# Red Hat OpenStack Platform 16.1

## ネットワーク機能仮想化 (NFV) のプランニングおよび設定ガイド

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

Last Updated: 2024-04-03

# Red Hat OpenStack Platform 16.1 ネットワーク機能仮想化 (NFV) のプランニングおよび設定ガイド

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

OpenStack Team
rhos-docs@redhat.com

## 法律上の通知



## 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を提供すると共に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。

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

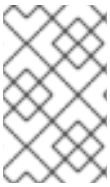
特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

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

# 第1章 NFV の概要

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

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



**注記**

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

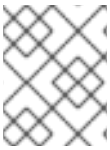
Red Hat OpenStack Platform director を使用して、特定のネットワーク種別 (外部、プロジェクト、内部 API 等) を分離します。ネットワークは、単一のネットワークインターフェイス上に、または複数のホストネットワークインターフェイスに分散してデプロイすることが可能です。Open vSwitch により、複数のインターフェイスを単一のブリッジに割り当てて、ボンディングを作成することができます。Red Hat OpenStack Platform インストール環境でネットワークの分離を設定するには、テンプレートファイルを使用します。テンプレートファイルを指定しない場合、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** 等のホストのネットワークテンプレート: 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が定義されます。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。

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。



**注記**

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、Getting Started with Kubernetes の YAML in a Nutshell を参照してください。

# 第2章 ハードウェア要件

本項では、NFV のハードウェア要件について説明します。

Red Hat OpenStack Platform の認定済みハードウェアの完全リストについては、Red Hat OpenStack Platform certified hardware を参照してください。

## 2.1. テスト済み 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。

Mellanox ConnectX-4 または ConnectX-5 ネットワークインターフェイス上に OVS-DPDK を設定する場合には、compute-ovs-dpdk.yaml ファイル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。

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

## 2.2. NUMA ノードのトポロジーについての理解

デプロイメントを計画する際には、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についての詳しい情報は、director のインストールと使用方法 を参照してください。

### 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** ファイルの **inspection_extras** パラメーターについての詳しい情報は、Director Installation and Usage の Configuring director 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

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,
```

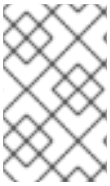
```
      23
    ],
    "numa_node": 0
  },
  {
    "cpu": 1,
    "thread_siblings": [
      9,
      25
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "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
    }
  ]
}
```

## 2.3. 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2.1 BIOS 設定

| パラメーター | 設定 |
|---|---|
| **C3 Power State** | Disabled |
| **C6 Power State** | Disabled |
| **MLC Streamer** | Enabled |
| **MLC Spacial Prefetcher** | Enabled |
| **DCU Data Prefetcher** | Enabled |
| **DCA** | Enabled |

| パラメーター | 設定 |
| --- | --- |
| **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。<br>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3章 ソフトウェア要件

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

## 3.1. 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ついての詳細は、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)
                     Red Hat Virtualization
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. 以下のコマンドで **Pool ID** の値を指定して、Red Hat OpenStack Platform 16.1 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-8-for-x86_64-baseos-eus-rpms \
--enable=rhel-8-for-x86_64-appstream-eus-rpms \
--enable=rhel-8-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-8-x86_64-rpms \
--enable=openstack-16.1-for-rhel-8-x86_64-rpms \
--enable=rhel-8-for-x86_64-nfv-rpms \
--enable=advanced-virt-for-rhel-8-x86_64-rpms \
--enable=fast-datapath-for-rhel-8-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
```

**注記**

オーバークラウドノードを登録するには、Ansible ベースのオーバークラウド登録 を参照してください。

## 3.2. 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)
- Data Plane Development Kit 対応 Open vSwitch (OVS-DPDK)

また、以下のどの機能と共に RHOSP をデプロイすることもできます。

- コンポーザブルロール
- ハイパーコンバージドインフラストラクチャー
- リアルタイムコンピュート
- OVS ハードウェアオフロード

デフォルトのソフトウェア定義ネットワーク (SDN) ソリューションとして Open Virtual Network (OVN) を使用する RHOSP NFV デプロイメントはサポートされません。以下に示す RHOSP NFV OVN の設定は、RHOSP 16.1.4 では一般提供として利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
- OVN と OVS TC Flower オフロードを組み合わせた設定

### 3.2.1. OVS メカニズムドライバーを使用する RHOSP のデプロイ

OVS メカニズムドライバーと共に RHOSP をデプロイします。

手順

1. **containers-prepare-parameter.yaml** ファイルを修正し、**neutron_driver** パラメーターを **ovs** に設定します。

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

2. **/usr/share/openstack-tripleo-heat-templates/environments/services** ディレクトリーの **neutron-ovs.yaml** 環境ファイルを、デプロイメント用スクリプトに追加します。

```
TEMPLATES=/usr/share/openstack-tripleo-heat-templates

openstack overcloud deploy --templates \
-e ${TEMPLATES}/environments/network-environment.yaml \
-e ${TEMPLATES}/environments/network-isolation.yaml \
-e ${TEMPLATES}/environments/services/neutron-ovs.yaml \
-e ${TEMPLATES}/environments/services/neutron-ovs-dpdk.yaml \
-e ${TEMPLATES}/environments/services/neutron-sriov.yaml \
-e /home/stack/containers-prepare-parameter.yaml
```

### 3.2.2. 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

**注記**

この RHOSP NFV OVN の設定は、RHOSP 16.1.4 では一般提供として利用することができます。

DPDK および SRIOV 仮想マシンを OVN と同じノードにデプロイします。

手順

1. **ComputeOvsDpdkSriov** ロールを作成します。

```
openstack overcloud roles generate -o roles_data.yaml Controller ComputeOvsDpdkSriov
```

2. **OS::TripleO::Services::OVNMetadataAgent** を Controller ロールに追加します。

3. **resource_registry** パラメーターを使用して、OVS-DPDK 用のカスタムリソースを追加します。

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

4. parameter_defaults セクションで、トンネル種別パラメーターの値を編集して **geneve** に設定します。

```
NeutronTunnelTypes: 'geneve'
NeutronNetworkType: ['geneve', 'vlan']
```

5. オプション: 集中ルーティングモデルを使用する場合は、分散仮想ルーター (DVR) を無効にします。

```
NeutronEnableDVR: false
```

6. **parameters_defaults** セクションで、ブリッジマッピングを設定します。

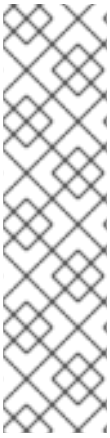
```
# The OVS logical-to-physical bridge mappings to use.
 NeutronBridgeMappings: "datacentre:br-ex,data1:br-link0,data2:br-link1"
```

7. **computeovsdpdksriov.yaml** ファイルでネットワークインターフェイスを設定します。

```
 - 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_port
     name: br-link0-dpdk-port0
     rx_queue: 1
     members:
     - type: interface
       name: eno3
 - type: sriov_pf
   name: eno4
   use_dhcp: false
   numvfs: 5
   defroute: false
   nm_controlled: true
   hotplug: true
   promisc: false
```

8. デプロイメントスクリプトに以下の yaml ファイルを追加します。

   - neutron-ovn-dpdk.yaml
   - neutron-ovn-sriov.yaml



**注記**

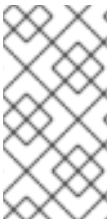
Open Virtual Networking (OVN) は、Red Hat OpenStack Platform 16.1 におけるデフォルトのネットワークメカニズムドライバーです。分散仮想ルーター (DVR) で OVN を使用する場合には、**openstack overcloud deploy** コマンドに **environments/services/neutron-ovn-dvr-ha.yaml** ファイルを追加する必要があります。DVR なしで OVN を使用する場合は、**environments/services/neutron-ovn-ha.yaml**ファイルを**openstack overcloud deploy**コマンドに含め、**NeutronEnableDVR**パラメーターを**false**に設定する必要があります。SR-IOV と OVN を使用する場合は、**openstack overcloud deploy**コマンドの最後の OVN 環境ファイルとして**environments/services/neutron-ovn-sriov.yaml**ファイルを含める必要があります。

### 3.2.3. OVN と OVS TC Flower オフロードを組み合わせた設定のデプロイ

OVS TC Flower オフロードを OVN と同じノードにデプロイします。

**注記**

この RHOSP NFV OVN の設定は、RHOSP 16.1.4 では一般提供として利用することができます。



**注記**

Red Hat Enterprise Linux Traffic Control (TC) サブシステムは、接続追跡 (conntrack) ヘルパーまたはアプリケーション層ゲートウェイ (ALG) をサポートしていません。したがって、ALG を使用している場合は、TC Flower オフロードを無効にする必要があります。

手順

1. **ComputeOvsDpdkSriov** ロールを作成します。

```
openstack overcloud roles generate -o roles_data.yaml ControllerSriov ComputeSriov
```

2. 実際のデプロイメントに応じて **physical_network** パラメーターを設定します。

   - 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。**NeutronBridgeMappings** パラメーターにもこの値を使用します。

   - ロール固有のパラメーター (**ComputeSriovOffloadParameters**) の下で、**OvsHwOffload** パラメーターの値が **true** であることを確認します。

```
parameter_defaults:
  NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-offload'
  NeutronNetworkVLANRanges: 'tenant:502:505'
  NeutronFlatNetworks: 'datacentre,tenant'
  NeutronPhysicalDevMappings:
    - tenant:ens1f0
    - tenant:ens1f1

  NovaPCIPassthrough:
  - address: "0000:17:00.1"
    physical_network: "tenant"
```

```
    - address: "0000:3b:00.1"
      physical_network: "tenant"
    NeutronTunnelTypes: ''
    NeutronNetworkType: 'vlan'
    ComputeSriovOffloadParameters:
      OvsHwOffload: True
      KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32
   intel_iommu=on iommu=pt isolcpus=1-11,13-23"
      IsolCpusList: "1-11,13-23"
      NovaReservedHostMemory: 4096
      NovaComputeCpuDedicatedSet: ['1-11','13-23']
      NovaComputeCpuSharedSet: ['0','12']
```

3. **computeovsdpdksriov.yaml** ファイルでネットワークインターフェイスを設定します。

```
- type: ovs_bridge
 name: br-offload
 mtu: 9000
 use_dhcp: false
 addresses:
 - ip_netmask:
    get_param: TenantIpSubnet
 members:
 - type: linux_bond
   name: bond-pf
   bonding_options: "mode=active-backup miimon=100"
   members:
   - type: sriov_pf
     name: ens1f0
     numvfs: 3
     primary: true
     promisc: true
     use_dhcp: false
     defroute: false
     link_mode: switchdev
   - type: sriov_pf
     name: ens1f1
     numvfs: 3
     promisc: true
     use_dhcp: false
     defroute: false
     link_mode: switchdev
```

4. デプロイメントスクリプトに以下の yaml ファイルを追加します。

   - ovs-hw-offload.yaml
   - neutron-ovn-sriov.yaml

```
TEMPLATES_HOME="/usr/share/openstack-tripleo-heat-templates"
  CUSTOM_TEMPLATES="/home/stack/templates"

  openstack overcloud deploy --templates \
    -r ${CUSTOM_TEMPLATES}/roles_data.yaml \
```

```
-e ${TEMPLATES_HOME}/environments/services/neutron-ovn-sriov.yaml \
-e ${TEMPLATES_HOME}/environments/ovs-hw-offload.yaml \
-e ${CUSTOM_TEMPLATES}/network-environment.yaml
```

## 3.3. 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、テスト済み NIC を参照してください。

## 3.4. 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4章 ネットワークの考慮事項

アンダークラウドのホストには、最低でも以下のネットワークが必要です。

- プロビジョニングネットワーク: オーバークラウドで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しい情報は、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。

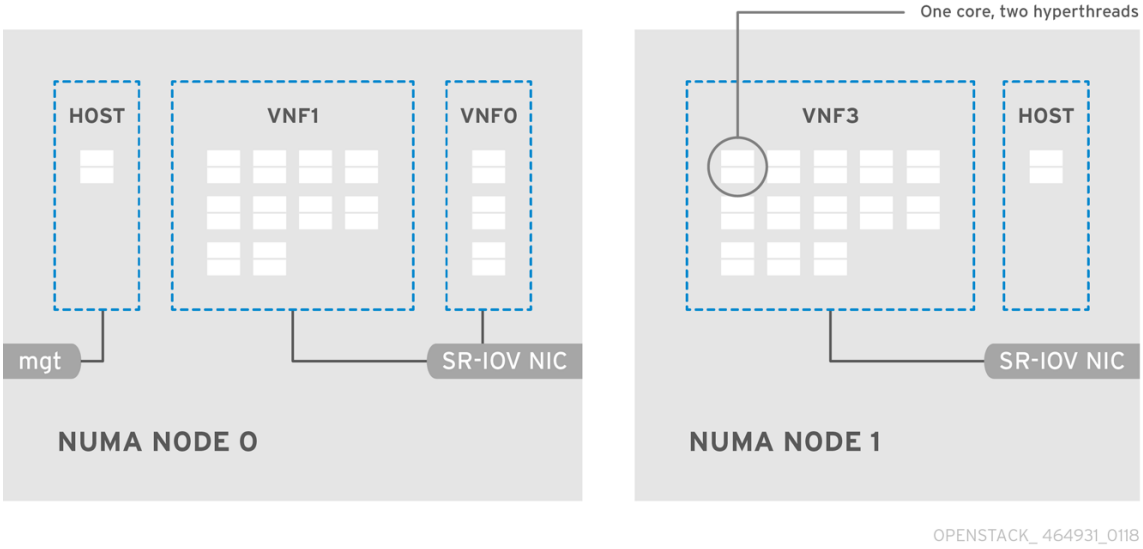
# 第5章 SR-IOV デプロイメントのプランニング

コンピュート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。

## 5.1. SR-IOV デプロイメント向けのハードウェアの分割

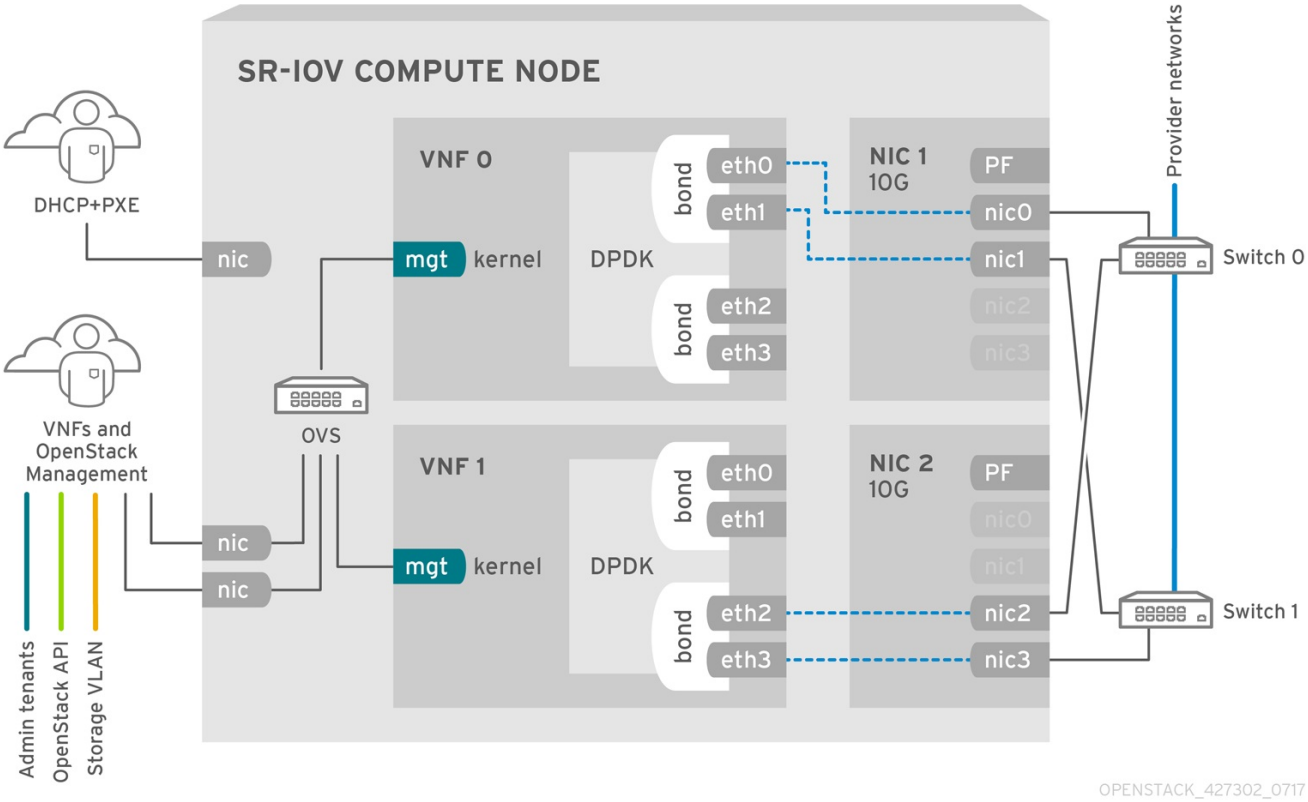
SR-IOV で高パフォーマンスを実現するには、ホストとゲストの間でリソースを分割します。




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。

## 5.2. NFV SR-IOV デプロイメントのトポロジー

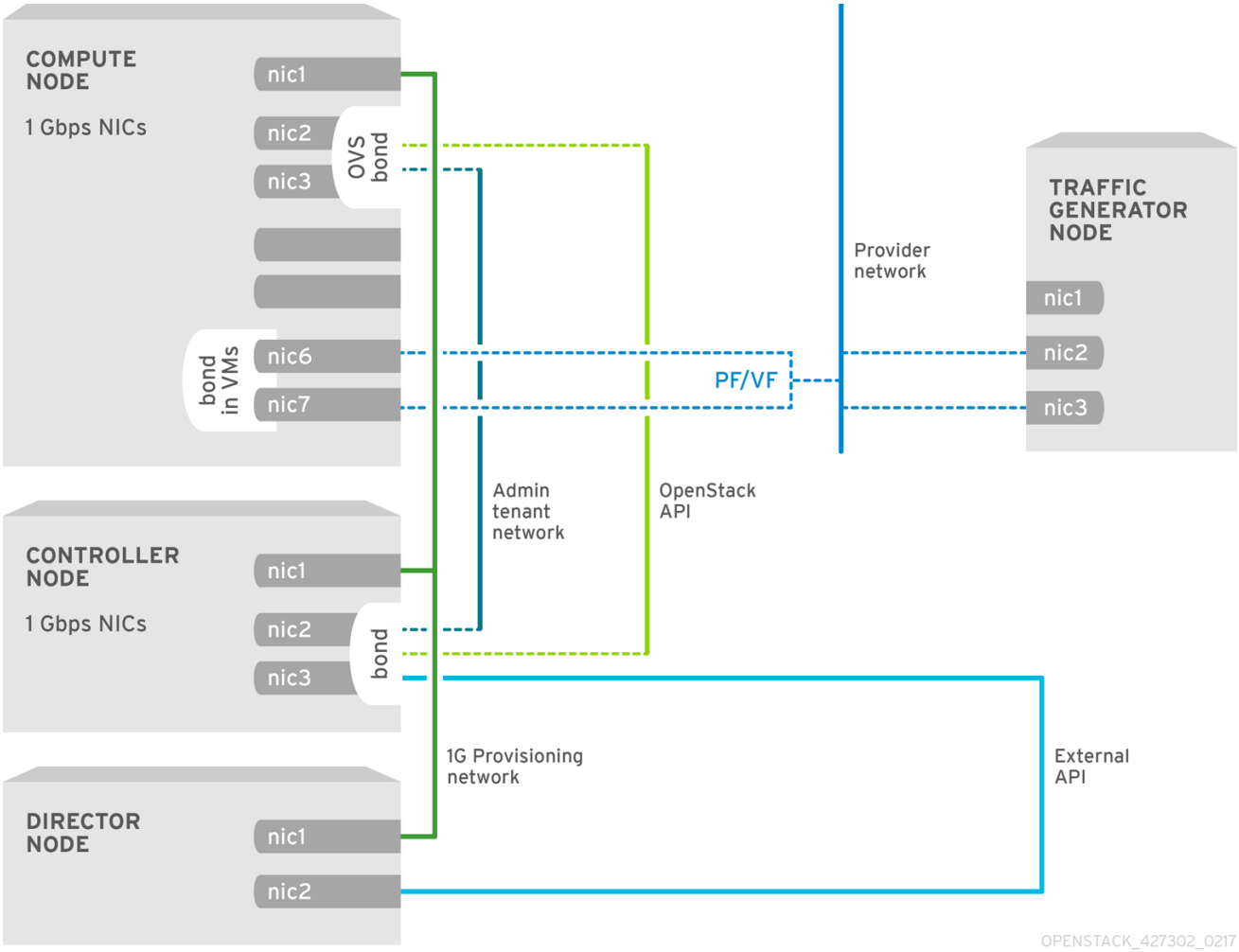
以下の図には、2 つの VNF が示されています。各 VNF には、**mgt** で示された管理インターフェイスおよびデータプレーンインターフェイスがあります。管理インターフェイスは **ssh** アクセスなどを管理します。データプレーンインターフェイスは VNF を DPDK にボンディングして、高可用性を確保します。VNF は DPDK ライブラリーを使用してデータプレーンインターフェイスをボンディングするためです。この図には、冗長性を確保するための 2 つのプロバイダーネットワークも示されています。コンピュートノードには 2 つの標準 NIC がボンディングされ、VNF 管理と Red Hat OpenStack Platform API 管理の間で共有されています。




この図は、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。

### 5.2.1. 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 を搭載したコンピュート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。

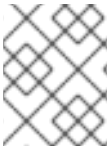

COMPUTE NODE
1 Gbps NICs
nic1
nic2
nic3
OVS bond
bond in VMs
nic6
nic7
CONTROLLER NODE
1 Gbps NICs
nic1
nic2
nic3
bond
DIRECTOR NODE
nic1
nic2
Admin tenant network
OpenStack API
1G Provisioning network
Provider network
PF/VF
TRAFFIC GENERATOR NODE
nic1
nic2
nic3
External API
OPENSTACK_427302_0217

# 第6章 SR-IOV テクノロジーのデプロイ

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。

## 6.1. 前提条件

- オーバークラウドのデプロイ前にアンダークラウドをインストールおよび設定する方法の詳細は、Director Installation and Usage を参照してください。



**注記**

director heat テンプレートが変更する **/etc/tuned/cpu-partitioning-variables.conf** の値を、手動で編集しないでください。

## 6.2. SR-IOV の設定

Single Root I/O Virtualization (SR-IOV) と共に Red Hat Open Stack Platform (RHOSP) をデプロイするには、インスタンスが直接アクセスを要求できる SR-IOV 機能を持つ共有 PCIe リソースを設定します。

**注記**

以下に示す CPU 割り当て、メモリーの確保、および NIC の設定は一例であり、実際のユースケースとは異なる場合があります。

手順

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

```
[stack@director ~]$ 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。

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

**ComputeSriov**は、RHOSP インストールで提供されるカスタムロールで、デフォルトの Compute サービスに加えて、**NeutronSriovAgent**、**NeutronSriovHostConfig**サービスが含まれます。

4. SR-IOV コンテナーを準備するには、**overcloud_images.yaml** ファイルを生成するときに**neutron-sriov.yaml**ファイルと**roles_data_compute_sriov.yaml**ファイルを含めます。

```
$ sudo openstack tripleo container image prepare \
  --roles-file ~/templates/roles_data_compute_sriov.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml \
```

```
-e ~/containers-prepare-parameter.yaml \
--output-env-file=/home/stack/templates/overcloud_images.yaml
```

コンテナーイメージの準備に関する詳しい情報は、**Director Installation and Usage**のPreparing container imagesを参照してください。

5. 環境ファイルディレクトリーに**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**ファイルのコピーを作成します。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml
/home/stack/templates/network-environment-sriov.yaml
```

6. **network-environment-sriov.yaml**ファイルの**parameter_defaults**に次のパラメーターを追加して、クラスターとハードウェア設定の SR-IOV ノードを設定します。

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

7. 各 PCI デバイスタイプの**vendor_id**および**product_id**を確認するには、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。

- デプロイされたオーバークラウドから**vendor_id**と**product_id**を返すには、次のコマンドを使用します。

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

- オーバークラウドをまだデプロイしていない場合に Physical Function (PF) の**vendor_id**と**product_id**を返すには、次のコマンドを使用します。

```
(undercloud) [stack@undercloud-0 ~]$ openstack baremetal introspection data save
<baremetal_node_name> | jq '.inventory.interfaces[] | .name, .vendor, .product'
```

8. **network-environment-sriov.yaml**ファイルで SR-IOV Compute ノードのロール固有のパラメーターを設定します。

```
  ComputeSriovParameters:
    IsolCpusList: "1-19,21-39"
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
intel_iommu=on isolcpus=1-19,21-39"
    TunedProfileName: "cpu-partitioning"
    NeutronBridgeMappings:
      - tenant:br-link0
    NeutronPhysicalDevMappings:
      - tenant:p7p1
    NovaComputeCpuDedicatedSet: '1-19,21-39'
    NovaReservedHostMemory: 4096
```

**注記**

**NovaVcpuPinSet**パラメーターは非推奨になり、専用の固定されたワークロード用に**NovaComputeCpuDedicatedSet**に置き換えられました。

9. **network-environment-sriov.yaml**ファイルで SR-IOV Compute ノードの PCI パススルーデバイスを設定します。

```
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。**address** パラメーターの設定方法に関する情報は、**Configuring the Compute Service for Instance Creation**のGuidelines for configuring **NovaPCIPassthrough**を参照してください。
- **<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。**NovaPCIPassthrough**を設定する方法の詳細については、**Configuring the Compute Service for Instance Creation**ガイドの Guidelines for configuring **NovaPCIPassthrough**を参照してください。

10. **compute.yaml** ネットワーク設定テンプレートで、SR-IOV が有効なインターフェイスを設定します。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。

```
    - type: sriov_pf
      name: p7p3
      mtu: 9000
      numvfs: 10
      use_dhcp: false
      defroute: false
      nm_controlled: true
```

```
    hotplug: true
    promisc: false

  - type: sriov_pf
    name: p7p4
    mtu: 9000
    numvfs: 10
    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 に SR-IOV ポートを持つ実行中のインスタンスが中断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。

11. デフォルトフィルターのリストに、値 **AggregateInstanceExtraSpecsFilter** が含まれる状態にします。

```
NovaSchedulerDefaultFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','AggregateInstanceExtraSpecsFilter']
```

12. **overcloud_deploy.sh** スクリプトを実行します。

## 6.3. NIC の分割

この機能は Red Hat OpenStack Platform (RHOSP) 16.1.2 から一般提供として利用することができ、Intel Fortville NIC および Mellanox CX-5 NIC で検証されています。

RHOSP ホストが Virtual Function (VF) を使用できるように、Single Root I/O Virtualization (SR-IOV) を設定することができます。

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。

手順

1. 選択したロールの NIC 設定ファイルを開きます。

2. 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> #optional (Defaults to true)
```

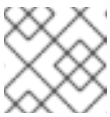
**注記**

**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。

3. 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。
- **<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。

4. スプーフィングの確認を無効にし、VF 上の **linux_bond** の **sriov_vf** に VLAN タグを適用します。
5. 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。以下に例を示します。

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

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。

6. NIC の分割が必要なすべてのノードで **IOMMU** を有効にします。たとえば、コンピュートノードに 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"
```

7. その他の環境ファイルと共にロールファイルおよび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 os-net-config.yaml
  -e [your environment files] \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

NIC パーティショニング設定の例

- VF 上で Linux ボンディングを設定するには、**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 で OVS ブリッジを設定するには、以下の例を使用します。

```
- 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 に OVS ユーザーブリッジを設定するには、VLAN タグを **ovs_user_bridge** パラメーターに適用します。

```
- 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:
          get_param: TenantIpSubnet
    routes:
      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
```

検証

1. VF の数を確認します。

```
[root@overcloud-compute-0 heat-admin]# cat /sys/class/net/p4p1/device/sriov_numvfs
10
[root@overcloud-compute-0 heat-admin]# cat /sys/class/net/p4p2/device/sriov_numvfs
10
```

2. Linux ボンディングを確認します。

```
[root@overcloud-compute-0 heat-admin]# cat /proc/net/bonding/intapi_bond
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: p4p1_1
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
```

```
Slave Interface: p4p1_1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 16:b4:4c:aa:f0:a8
Slave queue ID: 0

Slave Interface: p4p2_1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: b6:be:82:ac:51:98
Slave queue ID: 0
[root@overcloud-compute-0 heat-admin]# cat /proc/net/bonding/st_bond
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: p4p1_3
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: p4p1_3
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 9a:86:b7:cc:17:e4
Slave queue ID: 0

Slave Interface: p4p2_3
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: d6:07:f8:78:dd:5b
Slave queue ID: 0
```

3. OVS ボンディングをリスト表示します。

```
[root@overcloud-compute-0 heat-admin]# ovs-appctl bond/show
---- bond_prov ----
bond_mode: active-backup
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: off
lacp_fallback_ab: false
active slave mac: f2:ad:c7:00:f5:c7(dpdk2)

slave dpdk2: enabled
```

```
  active slave
  may_enable: true

slave dpdk3: enabled
  may_enable: true

---- bond_tnt ----
bond_mode: active-backup
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: off
lacp_fallback_ab: false
active slave mac: b2:7e:b8:75:72:e8(dpdk0)

slave dpdk0: enabled
  active slave
  may_enable: true

slave dpdk1: enabled
  may_enable: true
```

4. OVS の接続を表示します。

```
[root@overcloud-compute-0 heat-admin]# ovs-vsctl show
cec12069-9d4c-4fa8-bfe4-decfdf258f49
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-tenant
        fail_mode: standalone
        Port br-tenant
            Interface br-tenant
                type: internal
        Port bond_tnt
            Interface "dpdk0"
                type: dpdk
                options: {dpdk-devargs="0000:82:02.2"}
            Interface "dpdk1"
                type: dpdk
                options: {dpdk-devargs="0000:82:04.2"}
    Bridge "sriov2"
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "phy-sriov2"
            Interface "phy-sriov2"
                type: patch
                options: {peer="int-sriov2"}
        Port "sriov2"
            Interface "sriov2"
                type: internal
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
```

```
    Port "int-sriov2"
        Interface "int-sriov2"
            type: patch
            options: {peer="phy-sriov2"}
    Port br-int
        Interface br-int
            type: internal
    Port "vhu93164679-22"
        tag: 4
        Interface "vhu93164679-22"
            type: dpdkvhostuserclient
            options: {vhost-server-path="/var/lib/vhost_sockets/vhu93164679-22"}
    Port "vhu5d6b9f5a-0d"
        tag: 3
        Interface "vhu5d6b9f5a-0d"
            type: dpdkvhostuserclient
            options: {vhost-server-path="/var/lib/vhost_sockets/vhu5d6b9f5a-0d"}
    Port patch-tun
        Interface patch-tun
            type: patch
            options: {peer=patch-int}
    Port "int-sriov1"
        Interface "int-sriov1"
            type: patch
            options: {peer="phy-sriov1"}
    Port int-br-vfs
        Interface int-br-vfs
            type: patch
            options: {peer=phy-br-vfs}
Bridge br-vfs
    Controller "tcp:127.0.0.1:6633"
        is_connected: true
    fail_mode: secure
    Port phy-br-vfs
        Interface phy-br-vfs
            type: patch
            options: {peer=int-br-vfs}
    Port bond_prov
        Interface "dpdk3"
            type: dpdk
            options: {dpdk-devargs="0000:82:04.5"}
        Interface "dpdk2"
            type: dpdk
            options: {dpdk-devargs="0000:82:02.5"}
    Port br-vfs
        Interface br-vfs
            type: internal
Bridge "sriov1"
    Controller "tcp:127.0.0.1:6633"
        is_connected: true
    fail_mode: secure
    Port "sriov1"
        Interface "sriov1"
            type: internal
    Port "phy-sriov1"
        Interface "phy-sriov1"
```

```
            type: patch
            options: {peer="int-sriov1"}
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "vxlan-0a0a7315"
            Interface "vxlan-0a0a7315"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="10.10.115.10", out_key=flow,
remote_ip="10.10.115.21"}
    ovs_version: "2.10.0"
```

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

## 6.4. OVS ハードウェアオフロードの設定

OVS ハードウェアオフロードを設定する手順と SR-IOV を設定する手順は、多くの部分が共通です。

手順

1. 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。

```
openstack overcloud roles generate -o roles_data.yaml Controller
Compute:ComputeOvsHwOffload
```

2. 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。

3. 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。

4. neutron が iptables/ハイブリッドのファイアウォールドライバーの実装を使用するように設定するには、**NeutronOVSFirewallDriver: iptables_hybrid** の行を追加します。**NeutronOVSFirewallDriver** の詳細は、オーバークラウドの高度なカスタマイズ の Open vSwitch ファイアウォールの使用 を参照してください。

5. ご自分の環境に合わせて、**physical_network** パラメーターを設定します。

   - 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。
   以下に例を示します。

```
parameter_defaults:
  NeutronOVSFirewallDriver: iptables_hybrid
  ComputeSriovParameters:
    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
```

- **<vendor-id>** は、物理 NIC のベンダー ID に置き換えます。
- **<product-id>** は、NIC VF のプロダクト ID に置き換えます。
- **<address>** は、物理 NIC のアドレスに置き換えます。
  **NovaPCIPassthrough** の設定方法の詳細は**NovaPCIPassthrough** の設定に関するガイドライン を参照してください。

6. デフォルトフィルターのリストに **NUMATopologyFilter** が含まれるようにします。

```
 NovaSchedulerDefaultFilters:
[\'AvailabilityZoneFilter',\'ComputeFilter',\'ComputeCapabilitiesFilter',\'ImagePropertiesFilter',\'Se
rverGroupAntiAffinityFilter',\'ServerGroupAffinityFilter',\'PciPassthroughFilter',\'NUMATopologyF
ilter']
```

7. **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**

- Open vSwitch ハードウェアオフロードを設定する場合には、**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。

8. **overcloud deploy** コマンドに **ovs-hw-offload.yaml** ファイルを追加します。

```
TEMPLATES_HOME=”/usr/share/openstack-tripleo-heat-templates”
CUSTOM_TEMPLATES=”/home/stack/templates”

openstack overcloud deploy --templates \
  -r ${CUSTOM_TEMPLATES}/roles_data.yaml \
  -e ${TEMPLATES_HOME}/environments/ovs-hw-offload.yaml \
  -e ${CUSTOM_TEMPLATES}/network-environment.yaml \
  -e ${CUSTOM_TEMPLATES}/neutron-ovs.yaml
```

### 6.4.1. OVS ハードウェアオフロードの確認

1. PCI デバイスが **switchdev** モードにあることを確認します。

```
# devlink dev eswitch show pci/0000:03:00.0
pci/0000:03:00.0: mode switchdev inline-mode none encap enable
```

2. OVS でオフロードが有効かどうかを確認します。

```
# ovs-vsctl get Open_vSwitch . other_config:hw-offload
“true”
```

## 6.5. OVS ハードウェアオフロードの調整例

最適なパフォーマンスを得るには、追加の設定手順を完了する必要があります。

**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の調整**

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。

手順

- PF レプリゼンターで以下のコマンドを入力し、ホストが利用可能な CPU 数を調整します。$(nproc) を利用可能にする CPU 数に置き換えます。

```
$ sudo ethtool -L enp3s0f0 combined $(nproc)
```

### CPU ピニング

NUMA をまたがる操作によりパフォーマンスが低下するのを防ぐためには、NIC、そのアプリケーション、VF ゲスト、および OVS を同じ NUMA ノード内に配置します。詳細は、インスタンス作成のための **Compute** サービスの設定の コンピュートノードでの CPU ピニングの設定 を参照してください。

## 6.6. OVS ハードウェアオフロードのコンポーネント

Mellanox スマート NIC を使用した OVS HW オフロードのコンポーネントの設定およびトラブルシューティングを行うためのリファレンス。

### Nova

**NUMATopologyFilter** および **DerivePciWhitelistEnabled** パラメーターで **NovaPCIPassthrough** フィルターを使用するように Nova スケジューラーを設定します。OVS HW オフロードを有効にすると、Nova スケジューラーはインスタンスの起動に関して SR-IOV パススルーと同じ様に動作します。

### Neutron

OVS HW オフロードを有効にする場合は、**devlink** cli ツールを使用して NIC e-switch モードを **switchdev** に設定します。**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。

手順

1. **switchdev** 対応の NIC からポートを割り当てるには、 **binding-profile** の値を **capabilities** に設定して 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。

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

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。新規インスタンス作成についての詳しい情報は、SR-IOV 用インスタンスのデプロイ を参照してください。

### OVS

ハードウェアオフロードが設定された環境では、送信された最初のパケットが OVS カーネルパスを通過し、このパケットの移動によりインスタンスの送受信トラフィックに対する ml2 OVS ルールが確立されます。トラフィックストリームのフローが確立されると、OVS はトラフィック制御 (TC) Flower ユーティリティーを使用してこれらのフローを NIC ハードウェアにプッシュします。

手順

1. director を使用して、以下の設定を OVS に適用します。

```
$ sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

2. 再起動して HW オフロードを有効にします。

### トラフィック制御 (TC) サブシステム

**hw-offload** フラグを有効にすると、OVS は TC データパスを使用します。TC Flower は、ハードウェアにデータパスフローを書き込む iproute2 ユーティリティーです。これにより、フローがハードウェアデータパスとソフトウェアデータパスの両方でプログラムされるようになり、冗長性が確保されます。

手順

1. 以下の設定を適用します。明示的に **tc-policy** を設定していない場合、これがデフォルトのオプションです。

```
$ sudo ovs-vsctl set Open_vSwitch . other_config:tc-policy=none
```

2. OVS を再起動します。

### NIC PF および VF ドライバー

mlx5_core は、Mellanox ConnectX-5 NIC の PF ドライバーおよび VF ドライバーです。mlx5_core ドライバーは以下のタスクを実行します。

- ハードウェア上にルーティングテーブルを作成する。
- ネットワークフローの制御を管理する。
- イーサネットスイッチデバイスドライバーモデル **switchdev** を設定する。
- ブロックデバイスを作成する。

手順

- 以下の **devlink** コマンドを使用して、PCI デバイスモードのクエリーを行います。

```
$ sudo devlink dev eswitch set pci/0000:03:00.0 mode switchdev
$ sudo devlink dev eswitch show pci/0000:03:00.0
pci/0000:03:00.0: mode switchdev inline-mode none encap enable
```

### NIC ファームウェア

NIC ファームウェアは以下のタスクを実行します。

- ルーティングテーブルおよびルールを維持する。
- テーブルのパイプラインを修正する。
- ハードウェアリソースを管理する。
- VF を作成する。

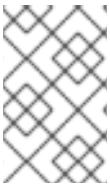
最適なパフォーマンスを得るために、ファームウェアはドライバーと連携します。

NIC ファームウェアは揮発性ではなくリブート後も維持されますが、ランタイム中に設定を変更することができます。

手順

- 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 enp3s0f0 hw-tc-offload on
```



**注記**

ファームウェアを更新された状態に維持します。**Yum** または **dnf** による更新では、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。

## 6.7. OVS ハードウェアオフロードのトラブルシューティング

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。

### OVS HW オフロードデプロイメントでのネットワーク設定

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。

以下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
```

検証された以下のボンディング設定を参照してください。

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

### インターフェイス設定の確認

以下の手順で、インターフェイス設定を確認します。

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。

```
[root@overcloud-computesriov-0 ~]# ethtool -k ens1f0 | grep tc-offload
hw-tc-offload: on
```

### インターフェイスモードの確認

以下の手順で、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。

```
[root@overcloud-computesriov-0 ~]# 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 のオフロード状態の確認

以下の手順で、OVS のオフロード状態を確認します。

- 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。

```
[root@overcloud-computesriov-0 ~]# ovs-vsctl get Open_vSwitch . other_config:hw-offload
"true"
```

### VF レプリゼンターポートの名前の確認

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。

```
root@overcloud-computesriov-0 ~]# cat /etc/udev/rules.d/80-persistent-os-net-config.rules
# 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の調査

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。

```
root@dni-7448-26:~# cl-bcmcmd l2 show

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
…
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。

```
[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。

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

### システムの検証

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。

**制限事項**

OVS 2.11 のオフロードパスでは、フローの接続追跡属性がサポートされないため、HW オフロードで OVS ファイアウォールドライバーを使用することはできません。

## 6.8. HW 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。

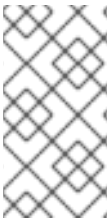
- レプリゼンターポートのリンク状態に加えた変更は、すべて VF のリンク状態にも影響を及ぼします。
- レプリゼンターポートの統計値には、VF の統計も表示されます。

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
```

## 6.9. SR-IOV 用インスタンスのデプロイ

ホストアグリゲートを使用して、ハイパフォーマンス Compute ホストを分離します。ホストアグリゲートの作成およびスケジューリング用の関連フレーバーについての情報は、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、同じコンピュート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 コンピュートノードでの CPU ピニングの設定 を参照してください。

以下の手順を実施して、Single Root I/O Virtualization (SR-IOV) 用インスタンスをデプロイします。

1. フレーバーを作成します。

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

**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、**Configuring the Compute Service for Instance Creation** の Flavor metadata を参照してください。

2. 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
```

3. ポートを作成します。

- SR-IOV Virtual Function (VF) ポートを作成するには、vnic-type に **direct** を使用します。

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

- ハードウェアオフロードを有効にして Virtual Function を作成するには、以下のコマンドを使用します。

```
# openstack port create --network net1 --vnic-type direct --binding-profile '{"capabilities": ["switchdev"]}' sriov_hwoffload_port
```

- vnic-type に **direct-physical** を使用して、単一のインスタンス専用の SR-IOV Physical Function (PF) ポートを作成します。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 net1 --vnic-type direct-physical sriov_port
```

4. インスタンスをデプロイします。

```
# openstack server create --flavor <flavor> --image <image> --nic port-id=<id> <instance name>
```

## 6.10. ホストアグリゲートの作成

パフォーマンスを向上させるために、CPU ピニングおよびヒュージページが設定され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。

1. デプロイメントテンプレートの **parameter_defaults** セクションで heat パラメーター **NovaSchedulerDefaultFilters** を使用して、**AggregateInstanceExtraSpecsFilter** の値およびその他の必要なフィルターを設定することができます。

-

```
  parameter_defaults:
    NovaSchedulerDefaultFilters:
['AggregateInstanceExtraSpecsFilter','AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitie
sFilter','ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPass
throughFilter','NUMATopologyFilter']
```

**注記**

このパラメーターを既存クラスターの設定に追加するには、パラメーターを heat テンプレートに追加し、元のデプロイメント用スクリプトを再度実行します。

2. SR-IOV 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: **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
```

3. フレーバーを作成します。

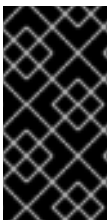
```
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

4. 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>
```

# 第7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**

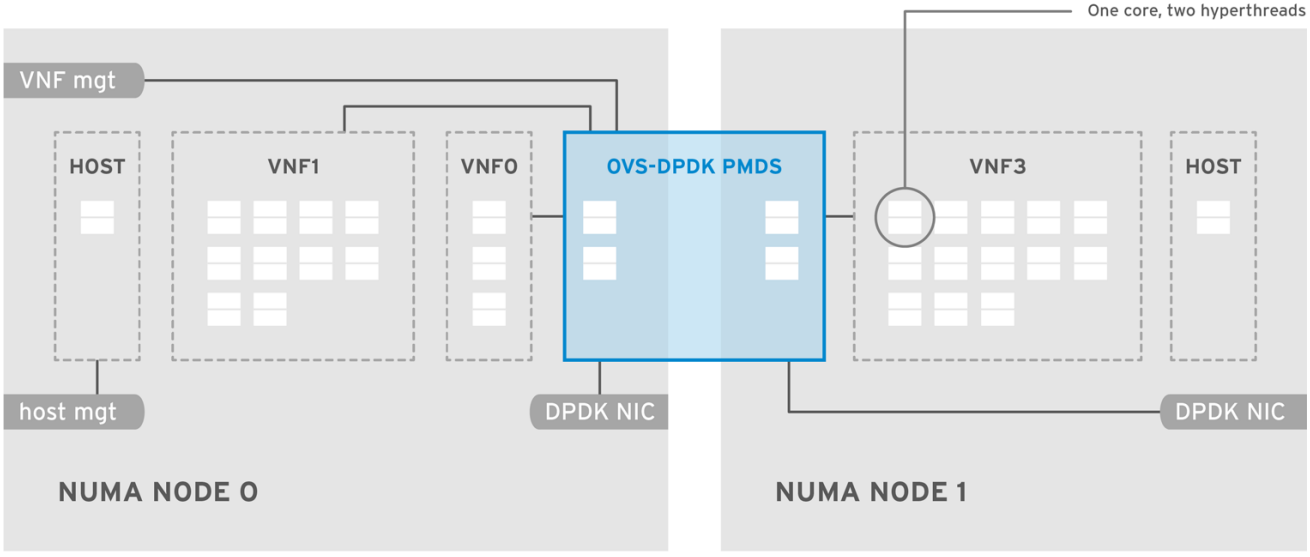
OVS-DPDK および OVS ネイティブファイアウォール (conntrack に基づくステートフルファイアウォール) を使用する場合、追跡することができるのは ICMPv4、ICMPv6、TCP、および UDP プロトコルを使用するパケットだけです。OVS は、その他すべてのネットワークトラフィック種別を無効と識別します。

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

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

OVS-DPDK は、ホスト、ゲスト、および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。




OPENSTACK_464931_0118

**注記**

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

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。

## 7.2. ワークフローと派生パラメーター

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。

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

**注記**

実際の環境に応じて **derive_params_formulas.yaml** を変更することができます。

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

ワークフローの詳細は、ワークフローおよび実行に関するトラブルシューティング を参照してください。

**-p** または **--plan-environment-file** オプションを使用して、ワークブックのリストおよび入力値が含まれるカスタムの **plan_environment.yaml** ファイルを **openstack overcloud deploy** コマンドに追加することができます。得られるワークフローは派生パラメーターをマージしてカスタムの **plan_environment.yaml** に戻し、オーバークラウドのデプロイメントに利用できるようになります。

デプロイメントでの **--plan-environment-file** オプションの使用方法に関する詳しい情報は、Advanced Overcloud Customization の Plan Environment Metadata を参照してください。

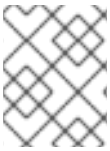
## 7.3. OVS-DPDK の派生パラメーター

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

ワークフローは、OVS-DPDK の以下のパラメーターを自動的に派生することができます。**NovaVcpuPinSet** パラメーターは非推奨となり、専用の固定されたワークフロー用の **NovaComputeCpuDedicatedSet** に置き換えられています。

- IsolCpusList
- KernelArgs
- NovaReservedHostMemory
- NovaComputeCpuDedicatedSet
- OvsDpdkSocketMemory

- OvsPmdCoreList



**注記**

エラーを回避するには、ロール固有のパラメーターにロール固有のタグ付けを設定する必要があります。

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

多くの場合、**OvsDpdkMemoryChannels** のデフォルト値は 4 です。ハードウェアのマニュアルを参照して 1 ソケットあたりのメモリーチャネル数を確認し、デフォルト値をその値で更新してください。

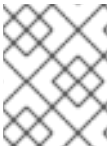
ワークフローパラメーターについての詳しい情報は、「ワークフローを使用した DPDK パラメーターの算出」を参照してください。

## 7.4. OVS-DPDK パラメーターの手動計算

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、コンピュート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**

上記の方法によらず、**derived_parameters.yaml** ワークフローを使用してこれらの値を生成する方法の詳細は、ワークフローと派生パラメーター を参照してください。



**注記**

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

コンピュートノード上の CPU と NUMA ノードを特定する方法の詳細は、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。

### 7.4.1. CPU パラメーター

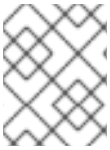
OVS-DPDK では、以下に示す CPU の分割用パラメーターが使用されます。

**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。

```
[heat-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]"}
```

### 7.4.2. メモリーパラメーター

OVS-DPDK は、以下のメモリーパラメーターを使用します。

**OvsDpdkMemoryChannels**

NUMA ノードごとに、CPU 内のメモリーチャネルをマッピングします。**OvsDpdkMemoryChannels** 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"
```

### 7.4.3. 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。

### 7.4.4. その他のパラメーター

**NovaSchedulerDefaultFilters**

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

**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。

**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 分離の詳細については、Red Hat ナレッジベースソリューションOpenStack CPU isolation guidance for RHEL 8 and RHEL 9を参照してください。

### 7.4.5. インスタンスの追加仕様

NFV 環境で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。

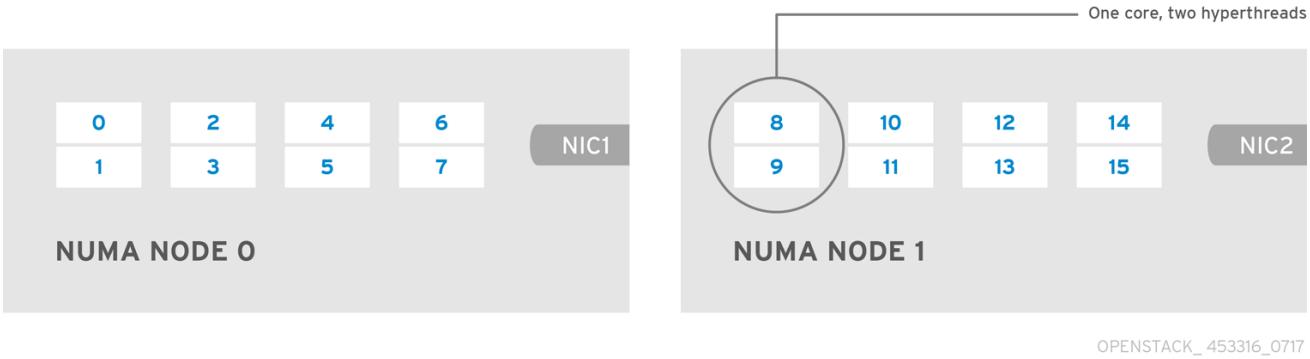
**hw:emulator_threads_policy**

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。

## 7.5. 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。




**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。

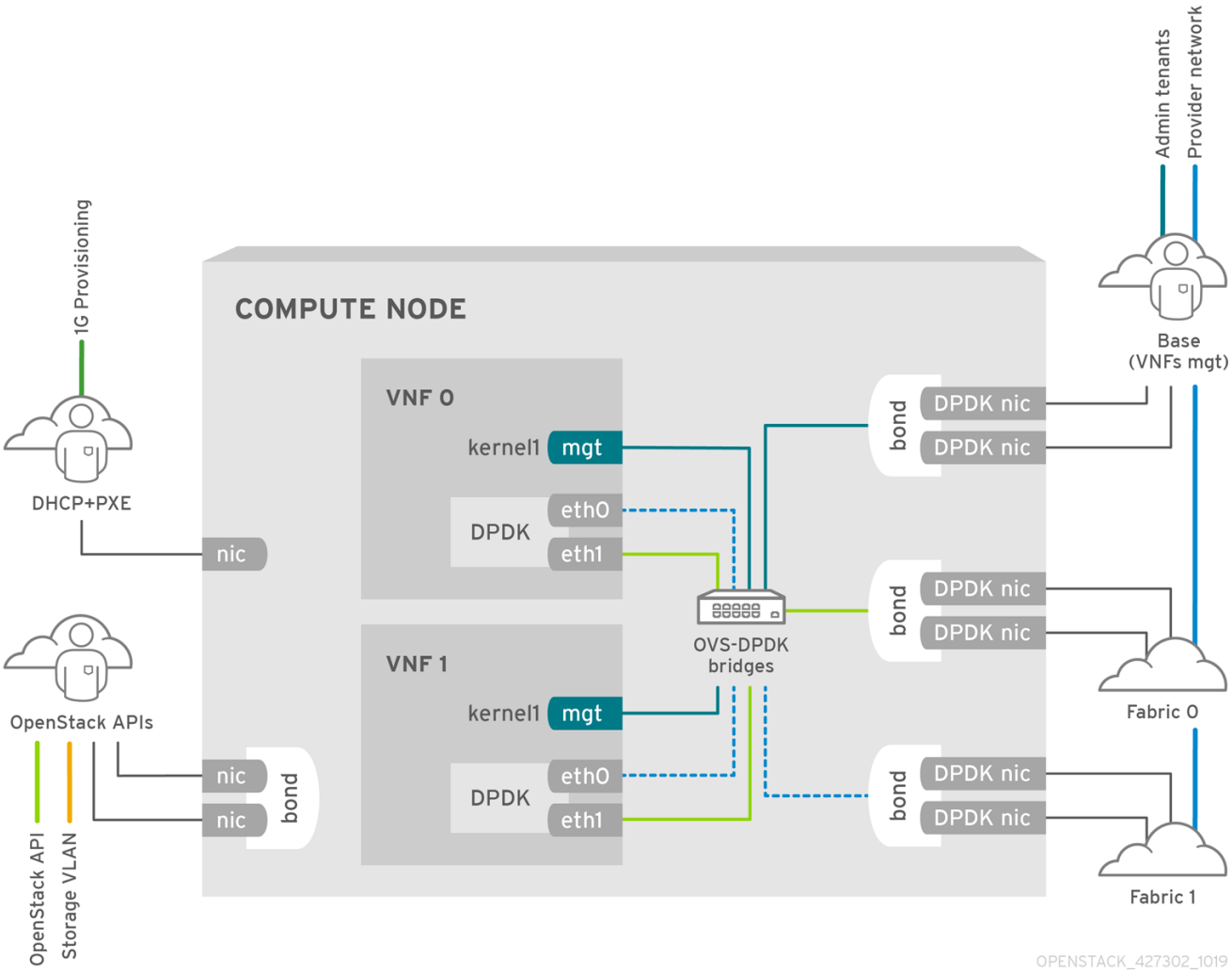
```
OvsPmdCoreList: "2,3,4,5,10,11,12,13"
NovaComputeCpuDedicatedSet: "6,7,14,15"
```

## 7.6. 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。

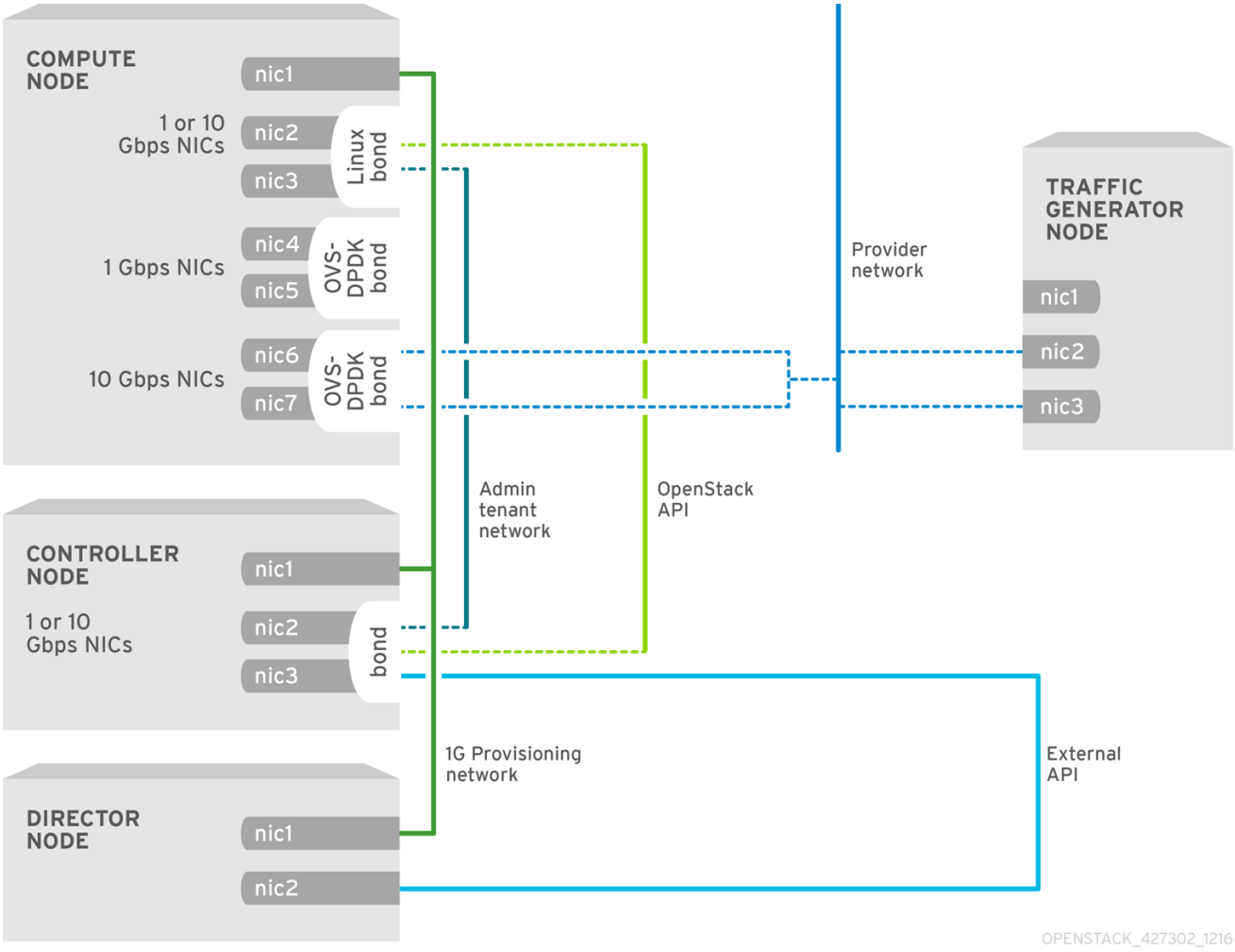
- **mgt** で表される管理インターフェイス
- データプレーンインターフェイス

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。




### 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 を搭載したコンピュート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。


COMPUTE NODE
nic1
1 or 10 Gbps NICs
nic2
nic3
Linux bond
1 Gbps NICs
nic4
nic5
OVS-DPDK bond
10 Gbps NICs
nic6
nic7
OVS-DPDK bond
CONTROLLER NODE
nic1
1 or 10 Gbps NICs
nic2
nic3
bond
DIRECTOR NODE
nic1
nic2
Admin tenant network
OpenStack API
1G Provisioning network
Provider network
External API
TRAFFIC GENERATOR NODE
nic1
nic2
nic3
OPENSTACK_427302_1216

# 第8章 OVS-DPDK デプロイメントの設定

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

オーバークラウドをデプロイする前に、アンダークラウドのインストールと設定が完了している必要があります。詳しくは、Director Installation and Usage を参照してください。

**重要**

OVS-DPDK 向けの OpenStack ネットワークを最適化するには、**network-environment.yaml** ファイルの OVS-DPDK パラメーターの最も適切な値を決定する必要があります。

**注記**

**etc/tuned/cpu-partitioning-variables.conf** の **isolated_cores** や他の値を手動で編集したり変更したりしないでください。これらは director の heat テンプレートにより変更されます。

## 8.1. ワークフローを使用した DPDK パラメーターの算出

**重要**

この機能は、本リリースでは テクノロジープレビュー として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト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。

DPDK 向けの Mistral ワークフローに関する概要は、「ワークフローと派生パラメーター」を参照してください。

### 前提条件

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

### DPDK 向けのワークフローと入力パラメーターの定義

OVS-DPDK ワークフローで指定することができる入力パラメーターの概要を、リストにして以下に示します。

**num_phy_cores_per_numa_node_for_pmd**

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

**huge_page_allocation_percentage**

この入力パラメーターは、**NovaReservedHostMemory** を除く合計メモリーに対して、ヒュージページとして設定可能な必要パーセンテージを指定します。**KernelArgs** パラメーターは、指定した

**huge_page_allocation_percentage** に基づいて計算されたヒュージページを使用して派生されます。このパラメーターは 50 に設定するようにしてください。

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

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

1. **usr/share/openstack-tripleo-heat-templates/plan-samples/plan-environment-derived-params.yaml** ファイルをローカルディレクトリーにコピーし、ご自分の環境に合わせて入力パラメーターを設定します。

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

2. **openstack overcloud deploy** コマンドを実行し、以下の情報を追加します。

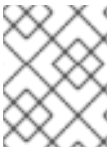
   - **update-plan-only** オプション
   - ロールファイルおよびご自分の環境に固有の全環境ファイル
   - **plan-environment-derived-parms.yaml** ファイル (**--plan-environment-file** オプションの引数)

```
$ openstack overcloud deploy --templates --update-plan-only \
-r /home/stack/roles_data.yaml \
-e /home/stack/<environment-file> \
... _#repeat as necessary_ ...
**-p /home/stack/plan-environment-derived-params.yaml**
```

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

```
Started Mistral Workflow tripleo.validations.v1.check_pre_deployment_validations. Execution ID:
55ba73f2-2ef4-4da1-94e9-eae2fdc35535
Waiting for messages on queue '472a4180-e91b-4f9e-bd4c-1fbdfbcf414f' with no timeout.
Removing the current plan files
Uploading new plan files
Started Mistral Workflow tripleo.plan_management.v1.update_deployment_plan. Execution ID:
7fa995f3-7e0f-4c9e-9234-dd5292e8c722
Plan updated.
Processing templates in the directory /tmp/tripleoclient-SY6RcY/tripleo-heat-templates
Invoking workflow (tripleo.derive_params.v1.derive_parameters) specified in plan-environment file
Started Mistral Workflow tripleo.derive_params.v1.derive_parameters. Execution ID: 2d4572bf-4c5b-
41f8-8981-c84a363dd95b
Workflow execution is completed. result:
```

```
ComputeOvsDpdkParameters:
 IsolCpusList: 1,2,3,4,5,6,7,9,10,17,18,19,20,21,22,23,11,12,13,14,15,25,26,27,28,29,30,31
 KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on
   isolcpus=1,2,3,4,5,6,7,9,10,17,18,19,20,21,22,23,11,12,13,14,15,25,26,27,28,29,30,31
 NovaReservedHostMemory: 4096
 NovaComputeCpuDedicatedSet: 2,3,4,5,6,7,18,19,20,21,22,23,10,11,12,13,14,15,26,27,28,29,30,31
 OvsDpdkMemoryChannels: 4
 OvsDpdkSocketMemory: 1024,1024
 OvsPmdCoreList: 1,17,9,25
```



**注記**

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

### 派生パラメーターを使用したオーバークラウドのデプロイ

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

1. 派生パラメーターをデプロイコマンドの出力から **network-environment.yaml** ファイルにコピーします。

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

**注記**

これらのパラメーターは、特定のロール ComputeOvsDpdk に適用されます。これらのパラメーターをグローバルに適用することはできますが、グローバルパラメーターはロール固有のパラメーターによってオーバーライドされます。

2. ロールファイルおよびご自分の環境に固有の全環境ファイルを使用して、オーバークラウドをデプロイします。

```
openstack overcloud deploy --templates \
-r /home/stack/roles_data.yaml \
-e /home/stack/<environment-file> \
... #repeat as necessary ...
```

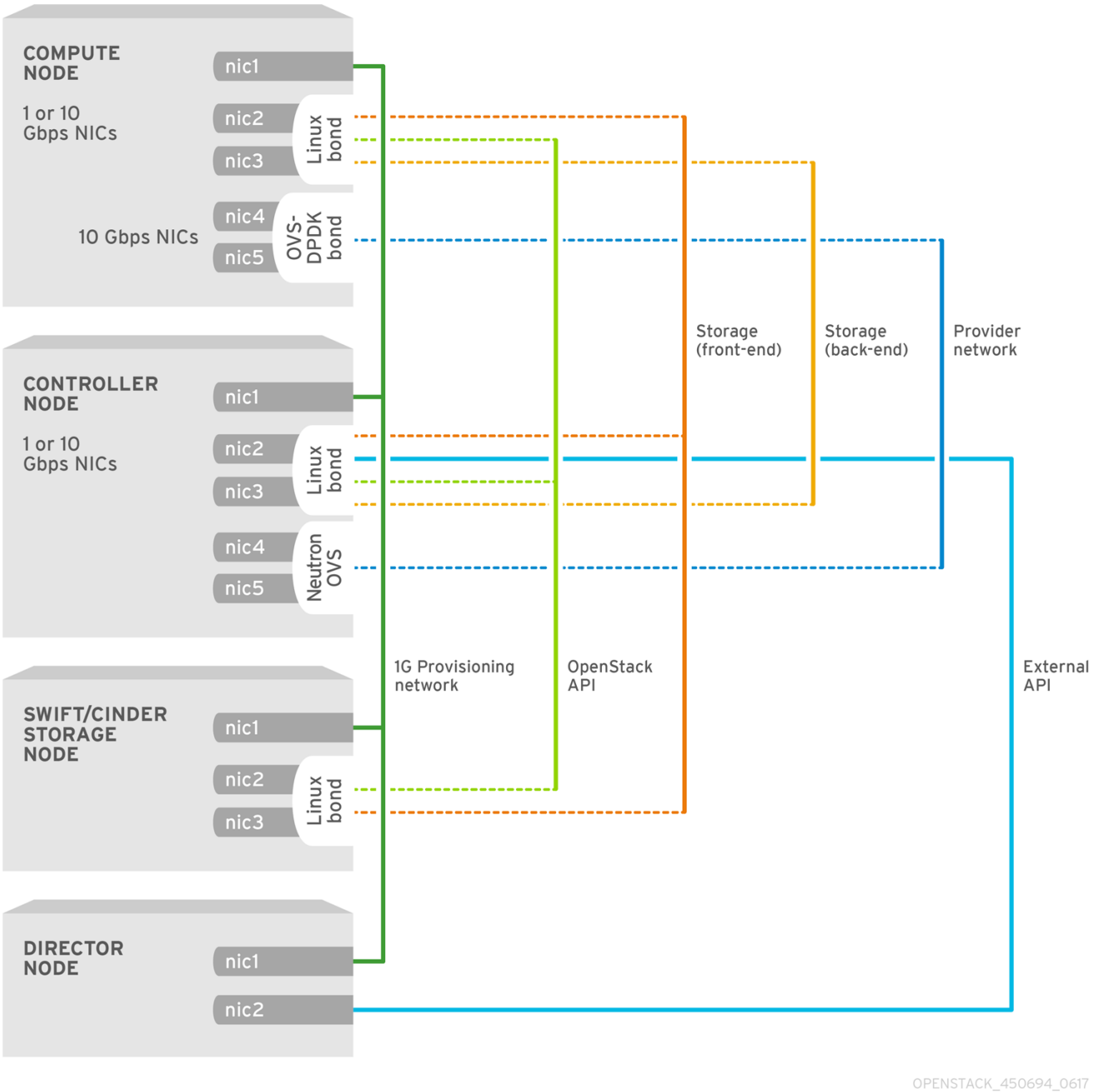
**注記**

Compute、ComputeOvsDpdk、および ComputeSriov ロールが含まれるクラスターでは、ワークフローは ComputeOvsDpdk ロールにだけ式を適用し、Compute または ComputeSriovs には適用しません。

## 8.2. OVS-DPDK のトポロジー

Red Hat OpenStack Platform では、コンポーザブルロール機能を使用してカスタムのデプロイメントロールを作成し、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の詳細は、オーバークラウドの高度なカスタマイズの コンポーザブルサービスとカスタムロール を参照してください。

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




OVS-DPDK を設定するには、以下のタスクを実行します。

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

**注記**

本ガイドでは、CPU の割り当て、メモリーの確保、NIC の設定の例を紹介します。これらは、トポロジーとユースケースによって異なる場合があります。ハードウェアと設定オプションの詳細は、ネットワーク機能仮想化 (NFV) の製品ガイド および*2章ハードウェア要件*を参照してください。

**前提条件**

- OVS 2.10
- DPDK 17
- サポートされている NIC。NFV 向けにサポートされている NIC のリストを表示するには、「テスト済み NIC」を参照してください。

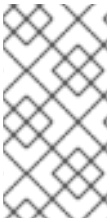
**注記**

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

## 8.3. OVS-DPDK インターフェイスの MTU 値の設定

Red Hat OpenStack Platform は OVS-DPDK 向けにジャンボフレームをサポートしています。ジャンボフレーム用の最大伝送単位 (MTU) 値を設定するには、以下の操作を行う必要があります。

- **network-environment.yaml** ファイルで、ネットワークのグローバル MTU 値を設定する。
- **compute.yaml** ファイルで、物理 DPDK ポートの MTU 値を設定する。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。



**注記**

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

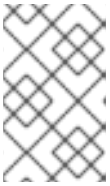
**注記**

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

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

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

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



**注記**

**network-environment.yaml** ファイルの 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。詳しくは、「メモリーパラメーター」を参照してください。

2. **controller.yaml** ファイルで Compute ノードへのブリッジ上の MTU 値を設定します。

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

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

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

```
      members:
        - type: interface
          name: nic5
```

## 8.4. セキュリティーグループの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、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。

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、**NeutronOVSFirewallDriver** パラメーターを **openvswitch** に設定します。フローベースの OVS ファイアウォールドライバーを使用するには、**network-environment.yaml** ファイルの **parameter_defaults** セクションを変更します。

以下に例を示します。

```
parameter_defaults:
  NeutronOVSFirewallDriver: openvswitch
```

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するには、**openstack port set** コマンドを使用します。

以下に例を示します。

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

## 8.5. OVS-DPDK インターフェイス向けのマルチキューの設定

**注記**

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。

手順

- Compute ノード上の OVS-DPDK のインターフェイスに同じ数のキューを設定するには、**compute.yaml** ファイルを変更します。

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

```
name: dpdk1
mtu: 9000
members:
  - type: interface
    name: nic5
```

## 8.6. 既知の制限

NFV 向けに Red Hat OpenStack Platform で OVS-DPDK を設定する場合には、以下の制限事項を確認してください。

- 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。

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

NFV を実装する Red Hat OpenStack Platform デプロイメント向けに 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 コンピュートノードでの 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、専用の物理 CPU で実行される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>
```

## 8.8. 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9章 RED HAT OPENSTACK PLATFORM 環境の調整

## 9.1. 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。

### 9.1.1. エミュレータースレッドをホストする CPU の設定

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。

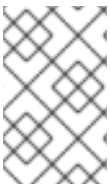
```
parameter_defaults:
  ComputeOvsDpdkParameters:
    NovaComputeCpuSharedSet: "0-1,16-17"
    NovaComputeCpuDedicatedSet: "2-15,18-31"
```

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。

### 9.1.2. エミュレータースレッドが固定されていることの確認

手順

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、特定したホストに heat-admin としてログオンします。

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

## 9.2. NFV ワークロードに向けた RT-KVM の有効化

Red Hat Enterprise Linux 8.2 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 コンピュートノードロール
- 追加の RT-KVM カーネルモジュール
- Compute ノードの自動設定

### 9.2.1. RT-KVM コンピュートノードのプランニング

RT-KVM コンピュートノードには、Red Hat 認定済みサーバーを使用する必要があります。詳しくは、Red Hat Enterprise Linux for Real Time 7 用認定サーバー を参照してください。

RT-KVM 用の **rhel-8-server-nfv-rpms** リポジトリーを有効にしてシステムを最新の状態に維持する方法についての詳細は、Director Installation and Usage の Registering the undercloud and attaching subscriptions を参照してください。

**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。

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

3. デフォルトのイメージをコピーします。

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

4. 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 8.2
```

**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。

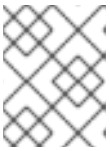
```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms \
--enable=rhel-8-for-x86_64-appstream-eus-rpms \
--enable=rhel-8-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-8-x86_64-rpms \
--enable=openstack-16.1-for-rhel-8-x86_64-rpms \
--enable=rhel-8-for-x86_64-nfv-rpms \
--enable=advanced-virt-for-rhel-8-x86_64-rpms \
--enable=fast-datapath-for-rhel-8-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。

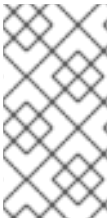
#### RT-KVM Compute ノード上での BIOS 設定の変更

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

これらの設定に関する説明と、無効化の影響については、Red Hat Enterprise Linux for Real Time チューニングガイドの BIOS パラメーターの設定 を参照してください。BIOS 設定の変更方法に関する詳しい情報は、ハードウェアの製造会社のドキュメントを参照してください。

### 9.2.2. RT-KVM 対応の OVS-DPDK の設定



**注記**

OVS-DPDK 用の OpenStack ネットワークを最適化するには、**network-environment.yaml** ファイルに設定する OVS-DPDK パラメーターの最適な値を判断する必要があります。詳しくは、「ワークフローを使用した DPDK パラメーターの算出」を参照してください。

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

**ComputeOvsDpdkRT** ロールを使用して、real-time の Compute イメージ用の Compute ノードを指定します。

**ComputeOvsDpdkRT** ロール向けに **roles_data.yaml** を生成します。

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

#### 9.2.2.2. OVS-DPDK パラメーターの設定

**重要**

デプロイメントを最適化するには、**network-environment.yaml** ファイルの OVS-DPDK パラメーターの最適な値を判断します。詳細は、「ワークフローを使用した DPDK パラメーターの算出」 を参照してください。

1. **resource_registry** セクションに、使用する OVS-DPDK ロールの NIC 設定を追加します。

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

2. **parameter_defaults** セクションで、OVS-DPDK および RT-KVM のパラメーターを設定します。

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

#### 9.2.2.3. オーバークラウドのデプロイ

ML2-OVS 向けのオーバークラウドをデプロイします。

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy \
--templates \
-r /home/stack/ospd-16-vlan-dpdk-ctlplane-bonding-rt/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs-dpdk.yaml \
-e /home/stack/ospd-16-vxlan-dpdk-data-bonding-rt-hybrid/containers-prepare-parameter.yaml \
-e /home/stack/ospd-16-vxlan-dpdk-data-bonding-rt-hybrid/network-environment.yaml
```

### 9.2.3. 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 99 4096 20 4
# openstack flavor set --property hw:cpu_policy=dedicated 99
# openstack flavor set --property hw:cpu_realtime=yes 99
# openstack flavor set --property hw:mem_page_size=1GB 99
# openstack flavor set --property hw:cpu_realtime_mask="^0-1" 99
# openstack flavor set --property hw:cpu_emulator_threads=isolate 99
```

2. RT-KVM インスタンスを起動します。

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

## 9.3. 信頼済み VIRTUAL FUNCTION

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。

### 9.3.1. Virtual Function と Physical Function 間の信頼の設定

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。

### 9.3.2. 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
```

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。

## 9.4. 受信/送信キューサイズの設定

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**

- 受信キューサイズを設定するには、libvirt v2.3 および QEMU v2.7 が必要です。
- 送信キューサイズを設定するには、libvirt v3.7 および QEMU v2.10 が必要です。

**手順**

- 受信および送信キューサイズを増やすには、該当する director ロールの **parameter_defaults:** セクションに以下の行を追加します。ComputeOvsDpdk ロールにおける例を以下に示します。

```
parameter_defaults:
  ComputeOvsDpdkParameters:
    -NovaLibvirtRxQueueSize: 1024
    -NovaLibvirtTxQueueSize: 1024
```

**テスト**

- nova.conf ファイルで、受信キューサイズおよび送信キューサイズの値を確認することができます。

```
[libvirt]
rx_queue_size=1024
tx_queue_size=1024
```

- コンピュートホストの 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、受信キューサイズおよび送信キューサイズの値を確認することができます。

```
<devices>
  <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>
</devices>
```

受信キューサイズおよび送信キューサイズの値を検証するには、KVM ホストで以下のコマンドを使用します。

```
$ virsh dumpxml <vm name> | grep queue_size
```

- パフォーマンスの向上を確認することができます (例: 3.8 mpps/コアのレートでフレーム損失なし)。

## 9.5. 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** を有効にしていること

**手順**

- 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。
- 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>
```

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
```

- 以下の例では、**eno2** という名前のデバイスの物理ネットワークを NUMA 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
```

以下の heat テンプレートの例で、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
```

### NUMA 対応 vSwitch のテスト

- ファイル /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf の設定を確認します。

```
[neutron_physnet_tenant]
numa_nodes=1
[neutron_tunnel]
numa_nodes=1
```

- **lscpu** コマンドで新たな設定を確認します。

```
$ lscpu
```

- 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。

### 既知の制限事項

- 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。

## 9.6. NFVI 環境における QUALITY OF SERVICE (QOS) の設定

QoS の設定については、Configuring Quality of Service (QoS) policies を参照してください。サポートされる QoS ルールは、以下の種別に限定されます。

- 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**

## 9.7. HCI および DPDK を使用するオーバークラウドのデプロイ

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) についての詳しい情報は、ハイパーコンバージドインフラストラクチャーガイド を参照してください。

前提条件

- Red Hat OpenStack Platform 16.1
- Red Hat Ceph Storage 4 の最新バージョン
- **rhceph-4-tools-for-rhel-8-x86_64-rpms** リポジトリーで提供される ceph-ansible 4 の最新バージョン

手順

1. アンダークラウドに **ceph-ansible** をインストールします。

```
$ sudo yum install ceph-ansible -y
```

2. ComputeHCI ロール用に **roles_data.yaml** ファイルを生成します。

```
$ openstack overcloud roles generate -o ~/<templates>/roles_data.yaml Controller \
 ComputeHCIOvsDpdk
```

3. **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。フレーバー作成についての詳細は、オーバークラウドの高度なカスタマイズの 新規ロールの作成 を参照してください。

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。

```
# time openstack overcloud deploy --templates \
 --timeout 360 \
 -r ~/<templates>/roles_data.yaml \
 -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.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>
```

### 9.7.1. 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 |

| | NUMA-0 | NUMA-1 |
|---|---|---|
| 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 |

### 9.7.2. ceph 設定ファイルの例

```
parameter_defaults:
  CephPoolDefaultSize: 3
  CephPoolDefaultPgNum: 64
  CephPools:
    - {"name": backups, "pg_num": 128, "pgp_num": 128, "application": "rbd"}
    - {"name": volumes, "pg_num": 256, "pgp_num": 256, "application": "rbd"}
    - {"name": vms, "pg_num": 64, "pgp_num": 64, "application": "rbd"}
    - {"name": images, "pg_num": 32, "pgp_num": 32, "application": "rbd"}
  CephConfigOverrides:
    osd_recovery_op_priority: 3
    osd_recovery_max_active: 3
    osd_max_backfills: 1
  CephAnsibleExtraConfig:
    nb_retry_wait_osd_up: 60
    delay_wait_osd_up: 20
    is_hci: true
    # 3 OSDs * 4 vCPUs per SSD = 12 vCPUs (list below not used for VNF)
    ceph_osd_docker_cpuset_cpus: "32,34,36,38,40,42,76,78,80,82,84,86" # 1
    # cpu_limit 0 means no limit as we are limiting CPUs with cpuset above
    ceph_osd_docker_cpu_limit: 0                                       # 2
    # numactl preferred to cross the numa boundary if we have to
    # but try to only use memory from numa node0
    # cpuset-mems would not let it cross numa boundary
    # lots of memory so NUMA boundary crossing unlikely
    ceph_osd_numactl_opts: "-N 0 --preferred=0"                        # 3
  CephAnsibleDisksConfig:
    osds_per_device: 1
    osd_scenario: lvm
    osd_objectstore: bluestore
    devices:
      - /dev/sda
      - /dev/sdb
      - /dev/sdc
```

以下のパラメーターを使用して、ceph OSD プロセスの CPU リソースを割り当てます。ワークロードおよびこのハイパーコンバージド環境のハードウェアに基づいて、値を調整します。

1. ceph_osd_docker_cpuset_cpus: SSD ディスクの場合は、OSD ごとに 4 つの CPU スレッドを割り当てます。HDD ディスクの場合は、OSD ごとに 1 つの CPU を割り当てます。ceph に関連付けられた NUMA ノードからのコアおよびシブリングスレッド、ならびに 3 つのリスト **NovaComputeCpuDedicatedSet** および **OvsPmdCoreList** に記載されていない CPU のリストを含めます。

2. ceph_osd_docker_cpu_limit: ceph OSD を **ceph_osd_docker_cpuset_cpus** からの CPU リストにピニングするには、この値を **0** に設定します。

3 ceph_osd_numactl_opts: 念のため、複数 NUMA にまたがる操作用にこの値を **preferred** に設定します。

### 9.7.3. 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
    NumDpdkInterfaceRxQueues: 1
```

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ついての詳しい情報は、ovsdpdk パラメーター 関する説明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。

### 9.7.4. 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。

### 9.7.5. HCI-DPDK デプロイメントに推奨される設定

表9.1 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<br>デバイスごとの OSD 数: 4<br>デバイスごとの仮想 CPU 数: 3 |
| SSD | メモリー : OSD ごとに 5 GB<br>デバイスごとの OSD 数: 1<br>デバイスごとの仮想 CPU 数: 4 |
| HDD | メモリー : OSD ごとに 5 GB<br>デバイスごとの OSD 数: 1<br>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

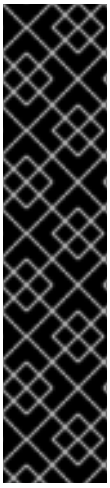
DPDK プロバイダーネットワークの以下の機能には、別の NUMA ノードを割り当てます。

- NIC

- PMD CPU
- ソケットメモリー

# 第10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
```

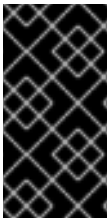
**OS::TripleO::Services::Tuned** をコメントアウトまたは削除した場合は、要件に合わせて **TunedProfileName** パラメーターを設定することができます (例: **"cpu-partitioning"**)。**OS::TripleO::Services::Tuned** 行をコメントアウトまたは削除せずに再デプロイすると、**TunedProfileName** パラメーターには、設定した他の値ではなく **"throughput-performance"** のデフォルト値が設定されます。

## 10.1. 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。Red Hat OpenStack Platform でのロール作成に関する情報は、オーバークラウドの高度なカスタマイズ を参照してください。以下の例で使用する特定のロールの詳細については、roles_data.yaml を参照してください。

## 10.2. OVS-DPDK パラメーターの設定



**重要**

OVS-DPDK 用の OpenStack ネットワークを最適化するには、**network-environment.yaml** ファイルに設定する OVS-DPDK パラメーターの最適な値を判断する必要があります。詳しくは、ワークフローを使用した DPDK パラメーターの算出 を参照してください。

1. **resource_registry** セクションに OVS-DPDK 用のカスタムリソースを追加します。

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

2. **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'
```

3. **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
```

4. **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。

5. 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"
```

## 10.3. 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
```

## 10.4. 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。

## 10.5. オーバークラウドのデプロイ

1. **overcloud_deploy.sh** スクリプトを実行します。

# 第11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ついての詳しい情報は、13 から 16.1 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12章 NFV のパフォーマンス

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コンピュート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。

CPU と NUMA トポロジーの概要は、NFV のパフォーマンスの考慮事項 および エミュレータースレッドの設定 を参照してください。

# 第13章 その他の参考資料

以下の表には、参考となるその他の Red Hat ドキュメントのリストを記載しています。

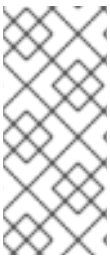
Red Hat OpenStack Platform のドキュメントスイートは Red Hat OpenStack Platform の製品ドキュメントスイート から参照してください。

表13.1 参考資料リスト

| コンポーネント | 参考情報 |
| --- | --- |
| Red Hat Enterprise Linux | Red Hat OpenStack Platform は Red Hat Enterprise Linux 8.0 でサポートされています。Red Hat Enterprise Linux のインストールに関する情報は、Red Hat Enterprise Linux のドキュメント から対応するインストールガイドを参照してください。 |
| Red Hat OpenStack Platform | OpenStack のコンポーネントとそれらの依存関係をインストールするには、Red Hat OpenStack Platform director を使用します。director は、基本的な OpenStack 環境をアンダークラウドとして使用して、最終的なオーバークラウド内の OpenStack ノードをインストール、設定、および管理します。アンダークラウドのインストールには、デプロイするオーバークラウドに必要な環境に加えて、追加のホストマシンが 1 台必要です。詳しい手順は、director のインストールと使用方法 を参照してください。<br><br>ネットワークの分離、ストレージ設定、SSL 通信など、Red Hat OpenStack Platform director を使用して Red Hat OpenStack Platform エンタープライズ環境に高度な機能を設定する方法、および一般的な設定方法については、オーバークラウドの高度なカスタマイズ を参照してください。 |
| NFV のドキュメント | NFV の概念に関する俯瞰的な情報は、ネットワーク機能仮想化 (NFV) の製品ガイド を参照してください。 |

# 付録A DPDK SRIOV YAML ファイルのサンプル

本項では、同じコンピュートノードに 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。

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

### A.1.1. roles_data.yaml

1. **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)。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 controler 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
```

### A.1.2. network-environment-overrides.yaml

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

# Customize all these values to match the local environment
parameter_defaults:
  # The tunnel type for the project network (vxlan or gre). Set to '' to disable tunneling.
  NeutronTunnelTypes: 'vxlan'
  # The project network type for Neutron (vlan or vxlan).
  NeutronNetworkType: 'vxlan,vlan'
  # The OVS logical->physical bridge mappings to use.
  NeutronBridgeMappings: 'access:br-access,dpdk-mgmt:br-link0'
  # The Neutron ML2 and OpenVSwitch vlan mapping range to support.
  NeutronNetworkVLANRanges: 'access:423:423,dpdk-mgmt:134:137,sriov-1:138:139,sriov-
2:138:139'
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["10.46.0.31","10.46.0.32"]
  # Nova flavor to use.
  OvercloudControllerFlavor: controller
  OvercloudComputeOvsDpdkSriovFlavor: computeovsdpdksriov
  # Number of nodes to deploy.
  ControllerCount: 3
  ComputeOvsDpdkSriovCount: 2
  # 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
  SshServerOptions:
    UseDns: 'no'
  # Enable log level DEBUG for supported components
  Debug: True

  ControllerHostnameFormat: 'controller-%index%'
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeOvsDpdkSriovHostnameFormat: 'computeovsdpdksriov-%index%'
  ComputeOvsDpdkSriovSchedulerHints:
```

```
    'capabilities:node': 'computeovsdpdksriov-%index%'

  # From Rocky live migration with NumaTopologyFilter disabled by default
  # https://bugs.launchpad.net/nova/+bug/1289064
  NovaEnableNUMALiveMigration: true

  ############################
  # OVS DPDK configuration #
  ############################

  # In the future, most parameters will be derived by mistral plan.
  # Currently mistral derive parameters is blocked:
  # https://bugzilla.redhat.com/show_bug.cgi?id=1777841
  # https://bugzilla.redhat.com/show_bug.cgi?id=1777844
  ComputeOvsDpdkSriovParameters:
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt
intel_iommu=on isolcpus=2-19,22-39"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList: "2-19,22-39"
    NovaComputeCpuDedicatedSet: ['2-10,12-17,19,22-30,32-37,39']
    NovaReservedHostMemory: 4096
    OvsDpdkSocketMemory: "1024,3072"
    OvsDpdkMemoryChannels: "4"
    OvsPmdCoreList: "11,18,31,38"
    NovaComputeCpuSharedSet: [0,20,1,21]
    # When using NIC partioning on SR-IOV enabled setups, 'derive_pci_passthrough_whitelist.py'
    # script will be executed which will override NovaPCIPassthrough.
    # No option to disable as of now - https://bugzilla.redhat.com/show_bug.cgi?id=1774403
    NovaPCIPassthrough:
      - address: "0000:19:0e.3"
        trusted: "true"
        physical_network: "sriov1"
      - address: "0000:19:0e.0"
        trusted: "true"
        physical_network: "sriov-2"
    # NUMA aware vswitch
    NeutronPhysnetNUMANodesMapping: {dpdk-mgmt: [0]}
    NeutronTunnelNUMANodes: [0]
    NeutronPhysicalDevMappings:
    - sriov1:enp6s0f2
    - sriov2:enp6s0f3

  ##############################
  #  Scheduler configuration #
  ##############################
  NovaSchedulerDefaultFilters:
    - "AvailabilityZoneFilter"
    - "ComputeFilter"
    - "ComputeCapabilitiesFilter"
    - "ImagePropertiesFilter"
    - "ServerGroupAntiAffinityFilter"
    - "ServerGroupAffinityFilter"
    - "PciPassthroughFilter"
    - "NUMATopologyFilter"
    - "AggregateInstanceExtraSpecsFilter"
```

### A.1.3. controller.yaml

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure VLANs for the controller role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal_api network
    type: string
  InternalApiInterfaceRoutes:
    default: []
    description: >
      Routes for the internal_api network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage_mgmt network
    type: string
  StorageMgmtInterfaceRoutes:
    default: []
    description: >
      Routes for the storage_mgmt network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
```

```
attribute.
    type: json
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  TenantInterfaceRoutes:
    default: []
    description: >
      Routes for the tenant network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  ManagementIpSubnet: # Only populated when including environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  ManagementInterfaceRoutes:
    default: []
    description: >
      Routes for the management network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  BondInterfaceOvsOptions:
    default: bond_mode=active-backup
    description: >-
      The ovs_options string for the bond interface. Set things like lacp=active and/or
bond_mode=balance-slb using this option.
    type: string
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: 20
    description: Vlan ID for the internal_api network traffic.
    type: number
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage_mgmt network traffic.
    type: number
  TenantNetworkVlanID:
    default: 50
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
```

```
  ExternalInterfaceDefaultRoute:
    default: 10.0.0.1
    description: default route for the external network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is automatically resolved from the
ctlplane subnet's cidr
      attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: >-
      The default route of the control plane network. (The parameter is automatically resolved from the
ctlplane subnet's
      gateway_ip attribute.)
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations). If not set the
nameservers configured in the
      ctlplane subnet's dns_nameservers attribute will be used.
    type: comma_delimited_list
  EC2MetadataIp:
    default: ''
    description: >-
      The IP address of the EC2 metadata server. (The parameter is automatically resolved from the
ctlplane subnet's host_routes
      attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the network. (The parameter is automatically resolved from the ctlplane network's mtu
attribute.)
    type: number
  StorageMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the Storage network.
    type: number
  StorageMgmtMtu:
```

```
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the StorageMgmt network.
    type: number
  InternalApiMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the InternalApi network.
    type: number
  TenantMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the Tenant network.
    type: number
  ExternalMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the External network.
    type: number
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
              - type: interface
                name: nic1
                use_dhcp: false
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop:
                    get_param: EC2MetadataIp

              - type: ovs_bridge
                name: br-link0
                use_dhcp: false
```

```
            mtu: 9000
            members:
            - type: interface
              name: nic2
              mtu: 9000

            - type: vlan
              vlan_id:
                get_param: TenantNetworkVlanID
              mtu: 9000
              addresses:
              - ip_netmask:
                  get_param: TenantIpSubnet

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

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

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

          - type: ovs_bridge
            name: br-access
            use_dhcp: false
            mtu: 9000
            members:
            - type: interface
              name: nic3
              mtu: 9000
            - type: vlan
              vlan_id:
                get_param: ExternalNetworkVlanID
              mtu: 9000
              addresses:
              - ip_netmask:
                  get_param: ExternalIpSubnet
              routes:
              - default: true
                next_hop:
                  get_param: ExternalInterfaceDefaultRoute
outputs:
  OS::stack_id:
```

```
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

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

```
heat_template_version: rocky

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

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal_api network
    type: string
  InternalApiInterfaceRoutes:
    default: []
    description: >
      Routes for the internal_api network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  StorageMgmtIpSubnet:
    default: ''
```

```
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ManagementIpSubnet: # Only populated when including environments/network-management.yaml
  default: ''
  description: IP address/subnet on the management network
  type: string
ManagementInterfaceRoutes:
  default: []
  description: >
    Routes for the management network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
BondInterfaceOvsOptions:
  default: 'bond_mode=active-backup'
  description: The ovs_options string for the bond interface. Set things like
               lacp=active and/or bond_mode=balance-slb using this option.
  type: string
ExternalNetworkVlanID:
  default: 10
  description: Vlan ID for the external network traffic.
  type: number
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgmt network traffic.
  type: number
TenantNetworkVlanID:
```

```
    default: 50
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  ExternalInterfaceDefaultRoute:
    default: '10.0.0.1'
    description: default route for the external network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  EC2MetadataIp:
    default: ''
    description: The IP address of the EC2 metadata server. (The parameter
      is automatically resolved from the ctlplane subnet's host_routes attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic. JSON route e.g. [{'destination':'10.0.0.0/16',
'nexthop':'10.0.0.1'}] Unless
      the default is changed, the parameter is automatically resolved from the subnet host_routes
attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the network. (The parameter is automatically resolved from the ctlplane network's mtu
attribute.)
    type: number
  StorageMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the Storage network.
```

```
    type: number
  InternalApiMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the InternalApi network.
    type: number
  TenantMtu:
    default: 1500
    description: >-
      The maximum transmission unit (MTU) size(in bytes) that is guaranteed to pass through the data
path of the segments
      in the Tenant network.
    type: number

resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
              - type: interface
                name: nic1
                use_dhcp: false
                defroute: false

              - type: interface
                name: nic2
                use_dhcp: false
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop:
                    get_param: EC2MetadataIp
                - default: true
                  next_hop:
                    get_param: ControlPlaneDefaultRoute

              - type: linux_bond
                name: bond_api
                bonding_options: mode=active-backup
                use_dhcp: false
                dns_servers:
                  get_param: DnsServers
```

```
      members:
      - type: interface
        name: nic3
        primary: true
      - type: interface
        name: nic4

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

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

    - type: ovs_user_bridge
      name: br-link0
      use_dhcp: false
      ovs_extra:
      - str_replace:
          template: set port br-link0 tag=_VLAN_TAG_
          params:
            _VLAN_TAG_:
              get_param: TenantNetworkVlanID
      addresses:
      - ip_netmask:
          get_param: TenantIpSubnet
      members:
      - type: ovs_dpdk_bond
        name: dpdkbond0
        mtu: 9000
        rx_queue: 2
        members:
        - type: ovs_dpdk_port
          name: dpdk0
          members:
          - type: interface
            name: nic7
        - type: ovs_dpdk_port
          name: dpdk1
          members:
          - type: interface
            name: nic8

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

### A.1.5. overcloud_deploy.sh

```
#!/bin/bash

THT_PATH='/home/stack/ospd-16-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid'

openstack overcloud deploy \
--templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs-dpdk.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml \
-e /home/stack/containers-prepare-parameter.yaml \
-r $THT_PATH/roles_data.yaml \
-e $THT_PATH/network-environment-overrides.yaml \
-n $THT_PATH/network-data.yaml
```