "thread_siblings": [
    8,
    24
  ],
  "numa_node": 1
},
{
  "cpu": 6,
  "thread_siblings": [
    14,
    30
  ],
  "numa_node": 1
}
```

```
    },
    {
      "cpu": 3,
      "thread_siblings": [
        3,
        19
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        2,
        18
      ],
      "numa_node": 0
    }
  ],
  "ram": [
    {
      "size_kb": 66980172,
      "numa_node": 0
    },
    {
      "size_kb": 67108864,
      "numa_node": 1
    }
  ],
  "nics": [
    {
      "name": "ens3f1",
      "numa_node": 1
    },
    {
      "name": "ens3f0",
      "numa_node": 1
    },
    {
      "name": "ens2f0",
      "numa_node": 0
    },
    {
      "name": "ens2f1",
      "numa_node": 0
    },
    {
      "name": "ens1f1",
      "numa_node": 0
    },
    {
      "name": "ens1f0",
      "numa_node": 0
    },
    {
      "name": "eno4",
      "numa_node": 0
    }
  ]
}
```

```
    },  
    {  
      "name": "eno1",  
      "numa_node": 0  
    },  
    {  
      "name": "eno3",  
      "numa_node": 0  
    },  
    {  
      "name": "eno2",  
      "numa_node": 0  
    }  
  ]  
}
```

## 2.3. BIOS 設定の確認

以下のリストには、NFV に必要な BIOS 設定を記載します。

- **C3 Power State:** 無効
- **C6 Power State:** 無効
- **MLC Streamer:** 有効
- **MLC Spacial Prefetcher:** 有効
- **DCU Data Prefetcher:** 有効
- **DCA:** 有効
- **CPU Power and Performance:** Performance.
- **Memory RAS and Performance Config → NUMA Optimized:** 有効
- **Turbo Boost:** 無効

## 第3章 ソフトウェア要件

本項では、サポートされている設定とドライバー、および NFV に必要なサブスクリプションの詳細について説明します。

Red Hat OpenStack Platform をインストールするには、OpenStack 環境にある全システムを Red Hat サブスクリプションマネージャーで登録して、必要なチャンネルをサブスクライブします。詳しくは、[「アンダークラウドの登録と更新」](#) を参照してください。

### 3.1. NFV デプロイメントでサポートされている構成

Red Hat OpenStack Platform は、director を使用した SR-IOV および OVS-DPDK のインストール向けの NFV デプロイメントをサポートしています。Red Hat OpenStack Platform director で利用可能なコンポーザブルロールを使用して、カスタムのデプロイメントロールを作成できます。今回のリリースではサポートが制限されていますが、ハイパーコンバージドインフラストラクチャー (HCI) が提供されており、分散型の NFV 向けにコンピューターノードと Red Hat Ceph Storage ノードを同じ場所に配置することができます。HCI のパフォーマンスを向上するために、CPU ピニングを使用します。HCI モデルでは、NFV のユースケースにおいてより効率的な管理を行うことができます。また、今回のリリースでは、サポート対象の機能として OpenDaylight および Real-Time KVM が提供されています。OpenDaylight とは、Software-Defined Network (SDN) デプロイメントに向けた、オープンソースのモジュール型マルチプロトコルコントローラーです。また、本リリースには OVS ハードウェアオフロードの機能がテクノロジープレビューとして含まれています。テクノロジープレビューとして提供されている機能のサポート範囲に関する詳しい情報は、[「テクノロジープレビュー機能のサポート範囲」](#) を参照してください。

### 3.2. サポートされているドライバー

サポートされるドライバーの完全な一覧は [「Red Hat OpenStack Platform におけるコンポーネント、プラグイン、およびドライバーのサポート」](#) を参照してください。

Red Hat OpenStack の NFV デプロイメント向けにテスト済みの NIC の一覧は、[「テスト済みの NIC」](#) を参照してください。

### 3.3. サードパーティー製のソフトウェアとの互換性

Red Hat のテクノロジー (Red Hat OpenStack Platform) で機能することを検証、サポート、認定済みの製品およびサービスの完全な一覧は、[Red Hat OpenStack Platform と互換性のあるサードパーティー製のソフトウェア](#) の情報を参照してください。製品バージョンやソフトウェアカテゴリ別に一覧をフィルタリングすることができます。

Red Hat のテクノロジー (Red Hat Enterprise Linux) で機能することを検証、サポート、認定済みの製品およびサービスの完全な一覧は、[Red Hat Enterprise Linux と互換性のあるサードパーティー製のソフトウェア](#) の情報を参照してください。製品バージョンやソフトウェアカテゴリ別に一覧をフィルタリングすることができます。

## 第4章 ネットワークの考慮事項

アンダークラウドのホストには、最低でも以下のネットワークが必要です。

- プロビジョニングネットワーク: オーバークラウドで利用できるベアメタルシステムの検出に役立つ DHCP および PXE ブート機能を提供します。
- 外部ネットワーク: 全ノードへのリモート接続に使用する別個のネットワーク。このネットワークに接続するこのインターフェースには、静的または外部の DHCP サービス経由で動的に定義された、ルーティング可能な IP アドレスが必要です。

最小のオーバークラウドの構成は、以下のとおりです。

- シングル NIC 構成: ネイティブの VLAN および異なる種別のオーバークラウドネットワークのサブネットを使用するタグ付けされた VLAN 上にプロビジョニングネットワーク用の NIC を 1 つ。
- デュアル NIC 構成: プロビジョニングネットワーク用の NIC を 1 つと、外部ネットワーク用の NIC を 1 つ。
- デュアル NIC 構成: ネイティブの VLAN 上にプロビジョニングネットワーク用の NIC を 1 つと、異なる種別のオーバークラウドネットワークのサブネットを使用するタグ付けされた VLAN 用の NIC を 1 つ。
- 複数 NIC 構成: 各 NIC は、異なる種別のオーバークラウドネットワークのサブセットを使用します。

ネットワーク要件の詳しい情報は「[ネットワーク要件](#)」を参照してください。



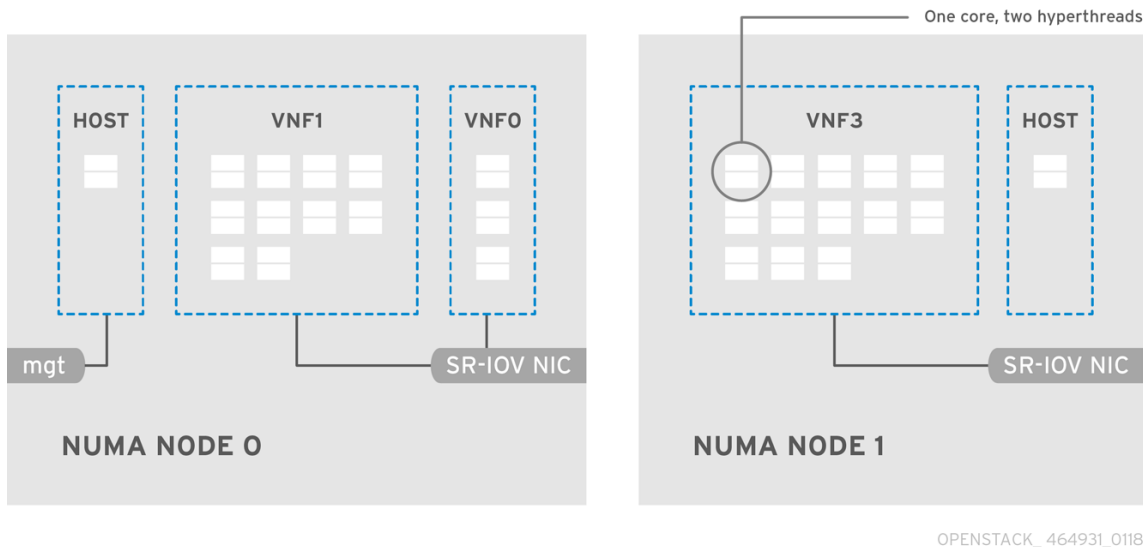
## 第5章 SR-IOV デプロイメントのプランニング

NFV 向けの SR-IOV デプロイメントを最適化するには、コンピュータノードのハードウェアに応じて、個別の OVS-DPDK パラメーターを設定する方法を理解しておく必要があります。

SR-IOV パラメーターに対するハードウェアの影響を評価するには、[「NUMA ノードトポロジーについての理解」](#) を参照してください。

### 5.1. SR-IOV デプロイメント向けのハードウェアの分割

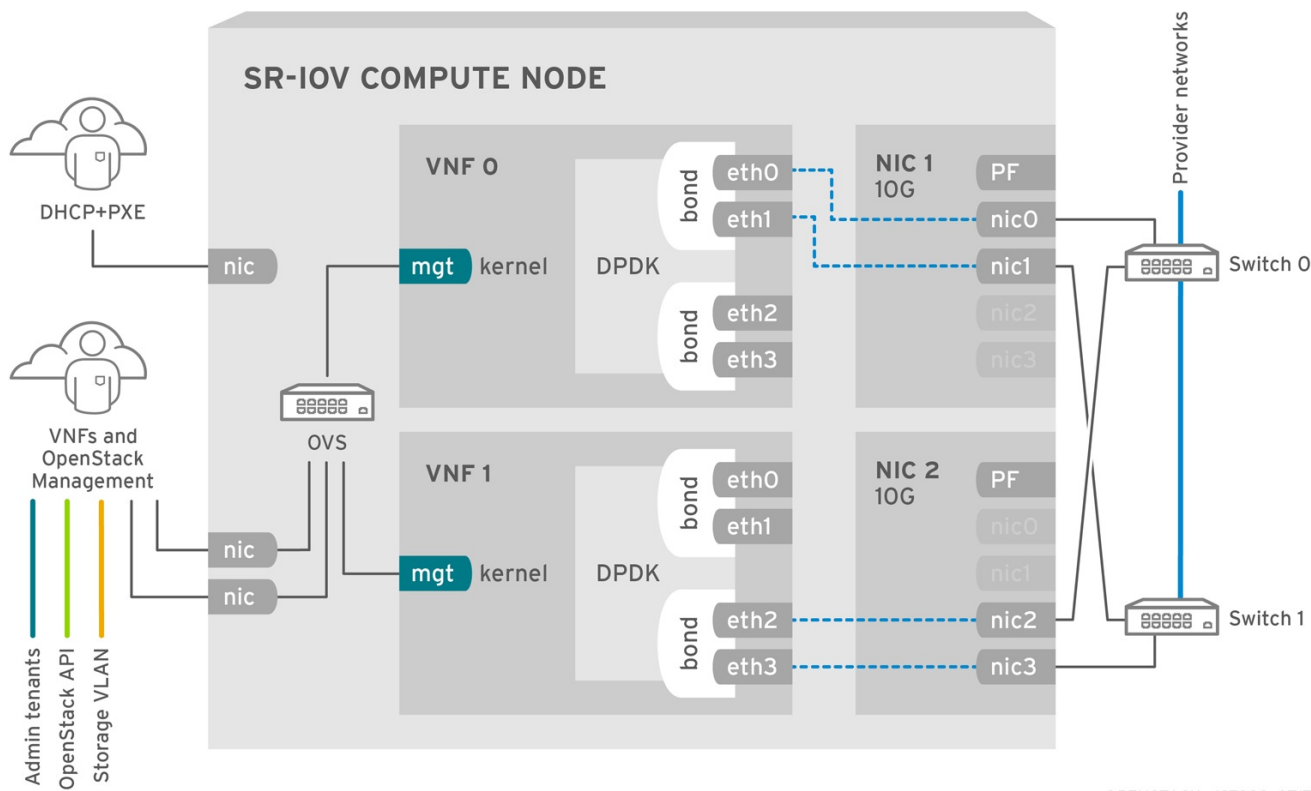
SR-IOV で高パフォーマンスを実現するには、ホストとゲストの間でリソースを分割する必要があります。



標準的なトポロジーでは、デュアルコアソケットのコンピュータノード上の NUMA ノードにはそれぞれ 14 のコアが実装されます。HT (ハイパースレッド) および非 HT のコアがサポートされています。各コアには 2 つのシプリングスレッドがあります。1 つのコアは、各 NUMA ノード上のホスト専用です。VNF は SR-IOV インターフェースのボンディングを処理します。すべての割り込み要求 (IRQ) はホストのコア上でルーティングされます。VNF コアは VNF 専用です。これらのコアは、他の VNF からの分離と、ホストからの分離を提供します。各 VNF は単一の NUMA ノード上のリソースを使用する必要があります。VNF によって使用される SR-IOV NIC はその同じ NUMA ノードに関連付ける必要があります。このトポロジーでは、仮想化のオーバーヘッドはありません。ホスト、OpenStack Networking (neutron)、および Compute (nova) の設定パラメーターは単一のファイルで公開されるので、管理が簡単で、整合性を保つことができます。また、プリエンプションやパケット損失の原因となり、分離を適切に行うにあたって致命的となる一貫性の欠如を回避します。ホストと仮想マシンの分離は、**tuned** プロファイルに依存します。このプロファイルは、分離する CPU の一覧に基づいて、ブートパラメーターや OpenStack の変更を管理します。

### 5.2. NFV SR-IOV デプロイメントのトポロジー

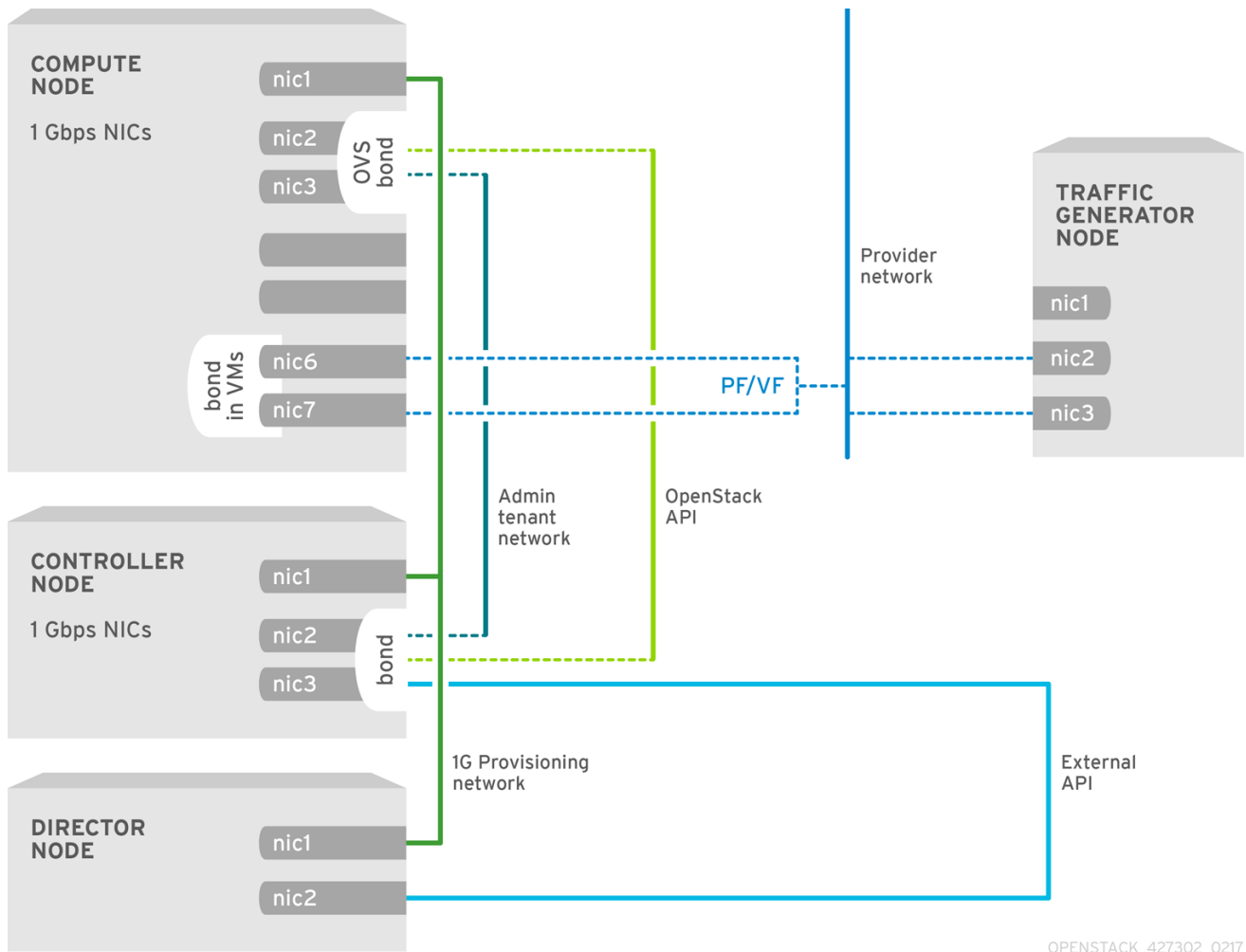
以下の図には、2 つの VNF が示されています。各 VNF には、それぞれに **mgt** で示された管理インターフェースがあります。管理インターフェースは **ssh** アクセスなどを管理します。データプレーンインターフェースは VNF を DPDK にボンディングして、高可用性を確保します (VNF は DPDK ライブラリーを使用してデータプレーンインターフェースをボンディングします)。この図には、2 つの重複したプロバイダーネットワークも示されています。コンピュータノードには 2 つの標準 NIC がボンディングされ、VNF 管理と Red Hat OpenStack Platform API 管理の間で共有されています。



この図では、アプリケーションレベルで DPDK を活用し、SR-IOV VF/PF へのアクセスが可能な VNF を示しています。これらの両方を実装することにより、可用性またはパフォーマンスが向上します (ファブリックの設定に依存)。DPDK はパフォーマンスを向上させる一方、VF/PF DPDK のボンディングはフェイルオーバーに対応します (可用性)。VNF ベンダーは、DPDK PMD ドライバーが VF/PF として公開される SR-IOV カードを必ずサポートするようする必要があります。また、管理ネットワークは OVS を使用するので、VNF は標準の VirtIO ドライバーを使用する「mgmt」ネットワークデバイスを認識します。オペレーターは、VNF への初回の接続にそのデバイスを使用して、DPDK アプリケーションが 2 つの VF/PF を適切にボンディングすることができます。

### 5.2.1. HCI を使用しない NFV SR-IOV

以下の図には、NFV ユースケース向けの非 HCI の SR-IOV のトポロジーを示しています。この環境は、1 Gbps の NIC を搭載したコンピュータノードおよびコントローラーノードと、director ノードで構成されます。



## 第6章 SR-IOV デプロイメントの設定

本項では、Red Hat OpenStack 向けの Single Root Input/Output Virtualization (SR-IOV) の設定方法について説明します。

オーバークラウドをデプロイする前に、アンダークラウドのインストールと設定が完了している必要があります。詳しくは、『[director のインストールと使用方法](#)』を参照してください。

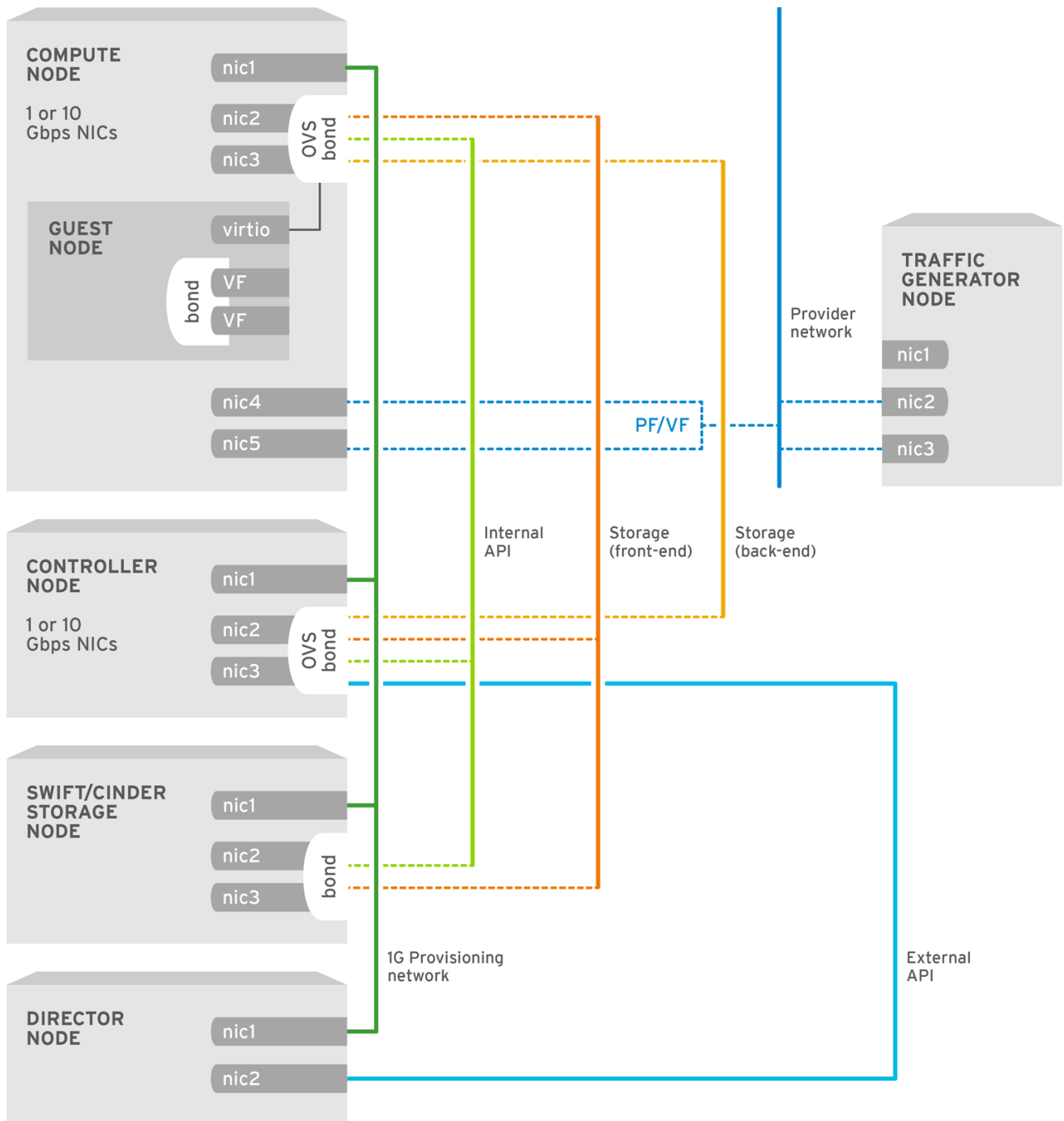


### 注記

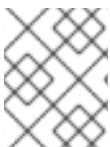
これらの director Heat テンプレートによって変更されている **etc/tuned/cpu-partitioning-variables.conf** の **isolated\_cores** またはその他の値は編集/変更しないでください。

### 6.1. SR-IOV 設定パラメーターについての概要

**network-environment.yaml** ファイルを更新して、カーネルの引数、SR-IOV ドライバー、PCI パススルーなどのパラメーターを追加します。また、**compute.yaml** ファイルも更新して SR-IOV インターフェースのパラメーターを追加してから、**overcloud\_deploy.sh** スクリプトを実行して、その SR-IOV パラメーターを使用してオーバークラウドをデプロイする必要もあります。



OPENSTACK\_450694\_0617



### 注記

本ガイドでは、CPU の割り当て、メモリーの確保、NIC の設定の例を紹介します。これらは、トポロジーとユースケースによって異なる場合があります。

## 6.2. OVS ハードウェアオフロード

OVS ハードウェアオフロードを対応の SR-IOV の設定には、ネットワーク設定テンプレートに特定の変更を加える必要はありません。本項では、OVS ハードウェアオフロードデプロイメント固有の変更について説明します。以下の手順では、一般的なネットワーク分離のデプロイメントに必要なその他のパラメーターおよびネットワーク設定テンプレートファイルについては説明していません。詳しくは、「[5章SR-IOV デプロイメントのプランニング](#)」を参照してください。

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。

### 6.2.1. VLAN を使用した OVS ハードウェアオフロード対応の SR-IOV の設定

**ComputeSriov** ロールで OpenvSwitch (OVS) ハードウェアオフロードを有効化するには、以下の手順を実行します。

1. **ComputeSriov** ロールを作成します。

```
# openstack overcloud roles generate -o roles_data.yaml Controller
ComputeSriov
```

2. **openstack overcloud deploy** コマンドに環境ファイルを追加して、OVS ハードウェアオフロードとその依存関係を有効化します。

```
# Enables OVS Hardware Offload in the ComputeSriov role
/usr/share/openstack-tripleo-heat-templates/environments/ovs-hw-
offload.yaml

# Applies the KernelArgs and TunedProfile with a reboot
/usr/share/openstack-tripleo-heat-templates/environments/host-
config-and-reboot.yaml
```

3. 以下の環境ファイルを **openstack overcloud deploy** コマンドに追加して、ML2-ODL を有効化します。

```
#Enables ml2-odl with SR-IOV deployment
/usr/share/openstack-tripleo-heat-templates/environments/services-
docker/neutron-opensdaylight.yaml
```



#### 注記

The OVS ハードウェアオフロードの環境ファイルは、すべての neutron ML2 プラグインで共通しています。デプロイメント用のデフォルトの ML2 プラグインは ML2-OVS ですが、NFV デプロイメントには ML2-ODL を使用すべきです。

4. **sriov-environment.yaml** という名前の環境ファイルに SR-IOV ノードのパラメーターを設定します。

```
parameter_defaults:
  NeutronTunnelTypes: ''
  NeutronNetworkType: 'vlan'
  NeutronBridgeMappings:
    - <network_name>:<bridge_name>
  NeutronNetworkVLANRanges:
    - <network_name>:<vlan_ranges>
  NeutronSriovNumVFs:
    - <interface_name>:<number_of_vfs>:switchdev
  NeutronPhysicalDevMappings:
    - <network_name>:<interface_name>
```

```
NovaPCIPassthrough:
- devname: <interface_name>
  physical_network: <network_name>
```



### 注記

SR-IOV ノードの設定および要件に応じて、<network\_name>、<interface\_name>、<number\_of\_vfs> を設定します。

ML2-ODL でデプロイする場合

```
THT_ROOT="/usr/share/openstack-tripleo-heat-templates/"
openstack overcloud deploy --templates \
  -r roles_data.yaml \
  -e $THT_ROOT/environments/ovs-hw-offload.yaml \
  -e $THT_ROOT/environments/services-docker/neutron-opendaylight.yaml \
  -e $THT_ROOT/environments/host-config-and-reboot.yaml \
  -e sriov-environment.yaml \
  -e <<network isolation and network config environment files>>
```

ML2-OVS でデプロイする場合

```
THT_ROOT="/usr/share/openstack-tripleo-heat-templates/"
openstack overcloud deploy --templates \
  -r roles_data.yaml \
  -e $THT_ROOT/environments/ovs-hw-offload.yaml \
  -e $THT_ROOT/environments/host-config-and-reboot.yaml \
  -e sriov-environment.yaml \
  -e <<network isolation and network config environment files>>
```

## 6.2.2. VXLAN を使用した OVS ハードウェアオフロード対応の SR-IOV の設定

**ComputeSriov** ロールで OpenvSwitch (OVS) ハードウェアオフロードを有効化するには、以下の手順を実行します。

1. **ComputeSriov** ロールを作成します。

```
# openstack overcloud roles generate -o roles_data.yaml Controller
ComputeSriov
```

2. **openstack overcloud deploy** コマンドに環境ファイルを追加して、OVS ハードウェアオフロードとその依存関係を有効化します。

```
# Enables OVS Hardware Offload in the ComputeSriov role
/usr/share/openstack-tripleo-heat-templates/environments/ovs-hw-offload.yaml

# Applies the KernelArgs and TunedProfile with a reboot
/usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml
```

3. SR-IOV インターフェース上で以下のネットワーク設定変更を適用します。

■

```
- type: interface
  name: <interface_name>
  addresses:
    - ip_netmask:
      get_param: TenantIpSubnet
```



### 注記

Mellanox ドライバーは、OVS ブリッジの VLAN インターフェース上の VXLAN トンネルで問題があります (single-nic-vlans ネットワーク設定を使用する VXLAN 環境と同様)。Mellanox ドライバーでこの問題が解決するまでは、(OVS ブリッジ上で VLAN インターフェースを使用する代わりに) VXLAN ネットワークをインターフェース上または OVS ブリッジ上に直接設定することができません。

4. 以下の環境ファイルを **openstack overcloud deploy** コマンドに追加して、ML2-ODL を有効化します。

```
#Enables ml2-odl with SR-IOV deployment
/usr/share/openstack-tripleo-heat-templates/environments/services-
docker/neutron-opendaylight.yaml
```



### 注記

The OVS ハードウェアオフロードの環境ファイルは、すべての neutron ML2 プラグインで共通しています。デプロイメント用のデフォルトの ML2 プラグインは ML2-OVS ですが、NFV デプロイメントには ML2-ODL を使用すべきです。

5. VXLAN デプロイメントには、**sriov-environment.yaml** という名前の環境ファイルに SR-IOV ノードのパラメーターを設定します。

```
parameter_defaults:
  NeutronSriovNumVFs:
    - <interface_name>:<number_of_vfs>:switchdev
  NovaPCIPassthrough:
    - devname: <interface_name>
      physical_network: null
```



### 注記

SR-IOV ノードの設定および要件に応じて、<network\_name>、<interface\_name>、<number\_of\_vfs> を設定します。

## ML2-ODL でデプロイする場合

```
THT_ROOT="/usr/share/openstack-tripleo-heat-templates/"
openstack overcloud deploy --templates \
  -r roles_data.yaml \
  -e $THT_ROOT/environments/ovs-hw-offload.yaml \
  -e $THT_ROOT/environments/services-docker/neutron-opendaylight.yaml \
```



```
-e $THT_ROOT/environments/host-config-and-reboot.yaml \
-e sriov-environment.yaml \
-e <<network isolation and network config environment files>>
```

ML2-OVS でデプロイする場合

```
THT_ROOT="/usr/share/openstack-tripleo-heat-templates/"
openstack overcloud deploy --templates \
  -r roles_data.yaml \
  -e $THT_ROOT/environments/ovs-hw-offload.yaml \
  -e $THT_ROOT/environments/host-config-and-reboot.yaml \
  -e sriov-environment.yaml \
  -e <<network isolation and network config environment files>>
```

### 6.3. SR-IOV 用のフレーバーの作成とインスタンスのデプロイ

NFV を実装する Red Hat OpenStack Platform デプロイメントの SR-IOV の設定を完了した後は、以下の手順に従ってフレーバーを作成してインスタンスをデプロイします。

1. アグリゲートグループを作成して、SR-IOV 用にホストを追加します。

```
# openstack aggregate create --zone=sriov sriov
# openstack aggregate add host sriov compute-sriov-0.localdomain
```



#### 注記

CPU ピニングされたインスタンスをピンングされていないインスタンスと分けるには、ホストアグリゲートを使用すべきです。CPU ピニングを使用していないインスタンスは、CPU ピニングを使用するインスタンスのリソース要件は順守しません。

2. フレーバーを作成します。

```
# openstack flavor create compute --ram 4096 --disk 150 --vcpus 4
```

**compute** はフレーバー名、**4096** は MB 単位のメモリー容量、**150** は GB 単位のディスク容量 (デフォルトでは 0 G)、**4** は仮想 CPU 数を設定しています。

3. フレーバーの追加のプロパティを設定します。

```
# openstack flavor set --property hw:cpu_policy=dedicated --property
hw:mem_page_size=1GB compute
```

**compute** はフレーバー名で、それ以外のパラメーターはそのフレーバーのその他のプロパティを設定します。

4. ネットワークを作成します。

```
# openstack network create net1 --provider-physical-network tenant -
-provider-network-type vlan --provider-segment <VLAN-ID>
```

5. ポートを作成します。

- a. **vnic-type direct** を使用して SR-IOV VF ポートを作成します。

```
# openstack port create --network net1 --vnic-type direct  
sriov_port
```

- b. **vnic-type direct-physical** を使用して SR-IOV PF ポートを作成します。

```
# openstack port create --network net1 --vnic-type direct-  
physical sriov_port
```

6. インスタンスをデプロイします。

```
# openstack server create --flavor compute --availability-zone sriov  
--image rhel_7.3 --nic port-id=sriov_port sriov_vm
```

ここで

- **compute** はフレーバー名または ID です。
- **sriov** はサーバーのアベイラビリティゾーンです。
- **rhel\_7.3** はインスタンスの作成に使用するイメージ (名前または ID) です。
- **sriov\_port** はサーバー上の NIC です。
- **sriov\_vm** はインスタンスの名前です。

これで、NFV ユースケースの SR-IOV 向けインスタンスのデプロイが完了しました。

## 第7章 OVS-DPDK デプロイメントのプランニング

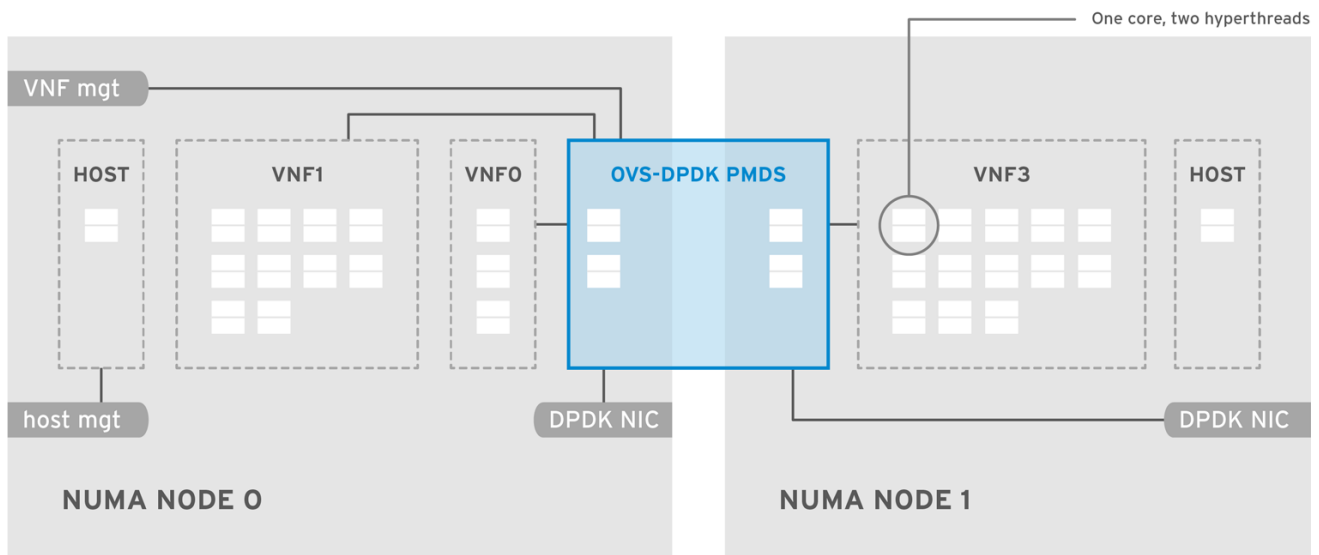
NFV 向けの OVS-DPDK デプロイメントを最適化するには、OVS-DPDK がコンピューターノードのハードウェア (CPU、NUMA ノード、メモリー、NIC) をどのように使用するかと、コンピューターノードに応じた OVS-DPDK の各パラメーターを決定するにあたっての考慮事項を理解しておくべきです。

CPU と NUMA トポロジーの概要は「[NFV のパフォーマンスの考慮事項](#)」を参照してください。

### 7.1. CPU 分割と NUMA トポロジーを使用する OVS-DPDK

OVS-DPDK はホスト、ゲスト、および OVS-DPDK 自体用にハードウェアリソースを分割します。OVS-DPDK Poll Mode Driver (PMD) は、専用のコアを必要とする DPDK アクティブループを実行します。これは、CPU 一覧とヒューズページが OVS-DPDK で専用であることを意味します。

サンプルの分割では、デュアルソケットのコンピューターノード上の 1 NUMA ノードにつき 16 コアが含まれます。ホストと OVS-DPDK では NIC を共有できないので、このトラフィックには追加の NIC が必要です。



OPENSTACK\_464931\_0118



#### 注記

NUMA ノードに DPDK NIC が関連付けられていない場合でも、両方の NUMA ノードで DPDK PMD スレッドを確保する必要があります。

OVS-DPDK のパフォーマンスは、NUMA ノードにローカルなメモリーブロックの確保にも左右されます。メモリーと CPU ピニングに使用する同じ NUMA ノードに関連付けられた NIC を使用してください。また、ボンディングを構成する両方のインターフェースには、同じ NUMA ノード上の NIC を必ず使用してください。

### 7.2. ワークフローと派生パラメーターについての概要



#### 重要

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。

OpenStack Workflow (mistral) サービスを使用すると、利用可能なベアメタルノードのキャパビリティに基づいてパラメーターを派生することができます。Openstack Workflow は .yaml ファイルを使用して実行するタスクとアクションのセットを定義します。tripleo-common/workbooks/ ディレクトリーにある **derive\_params.yaml** という事前定義済みのワークブックを使用することができます。このワークブックは、ベアメタルのイントロスペクションで取得した結果から、サポートされる各パラメーターを派生するワークフローを提供します。**derive\_params.yaml** のワークフローは、tripleo-common/workbooks/derive\_params\_formulas.yaml の計算式を使用して、派生パラメーターを計算します。



#### 注記

**derive\_params\_formulas.yaml** の計算式は、お使いの環境に応じて変更することができます。

**derive\_params.yaml** ワークブックは、**given composable** ロール用の全ノードのハードウェア仕様が同じであることを前提としています。ワークフローは、フレーバーとプロファイルの関連付けと、nova の配置スケジューラーを考慮して、ロールに関連付けられたノードを照合し、そのロールと一致する最初のロールのイントロスペクションデータを使用します。

OpenStack のワークフローに関する詳細は、[「workflow および execution のトラブルシューティング」](#)を参照してください。

**-p** または **--plan-environment-file** オプションを使用して、カスタムの **plan\_environment.yaml** ファイルを **openstack overcloud deploy** コマンドに追加することができます。カスタムの **plan\_environment.yaml** ファイルは、ワークブックの一覧と、ワークブックに渡す値を指定します。トリガーされるワークフローは派生パラメーターをマージしてカスタムの **plan\_environment.yaml** に戻し、オーバークラウドのデプロイメントに利用できるようになります。これらの派生パラメーターの結果を使用して、オーバークラウドのイメージを準備することができます。

デプロイメントでの **--plan-environment-file** オプションの使用方法に関する詳しい情報は、[『プランの環境メタデータ』](#)を参照してください。

## 7.3. OVS-DPDK の派生パラメーター

**derive\_params.yaml** のワークフローは、**ComputeNeutronOvsDpdk** サービスを使用する、対応するロールに関連付けられた DPDK パラメーターを派生します。

ワークフローによって、自動的に派生できる OVS-DPDK のパラメーターの一覧は以下のとおりです。

- IsolCpusList
- KernelArgs
- NovaReservedHostMemory
- NovaVcpuPinSet
- OvsDpdkCoreList
- OvsDpdkSocketMemory
- OvsPmdCoreList

**OvsDpdkMemoryChannels** パラメーターは、イントロスペクションのメモリーバンクデータからは派生できません。これは、メモリースロット名の形式がハードウェア環境によって異なるためです。

大半の場合には、**OvsDpdkMemoryChannels** は 4 (デフォルト) です。ハードウェアのマニュアルを参照して 1 ソケットあたりのメモリーチャンネル数を確認し、その値でデフォルト値をオーバーライドしてください。

設定の詳細については、「[ワークフローを使用した DPDK パラメーターの算出](#)」を参照してください。

## 7.4. 手動で計算した OVS-DPDK のパラメーターについての概要

本項では、OVS-DPDK が director の **network\_environment.yaml** HEAT テンプレート内のパラメーターを使用して CPU とメモリーを設定し、パフォーマンスを最適化する方法について説明します。この情報は、コンピュートノード上のハードウェアサポートと、そのハードウェアを分割して OVS-DPDK デプロイメントを最適化する最も有効な方法を評価するのに使用してください。



### 注記

**derived\_parameters.yaml** ワークフローを使用してこれらのパラメーターの値を自動生成した場合には、手動で計算する必要はありません。「[ワークフローと派生パラメーターについての概要](#)」を参照してください。



### 注記

CPU コアを割り当てる際には必ず、同じ物理コア上の CPU シブリングスレッド (論理 CPU) をペアにしてください。

コンピュートノード上の CPU と NUMA ノードを特定するには、「[NUMA ノードのトポロジーについての理解](#)」を参照してください。この情報を使用して、CPU と他のパラメーターをマッピングして、ホスト、ゲストインスタンス、OVS-DPDK プロセスのニーズに対応します。

### 7.4.1. CPU パラメーター

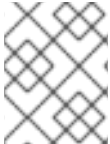
OVS-DPDK は以下の CPU 分割パラメーターを使用します。

#### OvsPmdCoreList

DPDK Poll Mode Driver (PMD) に使用する CPU コアを提供します。DPDK インターフェースのローカルの NUMA ノードに関連付けられた CPU コアを選択します。**OvsPmdCoreList** は、Open vSwitch の **pmd-cpu-mask** の値に使用されます。

- シブリングスレッドをペアにします。
- **OvsDpdkCoreList** からすべてのコアを除外します。
- 両方の NUMA ノード上の 1 番目の物理コアの論理 CPU (両方のスレッドシブリング) が割り当てられないようにしてください。これらは、**OvsDpdkCoreList** パラメーターに使用する必要があります。
- パフォーマンスは、この PMD コアリストに割り当てられている物理コアの数によって異なります。DPDK 用の NIC に関連付けられている NUMA ノードで、必要なコアを割り当てます。
- DPDK 用の NIC が 1 つある NUMA ノードの場合:

- パフォーマンス要件に基づいて、必要な物理コア数を決定し、各物理コアに全シブリングスレッド (論理 CPU) を追加します。
- DPDK 用の NIC がない NUMA ノードの場合:
  - 1 つの物理コアのシブリングスレッド (論理 CPU) を割り当てます (NUMA ノードの 1 番目の物理コアを除く)。DPDK 用の NIC がない場合でも、ゲストインスタンス作成が失敗するのを回避するために、NUMA ノード上に最小限の DPDK Poll Mode Driver が必要です。



### 注記

NUMA ノードに DPDK NIC が関連付けられていない場合でも、両方の NUMA ノードで DPDK PMD スレッドを確保する必要があります。

## NovaVcpuPinSet

CPU ピニング用のコアを設定します。コンピュートノードは、ゲストインスタンスにこれらのコアを使用します。**NovaVcpuPinSet** は **nova.conf** ファイルの **vcpu\_pin\_set** 値として使用されます。

- **OvsPmdCoreList** と **OvsDpdkCoreList** からすべてのコアを除外します。
- 残りのコアをすべて追加します。
- シブリングスレッドをペアにします。

## IsolCpusList

ホストのプロセスから分離される CPU コアのセット。このパラメーターは、**tuned-profiles-cpu-partitioning** コンポーネント用の **cpu-partitioning-variable.conf** ファイルの **isolated\_cores** 値として使用されます。

- **OvsPmdCoreList** と **NovaVcpuPinSet** のコアー一覧が一致するようにします。
- シブリングスレッドをペアにします。

## OvsDpdkCoreList

handler および revalidator スレッドなどの、データパス以外の OVS-DPDK プロセス用の CPU コアを提供します。このパラメーターは、マルチ NUMA ノードハードウェア上でのデータパスの全体的なパフォーマンスには影響は及ぼしません。このパラメーターは Open vSwitch の **dpdk-lcore-mask** 値に使用され、それらのコアはホストと共有されます。

- 各 NUMA ノードから、1 番目のコア (およびシブリングスレッド) を割り当てます (NUMA に関連付けられている DPDK 用の NIC がない場合も)。
- これらのコアは、**OvsPmdCoreList** および **NovaVcpuPinSet** のコアの一覧と相互に排他的である必要があります。

## 7.4.2. メモリーパラメーター

OVS-DPDK は、以下のメモリーパラメーターを使用します。

### OvsDpdkMemoryChannels

NUMA ノードごとに、CPU 内のメモリーチャネルをマッピングします。**OvsDpdkMemoryChannels** パラメーターは Open vSwitch により **other\_config:dpgk-extra="-n <value>"** 値として使用されます。

- **dmidecode -t memory** のコマンドを使用するか、お使いのハードウェアのマニュアルを参照して、利用可能なメモリーチャネルの数を確認します。
- **ls /sys/devices/system/node/node\* -d** のコマンドで NUMA ノードの数を確認します。
- 利用可能なメモリーチャネル数を NUMA ノード数で除算します。

### NovaReservedHostMemory

ホスト上のタスク用にメモリーを MB 単位で確保します。この値は、コンピュートノードにより **nova.conf** の **reserved\_host\_memory\_mb** 値として使用されます。

- 静的な推奨値 4096 MB を使用します。

### OvsDpdkSocketMemory

NUMA ノードごとにヒュージページプールから事前に割り当てるメモリーの容量を MB 単位で指定します。この値は、Open vSwitch により **other\_config:dpgk-socket-mem** 値として使用されます。

- コンマ区切りリストで指定します。**OvsDpdkSocketMemory** 値は NUMA ノード上の各 NIC の MTU 値から計算されます。
- DPDK NIC のない NUMA ノードの場合は、推奨される静的な値である 1024 MB (1GB) を使用します。
- **OvsDpdkSocketMemory** の値は、以下の等式で概算します。
  - $\text{MEMORY\_REQD\_PER\_MTU} = (\text{ROUNDUP\_PER\_MTU} + 800) * (4096 * 64) \text{ Bytes}$ 
    - 800 はオーバーヘッドの値です。
    - $4096 * 64$  は mempool 内のパケット数です。
- NUMA ノードで設定される各 MTU 値の **MEMORY\_REQD\_PER\_MTU** を追加し、バッファとして 512 MB をさらに加算します。その値を 1024 の倍数に丸めます。

### 計算例: MTU 2000 および MTU 9000

DPDK NIC dpdk0 と dpdk1 は同じ NUMA ノード上にあり、それぞれ MTU 9000 と MTU 2000 で設定されています。必要なメモリーを算出する計算例を以下に示します。

1. MTU 値を 1024 バイトの倍数に丸めます。

MTU 値 9000 は、丸めると 9216 バイトになります。  
MTU 値 2000 は、丸めると 2048 バイトになります。

2. それらの丸めたバイト値に基づいて、各 MTU 値に必要なメモリーを計算します。

MTU 値 9000 に必要なメモリー =  $(9216 + 800) * (4096 * 64) = 2625634304$   
MTU 値 2000 に必要なメモリー =  $(2048 + 800) * (4096 * 64) = 746586112$

3. それらを合わせた必要なメモリーの合計を計算します (バイト単位)。

```
2625634304 + 746586112 + 536870912 = 3909091328 バイト
```

この計算は、(MTU 値 9000 に必要なメモリー) + (MTU 値 2000 に必要なメモリー) + (512 MB バッファ) を示しています。

4. 必要合計メモリーを MB に変換します。

```
3909091328 / (1024*1024) = 3728 MB
```

5. この値を 1024 の倍数に丸めます。

```
3728 MB を丸めると 4096 MB になります。
```

6. この値を使用して **OvsDpdkSocketMemory** を設定します。

```
OvsDpdkSocketMemory: "4096, 1024"
```

### サンプルの計算 - MTU 2000

DPDK NIC dpdk0 と dpdk1 は同じ NUMA ノード 0 上にあり、それぞれ MTU 2000 と MTU 2000 で設定されています。必要なメモリーを算出する計算例を以下に示します。

1. MTU 値を 1024 バイトの倍数に丸めます。

```
MTU 値 2000 は 2048 バイトになります。
```

2. それらの丸めたバイト値に基づいて、各 MTU 値に必要なメモリーを計算します。

```
MTU 値 2000 に必要なメモリー = (2048 + 800) * (4096*64) = 746586112
```

3. それらを合わせた必要なメモリーの合計を計算します (バイト単位)。

```
746586112 + 536870912 = 1283457024 バイト
```

この計算は、(MTU 値 2000 に必要なメモリー) + (512 MB バッファ) を示しています。

4. 必要合計メモリーを MB に変換します。

```
1283457024 / (1024*1024) = 1224 MB
```

5. この値を 1024 の倍数に丸めます。

```
1224 MB を丸めると 2048 MB になります。
```

6. この値を使用して **OvsDpdkSocketMemory** を設定します。

```
OvsDpdkSocketMemory: "2048, 1024"
```

### 7.4.3. ネットワークパラメーター



### NeutronDpdkDriverType

DPDK によって使用されるドライバーの種別を設定します。**vfio-pci** のデフォルト値を使用してください。

### NeutronDatapathType

OVS ブリッジ用のデータパスの種別を設定します。DPDK は **netdev** のデフォルト値を使用してください。

### NeutronVhostuserSocketDir

OVS 向けに vhost-user ソケットディレクトリーを設定します。vhost クライアントモード用の **/var/lib/vhost\_sockets** を使用してください。

## 7.4.4. その他のパラメーター

### NovaSchedulerDefaultFilters

要求されたゲストインスタンスに対してコンピュートノードが使用するフィルターの順序付きリストを指定します。

### KernelArgs

コンピュートノードのブート時用に、複数のカーネル引数を **/etc/default/grub** に指定します。設定に応じて、以下のパラメーターを追加します。

- **hugepagesz**: CPU 上のヒュージページのサイズを設定します。この値は、CPU のハードウェアによって異なります。OVS-DPDK デプロイメントには 1G に指定します (**default\_hugepagesz=1GB hugepagesz=1G**)。 **pdpe1gb** CPU フラグが出力されるかどうかをチェックして、CPU が 1G をサポートしていることを確認してください。

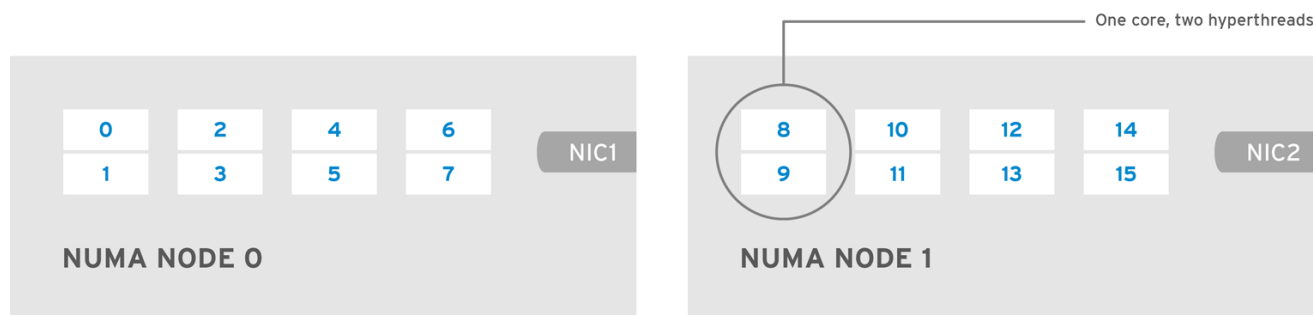
```
lshw -class processor | grep pdpe1gb
```

- **hugepages count**: ヒュージページの数を設定します。この値は、ホストの使用可能なメモリーの量によって異なります。利用可能なメモリーの大半を使用してください (**NovaReservedHostMemory** を除く)。ヒュージページ数の値は、お使いのコンピュートノードに関連付けられている OpenStack フレーバーの範囲内で設定する必要もあります。
- **iommu**: Intel CPU の場合は、**"intel\_iommu=on iommu=pt"** を追加します。
- **isolcpus**: チューニングされる CPU コアを設定します。この値は **IsolCpusList** と一致します。

## 7.5. 2 NUMA ノード構成の OVS-DPDK デプロイメントの例

本項に例示するコンピュートノードは、以下のような 2 つの NUMA ノードで構成されます。

- NUMA 0 にはコア 0-7 があり、シブリングスレッドペアは (0,1)、(2,3)、(4,5)、(6,7)。
- NUMA 1 にはコア 8-15 があり、シブリングスレッドペアは (8,9)、(10,11)、(12,13)、(14,15)。
- 各 NUMA ノードが物理 NIC (NUMA 0 上の NIC1 と NUMA 1 上の NIC2) に接続されている。



OPENSTACK\_453316\_0717

**注記**

各 NUMA ノード上の 1 番目の物理コアの両スレッドペア (0、1 および 8、9) は、データパス以外の DPDK プロセス (**OvsDpdkCoreList**) 用に確保します。

この例では、MTU が 1500 に設定されており、全ユースケースで **OvsDpdkSocketMemory** が同じであることも前提です。

```
OvsDpdkSocketMemory: "1024, 1024"
```

**NIC 1 は DPDK 用で、1 つの物理コアは PMD 用です。**

このユースケースでは、PMD 用の NUMA 0 に物理コアを 1 つ 割り当てます。NUMA 1 ノードでは NIC に DPDK が有効化されていませんが、NUMA 1 にも物理コアを 1 つ 割り当てる必要があります。残りのコア (**OvsDpdkCoreList** 用に確保されていないコア) はゲストインスタンスに割り当てられます。その結果、パラメーターの設定は以下のようになります。

```
OvsPmdCoreList: "2, 3, 10, 11"
NovaVcpuPinSet: "4, 5, 6, 7, 12, 13, 14, 15"
```

**NIC 1 は DPDK 用で、2 つの物理コアは PMD 用**

このユースケースでは、PMD 用の NUMA 0 に物理コアを 2 つ 割り当てます。NUMA 1 ノードでは NIC に DPDK が有効化されていませんが、NUMA 1 にも物理コアを 1 つ 割り当てる必要があります。残りのコア (**OvsDpdkCoreList** 用に確保されていないコア) はゲストインスタンスに割り当てられます。その結果、パラメーターの設定は以下のようになります。

```
OvsPmdCoreList: "2, 3, 4, 5, 10, 11"
NovaVcpuPinSet: "6, 7, 12, 13, 14, 15"
```

**NIC 2 は DPDK 用で、1 つの物理コアは PMD 用です。**

このユースケースでは、PMD 用の NUMA 1 に物理コアを 1 つ 割り当てます。NUMA 0 ノードでは NIC に DPDK が有効化されていませんが、NUMA 0 にも物理コアを 1 つ 割り当てる必要があります。残りのコア (**OvsDpdkCoreList** 用に確保されていないコア) はゲストインスタンスに割り当てられます。その結果、パラメーターの設定は以下のようになります。

```
OvsPmdCoreList: "2, 3, 10, 11"
NovaVcpuPinSet: "4, 5, 6, 7, 12, 13, 14, 15"
```

**NIC 2 は DPDK 用で、2 つの物理コアは PMD 用**

このユースケースでは、PMD 用の NUMA 1 に物理コアを 2 つ 割り当てます。NUMA 0 ノードでは NIC

に DPDK が有効化されていませんが、NUMA 0 にも物理コアを 1 つ割り当てる必要があります。残りのコア (**OvsDpdkCoreList** 用に確保されていないコア) はゲストインスタンスに割り当てられます。その結果、パラメーターの設定は以下のようになります。

```
OvsPmdCoreList: "2, 3, 10, 11, 12, 13"  
NovaVcpuPinSet: "4, 5, 6, 7, 14, 15"
```

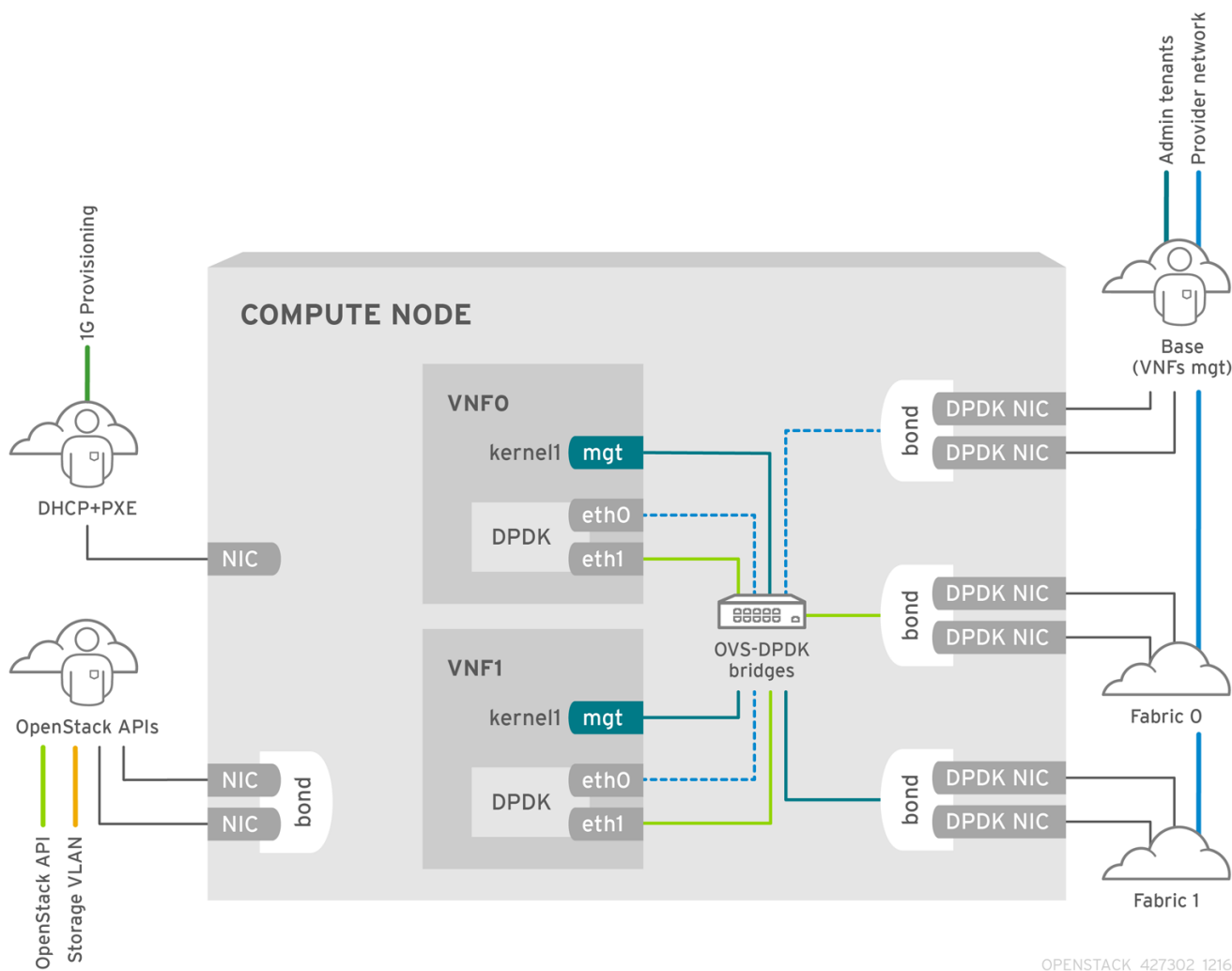
### NIC 1 と NIC2 は DPDK 用で、2 つの物理コアは PMD 用

このユースケースでは、PMD 用の各 NUMA ノードに物理コアを 2 つ 割り当てます。残りのコア (**OvsDpdkCoreList** 用に確保されていないコア) はゲストインスタンスに割り当てられます。その結果、パラメーターの設定は以下のようになります。

```
OvsPmdCoreList: "2, 3, 4, 5, 10, 11, 12, 13"  
NovaVcpuPinSet: "6, 7, 14, 15"
```

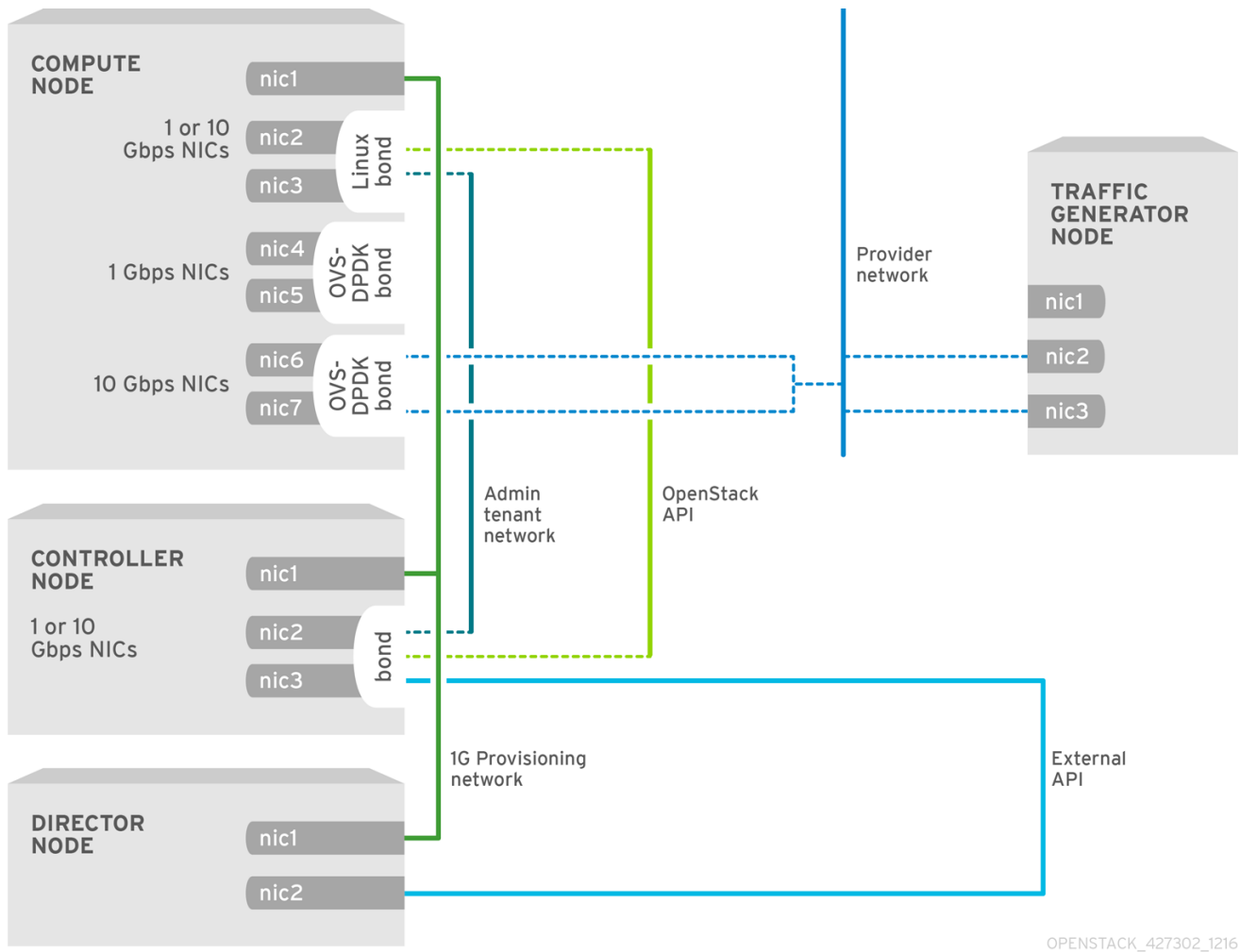
## 7.6. NFV OVS-DPDK デプロイメントのトポロジー

この例の OVS-DPDK デプロイメントは 2 つの VNF で構成され、それぞれに 2 つのインターフェース (**mgt** で示されている管理インターフェースとデータプレーンインターフェース) があります。OVS-DPDK デプロイメントでは、VNF は、物理インターフェースをサポートする組み込みの DPDK で稼働します。OVS-DPDK は、vSwitch レベルでボンディングを管理します。OVS-DPDK デプロイメントでは、カーネルと OVS-DPDK の NIC を **混在させない** ことを推奨します。混在させた場合には、パフォーマンスが低下する可能性があります。仮想マシン向けのベースプロバイダーネットワークに接続された管理 (**mgt**) ネットワークを分離するには、追加の NIC があることを確認する必要があります。コンピュータノードは、OpenStack API 管理向けの標準の NIC 2 つで構成されます。これは、Ceph API で再利用できますが、OpenStack テナントとは一切共有できません。



## NFV OVS-DPDK のトポロジー

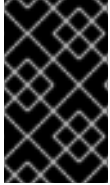
以下の図には、NFV ユースケース向けの OVS\_DPDK のトポロジーを示しています。この環境は、1 Gbps または 10 の 1 Gbps の NIC を搭載したコンピュータードおよびコントローラーノードと、director ノードで構成されます。



## 第8章 OVS-DPDK デプロイメントの設定

本項では、DPDK (OVS-DPDK) を Open vSwitch とともに Red Hat OpenStack Platform 環境内にデプロイします。オーバークラウドは、通常コントローラーノードやコンピュートノードなどの事前定義済みロールのノードと、異なる種別のストレージノードで構成されます。これらのデフォルトロールにはそれぞれ、director ノード上のコア Heat テンプレートで定義されている一式のサービスが含まれます。

オーバークラウドをデプロイする前に、アンダークラウドのインストールと設定が完了している必要があります。詳しくは、『[director のインストールと使用方法](#)』を参照してください。



### 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** ファイルで設定する OVS-DPDK パラメーターの最も適切な値を決定する必要があります。



### 注記

これらの director Heat テンプレートによって変更されている **etc/tuned/cpu-partitioning-variables.conf** の **isolated\_cores** またはその他の値は編集/変更しないでください。

### 8.1. ワークフローを使用した DPDK パラメーターの算出



### 重要

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。

DPDK 向けの Mistral ワークフローに関する概要は、「[ワークフローと派生パラメーターについての概要](#)」を参照してください。

#### 前提条件

このワークフローで取得されるデータを提供するには、ハードウェア検査で追加情報を取得するための追加のパラメーター (**inspection\_extras**) を含むベアメタルのイントロスペクションを有効化しておく必要があります。ハードウェア検査の追加パラメーターはデフォルトで有効化されます。「[ノードのハードウェアの検査](#)」を参照してください。

#### DPDK 向けのワークフローと入力パラメーターの定義

OVS-DPDK ワークフローで指定することができる入力パラメーターの一覧を以下に示します。

##### num\_phy\_cores\_per\_numa\_node\_for\_pmd

この入力パラメーターは、DPDK NIC に関連付けられた NUMA ノードの必要最小限のコア数を指定します。DPDK NIC に関連付けられていないその他の NUMA ノードには、物理コアが 1 つ割り当てられます。このパラメーターは 1 に設定すべきです。

##### huge\_page\_allocation\_percentage

この入力パラメーターは、ヒュージページとして設定可能な合計メモリー中 (**NovaReservedHostMemory** を除く) の必要なパーセンテージを指定します。 **KernelArgs** パラメーターは、指定した **huge\_page\_allocation\_percentage** に基づいて計算されたヒュージ

ページを使用して派生されます。このパラメーターは 50 に設定すべきです。

ワークフローは、これらの入力パラメーターとベアメタルのイントロスペクションの情報を使用して、適切な DPDK パラメーター値を算出します。

DPDK 用のワークフローと入力パラメーターを定義するには、以下の手順を実行します。

1. **tripleo-heat-templates/plan-samples/plan-environment-derived-params.yaml** ファイルをローカルのディレクトリーにコピーして、お使いの環境に適した入力パラメーターを設定します。

```
workflow_parameters:
  tripleo.derive_params.v1.derive_parameters:
    # DPDK Parameters #
    # Specifies the minimum number of CPU physical cores to be
    allocated for DPDK
    # PMD threads. The actual allocation will be based on network
    config, if
    # the a DPDK port is associated with a numa node, then this
    configuration
    # will be used, else 1.
    num_phy_cores_per_numa_node_for_pmd: 1
    # Amount of memory to be configured as huge pages in
    percentage. Out of the
    # total available memory (excluding the
    NovaReservedHostMemory), the
    # specified percentage of the remaining is configured as huge
    pages.
    huge_page_allocation_percentage: 50
```

2. **update-plan-only** パラメーターを使用してオーバークラウドをデプロイし、派生パラメーターを計算します。

```
$ openstack overcloud deploy --templates --update-plan-only -r
/home/stack/ospd-13-sriov-dpdk-heterogeneous-cluster/roles_data.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml -e /home/stack/ospd-13-sriov-dpdk-heterogeneous-
cluster/docker-images.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/host-config-and-reboot.yaml -e
/home/stack/ospd-13-sriov-dpdk-heterogeneous-cluster/network-
environment.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/neutron-sriov.yaml
-e /usr/share/openstack-tripleo-heat-
templates/environments/neutron-ovs-dpdk.yaml
-p /home/stack/plan-environment-derived-params.yaml
```

このコマンドの出力には、派生した結果が表示されます。これは、**plan-environment.yaml** ファイルにもマージされます。

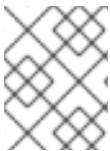
```
Started Mistral Workflow
tripleo.validations.v1.check_pre_deployment_validations. Execution ID:
55ba73f2-2ef4-4da1-94e9-eae2fdc35535
Waiting for messages on queue 472a4180-e91b-4f9e-bd4c-1fbdfbcf414f with no
timeout.
Removing the current plan files
```

```

Uploading new plan files
Started Mistral Workflow
tripleo.plan_management.v1.update_deployment_plan. Execution ID: 7fa995f3-
7e0f-4c9e-9234-dd5292e8c722
Plan updated.
Processing templates in the directory /tmp/tripleoclient-SY6RcY/tripleo-
heat-templates
Invoking workflow (tripleo.derive_params.v1.derive_parameters) specified
in plan-environment file
Started Mistral Workflow tripleo.derive_params.v1.derive_parameters.
Execution ID: 2d4572bf-4c5b-41f8-8981-c84a363dd95b
Workflow execution is completed. result:
ComputeOvsDpdkParameters:
  IsolCpusList:
1, 2, 3, 4, 5, 6, 7, 9, 10, 17, 18, 19, 20, 21, 22, 23, 11, 12, 13, 14, 15, 25, 26, 27, 28, 29, 30, 3
1
  KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
intel_iommu=on

isolcpus=1, 2, 3, 4, 5, 6, 7, 9, 10, 17, 18, 19, 20, 21, 22, 23, 11, 12, 13, 14, 15, 25, 26, 27, 2
8, 29, 30, 31
NovaReservedHostMemory: 4096
NovaVcpuPinSet:
2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22, 23, 10, 11, 12, 13, 14, 15, 26, 27, 28, 29, 30, 31
OvsDpdkCoreList: 0, 16, 8, 24
OvsDpdkMemoryChannels: 4
OvsDpdkSocketMemory: 1024, 1024
OvsPmdCoreList: 1, 17, 9, 25

```



## 注記

**OvsDpdkMemoryChannels** パラメーターはイントロスペクションの情報からは派生できません。大半の場合、この値は 4 に設定すべきです。

## 派生パラメーターを使用したオーバークラウドのデプロイ

これらの派生パラメーターを使用してオーバークラウドをデプロイするには、以下の手順を実行します。

1. 派生パラメーターを **plan-environment.yaml** ファイルから **network-environment.yaml** ファイルにコピーします。

```

# DPDK compute node.
ComputeOvsDpdkParameters:
  KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=32
iommu=pt intel_iommu=on
  TunedProfileName: "cpu-partitioning"
  IsolCpusList:
"1, 2, 3, 4, 5, 6, 7, 9, 10, 17, 18, 19, 20, 21, 22, 23, 11, 12, 13, 14, 15, 25, 26, 27, 28,
29, 30, 31"
  NovaVcpuPinSet:
[ '2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22, 23, 10, 11, 12, 13, 14, 15, 26, 27, 28, 29, 30, 31'
]
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "1024, 1024"

```



```
OvsDpdkMemoryChannels: "4"
OvsDpdkCoreList: "0,16,8,24"
OvsPmdCoreList: "1,17,9,25"
```



### 注記

DPDK PMD 向けに DPDK NIC のある場合またはない場合も、各 NUMA ノードで少なくとも 1 CPU を (シブリングスレッドとともに) 割り当てて、ゲストインスタンスの作成でエラーが発生するのを回避する必要があります。



### 注記

これらのパラメーターは、特定のロール (ComputeOvsDpdk) に適用されます。これらのパラメーターは、グローバルで適用可能ですが、グローバルパラメーターはロール固有のパラメーターによってオーバーライドされます。

## 2. 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-sriov-dpdk-heterogeneous-cluster/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/neutron-
sriov.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/neutron-ovs-
dpdk.yaml \
-e /home/stack/ospd-13-sriov-dpdk-heterogeneous-cluster/network-
environment.yaml
```

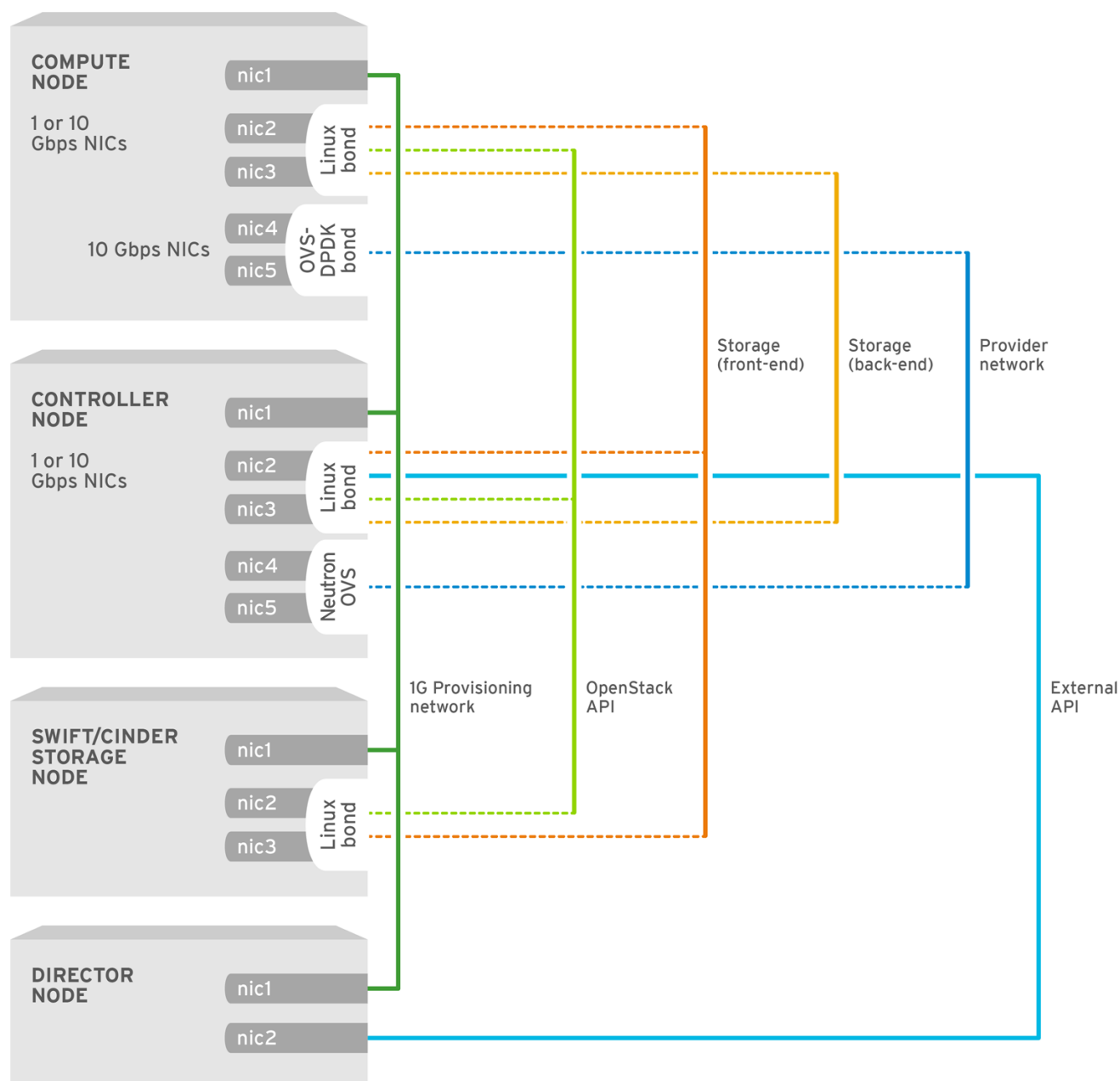
### 注記

Compute、ComputeOvsDpdk、ComputeSriov で構成されるクラスターでは、既存の派生パラメーターのワークフローにより ComputeOvsDpdk ロールの計算式のみが適用され、他のロールは影響を受けません。

## 8.2. OVS-DPDK のトポロジー

Red Hat OpenStack Platform では、コンポーザブルロール機能を使用し、各ロールにサービスを追加/削除してカスタムのデプロイメントロールを作成できます。コンポーザブルロールの詳細情報は「[コンポーザブルサービスとカスタムロール](#)」を参照してください。

以下の図は、コントロールプレーンとデータプレーン用にポートが 2 つボンディングされている OVS-DPDK トポロジーの例を示しています。



OPENSTACK\_450694\_0617

OVS-DPDK の設定は、以下の作業で構成されます。

- コンポーザブルロールを使用する場合には、**roles\_data.yaml** ファイルをコピーして編集し、OVS-DPDK 用のカスタムロールを追加します。
- 適切な **network-environment.yaml** ファイルを更新して、カーネル引数と DPDK 引数のパラメーターを追加します。
- **compute.yaml** ファイルを更新して、DPDK インターフェース用のブリッジを追加します。
- **controller.yaml** ファイルを更新して、DPDK インターフェースパラメーター用の同じブリッジ情報を追加します。
- **overcloud\_deploy.sh** スクリプトを実行して、DPDK パラメーターを使用してオーバークラウドをデプロイします。



### 注記

本ガイドでは、CPU の割り当て、メモリーの確保、NIC の設定の例を紹介します。これらは、トポロジーとユースケースによって異なる場合があります。ハードウェアと設定のオプションについて理解するには、『[ネットワーク機能仮想化 \(NFV\) の製品ガイド](#)』と「[2章 ハードウェア要件](#)」を参照してください。

手順を開始する前に、以下の項目が揃っていることを確認します。

- OVS 2.9
- DPDK 17
- テスト済み NIC。NFV 向けのテスト済み NIC の一覧は、『[テスト済みの NIC](#)』を参照してください。



### 注記

OVS-DPDK デプロイメントでは、Red Hat OpenStack Platform は、OVS クライアントモードで稼働します。

## 8.3. OVS-DPDK インターフェースの MTU 値の設定

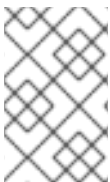
Red Hat OpenStack Platform は OVS-DPDK 向けにジャンボフレームをサポートしています。ジャンボフレーム用の MTU 値を設定するには、以下の作業を行う必要があります。

- **network-environment.yaml** ファイルで、グローバルの MTU 値を設定します。
- **compute.yaml** ファイルで物理 DPDK ポートの MTU 値を設定します。この値は、vhost のユーザーインターフェースでも使用されます。
- コンピュートノード上の任意のゲストインスタンスで MTU 値を設定し、設定内でエンドツーエンドに同等の MTU 値が設定されるようにします。



### 注記

VXLAN パケットには追加で 50 バイトがヘッダーに含まれます。MTU の必要値は、ヘッダーの追加バイト値に基づいて計算してください。たとえば、MTU 値が 9000 の場合には、これらの追加バイト値を計算に入れると、VXLAN トンネルの MTU 値は 8950 となります。



### 注記

物理 NIC は DPDK PMD によって制御され、**compute.yaml** ファイルで設定されているのと同じ MTU 値が適用されるので、特別な設定は必要ありません。MTU 値には、物理 NIC でサポートされているよりも高い値を設定することはできません。

OVS-DPDK インターフェースの MTU 値を設定するには、以下の手順を実行します。

1. **network-environment.yaml** ファイルで **NeutronGlobalPhysnetMtu** パラメーターを設定します。

```
parameter_defaults:
  # MTU global configuration
  NeutronGlobalPhysnetMtu: 9000
```



#### 注記

**network-environment.yaml** ファイルの `NeutronDpdkSocketMemory` の値がジャンボフレームをサポートするのに十分に大きな値であることを確認します。詳しくは、「[メモリーパラメーター](#)」を参照してください。

2. **controller.yaml** ファイルでコンピュートノードへのブリッジ上の MTU 値を設定します。

```
-
  type: ovs_bridge
  name: br-link0
  use_dhcp: false
  members:
    -
      type: interface
      name: nic3
      mtu: 9000
```

3. **compute.yaml** ファイルで OVS-DPDK ボンディング用の MTU 値を設定します。

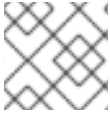
```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 9000
          members:
            - type: interface
              name: nic5
```

## 8.4. セキュリティーグループの設定

Red Hat OpenStack Platform director で OVS ファイアウォールドライバーを使用するためのセキュリティーグループを設定することができます。**NeutronOVSEFirewallDriver** パラメーターで、どのファイアウォールドライバーを使用するかを制御することができます。

- **iptables\_hybrid**: OpenStack Networking が iptables/ハイブリッドベースの実装を使用するように設定します。
- **openvswitch**: OpenStack Networking で OVS ファイアウォールのフローベースのドライバーを使用するように設定します。

**openvswitch** OVS ファイアウォールドライバーはパフォーマンスがより高く、ゲストをプロジェクトネットワークに接続するためのインターフェースとブリッジの数を削減します。



#### 注記

**iptables\_hybrid** オプションは、OVS-DPDK との互換性はありません。

**network-environment.yaml** ファイルで **NeutronOVSFirewallDriver** パラメーターを設定します。

```
# Configure the classname of the firewall driver to use for implementing
security groups.
NeutronOVSFirewallDriver: openvswitch
```

セキュリティーグループの有効化/無効化についての情報は、[「Basic security group operations」](#) を参照してください。

## 8.5. OVS-DPDK インターフェース向けのマルチキューの設定

コンピュータノード上の OVS-DPDK のインターフェースに同じ数のキューを設定するには、**compute.yaml** ファイルを以下のように変更します。

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 9000
          members:
            - type: interface
              name: nic5
```

## 8.6. 既知の制限事項

NFV のユースケース向けに Red Hat OpenStack Platform で OVS-DPDK を設定する場合には特定の制限事項があります。

- コントロールプレーンのネットワークには、Linux ボンディングを使用します。パフォーマンスを最適化するには、ボンディングに使用されている両方の PCI デバイスが同じ NUMA ノード上にあることを確認してください。Red Hat では、Neutron の Linux ブリッジ構成はサポートしていません。
- ヒュージページは OVS-DPDK を使用するホスト上で実行される全インスタンスに必要です。ゲストのヒュージページがない場合には、インターフェースは表示されても機能しません。
- TAP デバイスは DPDK をサポートしていないため、それらのデバイスを使用するサービスのパフォーマンスが低下します。たとえば、DVR、FWaaS、LBaaS などのサービスは TAP デバイスを使用します。
  - OVS-DPDK では、**netdev datapath** で DVR を有効化することができますが、パフォーマンスが低いので、実稼働環境には適していません。DVR はカーネルの名前空間と TAP デバイスを使用してルーティングを実行します。
  - OVS-DPDK で DVR ルーティングのパフォーマンスを良好な状態にするには、OpenFlow ルールとしてルーティングを実装する ODL などのコントローラーを使用する必要があります。OVS-DPDK では、OpenFlow ルーティングは、Linux カーネルインターフェースによって生じるボトルネックをなくすので、データパスの完全なパフォーマンスが維持されます。
- OVS-DPDK を使用する場合には、同じコンピュータード上の **すべて** のブリッジが **ovs\_user\_bridge** の種別である必要があります。director は設定を受け入れることができますが、Red Hat OpenStack Platform は同じノード上で **ovs\_bridge** と **ovs\_user\_bridge** が混在する構成はサポートしていません。

## 8.7. OVS-DPDK 用のフレーバーの作成とインスタンスのデプロイ

NFV を実装する Red Hat OpenStack Platform デプロイメントの OVS-DPDK の設定を完了した後は、以下の手順に従ってフレーバーを作成してインスタンスをデプロイすることができます。

1. アグリゲートグループを作成して、OVS-DPDK 用にホストを追加します。

```
# openstack aggregate create --zone=dpdk dpdk
# openstack aggregate add host dpdk compute-ovs-dpdk-0.localdomain
```



### 注記

CPU ピニングされたインスタンスをピニングされていないインスタンスと分けるには、ホストアグリゲートを使用すべきです。CPU ピニングを使用していないインスタンスは、CPU ピニングを使用するインスタンスのリソース要件は順守しません。

2. フレーバーを作成します。

```
# openstack flavor create m1.medium_huge_4cpu --ram 4096 --disk 150
--vcpus 4
```

このコマンドでは、**m1.medium\_huge\_4cpu** はフレーバー名、**4096** は MB 単位のメモリー容量、**150** は GB 単位のディスク容量 (デフォルトでは 0 G)、**4** は仮想 CPU 数を設定しています。

3. フレーバーの追加のプロパティを設定します。

```
# openstack flavor set --property hw:cpu_policy=dedicated --property
hw:mem_page_size=1GB m1.medium_huge_4cpu --property
hw:emulator_threads_policy=isolate
```

このコマンドでは、**m1.medium\_huge\_4cpu** はフレーバー名を指定しており、残りはそのフレーバーのその他のプロパティを設定しています。

パフォーマンス向上のためのエミュレータスレッドポリシーについての詳しい情報は、[「Configure Emulator Threads to run on a Dedicated Physical CPU」](#) を参照してください。

1. ネットワークを作成します。

```
# openstack network create net1 --provider-physical-network tenant -
-provider-network-type vlan --provider-segment <VLAN-ID>
```

2. インスタンスをデプロイします。

```
# openstack server create --flavor m1.medium_huge_4cpu --
availability-zone dpdk --image rhel_7.3 --nic net-id=net1
```

ここで

- **m1.medium\_huge\_4cpu** はフレーバー名または ID です。
- **dpdk** はサーバーのアベイラビリティゾーンです。
- **rhel\_7.3** はインスタンスの作成に使用するイメージ (名前または ID) です。
- **net1** はサーバー上の NIC です。

これで、NFV ユースケースの OVS-DPDK 向けインスタンスのデプロイが完了しました。

**multi-queue** を OVS-DPDK で使用するには、上記の手順に数ステップを追加する必要があります。フレーバーを作成する前に、以下のステップを実行してください。

1. イメージのプロパティを設定します。

```
# openstack image set --property hw_vif_multiqueue_enabled=true
<image-id>
```

ここで、**hw\_vif\_multiqueue\_enabled=true** はこのイメージ上でマルチキューを有効にするためのプロパティで、**<image-id>** は変更するイメージの名前または ID です。

2. フレーバーの追加のプロパティを設定します。

```
# openstack flavor set m1.vm_mq set hw:vif_multiqueue_enabled=true
```

ここで、**m1.vm\_mq** はフレーバーの ID または名前、残りのオプションはそのフレーバーのマルチキューを有効化します。

## 8.8. 設定のトラブルシューティング

本項では、DPDK-OVS 設定のトラブルシューティングの手順を説明します。

1. ブリッジの設定を見直して、ブリッジが **datapath\_type=netdev** で作成されたことを確認します。

```
# ovs-vsctl list bridge br0
_uuid                : bdce0825-e263-4d15-b256-f01222df96f3
auto_attach          : []
controller           : []
datapath_id          : "00002608cebd154d"
datapath_type        : netdev
datapath_version     : "<built-in>"
external_ids         : {}
fail_mode            : []
flood_vlans          : []
flow_tables          : {}
ipfix                : []
mcast_snooping_enable: false
mirrors              : []
name                 : "br0"
netflow              : []
other_config         : {}
ports                : [52725b91-de7f-41e7-bb49-3b7e50354138]
protocols            : []
rstp_enable          : false
rstp_status          : {}
sflow                : []
status               : {}
stp_enable           : false
```

2. **neutron-ovs-agent** が自動的に起動するように設定されていることを確認して、OVS サービスをレビューします。

```
# systemctl status neutron-openvswitch-agent.service
neutron-openvswitch-agent.service - OpenStack Neutron Open vSwitch
Agent
Loaded: loaded (/usr/lib/systemd/system/neutron-openvswitch-
agent.service; enabled; vendor preset: disabled)
Active: active (running) since Mon 2015-11-23 14:49:31 AEST; 25min
ago
```

サービスの起動に問題がある場合には、以下のコマンドを実行して関連のメッセージを表示することができます。

```
# journalctl -t neutron-openvswitch-agent.service
```

3. **ovs-dpdk** の PMD CPU マスクが CPU にピンングされていることを確認します。HT の場合には、シブリング CPU を使用します。  
たとえば **CPU4** を例に取ります。

```
# cat /sys/devices/system/cpu/cpu4/topology/thread_siblings_list
4,20
```



CPU 4 と 20 を使用します。

```
# ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x100010
```

ステータスを表示します。

```
# tuna -t ovs-vswitchd -CP
thread  ctxt_switches pid SCHED_ rtpri affinity voluntary
nonvoluntary      cmd
3161 OTHER  0      6 765023      614 ovs-vswitchd
3219  OTHER  0      6      1      0 handler24
3220  OTHER  0      6      1      0 handler21
3221  OTHER  0      6      1      0 handler22
3222  OTHER  0      6      1      0 handler23
3223  OTHER  0      6      1      0 handler25
3224  OTHER  0      6      1      0 handler26
3225  OTHER  0      6      1      0 handler27
3226  OTHER  0      6      1      0 handler28
3227  OTHER  0      6      2      0 handler31
3228  OTHER  0      6      2      4 handler30
3229  OTHER  0      6      2      5 handler32
3230  OTHER  0      6 953538      431 revalidator29
3231  OTHER  0      6 1424258      976 revalidator33
3232  OTHER  0      6 1424693      836 revalidator34
3233  OTHER  0      6 951678      503 revalidator36
3234  OTHER  0      6 1425128      498 revalidator35
*3235  OTHER  0      4 151123      51 pmd37*
*3236  OTHER  0     20 298967      48 pmd38*
3164  OTHER  0      6 47575      0 dpdk_watchdog3
3165  OTHER  0      6 237634      0 vhost_thread1
3166  OTHER  0      6 3665      0 urcu2
```

## 第9章 NFV ワークロードに向けた RT-KVM の有効化

本項では、Red Hat OpenStack Platform 向けに Red Hat Enterprise Linux 7.5 Real Time KVM (RT-KVM) をインストールおよび設定する手順を説明します。Red Hat OpenStack Platform は Red Hat Enterprise Linux for Real-Time に加えて、追加の RT-KVM カーネルモジュールおよびコンピュートノードの自動設定をプロビジョニングする新しい Real-Time コンピュートノードロールを使用したリアルタイムの機能を提供します。

### 9.1. RT-KVM コンピュートノードのプランニング

RT-KVM コンピュートノードには、Red Hat 認定のサーバーを使用する必要があります。詳しくは、[Red Hat Enterprise Linux for Real Time 7 certified servers](#) を参照してください。

RT-KVM 用の **rhel-7-server-nfv-rpms** リポジトリを有効にする方法については、「[アンダークラウドの登録と更新](#)」を参照してください。



#### 注記

このリポジトリにアクセスできるようにするには、**Red Hat OpenStack Platform for Real Time** SKU とは別のサブスクリプションが必要となります。

正しいパッケージがインストールされていることを確認できます。

```
$ yum --disablerepo=beaker-tasks repo-pkgs rhel-7-server-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
kernel-rt.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-debug.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-debug-devel.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-debug-kvm.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-devel.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-doc.noarch
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-kvm.x86_64
3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
[ output omitted...]
```

**real-time のイメージをビルドします。**

Real-time コンピュートノードのオーバークラウドイメージをビルドするには、以下のステップを実行します。

1. アンダークラウドに libguestfs-tools パッケージをインストールして、virt-customize ツールを取得します。

```
(undercloud) [stack@undercloud-0 ~]$ sudo yum install libguestfs-  
tools
```

2. イメージを抽出します。

```
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-  
director-images/overcloud-full.tar  
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-  
director-images/ironic-python-agent.tar
```

3. デフォルトのイメージをコピーします。

```
(undercloud) [stack@undercloud-0 ~]$ cp overcloud-full.qcow2  
overcloud-realtime-compute.qcow2
```

4. <https://access.redhat.com/articles/1556833> に記載の手順に従ってサブスクリプションを設定します。現在必要なのは、NFV のサブスクリプションが 1 つです。

5. イメージ上で rt を設定するためのスクリプトを作成します (# END OF SCRIPT まで)。

```
(undercloud) [stack@undercloud-0 ~]$ cat rt.sh  
#!/bin/bash  
  
set -eux  
  
yum -v -y --setopt=protected_packages= erase kernel.$(uname -m)  
yum -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host  
# END OF SCRIPT
```

6. RT イメージを設定するスクリプトを実行します。

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-  
realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee virt-customize.log
```

7. SELinux の再ラベル付けをします。

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-  
realtime-compute.qcow2 --selinux-relabel
```

8. vmlinuz および initrd を抽出します。

```
(undercloud) [stack@undercloud-0 ~]$ mkdir image  
(undercloud) [stack@undercloud-0 ~]$ guestmount -a overcloud-  
realtime-compute.qcow2 -i --ro image  
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/vmlinuz-3.10.0-  
862.rt56.804.el7.x86_64 ./overcloud-realtime-compute.vmlinuz  
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-3.10.0-  
862.rt56.804.el7.x86_64.img ./overcloud-realtime-compute.initrd  
(undercloud) [stack@undercloud-0 ~]$ guestunmount image
```

**注記**

The software version in the **vmlinux** および **initramfs** のファイル名に含まれるソフトウェアバージョンは、カーネルバージョンによって異なります。

9. イメージをアップロードします。

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud image
upload --update-existing --os-image-name overcloud-realtime-
compute.qcow2
```

これで、選択したコンピュータード上の **ComputeOvsDpdkRT** コンポーザブルロールで使用するこ  
のできる real-time イメージの準備ができました。

## RT-KVM コンピュータード上での BIOS 設定の変更

RT-KVM コンピュータードのレイテンシーを短縮するために、BIOS 設定を変更する必要があります。コンピュータードの BIOS 設定で、以下のセクションの全オプションを無効にする必要があります。

- 電源管理
- ハイパースレッディング
- CPU のスリープ状態
- 論理プロセッサ

これらの設定に関する説明と、無効化の影響については、[「Setting BIOS parameters」](#) を参照してください。BIOS 設定の変更方法に関する詳しい情報は、ハードウェアの製造会社のドキュメントを参照してください。

## 9.2. RT-KVM 対応の OVS-DPDK の設定

**注記**

OVS-DPDK 向けの OpenStack ネットワークを最適化するには、network-environment.yaml ファイルで設定する OVS-DPDK パラメーターの最も適切な値を決定する必要があります。詳しくは、[「ワークフローを使用した DPDK パラメーターの算出」](#) を参照してください。

### 9.2.1. ComputeOvsDpdk コンポーザブルロールの生成

**ComputeOvsDpdkRT** ロールを使用して、real-time の Compute イメージを使用するコンピュータードを指定します。

**ComputeOvsDpdkRT** ロール向けに **roles\_data.yaml** を生成します。

```
# (undercloud) [stack@undercloud-0 ~]$ openstack overcloud roles generate
-o roles_data.yaml Controller ComputeOvsDpdkRT
```

### 9.2.2. CPU アフィニティー向けの tuned の設定

1. **tuned** の設定で CPU アフィニティを有効にするように設定します。

```

heat_template_version: 2014-10-16

description: >
  Example extra config for post-deployment

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

resources:

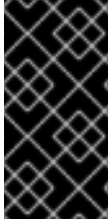
  ExtraDeployments:
    type: OS::Heat::StructuredDeployments
    properties:
      servers: {get_param: servers}
      config: {get_resource: ExtraConfig}
      actions: ['CREATE', 'UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

  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

```

### 9.2.3. 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** ファイルで設定する OVS-DPDK パラメーターの最も適切な値を決定する必要があります。詳しくは、[「ワークフローを使用した DPDK パラメーターの算出」](#)を参照してください。

1. **resource\_registry** 下には、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::TripleO::ComputeOvsDpdkRT::Net::SoftwareConfig: nic-
  configs/compute-ovs-dpdk.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-
  configs/controller.yaml
  OS::TripleO::NodeExtraConfigPost: post-install.yaml
```

2. **parameter\_defaults** 下には、OVS-DPDK および RT-KVM のパラメーターを設定します。

```
# DPDK compute node.
ComputeOvsDpdkRTParameters:
  KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=32
  iommu=pt intel_iommu=on
  TunedProfileName: "realtime-virtual-host"
  IsolCpusList:
    "1,2,3,4,5,6,7,9,10,17,18,19,20,21,22,23,11,12,13,14,15,25,26,27,28,
    29,30,31"
  NovaVcpuPinSet:
    ['2,3,4,5,6,7,18,19,20,21,22,23,10,11,12,13,14,15,26,27,28,29,30,31']
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "1024,1024"
  OvsDpdkMemoryChannels: "4"
  OvsDpdkCoreList: "0,16,8,24"
  OvsPmdCoreList: "1,17,9,25"
  VhostuserSocketGroup: "hugetlbfs"
  ComputeOvsDpdkRTImage: "overcloud-realtime-compute"
```



#### 注記

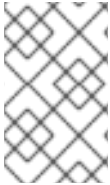
DPDK PMD 向けに DPDK NIC のある場合またはない場合も、各 NUMA ノードで少なくとも 1 CPU を (シブリングスレッドとともに) 割り当てて、ゲストインスタンスの作成でエラーが発生するのを回避する必要があります。



### 注記

これらのヒュージページは、仮想マシンにより消費されます。また、本手順に示したように **OvsDpdkSocketMemory** パラメーターを使用すると OVS-DPDK により消費されます。仮想マシンが利用可能なヒュージページ数は、**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たフレーバーにも **hw:mem\_page\_size=1GB** を追加する必要があります。



### 注記

**OvsDPDKCoreList** と **OvsDpdkMemoryChannels** は、この手順の **必須** の設定です。適切な値なしに DPDK をデプロイしようとすると、デプロイが失敗するか、デプロイが不安定になる可能性があります。

## 9.2.4. コンテナイメージの準備

コンテナイメージを準備します。

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud container image
prepare --namespace=192.0.40.1:8787/rhosp13 --env-file=/home/stack/ospd-
13-vlan-dpdk/docker-images.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/docker.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/docker-ha.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/services-docker/neutron-ovs-dpdk.yaml -e
/home/stack/ospd-13-vlan-dpdk/network-environment.yaml --roles-file
/home/stack/ospd-13-vlan-dpdk/roles_data.yaml --prefix=openstack- --
tag=2018-03-29.1 --set ceph_namespace=registry.access.redhat.com/rhceph --
set ceph_image=rhceph-3-rhel7 --set ceph_tag=latest
```

## 9.2.5. オーバークラウドのデプロイ

ML2-OVS 向けのオーバークラウドをデプロイします。

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy \
--templates \
-r /home/stack/ospd-13-vlan-dpdk-ctlplane-bonding-rt/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-ovs-dpdk.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ovs-dpdk-
permissions.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-data-bonding-rt-hybrid/docker-
images.yaml \
-e /home/stack/ospd-13-vxlan-dpdk-data-bonding-rt-hybrid/network-
environment.yaml
```

## 9.3. RT-KVM インスタンスの起動

リアルタイム対応のコンピュータノードで RT-KVM インスタンスを起動するには、以下の手順を実行します。

1. オーバークラウド上に RT-KVM フレーバーを作成します。

```
# openstack flavor create  r1.small 99 4096 20 4
# openstack flavor set  --property hw:cpu_policy=dedicated 99
# openstack flavor set  --property hw:cpu_realtime=yes 99
# openstack flavor set  --property hw:mem_page_size=1GB 99
# openstack flavor set  --property hw:cpu_realtime_mask="^0-1" 99
# openstack flavor set  --property hw:cpu_emulator_threads=isolate 99
```

2. RT-KVM インスタンスを起動します。

```
# openstack server create  --image <rhel> --flavor r1.small --nic
net-id=<dppdk-net> test-rt
```

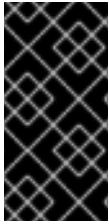
3. オプションとして、割り当てられたエミュレータスレッドをインスタンスが使用していることを確認します。

```
# virsh dumpxml <instance-id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'>/>
  <vcpupin vcpu='1' cpuset='3'>/>
  <vcpupin vcpu='2' cpuset='5'>/>
  <vcpupin vcpu='3' cpuset='7'>/>
  <emulatorpin cpuset='0-1'>/>
  <vcpusched vcpus='2-3' scheduler='fifo'
    priority='1'>/>
</cputune>
```



## 第10章 例: ODL および VXLAN トンネリングを使用する OVS-DPDK の設定

本項では、OVS-DPDK with ODL および VXLAN トンネリングを使用した OVS-DPDK 設定の例について説明します。



### 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** ファイルで設定する OVS-DPDK パラメーターの最も適切な値を決定する必要があります。詳しくは、「[ワークフローを使用した DPDK パラメーターの算出](#)」を参照してください。

### 10.1. 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
```

### 10.2. CPU アフィニティー向けの TUNED の設定

1. **tuned** の設定で CPU アフィニティーを有効にするように設定します。

```
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
```

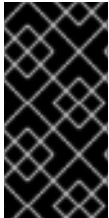
```

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

```

### 10.3. 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** ファイルで設定する OVS-DPDK パラメーターの最も適切な値を決定する必要があります。詳しくは、[「ワークフローを使用した DPDK パラメーターの算出」](#)を参照してください。

1. **resource\_registry** 下には、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::TripleO::ComputeOvsDpdk::Net::SoftwareConfig: nic-
configs/computeovsdpdk.yaml
    OS::TripleO::Controller::Net::SoftwareConfig: nic-
configs/controller.yaml
    OS::TripleO::NodeExtraConfigPost: post-install.yaml

```

2. **parameter\_defaults** 下には、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** 下には、ブリッジ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** 下には、**ComputeOvsDpdk** ロール向けにロール固有の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

DPDK PMD 向けに DPDK NIC のある場合またはない場合も、各 NUMA ノードで少なくとも 1 CPU を (シブリングスレッドとともに) 割り当てて、ゲストインスタンスの作成でエラーが発生するのを回避する必要があります。

#### 注記

これらのヒュージページは、仮想マシンにより消費されます。また、本手順に示したように **OvsDpdkSocketMemory** パラメーターを使用すると OVS-DPDK により消費されます。仮想マシンが利用可能なヒュージページ数は、**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たフレーバーにも **hw:mem\_page\_size=1GB** を追加する必要があります。

#### 注記

**OvsDPDKCoreList** と **OvsDpdkMemoryChannels** は、この手順の **必須** の設定です。適切な値なしに DPDK をデプロイしようとすると、デプロイが失敗するか、デプロイが不安定になる可能性があります。

## 10.4. コントローラーノードの設定

1. 分離ネットワーク用に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接続を可能にする Floating IP 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
```

## 10.5. DPDK インターフェイス用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** を作成し、以下のように変更します。

1. 分離ネットワーク用に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ための DPDK ポートを使用し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 デバイスが含まれるようにするには、追加する各 DPDK デバイスごとに **type** コードセクションを繰り返してください。



#### 注記

OVS-DPDK を使用する場合には、同じコンピュートノード上の **すべての** ブリッジが **ovs\_user\_bridge** の種別である必要があります。director は設定を受け入れることができますが、Red Hat OpenStack Platform は同じノード上で **ovs\_bridge** と **ovs\_user\_bridge** が混在する構成はサポートしていません。

## 10.6. 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
```

## 第11章 NFV を実装した RED HAT OPENSTACK PLATFORM のアップグレード

OVS-DPDK を設定している場合には、Red Hat OpenStack Platform のアップグレードで追加の考慮事項とステップが必要です。このステップについては、「[NFV を設定したオーバークラウドの準備](#)」で説明しています。



## 第12章 パフォーマンス

Red Hat OpenStack Platform director は、コンピュートノードがリソースの分割および微調整を有効にしてゲスト VNF のラインレートパフォーマンスを実現できるように設定します。NFV ユースケースの中での主要なパフォーマンス要素は、スループット、レイテンシー、ジッターです。

DPDK-accelerated OVS は、物理 NIC と仮想マシンの間で高性能のパケット切り替えを有効にします。OVS 2.7 は、DPDK 16.11 に対応しており、**vhost-user** のマルチキューをサポートしているので、スケーラブルなパフォーマンスを実現できます。OVS-DPDK は、ゲスト VNF のラインレートパフォーマンスを提供します。

SR-IOV ネットワークでは、固有なネットワークや仮想マシンのスループットの向上など、強化されたパフォーマンス特性が提供されます。

パフォーマンスチューニングの他の重要な機能には、ヒュージページ、NUMA 調整、ホストの分離、CPU ピニングなどが挙げられます。VNF フレーバーには、パフォーマンス向上のためにヒュージページとエミュレータスレッドの分離が必要です。ホストの分離や CPU ピニングにより、NFV パフォーマンスが向上され、擬似パケットロスが回避されます。

CPU と NUMA トポロジーに関する概要は、[「NFV のパフォーマンスの考慮事項」](#) および [「Configure Emulator Threads to run on a Dedicated Physical CPU」](#) を参照してください。

## 第13章 その他の参考資料

以下の表には、その他の参考となる Red Hat のドキュメントの一覧を記載しています。

Red Hat OpenStack Platform のドキュメントスイートは [Red Hat OpenStack Platform の製品ドキュメントスイート](#) から参照してください。

表13.1 参考資料一覧

コンポーネント	参考情報
Red Hat Enterprise Linux	Red Hat OpenStack Platform は Red Hat Enterprise Linux 7.5 でサポートされています。Red Hat Enterprise Linux のインストールに関する情報は、 <a href="#">Red Hat Enterprise Linux のドキュメント</a> から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 環境をアンダークラウドとして使用して、最終的な <b>オーバークラウド</b> 内の OpenStack ノードをインストール、設定、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、<a href="#">『Red Hat OpenStack Platform director のインストールと使用方法』</a> を参照してください。</p> <p>Red Hat OpenStack Platform director を使用して、ネットワークの分離、ストレージ設定、SSL 通信、一般的な設定方法など Red Hat OpenStack Platform のエンタープライズ環境の高度な機能設定に関する情報は <a href="#">『オーバークラウドの高度なカスタマイズ』</a> を参照してください。</p>
NFV のドキュメント	NFV の概念に関する俯瞰的な情報は、 <a href="#">『ネットワーク機能仮想化 (NFV) の製品ガイド』</a> を参照してください。

## 付録A ODL DPDK YAML ファイルのサンプル

本項では、参考として ODL DPDK YAML ファイルのサンプルを紹介します。

### A.1. VXLAN DPDK ODL YAML ファイルのサンプル

#### A.1.1. post-install.yaml

```
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
```

```

        fi
    }

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

```

### A.1.2. network.environment.yaml

```

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

parameter_defaults:
    # Customize all these values to match the local environment
    InternalApiNetCidr: 10.10.131.0/24
    TenantNetCidr: 10.10.132.0/24
    StorageNetCidr: 10.10.133.0/24
    StorageMgmtNetCidr: 10.10.134.0/24
    ExternalNetCidr: 10.35.185.64/28
    # CIDR subnet mask length for provisioning network
    ControlPlaneSubnetCidr: '24'
    InternalApiAllocationPools: [{'start': '10.10.131.100', 'end':
'10.10.131.200'}]
    TenantAllocationPools: [{'start': '10.10.132.100', 'end':
'10.10.132.200'}]
    StorageAllocationPools: [{'start': '10.10.133.100', 'end':
'10.10.133.200'}]
    StorageMgmtAllocationPools: [{'start': '10.10.134.100', 'end':
'10.10.134.200'}]
    # Use an External allocation pool which will leave room for floating IPs
    ExternalAllocationPools: [{'start': '10.35.185.65', 'end':
'10.35.185.77'}]
    # Set to the router gateway on the external network
    ExternalInterfaceDefaultRoute: 10.35.185.78
    # Gateway router for the provisioning network (or Undercloud IP)
    ControlPlaneDefaultRoute: 192.0.90.1
    # Generally the IP of the Undercloud
    EC2MetadataIp: 192.0.90.1
    InternalApiNetworkVlanID: 531
    TenantNetworkVlanID: 532
    StorageNetworkVlanID: 533
    StorageMgmtNetworkVlanID: 534
    ExternalNetworkVlanID: 422
    # Define the DNS servers (maximum 2) for the overcloud nodes
    DnsServers: ["10.35.28.1", "10.35.28.28"]
    # 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: 'vxlan'
# The tenant network type for Neutron (vlan or vxlan).
NeutronNetworkType: 'vxlan'
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings: 'tenant:br-link0'
OpenDaylightProviderMappings: 'tenant:br-link0'
# The Neutron ML2 and OpenVSwitch vlan mapping range to support.
NeutronNetworkVLANRanges: 'tenant:423:423,tenant:535:537'
# Nova flavor to use.
OvercloudControllerFlavor: controller
OvercloudComputeOvsDpdkFlavor: computeovsdpdk
# Number of nodes to deploy.
ControllerCount: 3
ComputeOvsDpdkCount: 2
# NTP server configuration.
NtpServer: clock.redhat.com

#####
# 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

# MTU global configuration
NeutronGlobalPhysnetMtu: 9000
# Configure the classname of the firewall driver to use for implementing
security groups.

SshServerOptions:
  UseDns: 'no'

```

### A.1.3. controller.yaml

```

heat_template_version: queens

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
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage mgmt 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
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 via parameter_defaults
  default: '24'
```

```

    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this via parameter_defaults
    description: The default route of the control plane network.
    type: string
  DnsServers: # Override this via 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 via parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string

resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-
templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
                -
                  type: interface
                  name: eth0
                  use_dhcp: false
                  addresses:
                    -
                      ip_netmask:
                        list_join:
                          - '/'
                          - - {get_param: ControlPlaneIp}
                            - {get_param: ControlPlaneSubnetCidr}
                  routes:
                    -
                      ip_netmask: 169.254.169.254/32
                      next_hop: {get_param: EC2MetadataIp}

                - 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

- 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

- 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

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

```

#### A.1.4. compute-ovs-dpdk.yaml

```

heat_template_version: queens

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
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage mgmt 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 via parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
ControlPlaneDefaultRoute: # Override this via parameter_defaults
    description: The default route of the control plane network.
    type: string
DnsServers: # Override this via 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 via 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::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-
templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
                - type: interface
                  name: nic1
                  use_dhcp: false
                  defroute: false

                - type: interface
                  name: nic2
                  use_dhcp: false
                  addresses:
                    - ip_netmask:

```

```

        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
    routes:
    - ip_netmask: 169.254.169.254/32
      next_hop:
        get_param: EC2MetadataIp
    - default: true
      next_hop:
        get_param: ControlPlaneDefaultRoute

- 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

- 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: 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

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

```

### A.1.5. overcloud\_deploy.sh

```

#!/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-opendaylight.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-
docker/neutron-opendaylight-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 \
--log-file overcloud_install.log &> overcloud_install.log

```

---

## 付録B 改訂履歴

改訂 1.4-0	August 23 2018	Fixed parameter alignment for step 4 of `Configuring SR-IOV with OVS Hardware Offload with VLAN`.
改訂 1.3-0	August 20 2018	Added note about SKU requirement for RT-KVM repository.
改訂 1.2-0	July 31 2018	Updated network creation steps to use OSC parameters. Added description of BIOS settings.
改訂 1.1-0	July 12 2018	DPDK ODL yaml ファイルのサンプルと手順を追加
改訂 1.0-0	June 27 2018	Red Hat OpenStack 13 GA リリースの初期バージョン。RT-KVM および OVS HW オフロードの手順を記載しています。ovs 2.9 をサポートします。