Red Hat OpenStack Platform 17.1

ネットワーク機能仮想化の設定

Red Hat Openstack Platform でのネットワーク機能仮想化 (NFV) の計画と設定

Last Updated: 2024-05-29

Red Hat OpenStack Platform 17.1 ネットワーク機能仮想化の設定

Red Hat Openstack Platform でのネットワーク機能仮想化 (NFV) の計画と設定

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

本ガイドでは、Red Hat OpenStack Platform デプロイメントのネットワーク機能仮想化インフラストラクチャー (NFVi) 向け Single Root Input/Output Virtualization (SR-IOV) および Data Plane Development Kit (DPDK) について、プランニングに関する重要な情報とその設定の手順を説明します。

目次

多様性を受け入れるオープンソースの強化

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

RED HAT ドキュメントへのフィードバック (英語のみ)

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

Jira でドキュメントのフィードバックを提供する

ドキュメントに関するフィードバックを提供するには、Create Issue フォームを使用します。Red Hat OpenStack Platform Jira プロジェクトで Jira Issue が作成され、フィードバックの進行状況を追跡できます。

1. Jira にログインしていることを確認してください。Jira アカウントをお持ちでない場合は、アカウントを作成してフィードバックを送信してください。
2. Create Issue をクリックして、**Create Issue** ページを開きます。
3. **Summary** フィールドと **Description** フィールドに入力します。**Description** フィールドに、ドキュメントの URL、章またはセクション番号、および問題の詳しい説明を入力します。フォーム内の他のフィールドは変更しないでください。
4. **Create** をクリックします。

第1章 RED HAT ネットワーク機能仮想化 (NFV) の理解

ネットワーク機能仮想化 (NFV: Network Functions Virtualization)とは、通信事業者 (CSP) が従来のプロプライエタリーハードウェアの範囲を超えて、運用コストを抑えつつ、効率性および俊敏性を向上させる上で役立つソフトウェアベースのソリューションのことです。

NFV 環境では、スイッチ、ルーター、ストレージなどの標準のハードウェアデバイス上で実行する標準の仮想化技術を使用して仮想化インフラストラクチャーを提供することで、IT およびネットワークの収束が可能になり、ネットワーク機能を仮想化 (VNF) します。管理およびオーケストレーションロジックにより、これらのサービスをデプロイおよび維持します。NFV には、手動操作の必要性を軽減するシステム管理、自動化、ライフサイクル管理も含まれます。

1.1. NFV の利点

ネットワーク機能仮想化 (NFV) を実装する主な利点は以下のとおりです。

- 新たなネットワークサービスを迅速にデプロイおよびスケーリングすることが可能なので、市場投入までの時間を短縮し、ニーズの変化に対応することができます。
- サービスの開発者は、実稼動環境で使用する場合と同じプラットフォームを使用して、リソースやプロトタイプを自己管理できるので、イノベーションがサポートされます。
- セキュリティーやパフォーマンスを犠牲にせず、週/日単位ではなく、時間/分単位で顧客のニーズに対応します。
- カスタマイズされた高額な設備ではなく、商用オフザシェルフ (COTS) のハードウェアを使用するため、資本支出が削減されます。
- 運用の合理化と自動化で日常のタスクを最適化することにより、運用コストを削減し、従業員の生産性を向上させます。

1.2. NFV デプロイメントでサポートされている設定

Red Hat OpenStack Platform director のツールキットを使用して、特定のネットワーク種別 (外部、プロジェクト、内部 API など) を分離します。ネットワークは、単一のネットワークインターフェイス上に、または複数のホストネットワークインターフェイスに分散してデプロイすることが可能です。Open vSwitch により、複数のインターフェイ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 インストール環境でネットワークの分離を設定するには、テンプレートファイルを使用します。テンプレートファイルを指定しない場合、サービスネットワークは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には、2 つの種類があります。

- **network-environment.yaml**
  このファイルには、サブネットおよび IP アドレス範囲などの、オーバークラウドノードのネットワーク情報が含まれます。このファイルには、さまざまなシナリオで使用できるように、デフォルトのパラメーター値を上書きする別の設定も含まれます。
- compute.yaml および controller.yaml などのホストネットワークテンプレート
  これらのテンプレートは、オーバークラウドノードのネットワークインターフェイス設定を定義します。ネットワーク情報の値は、**network-environment.yaml** ファイルで指定します。

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。NFV 向けのこれらの heat テンプレートファイルの例については、DPDK SR-IOV YAML ファイルのサンプル を参照してください。

ハードウェア要件およびソフトウェア要件の項では、Red Hat OpenStack Platform director を使用した NFV 用 Heat テンプレートファイルのプランニングおよび設定の方法について説明します。

YAML ファイルを編集して NFV を設定することができます。YAML ファイル形式の概要は、YAML in a Nutshell を参照してください。

Data Plane Development Kit (DPDK) および Single Root I/O Virtualization (SR-IOV)

Red Hat OpenStack Platform (RHOSP) では、OVS-DPDK および SR-IOV 設定の自動化が追加され、NFV デプロイメントがサポートされています。

重要

Red Hat は、非 NFV ワークロードでの OVS-DPDK の使用をサポートしていません。NFV 以外のワークロードに OVS-DPDK 機能が必要な場合は、テクニカルアカウントマネージャー (TAM) に連絡するか、カスタマーサービスリクエストケースを開いて、サポートの例外やその他のオプションについて話し合ってください。カスタマーサービスリクエストケースを作成するには、ケースの作成 に移動し、**アカウント > カスタマーサービスリクエスト** を選択します。

ハイパーコンバージドインフラストラクチャー (HCI)

Compute サブシステムと Red Hat Ceph Storage ノードを共存させることができます。NFV ユースケースにおいて、このハイパーコンバージドモデルを使用すると、初期コストの軽減、初期デプロイメントのフットプリントの縮小、稼働率の最大化、管理の効率化などが実現されます。HCI の詳細は、**ハイパーコンバージドインフラストラクチャーのデプロイ** を参照してください。

コンポーザブルロール

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を作成できます。コンポーザブルロールを使用して、各ロールにサービスを追加/削除することができます。コンポーザブルロールに関する詳しい情報は、**Red Hat OpenStack Platform デプロイメントのカスタマイズ** の コンポーザブルサービスとカスタムロール を参照してください。

Open vSwitch (OVS) と LACP の組み合わせ

OVS 2.9 では、LACP と OVS の組み合わせが完全にサポートされています。この設定は、Openstack のコントロールプレーンのトラフィックには推奨されません。OVS または Openstack Networking で機能の中断が発生すると、管理機能が損なわれるためです。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** の Open vSwitch (OVS) のボンディングオプション を参照してください。

OVS ハードウェアオフロード

Red Hat OpenStack Platform では、一部制約はあるものの、OVS ハードウェアオフロードのデプロイメントをサポートしています。ハードウェアオフロードを使用した OVS のデプロイについては、OVS ハードウェアオフロードの設定 を参照してください。

Open Virtual Network (OVN)

RHOSP 16.1.4 では、以下の NFV OVN の設定を使用することができます。

- OVN と OVS-DPDK の組み合わせが SR-IOV と共存する設定のデプロイ
- OVN と OVS TC Flower オフロードを組み合わせた設定のデプロイ

1.3. NFV データプレーンの接続性

NFV が導入され、従来のデバイスを VNF として実装を始めているネットワークベンダーが増えています。これらのネットワークベンダーの大半が仮想マシンに着目していますが、選択した設計に合わせた

コンテナーベースのアプローチに注目しているベンダーもあります。OpenStack ベースのソリューションは、主に以下の 2 つの理由からリッチかつ柔軟でなければなりません。

- **アプリケーションの即応性**: ネットワークベンダーは現在、デバイスを VNF に変換している段階にあります。VNF 市場では、VNF ごとに成熟度レベルは異なっており、即応性に関する共通の障害として、API での RESTful インターフェイスの有効化、データモデルのステートレスへの進化、自動化管理オペレーションの提供などが挙げられます。OpenStack は、すべてに共通のプラットフォームを提供する必要があります。

- **幅広いユースケース**: NFV には、異なるユースケースに対応する多様なアプリケーションが含まれます。たとえば、Virtual Customer Premise Equipment (vCPE) は、ルーティング、ファイアウォール、仮想プライベートネットワーク (VPN)、ネットワークアドレス変換 (NAT) など多くのネットワーク機能を顧客のサイトで提供することを目的としています。Virtual Evolved Packet Core (vEPC) は、Long-Term Evolution (LTE) ネットワークのコアコンポーネントにコスト効果の高いプラットフォームを提供するクラウドアーキテクチャーで、ゲートウェイやモバイルエンドポイントをダイナミックにプロビジョニングして、スマートフォンやその他のデバイスからの増加するデータトラフィック量に対応できるようにします。
これらのユースケースは、異なるネットワークアプリケーションとプロトコルを使用して実装され、インフラストラクチャーとは異なる接続性、分離、パフォーマンスの特性を必要とします。また、コントロールプレーンのインターフェイスとプロトコル、実際の転送プレーンを分離することが一般的です。OpenStack には、さまざまなデータパスの接続性オプションを提供できるように十分な柔軟性が必要です。

基本的に、仮想マシンにデータプレーンの接続性を提供する一般的なアプローチは 2 種類あります。

- **ハードウェアへのダイレクトアクセス**: PCI パススルー (OpenStack では SR-IOV PF と表記) や、VF (Virtual Function) と PF (Physical Function) の両パススルー向けの Single Root I/O Virtualization (SR-IOV) などの技術を使用し、Linux カーネルを迂回して物理ネットワークインターフェイスカード (NIC) にセキュアな Direct Memory Access (DMA) を提供します。

- **仮想スイッチ (vswitch) の使用**: ハイパーバイザーのソフトウェアサービスとして実装されています。仮想マシンは、仮想インターフェイス (vNIC) を使用して vSwitch に接続され、vSwitch は仮想マシン間や仮想マシンと物理ネットワーク間のトラフィックを転送することができます。

高速データパスには、以下のようなオプションがあります。

- **Single Root I/O Virtualization (SR-IOV)** 単一の PCI ハードウェアデバイスを複数の仮想 PCI デバイスのように見せる標準のことです。これは、Physical Function (PF) を導入して機能します。PF とは、物理ハードウェアポートと Virtual Function (VF: 仮想マシンに割り当てられた軽量機能) の機能を果たすフル装備の PCIe 機能です。仮想マシンは、VF をハードウェアと直接通信する通常の NIC とみなします。NIC は複数の VF をサポートします。

- **Open vSwitch (OVS)**: 仮想化サーバー環境内で仮想スイッチとして使用されるように設計されたオープンソースのソフトウェアスイッチです。OVS は、通常の L2-L3 スイッチのケーパビリティーだけでなく、ユーザー定義のオーバーレイネットワーク (例: VXLAN) を作成する OpenFlow などの SDN プロトコルもサポートします。OVS は、物理 NIC を使用する仮想マシン間およびホスト全体のパケットの切り替えに、Linux のカーネルネットワークを使用します。OVS は、iptables/ebtables で Linux ブリッジのオーバーヘッドを回避する接続トラッキング (Conntrack) と内蔵のファイアウォール機能をサポートするようになりました。Red Hat OpenStack Platform 環境の Open vSwitch は、カスタマイズなしに OVS と OpenStack Networking (neutron) を統合できます。

- **Data Plane Development Kit (DPDK)**: 高速なパケット処理に向けてライブラリーセットや Poll Mode Driver (PMD) で設定されます。DPDK はユーザー空間で大半を実行するように設計されており、アプリケーションが NIC から (または NIC へ) 直接独自のパケット処理を実行できるよ

うになります。DPDK は、レイテンシーを減らし、処理するパケット数を増やすことができます。DPDK Poll Mode Drivers (PMDs) はビジーループで実行され、ホストの NIC ポートやゲストの vNIC ポートにパケットが到着していないかを絶えずスキャンします。

- **DPDK accelerated Open vSwitch (OVS-DPDK)**: Linux カーネルバイパスと物理 NIC への Direct Memory Access (DMA) を用いた高性能のユーザー空間ソリューションに向け DDPK がバンドルされた Open vSwitch。これは、標準の OVS カーネルデータパスを DPDK ベースのデータパスに置き換えて、内部で DPDK をパケット転送に使用するユーザー空間の vSwitch をホストに構築するという発想になります。このアーキテクチャーの利点は、大半がユーザーに透過的である点です。OpenFlow、OVSDB、コマンドラインなどの公開されるインターフェイスは、ほぼ同じ状態のままになります。

1.4. ETSI NFV アーキテクチャー

欧州電気通信標準化機構 (ETSI) は、ヨーロッパの情報通信技術 (ICT: Information and Communication Technology) の標準を開発する独立した標準化組織です。

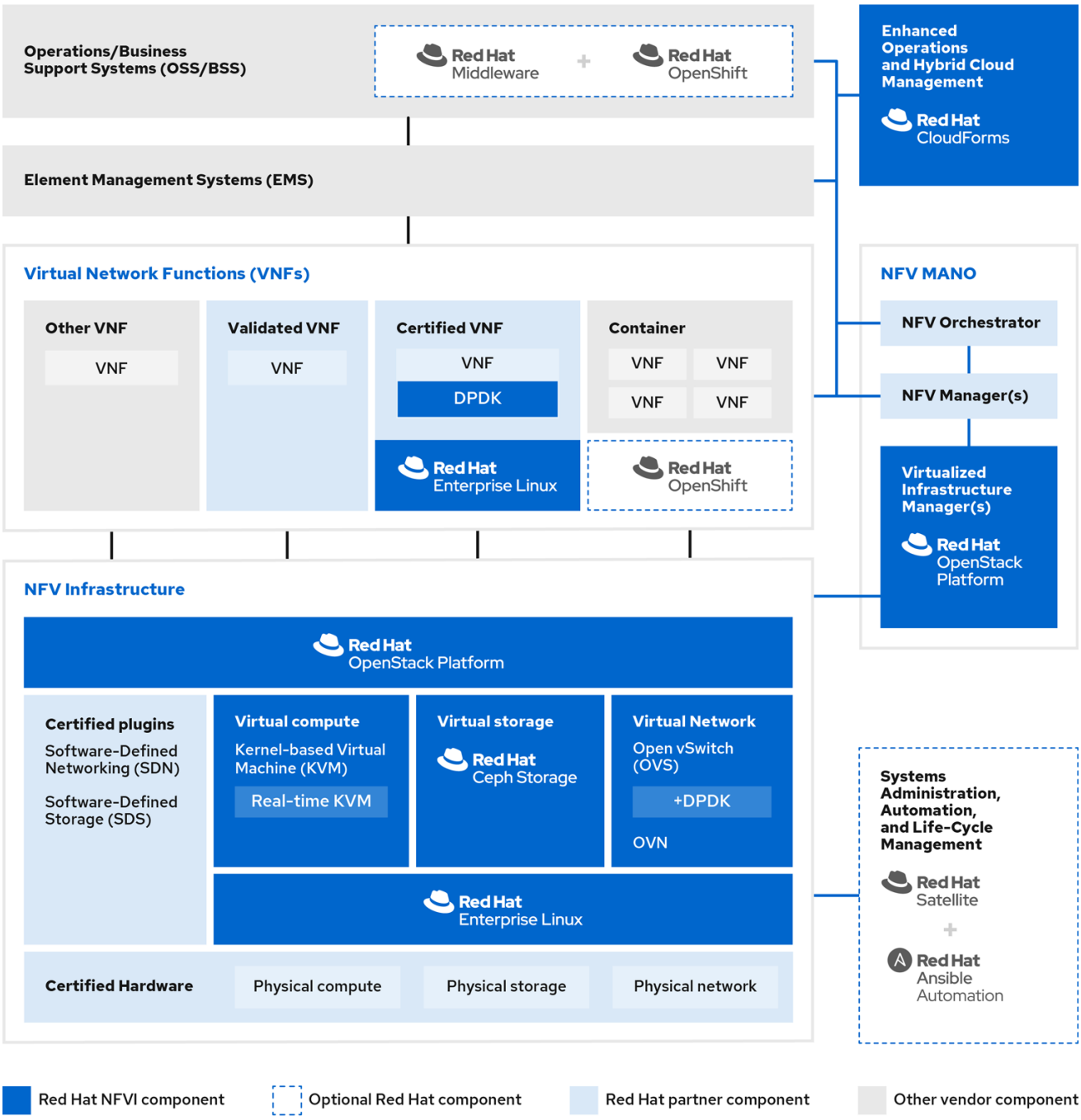
ネットワーク機能仮想化 (NFV) は、プロプライエタリーのハードウェアデバイスの使用に伴う問題への対処に重点を置いています。NFV を使用すると、ユースケースの要件や経済的な利点によっては、ネットワーク固有の設備をインストールする必要性が軽減されます。ETSI Industry Specification Group for Network Functions Virtualization (ETSI ISG NFV) は、VF を確実にサポートするために必要な要件、リファレンスアーキテクチャー、インフラストラクチャー仕様を設定します。

Red Hat では、通信事業者 (CSP) の IT とネットワークのコンバージェンス実現を支援するオープンソースベースのクラウド最適化ソリューションを提供しています。Red Hat は、Single Root I/O Virtualization (SR-IOV) や Open vSwitch with Data Plane Development Kit (OVS-DPDK) などの NFV 機能を Red Hat OpenStack に追加します。

1.5. NFV ETSI のアーキテクチャーおよびコンポーネント

通常、ネットワーク機能仮想化 (NFV) プラットフォームには、以下のコンポーネントが含まれます。

図1.1 NFV ETSI のアーキテクチャーおよびコンポーネント



- **仮想ネットワーク機能 (VNF)**: ルーター、ファイアウォール、ロードバランサー、ブロードバンドのゲートウェイ、モバイルパケットのプロセッサー、サービス提供ノード、シグナリング、位置情報サービスなどのネットワーク機能のソフトウェア実装。

- **ネットワーク機能仮想化インフラストラクチャー (NFVi)**: インフラストラクチャーを設定する物理リソース (Compute、ストレージ、ネットワーク) および仮想化層。このネットワークには、仮想マシン間およびホスト全体でパケットを転送するためのデータパスが含まれます。これにより、基盤のハードウェアの情報を考慮せずに VNF をインストールできます。NFVi は、NFV スタックの基盤を形成します。NFVi は、マルチテナントをサポートし、Virtual Infrastructure Manager (VIM) で管理されます。Enhanced Platform Awareness (EPA) は低レベルの CPU および NIC アクセラレーション機能を VNF に公開し、仮想マシンのパケット転送のパフォーマンス (スループット、レイテンシー、ジッター) を向上させます。

- **NFV Management and Orchestration (MANO)**: VNF のライフサイクル全体で必要とされる全サービス管理タスクにフォーカスする管理およびオーケストレーション層。MANO の主要な目的は、オペレーターが顧客に提供するネットワーク機能のサービス定義、自動化、エラーの相関関係、監視、ライフサイクル管理を物理インフラストラクチャーから切り離せるようにすることです。このような切り離しを行うには、Virtual Network Function Manager (VNFM) が提供する管理層が追加で必要になります。VNFM は、直接対話するか、VFN ベンダーが提供する Element Management System (EMS) を使用して、仮想マシンのライフサイクルや VNF を管理します。MANO が定義するコンポーネントでもう 1 つ重要なのは、NFVO として知られるオーケストレーターです。NFVO は、最上部のオペレーション/ビジネスサポートシステム (OSS/BSS) や、最下部の VNFM など、さまざまなデータベースやシステムにインターフェイスを提供します。NFVO は、顧客向けの新規サービスを構築する場合、VNFM に対して VNF のインスタンス化をトリガーするよう頼みます (これにより、複数の仮想マシンが作成される場合があります)。

- **オペレーション/ビジネスサポートシステム (OSS/BSS: Operations/Business Support System)**: オペレーションサポートや請求など必要不可欠なビジネス機能アプリケーションを提供します。OSS/BSS は、NFV に適応する必要があり、レガシーシステムと新規の MANO コンポーネントを統合しています。BSS システムは、サービスサブスクリプションをベースにポリシーを設定して、レポートと請求を管理します。

- **システム管理、自動化、ライフサイクル管理**: システム管理、インフラストラクチャーコンポーネントの自動化、および NFVi プラットフォームのライフサイクルを管理します。

1.6. RED HAT NFV のコンポーネント

Red Hat の NFV 向けのソリューションには、ETSI モデルに含まれる NFV フレームワークの異なるコンポーネントとしてのロールを果たすことのできる各種製品が含まれます。NFV ソリューションには、Red Hat ポートフォリオの以下の製品が統合されます。

- Red Hat OpenStack Platform: IT および NFV ワークロードをサポートします。Enhanced Platform Awareness (EPA) 機能は、SR-IOV や OVS-DPDK をサポートする CPU ピニング、ヒュージページ、Non-Uniform Memory Access (NUMA) アフィニティー、ネットワークアダプター (NIC) などを使用して、決定的なパフォーマンスの向上を図ることができます。

- Red Hat Enterprise Linux および Red Hat Enterprise Linux Atomic Host: VNF として仮想マシンやコンテナーを作成します。

- Red Hat Ceph Storage: サービスプロバイダーのワークロードのすべてのニーズに対応する弾力性があり、統一された高性能なストレージ層を提供します。

- Red Hat JBoss Middleware および Red Hat 提供の OpenShift Enterprise: オプションで OSS/BSS コンポーネントの最新化に使用することができます。

- Red Hat CloudForms: VNF マネージャーを提供し、統合された画面に、VIM や NFVI などの複数のソースのデータを表示することができます。

- Red Hat Satellite および Red Hat 提供の Ansible: オプションでシステムの管理、自動化、ライフサイクル管理を強化します。

1.7. NFV インストールの概要

Red Hat OpenStack Platform director は完全な OpenStack 環境をインストールおよび管理します。director は、アップストリームの OpenStack TripleO プロジェクトをベースとしています。TripleO とは、OpenStack-On-OpenStack の頭文字の O が 3 つあることを意味しています。このプロジェクトは、OpenStack コンポーネントを活用して、完全に機能する OpenStack 環境をインストールします。これには、アンダークラウドと呼ばれる、最小限の OpenStack ノードが含まれます。アンダークラウ

ドは、オーバークラウド (実稼働環境用の OpenStack ノードとして使用される一連のベアメタルシステム) のプロビジョニングと制御を行います。director は、リーンかつ堅牢な Red Hat OpenStack Platform の完全な環境を簡単にインストールできる方法を提供します。

アンダークラウドおよびオーバークラウドのインストールに関する詳細は、**Red Hat OpenStack Platform director を使用した Red Hat OpenStack Platform のインストールと管理** を参照してください。

NFV 機能をインストールするには、以下の手順を実施します。

- **network-environment.yaml** ファイルへの SR-IOV および PCI パススルーのパラメーターの追加、CPU チューニング用 **post-install.yaml** ファイルの更新、**compute.yaml** ファイルの変更、そして **overcloud_deploy.sh** スクリプトの実行により、オーバークラウドをデプロイします。

- NIC から直接データをポーリングして、高速パケット処理用の DPDK ライブラリーおよびドライバーをインストールします。**network-environment.yaml** ファイルへの DPDK パラメーターの追加、CPU チューニング用 **post-install.yaml** ファイルの更新、**compute.yaml** ファイルの更新による DPDK ポートでのブリッジの設定、**controller.yaml** ファイルの更新によるブリッジと VLAN が設定されたインターフェイスの設定、そして **overcloud_deploy.sh** の実行により、オーバークラウドをデプロイします。

第2章 NFV パフォーマンスに関する考慮事項

ネットワーク機能仮想化 (NFV) ソリューションが有用であるためには、VF が物理実装のパフォーマンス以上である必要があります。Red Hat の仮想化技術は、OpenStack およびクラウドのデプロイメントで一般的に使用されている高フォーマンスの Kernel-based Virtual Machine (KVM) ハイパーバイザーをベースにしています。

Red Hat OpenStack Platform director は、ゲスト仮想ネットワーク機能 (VNF) 用にラインレートパフォーマンスを実現するために、リソースの分割および微調整を実施するように Compute ノードを設定します。NFV のユースケースにおける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、およびジッターです。

Data Plane Development Kit (DPDK) で高速化した仮想マシンを使用して、物理 NIC と仮想マシン間で高速なパケット切り替えを有効にすることができます。OVS 2.10 は、DPDK 17 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 用のラインレートパフォーマンスを提供します。

Single Root I/O Virtualization (SR-IOV) ネットワークでは、特定ネットワークや仮想マシンのスループット向上など、強化されたパフォーマンスが提供されます。

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ー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。

2.1. CPU と NUMA ノード

以前は、x86 システムの全メモリーは、システム内のどの CPU からでも同等にアクセスできていました。これにより、システム内で操作を行う CPU、Uniform Memory Access (UMA) を参照する CPU はどれでも、メモリーのアクセス時間が同じでした。

Non-Uniform Memory Access (NUMA) では、システムメモリーは、特定の CPU またはソケットに割り当てられるノードと呼ばれるゾーンに分割されます。CPU のローカルにあるメモリーには、そのシステムのリモートの CPU に接続されているメモリーにアクセスするよりも高速です。通常、NUMA システム上のソケットにはそれぞれローカルのメモリーノードがあり、別の CPU のローカルにあるノードのメモリーや、全 CPU で共有されるバス上のメモリーよりも、コンテンツに早くアクセスできます。

同様に、物理 NIC は Compute ノードのハードウェア上の PCI スロットに配置されます。これらのスロットは、特定の NUMA ノードに関連付けられる特定の CPU ソケットに接続されます。パフォーマンスを最適化するには、CPU の設定 (SR-IOV または OVS-DPDK) と同じ NUMA ノードにデータパス NIC を接続します。

NUMA を使用しない場合のパフォーマンスへの影響は大きく、一般的に、パフォーマンス の 10 % 以上が影響を受けます。各 CPU ソケットには、仮想化を目的とする個別の CPU として扱われる複数の CPU コアを配置することができます。

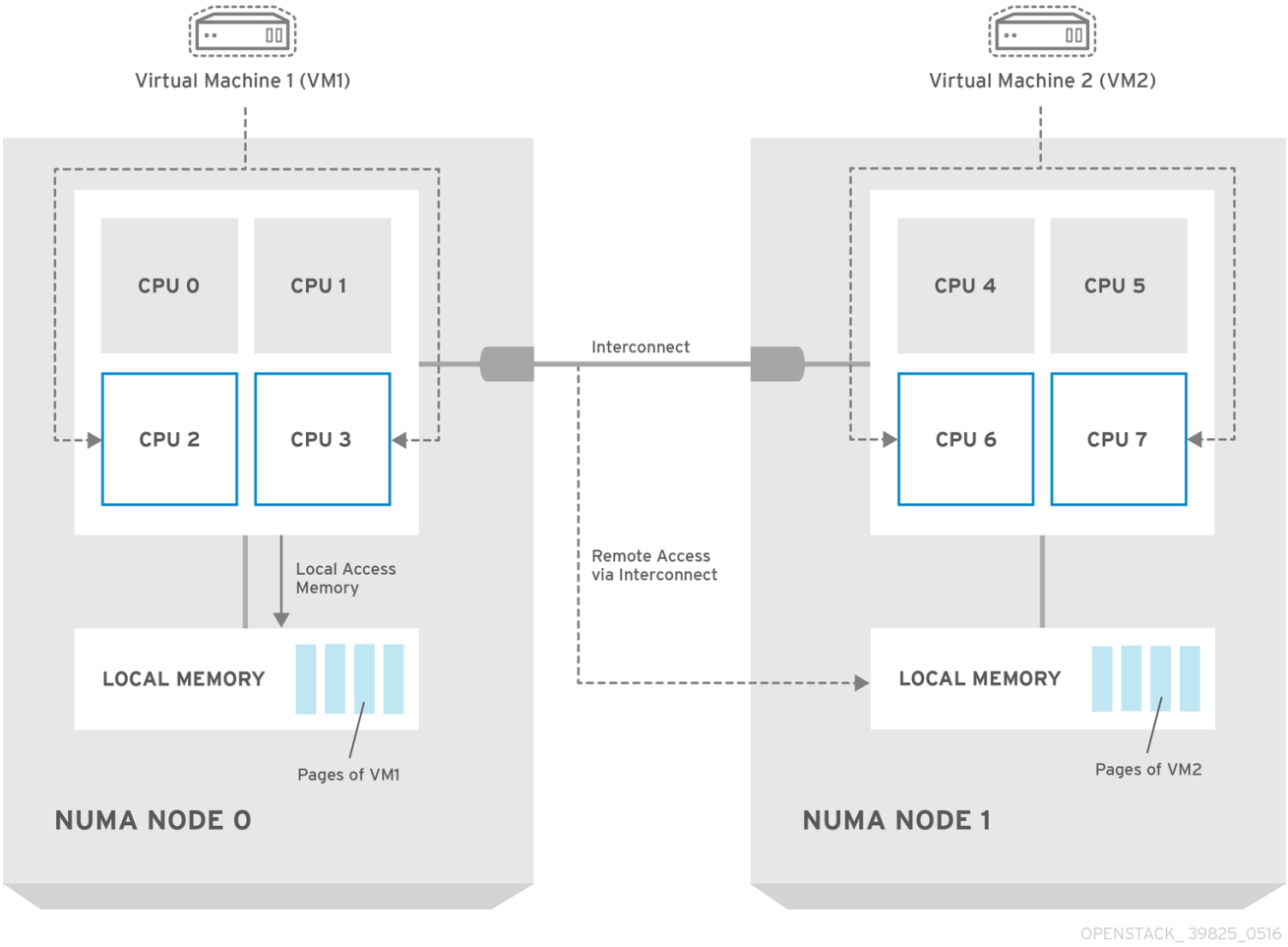
ヒント

NUMA の詳細は、What is NUMA and how does it work on Linux? を参照してください。

2.1.1. NUMA ノードの例

以下の図は、2 つのノードからなる NUMA システムの例と、CPU コアとメモリーページが利用可能になる仕組みを示しています。

図2.1 例: 2 ノードの NUMA システム



注記

Interconnect 経由で利用可能なリモートのメモリーには、NUMA ノード 0 からの VM1 に NUMA ノード 1 の CPU コアがある場合 **のみ**、アクセスすることができます。この場合、NUMA ノード 1 のメモリーは、VM1 の 3 番目の CPU コアのローカルとして機能します (例: 上記の図では VM1 は CPU4 が割り当てられています) が、それと同時に、同じ仮想マシンの別の CPU コアに対してはリモートメモリーとして機能します。

2.1.2. NUMA 対応のインスタンス

NUMA アーキテクチャーを持つシステム上で NUMA トポロジー認識を使用するように OpenStack 環境を設定することができます。仮想マシン (VM) でゲストオペレーティングシステムを実行する場合は、NUMA トポロジーが 2 つあります。

- ホストの物理ハードウェアの NUMA トポロジー
- ゲストオペレーティングシステムに公開される仮想ハードウェアの NUMA トポロジー

仮想ハードウェアを物理ハードウェア NUMA トポロジーに合わせて調整することで、ゲストオペレーティングシステムのパフォーマンスを最適化することができます。

2.2. CPU ピニング

CPU ピニングは、指定のホスト内にある特定の物理 CPU 上で特定の仮想マシンの仮想 CPU を実行する機能のことです。vCPU ピニングでは、ベアメタルシステムへのピニングタスクと同様の利点が得ら

れます。仮想マシンは、ホストのオペレーティングシステムのユーザー空間タスクとして実行されるので、ピニングすることでキャッシュの効率性が向上します。

CPU ピニングの設定方法の詳細は、link:https://access.redhat.com/documentation/ja-jp/red_hat_openstack_platform/17.1/html/configuring_the_compute_service_for_instance_creation/assem cpus-on-compute-nodes#assembly_configuring-cpu-pinning-on-compute-nodes_cpu-pinning [Configuring CPU pinning on Compute nodes] in the **インスタンス作成のための Compute サービスの設定** ガイドを参照してください。

2.3. ヒュージページ

物理メモリーは、ページと呼ばれる連続した一連のリージョンに分割されます。効率化を図るため、システムは、各メモリーバイトにアクセスするのではなく、ページ全体にアクセスしてメモリーを取得します。このような変換を実行するには、システムは、最近使用されたページまたは頻繁に使用されるページの物理から仮想アドレスへのマッピングが含まれるトランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffers) をチェックします。検索したマッピングが TLB にない場合には、プロセッサーは全ページテーブルで同じ処理を反復して、アドレスマッピングを判断する必要があります。TLB を最適化し、これらの TLB ミス時に発生するパフォーマンスペナルティーを最小限に抑えます。

X86 システムの通常のページサイズは 4 KB ですが、他の大きなページサイズを利用することもできます。ページサイズが大きいと全体的なページ数が少なくなるので、TLB に仮想から物理アドレスへの変換を保管可能なシステムメモリー量が増えることになります。その結果、TLB ミスの可能性が低くなり、パフォーマンスが向上します。ページサイズが大きいと、プロセスはページに割り当てる必要があるため、メモリーを無駄にする可能性が高くなりますが、すべてのメモリーが必要となることはあまりありません。そのため、ページサイズを選択する際には、より大きいページを使用してアクセス時間を速くするか、より小さいページを使用して最大限にメモリーが使用されるようにするかで、トレードオフが生じます。

2.4. ポートセキュリティー

ポートセキュリティーは、不正なアクセスを防ぐ手段です。発信元ネットワークポートのソース IP およびソース MAC アドレスと一致しない送信トラフィックをブロックします。セキュリティーグループルールを使用して、この挙動を監視または変更することはできません。

デフォルトでは、OpenStack で新たに作成される Neutron ネットワークの **port_security_enabled** パラメーターは、**enabled** に設定されます。ネットワーク上で新たに作成されるポートは、そのネットワークから **port_security_enabled** パラメーターの値をコピーします。

ファイアウォールまたはルーターの構築など一部の NFV のユースケースでは、ポートセキュリティーを無効にしなければならない場合があります。

特定のポートでポートセキュリティーを無効にするには、以下のコマンドを実行します。

```
openstack port set --disable-port-security <port-id>
```

ネットワークで作成されるすべての新規ポートで、ポートセキュリティーの有効化を阻止するには、以下のコマンドを実行します。

```
openstack network set --disable-port-security <network-id>
```

第3章 NFV のハードウェア要件

本項では、NFV のハードウェア要件について説明します。

Red Hat は、Red Hat OpenStack Platform で使用するハードウェアを認定します。詳細は、認定済みハードウェア を参照してください。

3.1. NFV 向けのテスト済み NIC

NFV 向けのテスト済み NIC のリストは、Network Adapter Fast Datapath Feature Support Matrix を参照してください。

NVIDIA (Mellanox) ネットワークインターフェイスで OVS-DPDK を設定している場合を除き、サポート対象の NIC のデフォルトドライバーを使用してください。NVIDIA ネットワークインターフェイスの場合は、j2 ネットワーク設定テンプレートで該当するカーネルドライバーを設定する必要があります。

例

この例では、**mlx5_core** ドライバーが Mellanox ConnectX-5 ネットワークインターフェイスに設定されています。

```
members
 - type: ovs_dpdk_port
    name: dpdk0
    driver: mlx5_core
    members:
    - type: interface
      name: enp3s0f0
```

3.2. ハードウェアオフロードに関するトラブルシューティング

Red Hat Open Stack Platform (RHOSP) 17.1 デプロイメントでは、OVS ハードウェアオフロードは、**switchdev**対応ポートおよび Mellanox ConnectX5 NIC を備えた VM のフローをオフロードしない場合があります。このシナリオのフローのオフロードに関するトラブルシューティングと設定を行うには、**ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox ファームウェアパラメーターを無効にします。RHOSP 17.1 での OVS ハードウェアオフロードのトラブルシューティング情報については、Red Hat ナレッジベースのソリューション OVS Hardware Offload with Mellanox NIC in OpenStack Platform 16.2 を参照してください。

手順

1. 設定する Mellanox NIC を備えた RHOSP デプロイメントの Compute ノードにログインします。
2. **mstflint**ユーティリティーを使用して、**ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox ファームウェアパラメーターを照会します。

```
[root@compute-1 ~]# yum install -y mstflint
[root@compute-1 ~]# mstconfig -d <PF PCI BDF> q
ESWITCH_IPV4_TTL_MODIFY_ENABLE
```

3. **ESWITCH_IPV4_TTL_MODIFY_ENABLE**パラメーターが有効で**1**に設定されている場合は、値を**0**に設定して無効にします。

```
[root@compute-1 ~]# mstconfig -d <PF PCI BDF> s
ESWITCH_IPV4_TTL_MODIFY_ENABLE=0`
```

4. ノードを再起動します。

3.3. 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、最高のパフォーマンスが得られるように、Compute ノードの NUMA トポロジーを理解して CPU およびメモリーのリソースを分割する必要があります。NUMA 情報を確認するには、以下のいずれかのタスクを実行します。

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

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。アンダークラウド設定の詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理ガイド**を参照してください。

3.4. 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、オーバークラウド設定の追加ハードウェア情報を取得する機能がデフォルトで有効です。**undercloud.conf** ファイル内の **Inspection_extras** パラメーターの詳細は、Director 設定パラメーター を参照してください。

たとえば、**numa_topology** コレクターは、追加ハードウェアイントロスペクションの一部で、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

手順

- 上記の情報を取得するには、<UUID> をベアメタルノードの UUID に置き換えて、以下のコマンドを実行します。

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

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
    }
  ]
}
```

3.5. NFV BIOS 設定

以下の表に NFV に必要な BIOS 設定をまとめます。

注記

BIOS で SR-IOV グローバルおよび NIC 設定を有効にする必要があります。そうしないと、SR-IOV Compute ノードを使用した Red Hat OpenStack Platform (RHOSP) のデプロイメントが失敗します。

表3.1 BIOS 設定

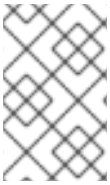
パラメーター	設定
C3 Power State	Disabled
C6 Power State	Disabled
MLC Streamer	Enabled
MLC Spatial Prefetcher	Enabled
DCU Data Prefetcher	Enabled
DCA	Enabled
CPU Power and Performance	Performance
Memory RAS and Performance Config → NUMA Optimized	Enabled
Turbo Boost	確定的なパフォーマンスを必要とする NFV デプロイメントでは無効になっています。 他のすべてのシナリオで有効です。
VT-d	Intel カードで VFIO 機能が必要な場合には Enabled
NUMA memory interleave	Disabled

intel_idle ドライバーを使用するプロセッサーでは、Red Hat Enterprise Linux は BIOS 設定を無視し、プロセッサーの C ステートを再度有効にすることができます。

カーネルブートコマンドラインでキーと値のペア **intel_idle.max_cstate=0** を指定すると、**intel_idle** を 無効にし、代わりに **acpi_idle** ドライバーを使用できます。

current_driver の内容をチェックして、プロセッサーが **acpi_idle** ドライバーを使用していることを確認します。

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
acpi_idle
```



注記

Tuned デーモンの起動に時間がかかるため、ドライバーを変更した後は多少の遅延が発生します。ただし、Tuned のロード後、プロセッサーはより深い C ステートを使用しません。

第4章 NFV のソフトウェア要件

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。

4.1. リポジトリーの登録と有効化

Red Hat OpenStack Platform をインストールするには、Red Hat OpenStack Platform director 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。アンダークラウドの登録および更新に関する詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** の アンダークラウドの登録およびサブスクリプションのアタッチ を参照してください。

手順

1. コンテンツ配信ネットワークにシステムを登録します。プロンプトが表示されたら、カスタマーポータルのユーザー名とパスワードを入力します。

```
[stack@director ~]$ sudo subscription-manager register
```

2. Red Hat OpenStack Platform director のエンタイトルメントプール ID を確認します (たとえば、以下のコマンド出力の {Pool ID})。

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             {Pool-ID}-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

3. 次のコマンドに **Pool ID** の値を含めて、Red Hat Open Stack Platform 17.1 のエンタイトルメントをアタッチします。

```
[stack@director ~]$ sudo subscription-manager attach --pool={Pool-ID}-123456
```

4. デフォルトのリポジトリーを無効にします。

```
subscription-manager repos --disable=*
```

5. Red Hat OpenStack Platform で NFV を使用するのに必要なリポジトリーを有効にします。

```
$ sudo subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms \
--enable=rhel-9-for-x86_64-appstream-eus-rpms \
--enable=rhel-9-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-9-x86_64-rpms \
--enable=openstack-17.1-for-rhel-9-x86_64-rpms \
--enable=rhel-9-for-x86_64-nfv-rpms \
--enable=fast-datapath-for-rhel-9-x86_64-rpms
```

6. システムを更新して、ベースシステムパッケージが最新の状態になるようにします。

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

4.2. NFV デプロイメントでサポートされている設定

Red Hat OpenStack Platform (RHOSP) では、director を使用して以下の NFV デプロイメントがサポートされます。

- Single Root I/O Virtualization (SR-IOV)
  詳細は、SR-IOV の設定 を参照してください。
- Open vSwitch ハードウェアオフロード
  詳細は、OVS ハードウェアオフロードの設定 を参照してください。
- Data Plane Development Kit 対応 Open vSwitch (OVS-DPDK)
  詳細は、OVS-DPDK デプロイメントの設定 を参照してください。

また、以下いずれかを指定して RHOSP をデプロイすることもできます。

- コンポーザブルサービスとカスタムロールの実装。
  詳細は、**Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドの コンポーザブルサービスとカスタムロール を参照してください。
- Compute サービスと Ceph Storage サービスが同一ホスト上で共存。
  詳細は、**ハイパーコンバージドインフラストラクチャーのデプロイ** を参照してください。
- Red Hat Enterprise Linux Real Time KVM (RT-KVM)の設定。
  詳細は、NFV ワークロード向け RT-KVM の有効化 を参照してください。

4.3. NFV で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リストは Component, Plug-In, and Driver Support in Red Hat OpenStack Platform を参照してください。

Red Hat OpenStack Platform デプロイメントと NFV の組み合わせ向けにテスト済みの NIC のリストは、NFV 向けテスト済み NIC を参照してください。

4.4. サードパーティー製のソフトウェアとの互換性

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リストは、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ー別にリストをフィルタリングすることができます。

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リストは、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ー別にリストをフィルタリングすることができます。

第5章 NFV のネットワークに関する考慮事項

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使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インターフェイスには、ルーティング可能な IP アドレスが必要です。この IP アドレスは、静的に定義されたアドレスまたは外部の DHCP サービスから動的に生成されたアドレスのいずれかです。

最小限のオーバークラウドのネットワーク設定には、以下の NIC 設定が含まれます。

- シングル NIC 設定: ネイティブ VLAN 上のプロビジョニングネットワークと、オーバークラウドネットワークの種別ごとのサブネットを使用するタグ付けされた VLAN 用に NIC を 1 つ。
- デュアル NIC 設定: 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 設定: ネイティブ VLAN 上のプロビジョニングネットワーク用の NIC を 1 つと、オーバークラウドネットワークの種別ごと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 設定 - 各 NIC は、オーバークラウドネットワークの種別ごとのサブセットを使用します。

ネットワーク要件の詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理ガイド** の アンダークラウドネットワークの準備 を参照してください。

第6章 SR-IOV デプロイメントのプランニング

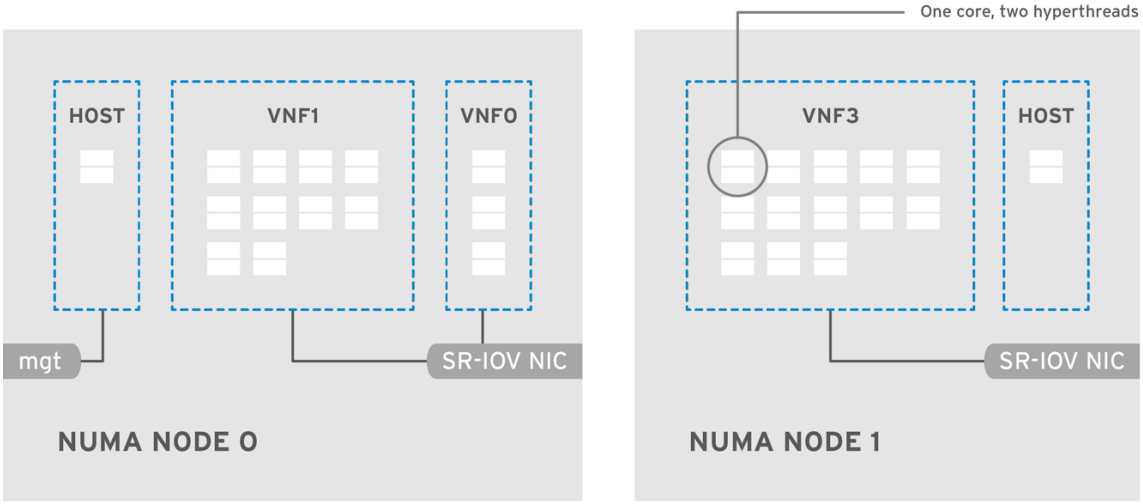
Compute ノードのハードウェアに応じて個別のパラメーターを設定し、NFV 向けの Single Root I/O Virtualization (SR-IOV) デプロイメントを最適化します。

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 ノードのトポロジーについての理解 を参照してください。

6.1. 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します。

図6.1 NUMA ノードトポロジー

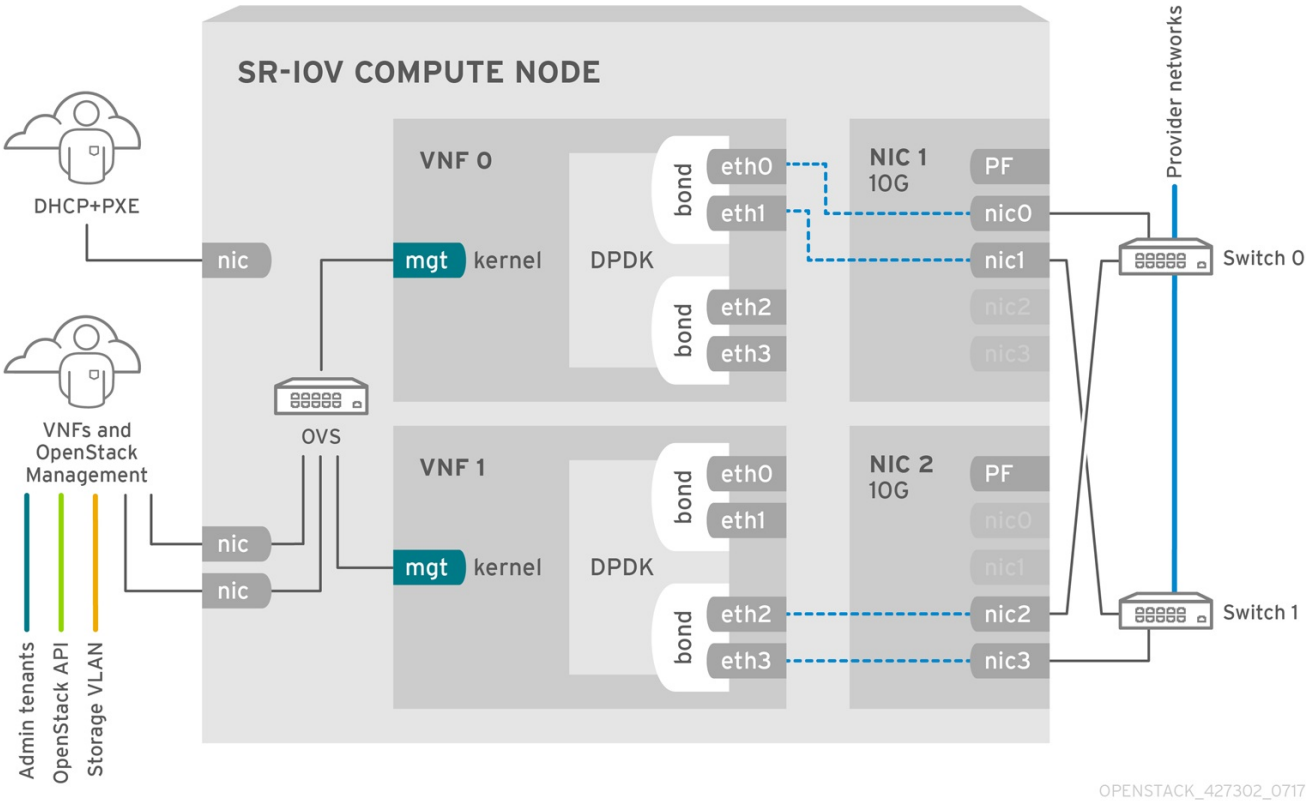


標準的なトポロジーでは、デュアルコアソケットの Compute ノード上の NUMA ノードにはそれぞれ 14 のコアが実装されます。HT (ハイパースレッド) および非 HT のコアがサポートされています。各コアには 2 つのシブリングスレッドがあります。1 つのコアは、各 NUMA ノード上のホスト専用です。仮想ネットワーク機能 (VNF) は SR-IOV インターフェイスのボンディングを処理します。すべての割り込み要求 (IRQ) はホストのコア上でルーティングされます。VNF コアは VNF 専用です。これらのコアは、他の VNF からの分離と、ホストからの分離を提供します。各 VNF 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もあります。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ロスの原因となり、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 のリストに基づいて、ブートパラメーターや Red Hat OpenStack Platform の変更を定義します。

6.2. 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、**mgt** で示された管理インターフェイスおよびデータプレーンインターフェイスがあります。管理インターフェイスは **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イスは 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イスをボンディングするためです。この図には、冗長性を確保するための 2 つのプロバイダーネットワークも示されています。Compute 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。

図6.2 NFV SR-IOV トポロジー

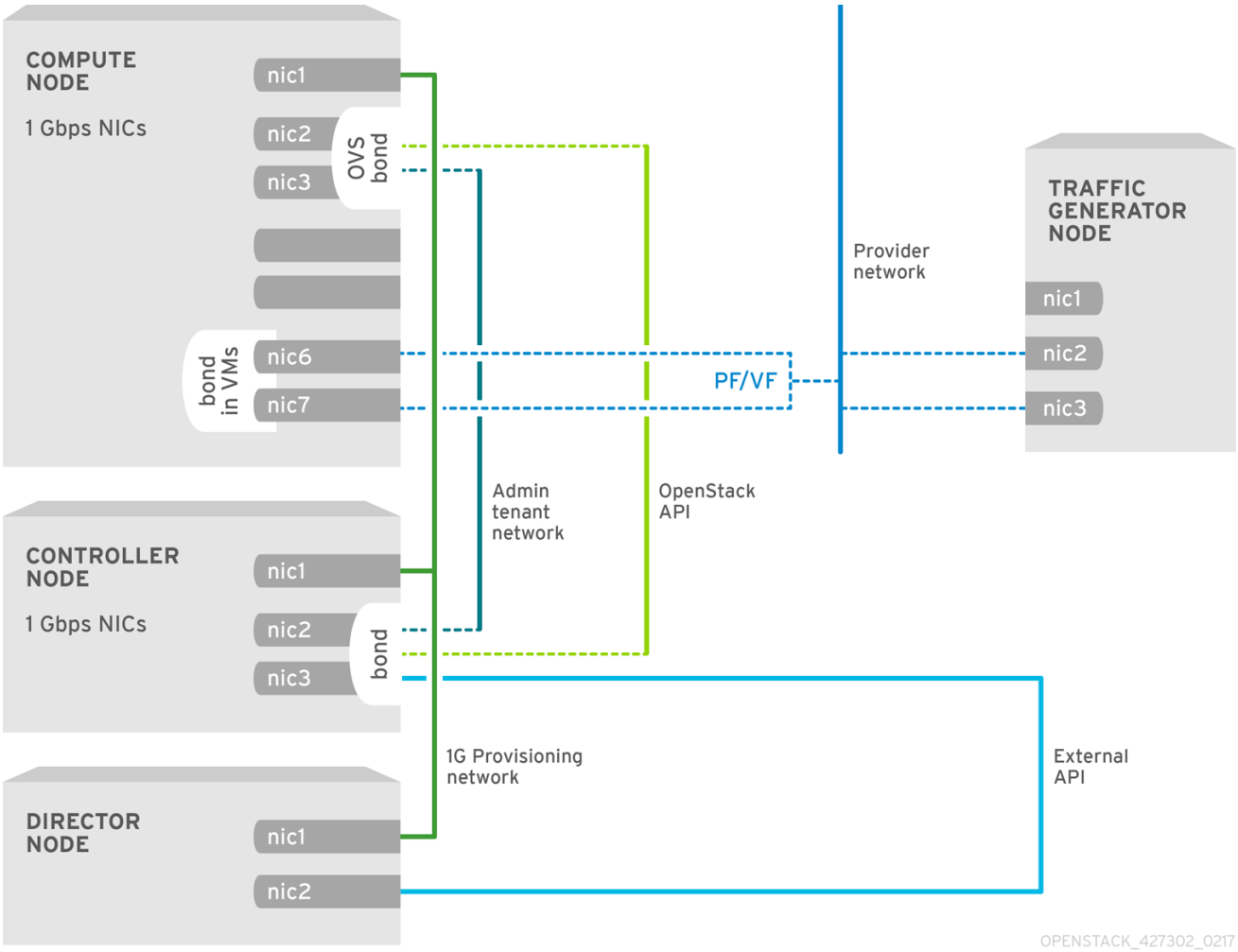


この図は、アプリケーションレベルで DPDK を使用し、SR-IOV Virtual Function (VF) および Physical Function (PF) へのアクセスが可能な VNF を示しています。これにより、ファブリックの設定に応じて可用性またはパフォーマンスが向上します。DPDK はパフォーマンスを向上させる一方、VF/PF DPDK のボンディングはフェイルオーバーおよび高可用性に対応します。VNF ベンダーは、DPDK Poll Mode Driver (PMD) による SR-IOV カード (VF/PF として公開されている) のサポートを確実にする必要があります。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。VNF への初回の接続にそのデバイスを使用して、DPDK アプリケーションに 2 つの VF/PF をボンディングさせることができます。

6.3. HCI を使用しない NFV SR-IOV のトポロジー

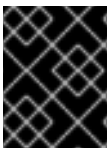
以下の図には、NFV 向けのハイパーコンバージドインフラストラクチャー (HCI) を使用しない SR-IOV のトポロジーを示しています。この環境は、1 Gbps の NIC を搭載した Compute ノードおよびコントローラーノードと、director ノードで設定されます。

図6.3 HCI を使用しない NFV SR-IOV トポロジー



第7章 SR-IOV デプロイメントの設定

Red Hat OpenStack Platform NFV デプロイメントでは、仮想リソースを通じたインスタンスから共有 PCIe リソースへの直接アクセスを設定した場合、Single Root I/O Virtualization (SR-IOV) を使用してより高いパフォーマンスが得られます。



重要

このセクションには、トポロジーとユースケースに合わせて変更する必要がある例が含まれています。詳細は、NFV のハードウェア要件 を参照してください。

前提条件

- RHOSP アンダークラウド。
  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。詳細は、**director を使用した Red Hat OpenStack Platform のインストールおよび管理** を参照してください。

  注記

  RHOSP director は、テンプレートとカスタム環境ファイルで指定したキーと値のペアを通じて SR-IOV 設定ファイルを変更します。SR-IOV ファイルを直接変更しないでください。

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- NIC を含むホストへのアクセス。
- NIC ファームウェアを常に最新の状態に保ってください。
  Yum または **dnf** 更新ではファームウェアの更新が完了しない可能性があります。詳細は、ベンダーのドキュメントを参照してください。

手順

Red Hat OpenStack Platform (RHOSP) ディレクターを使用して、SR-IOV 環境に RHOSP をインストールおよび設定します。大まかな手順は次のとおりです。

1. **director を使用した Red Hat OpenStack Platform のインストールと管理**の オーバークラウドネットワークの設定 の手順に従って、ネットワーク設定ファイル **network_data.yaml** を作成し、オーバークラウドの物理ネットワークを設定します。
2. ロールとイメージファイルを生成します。
3. SR-IOV 用に PCI パススルーデバイスを設定します。
4. ロール固有のパラメーターと設定のオーバーライドを追加します。
5. ベアメタルノード定義ファイルを作成します。
6. SR-IOV 用の NIC 設定テンプレートを作成します。
7. (オプション) NIC をパーティション分割します。
8. オーバークラウドネットワークと仮想 IP をプロビジョニングします。
   詳細は以下を参照してください。

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング

9. オーバークラウドベアメタルノードをプロビジョニングします。
詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング を参照してください。
10. SR-IOV オーバークラウドをデプロイします。

関連情報

- 「NIC パーティションの設定例」
- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのホストアグリゲートの作成」
- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのインスタンスの作成」

7.1. SR-IOV のロールとイメージファイルの生成

Red Hat OpenStack Platform (RHOSP) ディレクターは、ロールを使用してノードにサービスを割り当てます。SR-IOV 環境に RHOSP をデプロイする場合、**ComputeSriov** は、デフォルトのコンピューティングサービスに加えて、**NeutronSriovAgent** サービスを含む RHOSP インストールで提供されるデフォルトのロールです。

アンダークラウドのインストールには、コンテナーイメージの取得先およびその保存方法を定義するための環境ファイルが必要です。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **Controller**と**ComputeSriov**のロールを含む**roles_data_compute_sriov.yaml**という名前の新しいロールデータファイルを生成します。

```
$  openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_compute_sriov.yaml \
 Controller ComputeSriov
```

注記

RHOSP 環境で OVS-DPDK、SR-IOV、および OVS ハードウェアオフロードなどの複数のテクノロジーを使用している場合は、すべてのロールを含めるために 1 つのロールデータファイルを生成します。

```
$ openstack overcloud roles generate -o /home/stack/templates/\
roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
Compute:ComputeOvsHwOffload
```

4. イメージファイルを生成するには、**openstack tripleo container image prepare** コマンドを実行します。以下の入力が必要です。
   - 前の手順で生成したロールデータファイル (例: **roles_data_compute_sriov.yaml**)。
   - Networking サービスメカニズムドライバーに適した SR-IOV 環境ファイル:
     - ML2/OVN 環境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml**
     - ML2/OVS 環境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml**

       例

       この例では、**overcloud_images.yaml** ファイルが ML2/OVS 環境用に生成されています。

       ```
       $ sudo openstack tripleo container image prepare \
         --roles-file ~/templates/roles_data_compute_sriov.yaml \
         -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
         -e ~/containers-prepare-parameter.yaml \
         --output-env-file=/home/stack/templates/overcloud_images.yaml
       ```

5. 作成したロールデータファイルとイメージファイルのパスとファイル名をメモします。これらのファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「SR-IOV 用の PCI パススルーデバイスの設定」 に進みます。

関連情報

- 詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール を参照してください。
- **director を使用した Red Hat OpenStack Platform のインストールと管理**で コンテナーイメージを準備 します。

7.2. SR-IOV 用の PCI パススルーデバイスの設定

SR-IOV 環境用に Red Hat OpenStack Platform をデプロイする場合は、カスタム環境ファイルで SR-IOV コンピュートノードの PCI パススルーデバイスを設定する必要があります。

前提条件

- PCI カードが搭載されている 1 台以上の物理サーバーにアクセスします。
- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. PCI カードを備えた物理サーバーで、次のコマンドのいずれかを使用します。
   - オーバークラウドがデプロイされている場合:

     ```
     $ lspci -nn -s  <pci_device_address>
     ```

     出力例

     ```
     3b:00.0 Ethernet controller [0200]: Intel Corporation Ethernet
     Controller X710 for 10GbE SFP+ [<vendor_id>: <product_id>] (rev 02)
     ```

   - オーバークラウドがデプロイされていない場合:

     ```
     $ openstack baremetal introspection data save <baremetal_node_name> | jq
     '.inventory.interfaces[] | .name, .vendor, .product'
     ```

2. SR-IOV コンピュートノード上の PCI パススルーデバイスのベンダー ID と製品 ID を保持します。これらの ID は後の手順で必要になります。
3. アンダークラウドに **stack** ユーザーとしてログインします。
4. **stackrc** ファイルを取得します。

   ```
   $ source ~/stackrc
   ```

5. カスタム環境 YAML ファイル (例: **sriov-overrides.yaml**) を作成します。次の内容をファイルに追加して、SR-IOV コンピュートノードの PCI パススルーデバイスを設定します。

   ```
   parameter_defaults:
     ComputeSriovParameters:
       ...
       NovaPCIPassthrough:
         - vendor_id: "<vendor_id>"
           product_id: "<product_id>"
           address: <NIC_address>
           physical_network: "<physical_network>"
       ...
   ```

   - **<vendor_id>** を PCI デバイスのベンダー ID に置き換えます。
   - **<product_id>** を PCI デバイスの製品 ID に置き換えます。
   - **<NIC_address>** を PCI デバイスのアドレスに置き換えます。

- **<physical_network>** を、PCI デバイスが配置されている物理ネットワークの名前に置き換えます。

注記

NIC のデバイス名は変更される可能性があるため、PCI パススルーを設定する場合は **devname** パラメーターを使用しないでください。PF で Networking サービス (neutron) ポートを作成するには、**NovaPCIPassthrough** に **vendor_id**、**product_id**、および PCI デバイスアドレスを指定し、**--vnic-type direct-physical** オプションでポートを作成します。Virtual Function (VF) に Networking サービスのポートを作成するには、**NovaPCIPassthrough**で**vendor_id**と**product_id**を指定し、**--vnic-type direct**オプションを使用してポートを作成します。**vendor_id**および**product_id**パラメーターの値は、Physical Function (PF) コンテキストと VF コンテキストの間で異なる場合があります。

6. また、カスタム環境ファイルで、Compute サービス (nova) がノードをフィルター処理するために使用する **NovaSchedulerEnabledFilters** パラメーター用のフィルターのリストに **PciPassthroughFilter** と **AggregateInstanceExtraSpecsFilter** が含まれていることを確認します。

```
parameter_defaults:
  ComputeSriovParameters:
    ...
    NovaPCIPassthrough:
      - vendor_id: "<vendor_id>"
        product_id: "<product_id>"
        address: <NIC_address>
        physical_network: "<physical_network>"
    ...
    NovaSchedulerEnabledFilters:
      - AvailabilityZoneFilter
      - ComputeFilter
      - ComputeCapabilitiesFilter
      - ImagePropertiesFilter
      - ServerGroupAntiAffinityFilter
      - ServerGroupAffinityFilter
      - PciPassthroughFilter
      - AggregateInstanceExtraSpecsFilter
```

7. 作成したカスタム環境ファイルのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「ロール固有のパラメーターと設定のオーバーライドの追加」 に進みます。

関連情報

- **インスタンス作成のためのコンピュートサービスの設定** における **NovaPCIPassthrough** の設定に関するガイドライン

7.3. ロール固有のパラメーターと設定のオーバーライドの追加

SR-IOV コンピュートノードのロール固有のパラメーターを追加し、SR-IOV 環境をデプロイするときに Red Hat OpenStack Platform (RHOSP) ディレクターが使用するカスタム環境 YAML ファイル内のデフォルトの設定値をオーバーライドできます。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「SR-IOV 用の PCI パススルーデバイスの設定」 に作成したカスタム環境 YAML ファイルを開くか、新しいものを作成します。
4. SR-IOV Compute ノードのロール固有のパラメーターをカスタム環境ファイルに追加します。

例

```
  ComputeSriovParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
```

5. RHOSP director が SR-IOV を設定するために使用する設定のデフォルトを確認します。これらのデフォルトはファイルで提供されており、メカニズムドライバーによって異なります。
   - ML2/OVN
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-sriov.yaml
   - ML2/OVS
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-sriov.yaml
6. 設定のデフォルトをオーバーライドする必要がある場合は、オーバーライドをカスタム環境ファイルに追加します。
   たとえば、このカスタム環境ファイルでは、Nova PCI ホワイトリスト値を追加したり、ネットワークタイプを設定したりできます。

例

この例では、ネットワークサービス (neutron) ネットワークタイプが VLAN に設定され、テナントの範囲が追加されます。

```
parameter_defaults:
  NeutronNetworkType: 'vlan'
  NeutronNetworkVLANRanges:
```

```
    - tenant:22:22
    - tenant:25:25
  NeutronTunnelTypes: ''
```

7. 新しいカスタム環境ファイルを作成した場合は、そのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「SR-IOV 用のベアメタルノード定義ファイルの作成」 に進みます。

関連情報

- **Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドで サポートされているカスタムロール

7.4. SR-IOV 用のベアメタルノード定義ファイルの作成

Red Hat OpenStack Platform (RHOSP) ディレクターと定義ファイルを使用して、SR-IOV 環境用にベアメタルノードをプロビジョニングします。ベアメタルノード定義ファイルで、デプロイするベアメタルノードの数量と属性を定義し、これらのノードにオーバークラウドロールを割り当てます。ノードのネットワークレイアウトも定義します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング の指示に従って、**overcloud-baremetal-deploy.yaml** などのベアメタルノード定義ファイルを作成します。
4. ベアメタルノード定義ファイルで、Ansible の Playbook **cli-overcloud-node-kernelargs.yaml** に宣言を追加します。
   Playbook には、ベアメタルノードをプロビジョニングするときに使用するカーネル引数が含まれています。

```
- name: ComputeSriov
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. Playbook の実行時に追加の Ansible 変数を設定する場合は、**extra_vars** プロパティーを使用して設定します。

注記

extra_vars に追加する変数は、「ロール固有のパラメーターと設定のオーバーライドの追加」 でカスタム環境ファイルに追加した SR-IOV Compute ノードのロール固有のパラメーターと同じである必要があります。

例

```
- name: ComputeSriov
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: 'default_hugepagesz=1GB hugepagesz=1G hugepages=100
amd_iommu=on iommu=pt isolcpus=9-63,73-127'
        tuned_isolated_cores: '9-63,73-127'
        tuned_profile: 'cpu-partitioning'
        reboot_wait_timeout: 1800
```

6. 作成したベアメタルノード定義ファイルのパスとファイル名をメモします。このファイルは、後で NIC を設定するときに使用し、ノードをプロビジョニングするときに **overcloud node provision** コマンドの入力ファイルとして使用します。

次のステップ

- 「SR-IOV 用の NIC 設定テンプレートの作成」 に進みます。

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール
- NFV 向けのテスト済み NIC
- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードのプロビジョニング属性

7.5. SR-IOV 用の NIC 設定テンプレートの作成

Red Hat OpenStack Platform (RHOSP) に同梱されているサンプル Jinja2 テンプレートのコピーを変更して、NIC 設定テンプレートを定義します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. サンプルネットワーク設定テンプレートをコピーします。**/usr/share/ansible/roles/tripleo_network_config/templates/** ディレクトリー内の例から NIC 設定 Jinja2 テンプレートをコピーします。NIC 要件に最も近いものを選択してください。必要に応じて変更してください。

4. NIC 設定テンプレート (たとえば、**single_nic_vlans.j2**) に、PF インターフェイスと VF インターフェイスを追加します。SR-IOV VF を作成するには、インターフェイスをスタンドアロン NIC として設定します。

   例

```
...
- type: sriov_pf
  name: enp196s0f0np0
  mtu: 9000
  numvfs: 16
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
...
```

注記

numvfs パラメーターは、ネットワーク設定テンプレートの **NeutronSriovNumVFs** パラメーターに代わるものです。Red Hat では、デプロイ後の **NeutronSriovNumVFs** パラメーターまたは **numvfs** パラメーターの変更をサポートしません。デプロイメント後にいずれかのパラメーターを変更すると、その PF 上に SR-IOV ポートを持つ実行中のインスタンスに中断が発生する可能性があります。この場合、これらのインスタンスをハードリブートして、SR-IOV PCI デバイスを再び利用可能にする必要があります。

5. 「SR-IOV 用のベアメタルノード定義ファイルの作成」 に作成したベアメタルノード定義ファイルにカスタムネットワーク設定テンプレートを追加します。

   例

```
- name: ComputeSriov
  count: 2
  hostname_format: compute-%index%
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. 作成した NIC 設定テンプレートのパスとファイル名をメモします。後で NIC をパーティション分割する場合は、このファイルを使用します。

次のステップ

1. NIC をパーティション分割したい場合は、「NIC パーティションの設定」 に進んでください。
2. それ以外の場合は、次の手順を実行します。
   a. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
   b. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング
   c. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング
   d. 「SR-IOV オーバークラウドのデプロイ」

7.6. NIC パーティションの設定

Red Hat OpenStack Platform (RHOSP)管理ネットワークおよびプロバイダーネットワークに Single Root I/O Virtualization (SR-IOV) Virtual Function (VF) を設定して、各ホストに必要な NIC の数を減らすことができます。1 つの高速 NIC を複数の VF に分割する場合、NIC をコントロールプレーンおよびデータプレーントラフィックの両方に使用することができます。この機能は、Intel Fortville NIC および Mellanox CX-5 NIC で検証されています。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- NIC、そのアプリケーション、VF ゲスト、および OVS が同じ NUMA コンピュートノードに存在することを確認します。
  そうすることで、NUMA 間の操作によるパフォーマンスの低下を防ぐことができます。
- NIC ファームウェアを常に最新の状態に保ってください。
  Yum または **dnf** 更新ではファームウェアの更新が完了しない可能性があります。詳細は、ベンダーのドキュメントを参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「SR-IOV 用の NIC 設定テンプレートの作成」 で作成した NIC 設定テンプレート (例: **single_nic_vlans.j2**) を開きます。

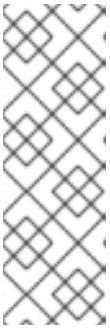
ヒント

このセクションの手順を完了したら、「NIC パーティションの設定例」 を参照してください。

4. インターフェイス種別 **sriov_pf** のエントリーを追加して、ホストが使用できる Physical Function を設定します。

```
- type: sriov_pf
  name: <interface_name>
  use_dhcp: false
  numvfs: <number_of_vfs>
  promisc: <true/false>
```

- **<interface_name>** は、インターフェイスの名前に置き換えます。
- **<number_of_vfs>** は VF の数に置き換えます。
- オプション: **<true/false>** を **true** に置き換えてプロミスキャスモードを設定するか、**false** に置き換えてプロミスキャスモードを無効にします。デフォルト値は **true** です。



注記

numvfs パラメーターは、ネットワーク設定テンプレートの **NeutronSriovNumVFs** パラメーターに代わるものです。Red Hat では、デプロイ後の **NeutronSriovNumVFs** パラメーターまたは **numvfs** パラメーターの変更をサポートしません。デプロイ後にいずれかのパラメーターを変更すると、その Physical Function (PF) 上に SR-IOV ポートを持つ実行中のインスタンスが使用できなくなる可能性があります。この場合、これらのインスタンスをハードリブートして、SR-IOV PCI デバイスを再び利用可能にする必要があります。

5. インターフェイス種別 **sriov_vf** のエントリーを追加して、ホストが使用できる Virtual Function を設定します。

```
- type: <bond_type>
  name: internal_bond
  bonding_options: mode=<bonding_option>
  use_dhcp: false
  members:
  - type: sriov_vf
      device: <pf_device_name>
      vfid: <vf_id>
  - type: sriov_vf
      device:  <pf_device_name>
      vfid: <vf_id>

- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  spoofcheck: false
  device: internal_bond
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    list_concat_unique:
    - get_param: InternalApiInterfaceRoutes
```

- **<bond_type>** を必要なボンディング種別 (例: **linux_bond**) に置き換えます。**ovs_bond** 等の他のボンディング種別のボンディングに VLAN タグを適用することができます。

- **<bonding_option>** を、以下のサポートされるボンディングモードのいずれかに置き換えます。
  - **active-backup**
  - **Balance-slb**

注記

LACP ボンディングはサポートされません。

- **members** セクションで、ボンディングのインターフェイス種別として **sriov_vf** を指定します。

注記

インターフェイス種別として OVS ブリッジを使用している場合は、**sriov_pf** デバイスの **sriov_vf** に OVS ブリッジを 1 つだけ設定することができます。単一の **sriov_pf** デバイス上に複数の OVS ブリッジがあると、VF 間でパケットが重複し、パフォーマンスが低下する可能性があります。

- **<pf_device_name>** を PF デバイスの名前に置き換えます。
- **linux_bond** を使用する場合は、VLAN タグを割り当てる必要があります。VLAN タグを設定する場合は、1 つの **sriov_pf** デバイスに関連付けられた各 VF に一意のタグを設定するようにしてください。同じ VLAN 上の同じ PF の VF を 2 つ指定できません。
- **<vf_id>** を VF の ID に置き換えます。適用可能な VF ID の範囲は、ゼロから VF の最大数から 1 を引いた数値までです。
- スプーフィングチェックを無効にします。
- VF 上の **linux_bond** の **sriov_vf** に VLAN タグを適用します。

6. インスタンスに VF を確保するには、環境ファイルに **NovaPCIPassthrough** パラメーターを追加します。

例

```
NovaPCIPassthrough:
 - address: "0000:19:0e.3"
   trusted: "true"
   physical_network: "sriov1"
 - address: "0000:19:0e.0"
   trusted: "true"
   physical_network: "sriov2"
```

RHOSP director はホストの VF を把握し、インスタンスで利用可能な VF の PCI アドレスを派生します。

7. NIC の分割が必要なすべてのノードで **IOMMU** を有効にします。

例

たとえば、Compute ノードに NIC 分割を設定する場合は、そのロールの **KernelArgs** パラメーターを使用して IOMMU を有効にします。

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "intel_iommu=on iommu=pt"
```

注記

KernelArgs パラメーターをロールの設定に初めて追加すると、オーバークラウドノードが自動的に再起動されます。必要に応じて、ノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイ後にノードの再起動を手動で実行できます。

8. この NIC 設定テンプレート (例: **single_nic_vlans.j2**) を、「SR-IOV 用のベアメタルノード定義ファイルの作成」 で作成したベアメタルノード定義ファイルに追加してください。

次のステップ

1. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
2. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング
3. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング
4. 「SR-IOV オーバークラウドのデプロイ」

関連情報

- 「NIC パーティションの設定例」

7.7. NIC パーティションの設定例

Red Hat OpenStack Platform SR-IOV 環境で NIC をパーティション分割する場合は、これらの設定例を参照してください。

VF 上の Linux ボンディング

以下の例では、VF 上で Linux ボンディングを設定して、**spoofcheck** を無効にし、VLAN タグを **sriov_vf** に適用します。

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  members:
    - type: sriov_vf
      device: eno2
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
```

```
    - type: sriov_vf
      device: eno3
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
  addresses:
    - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    list_concat_unique:
    - get_param: InternalApiInterfaceRoutes
```

VF 上の OVS ブリッジ

以下の例では、VF に OVS ブリッジを設定します。

```
- type: ovs_bridge
  name: br-bond
  use_dhcp: true
  members:
    - type: vlan
      vlan_id:
      get_param: TenantNetworkVlanID
  addresses:
  - ip_netmask:
    get_param: TenantIpSubnet
  routes:
    list_concat_unique:
      - get_param: ControlPlaneStaticRoutes
  - type: ovs_bond
    name: bond_vf
    ovs_options: "bond_mode=active-backup"
    members:
      - type: sriov_vf
        device: p2p1
        vfid: 2
      - type: sriov_vf
        device: p2p2
        vfid: 2
```

VF 上の OVS ユーザーブリッジ

以下の例では、VF で OVS ユーザーブリッジを設定し、VLAN タグを **ovs_user_bridge** に適用します。

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_:
            get_param: TenantNetworkVlanID
```

```
addresses:
  - ip_netmask:
  list_concat_unique:
    - get_param: TenantInterfaceRoutes
members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    ovs_extra:
      - set port dpdkbond0 bond_mode=balance-slb
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
          - type: sriov_vf
            device: eno2
            vfid: 3
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: sriov_vf
            device: eno3
            vfid: 3
```

関連情報

- 「NIC パーティションの設定」

7.8. SR-IOV オーバークラウドのデプロイ

SR-IOV 環境で Red Hat OpenStack Platform (RHOSP) オーバークラウドを設定する最後の手順は、**openstack overcloud deploy** コマンドを実行することです。このコマンドに、作成したさまざまなオーバークラウドテンプレートと環境ファイルをすべて入力します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- このセクションの前の手順にリストされているすべてのステップを実行し、**overcloud deploy** コマンドの入力として使用するさまざまな heat テンプレートおよび環境ファイルをすべてアセンブルしました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. **openstack overcloud deploy** コマンドを実行します。
openstack overcloud deploy コマンドへの入力を特定の順序でリストすることが重要です。一般的なルールは、デフォルトの heat テンプレートファイルを最初に指定し、次にカスタム環境ファイルと、デフォルトプロパティーのオーバーライドなどのカスタム設定を含むカスタム

テンプレートを指定することです。

次の順序で、**openstack overcloud deploy** コマンドに入力を追加します。

a. オーバークラウド上の SR-IOV ネットワークの仕様が含まれるカスタムネットワーク定義ファイル (例: **network-data.yaml**)。
詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ネットワーク定義ファイル設定のオプション を参照してください。

b. RHOSP director が OVS ハードウェアオフロード環境をデプロイするために使用する **Controller** および **ComputeOvsHwOffload** ロールを含むロールファイル。
例: **roles_data_compute_sriov.yaml**

詳細は、「SR-IOV のロールとイメージファイルの生成」 を参照してください。

c. オーバークラウドネットワークのプロビジョニングからの出力ファイル。
例: **overcloud-networks-deployed.yaml**

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング を参照してください。

d. オーバークラウド仮想 IP のプロビジョニングからの出力ファイル。
例: **overcloud-vip-deployed.yaml**

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング を参照してください。

e. ベアメタルノードのプロビジョニングからの出力ファイル。
たとえば、**overcloud-baremetal-deployed.yaml** です。

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング を参照してください。

f. コンテナーイメージを取得する場所と保存方法を決定するためにディレクターが使用するイメージファイル。
例: **overcloud_images.yaml**

詳細は、「SR-IOV のロールとイメージファイルの生成」 を参照してください。

g. 環境が使用する Networking サービス (neutron) メカニズムドライバーとルータースキームの環境ファイル:

- ML2/OVN
  - 分散仮想ルーティング (DVR): **neutron-ovn-dvr-ha.yaml**
  - 集中型仮想ルーティング: **neutron-ovn-ha.yaml**
- ML2/OVS
  - 分散仮想ルーティング (DVR): **neutron-ovs-dvr.yaml**
  - 集中型仮想ルーティング: **neutron-ovs.yaml**

h. メカニズムドライバーに応じた SR-IOV の環境ファイル:

- ML2/OVN

- **neutron-ovn-sriov.yaml**

- ML2/OVS
  - **neutron-sriov.yaml**

注記

OVS-DPDK 環境もあり、OVS-DPDK インスタンスと SR-IOV インスタンスを同じノードに配置する場合は、デプロイメントスクリプトに次の環境ファイルを含めます。

- ML2/OVN
  neutron-ovn-dpdk.yaml
- ML2/OVS
  neutron-ovs-dpdk.yaml

i. 以下の設定を含む 1 つ以上のカスタム環境ファイル:

- SR-IOV ノード用の PCI パススルーデバイス。
- SR-IOV ノードのロール固有のパラメーター。
- SR-IOV 環境のデフォルト設定値をオーバーライドします。
  例: **neutron-ovs.yaml**

  詳細は以下を参照してください。
- 「SR-IOV 用の PCI パススルーデバイスの設定」。
- 「ロール固有のパラメーターと設定のオーバーライドの追加」。

例

サンプルの **openstack overcloud deploy** コマンドからのこの抜粋は、DVR を使用する SR-IOV、ML2/OVN 環境のコマンド入力の適切な順序を示しています。

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_sriov.yaml \
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-sriov.yaml \
-e /home/stack/templates/sriov-overrides.yaml
```

4. **openstack overcloud deploy** コマンドを実行します。

オーバークラウドの作成が完了すると、RHOSP director は、オーバークラウドへのアクセスに役立つ詳細を提供します。

検証

1. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのデプロイメントの検証 のステップを実行します。
2. NIC が適切にパーティション分割されていることを確認するには、次の手順を実行します。
   a. **tripleo-admin** としてオーバークラウドの Compute ノードにログインし、VF の数を確認します。

      例

      この例では、**p4p1** と **p4p2** の両方の VF の数は **10** です。

      ```
      $ sudo cat /sys/class/net/p4p1/device/sriov_numvfs

      10

      $ sudo cat /sys/class/net/p4p2/device/sriov_numvfs

      10
      ```

   b. OVS 接続を表示します。

      ```
      $ sudo ovs-vsctl show
      ```

      出力例

      以下のような出力が表示されるはずです。

      ```
      b6567fa8-c9ec-4247-9a08-cbf34f04c85f
          Manager "ptcp:6640:127.0.0.1"
              is_connected: true
          Bridge br-sriov2
              Controller "tcp:127.0.0.1:6633"
                  is_connected: true
              fail_mode: secure
              datapath_type: netdev
              Port phy-br-sriov2
                  Interface phy-br-sriov2
                      type: patch
                      options: {peer=int-br-sriov2}
              Port br-sriov2
                  Interface br-sriov2
                      type: internal
          Bridge br-sriov1
              Controller "tcp:127.0.0.1:6633"
                  is_connected: true
              fail_mode: secure
              datapath_type: netdev
              Port phy-br-sriov1
                  Interface phy-br-sriov1
                      type: patch
      ```

```
            options: {peer=int-br-sriov1}
        Port br-sriov1
            Interface br-sriov1
                type: internal
    Bridge br-ex
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port br-ex
            Interface br-ex
                type: internal
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
    Bridge br-tenant
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port br-tenant
            tag: 305
            Interface br-tenant
                type: internal
        Port phy-br-tenant
            Interface phy-br-tenant
                type: patch
                options: {peer=int-br-tenant}
        Port dpdkbond0
            Interface dpdk0
                type: dpdk
                options: {dpdk-devargs="0000:18:0e.0"}
            Interface dpdk1
                type: dpdk
                options: {dpdk-devargs="0000:18:0a.0"}
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port vxlan-98140025
            Interface vxlan-98140025
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.37"}
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port vxlan-98140015
            Interface vxlan-98140015
                type: vxlan
```

```
            options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.21"}
        Port vxlan-9814009f
            Interface vxlan-9814009f
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.159"}
        Port vxlan-981400cc
            Interface vxlan-981400cc
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.204"}
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port int-br-tenant
            Interface int-br-tenant
                type: patch
                options: {peer=phy-br-tenant}
        Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
        Port int-br-sriov1
            Interface int-br-sriov1
                type: patch
                options: {peer=phy-br-sriov1}
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
        Port int-br-sriov2
            Interface int-br-sriov2
                type: patch
                options: {peer=phy-br-sriov2}
        Port vhu4142a221-93
            tag: 1
            Interface vhu4142a221-93
                type: dpdkvhostuserclient
                options: {vhost-server-path="/var/lib/vhost_sockets/vhu4142a221-93"}
    ovs_version: "2.13.2"
```

c. SR-IOV コンピュートノードに **tripleo-admin** としてログインし、Linux ボンドを確認します。

```
$ cat /proc/net/bonding/<bond_name>
```

出力例

以下のような出力が表示されるはずです。

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eno3v1
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: eno3v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 4e:77:94:bd:38:d2
Slave queue ID: 0

Slave Interface: eno4v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 4a:74:52:a7:aa:7c
Slave queue ID: 0
```

3. OVS ボンドをリストします。

```
$ sudo ovs-appctl bond/show
```

出力例

以下のような出力が表示されるはずです。

```
---- dpdkbond0 ----
bond_mode: balance-slb
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
next rebalance: 9491 ms
lacp_status: off
lacp_fallback_ab: false
active slave mac: ce:ee:c7:58:8e:b2(dpdk1)

slave dpdk0: enabled
  may_enable: true

slave dpdk1: enabled
  active slave
  may_enable: true
```

4. **NovaPCIPassthrough** を使用して VF をインスタンスに渡した場合は、SR-IOV インスタンスをデプロイしてテストを行います。

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドの作成
- **コマンドラインインターフェイスリファレンス** の overcloud deploy
- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのインスタンスの作成」

7.9. SR-IOV または OVS TC-FLOWER ハードウェアオフロード環境でのホストアグリゲートの作成

Red Hat OpenStack Platform (RHOSP) SR-IOV または OVS TC-flower ハードウェアオフロード環境でパフォーマンスを向上させるには、CPU ピンニングと huge page を備えたゲストをデプロイします。アグリゲートメタデータをフレーバーメタデータに一致させることで、ホストのサブセット上にハイパフォーマンスインスタンスをスケジュールすることができます。

前提条件

- SR-IOV または OVS ハードウェアオフロード環境用に設定された RHOSP オーバークラウド。
- RHOSP オーバークラウドは **AggregateInstanceExtraSpecsFilter** 用に設定されている必要があります。
  詳細は、「SR-IOV 用の PCI パススルーデバイスの設定」 を参照してください。

手順

1. 集約グループを作成し、関連するホストを追加します。
   定義するフレーバーメタデータに一致するメタデータを定義します (例: **sriov=true**)。

```
$ openstack aggregate create sriov_group
$ openstack aggregate add host sriov_group compute-sriov-0.localdomain
$ openstack aggregate set --property sriov=true sriov_group
```

2. フレーバーを作成します。

```
$ openstack flavor create <flavor> --ram <size_mb> --disk <size_gb> \
--vcpus <number>
```

3. 追加のフレーバー属性を設定します。
   定義したメタデータ (**sriov=true**) と SR-IOV アグリゲートで定義したメタデータが一致している点に注意してください。

```
$ openstack flavor set --property sriov=true \
--property hw:cpu_policy=dedicated \
--property hw:mem_page_size=1GB <flavor>
```

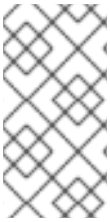
関連情報

- **コマンドラインインターフェイスリファレンス** の aggregate
- **コマンドラインインターフェイスリファレンス** の flavor

7.10. SR-IOV または OVS TC-FLOWER ハードウェアオフロード環境でのインスタンスの作成

Red Hat OpenStack Platform (RHOSP) SR-IOV または OVS TC-flower ハードウェアオフロード環境でインスタンスを作成するには、いくつかのコマンドを使用します。

ホストアグリゲートを使用して、ハイパフォーマンスコンピュートホストを分離します。詳細は、「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのホストアグリゲートの作成」を参照してください。



注記

CPU ピニングを設定したインスタンスと設定していないインスタンスを、同じ Compute ノードに配置することができます。詳細は、**インスタンス作成のためのコンピューティングサービスの設定** ガイドの Compute ノードでの CPU ピンニングの設定 を参照してください。

前提条件

- SR-IOV または OVS ハードウェアオフロード環境用に設定された RHOSP オーバークラウド。

手順

1. フレーバーを作成します。

```
$ openstack flavor create <flavor_name> --ram <size_mb> \
--disk <size_gb> --vcpus <number>
```

ヒント

フレーバーに追加スペック **hw:pci_numa_affinity_policy** を追加して、PCI パススルーデバイスおよび SR-IOV インターフェイスの NUMA アフィニティーポリシーを指定することができます。詳細は、**インスタンス作成のための Compute サービスの設定** の フレーバーメタデータ を参照してください。

2. ネットワークとサブネットを作成します。

```
$ openstack network create <network_name> \
--provider-physical-network tenant \
--provider-network-type vlan --provider-segment <vlan_id>

$ openstack subnet create <name> --network <network_name> \
--subnet-range <ip_address_cidr> --dhcp
```

3. Virtual Function (VF) ポートまたは Physical Fundtion (PF) ポートを作成します。

- VF ポート:

```
$ openstack port create --network <network_name> \
--vnic-type direct <port_name>
```

- 単一インスタンス専用の PF ポート:

この PF ポートは Networking サービス (neutron) ポートですが、Networking サービスによって制御されておらず、インスタンスにパススルーされる PCI デバイスであるため、ネットワークアダプターとして表示されません。

```
$ openstack port create --network <network_name> \
--vnic-type direct-physical <port_name>
```

4. インスタンスを作成します。

```
$ openstack server create --flavor <flavor> --image <image_name> \
--nic port-id=<id> <instance_name>
```

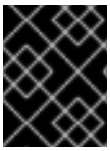
関連情報

- **コマンドラインインターフェイスリファレンス** の flavor create
- **コマンドラインインターフェイスリファレンス** の network create
- **コマンドラインインターフェイスリファレンス** の subnet create
- **コマンドラインインターフェイスリファレンス** の port create
- **コマンドラインインターフェイスリファレンス** の server create

第8章 OVS TC-FLOWER ハードウェアオフロードの設定

Red Hat OpenStack Platform (RHOSP) ネットワーク機能仮想化 (NFV) デプロイメントでは、Open vSwitch (OVS) TC-flower ハードウェアオフロードを使用して、より高いパフォーマンスを実現できます。ハードウェアオフロードは、ネットワークタスクを CPU からネットワークインターフェイスコントローラー (NIC) 上の専用プロセッサーに転送します。これらの特殊なハードウェアリソースは追加の計算能力を提供し、CPU がより価値のある計算タスクを実行できるようにします。

OVS ハードウェアオフロード用に RHOSP を設定する方法は、SR-IOV 用に RHOSP を設定する方法と似ています。



重要

このセクションには、トポロジーと機能要件に合わせて変更する必要がある例が含まれています。詳細は、NFV のハードウェア要件 を参照してください。

前提条件

- RHOSP アンダークラウド。
  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。詳細は、**director を使用した Red Hat OpenStack Platform のインストールおよび管理** を参照してください。

  注記

  RHOSP director は、director テンプレートとカスタム環境ファイルで指定したキーと値のペアを通じて、OVS ハードウェアオフロード設定ファイルを変更します。OVS ハードウェアオフロード設定ファイルを直接変更しないでください。

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- NIC、そのアプリケーション、VF ゲスト、および OVS が同じ NUMA コンピュートノード上に存在していることを確認します。
  そうすることで、NUMA 間の操作によるパフォーマンスの低下を防ぐことができます。
- NIC を含むホスト上の sudo へのアクセス。
- NIC ファームウェアを常に最新の状態に保ってください。
  Yum または **dnf** 更新ではファームウェアの更新が完了しない可能性があります。詳細は、ベンダーのドキュメントを参照してください。
- OpenFlow フローをハードウェアにオフロードするには、接続追跡 (conntrack) モジュールの **switchdev** ポートでセキュリティーグループとポートセキュリティーを有効にします。

手順

RHOSP director を使用して、OVS ハードウェアオフロード環境で RHOSP をインストールおよび設定します。大まかな手順は次のとおりです。

1. **director を使用した Red Hat OpenStack Platform のインストールと管理**の オーバークラウドネットワークの設定 の手順に従って、ネットワーク設定ファイル **network_data.yaml** を作成し、オーバークラウドの物理ネットワークを設定します。
2. ロールとイメージファイルを生成します。

3. OVS ハードウェアオフロード用に PCI パススルーデバイスを設定します。
4. ロール固有のパラメーターとその他の設定オーバーライドを追加します。
5. ベアメタルノード定義ファイルを作成します。
6. OVS ハードウェアオフロード用の NIC 設定テンプレートを作成します。
7. オーバークラウドネットワークと仮想 IP をプロビジョニングします。
   詳細は以下を参照してください。
   - **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
   - **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング
8. オーバークラウドベアメタルノードをプロビジョニングします。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング を参照してください。
9. OVS ハードウェアオフロードオーバークラウドをデプロイします。

関連情報

- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのホストアグリゲートの作成」
- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのインスタンスの作成」
- 「OVS TC-flower ハードウェアオフロードのトラブルシューティング」
- 「TC-flower ハードウェアオフロードフローのデバッグ」

8.1. OVS TC-FLOWER ハードウェアオフロード用のロールとイメージファイルの生成

Red Hat OpenStack Platform (RHOSP) ディレクターは、ロールを使用してノードにサービスを割り当てます。OVS TC-flower ハードウェアオフロード環境で RHOSP を設定する場合は、RHOSP インストールで提供されるデフォルトのロールである **Compute** に基づいて新しいロールを作成します。

アンダークラウドのインストールには、コンテナーイメージの取得先およびその保存方法を定義するための環境ファイルが必要です。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **Compute** ロールをベースとする OVS ハードウェアオフロード用のオーバークラウドロールを生成します。

   例

   この例では、コンピュートロールに基づいて、ComputeOvsHwOffload というロールが作成されます。コマンドによって生成されるロールファイルの名前は **roles_data_compute_ovshwol.yaml** です。

   ```
   $ openstack overcloud roles generate -o \
   roles_data_compute_ovshwol.yaml Controller Compute:ComputeOvsHwOffload
   ```

   注記

   RHOSP 環境に OVS-DPDK、SR-IOV、および OVS TC-flower ハードウェアオフロードテクノロジーが混在している場合は、すべてのロールを含めるために、**roles_data.yaml** などのロールデータファイルを 1 つだけ生成します。

   ```
   $ openstack overcloud roles generate -o /home/stack/templates/\
   roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
   Compute:ComputeOvsHwOffload
   ```

4. (オプション): **ComputeOvsHwOffload** ロール向けの **HostnameFormatDefault: '%stackname%-compute-%index%'** の名前を変更します。

5. イメージファイルを生成するには、**openstack tripleo container image prepare** コマンドを実行します。以下の入力が必要です。

   - 前の手順で生成したロールデータファイル (例: **roles_data_compute_ovshwol.yaml**)。
   - Networking サービスメカニズムドライバーに適した SR-IOV 環境ファイル:
     - ML2/OVN 環境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml**
     - ML2/OVS 環境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml**

       例

       この例では、**overcloud_images.yaml** ファイルが ML2/OVS 環境用に生成されています。

       ```
       $ sudo openstack tripleo container image prepare \
         --roles-file ~/templates/roles_data_compute_ovshwol.yaml \
         -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
         -e ~/containers-prepare-parameter.yaml \
         --output-env-file=/home/stack/templates/overcloud_images.yaml
       ```

6. 作成したロールデータファイルとイメージファイルのパスとファイル名をメモします。これらのファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS TC-flower ハードウェアオフロード用の PCI パススルーデバイスの設定」 に進みます。

関連情報

- 詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール を参照してください。
- **director を使用した Red Hat OpenStack Platform のインストールと管理**で コンテナーイメージを準備 します。

8.2. OVS TC-FLOWER ハードウェアオフロード用の PCI パススルーデバイスの設定

OVS TC-flower ハードウェアオフロード環境用に Red Hat OpenStack Platform をデプロイする場合は、カスタム環境ファイルでコンピュートノードの PCI パススルーデバイスを設定する必要があります。

前提条件

- PCI カードが搭載されている 1 台以上の物理サーバーへのアクセス。
- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. PCI カードが搭載されている物理サーバー上で、次のいずれかのコマンドを使用します。
   - オーバークラウドがデプロイされている場合:

     ```
     $ lspci -nn -s  <pci_device_address>
     ```

     出力例

     ```
     3b:00.0 Ethernet controller [0200]: Intel Corporation Ethernet
     Controller X710 for 10GbE SFP+ [<vendor_id>: <product_id>] (rev 02)
     ```

   - オーバークラウドがデプロイされていない場合:

     ```
     $ openstack baremetal introspection data save <baremetal_node_name> | jq
     '.inventory.interfaces[] | .name, .vendor, .product'
     ```

2. ComputeOvsHwOffload ノード上の PCI パススルーデバイスのベンダー ID と製品 ID をメモします。これらの ID は後の手順で必要になります。
3. アンダークラウドに **stack** ユーザーとしてログインします。
4. **stackrc** ファイルを取得します。

   ```
   $ source ~/stackrc
   ```

5. カスタム環境 YAML ファイル (例: **ovshwol-overrides.yaml**) を作成します。次の内容をファイルに追加して、コンピュートノードの PCI パススルーデバイスを設定します。

```
parameter_defaults:
  NeutronOVSFirewallDriver: iptables_hybrid
  ComputeOvsHwOffloadParameters:
    IsolCpusList: 2-9,21-29,11-19,31-39
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=128 intel_iommu=on
iommu=pt"
    OvsHwOffload: true
    TunedProfileName: "cpu-partitioning"
    NeutronBridgeMappings:
      - tenant:br-tenant
    NovaPCIPassthrough:
      - vendor_id: <vendor-id>
        product_id: <product-id>
        address: <address>
        physical_network: "tenant"
      - vendor_id: <vendor-id>
        product_id: <product-id>
        address: <address>
        physical_network: "null"
    NovaReservedHostMemory: 4096
    NovaComputeCpuDedicatedSet: 1-9,21-29,11-19,31-39
    ...
```

注記

Mellanox スマート NIC を使用している場合は、**ComputeOvsHwOffloadParameters** パラメーターの下に **DerivePciWhitelistEnabled: true** を追加します。OVS ハードウェアオフロードを使用する場合、コンピュートサービス (nova) スケジューラーは、インスタンス生成の SR-IOV パススルーと同様に動作します。

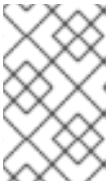
- **<vendor_id>** を PCI デバイスのベンダー ID に置き換えます。
- **<product_id>** を PCI デバイスの製品 ID に置き換えます。
- **<NIC_address>** を PCI デバイスのアドレスに置き換えます。
- **<physical_network>** を、PCI デバイスが配置されている物理ネットワークの名前に置き換えます。
- VLAN の場合には、**physical_network** パラメーターをデプロイメント後に neutron で作成するネットワークの名前に設定します。この値は、**NeutronBridgeMappings** にも設定する必要があります。
- VXLAN の場合には、**physical_network** パラメーターを **null** に設定します。

注記

NIC のデバイス名は変更される可能性があるため、PCI パススルーを設定する場合は **devname** パラメーターを使用しないでください。PF で Networking サービス (neutron) ポートを作成するには、**NovaPCIPassthrough** に **vendor_id**、**product_id**、および PCI デバイスアドレスを指定し、**--vnic-type direct-physical** オプションでポートを作成します。Virtual Function (VF) に Networking サービスのポートを作成するには、**NovaPCIPassthrough**で**vendor_id**と**product_id**を指定し、**--vnic-type direct**オプションを使用してポートを作成します。**vendor_id**および**product_id**パラメーターの値は、Physical Function (PF) コンテキストと VF コンテキストの間で異なる場合があります。

6. カスタム環境ファイルで、**PciPassthroughFilter** と **NUMATopologyFilter** が **NovaSchedulerEnabledFilters** パラメーターのフィルターリストに含まれていることを確認します。コンピュートサービス (nova) は、このパラメーターを使用してノードをフィルタリングします。

```
parameter_defaults:
  ...
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
    - AggregateInstanceExtraSpecsFilter
```



注記

オプション: Mellanox ConnectX5 NIC を使用する RHOSP 17.1 での OVS ハードウェアオフロードに関する問題のトラブルシューティングと設定方法の詳細は、ハードウェアオフロードのトラブルシューティング を参照してください。

7. 作成したカスタム環境ファイルのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS TC-flower ハードウェアオフロード用のロール固有のパラメーターと設定オーバーライドの追加」 に進みます。

関連情報

- **インスタンス作成のためのコンピュートサービスの設定** における **NovaPCIPassthrough** の設定に関するガイドライン

8.3. OVS TC-FLOWER ハードウェアオフロード用のロール固有のパラメーターと設定オーバーライドの追加

ComputeOvsHwOffload ノードのロール固有のパラメーターを追加し、Red Hat OpenStack Platform (RHOSP) director が OVS TC-flower ハードウェアオフロード環境をデプロイするときに使用するカスタム環境 YAML ファイル内のデフォルトの設定値をオーバーライドできます。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「OVS TC-flower ハードウェアオフロード用の PCI パススルーデバイスの設定」 に作成したカスタム環境 YAML ファイルを開くか、新しいものを作成します。
4. カスタム環境ファイルに、ComputeOvsHwOffload ノードのロール固有のパラメーターを追加します。

例

```
  ComputeOvsHwOffloadParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
    TunedProfileName: "cpu-partitioning"
```

5. ロール固有のパラメーターセクションに **OvsHwOffload** パラメーターを追加し、値を **true** に設定してます。

```
  ComputeOvsHwOffloadParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
    TunedProfileName: "cpu-partitioning"
    OvsHwOffload: true
  ...
```

6. RHOSP director が OVS ハードウェアオフロードを設定するのに使用する設定のデフォルトを確認します。これらのデフォルトはファイルで提供されており、メカニズムドライバーによって異なります。

- ML2/OVN
  /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-sriov.yaml

- ML2/OVS
  /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-sriov.yaml

7. 設定のデフォルトをオーバーライドする必要がある場合は、オーバーライドをカスタム環境ファイルに追加します。
   たとえば、このカスタム環境ファイルでは、Nova PCI ホワイトリスト値を追加したり、ネットワークタイプを設定したりできます。

   例

   この例では、ネットワークサービス (neutron) ネットワークタイプが VLAN に設定され、テナントの範囲が追加されます。

   ```
   parameter_defaults:
     NeutronNetworkType: vlan
     NeutronNetworkVLANRanges:
       - tenant:22:22
       - tenant:25:25
     NeutronTunnelTypes: ''
   ```

8. 新しいカスタム環境ファイルを作成した場合は、そのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS TC-flower ハードウェアオフロード用のベアメタルノード定義ファイルの作成」 に進みます。

関連情報

- **Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドで サポートされているカスタムロール

8.4. OVS TC-FLOWER ハードウェアオフロード用のベアメタルノード定義ファイルの作成

Red Hat OpenStack Platform (RHOSP) director と定義ファイルを使用して、OVS TC-flower ハードウェアオフロード環境用にベアメタルノードをプロビジョニングします。ベアメタルノード定義ファイルで、デプロイするベアメタルノードの数量と属性を定義し、これらのノードにオーバークラウドロールを割り当てます。ノードのネットワークレイアウトも定義します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

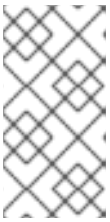
   ```
   $ source ~/stackrc
   ```

3. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング の指示に従って、**overcloud-baremetal-deploy.yaml** などのベアメタルノード定義ファイルを作成します。

4. ベアメタルノード定義ファイルで、Ansible の Playbook **cli-overcloud-node-kernelargs.yaml** に宣言を追加します。
Playbook には、ベアメタルノードをプロビジョニングするときに使用するカーネル引数が含まれています。

```
- name: ComputeOvsHwOffload
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. Playbook の実行時に追加の Ansible 変数を設定する場合は、**extra_vars** プロパティーを使用して設定します。



注記

extra_vars に追加する変数は、「OVS TC-flower ハードウェアオフロード用のロール固有のパラメーターと設定オーバーライドの追加」 でカスタム環境ファイルに追加した SR-IOV Compute ノードのロール固有のパラメーターと同じである必要があります。

例

```
- name: ComputeOvsHwOffload
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: 'default_hugepagesz=1GB hugepagesz=1G hugepages=100
amd_iommu=on iommu=pt isolcpus=9-63,73-127'
        tuned_isolated_cores: '9-63,73-127'
        tuned_profile: 'cpu-partitioning'
        reboot_wait_timeout: 1800
```

6. 作成したベアメタルノード定義ファイルのパスとファイル名をメモします。このファイルは、後で NIC を設定するときに使用し、ノードをプロビジョニングするときに **overcloud node provision** コマンドの入力ファイルとして使用します。

次のステップ

- 「OVS TC-flower ハードウェアオフロード用の NIC 設定テンプレートの作成」 に進みます。

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール
- NFV 向けのテスト済み NIC

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードのプロビジョニング属性

8.5. OVS TC-FLOWER ハードウェアオフロード用の NIC 設定テンプレートの作成

Red Hat OpenStack Platform (RHOSP) に同梱されているサンプル Jinja2 テンプレートのコピーを変更して、OVS TC-flower ハードウェアオフロード環境の NIC 設定テンプレートを定義します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- NIC、そのアプリケーション、VF ゲスト、および OVS が同じ NUMA コンピュートノード上に存在していることを確認します。
  そうすることで、NUMA 間の操作によるパフォーマンスの低下を防ぐことができます。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. サンプルネットワーク設定テンプレートをコピーします。
   /usr/share/ansible/roles/tripleo_network_config/templates/ ディレクトリー内の例から NIC 設定 Jinja2 テンプレートをコピーします。NIC 要件に最も近いものを選択してください。必要に応じて変更してください。
4. NIC 設定テンプレート (たとえば、**single_nic_vlans.j2**) に、PF インターフェイスと VF インターフェイスを追加します。VF を作成するには、インターフェイスをスタンドアロン NIC として設定します。

   例

```
...
- type: sriov_pf
  name: enp196s0f0np0
  mtu: 9000
  numvfs: 16
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
  link_mode: switchdev
...
```

注記

numvfs パラメーターは、ネットワーク設定テンプレートの **NeutronSriovNumVFs** パラメーターに代わるものです。Red Hat では、デプロイ後の **NeutronSriovNumVFs** パラメーターまたは **numvfs** パラメーターの変更をサポートしません。デプロイメント後にいずれかのパラメーターを変更すると、その PF 上に SR-IOV ポートを持つ実行中のインスタンスに中断が発生する可能性があります。この場合、これらのインスタンスをハードリブートして、SR-IOV PCI デバイスを再び利用可能にする必要があります。

5. 「OVS TC-flower ハードウェアオフロード用のベアメタルノード定義ファイルの作成」 に作成したベアメタルノード定義ファイルにカスタムネットワーク設定テンプレートを追加します。

例

```
- name: ComputeOvsHwOffload
  count: 2
  hostname_format: compute-%index%
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. **compute-sriov.yaml** 設定ファイルで、ハードウェアオフロードに使用するネットワークインターフェイスを 1 つまたは複数設定します。

```
  - type: ovs_bridge
    name: br-tenant
    mtu: 9000
    members:
    - type: sriov_pf
      name: p7p1
      numvfs: 5
      mtu: 9000
      primary: true
      promisc: true
      use_dhcp: false
      link_mode: switchdev
```

注記

- OVS ハードウェアオフロードを設定するときは、**NeutronSriovNumVFs** パラメーターを使用しないでください。Virtual Function の数は、**os-net-config** で使用されるネットワーク設定ファイルの **numvfs** パラメーターを使用して指定します。Red Hat では、更新または再デプロイ時の **numvfs** 設定の変更をサポートしません。

- Mellanox ネットワークインターフェイスの nic-config インターフェイス種別を **ovs-vlan** に設定しないでください。ドライバーの制約により、VXLAN 等のトンネルエンドポイントがトラフィックを渡さなくなるためです。

7. 作成した NIC 設定テンプレートのパスとファイル名をメモします。後で NIC をパーティション分割する場合は、このファイルを使用します。

次のステップ

1. オーバークラウドネットワークをプロビジョニングします。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング を参照してください。

2. オーバークラウド仮想 IP をプロビジョニングします。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング を参照してください。

3. ベアメタルノードをプロビジョニングする。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング を参照してください。

4. オーバークラウドをデプロイします。
   詳細は、「OVS TC-flower ハードウェアオフロードオーバークラウドのデプロイ」 を参照してください。

8.6. OVS TC-FLOWER ハードウェアオフロードオーバークラウドのデプロイ

OVS TC-flower ハードウェアオフロード環境に Red Hat OpenStack Platform (RHOSP) オーバークラウドをデプロイする最後の手順は、**openstack overcloud deploy** コマンドを実行することです。このコマンドに、作成したさまざまなオーバークラウドテンプレートと環境ファイルをすべて入力します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

- NIC を含むホスト上の sudo へのアクセス。

- このセクションの前の手順にリストされているすべてのステップを実行し、**overcloud deploy** コマンドの入力として使用するさまざまな heat テンプレートおよび環境ファイルをすべてアセンブルしました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。

2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. **openstack overcloud deploy** コマンドを実行します。
openstack overcloud deploy コマンドへの入力を特定の順序でリストすることが重要です。一般的なルールは、デフォルトの heat テンプレートファイルを最初に指定し、次にカスタム環境ファイルと、デフォルトプロパティーのオーバーライドなどのカスタム設定を含むカスタムテンプレートを指定することです。

   次の順序で、**openstack overcloud deploy** コマンドに入力を追加します。

   a. オーバークラウド上の SR-IOV ネットワークの仕様が含まれるカスタムネットワーク定義ファイル (例: **network-data.yaml**)。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ネットワーク定義ファイル設定のオプション を参照してください。

   b. RHOSP director が OVS ハードウェアオフロード環境をデプロイするために使用する **Controller** および **ComputeOvsHwOffload** ロールを含むロールファイル。
   例: **roles_data_compute_ovshwol.yaml**

   詳細は、「OVS TC-flower ハードウェアオフロード用のロールとイメージファイルの生成」 を参照してください。

   c. オーバークラウドネットワークのプロビジョニングからの出力ファイル。
   例: **overcloud-networks-deployed.yaml**

   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング を参照してください。

   d. オーバークラウド仮想 IP のプロビジョニングからの出力ファイル。
   例: **overcloud-vip-deployed.yaml**

   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング を参照してください。

   e. ベアメタルノードのプロビジョニングからの出力ファイル。
   たとえば、**overcloud-baremetal-deployed.yaml** です。

   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング を参照してください。

   f. コンテナーイメージを取得する場所と保存方法を決定するためにディレクターが使用するイメージファイル。
   例: **overcloud_images.yaml**

   詳細は、「OVS TC-flower ハードウェアオフロード用のロールとイメージファイルの生成」 を参照してください。

   g. 環境が使用する Networking サービス (neutron) メカニズムドライバーとルータースキームの環境ファイル:

      - ML2/OVN
        - 分散仮想ルーティング (DVR): **neutron-ovn-dvr-ha.yaml**

- 集中型仮想ルーティング: **neutron-ovn-ha.yaml**

- ML2/OVS
  - 分散仮想ルーティング (DVR): **neutron-ovs-dvr.yaml**
  - 集中型仮想ルーティング: **neutron-ovs.yaml**

h. メカニズムドライバーに応じた SR-IOV の環境ファイル:

- ML2/OVN
  - **neutron-ovn-sriov.yaml**
- ML2/OVS
  - **neutron-sriov.yaml**

注記

OVS-DPDK 環境もあり、OVS-DPDK インスタンスと SR-IOV インスタンスを同じノードに配置する場合は、デプロイメントスクリプトに次の環境ファイルを含めます。

- ML2/OVN
  neutron-ovn-dpdk.yaml
- ML2/OVS
  neutron-ovs-dpdk.yaml

i. 以下の設定を含む 1 つ以上のカスタム環境ファイル:

- ComputeOvsHwOffload ノード用の PCI パススルーデバイス。
- ComputeOvsHwOffload ノードのロール固有のパラメーター。
- OVS ハードウェアオフロード環境のデフォルト設定値をオーバーライドします。
  例: **ovshwol-overrides.yaml**

  詳細は以下を参照してください。

- 「OVS TC-flower ハードウェアオフロード用の PCI パススルーデバイスの設定」。
- 「OVS TC-flower ハードウェアオフロード用のロール固有のパラメーターと設定オーバーライドの追加」。

例

サンプルの **openstack overcloud deploy** コマンドからのこの抜粋は、DVR を使用する SR-IOV、ML2/OVN 環境のコマンド入力の適切な順序を示しています。

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_ovshwol.yaml \
```

```
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-sriov.yaml \
-e /home/stack/templates/ovshwol-overrides.yaml
```

4. **openstack overcloud deploy** コマンドを実行します。
オーバークラウドの作成が完了すると、RHOSP director は、オーバークラウドへのアクセスに役立つ詳細を提供します。

検証

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのデプロイメントの検証 のステップを実行します。

次のステップ

1. NIC の e-switch モードが **switchdev** に設定されていることを確認します。
switchdev モードにより、NIC 上にレプリゼンターポートが確立され、VF にマッピングされます。

重要

OpenFlow フローをハードウェアにオフロードするには、接続追跡 (conntrack) モジュールの **switchdev** ポートでセキュリティーグループとポートセキュリティーを有効にする必要があります。

a. 次のコマンドを実行して NIC を確認します。

例

この例では、NIC **pci/0000:03:00.0** が照会されます。

```
$ sudo devlink dev eswitch show pci/0000:03:00.0
```

出力例

以下のような出力が表示されるはずです。

```
pci/0000:03:00.0: mode switchdev inline-mode none encap enable
```

b. NIC を **switchdev** モードに設定するには、次のコマンドを実行します。

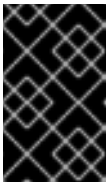
例

この例では、NIC **pci/0000:03:00.0** の e-switch モードが **switchdev** に設定されています。

```
$ sudo devlink dev eswitch set pci/0000:03:00.0 mode switchdev
```

2. **switchdev** 対応 NIC からポートを割り当てるには、次の手順を実行します。

a. **admin** ロールを持つ RHOSP ユーザーとしてログインし、**binding-profile** 値が **capabilities** の neutron ポートを作成し、ポートセキュリティーを無効にします。



重要

OpenFlow フローをハードウェアにオフロードするには、接続追跡 (conntrack) モジュールの **switchdev** ポートでセキュリティーグループとポートセキュリティーを有効にする必要があります。

```
$ openstack port create --network private --vnic-type=direct --binding-profile
'{"capabilities": ["switchdev"]}' direct_port1 --disable-port-security
```

b. インスタンスの作成時にこのポート情報を渡します。
レプリゼンターポートをインスタンスの VF インターフェイスに関連付け、ワンタイム OVS データパス処理のためにレプリゼンターポートを OVS ブリッジ **br-int** に接続します。VF ポートのレプリゼンターは、物理パッチパネルフロントエンドのソフトウェアバージョンのように機能します。

新しいインスタンスの作成の詳細は、「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのインスタンスの作成」 を参照してください。

3. インターフェイスおよびレプリゼンターポートに以下の設定を適用し、TC Flower がポートレベルでフロープログラミングをプッシュするようにします。

```
$ sudo ethtool -K <device-name> hw-tc-offload on
```

4. パフォーマンスを向上させるための、各ネットワークインターフェイスチャンネル数を調整します。
チャンネルには、割り込み要求 (IRQ) および IRQ のトリガーとなるキューのセットが含まれます。**mlx5_core** ドライバーを **switchdev** モードに設定すると、**mlx5_core** ドライバーはデフォルトである 単一の結合チャンネルに設定されます。この設定では、最適なパフォーマンスを得られない可能性があります。

Physical Fundtion (PF) マネージャーで次のコマンドを実行して、ホストで使用可能な CPU の数を調整します。

例

この例では、ネットワークインターフェイス **eno3s0f0** の多目的チャネルの数が **3** に設定されています。

```
$ sudo ethtool -L enp3s0f0 combined 3
```

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドの作成
- **コマンドラインインターフェイスリファレンス** の overcloud deploy
- 「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのインスタンスの作成」
- **ethtool** の man ページ

- **devlink** の man ページ
- **インスタンス作成のためのコンピュートサービスの設定** における コンピュートノードの CPU 固定の設定

8.7. SR-IOV または OVS TC-FLOWER ハードウェアオフロード環境でのホストアグリゲートの作成

Red Hat OpenStack Platform (RHOSP) SR-IOV または OVS TC-flower ハードウェアオフロード環境でパフォーマンスを向上させるには、CPU ピンニングと huge page を備えたゲストをデプロイします。アグリゲートメタデータをフレーバーメタデータに一致させることで、ホストのサブセット上にハイパフォーマンスインスタンスをスケジュールすることができます。

前提条件

- SR-IOV または OVS ハードウェアオフロード環境用に設定された RHOSP オーバークラウド。
- RHOSP オーバークラウドは **AggregateInstanceExtraSpecsFilter** 用に設定されている必要があります。
  詳細は、「OVS TC-flower ハードウェアオフロード用の PCI パススルーデバイスの設定」 を参照してください。

手順

1. 集約グループを作成し、関連するホストを追加します。
   定義するフレーバーメタデータに一致するメタデータを定義します (例: **sriov=true**)。

```
$ openstack aggregate create sriov_group
$ openstack aggregate add host sriov_group compute-sriov-0.localdomain
$ openstack aggregate set --property sriov=true sriov_group
```

2. フレーバーを作成します。

```
$ openstack flavor create <flavor> --ram <size_mb> --disk <size_gb> \
--vcpus <number>
```

3. 追加のフレーバー属性を設定します。
   定義したメタデータ (**sriov=true**) と SR-IOV アグリゲートで定義したメタデータが一致している点に注意してください。

```
$ openstack flavor set --property sriov=true \
--property hw:cpu_policy=dedicated \
--property hw:mem_page_size=1GB <flavor>
```

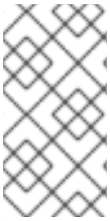
関連情報

- **コマンドラインインターフェイスリファレンス** の aggregate
- **コマンドラインインターフェイスリファレンス** の flavor

8.8. SR-IOV または OVS TC-FLOWER ハードウェアオフロード環境でのインスタンスの作成

Red Hat OpenStack Platform (RHOSP) SR-IOV または OVS TC-flower ハードウェアオフロード環境でインスタンスを作成するには、いくつかのコマンドを使用します。

ホストアグリゲートを使用して、ハイパフォーマンスコンピュートホストを分離します。詳細は、「SR-IOV または OVS TC-flower ハードウェアオフロード環境でのホストアグリゲートの作成」を参照してください。



注記

CPU ピニングを設定したインスタンスと設定していないインスタンスを、同じ Compute ノードに配置することができます。詳細は、**インスタンス作成のためのコンピューティングサービスの設定** ガイドの Compute ノードでの CPU ピンニングの設定 を参照してください。

前提条件

- SR-IOV または OVS ハードウェアオフロード環境用に設定された RHOSP オーバークラウド。
- OVS ハードウェアオフロード環境では、インスタンスを作成するには、RHOSP 管理者から Virtual Function (VF) ポートまたは Physical Fundtion (PF) ポートを取得する必要があります。OVS ハードウェアオフロードでは、VF または PF を作成するためのバインディングプロファイルが必要です。バインディングプロファイルを使用できるのは、**admin** ロールを持つ RHOSP ユーザーのみです。

手順

1. フレーバーを作成します。

```
$ openstack flavor create <flavor_name> --ram <size_mb> \
--disk <size_gb> --vcpus <number>
```

ヒント

フレーバーに追加スペック **hw:pci_numa_affinity_policy** を追加して、PCI パススルーデバイスおよび SR-IOV インターフェイスの NUMA アフィニティーポリシーを指定することができます。詳細は、**インスタンス作成のための Compute サービスの設定** の フレーバーメタデータ を参照してください。

2. ネットワークとサブネットを作成します。

```
$ openstack network create <network_name> \
--provider-physical-network tenant \
--provider-network-type vlan --provider-segment <vlan_id>

$ openstack subnet create <name> --network <network_name> \
--subnet-range <ip_address_cidr> --dhcp
```

3. **admin** ロールを持つ RHOSP ユーザーでない場合は、インスタンスを作成するために必要な VF または PF を RHOSP 管理者が提供できます。手順 5 に進みます。
4. **admin** ロールを持つ RHOSP ユーザーの場合は、VF ポートまたは PF ポートを作成できます。
   - VF ポート:

```
$ openstack port create --network <network_name> --vnic-type direct \
--binding-profile '{"capabilities": ["switchdev"]} <port_name>
```

- 単一インスタンス専用の PF ポート:
  この PF ポートは Networking サービス (neutron) ポートですが、Networking サービスによって制御されておらず、インスタンスにパススルーされる PCI デバイスであるため、ネットワークアダプターとして表示されません。

```
$ openstack port create --network <network_name> \
--vnic-type direct-physical <port_name>
```

5. インスタンスを作成します。

```
$ openstack server create --flavor <flavor> --image <image_name> \
--nic port-id=<id> <instance_name>
```

関連情報

- **コマンドラインインターフェイスリファレンス** の flavor create
- **コマンドラインインターフェイスリファレンス** の network create
- **コマンドラインインターフェイスリファレンス** の subnet create
- **コマンドラインインターフェイスリファレンス** の port create
- **コマンドラインインターフェイスリファレンス** の server create

8.9. OVS TC-FLOWER ハードウェアオフロードのトラブルシューティング

OVS TC-flower ハードウェアオフロードを使用する Red Hat OpenStack Platform (RHOSP) 環境のトラブルシューティングを行う場合は、ネットワークとインターフェイスの前提条件と設定を確認してください。

前提条件

- Linux カーネル 4.13 以降
- OVS 2.8 以降
- RHOSP 12 以降
- Iproute 4.12 以降
- Mellanox NIC ファームウェア (例: FW ConnectX-5 16.21.0338 以降)

サポート対象となる前提条件の詳細は、Red Hat ナレッジベースのソリューション Network Adapter Fast Datapath Feature Support Matrix を参照してください。

ネットワーク設定

HW オフロードのデプロイメントでは、ネットワーク設定として、以下のシナリオのどちらかを要件に応じて使用することができます。

- ボンディングに接続された同じインターフェイスセットを使用するか、種別ごとに異なる NIC セットを使用して、VXLAN および VLAN 上でゲスト仮想マシンをホストすることができます。
- Linux ボンディングを使用して、Mellanox NIC の 2 つのポートをボンディングすることができます。
- Mellanox Linux ボンディングに加えて、VLAN インターフェイス上でテナント VXLAN ネットワークをホストすることができます。

個々の NIC およびボンディングが ovs-bridge のメンバーになるように設定します。

次のネットワーク設定例を参照してください。

```
...
- type: ovs_bridge
  name: br-offload
  mtu: 9000
  use_dhcp: false
  members:
  - type: linux_bond
    name: bond-pf
    bonding_options: "mode=active-backup miimon=100"
    members:
    - type: sriov_pf
      name: p5p1
      numvfs: 3
      primary: true
      promisc: true
      use_dhcp: false
      defroute: false
      link_mode: switchdev
    - type: sriov_pf
      name: p5p2
      numvfs: 3
      promisc: true
      use_dhcp: false
      defroute: false
      link_mode: switchdev
...
 - type: vlan
   vlan_id:
     get_param: TenantNetworkVlanID
   device: bond-pf
   addresses:
   - ip_netmask:
       get_param: TenantIpSubnet
...
```

次のボンディング設定がサポートされています。

- active-backup: mode=1
- active-active または balance-xor: mode=2
- 802.3ad (LACP): mode=4

以下のボンディング設定はサポートされません。

- xmit_hash_policy=layer3+4

インターフェイス設定

インターフェイス設定を確認するには、次の手順に従います。

手順

1. デプロイメント時に、ホストネットワーク設定ツール **os-net-config** を使用して **hw-tc-offload** を有効にします。
2. Compute ノードをリブートするたびに、**sriov_config** サービスで **hw-tc-offload** を有効にします。
3. ボンディングに接続されている NIC について、**hw-tc-offload** パラメーターを **on** に設定します。

   例

   ```
   $ ethtool -k ens1f0 | grep tc-offload

   hw-tc-offload: on
   ```

インターフェイスモード

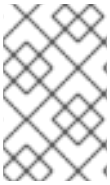
次の手順でインターフェイスモードを確認します。

手順

1. HW オフロードに使用するインターフェイスの eswitch モードを **switchdev** に設定します。
2. ホストネットワーク設定ツール **os-net-config** を使用して、デプロイメント時に **eswitch** を有効にします。
3. Compute ノードをリブートするたびに、**sriov_config** サービスで **eswitch** を有効にします。

   例

   ```
   $ devlink dev eswitch show pci/$(ethtool -i ens1f0 | grep bus-info \
   | cut -d ':' -f 2,3,4 | awk '{$1=$1};1')
   ```



注記

PF インターフェイスのドライバーが **"mlx5e_rep"** に設定され、e-switch アップリンクポートのレプリゼンターであることが示されます。これは機能には影響を及ぼしません。

OVS オフロード状態

OVS オフロードの状態を確認するには、次の手順に従います。

- Compute ノードにおいて、OVS のハードウェアオフロードを有効にします。

```
$ ovs-vsctl get Open_vSwitch . other_config:hw-offload

"true"
```

VF 代表ポート名

VF レプリゼンターポートの命名に一貫性を持たせるために、**os-net-config** は udev ルールを使用してポートの名前を <PF-name>_<VF_id> の形式で変更します。

手順

- デプロイメント後に、VF レプリゼンターポートの名前が正しく付けられていることを確認します。

  例

  ```
  $ cat /etc/udev/rules.d/80-persistent-os-net-config.rules
  ```

  出力例

  ```
  # This file is autogenerated by os-net-config

  SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}!="",
  ATTR{phys_port_name}=="pf*vf*", ENV{NM_UNMANAGED}="1"
  SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", KERNELS=="0000:65:00.0",
  NAME="ens1f0"
  SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}=="98039b7f9e48",
  ATTR{phys_port_name}=="pf0vf*", IMPORT{program}="/etc/udev/rep-link-name.sh
  $attr{phys_port_name}", NAME="ens1f0_$env{NUMBER}"
  SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", KERNELS=="0000:65:00.1",
  NAME="ens1f1"
  SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}=="98039b7f9e49",
  ATTR{phys_port_name}=="pf1vf*", IMPORT{program}="/etc/udev/rep-link-name.sh
  $attr{phys_port_name}", NAME="ens1f1_$env{NUMBER}"
  ```

ネットワークトラフィックフロー

HW オフロードが設定されたネットワークフローは、特定用途向け集積回路 (ASIC) チップを持つ物理スイッチまたはルーターと同じ様に機能します。

スイッチまたはルーターの ASIC シェルにアクセスして、ルーティングテーブルの調査や他のデバッグを行うことができます。以下の手順では、例として Cumulus Linux スイッチの Broadcom チップセットを使用しています。実際の環境に応じて値を置き換えてください。

手順

1. Broadcom チップのルーティングテーブルの内容を取得するには、**bcmcmd** コマンドを使用します。

   ```
   $ cl-bcmcmd l2 show
   ```

   出力例

```
mac=00:02:00:00:00:08 vlan=2000 GPORT=0x2 modid=0 port=2/xe1
mac=00:02:00:00:00:09 vlan=2000 GPORT=0x2 modid=0 port=2/xe1 Hit
```

2. トラフィック制御 (TC) レイヤーを検査します。

```
$ tc -s filter show dev p5p1_1 ingress
```

出力例

```
…
filter block 94 protocol ip pref 3 flower chain 5
filter block 94 protocol ip pref 3 flower chain 5 handle 0x2
  eth_type ipv4
  src_ip 172.0.0.1
  ip_flags nofrag
  in_hw in_hw_count 1
        action order 1: mirred (Egress Redirect to device eth4) stolen
        index 3 ref 1 bind 1 installed 364 sec used 0 sec
        Action statistics:
        Sent 253991716224 bytes 169534118 pkt (dropped 0, overlimits 0 requeues 0)
        Sent software 43711874200 bytes 30161170 pkt
        Sent hardware 210279842024 bytes 139372948 pkt
        backlog 0b 0p requeues 0
        cookie 8beddad9a0430f0457e7e78db6e0af48
        no_percpu
```

3. この出力で **in_hw** フラグおよび統計値を調べます。**hardware** という言葉は、ハードウェアがネットワークトラフィックを処理していることを示しています。**tc-policy=none** を使用する場合は、この出力または tcpdump を確認して、ハードウェアまたはソフトウェアがパケットを処理するタイミングを調べることができます。ドライバーがパケットをオフロードできない場合は、**dmesg** または **ovs-vswitch.log** に対応するログメッセージが表示されます。

4. Mellanox を例に取ると、ログエントリーは **dmesg** の徴候メッセージに類似しています。

出力例

```
[13232.860484] mlx5_core 0000:3b:00.0: mlx5_cmd_check:756:(pid 131368):
SET_FLOW_TABLE_ENTRY(0x936) op_mod(0x0) failed, status bad parameter(0x3),
syndrome (0x6b1266)
```

この例では、エラーコード (0x6b1266) は以下の動作を表します。

出力例

```
0x6B1266 |  set_flow_table_entry: pop vlan and forward to uplink is not allowed
```

Systems

以下の手順で、ご自分のシステムを検証します。

手順

1. システムで SR-IOV および VT-d が有効であることを確認します。

2. たとえば GRUB を使用して、カーネルパラメーターに **intel_iommu=on** を追加して Linux の IOMMU を有効にします。

8.10. TC-FLOWER ハードウェアオフロードフローのデバッグ

ovs-vswitch.log ファイルに次のメッセージが表示される場合は、以下の手順を使用することができます。

```
2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow: Operation not supported: p6p1_5
```

手順

1. オフロードモジュールのロギングを有効にし、この障害に関する追加のログ情報を取得するには、Compute ノードで以下のコマンドを使用します。

```
ovs-appctl vlog/set dpif_netlink:file:dbg
# Module name changed recently (check based on the version used
ovs-appctl vlog/set netdev_tc_offloads:file:dbg [OR] ovs-appctl vlog/set
netdev_offload_tc:file:dbg
ovs-appctl vlog/set tc:file:dbg
```

2. 再度 **ovs-vswitchd** ログを調べ、問題に関する追加情報を確認します。
以下に示すログの例では、サポートされない属性マークが原因でオフロードに失敗しています。

```
 2020-01-31T06:22:11.218Z|00471|dpif_netlink(handler402)|DBG|system@ovs-system:
put[create] ufid:61bd016e-eb89-44fc-a17e-958bc8e45fda
recirc_id(0),dp_hash(0/0),skb_priority(0/0),in_port(7),skb_mark(0),ct_state(0/0),ct_zone(0/0),ct
_mark(0/0),ct_label(0/0),eth(src=fa:16:3e:d2:f5:f3,dst=fa:16:3e:c4:a3:eb),eth_type(0x0800),ipv
4(src=10.1.1.8/0.0.0.0,dst=10.1.1.31/0.0.0.0,proto=1/0,tos=0/0x3,ttl=64/0,frag=no),icmp(type=0/
0,code=0/0),
actions:set(tunnel(tun_id=0x3d,src=10.10.141.107,dst=10.10.141.124,ttl=64,tp_dst=4789,flags(
df|key))),6

2020-01-31T06:22:11.253Z|00472|netdev_tc_offloads(handler402)|DBG|offloading attribute
pkt_mark isn't supported

2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow:
Operation not supported: p6p1_5
```

Mellanox NIC のデバッグ

Mellanox は、Red Hat の SOS レポートに類似したシステム情報スクリプトを提供しています。

https://github.com/Mellanox/linux-sysinfo-snapshot/blob/master/sysinfo-snapshot.py

このコマンドを実行すると、サポートケースで役立つ、関連ログ情報の zip ファイルが作成されます。

手順

- 以下のコマンドを使用して、このシステム情報スクリプトを実行することができます。

```
# ./sysinfo-snapshot.py --asap --asap_tc --ibdiagnet --openstack
```

-

Mellanox Firmware Tools (MFT)、mlxconfig、mlxlink、および OpenFabrics Enterprise Distribution (OFED) ドライバーをインストールすることもできます。

有用な CLI コマンド

以下のオプションと共に **ethtool** ユーティリティーを使用して、診断情報を収集します。

- ethtool -l <uplink representor>: チャンネル数の表示
- ethtool -l <uplink/VFs>: 統計値の確認
- ethtool -i <uplink rep>: ドライバー情報の表示
- ethtool -g <uplink rep>: リングサイズの確認
- ethtool -k <uplink/VFs>: 有効な機能の表示

レプリゼンターポートおよび PF ポートで **tcpdump** ユーティリティーを使用して、同様にトラフィックフローを確認します。

- レプリゼンターポートのリンク状態に加えた変更は、すべて VF のリンク状態にも影響を及ぼします。
- レプリゼンターポートの統計値には、VF の統計も表示されます。

以下のコマンドを使用して、有用な診断情報を取得します。

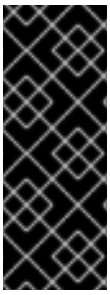
```
$ ovs-appctl dpctl/dump-flows -m type=offloaded

$ ovs-appctl dpctl/dump-flows -m

$ tc filter show dev ens1_0 ingress

$ tc -s filter show dev ens1_0 ingress

$ tc monitor
```

第9章 OVS-DPDK デプロイメントのプランニング

NFV 向けの Data Plane Development Kit 対応 Open vSwitch (OVS-DPDK) デプロイメントを最適化するには、OVS-DPDK が Compute 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、Compute 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。

重要

OVS-DPDK および OVS ネイティブファイアウォール (conntrack に基づくステートフルファイアウォール) を使用する場合、追跡することができるのは ICMPv4、ICMPv6、TCP、および UDP プロトコルを使用するパケットだけです。OVS は、その他すべてのネットワークトラフィック種別を無効と識別します。



重要

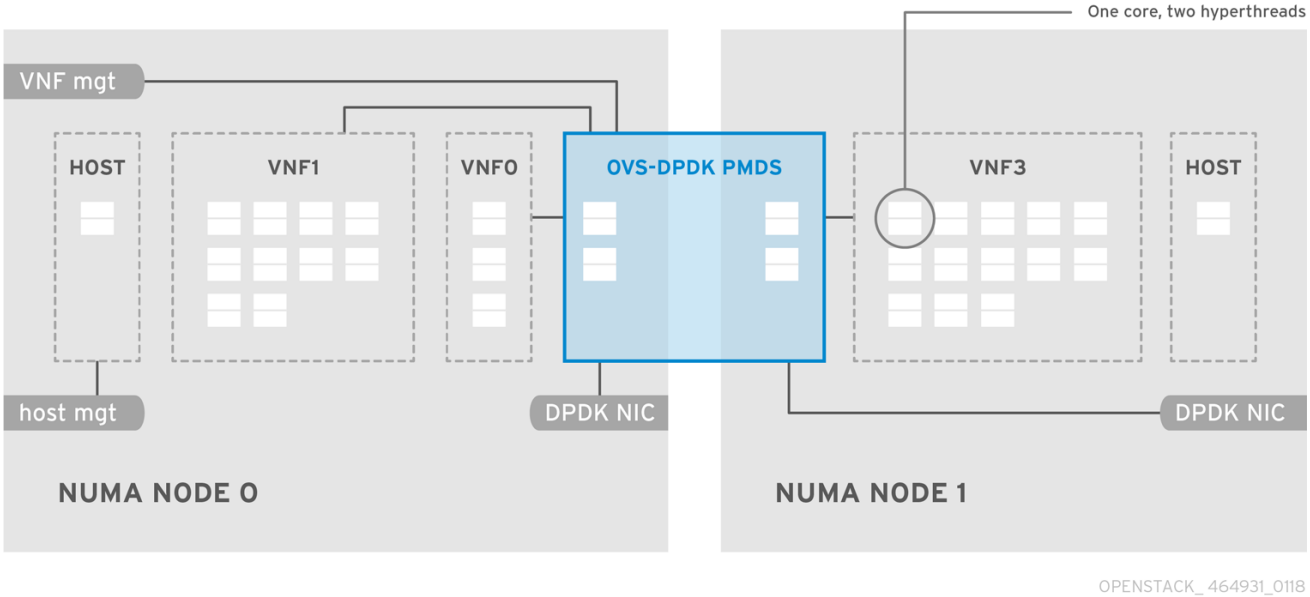
Red Hat は、非 NFV ワークロードでの OVS-DPDK の使用をサポートしていません。NFV 以外のワークロードに OVS-DPDK 機能が必要な場合は、テクニカルアカウントマネージャー (TAM) に連絡するか、カスタマーサービスリクエストケースを開いて、サポートの例外やその他のオプションについて話し合ってください。カスタマーサービスリクエストケースを作成するには、ケースの作成 に移動し、**アカウント > カスタマーサービスリクエスト** を選択します。

9.1. 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、およびそれ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 CPU コアを必要とする DPDK アクティブループを実行します。したがって、一部の CPU およびヒュージページを OVS-DPDK に割り当てる必要があります。

サンプルの分割では、デュアルソケットの Compute 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 間で NIC を共有することができないので、トラフィックには追加の NIC が必要です。

図9.1 NUMA トポロジー: CPU パーティショニングを備えた OVS-DPDK



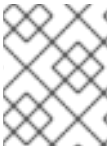
注記

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。

最高の OVS-DPDK パフォーマンスを得るためには、NUMA ノードにローカルなメモリーブロックを確保します。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 を選択してください。ボンディングを設定する両方のインターフェイスには、同じ NUMA ノード上の NIC を使用するようにしてください。

9.2. OVS-DPDK パラメーター

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。この情報を使用して、Compute ノードでのハードウェアサポートを評価すると共に、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方法を評価します。



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 (あるいは論理 CPU) をペアにしてください。

Compute ノード上の CPU と NUMA ノードを特定する方法の詳細は、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。

9.2.1. CPU パラメーター

OVS-DPDK では、以下に示す CPU の分割用パラメーターが使用されます。

OvsPmdCoreList

DPDK Poll Mode Driver (PMD) に使用する CPU コアを提供します。DPDK インターフェイスのローカルの NUMA ノードに関連付けられた CPU コアを選択します。OVS の **pmd-cpu-mask** の値に **OvsPmdCoreList** を使用します。**OvsPmdCoreList** に関する以下の推奨事項に従ってください。

- シブリングスレッドをペアにします。
- 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 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 NIC を持つ NUMA ノードの場合には、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、各物理コアの全シブリングスレッド (あるいは論理 CPU) を追加します。
- DPDK NIC を持たない NUMA ノードの場合には、任意の物理コア (ただし NUMA ノードの 1 番目の物理コアを除く) のシブリングスレッド (あるいは論理 CPU) を割り当てます。

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。

NovaComputeCpuDedicatedSet

ピニングされたインスタンス CPU のプロセスをスケジューリングできる物理ホスト CPU 番号のコンマ区切りリストまたは範囲。たとえば、**NovaComputeCpuDedicatedSet: [4-12,^8,15]** は、コア 4 - 12 の範囲 (ただし 8 を除く) および 15 を確保します。

- **OvsPmdCoreList** のコアをすべて除外します。
- 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。

NovaComputeCpuSharedSet

物理ホスト CPU 番号のコンマ区切りリストまたは範囲。インスタンスエミュレータースレッド用のホスト CPU を決定するのに使用します。

IsolCpusList

ホストのプロセスから分離される CPU コアのセット。**IsolCpusList** 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** の値として使用されます。**IsolCpusList** に関する以下の推奨事項に従ってください。

- **OvsPmdCoreList** および **NovaComputeCpuDedicatedSet** のコアリストと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。

DerivePciWhitelistEnabled

仮想マシン用に Virtual Function (VF) を確保するには、**NovaPCIPassthrough** パラメーターを使用して Nova に渡される VF のリストを作成します。リストから除外された VF は、引き続きホスト用に利用することができます。
リスト内の VF ごとに、アドレス値に解決する正規表現でアドレスパラメーターを反映させます。

手動でリストを作成するプロセスの例を以下に示します。**eno2** という名前のデバイスで NIC の分割が有効な場合は、以下のコマンドで VF の PCI アドレスをリスト表示します。

```
[tripleo-admin@compute-0 ~]$ ls -lh /sys/class/net/eno2/device/ | grep virtfn
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn0 -> ../0000:18:06.0
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn1 -> ../0000:18:06.1
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn2 -> ../0000:18:06.2
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn3 -> ../0000:18:06.3
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn4 -> ../0000:18:06.4
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn5 -> ../0000:18:06.5
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn6 -> ../0000:18:06.6
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn7 -> ../0000:18:06.7
```

この場合、VF 0、4、および 6 が NIC の分割用に **eno2** で使用されます。以下の例に示すように、**NovaPCIPassthrough** を手動で設定して VF 1 - 3、5、および 7 を含めます。したがって、VF 0、4、および 6 は除外します。

```
NovaPCIPassthrough:
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[1-3]"}
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[5]"}
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[7]"}
```

9.2.2. 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** は、OVS の **other_config:dpdk-extra="-n <value>"** の値に使用されます。**OvsDpdkMemoryChannels** に関する以下の推奨事項を確認してください。

- **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。**NovaReservedHostMemory** は、Compute ノードの **nova.conf** の **reserved_host_memory_mb** の値に使用されます。**NovaReservedHostMemory** に関する以下の推奨事項を確認してください。

- 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。**OvsDpdkSocketMemory** は、OVS の **other_config:dpdk-socket-mem** の値に使用されます。**OvsDpdkSocketMemory** に関する以下の推奨事項を確認してください。

- コンマ区切りリストで指定します。
- 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 (1 GB) を使用します。
- NUMA ノード上の各 NIC の MTU 値から、**OvsDpdkSocketMemory** の値を計算します。
- **OvsDpdkSocketMemory** の値は、以下の式で概算します。
  - MEMORY_REQD_PER_MTU = (ROUNDUP_PER_MTU + 800) x (4096 x 64) バイト
    - 800 はオーバーヘッドの値です。
    - 4096 x 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、バッファーとして 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 0 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。

```
The MTU value of 9000 becomes 9216 bytes.
The MTU value of 2000 becomes 2048 bytes.
```

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。

```
Memory required for 9000 MTU = (9216 + 800) * (4096*64) = 2625634304
Memory required for 2000 MTU = (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 bytes.
```

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 バッファー) 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。

```
3724 MB rounds up to 4096 MB.
```

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

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

計算例: MTU 2000

DPDK NIC dpdk0 と dpdk1 は同じ NUMA ノード 0 上にあり、共に 2000 の MTU が設定されています。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。

```
The MTU value of 2000 becomes 2048 bytes.
```

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。

```
Memory required for 2000 MTU = (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 bytes.
```

この計算は、(MTU 値 2000 に必要なメモリー) + (512 MB バッファー) 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 rounds up to 2048 MB.
```

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

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

9.2.3. ネットワークパラメーター

OvsDpdkDriverType

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。

9.2.4. その他のパラメーター

NovaSchedulerEnabledFilters

要求されたゲストインスタンスに対して Compute 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。

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。デフォルト値は **qemu** です。**VhostuserSocketGroup** を **hugetlbfs** に設定します。これにより、**ovs-vswitchd** および **qemu** プロセスが、共有ヒュージページおよび virtio-net デバイスを設定する unix ソケットにアクセスすることができます。この値はロールに固有で、OVS-DPDK を利用するすべてのロールに適用する必要があります。

重要

パラメーター **VhostuserSocketGroup** を使用するには、**NeutronVhostuserSocketDir** も設定する必要があります。詳細は、「ネットワークパラメーター」 を参照してください。

KernelArgs

Compute 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、以下の値を追加します。

- **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**)。このコマンドを使用して **pdpe1gb** CPU フラグが出力されるかどうかをチェックして、CPU が 1 G 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**: 利用可能なホストメモリーに基づいてヒュージページの数を設定します。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、Compute ノードの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**: チューニングする CPU コアを設定します。この値は **IsolCpusList** と一致します。

CPU 分離の詳細は、Red Hat ナレッジベースのソリューション記事 OpenStack CPU isolation guidance for RHEL 8 and RHEL 9 を参照してください。

DdpPackage

Dynamic Device Personalization (DDP) を設定して、デプロイメント時にプロファイルパッケージをデバイスに適用し、デバイスのパケット処理パイプラインを変更します。次の行を network_environment.yaml テンプレートに追加して、DDP パッケージを含めます。

```
parameter_defaults:
  ComputeOvsDpdkSriovParameters:
    DdpPackage: "ddp-comms"
```

9.2.5. VM インスタンスフレーバーの仕様

NFV 環境で VM インスタンスをデプロイする前に、CPU ピニング、ヒュージページ、およびエミュレータースレッドピニングを活用するフレーバーを作成します。

hw:cpu_policy

このパラメーターを **dedicated** に設定すると、ゲストはピニングされた CPU を使用します。このパラメーターセットのフレーバーから作成したインスタンスの有効オーバーコミット比は、1:1 です。デフォルト値は **shared** です。

hw:mem_page_size

このパラメーターを、特定の値と標準の単位からなる有効な文字列に設定します (例: **4KB**、**8MB**、または **1GB**)。**hugepagesz** ブートパラメーターに一致させるには、1GB を使用します。ブートパラメーターから **OvsDpdkSocketMemory** を減算して、仮想マシンが利用可能なヒュージページ数を計算します。以下の値も有効です。

- small (デフォルト): 最少のページサイズが使用されます。
- large: 大型のページサイズだけを使用します。x86 アーキテクチャーでは、ページサイズは 2 MB または 1 GB です。
- any: Compute ドライバーは大型ページの使用を試みることができますが、利用できるページがない場合にはデフォルトの小型ページが使用されます。

hw:emulator_threads_policy

heat パラメーター **NovaComputeCpuSharedSet** で識別した CPU にエミュレータースレッドが固定されるように、このパラメーターの値を **share** に設定します。エミュレータースレッドが Poll Mode Driver (PMD) またはリアルタイム処理用の仮想 CPU 上で実行されている場合には、パケットロスなどの悪影響が生じる場合があります。

9.3. OVS-DPDK デプロイメントにおける電力節約

省電力プロファイル **cpu-partitioning-powersave** が Red Hat Enterprise Linux 9 (RHEL 9) で導入され、Red Hat OpenStack Platform (RHOSP) 17.1.3 で利用できるようになりました。この TuneD プロファイルは、RHOSP 17.1 NFV 環境で電力を節約するための基本的なビルディングブロックです。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- 電力節約を実現したい CPU では、より高い C ステートを許可できるようになります。詳細は、**tuned-profiles-cpu-partitioning(7)** の man ページの **max_power_state** オプションを参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. Ansible Playbook YAML ファイル (例: **/home/stack/cli-overcloud-tuned-maxpower-conf.yaml**) を作成します。
4. **cli-overcloud-tuned-maxpower-conf.yaml** ファイルに次の設定を追加します。

```
cat <<EOF > /home/stack/cli-overcloud-tuned-maxpower-conf.yaml
{% raw %}
---
#/home/stack/cli-overcloud-tuned-maxpower-conf.yaml
- name: Overcloud Node set tuned power state
  hosts: compute-0 compute-1
  any_errors_fatal: true
  gather_facts: false
  pre_tasks:
    - name: Wait for provisioned nodes to boot
      wait_for_connection:
        timeout: 600
        delay: 10
      connection: local
  tasks:
    - name: Check the max power state for this system
      become: true
      block:
        - name: Get power states
          shell: "for s in /sys/devices/system/cpu/cpu2/cpuidle/*; do grep . $s/{name,latency};
done"
          register: _list_of_power_states
        - name: Print available power states
          debug:
            msg: "{{ _list_of_power_states.stdout.split('\n') }}"
        - name: Check for active tuned power-save profile
          stat:
            path: "/etc/tuned/active_profile"
          register: _active_profile
        - name: Check the profile
          slurp:
            path: "/etc/tuned/active_profile"
          when: _active_profile.stat.exists
          register: _active_profile_name
        - name: Print states
          debug:
```

```
            var: (_active_profile_name.content|b64decode|string)
          - name: Check the max power state for this system
            block:
              - name: Check if the cstate config is present in the conf file
                lineinfile:
                  dest: /etc/tuned/cpu-partitioning-powersave-variables.conf
                  regexp: '^max_power_state'
                  line: 'max_power_state=cstate.name:C6'
                register: _cstate_entry_check
{% endraw %}
EOF
```

5. ロールデータファイルに省電力プロファイルを追加します。
   詳細は、10.2. ロールとイメージファイルの生成 を参照してください。

6. **cli-overcloud-tuned-maxpower-conf.yaml** Playbook をベアメタルノード定義ファイルに追加します。
   詳細は、10.5. ベアメタルノード定義ファイルの作成 を参照してください。

7. NIC 設定テンプレートでキューサイズが設定されていることを確認します。
   詳細は、10.6. NIC 設定テンプレートの作成 を参照してください。

関連情報

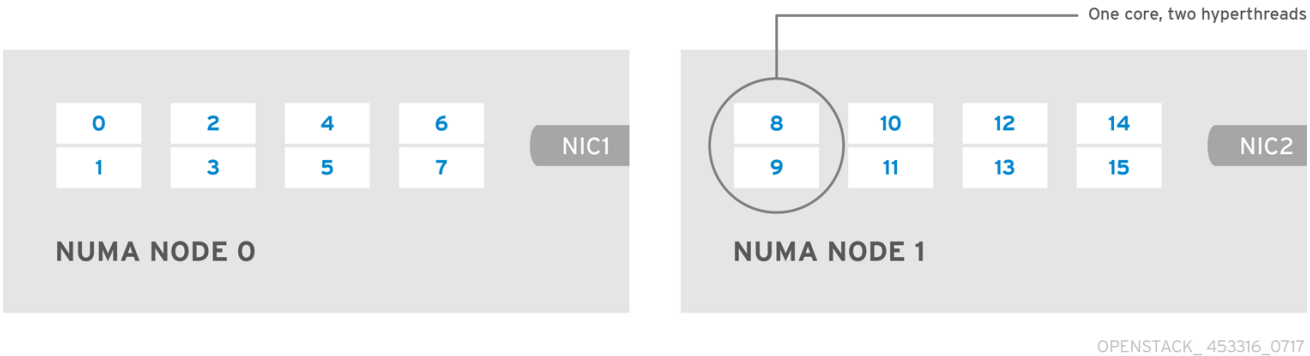
- **force_latency**

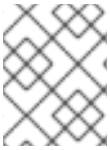
9.4. 2 NUMA ノード設定の OVS-DPDK デプロイメントの例

以下の例に示す Compute ノードは、2 つの NUMA ノードで設定されます。

- 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)、および (6,7) の設定。
- 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)、および (14,15) の設定。
- 各 NUMA ノードが物理 NIC (具体的には NUMA 0 上の NIC1 および NUMA 1 上の NIC2) に接続されている。

図9.2 OVS-DPDK: 2 つの NUMA ノードの例





注記

各 NUMA ノード上の 1 番目の物理コアまたは両スレッドペア (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

このユースケースでは、NUMA 0 の物理コアの 1 つを PMD 用に割り当てます。NUMA 1 の NIC では DPDK は有効化されていませんが、その NUMA ノードの物理コアの 1 つも割り当てる必要があります。残りの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"
NovaComputeCpuDedicatedSet: "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、NUMA 0 の物理コアの 2 つを PMD 用に割り当てます。NUMA 1 の NIC では DPDK は有効化されていませんが、その NUMA ノードの物理コアの 1 つも割り当てる必要があります。残りの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"
NovaComputeCpuDedicatedSet: "6,7,12,13,14,15"
```

NIC 2 は DPDK 用で、1 つの物理コアは PMD 用

このユースケースでは、NUMA 1 の物理コアの 1 つを PMD 用に割り当てます。NUMA 0 の NIC では DPDK は有効化されていませんが、その NUMA ノードの物理コアの 1 つも割り当てる必要があります。残りの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"
NovaComputeCpuDedicatedSet: "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、NUMA 1 の物理コアの 2 つを PMD 用に割り当てます。NUMA 0 の NIC では DPDK は有効化されていませんが、その NUMA ノードの物理コアの 1 つも割り当てる必要があります。残りの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"
NovaComputeCpuDedicatedSet: "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、各 NUMA ノードの物理コアの 2 つを PMD 用に割り当てます。残りの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"
NovaComputeCpuDedicatedSet: "6,7,14,15"
```

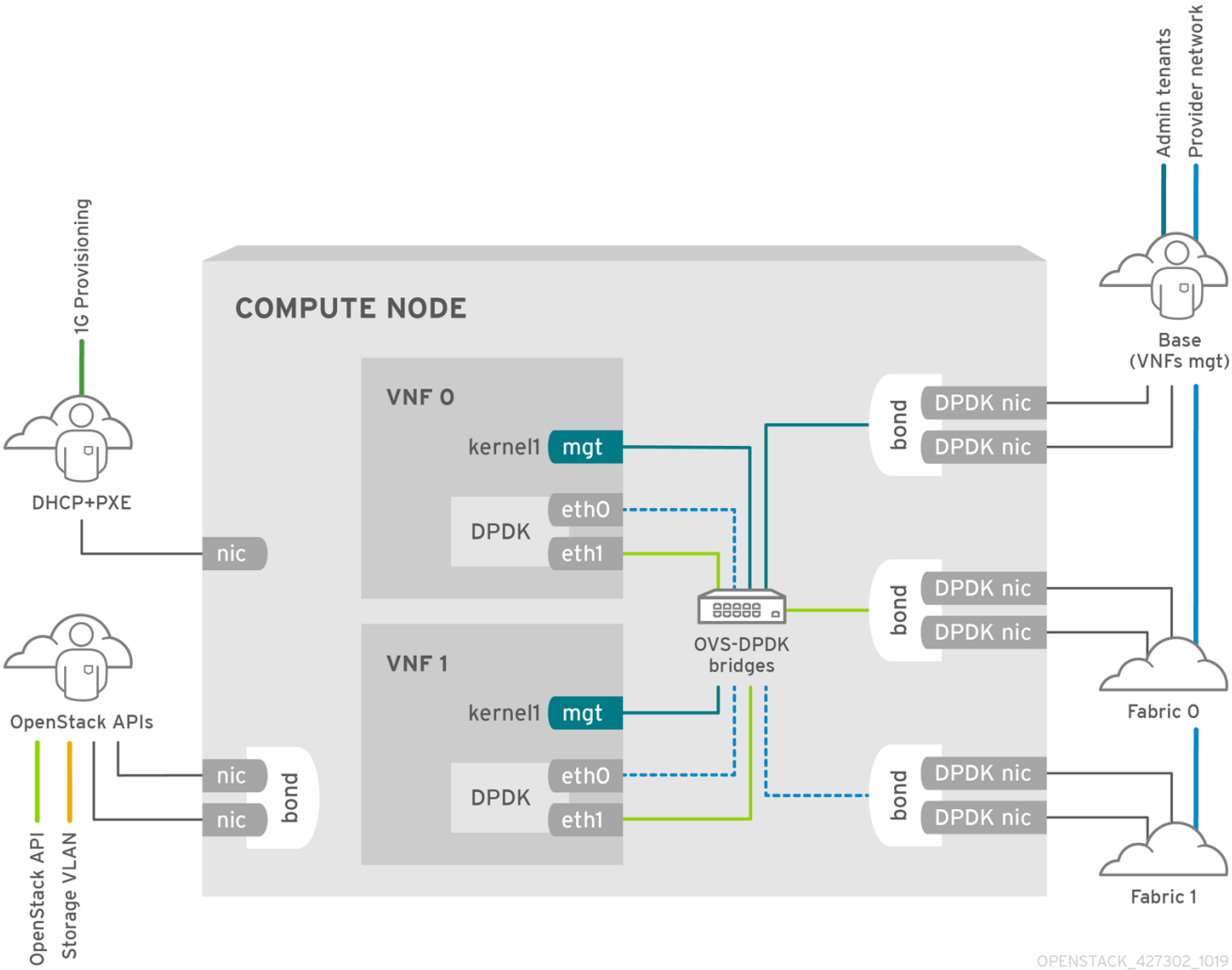
9.5. NFV OVS-DPDK デプロイメントのトポロジー

以下のデプロイメント例は、2 つの仮想ネットワーク機能 (VNF) からなる OVS-DPDK 設定を示しています。それぞれの NVF は、次の 2 つのインターフェイスを持ちます。

- **mgt** で表される管理インターフェイス
- データプレーンインターフェイス

OVS-DPDK デプロイメントでは、VNF は物理インターフェイスをサポートする組み込みの DPDK で動作します。OVS-DPDK は、vSwitch レベルでボンディングを有効にします。OVS-DPDK デプロイメントでのパフォーマンスを向上させるには、カーネルと OVS-DPDK NIC を分離することを推奨します。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 を利用できるようにします。Compute ノードは、Red Hat OpenStack Platform API 管理向けの標準 NIC 2 つで設定されます。これは、Ceph API で再利用できますが、OpenStack プロジェクトとは一切共有できません。

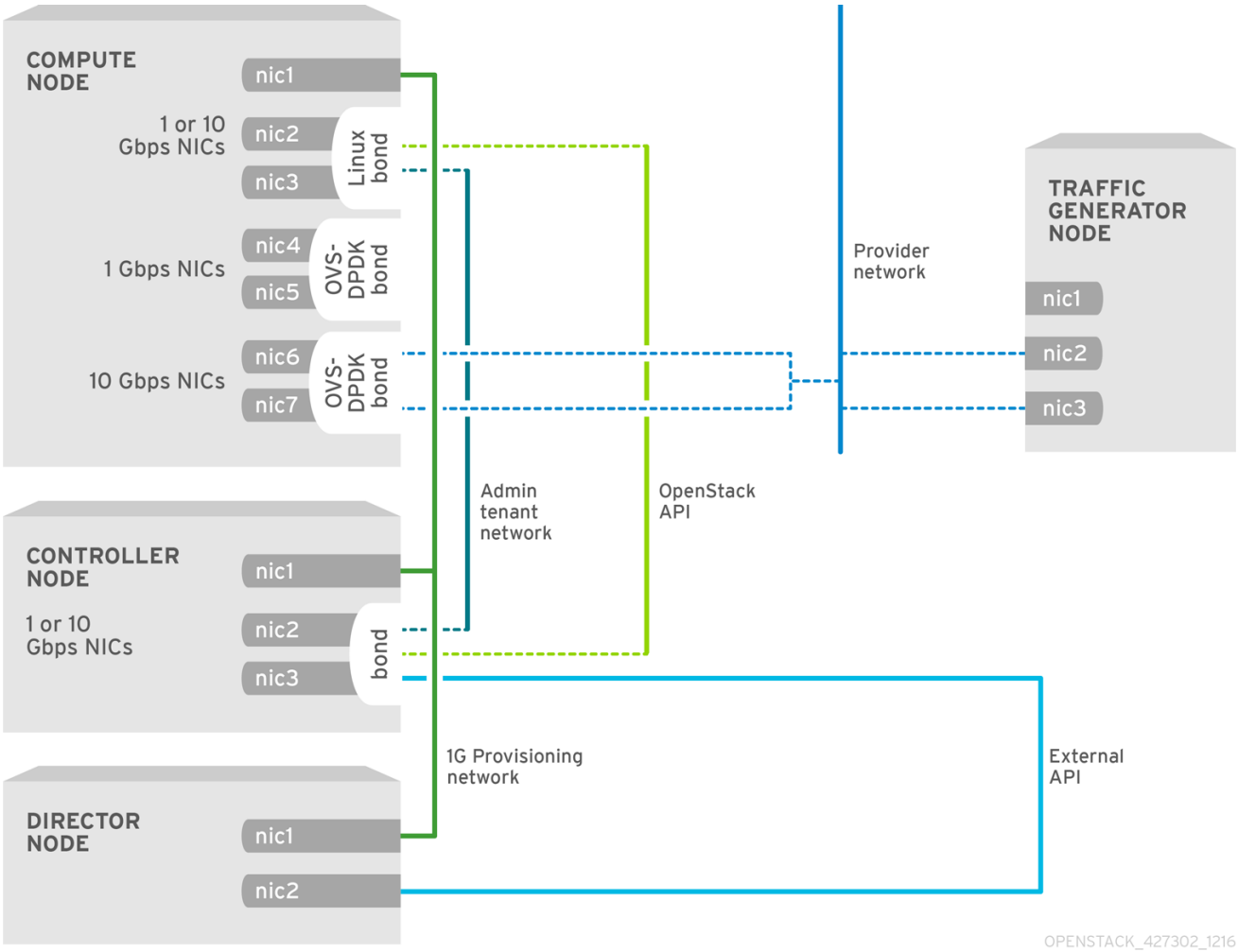
図9.3 Compute ノード: NFV OVS-DPDK



NFV OVS-DPDK のトポロジー

以下の図には、NFV 向けの OVS_DPDK のトポロジーを示しています。この環境は、1 または 10 Gbps の NIC を搭載した Compute ノードおよびコントローラーノードと、director ノードで設定されます。

図9.4 NFV トポロジー: OVS-DPDK

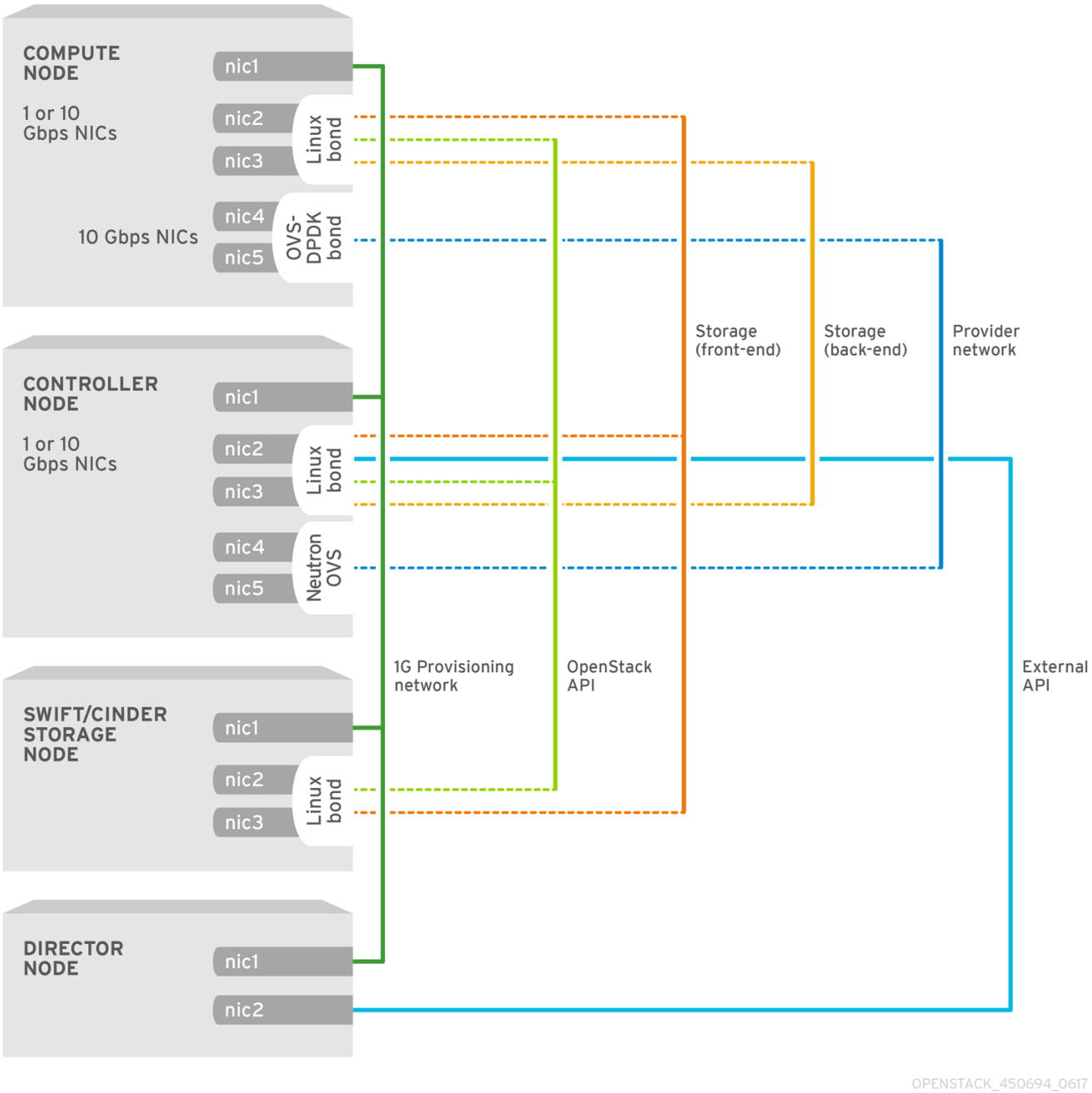


第10章 OVS-DPDK デプロイメントの設定

このセクションでは、Red Hat OpenStack Platform (RHOSP) 環境向けの Open vSwitch Data Plane Development Kit (OVS-DPDK) をデプロイ、使用、およびトラブルシューティングする方法を説明します。RHOSP は、OVS-DPDK デプロイメントでは OVS クライアントモードで動作します。

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 トポロジーを示しています。

図10.1 OVS-DPDK トポロジーの例



重要

このセクションには、トポロジーとユースケースに合わせて変更する必要がある例が含まれています。詳細は、NFV のハードウェア要件 を参照してください。

前提条件

- RHOSP アンダークラウド。
  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。詳細は、**director を使用した Red Hat OpenStack Platform のインストールおよび管理** を参照してください。

注記

RHOSP director は、テンプレートとカスタム環境ファイルで指定したキーと値のペアを通じて OVS-DPDK 設定ファイルを変更します。OVS-DPDK ファイルを直接変更しないでください。

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

Red Hat OpenStack Platform (RHOSP) director を使用して、RHOSP 環境に OVS-DPDK をインストールおよび設定します。大まかな手順は次のとおりです。

1. OVS-DPDK の既知の制限事項を確認します。
2. ロールとイメージファイルを生成します。
3. OVS-DPDK カスタマイズ用の環境ファイルを作成します。
4. セキュリティーグループのファイアウォールを設定します。
5. ベアメタルノード定義ファイルを作成します。
6. NIC 設定テンプレートを作成します。
7. OVS-DPDK インターフェイスの MTU 値の設定。
8. OVS-DPDK インターフェイス向けのマルチキューの設定。
9. ノードのプロビジョニング用の DPDK パラメーターを設定します。
10. オーバークラウドネットワークと仮想 IP をプロビジョニングします。
    詳細は以下を参照してください。
    - **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
    - **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング
11. ベアメタルノードをプロビジョニングします。
    director を使用した Red Hat OpenStack Platform のインストールと管理ガイドの オーバークラウドのベアメタルノードのプロビジョニング
12. OVS-DPDK オーバークラウドをデプロイします。

関連情報

- 「OVS-DPDK 用のフレーバーの作成とインスタンスのデプロイ」
- 「OVS-DPDK 設定のトラブルシューティング」

10.1. OVS-DPDK の既知の制限

Open vSwitch Data Plane Development Kit (OVS-DPDK) 環境で Red Hat OpenStack Platform を設定する場合は、次の制限事項に従ってください。

- 非 DPDK トラフィックおよびコントロールプレーンネットワーク (内部、管理、ストレージ、ストレージ管理、テナント等) 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 ブリッジ設定はサポートしていません。
- 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ヒュージページが必要です。ゲストのヒュージページがない場合には、インターフェイスは表示されても機能しません。
- OVS-DPDK を使用する場合には、分散仮想ルーター (DVR) 等の TAP 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。得られるパフォーマンスは、実稼働環境に適するものではありません。
- OVS-DPDK を使用する場合には、同じ Compute ノード上の全ブリッジが **ovs_user_bridge** の種別でなければなりません。同じノード上で **ovs_bridge** と **ovs_user_bridge** が混在する設定は、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。

次のステップ

- 「ロールとイメージファイルの生成」 に進みます。

10.2. ロールとイメージファイルの生成

Red Hat OpenStack Platform (RHOSP) ディレクターは、ロールを使用してノードにサービスを割り当てます。OVS-DPDK 環境に RHOSP をデプロイする場合、**ComputeOvsDpdk** は、デフォルトのコンピュートサービスに加えて **ComputeNeutronOvsDpdk** サービスを含む RHOSP インストールで提供されるカスタムロールです。

アンダークラウドのインストールには、コンテナーイメージの取得先およびその保存方法を定義するための環境ファイルが必要です。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **Controller** ロールと **ComputeOvsDpdk** ロールを含む新しいロールデータファイル (例: **roles_data_compute_ovsdpdk.yaml**) を生成します。

```
$ openstack overcloud roles generate \
-o /home/stack/templates/roles_data_compute_ovsdpdk.yaml \
Controller ComputeOvsDpdk
```

注記

RHOSP 環境で OVS-DPDK、SR-IOV、および OVS ハードウェアオフロードなどの複数のテクノロジーを使用している場合は、すべてのロールを含めるために 1 つのロールデータファイルを生成します。

```
$ openstack overcloud roles generate -o /home/stack/templates/\
roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
Compute:ComputeOvsHwOffload
```

4. オプション: TuneD プロファイル **cpu-partitioning-powersave** を使用して、パケットが転送されていないときに OVS-DPDK がスリープモードに入るように設定できます。**cpu-partitioning-powersave** を設定するには、生成されたロールデータファイル内のデフォルトの TuneD プロファイルを省電力 TuneD プロファイル **cpu-partitioning-powersave** に置き換えます。

```
TunedProfileName: "cpu-partitioning-powersave"
```

例

この生成されたロールデータファイル (**/home/stack/templates/roles_data_compute_ovsdpdk.yaml**) では、**TunedProfileName** のデフォルト値が **cpu-partitioning-powersave** に置き換えられます。

```
$ sed -i \
's/TunedProfileName:.*$/TunedProfileName: "cpu-partitioning-powersave"/' \
/home/stack/templates/roles_data_compute_ovsdpdk.yaml
```

5. イメージファイルを生成するには、**openstack tripleo container image prepare** コマンドを実行します。以下の入力が必要です。

   - 前の手順で生成したロールデータファイル (例: **roles_data_compute_ovsdpdk.yaml**)。
   - Networking サービスメカニズムドライバーに適した DPDK 環境ファイル:
     - ML2/OVN 環境用の **neutron-ovn-dpdk.yaml** ファイル。
     - ML2/OVS 環境用の **neutron-ovs-dpdk.yaml** ファイル。

       例

       この例では、**overcloud_images.yaml** ファイルが ML2/OVS 環境用に生成されています。

```
$ sudo openstack tripleo container image prepare \
  --roles-file ~/templates/roles_data_compute_ovsdpdk.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-
dpdk.yaml \
  -e ~/containers-prepare-parameter.yaml \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

6. 作成したロールデータファイルとイメージファイルのパスとファイル名をメモします。これらのファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS-DPDK カスタマイズ用の環境ファイルの作成」 に進みます。

関連情報

- OVS-DPDK デプロイメントにおける電力節約
- **director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール
- **director を使用した Red Hat OpenStack Platform のインストールと管理**で コンテナーイメージを準備 します。

10.3. OVS-DPDK カスタマイズ用の環境ファイルの作成

カスタム環境 YAML ファイル内の特定の Red Hat OpenStack Platform 設定値を使用して、OVS-DPDK デプロイメントを設定できます。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. カスタム環境 YAML ファイル (例: **ovs-dpdk-overrides.yaml**) を作成します。
4. カスタム環境ファイルで、コンピュートサービス (nova) がノードをフィルタリングするために使用する **NovaSchedulerEnabledFilters** パラメーターのフィルターリストに **AggregateInstanceExtraSpecsFilter** が含まれていることを確認します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - AggregateInstanceExtraSpecsFilter
```

5. OVS-DPDK コンピュートノードのロール固有のパラメーターをカスタム環境ファイルに追加します。

例

```
parameter_defaults:
```

```
  ComputeOvsDpdkParameters:
    NeutronBridgeMappings: "dpdk:br-dpdk"
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1GB hugepages=64 iommu=pt
intel_iommu=on
isolcpus=2,4,6,8,10,12,14,16,18,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,23,25,27,2
9,31,33,35,37,39"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList:
"2,4,6,8,10,12,14,16,18,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,23,25,27,29,31,33,
35,37,39"
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "4096,4096"
    OvsDpdkMemoryChannels: "4"
    OvsDpdkCoreList: "0,20,1,21"
    NovaComputeCpuDedicatedSet:
"4,6,8,10,12,14,16,18,24,26,28,30,32,34,36,38,5,7,9,11,13,15,17,19,27,29,31,33,35,37,39"
    NovaComputeCpuSharedSet: "0,20,1,21"
    OvsPmdCoreList: "2,22,3,23"
    OvsEnableDpdk: true
```

6. これらのファイル内の設定デフォルトをオーバーライドする必要がある場合は、手順 3 で作成したカスタム環境ファイルにオーバーライドを追加します。
RHOSP director は、OVS-DPDK を設定するために次のファイルを使用します。

   - ML2/OVN のデプロイメント
/usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-dpdk.yaml
   - ML2/OVS のデプロイメント
/usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovs-dpdk.yaml

7. 作成したカスタム環境ファイルのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「セキュリティーグループのファイアウォールの設定」 に進みます。

10.4. セキュリティーグループのファイアウォールの設定

データプレーンインターフェイスのステートフルファイアウォールには、高いパフォーマンスが要求されます。これらのインターフェイスを保護するためには、Red Hat OpenStack Platform (RHOSP) OVS-DPDK 環境に、仮想ネットワーク機能 (VNF) として通信業界グレードのファイアウォールをデプロイすることを検討してください。

ML2/OVS のデプロイメントでコントロールプレーンのインターフェイスを設定するには、カスタム環境ファイルの **parameter_defaults** で **NeutronOVSFirewallDriver** パラメーターを **openvswitch** に設定します。OVN のデプロイメントでは、アクセスコントロールリスト (ACL) を使用してセキュリティーグループを実装することができます。

オフロードパスではフローの接続追跡プロパティーがサポートされていないため、OVS ファイアウォールドライバーをハードウェアオフロードで使用することはできません。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「OVS-DPDK カスタマイズ用の環境ファイルの作成」 に作成したカスタム環境 YAML ファイルを開くか、新しいものを作成します。
4. **parameter_defaults** の下に、次のキーと値のペアを追加します。

```
parameter_defaults:
...

  NeutronOVSFirewallDriver: openvswitch
```

5. 新しいカスタム環境ファイルを作成した場合は、そのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。
6. オーバークラウドをデプロイした後、**openstack port set** コマンドを実行して、データプレーンインターフェイスの OVS ファイアウォールドライバーを無効にします。

```
$ openstack port set --no-security-group --disable-port-security ${PORT}
```

次のステップ

- 「ベアメタルノード定義ファイルの作成」 に進みます。

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール
- NFV 向けのテスト済み NIC

10.5. ベアメタルノード定義ファイルの作成

Red Hat OpenStack Platform (RHOSP) director を使用して、定義ファイルを使用して OVS-DPDK 環境のベアメタルノードをプロビジョニングします。ベアメタルノード定義ファイルで、デプロイするベアメタルノードの数量と属性を定義し、これらのノードにオーバークラウドロールを割り当てます。ノードのネットワークレイアウトも定義します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。

2. **stackrc** ファイルを取得します。

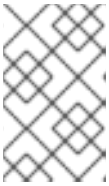
```
$ source ~/stackrc
```

3. **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング の指示に従って、**overcloud-baremetal-deploy.yaml** などのベアメタルノード定義ファイルを作成します。

4. **overcloud-baremetal-deploy.yaml** ファイルで、Ansible Playbook **cli-overcloud-node-kernelargs.yaml** に宣言を追加します。Playbook には、ベアメタルノードをプロビジョニングするときに使用するカーネル引数が含まれています。

```
- name: ComputeOvsDpdk
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. Playbook の実行時に追加の Ansible 変数を設定する場合は、**extra_vars** プロパティーを使用して設定します。
詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードのプロビジョニング属性 を参照してください。



注記

extra_vars に追加する変数は、先ほど OVS-DPDK カスタマイズ用の環境ファイルを作成する でカスタム環境ファイルに追加した OVS-DPDK コンピュートノードのロール固有のパラメーターと同じである必要があります。

例

```
- name: ComputeOvsDpdk
  ...
  ansible_playbooks:
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
    extra_vars:
      kernel_args: 'default_hugepagesz=1GB hugepagesz=1GB hugepages=64 iommu=pt
intel_iommu=on
isolcpus=2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,2
5,27,29,31,33,35,37,39'
      tuned_isolated_cores:
'2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,
31,33,35,37,39'
      tuned_profile: 'cpu-partitioning'
      reboot_wait_timeout: 1800
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-dpdk.yaml
    extra_vars:
      pmd: '2,22,3,23'
      memory_channels: '4'
      socket_mem: '4096,4096'
      pmd_auto_lb: true
      pmd_load_threshold: "70"
      pmd_improvement_threshold: "25"
      pmd_rebal_interval: "2"
```

6. オプション: TuneD プロファイル **cpu-partitioning-powersave** を使用して、パケットが転送されていないときに OVS-DPDK がスリープモードに入るように設定できます。
cpu-partitioning-powersave を設定するには、ベアメタルノード定義ファイルに次の行を追加します。

```
...

tuned_profile: "cpu-partitioning-powersave"

...

  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/cli-overcloud-tuned-maxpower-conf.yaml
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/overcloud-nm-config.yaml
    extra_vars:
      reboot_wait_timeout: 900

...

      pmd_sleep_max: "50"

...
```

例

```
- name: ComputeOvsDpdk
  ...
  ansible_playbooks:
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
    extra_vars:
      kernel_args: default_hugepagesz=1GB hugepagesz=1GB hugepages=64 iommu=pt intel_iommu=on isolcpus=2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39
      tuned_isolated_cores: 2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39
      tuned_profile: cpu-partitioning
      reboot_wait_timeout: 1800
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/cli-overcloud-tuned-maxpower-conf.yaml
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/overcloud-nm-config.yaml
    extra_vars:
      reboot_wait_timeout: 900
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-dpdk.yaml
    extra_vars:
      pmd: 2,22,3,23
      memory_channels: 4
      socket_mem: 4096,4096
      pmd_auto_lb: true
      pmd_load_threshold: "70"
```

```
    pmd_improvement_threshold: "25"
    pmd_rebal_interval: "2"
    pmd_sleep_max: "50"
```

7. 作成したベアメタルノード定義ファイルのパスとファイル名をメモします。このファイルは、後で NIC を設定するときに使用し、ノードをプロビジョニングするときに **overcloud node provision** コマンドの入力ファイルとして使用します。

次のステップ

- 「NIC 設定テンプレートの作成」 に進みます。

関連情報

- OVS-DPDK デプロイメントにおける電力節約
- **director を使用した Red Hat OpenStack Platform のインストールと管理**の コンポーザブルサービスとカスタムロール
- NFV 向けのテスト済み NIC

10.6. NIC 設定テンプレートの作成

Red Hat OpenStack Platform (RHOSP) に同梱されているサンプル Jinja2 テンプレートのコピーを変更して、NIC 設定テンプレートを定義します。

前提条件

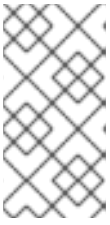
- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. サンプルネットワーク設定テンプレートをコピーします。**/usr/share/ansible/roles/tripleo_network_config/templates/** ディレクトリー内の例から NIC 設定 Jinja2 テンプレートをコピーします。NIC 要件に最も近いものを選択してください。必要に応じて変更してください。
4. NIC 設定テンプレート (例: **single_nic_vlans.j2**) に、DPDK インターフェイスを追加します。



注記

サンプル NIC 設定テンプレート **single_nic_vlans.j2** では、ノードが VLAN を持つトランクとして 1 つのネットワークインターフェイスのみを使用します。ネイティブ VLAN (タグなしトラフィック) はコントロールプレーンであり、各 VLAN は RHOSP ネットワーク (内部 API、ストレージなど) の 1 つに対応します。

例

```
...
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    rx_queue: 1
    ovs_extra:
      - set Interface dpdk0 options:n_rxq_desc=4096
      - set Interface dpdk0 options:n_txq_desc=4096
      - set Interface dpdk1 options:n_rxq_desc=4096
      - set Interface dpdk1 options:n_txq_desc=4096
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      driver: vfio-pci
      members:
      - type: interface
        name: nic5
    - type: ovs_dpdk_port
      name: dpdk1
      driver: vfio-pci
      members:
      - type: interface
        name: nic6
...
```

5. カスタムネットワーク設定テンプレート (例: **single_nic_vlans.j2**) を、「ベアメタルノード定義ファイルの作成」 で作成したベアメタルノード定義ファイル (例: **overcloud-baremetal-deploy.yaml**) に追加します。

   例

```
- name: ComputeOvsDpdk
  count: 2
  hostname_format: compute-%index%
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. オプション: TuneD プロファイル **cpu-partitioning-powersave** を使用して、パケットが転送されていないときに OVS-DPDK がスリープモードに入るように設定できます。
   cpu-partitioning-powersave を設定するには、NIC 設定テンプレートでキューサイズが設定されていることを確認してください。

   例

```
...
  - type: ovs_dpdk_bond
```

```
    name: dpdkbond0
    mtu: 9000
    rx_queue: 1
    ovs_extra:
      - set Interface dpdk0 options:n_rxq_desc=4096
      - set Interface dpdk0 options:n_txq_desc=4096
      - set Interface dpdk1 options:n_rxq_desc=4096
      - set Interface dpdk1 options:n_txq_desc=4096
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      driver: vfio-pci
      members:
      - type: interface
        name: nic5
    - type: ovs_dpdk_port
      name: dpdk1
      driver: vfio-pci
      members:
      - type: interface
        name: nic6
...
```

7. 作成した NIC 設定テンプレートのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS-DPDK インターフェイスの MTU 値の設定」 に進みます。

関連情報

- OVS-DPDK デプロイメントにおける電力節約

10.7. OVS-DPDK インターフェイスの MTU 値の設定

Red Hat OpenStack Platform (RHOSP) は、OVS-DPDK の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最大伝送単位 (MTU) 値を設定するには、以下の操作を行う必要があります。

- カスタム環境ファイルでネットワークのグローバル MTU 値を設定します。
- NIC 設定テンプレートで物理 DPDK ポート MTU 値を設定します。
  この値は、vhost のユーザーインターフェイスでも使用されます。
- Compute 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する。

NIC は DPDK PMD によって制御され、NIC 設定テンプレートによって設定された MTU 値と同じであるため、物理 NIC に対して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。

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。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

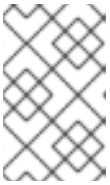
```
$ source ~/stackrc
```

3. 「OVS-DPDK カスタマイズ用の環境ファイルの作成」 に作成したカスタム環境 YAML ファイルを開くか、新しいものを作成します。
4. **parameter_defaults** の下で **NeutronGlobalPhysnetMtu** パラメーターを設定します。

   例

   この例では、**NeutronGlobalPhysnetMtu** は **9000** に設定されています。

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



注記

network-environment.yaml ファイルの **OvsDpdkSocketMemory** 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。詳細は、メモリーパラメーター を参照してください。

5. 「NIC 設定テンプレートの作成」 で作成した NIC 設定テンプレート (例: **single_nic_vlans.j2**) を開きます。
6. ブリッジ上の MTU 値をコンピュートノードに設定します。

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

7. OVS-DPDK ボンディングの 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. NIC 設定テンプレートとカスタム環境ファイルのパスとファイル名をメモします。これらのファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「OVS-DPDK インターフェイス向けのマルチキューの設定」 に進みます。

10.8. OVS-DPDK インターフェイス向けのマルチキューの設定

OVS-DPDK のデプロイメントを設定して、負荷およびキューの使用に基づいて、非分離ポーリングモードドライバー (PMD) へのキューを自動的に負荷分散することができます。Open vSwitch では、以下のシナリオにおいてキューの自動リバランスをトリガーすることができます。

- **pmd-auto-lb** の値を **true** に設定することで、RX キューのサイクルベースの割り当てを有効にしました。
- 2 つまたはそれ以上の非分離 PMD が存在する。
- 少なくとも 1 つの非分離 PMD について、複数のキューがポーリングする。
- 合計 PMD の負荷値が 1 分間で 95% を超える。

重要

マルチキューは実験的な機能で、手動によるキューの固定でのみサポートされます。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「NIC 設定テンプレートの作成」 で作成した NIC 設定テンプレート (例: **single_nic_vlans.j2**) を開きます。
4. コンピュートノード上の OVS-DPDK のインターフェイスのキューの数を設定します。

```
- 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
```

5. NIC 設定テンプレートのパスとファイル名をメモします。このファイルは、後でオーバークラウドをデプロイするときに使用します。

次のステップ

- 「ノードプロビジョニング用の DPDK パラメーターの設定」 に進みます。

10.9. ノードプロビジョニング用の DPDK パラメーターの設定

Red Hat OpenStack Platform (RHOSP) OVS-DPDK 環境を設定して、Open vSwitch (OVS) ポールモードドライバー (PMD) スレッドの負荷を自動的に分散することができます。これを行うには、ベアメタルノードのプロビジョニング中およびオーバークラウドのデプロイメント中に RHOSP director が使用するパラメーターを編集します。

OVS PMD スレッドは、ユーザー空間のコンテキスト切り替えのために次のタスクを実行します。

- 入力ポートを連続的にポーリングしてパケットを取得する。
- 受信したパケットを分類する。
- 分類後のパケットに対してアクションを実行する。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 「ベアメタルノード定義ファイルの作成」 に作成したベアメタルノード定義ファイルにパラメーターを設定します (例: **overcloud-baremetal-deploy**.yaml)。

pmd_auto_lb

PMD 自動ロードバランシングを有効にするには、**true** に設定します。

pmd_load_threshold

PMD ロードバランスをトリガーする前に、1 つの PMD スレッドで一貫して使用する必要がある処理サイクルの割合。0 - 100 の範囲の整数。

pmd_improvement_threshold

PMD 自動ロードバランシングをトリガーする、非分離 PMD スレッド全体で評価された改善の最小パーセンテージ。0 - 100 の範囲の整数。
推定される改善を計算するために、再割り当てのドライランを実行し、推定不可分散と現在の分散を比較します。デフォルトは 25% です。

pmd_rebal_interval

2 つの連続する PMD 自動ロードバランシングが実施される間の最小時間 (分単位)。範囲は 0 - 20,000 分です。
トラフィックパターンが変更可能な場合に、再割り当てが頻繁に発生しないようにするには、この値を設定します。たとえば、10 分に 1 回、または数時間に 1 回、再割り当てをトリガーできます。

例

```
ansible_playbooks:
…
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-dpdk.yaml
  extra_vars:
    …
    pmd_auto_lb: true
    pmd_load_threshold: "70"
    pmd_improvement_threshold: "25"
    pmd_rebal_interval: "2"
```

4. 「OVS-DPDK カスタマイズ用の環境ファイルの作成」 に作成したカスタム環境 YAML ファイルを開くか、新しいものを作成します。
5. カスタム環境ファイルに、手順 3 で設定したのと同じベアメタルノードの事前プロビジョニング値を追加します。次の同等のパラメーターを使用します。

OvsPmdAutoLb

pmd_auto_lb と同等の heat。
PMD 自動ロードバランシングを有効にするには、**true** に設定します。

OvsPmdLoadThreshold

pmd_load_threshold と同等の heat。
PMD ロードバランスをトリガーする前に、1 つの PMD スレッドで一貫して使用する必要がある処理サイクルの割合。0 - 100 の範囲の整数。

OvsPmdImprovementThreshold

pmd_improvement_threshold と同等の heat。
PMD 自動ロードバランシングをトリガーする、非分離 PMD スレッド全体で評価された改善の最小パーセンテージ。0 - 100 の範囲の整数。

推定される改善を計算するために、再割り当てのドライランを実行し、推定不可分散と現在の分散を比較します。デフォルトは 25% です。

OvsPmdRebalInterval

pmd_rebal_interval と同等の heat。
2 つの連続する PMD 自動ロードバランシングが実施される間の最小時間 (分単位)。範囲は 0 - 20,000 分です。

トラフィックパターンが変更可能な場合に、再割り当てが頻繁に発生しないようにするには、この値を設定します。たとえば、10 分に 1 回、または数時間に 1 回、再割り当てをトリガーできます。

例

```
parameter_merge_strategies:
  ComputeOvsDpdkSriovParameters:merge
…
parameter_defaults:
  ComputeOvsDpdkSriovParameters:
    …
    OvsPmdAutoLb: true
    OvsPmdLoadThreshold: 70
    OvsPmdImprovementThreshold: 25
    OvsPmdRebalInterval: 2
```

6. NIC 設定テンプレートとカスタム環境ファイルのパスとファイル名をメモします。これらのファイルは、後でベアメタルノードをプロビジョニングし、オーバークラウドをデプロイするときに使用します。

次のステップ

1. ネットワークと仮想 IP をプロビジョニングします。
2. ベアメタルノードをプロビジョニングする。
provision コマンドを実行するための入力として、**overcloud-baremetal-deploy.yaml** などのベアメタルノード定義ファイルを使用するようにしてください。
3. 「OVS-DPDK オーバークラウドのデプロイ」 に進みます。

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング
- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング
- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング

10.10. OVS-DPDK オーバークラウドのデプロイ

OVS-DPDK 環境で Red Hat OpenStack Platform (RHOSP) オーバークラウドをデプロイする最後の手順は、**openstack overcloud deploy** コマンドを実行することです。このコマンドに、作成したさまざまなオーバークラウドテンプレートと環境ファイルをすべて入力します。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。
- このセクションの前の手順にリストされているすべてのステップを実行し、**overcloud deploy** コマンドの入力として使用するさまざまな heat テンプレートおよび環境ファイルをすべてアセンブルしました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. **openstack overcloud deploy** コマンドを実行します。
openstack overcloud deploy コマンドへの入力を特定の順序でリストすることが重要です。一般的なルールは、デフォルトの heat テンプレートファイルを最初に指定し、次にカスタム環境ファイルと、デフォルトプロパティーのオーバーライドなどのカスタム設定を含むカスタムテンプレートを指定することです。

   次の順序で、**openstack overcloud deploy** コマンドに入力を追加します。

   a. オーバークラウド上の SR-IOV ネットワークの仕様が含まれるカスタムネットワーク定義ファイル (例: **network-data.yaml**)。
   詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ネットワーク定義ファイル設定のオプション を参照してください。

   b. RHOSP director が SR-IOV 環境をデプロイするために使用する **Controller** および **ComputeOvsDpdk** ロールを含むロールファイル。
   例: **roles_data_compute_ovsdpdk.yaml**

   詳細は、「ロールとイメージファイルの生成」 を参照してください。

   c. オーバークラウドネットワークのプロビジョニングからの出力ファイル。
   例: **overcloud-networks-deployed.yaml**

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク定義の設定とプロビジョニング を参照してください。

d. オーバークラウド仮想 IP のプロビジョニングからの出力ファイル。
例: **overcloud-vip-deployed.yaml**

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのネットワーク仮想 IP の設定とプロビジョニング を参照してください。

e. ベアメタルノードのプロビジョニングからの出力ファイル。
たとえば、**overcloud-baremetal-deployed.yaml** です。

詳細は以下を参照してください。

- 「ノードプロビジョニング用の DPDK パラメーターの設定」。
- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのベアメタルノードのプロビジョニング

f. コンテナーイメージを取得する場所と保存方法を決定するためにディレクターが使用するイメージファイル。
例: **overcloud_images.yaml**

詳細は、「ロールとイメージファイルの生成」 を参照してください。

g. 環境が使用する Networking サービス (neutron) メカニズムドライバーとルータースキームの環境ファイル:

- ML2/OVN
  - 分散仮想ルーティング (DVR): **neutron-ovn-dvr-ha.yaml**
  - 集中型仮想ルーティング: **neutron-ovn-ha.yaml**
- ML2/OVS
  - 分散仮想ルーティング (DVR): **neutron-ovs-dvr.yaml**
  - 集中型仮想ルーティング: **neutron-ovs.yaml**

h. メカニズムドライバーに応じた OVS-DPDK の環境ファイル:

- ML2/OVN
  - **neutron-ovn-dpdk.yaml**
- ML2/OVS
  - **neutron-ovs-dpdk.yaml**

注記

SR-IOV 環境もあり、SR-IOV インスタンスと OVS-DPDK インスタンスを同じノードに配置する場合は、デプロイメントスクリプトに次の環境ファイルを含めます。

- ML2/OVN
  neutron-ovn-sriov.yaml
- ML2/OVS
  neutron-sriov.yaml

i. 以下の設定を含む 1 つ以上のカスタム環境ファイル:

- OVS-DPDK 環境のデフォルト設定値を上書きします。
- 仮想ネットワーク機能 (VNF) としてのファイアウォール。
- ジャンボフレームの最大転送単位 (MTU) 値。
  例: **ovs-dpdk-overrides.yaml**

  詳細は以下を参照してください。
- 「OVS-DPDK カスタマイズ用の環境ファイルの作成」。
- 「セキュリティーグループのファイアウォールの設定」。
- 「OVS-DPDK インターフェイスの MTU 値の設定」。

例

サンプルの **openstack overcloud deploy** コマンドからのこの抜粋は、DVR を使用する OVS-DPDK、ML2/OVN 環境のコマンド入力の適切な順序を示しています。

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_ovsdpdk.yaml \
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dpdk.yaml \
-e /home/stack/templates/ovs-dpdk-overrides.yaml
```

4. **openstack overcloud deploy** コマンドを実行します。
   オーバークラウドの作成が完了すると、RHOSP director は、オーバークラウドへのアクセスに役立つ詳細を提供します。

検証

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドのデプロイメントの検証 のステップを実行します。

次のステップ

- ファイアウォールを設定している場合は、**openstack port set** コマンドを実行して、データプレーンインターフェイスの OVS ファイアウォールドライバーを無効にします。

```
$ openstack port set --no-security-group --disable-port-security ${PORT}
```

関連情報

- **director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの オーバークラウドの作成
- **コマンドラインインターフェイスリファレンス** の overcloud deploy

10.11. 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 デプロイメント向けに OVS-DPDK を設定したら、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. OVS-DPDK 用のアグリゲートグループを作成し、適切なホストを追加します。定義するフレーバーメタデータに一致するメタデータを定義します (例: **dpdk=true**)。

```
# openstack aggregate create dpdk_group
# openstack aggregate add host dpdk_group [compute-host]
# openstack aggregate set --property dpdk=true dpdk_group
```

注記

CPU ピニングを設定したインスタンスと設定していないインスタンスを、同じ Compute ノードに配置することができます。詳細は、**インスタンス作成のための Compute サービスの設定** の Compute ノードでの CPU ピニングの設定 を参照してください。

2. フレーバーを作成します。

```
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

3. フレーバーの属性を設定します。定義したメタデータ (**dpdk=true**) と DPDK アグリゲートで定義したメタデータが一致している点に注意してください。

```
# openstack flavor set <flavor> --property dpdk=true --property hw:cpu_policy=dedicated --
property hw:mem_page_size=1GB --property hw:emulator_threads_policy=isolate
```

パフォーマンス向上のためのエミュレータースレッドポリシーの詳細は、**インスタンス作成のための Compute サービスの設定** の エミュレータースレッドの設定 を参照してください。

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>
# openstack subnet create subnet1 --network net1 --subnet-range 192.0.2.0/24 --dhcp
```

5. オプション: OVS-DPDK と共にマルチキューを使用する場合、インスタンスの作成に使用するイメージで **hw_vif_multiqueue_enabled** 属性を設定します。

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

6. インスタンスをデプロイします。

```
# openstack server create --flavor <flavor> --image <glance image> --nic net-id=<network
ID> <server_name>
```

10.12. OVS-DPDK 設定のトラブルシューティング

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

1. ブリッジの詳細を調べ、**datapath_type=netdev** の設定を確認します。

```
# 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. オプションとして、コンテナーが起動に失敗したかどうかなど、エラーをログで確認することができます。

```
# less /var/log/containers/neutron/openvswitch-agent.log
```

3. **ovs-dpdk** の Poll Mode Driver CPU マスクが CPU にピニングされていることを確認します。ハイパースレッディングの場合は、シブリング CPU を使用します。
たとえば、**CPU4** のシブリングを確認するには、以下のコマンドを実行します。

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

CPU4 のシブリングは **CPU20** なので、続いて以下のコマンドを実行します。

```
# 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11章 RED HAT OPENSTACK PLATFORM 環境の調整

11.1. エミュレータースレッドの固定

エミュレータースレッドは、仮想マシンのハードウェアエミュレーションの割り込み要求およびノンブロッキングプロセスを処理します。これらのスレッドは、ゲストが処理用に使用する CPU 全体に存在します。Poll Mode Driver (PMD) またはリアルタイム処理に使用されるスレッドがこれらのゲスト CPU 上で実行される場合、パケットロスまたはデッドラインの超過が生じる可能性があります。

エミュレータースレッドを専用のゲスト CPU に固定して、スレッドを仮想マシン処理のタスクから分離することができます。その結果、パフォーマンスが向上します。

パフォーマンスを向上させるには、エミュレータースレッドをホストするためにホスト CPU のサブセットを確保します。

手順

1. 特定のロールに **NovaComputeCpuSharedSet** を定義してオーバークラウドをデプロイします。**NovaComputeCpuSharedSet** の値は、そのロール内のホストの **nova.conf** ファイルの **cpu_shared_set** パラメーターに適用されます。

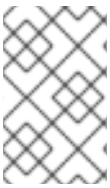
```
parameter_defaults:
  ComputeOvsDpdkParameters:
    NovaComputeCpuSharedSet: "0-1,16-17"
    NovaComputeCpuDedicatedSet: "2-15,18-31"
```

2. エミュレータースレッドが共有プールに分離されたインスタンスをビルドするためのフレーバーを作成します。

```
openstack flavor create --ram <size_mb> --disk <size_gb> --vcpus <vcpus> <flavor>
```

3. **hw:emulator_threads_policy** 追加仕様を追加し、値を **share** に設定します。このフレーバーで作成されたインスタンスは、nova.conf ファイルの **cpu_share_set** パラメーターで定義されたインスタンス CPU を使用します。

```
openstack flavor set <flavor> --property hw:emulator_threads_policy=share
```



注記

この追加仕様の共有ポリシーを有効にするには、**nova.conf** ファイルで **cpu_share_set** パラメーターを設定する必要があります。**nova.conf** を手動で編集した内容は再デプロイ後は維持されないので、この設定には可能な限り heat を使用するべきです。

検証

1. 対象インスタンスのホストおよび名前を特定します。

```
openstack server show <instance_id>
```

2. SSH を使用して、識別されたホストに tripleo-admin としてログオンします。

```
ssh tripleo-admin@compute-1
[compute-1]$ sudo virsh dumpxml instance-00001 | grep `'emulatorpin cpuset'`
```

11.2. VIRTUAL FUNCTION と PHYSICAL FUNCTION 間の信頼の設定

Virtual Function (VF) がプロミスキャスモードの有効化やハードウェアアドレスの変更などの特権を必要とする操作を実施できるように、Physical Function (PF) と VF 間に信頼を設定することができます。

前提条件

- 稼働状態にある Red Hat OpenStack Platform のインストール環境 (director を含む)

手順

Physical Function と Virtual Function 間の信頼が設定されたオーバークラウドを設定およびデプロイするには、以下の手順を実施します。

1. **parameter_defaults** セクションに **NeutronPhysicalDevMappings** パラメーターを追加して、論理ネットワーク名と物理インターフェイス間をリンクさせます。

```
parameter_defaults:
  NeutronPhysicalDevMappings:
    - sriov2:p5p2
```

2. SR-IOV パラメーターに新たな属性 **trusted** を追加します。

```
parameter_defaults:
  NeutronPhysicalDevMappings:
    - sriov2:p5p2
  NovaPCIPassthrough:
    - vendor_id: "8086"
      product_id: "1572"
      physical_network: "sriov2"
      trusted: "true"
```

注記

"true" のように、値を二重引用符で囲む必要があります。

11.3. 信頼済み VF ネットワークの活用

1. 種別 **vlan** のネットワークを作成します。

```
openstack network create trusted_vf_network  --provider-network-type vlan \
 --provider-segment 111 --provider-physical-network sriov2 \
 --external --disable-port-security
```

2. サブネットを作成します。

```
openstack subnet create --network trusted_vf_network \
  --ip-version 4 --subnet-range 192.168.111.0/24 --no-dhcp \
 subnet-trusted_vf_network
```

3. ポートを作成します。**vnic-type** オプションを **direct** に、**binding-profile** オプションを **true** に、それぞれ設定します。

```
openstack port create --network sriov111 \
--vnic-type direct --binding-profile trusted=true \
sriov111_port_trusted
```

4. インスタンスを作成し、それを前のステップで作成した信頼済みポートにバインドします。

```
openstack server create --image rhel --flavor dpdk  --network internal --port
trusted_vf_network_port_trusted --config-drive True --wait rhel-dpdk-sriov_trusted
```

検証

ハイパーバイザー上での信頼済み VF 設定の確認

1. インスタンスを作成した Compute ノード上で、以下のコマンドを入力します。

```
# ip link
7: p5p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:1c:40:fa brd ff:ff:ff:ff:ff:ff
    vf 6 MAC fa:16:3e:b8:91:c2, vlan 111, spoof checking off, link-state auto, trust on,
query_rss off
    vf 7 MAC fa:16:3e:84:cf:c8, vlan 111, spoof checking off, link-state auto, trust off, query_rss
off
```

2. VF の信頼ステータスが **trust on** であることを確認します。上記の出力例には、2 つのポートが含まれる環境の詳細が示されています。**vf 6** に **trust on** のテキストが含まれている点に注意してください。

3. Networking サービス (neutron) ネットワークで **port_security_enabled: false** を設定した場合、あるいは **openstack port create** コマンドの実行時に引数 **--disable-port-security** を含める場合には、スプーフィングの確認を無効にできます。

11.4. RX-TX キューサイズ管理によるパケット損失の防止

以下に示す理由により、3.5 百万パケット毎秒 (mpps) を超える高いパケットレートでは、パケットロスが生じる場合があります。

- ネットワークの中断
- SMI
- 仮想ネットワーク機能におけるパケット処理のレイテンシー

パケットロスを防ぐには、キューサイズをデフォルトの 512 から最大の 1024 に増やします。

前提条件

- アンダークラウドホストへのアクセスと **stack** ユーザーの認証情報。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。

2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. カスタム環境 YAML ファイルを作成し、**parameter_defaults** の下に次の定義を追加して、RX および TX キューのサイズを増やします。

```
parameter_defaults:
  NovaLibvirtRxQueueSize: 1024
  NovaLibvirtTxQueueSize: 1024
```

4. デプロイコマンドを実行します。コア Heat テンプレート、その他の環境ファイル、RX および TX キューサイズの変更を含む環境ファイルをコマンドに追加します。

例

```
$ openstack overcloud deploy --templates \
-e <other_environment_files> \
-e /home/stack/my_tx-rx_queue_sizes.yaml
```

検証

1. **nova.conf** ファイル内の RX キューサイズと TX キューサイズの値を確認します。

```
$ egrep "^[rt]x_queue_size" /var/lib/config-data/puppet-generated/\
nova_libvirt/etc/nova/nova.conf
```

以下のメッセージが表示されるはずです。

```
rx_queue_size=1024
tx_queue_size=1024
```

2. Compute ホストの libvirt により生成された仮想マシンインスタンスの XML ファイルで、RX キューサイズと TX キューサイズの値を確認します。

   a. 新規インスタンスを作成します。

   b. Compute ホストとインスタンス名を取得します。

```
$ openstack server show testvm-queue-sizes -c OS-EXT-SRV-ATTR:\
hypervisor_hostname -c OS-EXT-SRV-ATTR:instance_name
```

出力例

以下のような出力が表示されるはずです。

```
+-------------------------------------+----------------------------------+
| Field                               | Value                            |
+-------------------------------------+----------------------------------+
| OS-EXT-SRV-ATTR:hypervisor_hostname | overcloud-novacompute-1.sales    |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000059                |
+-------------------------------------+----------------------------------+
```

c. Compute ホストにログインし、インスタンス定義をダンプします。

例

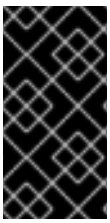
```
$ podman exec nova_libvirt virsh dumpxml instance-00000059
```

出力例

以下のような出力が表示されるはずです。

```
...
  <interface type='vhostuser'>
    <mac address='56:48:4f:4d:5e:6f'/>
    <source type='unix' path='/tmp/vhost-user1' mode='server'/>
    <model type='virtio'/>
    <driver name='vhost' rx_queue_size='1024'   tx_queue_size='1024' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x10' function='0x0'/>
  </interface>
...
```

11.5. NUMA 対応 VSWITCH の設定



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、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能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。

NUMA 対応 vSwitch を実装するには、ご自分のハードウェア設定の以下のコンポーネントを確認してください。

- 物理ネットワークの数
- PCI カードの配置
- サーバーの物理アーキテクチャー

PCIe NIC 等のメモリーマップト I/O (MMIO) デバイスは、特定の NUMA ノードに関連付けられます。仮想マシンと NIC が異なる NUMA ノードにあると、パフォーマンスが大幅に低下します。パフォーマンスを向上させるためには、PCIe NIC の配置とインスタンスの処理を同じ NUMA ノードに一致させます。

この機能を使用して、物理ネットワークを共有するインスタンスが同じ NUMA ノードに配置されるようにします。データセンターのハードウェア使用率を最適化するには、複数の物理ネットワークを使用する必要があります。

警告

サーバー使用率を最適化するために NUMA 対応ネットワークを設定するには、PCIe スロットと NUMA ノードのマッピングを把握する必要があります。お使いの特定ハードウェアの詳細情報は、ベンダーのドキュメントを参照してください。NUMA 対応 vSwitch の正しいプランニングまたは実装に失敗する場合は、サーバーが 1 つの NUMA ノードだけを使用するように設定することができます。

複数 NUMA にまたがる設定を防ぐためには、NIC の場所を Nova に提供して、仮想マシンを正しい NUMA ノードに配置します。

前提条件

- フィルター **NUMATopologyFilter** を有効にしている。

手順

1. 新たに **NeutronPhysnetNUMANodesMapping** パラメーターを設定して、物理ネットワークと物理ネットワークに関連付ける NUMA ノードをマッピングします。
2. VxLAN や GRE 等のトンネルを使用する場合には、**NeutronTunnelNUMANodes** パラメーターも設定する必要があります。

```
parameter_defaults:
  NeutronPhysnetNUMANodesMapping: {<physnet_name>: [<NUMA_NODE>]}
  NeutronTunnelNUMANodes: <NUMA_NODE>,<NUMA_NODE>
```

例

2 つの物理ネットワークを NUMA ノード 0 にトンネリングする例を以下に示します。

- NUMA ノード 0 に関連付けられた 1 つのプロジェクトネットワーク
- アフィニティーが設定されていない 1 つの管理ネットワーク

```
parameter_defaults:
  NeutronBridgeMappings:
    - tenant:br-link0
  NeutronPhysnetNUMANodesMapping: {tenant: [1], mgmt: [0,1]}
  NeutronTunnelNUMANodes: 0
```

- この例では、**eno2** という名前のデバイスの物理ネットワークを NUMA 番号 0 に割り当てます。

```
# ethtool -i eno2
bus-info: 0000:18:00.1

# cat /sys/devices/pci0000:16/0000:16:02.0/0000:18:00.1/numa_node
0
```

heat テンプレートのサンプルで物理ネットワークの設定を確認します。

```
NeutronBridgeMappings: 'physnet1:br-physnet1'
NeutronPhysnetNUMANodesMapping: {physnet1: [0] }

- type: ovs_user_bridge
                name: br-physnet1
                mtu: 9000
                members:
                  - type: ovs_dpdk_port
                    name: dpdk2
                    members:
                      - type: interface
                        name: eno2
```

検証

NUMA 対応 vSwitch をテストするには、以下の手順に従います。

1. **/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf** ファイルの設定を確認します。

```
[neutron_physnet_tenant]
numa_nodes=1
[neutron_tunnel]
numa_nodes=1
```

2. **lscpu** コマンドで新たな設定を確認します。

```
$ lscpu
```

3. NIC が適切なネットワークに接続された仮想マシンを起動します。

関連情報

- NUMA ノードのトポロジーについての理解
- 「NUMA 対応 vSwitch の既知の制限」

11.6. NUMA 対応 VSWITCH の既知の制限

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、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能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。

このセクションでは、Red Hat OpenStack Platform (RHOSP) ネットワーク機能仮想化インフラストラクチャー (NFVi) に NUMA 対応の vSwitch を実装する際の制約について記載しています。

- 2 ノードのゲスト NUMA トポロジーを指定しなかった場合、2 つの NIC が異なる NUMA ノード上の物理ネットワークに接続された仮想マシンを起動することはできません。

- 2 ノードのゲスト NUMA トポロジーを指定しなかった場合、1 つの NIC が物理ネットワークに接続され、別の NIC が異なる NUMA ノード上のトンネル化ネットワークに接続された仮想マシンを起動することはできません。
- 2 ノードのゲスト NUMA トポロジーを指定しなかった場合、異なる NUMA ノード上にある 1 つの仮想ホストポートおよび 1 つの Virtual Function を持つ仮想マシンを起動することはできません。
- NUMA 対応 vSwitch のパラメーターは、オーバークラウドロールごとに固有です。たとえば、Compute ノード 1 と Compute ノード 2 に、異なる NUMA トポロジーを設定することができます。
- 仮想マシンのインターフェイスに NUMA アフィニティーを設定する場合は、アフィニティーが単一の NUMA ノードだけを対象にするようにします。NUMA アフィニティーが設定されないインターフェイスは、任意の NUMA ノードに配置することができます。
- 管理ネットワークではなく、データプレーンネットワークに NUMA アフィニティーを設定します。
- トンネル化ネットワークの NUMA アフィニティーは、すべての仮想マシンに適用されるグローバルの設定です。

11.7. NFVI 環境における QUALITY OF SERVICE (QOS)

Quality of Service (QoS) を使用して、ネットワーク機能仮想化インフラストラクチャー (NFVi) 内にある Red Hat OpenStack Platform (RHOSP) ネットワークの egress および ingress トラフィックにレート制限を適用することで、VM インスタンスにさまざまなサービスレベルを提供できます。

NFVi 環境では、QoS のサポートは以下のルール種別に制限されます。

- SR-IOV での **minimum bandwidth** (ベンダーによりサポートされる場合)
- SR-IOV および OVS-DPDK 送信インターフェイスでの **bandwidth limit**

関連情報

- Quality of Service (QoS) ポリシーの設定

11.8. DPDK を使用する HCI オーバークラウドの作成

ハイパーコンバージドノードと共に NFV インフラストラクチャーをデプロイするには、リソースの使用率を最適化するために Compute サービスと Ceph Storage サービスを共存させて設定します。

ハイパーコンバージドインフラストラクチャー (HCI) の詳細は、**ハイパーコンバージドインフラストラクチャーのデプロイ** を参照してください。

次のセクションでは、さまざまな設定の例を説明します。

11.8.1. NUMA ノード設定の例

パフォーマンスを向上させるために、テナントネットワークおよび Ceph オブジェクトサービスデーモン (OSD) を 1 つの NUMA ノード (例: NUMA-0) に配置し、VNF および NFV 以外の仮想マシンを別の NUMA ノード (例: NUMA-1) に配置します。

CPU の割り当て

NUMA-0	NUMA-1
Ceph OSD 数 * 4 HT	VNF および NFV 以外の仮想マシン用のゲスト仮想 CPU
DPDK lcore - 2 HT	DPDK lcore - 2 HT
DPDK PMD - 2 HT	DPDK PMD - 2 HT

CPU 割り当ての例

	NUMA-0	NUMA-1
Ceph OSD	32,34,36,38,40,42,76,78,80,82,84,86	
DPDK-lcore	0,44	1,45
DPDK-pmd	2,46	3,47
nova		5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87

11.8.2. ceph 設定ファイルの例

このセクションでは、Red Hat Ceph Storage 設定ファイルのサンプルを説明します。Red Hat OpenStack Platform 環境に適した値を代入することで、これを基に設定ファイルをモデル化できます。

```
[osd]
osd_numa_node = 0                    # 1
osd_memory_target_autotune = true        # 2

[mgr]
mgr/cephadm/autotune_memory_target_ratio = 0.2 # 3
```

次のパラメーターを使用して、Ceph Object Storage Daemon (OSD) プロセスに CPU リソースを割り当てます。ここで示されている値は例です。ワークロードとハードウェアに応じて値を適切に調整します。

1. **osd_numa_node**: Ceph プロセスの NUMA ノードへのアフィニティーを設定します (たとえば、**NUMA-0** の場合は **0**、**NUMA-1** の場合は **1** など)。**-1** は、アフィニティーを NUMA ノードなしに設定します。

   この例では、**osd_numa_node** は **NUMA-0** に設定されています。「DPDK 設定ファイルの例」 に示すように、**OvsPmdCoreList** の要素が削除された後、**IsolCpusList** には **NUMA-1** 上の奇数番号の CPU が含まれます。レイテンシーの影響を受けやすいコンピュートサービス (nova) ワーク

ロードは **NUMA-1** でホストされているため、Ceph ワークロードを **NUMA-0** で分離する必要があります。この例では、ストレージネットワークのディスクコントローラーとネットワークインターフェイスの両方が **NUMA-0** 上にあることを前提としています。

2 **osd_memory_target_autotune**: true に設定すると、OSD デーモンは **osd_memory_target** 設定オプションに基づいてメモリー消費を調整します。

3 **autotune_memory_target_ratio**: OSD のメモリーを割り当てるために使用されます。デフォルトは **0.7** です。

システム内の合計 RAM の 70% が開始点となり、自動調整されていない Ceph デーモンによって消費されるメモリーがここから差し引かれます。すべての OSD に対して **osd_memory_target_autotune** が true の場合は、残りのメモリーが OSD ごとに分割されます。HCI デプロイメントの場合は、**mgr/cephadm/autotune_memory_target_ratio** を **0.2** に設定して、コンピュートサービスに使用できるメモリーを増やすことができます。必要に応じて調整し、各 OSD に少なくとも 5 GB のメモリーがあることを確認します。

関連情報

- 「HCI-DPDK オーバークラウドのデプロイ」

11.8.3. DPDK 設定ファイルの例

```
parameter_defaults:
  ComputeHCIParameters:
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=240 intel_iommu=on
iommu=pt                                  # 1

isolcpus=2,46,3,47,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,53,55,57,59,61,63,
65,67,69,71,73,75,77,79,81,83,85,87"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList:                                 # 2
      "2,46,3,47,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,
      53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87"
    VhostuserSocketGroup: hugetlbfs
    OvsDpdkSocketMemory: "4096,4096"                    # 3
    OvsDpdkMemoryChannels: "4"

    OvsPmdCoreList: "2,46,3,47"                       # 4
```

1 KernelArgs: **hugepages** を算出するには、合計メモリーから **NovaReservedHostMemory** パラメーターの値を減算します。

2 IsolCpusList: このパラメーターを使用して、ホストプロセスから分離する CPU コアのセットを割り当てます。**IsolCpusList** パラメーターの値を算出するには、**NovaComputeCpuDedicatedSet** パラメーターの値に **OvsPmdCoreList** パラメーターの値を加えます。

3 OvsDpdkSocketMemory: **OvsDpdkSocketMemory** パラメーターを使用して、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容量を指定します (MB 単位)。OVS-DPDK パラメーターの計算に関する詳細は、OVS-DPDK パラメーター を参照してください。

4 OvsPmdCoreList: このパラメーターを使用して、DPDK Poll Mode Driver (PMD) に使用される CPU コアを指定します。DPDK インターフェイスのローカルの NUMA ノードに関連付けられた CPU コアを選択します。**OvsPmdCoreList** パラメーターの値を算出するには、NUMA ノードごと

に 2 つの HT シブリングスレッドを割り当てます。

11.8.4. nova 設定ファイルの例

```
parameter_defaults:
  ComputeHCIExtraConfig:
    nova::cpu_allocation_ratio: 16 # 2
    NovaReservedHugePages:                                  # 1
      - node:0,size:1GB,count:4
      - node:1,size:1GB,count:4
  NovaReservedHostMemory: 123904                            # 2
  # All left over cpus from NUMA-1
  NovaComputeCpuDedicatedSet:                                  # 3
['5','7','9','11','13','15','17','19','21','23','25','27','29','31','33','35','37','39','41','43','49','51','|
53','55','57','59','61','63','65','67','69','71','73','75','77','79','81','83','85','87
```

1. NovaReservedHugePages: **NovaReservedHugePages** パラメーターを使用して、ヒュージページプールからメモリーを事前に割り当てます (MB 単位)。これは、**OvsDpdkSocketMemory** パラメーターの値と同じ合計メモリーです。

2. NovaReservedHostMemory: **NovaReservedHostMemory** パラメーターを使用して、ホスト上のタスク用にメモリーを確保します (MB 単位)。確保しなければならないメモリー容量を算出するには、以下のガイドラインを使用します。

   - OSD ごとに 5 GB
   - 仮想マシンごとに 0.5 GB のオーバーヘッド
   - 一般的なホストプロセス用に 4 GB。複数 NUMA にまたがる OSD 操作によって生じるパフォーマンスの低下を防ぐために、十分なメモリーを割り当てるようにしてください。

3. NovaComputeCpuDedicatedSet: **NovaComputeCpuDedicatedSet** パラメーターを使用して、**OvsPmdCoreList** または **Ceph_osd_docker_cpuset_cpus** に記載されていない CPU のリストを指定します。CPU は DPDK NIC と同じ NUMA ノードになければなりません。

11.8.5. HCI-DPDK デプロイメントに推奨される設定

表11.1 HCI デプロイメント用の調整可能なパラメーター

ブロックデバイスの種別	メモリー、デバイスごとの OSD および仮想 CPU
NVMe	メモリー : OSD ごとに 5 GB デバイスごとの OSD 数: 4 デバイスごとの仮想 CPU 数: 3
SSD	メモリー : OSD ごとに 5 GB デバイスごとの OSD 数: 1 デバイスごとの仮想 CPU 数: 4

ブロックデバイスの種別	メモリー、デバイスごとの OSD および仮想 CPU
HDD	メモリー : OSD ごとに 5 GB デバイスごとの OSD 数: 1 デバイスごとの仮想 CPU 数: 1

以下の機能には、同じ NUMA ノードを使用します。

- ディスクコントローラー
- ストレージネットワーク
- ストレージ CPU およびメモリー

DPDK プロバイダーネットワークの以下の機能には、別の NUMA ノードを割り当てます。

- NIC
- PMD CPU
- ソケットメモリー

11.8.6. HCI-DPDK オーバークラウドのデプロイ

以下の手順に従って、DPDK を使用するハイパーコンバージドオーバークラウドをデプロイします。

前提条件

- Red Hat OpenStack Platform (RHOSP) 17.1 以降。
- Red Hat Ceph Storage 6.1 の最新バージョン

手順

1. コントローラーロールと ComputeHCIOvsDpdk ロールの **role_data.yaml** ファイルを生成します。

```
$ openstack overcloud roles generate -o ~/<templates>/roles_data.yaml \
Controller ComputeHCIOvsDpdk
```

2. **openstack flavor create** および **openstack flavor set** コマンドを使用して、新規フレーバーを作成および設定します。
3. RHOSP director と Ceph 設定ファイルを使用して Ceph をデプロイします。

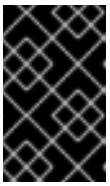
   例

```
$ openstack overcloud ceph deploy --config initial-ceph.conf
```

4. 生成したカスタムの **roles_data.yaml** 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。

   例

```
$ openstack overcloud deploy --templates \
  --timeout 360 \
  -r ~/<templates>/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/\
  cephadm/cephadm-rbd-only.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-ovs-dpdk.yaml \
  -e ~/<templates>/<custom environment file>
```



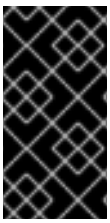
重要

この例では、Ceph RGW (オブジェクトストレージ) を使用せずに Ceph RBD (ブロックストレージ) をデプロイします。デプロイメントに RGW を含めるには、**cephadm-rbd-only.yaml** の代わりに **cephadm.yaml** を使用します。

関連情報

- **Red Hat OpenStack Platform デプロイメントのカスタマイズ**の コンポーザブルサービスとカスタムロール。
- 「ceph 設定ファイルの例」
- **director を使用した Red Hat Ceph Storage と Red Hat OpenStack Platform のデプロイ**で Red Hat Ceph Storage クラスターを設定します 。

11.9. COMPUTE ノードの TIMEMASTER との同期



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、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能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。

時刻プロトコルを使用して、システム間で一貫性のあるタイムスタンプを維持します。

Red Hat OpenStack Platform (RHOSP) には、Precision Time Protocol (PTP) および Network Time Protocol (NTP) のサポートが含まれています。

NTP を使用して、ネットワークのクロックをミリ秒の範囲で同期できます。また、PTP を使用して、クロックをより高いサブマイクロ秒単位の正確さで同期できます。PTP のユースケースの例として、高いスループットと高い干渉のリスクを提供する複数のアンテナが含まれる仮想ラジオアクセスネットワーク (vRAN) があります。

Timemaster は、**ptp4l**と**phc2sys**を**chronyd**または**ntpd**と組み合わせて使用し、システムクロックを NTP や PTP タイムソースに同期させるプログラムです。**phc2sys**および**ptp4l**プログラムは、SHM (Shared Memory Driver) 基準クロックを使用して PTP 時間を**chronyd**または**ntpd**に送信し、タイムソースを比較してシステムクロックを同期させます。

Red Hat Enterprise Linux (RHEL) カーネルにおける PTPv2 プロトコルの実装は**linuxptp**です。

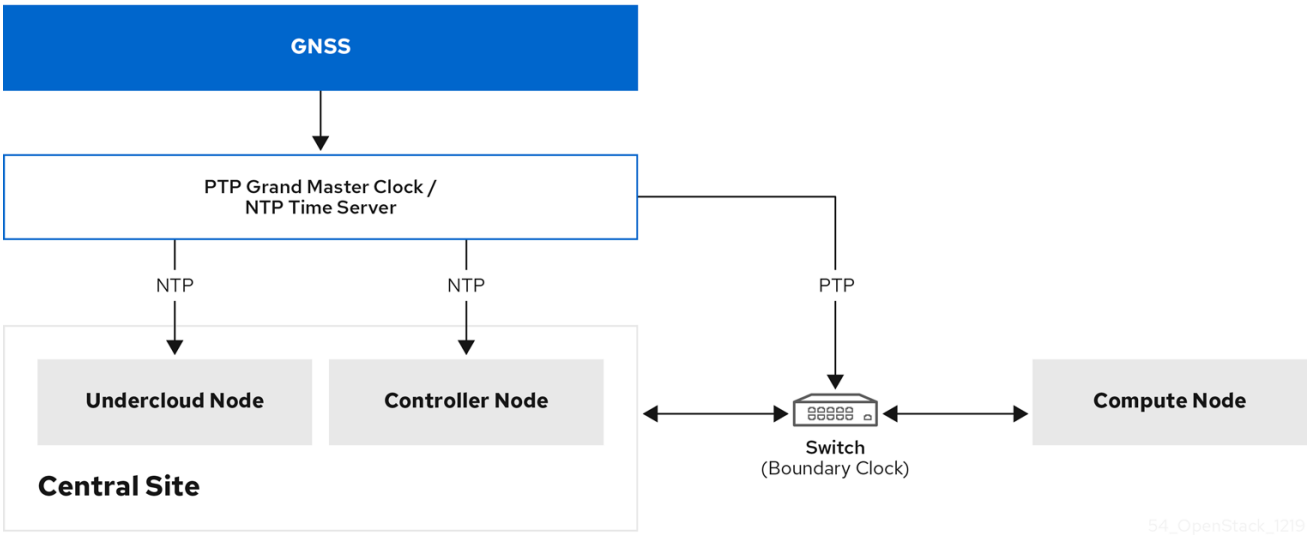
linuxptpパッケージには、PTP バウンダリクロックと通常クロック同期用の**ptp4l**プログラムと、ハードウェアタイムスタンプ用の**phc2sys**プログラムが含まれています。PTP の詳細は、**Red Hat Enterprise Linux System Administrator's Guide**の Introduction to PTP を参照してください。

Chrony は NTP プロトコルの実装です。Chrony の主な設定要素は、Chrony デーモンである**chronyd**と、Chrony コマンドラインインターフェイスである**chonyc**の 2 つです。

Chrony の詳細は、**Red Hat Enterprise Linux システム管理者ガイド**の Chrony スイートを使用して NTP を設定する を参照してください。

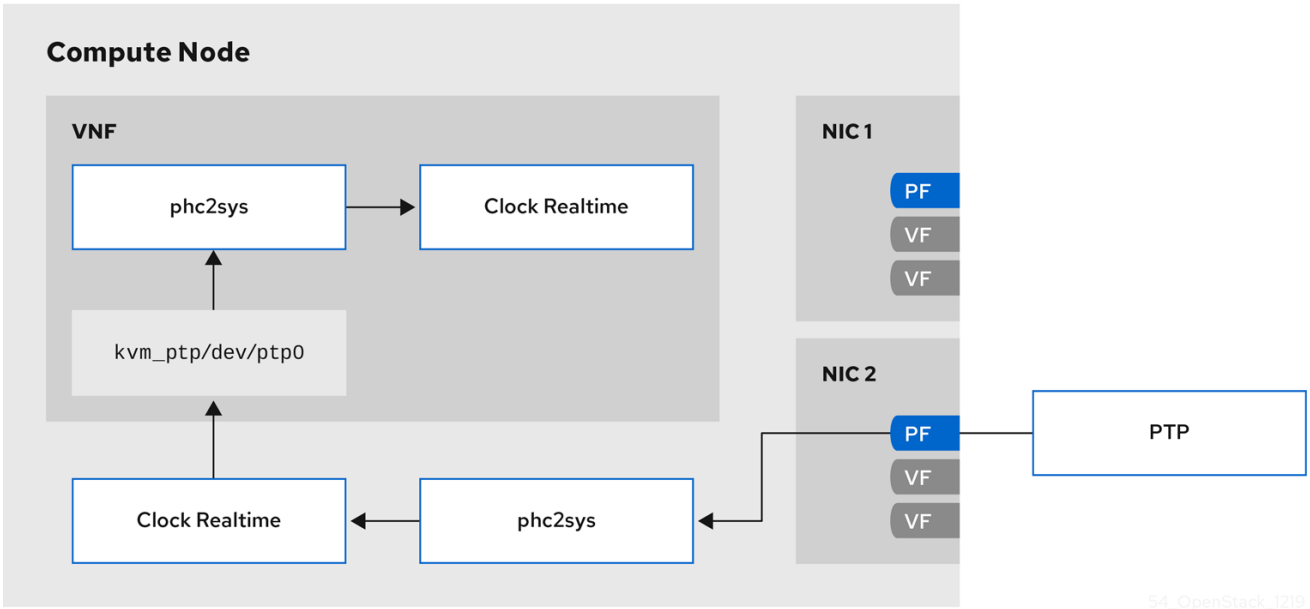
以下の図は、PTP 設定でのパケット移動の概要です。

図11.1 PTP パケット移動の概要



以下の図は、PTP 設定の Compute ノードでのパケット移動の概要です。

図11.2 PTP パケット移動の詳細



11.9.1. Timemaster のハードウェア要件

以下のハードウェア機能があることを確認します。

- NIC にハードウェアのタイムスタンプ機能を設定している。
- マルチキャストパケットを許可するようスイッチを設定している。
- スイッチが境界または透過的なクロックとしても機能するように設定している。

ハードウェアのタイムスタンプを確認するには、**ethtool -T <device>**というコマンドを使用します。

```
$ ethtool -T p5p1
Time stamping parameters for p5p1:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 6
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        ptpv1-l4-sync         (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
        ptpv1-l4-delay-req    (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
        ptpv2-event           (HWTSTAMP_FILTER_PTP_V2_EVENT)
```

透過型または境界クロックのスイッチを使用して、正確性が高く、レイテンシーを低くすることができます。境界クロックにアップリンクスイッチを使用できます。バウンダリクロックスイッチは、PTPv2 ヘッダーの 8 ビットの**correctionField**を使用して、遅延の変動を補正し、エンドクロックの精度を高めます。透過的なクロックスイッチでは、**correctionField**ではなく、エンドクロックが遅延変動を計算します。

11.9.2. Timemaster の設定

オーバークラウドノードの時刻同期に使用するデフォルトの Red Hat OpenStack Platform (RHOSP) サービスは **OS::TripleO::Services::Timesync** です。

既知の制限

- 仮想コントローラー向けに NTP を有効にし、ベアメタルノード用に PTP を有効にします。
- **ptp4l** には互換性のある PTP デバイスが必要なため、virtio インターフェイスには互換性がありません。
- SR-IOV を使用する仮想マシンに Physical Function (PF) を使用します。Virtual Function (VF) は PTP に必要なレジスターを公開せず、仮想マシンは **kvm_ptp** を使用して時刻を計算します。
- 複数のソースと複数のネットワークパスを持つ高可用性 (HA) インターフェイスには互換性がありません。

手順

1. 選択したロールに属するノードで Timemaster サービスを有効にするには、そのロールの **roles_data.yaml** ファイルセクションで、**OS::TripleO::Services::Timesync** が含まれる行を**OS::TripleO::Services::TimeMaster** 行に置き換えます。

```
#- OS::TripleO::Services::Timesync
- OS::TripleO::Services::TimeMaster
```

2. 使用する Compute ロールの heat パラメーターを設定します。

```
#Example
ComputeSriovParameters:
  PTPInterfaces: '0:eno1,1:eno2'
  PTPMessageTransport: 'UDPv4'
```

3. ご自分の環境に該当するその他の環境ファイルと共に、新しい環境ファイルを **openstack overcloud deploy** コマンドに追加します。

```
$ openstack overcloud deploy \
--templates \

…
-e <existing_overcloud_environment_files> \
-e <new_environment_file1> \
-e <new_environment_file2> \
…
```

- <existing_overcloud_environment_files> を既存のデプロイメントに含まれる環境ファイルのリストに置き換えます。
- <new_environment_file> を、オーバークラウドのデプロイメントプロセスに追加する新しい環境ファイルに置き換えます。

検証

- **ptp4linux** と共にインストールされたコマンド **phc_ctl** を使用して NIC ハードウェアクロックにクエリーを実行します。

```
# phc_ctl <clock_name> get
# phc_ctl <clock_name> cmp
```

11.9.3. Timemaster 設定の例

```
$ cat /etc/timemaster.conf
# Configuration file for timemaster

#[ntp_server ntp-server.local]
#minpoll 4
#maxpoll 4

[ptp_domain 0]
interfaces eno1
#ptp4l_setting network_transport l2
#delay 10e-6
```

```
[timemaster]
ntp_program chronyd

[chrony.conf]
#include /etc/chrony.conf
server clock.redhat.com iburst minpoll 6 maxpoll 10

[ntp.conf]
includefile /etc/ntp.conf

[ptp4l.conf]
#includefile /etc/ptp4l.conf
network_transport L2

[chronyd]
path /usr/sbin/chronyd

[ntpd]
path /usr/sbin/ntpd
options -u ntp:ntp -g

[phc2sys]
path /usr/sbin/phc2sys
#options -w

[ptp4l]
path /usr/sbin/ptp4l
#options -2 -i eno1
```

11.9.4. Timemaster 操作の例

```
$ systemctl status timemaster
● timemaster.service - Synchronize system clock to NTP and PTP time sources
   Loaded: loaded (/usr/lib/systemd/system/timemaster.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-08-25 19:10:18 UTC; 2min 6s ago
 Main PID: 2573 (timemaster)
    Tasks: 6 (limit: 357097)
   Memory: 5.1M
   CGroup: /system.slice/timemaster.service
           ├─2573 /usr/sbin/timemaster -f /etc/timemaster.conf
           ├─2577 /usr/sbin/chronyd -n -f /var/run/timemaster/chrony.conf
           ├─2582 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.0.conf -H -i eno1
           ├─2583 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.0.socket -t [0:eno1] -n
0 -E ntpshm -M 0
           ├─2587 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.1.conf -H -i eno2
           └─2588 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.1.socket -t [0:eno2] -n
0 -E ntpshm -M 1

Aug 25 19:11:53 computesriov-0 ptp4l[2587]: [152.562] [0:eno2] selected local clock
e4434b.fffe.4a0c24 as best master
```

第12章 NFV ワークロードに向けた RT-KVM の有効化

Red Hat Enterprise Linux Real Time KVM (RT-KVM) を容易にインストールおよび設定するために、Red Hat OpenStack Platform では以下の機能を使用することができます。

- Red Hat Enterprise Linux for Real Time をプロビジョニングする、real-time Compute ノードロール
- 追加の RT-KVM カーネルモジュール
- Compute ノードの自動設定

12.1. RT-KVM COMPUTE ノードのプランニング

RT-KVM Compute ノードを計画する際には、以下のタスクが完了していることを確認してください。

- RT-KVM Compute ノードには、Red Hat 認定済みサーバーを使用する必要があります。
  詳細は、Red Hat Enterprise Linux for Real Time 用認定サーバー を参照してください。
- アンダークラウドを登録し、有効な Red Hat OpenStack Platform サブスクリプションをアタッチします。
  詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の アンダークラウドの登録およびサブスクリプションのアタッチ を参照してください。
- RT-KVM 用の **rhel-9-server-nfv-rpms** リポジトリーなど、アンダークラウドに必要なリポジトリーを有効にし、システムパッケージを最新バージョンに更新します。

注記

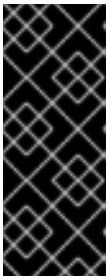
このリポジトリーにアクセスするには、別途 **Red Hat OpenStack Platform for Real Time** SKU のサブスクリプションが必要です。

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の アンダークラウド用リポジトリーの有効化 を参照してください。

real-time のイメージのビルド

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 dnf install libguestfs-tools
```



重要

アンダークラウドに **libguestfs-tools** パッケージをインストールする場合は、アンダークラウドの **tripleo_iscsid** サービスとのポートの競合を避けるために **iscsid.socket** を無効にします。

```
$ sudo systemctl disable --now iscsid.socket
```

2. イメージを抽出します。

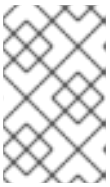
```
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/overcloud-hardened-uefi-full-17.1.x86_64.tar
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent-17.1.x86_64.tar
```

3. デフォルトのイメージをコピーします。

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

4. イメージを登録して、カスタマイズに適切な Red Hat のリポジトリーを有効にします。以下の例の **[username]** および **[password]** を有効な認証情報に置き換えてください。

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager register --username=[username] --password=[password]' \
subscription-manager release --set 9.0
```



注記

コマンドプロンプトで認証情報を使用したら、履歴ファイルから認証情報を削除してセキュリティーを確保することができます。**history -d** コマンドの後に行番号を指定して、履歴内の個々の行を削除することができます。

5. アカウントのサブスクリプションからプール ID のリストを検索し、適切なプール ID をイメージにアタッチします。

```
sudo subscription-manager list --all --available | less
...
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager attach --pool [pool-ID]'
```

6. Red Hat OpenStack Platform で NFV を使用するのに必要なリポジトリーを追加します。

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'sudo subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms \
--enable=rhel-9-for-x86_64-appstream-eus-rpms \
--enable=rhel-9-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-9-x86_64-rpms \
--enable=rhel-9-for-x86_64-nfv-rpms
--enable=fast-datapath-for-rhel-9-x86_64-rpms'
```

7. イメージ上でリアルタイム機能を設定するためのスクリプトを作成します。

```
(undercloud) [stack@undercloud-0 ~]$ cat <<'EOF' > rt.sh
 #!/bin/bash

 set -eux

 dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
 dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host
 grubby --set-default /boot/vmlinuz*rt*
 EOF
```

8. リアルタイム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
```

注記

rt.sh スクリプトの出力に **grubby fatal error: unable to find a suitable template** という行が表示されても、このエラーは無視してかまいません。

9. 前のステップで作成された **virt-customize.log** ファイルを調べ、**rt.sh** スクリプトによりパッケージが正しくインストールされたことを確認します。

```
(undercloud) [stack@undercloud-0 ~]$ cat virt-customize.log | grep Verifying

  Verifying  : kernel-3.10.0-957.el7.x86_64                                 1/1
  Verifying  : 10:qemu-kvm-tools-rhev-2.12.0-18.el7_6.1.x86_64              1/8
  Verifying  : tuned-profiles-realtime-2.10.0-6.el7_6.3.noarch              2/8
  Verifying  : linux-firmware-20180911-69.git85c5d90.el7.noarch             3/8
  Verifying  : tuned-profiles-nfv-host-2.10.0-6.el7_6.3.noarch              4/8
  Verifying  : kernel-rt-kvm-3.10.0-957.10.1.rt56.921.el7.x86_64            5/8
  Verifying  : tuna-0.13-6.el7.noarch                                       6/8
  Verifying  : kernel-rt-3.10.0-957.10.1.rt56.921.el7.x86_64                7/8
  Verifying  : rt-setup-2.0-6.el7.x86_64                                    8/8
```

10. 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
```

11. 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

vmlinuz 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。

12. 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 Compute ノード上の **ComputeOvsDpdkRT** コンポーザブルロールで使用することのできる real-time イメージの準備ができました。

RT-KVM Compute ノード上での BIOS 設定の変更

RT-KVM Compute ノードのレイテンシーを低減するには、Compute ノードの BIOS 設定で、以下のパラメーターのオプションをすべて無効にします。

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

12.2. RT-KVM 対応の OVS-DPDK の設定

12.2.1. Real-time Compute 用のノードの指定

Real-time Compute のノードを指定するには、新しいロールファイルを作成して Real-time Compute のロールを設定し、Real-time Compute リソースクラスを使用してベアメタルノードを設定して、リアルタイムの Compute ノードにタグを付けます。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、オーバークラウドをスケールダウンしてノードのプロビジョニングを解除してから、オーバークラウドをスケールアップして、新しいリソースクラスの割り当てでノードを再プロビジョニングします。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** の オーバークラウドノードのスケーリング を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
[stack@director ~]$ source ~/stackrc
```

3. **/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml** ファイルをベースに、**ComputeRealTime** ロールのパラメーターを設定する **compute-real-time.yaml** 環境ファイルを作成します。
4. **ComputeRealTime** ロールとオーバークラウドに必要なその他のロールを含む、**roles_data_rt.yaml** という名前の新しいロールデータファイルを生成します。次の例では、ロールデータファイル **role_data_rt.yaml** を生成します。これには、ロール **Controller**、**Compute**、および **ComputeRealTime** が含まれます。

```
(undercloud)$ openstack overcloud roles generate \
-o /home/stack/templates/roles_data_rt.yaml \
ComputeRealTime Compute Controller
```

5. ComputeRealTime ロールの roles_data_rt.yaml ファイルを更新します。

```
###############################################
# Role: ComputeRealTime                       #
###############################################
- name: ComputeRealTime
  description: |
    Real Time Compute Node role
  CountDefault: 1
  # Create external Neutron bridge
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computert-%index%'
  deprecated_nic_config_name: compute-rt.yaml
```

6. オーバークラウド用の ComputeRealTime ノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。
詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の オーバークラウドのノードの登録 を参照してください。

7. ノードのハードウェアを検査します。

```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

8. ComputeRealTime 用に指定する各ベアメタルノードに、カスタム ComputeRealTime リソースクラスをタグ付けします。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.RTCOMPUTE <node>
```

<node> は、ベアメタルノードの名前または UUID に置き換えます。

9. ComputeRealTime ロールをノード定義ファイル **overcloud-baremetal-deploy.yaml** に追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てるその他の属性を定義します。

```
- name: Controller
  count: 3
  ...
- name: Compute
  count: 3
  ...
- name: ComputeRealTime
  count: 1
```

```
  defaults:
    resource_class: baremetal.RTCOMPUTE
    network_config:
      template: /home/stack/templates/nic-config/<role_topology_file>
```

- **<role_topology_file>** を **ComputeRealTime** ロールに使用するトポロジーファイルの名前 (**myRoleTopology.j2** など) に置き換えます。既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。
  詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の カスタムネットワークインターフェイステンプレートの定義 を参照してください。デフォルトのネットワーク定義設定を使用するには、ロール定義に **network_config** を含めないでください。

  ノード定義ファイルでノード属性を設定するために使用できるプロパティーに関する詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の ベアメタルノードのプロビジョニング属性 を参照してください。

  ノード定義ファイルの例については、**director を使用した Red Hat OpenStack Platform のインストールと管理** の ノード定義ファイルの例 を参照してください。

10. 次の Ansible Playbook を作成してノードのプロビジョニング中にカーネルを設定し、Playbook を **/home/stack/templates/fix_rt_kernel.yaml** として保存します。

```
# RealTime KVM fix until BZ #2122949 is closed-
- name: Fix RT Kernel
  hosts: allovercloud
  any_errors_fatal: true
  gather_facts: false
  vars:
    reboot_wait_timeout: 900
  pre_tasks:
    - name: Wait for provisioned nodes to boot
      wait_for_connection:
        timeout: 600
        delay: 10
  tasks:
    - name: Fix bootloader entry
      become: true
      shell: |-
        set -eux
        new_entry=$(grep saved_entry= /boot/grub2/grubenv | sed -e s/saved_entry=//)
        source /etc/default/grub
        sed -i "s/options.*/options root=$GRUB_DEVICE ro $GRUB_CMDLINE_LINUX
$GRUB_CMDLINE_LINUX_DEFAULT/" /boot/loader/entries/$(</etc/machine-
id)$new_entry.conf
        cp -f /boot/grub2/grubenv /boot/efi/EFI/redhat/grubenv
  post_tasks:
    - name: Configure reboot after new kernel
      become: true
      reboot:
        reboot_timeout: "{{ reboot_wait_timeout }}"
      when: reboot_wait_timeout is defined
```

11. ノードプロビジョニングファイルの **ComputeOvsDpdkSriovRT** ロール定義に、Playbook として **/home/stack/templates/fix_rt_kernel.yaml** を含めます。

```
- name: ComputeOvsDpdkSriovRT
  ...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: "default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt
intel_iommu=on tsx=off isolcpus=2-19,22-39"
        reboot_wait_timeout: 900
        tuned_profile: "cpu-partitioning"
        tuned_isolated_cores: "2-19,22-39"
        defer_reboot: true
    - playbook: /home/stack/templates/fix_rt_kernel.yaml
      extra_vars:
        reboot_wait_timeout: 1800
```

ノード定義ファイルでノード属性を設定するために使用できるプロパティーに関する詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の ベアメタルノードのプロビジョニング属性 を参照してください。

ノード定義ファイルの例については、**director を使用した Red Hat OpenStack Platform のインストールと管理** の ノード定義ファイルの例 を参照してください。

12. ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
[--stack <stack> \ ]
[--network-config \]
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- オプション: **<stack>** をベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。デフォルトは **overcloud** です。
- オプション: **--network-config** オプションの引数を含めて、Ansible Playbook **cli-overcloud-node-network-config.yaml** にネットワーク定義を提供します。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。
- **<deployment_file>** は、デプロイメントコマンドに含めるために生成する heat 環境ファイルの名前に置き換えます (例 **:/home/stack/templates/overcloud-baremetal-deployed.yaml)**。

13. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

```
(undercloud)$ watch openstack baremetal node list
```

14. **--network-config** オプションを指定せずにプロビジョニングコマンドを実行した場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeAMDSEVNetworkConfigTemplate: /home/stack/templates/nic-
```

```
configs/<rt_compute>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<rt_compute> を **ComputeRealTime** ロールのネットワークトポロジーが含まれるファイルの名前に置き換えます。たとえば、デフォルトのネットワークトポロジーを使用する場合は **computert.yaml** です。

15. その他の環境ファイルとともに環境ファイルをスタックに追加して、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
 -r /home/stack/templates/roles_data_rt.yaml \
 -e /home/stack/templates/overcloud-baremetal-deployed.yaml
 -e /home/stack/templates/node-info.yaml \
 -e [your environment files] \
 -e /home/stack/templates/compute-real-time.yaml
```

12.2.2. OVS-DPDK パラメーターの設定

1. **parameter_defaults** セクションで、トンネル種別を **vxlan** に、ネットワーク種別を **vxlan,vlan** に、それぞれ設定します。

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

2. **parameters_defaults** セクションで、ブリッジマッピングを設定します。

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
  - dpdk-mgmt:br-link0
```

3. **parameter_defaults** セクションで、**ComputeOvsDpdkSriov** 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 #
  ############################
  ComputeOvsDpdkSriovParameters:
    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"
    NovaComputeCpuDedicatedSet: ['4-19,24-39']
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "3072,1024"
    OvsDpdkMemoryChannels: "4"
    OvsPmdCoreList: "2,22,3,23"
    NovaComputeCpuSharedSet: [0,20,1,21]
    NovaLibvirtRxQueueSize: 1024
    NovaLibvirtTxQueueSize: 1024
```

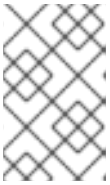
注記

ゲストの作成時にエラーが発生するのを防ぐためには、各 NUMA ノードで少なくとも 1 つの CPU を (シブリングスレッドと共に) 割り当てます。上記の例では、**OvsPmdCoreList** パラメーターの値は NUMA 0 からのコア 2 および 22 ならびに NUMA 1 からのコア 3 および 23 です。

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るフレーバーに **hw:mem_page_size=1GB** も追加する必要があります。



注記

OvsDpdkMemoryChannels は、この手順に必須の設定です。最大限の機能を得るためには、適切なパラメーターおよび値で DPDK をデプロイするようにしてください。

4. SR-IOV 向けにロール固有のパラメーターを設定します。

```
NovaPCIPassthrough:
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.0"
    trusted: "true"
    physical_network: "sriov-1"
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.1"
    trusted: "true"
    physical_network: "sriov-2"
```

12.3. RT-KVM インスタンスの起動

リアルタイム対応の Compute ノードで RT-KVM インスタンスを起動するには、以下の手順を実施します。

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 --id 99 --ram 4096 --disk 20 --vcpus 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。

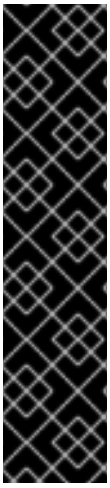
```
$ openstack server create  --image <rhel> --flavor r1.small --nic net-id=<dpdk-net> test-rt
```

3. 割り当てられたエミュレータースレッドをインスタンスが使用していることを確認するには、以下のコマンドを実行します。

```
$ 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13章 例: OVS-DPDK および SR-IOV ならびに VXLAN トンネリングの設定

OVS-DPDK および SR-IOV インターフェイスの両方を持つ Compute ノードをデプロイすることができます。クラスターには ML2/OVS および VXLAN トンネリングが含まれます。



重要

オーバークラウドロールを生成する際に、ロール設定ファイル (例: **roles_data.yaml**) で、**OS::TripleO::Services::Tuned** が含まれる行をコメントアウトまたは削除します。

```
ServicesDefault:
# - OS::TripleO::Services::Tuned
```

OS::TripleO::Services::Tuned をコメントアウトまたは削除した場合は、要件に合わせて **TunedProfileName** パラメーターを設定することができます (例: **"cpu-partitioning"**)。**OS::TripleO::Services::Tuned** 行をコメントアウトまたは削除せずに再デプロイすると、**TunedProfileName** パラメーターには、設定した他の値ではなく **"throughput-performance"** のデフォルト値が設定されます。

13.1. ロールデータの設定

Red Hat OpenStack Platform では、**roles_data.yaml** ファイルにデフォルトロールのセットが用意されています。独自の **roles_data.yaml** ファイルを作成して、必要なロールをサポートすることができます。

以下の例では、**ComputeOvsDpdkSriov** ロールを作成します。

関連情報

- **Red Hat OpenStack Platform デプロイメントのカスタマイズ** における 新しいロールの作成
- roles-data.yaml

13.2. OVS-DPDK パラメーターの設定

1. **parameter_defaults** セクションで、トンネル種別を **vxlan** に、ネットワーク種別を **vxlan,vlan** に、それぞれ設定します。

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

2. **parameters_defaults** セクションで、ブリッジマッピングを設定します。

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
  - dpdk-mgmt:br-link0
```

3. **parameter_defaults** セクションで、**ComputeOvsDpdkSriov** 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 #
  ##########################
  ComputeOvsDpdkSriovParameters:
    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"
    NovaComputeCpuDedicatedSet: ['4-19,24-39']
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "3072,1024"
    OvsDpdkMemoryChannels: "4"
    OvsPmdCoreList: "2,22,3,23"
    NovaComputeCpuSharedSet: [0,20,1,21]
    NovaLibvirtRxQueueSize: 1024
    NovaLibvirtTxQueueSize: 1024
```

注記

ゲストの作成時にエラーが発生するのを防ぐためには、各 NUMA ノードで少なくとも 1 つの CPU を (シブリングスレッドと共に) 割り当てます。上記の例では、**OvsPmdCoreList** パラメーターの値は NUMA 0 からのコア 2 および 22 ならびに NUMA 1 からのコア 3 および 23 です。

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るフレーバーに **hw:mem_page_size=1GB** も追加する必要があります。

注記

OvsDpdkMemoryChannels は、この手順に必須の設定です。最大限の機能を得るためには、適切なパラメーターおよび値で DPDK をデプロイするようにしてください。

4. SR-IOV 向けにロール固有のパラメーターを設定します。

```
NovaPCIPassthrough:
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.0"
    trusted: "true"
    physical_network: "sriov-1"
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.1"
    trusted: "true"
    physical_network: "sriov-2"
```

13.3. 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
  members:
  - type: interface
    name: nic2
    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. **neutron-dhcp-agent** および **neutron-metadata-agent** サービ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: nic3
    mtu: 9000
  - type: vlan
    vlan_id:
      get_param: TenantNetworkVlanID
    mtu: 9000
    addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
```

13.4. DPDK および SR-IOV 用 COMPUTE ノードの設定

デフォルトの **compute.yaml** ファイルから **computeovsdpdksriov.yaml** ファイルを作成し、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。

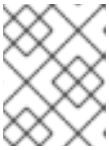
```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
  - type: interface
    name: nic3
    primary: true
  - type: interface
    name: nic4
```

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: nic7
        - type: ovs_dpdk_port
          name: dpdk1
          members:
            - type: interface
              name: nic8
```



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 Compute ノード上の全ブリッジが **ovs_user_bridge** の種別でなければなりません。Red Hat OpenStack Platform では、**ovs_bridge** と **ovs_user_bridge** の両方が同じノード上に存在する設定はサポートされません。

13.5. オーバークラウドのデプロイ

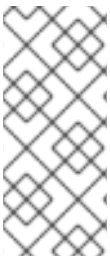
- **overcloud_deploy.sh** スクリプトを実行します。

第14章 NFV を実装した RED HAT OPENSTACK PLATFORM のアップグレード

OVS-DPDK が設定された Red Hat OpenStack Platform (RHOSP) のアップグレードに関する詳細は、**16.2 から 17.1 へのアップグレードフレームワーク** の ネットワーク機能仮想化 (NFV) の準備 を参照してください。

第15章 サンプル DPDK SR-IOV YAML および JINJA2 ファイル

本項では、同じ Compute ノードに Single Root I/O Virtualization (SR-IOV) と Data Plane Development Kit (DPDK) インターフェイスを追加する際の参考として、yaml ファイルの例を示します。



注記

以下のテンプレートは完全に設定された環境から取得したもので、NFV とは関係の無いパラメーターが含まれています。したがって、これらのパラメーターは、ご自分のデプロイメントには該当しない場合があります。コンポーネントのサポートレベルのリストは、Red Hat ナレッジベースのアーティクルComponent Support Graduation を参照してください。

15.1. ROLES_DATA.YAML

- **openstack overcloud roles generate** コマンドを実行して、**roles_data.yaml** ファイルを生成します。
  実際の環境にデプロイするロールに応じて、コマンドにロール名を追加します (例: **Controller**、**ComputeSriov**、**ComputeOvsDpdkRT**、**ComputeOvsDpdkSriov**、またはその他のロール)。

  例

  たとえば、**Controller** ロールおよび **ComputeHCIOvsDpdkSriov** ロールが含まれる **roles_data.yaml** ファイルを生成するには、以下のコマンドを実行します。

```
$ openstack overcloud roles generate -o roles_data.yaml \
Controller ComputeHCIOvsDpdkSriov
```

```
###############################################################################
# File generated by TripleO
###############################################################################
###############################################################################
# Role: Controller                                                            #
###############################################################################
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
```

```
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
# For systems with both IPv4 and IPv6, you may specify a gateway network for
# each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: '%stackname%-controller-%index%'
# Deprecated & backward-compatible values (FIXME: Make parameters consistent)
# Set uses_deprecated_params to True if any deprecated params are used.
uses_deprecated_params: True
deprecated_param_extraconfig: 'controllerExtraConfig'
deprecated_param_flavor: 'OvercloudControlFlavor'
deprecated_param_image: 'controllerImage'
deprecated_nic_config_name: 'controller.yaml'
update_serial: 1
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AodhApi
  - OS::TripleO::Services::AodhEvaluator
  - OS::TripleO::Services::AodhListener
  - OS::TripleO::Services::AodhNotifier
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BarbicanApi
  - OS::TripleO::Services::BarbicanBackendSimpleCrypto
  - OS::TripleO::Services::BarbicanBackendDogtag
  - OS::TripleO::Services::BarbicanBackendKmip
  - OS::TripleO::Services::BarbicanBackendPkcs11Crypto
  - OS::TripleO::Services::BootParams
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CeilometerAgentCentral
  - OS::TripleO::Services::CeilometerAgentNotification
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CephGrafana
  - OS::TripleO::Services::CephMds
  - OS::TripleO::Services::CephMgr
  - OS::TripleO::Services::CephMon
  - OS::TripleO::Services::CephRbdMirror
  - OS::TripleO::Services::CephRgw
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::CinderApi
  - OS::TripleO::Services::CinderBackendDellPs
  - OS::TripleO::Services::CinderBackendDellSc
  - OS::TripleO::Services::CinderBackendDellEMCPowermax
  - OS::TripleO::Services::CinderBackendDellEMCPowerStore
  - OS::TripleO::Services::CinderBackendDellEMCSc
  - OS::TripleO::Services::CinderBackendDellEMCUnity
  - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
  - OS::TripleO::Services::CinderBackendDellEMCVNX
  - OS::TripleO::Services::CinderBackendDellEMCVxFlexOS
  - OS::TripleO::Services::CinderBackendDellEMCXtremio
  - OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
  - OS::TripleO::Services::CinderBackendNetApp
  - OS::TripleO::Services::CinderBackendPure
  - OS::TripleO::Services::CinderBackendScaleIO
  - OS::TripleO::Services::CinderBackendVRTSHyperScale
```

```
    - OS::TripleO::Services::CinderBackendNVMeOF
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderHPELeftHandISCSI
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Clustercheck
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ContainerImagePrepare
    - OS::TripleO::Services::DesignateApi
    - OS::TripleO::Services::DesignateCentral
    - OS::TripleO::Services::DesignateProducer
    - OS::TripleO::Services::DesignateWorker
    - OS::TripleO::Services::DesignateMDNS
    - OS::TripleO::Services::DesignateSink
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Ec2Api
    - OS::TripleO::Services::Etcd
    - OS::TripleO::Services::ExternalSwiftProxy
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GnocchiApi
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatEngine
    - OS::TripleO::Services::Horizon
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicInspector
    - OS::TripleO::Services::IronicPxe
    - OS::TripleO::Services::IronicNeutronAgent
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::ManilaApi
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaBackendIsilon
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendUnity
    - OS::TripleO::Services::ManilaBackendVNX
    - OS::TripleO::Services::ManilaBackendVMAX
    - OS::TripleO::Services::ManilaScheduler
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MistralApi
    - OS::TripleO::Services::MistralEngine
    - OS::TripleO::Services::MistralExecutor
    - OS::TripleO::Services::MistralEventEngine
    - OS::TripleO::Services::Multipathd
```

```
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronBgpVpnApi
    - OS::TripleO::Services::NeutronSfcApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL2gwAgent
    - OS::TripleO::Services::NeutronL2gwApi
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronML2FujitsuCfab
    - OS::TripleO::Services::NeutronML2FujitsuFossw
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NeutronAgentsIBConfig
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaIronic
    - OS::TripleO::Services::NovaMetadata
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaVncProxy
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OctaviaApi
    - OS::TripleO::Services::OctaviaDeploymentConfig
    - OS::TripleO::Services::OctaviaHealthManager
    - OS::TripleO::Services::OctaviaHousekeeping
    - OS::TripleO::Services::OctaviaWorker
    - OS::TripleO::Services::OpenStackClients
    - OS::TripleO::Services::OVNDBs
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::PankoApi
    - OS::TripleO::Services::PlacementApi
    - OS::TripleO::Services::OsloMessagingRpc
    - OS::TripleO::Services::OsloMessagingNotify
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rear
    - OS::TripleO::Services::Redis
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::Rsyslog
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::SaharaApi
    - OS::TripleO::Services::SaharaEngine
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftDispersion
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
```

```
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::Zaqar
##############################################################################
####
# Role: ComputeHCIOvsDpdkSriov                                #
##############################################################################
####
- name: ComputeHCIOvsDpdkSriov
  description: |
    ComputeOvsDpdkSriov Node role hosting Ceph OSD too
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
    StorageMgmt:
      subnet: storage_mgmt_subnet
  # CephOSD present so serial has to be 1
  update_serial: 1
  RoleParametersDefault:
    TunedProfileName: "cpu-partitioning"
    VhostuserSocketGroup: "hugetlbfs"
    NovaLibvirtRxQueueSize: 1024
    NovaLibvirtTxQueueSize: 1024
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephOSD
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsDpdk
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::Multipathd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NovaAZConfig
    - OS::TripleO::Services::NovaCompute
```

```
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::OvsDpdkNetcontrold
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rear
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::Rsyslog
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
```

15.2. NETWORK-ENVIRONMENT-OVERRIDES.YAML

```
---
parameter_defaults:
  # The tunnel type for the tenant network (geneve or vlan). Set to '' to disable tunneling.
  NeutronTunnelTypes: "geneve"
  # The tenant network type for Neutron (vlan or geneve).
  NeutronNetworkType: ["geneve", "vlan"]
  NeutronExternalNetworkBridge: "'br-access'"
  # NTP server configuration.
  # NtpServer: ["clock.redhat.com"]
  # MTU global configuration
  NeutronGlobalPhysnetMtu: 9000
  # Configure the classname of the firewall driver to use for implementing security groups.
  NeutronOVSFirewallDriver: openvswitch
  SshServerOptionsOverrides:
    UseDns: "no"
  # Enable log level DEBUG for supported components
  Debug: true

  # From Rocky live migration with NumaTopologyFilter disabled by default
  # https://bugs.launchpad.net/nova/+bug/1289064
  NovaEnableNUMALiveMigration: true
  NeutronPluginExtensions: "port_security,qos,segments,trunk,placement"
  # RFE https://bugzilla.redhat.com/show_bug.cgi?id=1669584
  NeutronServicePlugins: "ovn-router,trunk,qos,placement"
  NeutronSriovAgentExtensions: "qos"

  ##############################
  #  Scheduler configuration #
  ##############################
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
```

```
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
    - AggregateInstanceExtraSpecsFilter
  ComputeOvsDpdkSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/nic-configs/computeovsdpdksriov.yaml"
  ControllerSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/nic-configs/controller.yaml"
```

15.3. コントローラー.J2

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks if network not in 'Tenant,External' %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: interface
  name: nic1
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes:
  - ip_netmask: 169.254.169.254/32
    next_hop: {{ ctlplane_ip }}

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu }}
  bonding_options: mode=active-backup
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: interface
    name: nic2
    primary: true

{% for network in role_networks if network not in 'Tenant,External' %}
- type: vlan
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  device: bond_api
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~ '_cidr') }}
{% endfor %}

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

```
    members:
    - type: interface
      name: nic3
      mtu: 9000
    - type: vlan
      vlan_id: {{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}
      mtu: 9000
      addresses:
      - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['Tenant'] ~ '_cidr') }}

  - type: ovs_bridge
    name: br-dpdk0
    use_dhcp: false
    mtu: 9000
    members:
    - type: interface
      name: nic4
      mtu: 9000

  - type: ovs_bridge
    name: br-dpdk1
    use_dhcp: false
    mtu: 9000
    members:
    - type: interface
      name: nic5
      mtu: 9000

  - type: ovs_bridge
    name: br-sriov1
    use_dhcp: false
    mtu: 9000
    members:
    - type: interface
      name: nic6
      mtu: 9000

  - type: ovs_bridge
    name: br-sriov2
    use_dhcp: false
    mtu: 9000
    members:
    - type: interface
      name: nic7
      mtu: 9000

  - type: interface
    name: nic8
    use_dhcp: false
    defroute: false

  - type: interface
    name: nic9
    use_dhcp: false
    defroute: false
```

```
- type: ovs_bridge
  name: br-access
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic10
    mtu: 9000
  - type: vlan
    vlan_id: {{ lookup('vars', networks_lower['External'] ~ '_vlan_id') }}
    mtu: 9000
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower['External'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['External'] ~ '_cidr') }}
    routes:
    - default: true
      next_hop: {{ lookup('vars', networks_lower['External'] ~ '_gateway_ip') }}
```

15.4. COMPUTE-OVS-DPDK.J2

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks if network not in 'Tenant,External' %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: interface
  name: nic1
  use_dhcp: false
  default: no

- type: interface
  name: nic2
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes:
  - ip_netmask: 169.254.169.254/32
    next_hop: {{ ctlplane_ip }}
  - default: true
    next_hop: {{ ctlplane_gateway_ip }}

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu }}
  bonding_options: mode=active-backup
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: interface
    name: nic2
    primary: true
```

```
{% for network in role_networks if network not in 'Tenant,External' %}
- type: vlan
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  device: bond_api
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
{% endfor %}

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra: "set port br-link0 tag={{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}"
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['Tenant'] ~ '_cidr')}}
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: 1
    ovs_extra: "set port dpdkbond0 bond_mode=balance-slb"
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
          - type: interface
            name: nic7
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: interface
            name: nic8

- type: ovs_user_bridge
  name: br-dpdk0
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
  members:
    - type: ovs_dpdk_port
      name: dpdk2
      members:
        - type: interface
          name: nic5

- type: ovs_user_bridge
  name: br-dpdk1
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
  members:
    - type: ovs_dpdk_port
      name: dpdk3
      members:
        - type: interface
```

```
        name: nic6

- type: sriov_pf
  name: nic9
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false

- type: sriov_pf
  name: nic10
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
```

15.5. OVERCLOUD_DEPLOY.SH

```
#!/bin/bash

tht_path='/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid'
[[ ! -d "$tht_path/roles" ]] && mkdir $tht_path/roles
openstack overcloud roles generate -o $tht_path/roles/roles_data.yaml ControllerSriov
ComputeOvsDpdkSriov

openstack overcloud deploy \
  --templates /usr/share/openstack-tripleo-heat-templates \
  --ntp-server
clock.redhat.com,time1.google.com,time2.google.com,time3.google.com,time4.google.com \
  --stack overcloud \
  --roles-file $tht_path/roles/roles_data.yaml \
  -n $tht_path/network/network_data_v2.yaml \
  --deployed-server \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/overcloud-networks-deployed.yaml \
  -e /home/stack/templates/overcloud-vip-deployed.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-ha.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-dpdk.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
  -e /home/stack/containers-prepare-parameter.yaml \
  -e $tht_path/network-environment-overrides.yaml \
  -e $tht_path/api-policies.yaml \
  -e $tht_path/bridge-mappings.yaml \
  -e $tht_path/neutron-vlan-ranges.yaml \
  -e $tht_path/dpdk-config.yaml \
  -e $tht_path/sriov-config.yaml \
  --log-file overcloud_deployment.log
```