VPC-DI 設置上の注意事項

このガイドでは、VPC-DIのコンポーネントが正しくインストールされ、市販（COTS）サーバ上の仮想マシン（VM）上で実行されていることを前提としています。この章では、インストールプロセスに役立ついくつかのインストールの注意事項について説明します。

起動パラメータファイルの作成

起動パラメータファイルは、起動する前にStarOSに設定項目を渡すための手段を提供します。通常、パラメータは、StarOSを正常にロードし、仮想スロット番号、VMのタイプ、NICの割り当て、ネットワークボンディングの設定などの項目を指定するために必要です。

デフォルトでは、VPC-DIはハイパーバイザによって提供される順序でvNICインターフェイスを割り当てます。特定の順序に従って手動でvNICsを設定するには、起動パラメータファイルを作成する必要があります。また、VNFMインターフェイスを有効にする場合も、起動パラメータファイルを作成する必要があります。

起動パラメータは複数の方法で送信され、すべての方式が同じパラメータ名と使用法を使用します。ブートパラメータファイルの最初の場所は、最初のVMドライブの最初のパーティション（たとえば、*/boot1/param.cfg*）にあります。検索された2番目の場所は、仮想CD-ROMドライブである設定ドライブ上にあります。OpenStackを使用している場合は、ターゲットの起動パラメータのファイル名を*staros_param.cfg*として指定します。OpenStackを使用していない場合は、ルートディレクトリに*staros_param.cfg*を使用してISOイメージを作成し、このISOをVMの最初の仮想CD-ROMドライブに接続します。

VMが起動すると、最初にCFEと呼ばれるブート前環境でparam.cfgファイルが解析されます。VMがLinuxを開始すると、仮想CD-ROMドライブにアクセスして、*staros_param.cfg*ファイルを解析します。*/boot1/param.cfg*ファイルに保存されている値と競合がある場合、staros_param.cfgのパラメータが優先されます。

起動パラメータファイルを作成しない場合は、デフォルトファイルが使用されます。起動パラメータファイルを作成する場合は、起動パラメータの設定 （7 ページ） で説明されているすべてのパラメータを定義する必要があります。

起動パラメータファイルのフォーマット

起動パラメータファイルの構造は次のとおりです。

VARIABLE_NAME = VALUE

行区切りとして改行を使用して、1 行につき 1 つの変数を指定します（UNIX テキストファイル形式）。変数名と値は大文字と小文字が区別されません。無効な値は無視され、エラー通知が VM コンソールに表示されます。変数の値が重複している場合（同じ変数名に 2 つの異なる値が指定されている場合）、最後に定義された値が使用されます。

数値にゼロを埋める必要はありません。たとえば、PCI_ID は 0:1:1.0 の場合、0000:01:01.0 と同様に扱われます。

ネットワーク インターフェイス ロール

ネットワーク インターフェイスは、VM が CF または SF に使用されているかどうかに応じて、特定の役割を果たします。

すべてのシステム VM には、DI 内部ネットワークへのネットワーク インターフェイス接続が備わっています。このネットワークは、VPC-DI インスタンス内のすべての VM 同士をリンクします。このネットワークは VPC-DI インスタンスに非公開とする必要があります。また、このネットワークはシステムソフトウェアによって設定されます。

仮想ネットワーク機能（VNF）マネージャ（VNFM）が存在する場合は、それに接続されているネットワーク インターフェイスを設定するオプションがすべての VM に備わっています。このインターフェイスは、DHCP またはスタティック IP 割り当てを使用して設定でき、VNFM 以上のレベルのオーケストレータとの通信に使用されます。このインターフェイスは、メインアプリケーションが起動する前に有効になります。

CF では、1 つの追加インターフェイスが管理ネットワーク インターフェイスに接続します。このインターフェイスは通常、StarOS で設定され、Day 0 設定に含まれている必要があります。管理インターフェイスは、メインの StarOS 設定ファイルを使用してスタティックアドレスの割り当てをサポートします。

SF では、追加の 0 ～ 12 個のネットワーク インターフェイスがサービスポートとして機能します。これらのインターフェイスは、StarOS によって設定されます。通常、これらのポートは VNF インフラストラクチャ（VNFI）のトランクポートとして設定されます。

表 *1:* ネットワーク インターフェイス ロール

インターフェイス ロール	説明
DI_INTERFACE	VM のすべてのタイプに必要な DI 内部ネットワークへのインターフェイス

インターフェイス ロール	説明
MGMT_INTERFACE	CF VM 上の管理ポートへのインターフェイス
SERVICE#_INTERFACE	SF VM のサービスポート番号 #（# は 1 ～ 12）
VNFM_INTERFACE	VNFM またはオーケストレータへのオプションのネットワーク インターフェイス（VM のすべてのタイプで有効）

（注） VIRTIO のインターフェイスは DI_INTERFACE ロールと SERVICE#_INTERFACE ロールに使用できますが、推奨されません。

ネットワーク インターフェイス ID

デフォルトでは、VPC-DI VM によって検出された最初の NIC には、DI 内部ネットワークロールが割り当てられます。追加ポートは、SF の CF またはサービスポートの管理インターフェイスとして機能します。デフォルトでは、インターフェイスは VNFM インターフェイスとして使用されません。

VPC-DIハイパーバイザによって提供される順序でvNICインターフェイスを割り当てます。ハイパーバイザの CLI や GUI にリストされている vNIC の順序が、ハイパーバイザが VM に提供する方法と同じであることは保証できません。

VPC-DI が vNIC を検出する順序は PCI バスの列挙順に従い、準仮想デバイスも PCI バスで表されます。PCI バスは、同じレベルの追加デバイスの前にブリッジが探索される深さ優先の方法で列挙されます。すべてのネットワーク インターフェイスのタイプが同じ場合、PCI トポロジを認識するだけで vNIC の正しい順序を取得できます。ネットワーク インターフェイスのタイプが異なる場合、その順序は PCI トポロジに加えて VM 内のデバイスドライバのロード順序に依存します。デバイスドライバのロード順序は、ソフトウェアのリリース順と同じである保証はありませんが、一般的には準仮想デバイスがパススルーデバイスよりも優先されます。

NIC を識別するために使用できる方式はいくつかあります。

- MAC アドレス：インターフェイスの MAC アドレス
- 仮想 PCI ID
- 結合インターフェイス：ネットワークデバイスのボンディングを使用すると、ネットワーク インターフェイスはスレーブ インターフェイス ロールとして機能するように識別されます。ボンドのスレーブインターフェイスは、MAC、PCI ID、またはインターフェイスタイプを使用して識別されます。
- インターフェイスタイプおよびインスタンス番号

仮想 PCI ID

PCI バス上のデバイスは、ドメイン、バス、デバイス、および機能番号と呼ばれる一意のタプルによって識別されます。これらの識別子は、いくつかの方法で識別できます。

ゲスト内では、**lspci** ユーティリティによって次のようにバスの設定が表示されます。

```
# lspci

00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

この仮想バスのドメイン、バス、デバイス、および機能番号を次に示します。

表 *2 :* 仮想 *PCI ID*

回線（**Line**）	ドメイン	バス	デバイス	機能
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)	0	0	0	0
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]	0	0	1	0
00:01.1 IDE インターフェイス：Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]	0	0	1	1
00:01.2 USB コントローラ：Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)	0	0	1	2
00:01.3 ブリッジ：Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)	0	0	1	3
00:02.0 VGA 互換コントローラ：Cirrus Logic GD 5446	0	0	2	0
00:03.0 システム周辺機器：Intel Corporation 6300ESB ウォッチドッグタイマー	0	0	3	0
00:04.0 未分類のデバイス [00ff]：Red Hat、Inc Virtio メモリバルーニング	0	0	4	0
00:05.0 イーサネットコントローラ：Red Hat、Inc Virtio ネットワークデバイス	0	0	5	0

回線（Line）	ドメイン	バス	デバイス	機能
00:06.0 イーサネットコントローラ：Red Hat、Inc Virtio ネットワークデバイス	0	0	6	0

libvirt ベースの仮想マシンの場合は、**virsh dumpxml** コマンドから仮想 PCI バストポロジを取得できます。libvirt スキーマでは、デバイス番号に *slot* という用語が使用されることに注意してください。これは、前の例で使用した仮想マシンにおける xml の説明のスニペットです。

```
<interface type='bridge'>
  <mac address='52:54:00:c2:d0:5f'/>
  <source bridge='br3043'/>
  <target dev='vnet0'/>
  <model type='virtio'/>
  <driver name='vhost' queues='8'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</interface>
<interface type='bridge'>
  <mac address='52:54:00:c3:60:eb'/>
  <source bridge='br0'/>
  <target dev='vnet1'/>
  <model type='virtio'/>
  <alias name='net1'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</interface>
```

インターフェイスタイプおよびインスタンス番号

ここで NIC は、Linux デバイスドライバの名前（virtio_net、vmxnet3、ixgbe、i40e など）とそのインスタンス番号を使用して、そのタイプによって識別されます。インスタンス番号は、そのタイプのインターフェイスの PCI 列挙順に基づいています。インスタンス番号 1 から始まります。インターフェイスタイプは、パススルーインターフェイスと SR-IOV 仮想機能だけでなく、両方の準仮想タイプを識別するために使用できます。PCI バス上のデバイスの PCI 列挙の順序は、**lspci** ユーティリティで確認できます。

たとえば、次のゲスト PCI トポロジの CF は、virtio_net インターフェイス番号 1 が 00:05.0 のイーサネットコントローラであり、virtio_net インターフェイス番号 2 が 00:06.0 のイーサネットコントローラであることを示しています。出力は、ゲストで実行された **lspci** コマンドからのものです。

```
# lspci

00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
```

```
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

サポートされている Linux ドライバの完全なリストを次に示します。

表 *3*: サポートされている *Linux* ドライバ

タイプ	PCI ベンダー/デバイス ID	ドライバ名
VIRTIO（KVM 用の準仮想 NIC）	0x10af / 0x1000	virtio_net
VMXNET3（VMware 用の準仮想 NIC）	0x15ad / 0x07b0	vmxnet3
Intel 10 ギガビットイーサネット	0x8086 / 0x10b6 0x8086 / 0x10c6 0x8086 / 0x10c7 0x8086 / 0x10c8 0x8086 / 0x150b 0x8086 / 0x10dd 0x8086 / 0x10ec 0x8086 / 0x10f1 0x8086 / 0x10e1 0x8086 / 0x10db 0x8086 / 0x1508 0x8086 / 0x10f7 0x8086 / 0x10fc 0x8086 / 0x1517 0x8086 / 0x10fb 0x8086 / 0x1507 0x8086 / 0x1514 0x8086 / 0x10f9 0x8086 / 0x152a 0x8086 / 0x1529 0x8086 / 0x151c 0x8086 / 0x10f8 0x8086 / 0x1528 0x8086 / 0x154d 0x8086 / 0x154f 0x8086 / 0x1557	ixgbe

タイプ	PCI ベンダー/デバイス ID	ドライバ名
Intel 10 ギガビット NIC 仮想機能	0x8086 / 0x10ed 0x8086 / 0x1515	ixgbevf
Cisco UCS NIC	0x1137 / 0x0043 0x1137 / 0x0044 0x1137 / 0x0071	enic
Mellanox ConnectX-5 （注） Mellanox は、ユーザプレーンでのみサポートされています。	0x15b3 / 0x1017 0x15b3 / 0x1018	mlx5_core
Intel 710 ファミリ NIC（PF）	0x8086 / 0x1572（40 ギガ） 0x8086 / 0x1574（40 ギガ） 0x8086 / 0x1580（40 ギガ） 0x8086 / 0x1581（40 ギガ） 0x8086 / 0x1583（40 ギガ） 0x8086 / 0x1584（40 ギガ） 0x8086 / 0x1585（40 ギガ） 0x8086 / 0x158a（25 ギガ） 0x8086 / 0x158b（25 ギガ）	i40e**
Intel 710 ファミリ NIC 仮想機能	0x8086 / 0x154c	i40evf

** 注：**i40e**ドライバの使用時に、ホスト上で作成された SRIOV VF に対して MAC アドレスの割り当てが動的に行われないという既知の問題が存在します。StarOS VM を起動するには、MACアドレスの割り当てが必要です。回避策として、MACアドレスの割り当てはホストから設定する必要があります。詳細は次のリンクを参照してください。https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf

起動パラメータの設定

起動パラメータファイルを作成しない場合は、デフォルトファイルが使用されます。起動パラメータファイルを作成する場合は、このタスクで説明されているすべてのパラメータを定義する必要があります。

始める前に

VMインターフェイスにおけるインターフェイス識別子の決定の詳細については、ネットワーク インターフェイス ロール （2 ページ）およびネットワーク インターフェイス ID （3 ページ）を参照してください。

ステップ 1 **CARDSLOT**=*slot-number*

slot_number は、スロット番号または VM を示す 1 ～ 32 の整数です。CF スロットには 1 または 2 を指定できます。SF スロットの範囲は 3 ～ 48 です。

ステップ 2 **CARDTYPE**=*card-type*

card-type は、VM が CF か SF かを識別します。

- コントロール機能には、0x40010100 を使用します。
- サービス機能には、0x42020100 を使用します。

ステップ 3 *interface-role*_**INTERFACE**=*interface-id*

interface-role の有効な値は次のとおりです。

- DI_INTERFACE
- MGMT_INTERFACE
- SERVICE # _INTERFACE、# の範囲は 1 ～ 12 です。
- VNFM_INTERFACE

インターフェイスロールの詳細については、ネットワーク インターフェイス ロール （2 ページ）を参照してください。

interface-id の有効な値は次のとおりです。

- MAC: xx:xx:xx:xx:xx:xx
- PCI_ID:xxxx:xx:xx.x (Domain:Bus:Device.Function)
- TYPE:*drive-name–instance-number*
- BOND: *slave-interface-A,slave-interface-B*

インターフェイス識別子の決定については、ネットワーク インターフェイス ID （3 ページ）を参照してください。

例：

この例では、MAC アドレスでインターフェイスを識別します。

```
DI_INTERFACE=MAC:00:01:02:03:04:05
```

この例では、ゲスト PCI アドレスでインターフェイスを識別します。

```
DI_INTERFACE=PCI_ID:0000:01:02.0
```

この例では、インターフェイスのタイプ（1 番目の virtio インターフェイス）でインターフェイスを識別します。

```
DI_INTERFACE=TYPE:enic-1
```

例：

この例では、インターフェイスをネットワーク ボンド インターフェイスとして識別します。次に、MAC アドレス、PCI 識別子、およびインターフェイスタイプを使用してインターフェイスを識別する例を示します。

```
DI_INTERFACE=BOND:MAC:00:01:02:03:04:05,MAC:00:01:02:03:04:06
# or
DI_INTERFACE=BOND:PCI_ID:0000:01:01.0,PCI_ID:0000:01:02.0
# or
DI_INTERFACE=BOND:TYPE:enic-1,TYPE:enic-2
```

ネットワーク インターフェイス ボンディングの設定

システムは、ネットワーク インターフェイスのペアをアクティブとスタンバイの結合インターフェイスに設定することをサポートしています。一度に 1 つのインターフェイスだけがアクティブになり、障害の検出は物理リンクの損失に限定されます。このタスクを使用して、結合インターフェイスを設定します。

すべてのボンディングの変数名は、*interface-role*_BOND 形式を使用します。インターフェイスロールについては、ネットワーク インターフェイス ロール （2 ページ） を参照してください。

始める前に

このタスクで説明されているすべての起動パラメータはオプションです。これらのパラメータが必要な場合は、起動パラメータの設定 （7 ページ） で説明されている必須のパラメータとともに、起動パラメータファイルに追加してください。

ステップ 1 *interface-role*_BOND_PRIMARY=*interface-id*

特定のインターフェイスが時間の大半をアクティブにするように設定している場合は、プライマリ スレーブ インターフェイスを設定します。デフォルトの結合設定では、プライマリスレーブは選択されません。

インターフェイスロールについては、ネットワーク インターフェイス ロール （2 ページ） を参照してください。インターフェイス識別子については、ネットワーク インターフェイス ID （3 ページ） を参照してください。

（注） デフォルトでは、復帰ポリシーは、新しいアクティブリンクの後続の障害時に、ボンドがプライマリインターフェイスに戻るだけというものです。

デフォルトでは、障害検出方法は、ボンドがドライバ状態を使用して、基本となるインターフェイスのリンク状態をポーリングすることです。

例：

次の例では、MAC アドレスを使用してプライマリインターフェイスを指定します。

```
DI_INTERFACE_BOND_PRIMARY=MAC:00:01:02:03:04:05
```

次に、PCI 識別子を使用してプライマリインターフェイスを指定する例を示します。

```
DI_INTERFACE_BOND_PRIMARY=BOND:PCI_ID:0000:01:01.0
```

次の例では、インターフェイスタイプの識別子を使用してプライマリインターフェイスを指定します。

例：

```
DI_INTERFACE_BOND_PRIMARY=BOND:TYPE:enic-1
```

ステップ 2 *interface-role*_BOND_MII_POLL = *poll-interval*

MII がリンク検出に使用される場合に使用するポーリング間隔（ミリ秒単位）を指定します。ポーリング間隔は 0 ～ 1000 の範囲で指定できます。デフォルトは 100 です。

ステップ 3 *interface-role*_BOND_MII_UPDELAY=*slave-enable-delay*

リンクの検出に MII が使用されている場合は、リンクの障害後にスレーブインターフェイスを有効にする前にリンクが安定するまでの待機時間を指定します。リンク状態は、最初に検出されたときにバウンスできます。この遅延により、インターフェイスを使用する前にリンクを安定させることができます。これにより、結合インターフェイスに対してアクティブなスレーブの過剰なフリップが回避されます。

スレーブの有効化遅延は、MII ポーリング間隔の倍数である必要があります。値はミリ秒単位で、デフォルトは 0 です。

ステップ 4 *interface-role*_BOND_MII_DOWNDELAY=*slave-disable-delay*

オプションです。この機能を使用すると、リンク検出に MII が使用されている場合に、スレーブインターフェイスがダウンしていることを宣言する前に、ボンドが待機するようになります。スレーブの無効化遅延は、MII ポーリング間隔の倍数である必要があります。値はミリ秒単位で、デフォルトは 0 です。

VNFM インターフェイスの設定

仮想ネットワーク機能マネージャ（VNFM）インターフェイスは、各 VM と VNFM の間で通信するように設計されています。このインターフェイスはメインのアプリケーションの前に起動され、起動パラメータのみを使用して設定できます。デフォルトでは、VNFMインターフェイスは無効になっています。

VNFM インターフェイスを設定するには、次のタスクを実行します。

始める前に

このタスクで説明されているすべての起動パラメータはオプションです。これらのパラメータが必要な場合は、起動パラメータの設定 （7 ページ） で説明されている必須のパラメータとともに、起動パラメータファイルに追加してください。

ステップ 1 VNFM_IPV4_ENABLE={true | false}

VNFM インターフェイスを有効にします。

ステップ 2 VNFM_CARTRIDGE_AGENT={true | false}

カートリッジエージェントを有効にします。VNFM がカートリッジエージェントを使用している場合は、これを有効にする必要があります。

ステップ 3 VNFM_IPV4_DHCP_ENABLE={true | false}

VNFM 上で DHCP を有効にします。

ステップ 4 VNFM_IPV4_ADDRESS=*x.x.x.x*

DHCP が使用されていない VNFM の IP アドレスを指定します。

ステップ 5 VNFM_IPV4_NETMASK=*x.x.x.x*

DHCP が使用されていない VNFM の IP アドレスのネットマスクを指定します。

ステップ 6 VNFM_IPV4_GATEWAY=*x.x.x.x*

DHCP が使用されていない VNFM の IP アドレスのゲートウェイを指定します。

VNFM インターフェイスオプション

（注） これらの設定オプションは任意です。

仮想ネットワーク機能マネージャ（VNFM）インターフェイスは、各 VM と VNFM の間で通信するように設計されています。VNFMインターフェイスはメインアプリケーションの前に初期化するため、インターフェイスを設定できるのは起動パラメータのみとなります。

デフォルトでは、VNFM インターフェイスは無効になっています。

VNFM IPv4 インターフェイスの有効化

デフォルト値は [False]（無効）です。

変数	有効な値
VNFM_IPV4_ENABLE	True または False

IPv4 DHCP クライアントの設定

変数	有効な値
VNFM_IPV4_DHCP_ENABLE	True または False

IPv4 スタティック IP の設定

（注） IPv4 DHCP クライアントが有効になっている場合、スタティック設定パラメータは無視されます。

変数	有効な値
VNFM_IPV4_ADDRESS	x.x.x.x
VNFM_IPV4_NETMASK	x.x.x.x
VNFM_IPV4_GATEWAY	x.x.x.x

VNFM IPv6 インターフェイスを有効にします。

変数	有効な値
VNFM_IPV6_ENABLE	True または False

IPv6 スタティック IP 設定の有効化

変数	有効な値
VNFM_IPV6_STATIC_ENABLE	True または False

True に設定すると、次の項に示すように、スタティック IP パラメータの設定がインターフェイスに適用されます。False に設定すると、インターフェイスはステートレス自動設定（RFC4862）と DHCPv6 の両方を使用してインターフェイスのアドレスを設定しようとします。

IPv6 スタティック IP の設定

（注） 「VNFM_IPV6_ENABLE」パラメータ値が false に設定されている場合、スタティック設定パラメータは無視されます。IPv6 アドレスフィールドは、RFC 5952 に準拠している必要があります。プレフィックスは /64 で固定されています。

変数	有効な値
VNFM_IPV6_ADDRESS	x:x:x:x:x:x:x:x
VNFM_IPV6_GATEWAY	x:x:x:x:x:x:x:x

DI ネットワーク VLAN の設定

DI ネットワークには、使用可能な一意の分離ネットワークが必要です。パススルーインターフェイスを使用する場合、カスタマーネットワークでの VPC-DI インスタンスの分離を容易にするために、VLAN ID を設定できます。VLAN を設定するには、このタスクを実行します。

始める前に

このタスクで説明されているすべての起動パラメータはオプションです。これらのパラメータが必要な場合は、起動パラメータの設定 （7 ページ） で説明されている必須のパラメータとともに、起動パラメータファイルに追加してください。

DI_Internal_VLANID=*vlan-id*

内部 DI ネットワークの VLAN ID を指定します。値の範囲は 1 ～ 4094 です。

例：

```
DI_INTERNAL_VLANID=10
```

IFTASK の調整可能なパラメータの設定

デフォルトでは、DPDK は CPU コアの 30% を内部フォワーダタスク （IFtask）のプロセスに割り当てます。これらの起動パラメータを使用して、IFTASK に割り当てられたリソースを設定できます。IFTASK の CPU コア割り当てに関する情報を表示するには、**show cpu info** と **show cpu verbose** コマンドを使用します。

（注） これらは、細心の注意を払って設定する必要があるオプションのパラメータです。

ステップ 1 （オプション）**IFTASK_CORES=***percentage-of-cores*

IFTASK に割り当てる CPU コアの割合を指定します。値の範囲は 0 ～ 100パーセントです。デフォルトは 30 です。

ステップ 2 （オプション）**IFTASK_MCDMA_CORES=***percentage-of-iftask-cores*

マルチチャネル ダイレクト メモリ アクセス（MCDMA）に割り当てられているコアの数を、iftask コアの総数の割合として設定します。まず、上記の **IFTASK_CORES** パラメータを定義する必要があります。

（注） NUMA 最適化が有効になっており、この MCDMA コアカウント設定が設定されている場合（**IFTASK_CORES=***percentage-of-cores*）、*percentage-of-cores* を偶数に設定する必要があります。これにより、MCDMA スレッドが NUMA ノード間で均等に分散されます。

ステップ 3 （オプション）**MCDMA_THREAD_DISABLE=***percentage-of-iftask-cores*

MCDMA と VNPU の分割を使用するのではなく、すべてのコアで PMD を実行するには、MCDMA_THREAD_DISABLE パラメータを 1 に設定します。

ステップ 4 （オプション）**IFTASK_SERVICE_TYPE=***value*

サービスメモリを計算しサービス固有の機能を有効にするために展開するサービスタイプを指定します。次のサービスタイプを指定できます。

- 0 = VPC サービスタイプ
- 1 = GiLAN サービスタイプ
- 2 = ePDG サービスタイプ
- 3 = CUPS コントローラサービスタイプ
- 4 = CUPS フォワーダサービスタイプ

デフォルトは 0 です。

ステップ 5 （オプション）**IFTASK_CRYPTO_CORES=***value*

IFTASK_SERVICE_TYPE が「2」（EPDG）に設定されている場合、このパラメータは、暗号処理に割り当てる iftask コアの割合を指定します。値の範囲は 0 ～ 50パーセントですが、専用のコアの上限は 4 です。デフォルトは 0 です。

（注） このパラメータは、**IFTASK_SERVICE_TYPE** が「2」（EPDG）に設定されている場合にのみ使用してください。他のサービスタイプに設定されている場合は、このパラメータを「0」に設定する必要があります。

ステップ 6 （オプション）**IFTASK_SW_TX-RSS=***value*

システムの送信パススループットの拡張を無効にするには、この設定を使用します。

- 0 = 有効（デフォルト）
- 1 = 無効

ステップ 7 （オプション）**IFTASK_DISABLE_NUMA_OPT=***value*

複数の NUMA ノードがホストによって VM に提供されている場合でも、この設定を使用して NUMA 最適化を無効にします。このオプションは、NUMAの最適化が何らかの理由で望ましくない場合に設定できます。

- NO = 有効（デフォルト）
- YES = 無効

NUMA の最適化は、次の場合を除き、デフォルトで有効になっています。

- NUMA ノードとセルの数が 2 に等しくない。
- カードタイプが制御機能（CF）、アプリケーション機能（AF）、またはネットワーク機能（NF）になっている。サービス機能（SF）VM のみが NUMA をサポートしている。
- VM のサービスタイプが VPC ではない。NUMAが VPC サービスタイプでのみサポートされている。
- この設定が、明示的に [YES]（**IFTASK_DISABLE_NUMA_OPT=YES**）に設定されている。

ステップ 8 （オプション）**IFTASK_VNPU_TX_MODE=***value*

Ultra M 展開のコンピューティングノードには、28 個のコアがあります。これらのコアのうち 2 つは、ホストで使用するために予約されています。26 個のコアを使用すると、MCDMA 機能を実行するために使用されるコア間で MCDMA チャネルが均等に分散されます。

この設定を有効にすると、iftask 内の MCDMA 機能のコアは MCDMA コアおよび VNPU TX ルックアップコアとして均等に分割されます。

- 0 = 無効（デフォルト）
- 1 = 有効

ステップ 9 （オプション）**MULTI_SEG_MBUF_ENABLE=***value*

リリース 21.6 以降のデフォルトでは、システムは Ixgbe pf/vf ドライバのすべてのメモリプールで、サイズの小さいバッファを使用したマルチセグメントの送受信の使用を有効にします。この機能により、IFTASK の全体的なメモリサイズが削減され、小規模な展開に適したものになります。

- 1 = true（xgbe NIC の場合はデフォルト）。
- 0 = false（他のすべての NIC の場合はデフォルト）。

重要 デフォルトでは、この機能が有効になっているため、Ixgbe NIC を使用するシステムで 21.6 にアップグレードする場合は注意が必要です。

この機能は、Ixgbe vf/pf NICs NICs を使用していないシステムでは自動的に無効になります。

例

StarOS の **show cloud hardware iftask** *card_number* コマンドを使用して、起動パラメータが有効になったことを確認します。

```
[local]mySystem# show cloud hardware iftask 4
Card 4:
  Total number of cores on VM:      24
```

```
  Number of cores for PMD only:     0
  Number of cores for VNPU only:    0
  Number of cores for PMD and VNPU: 3
  Number of cores for MCDMA:        4
  Number of cores for Crypto        0
Hugepage size:                2048 kB
Total hugepages:           3670016 kB
NPUSHM hugepages:                0 kB
CPU flags: avx sse sse2 ssse3 sse4_1 sse4_2
Poll CPU's: 1 2 3 4 5 6 7
KNI reschedule interval: 5 us
```

最大 IFtask スレッドサポートの改善

機能の概要と変更履歴

要約データ

該当製品または機能エリア	すべて
該当プラットフォーム	VPC-DI
機能のデフォルト	有効、常時オン
このリリースでの関連する変更点	N/A
関連資料	*VPC-DI* システム管理ガイド

マニュアルの変更履歴

重要 リリース 21.2 および N5.1 よりも前に導入された機能の改訂履歴の詳細は示していません。

改訂の詳細	リリース
このリリース以降、サポートされる IFtask スレッド設定の最大数は 22 コアに増えました。	21.8
最初の導入。	21.2 よりも前

変更された機能

設定されている（/tmp/iftask.cfg 内）DPDK 内部フォワーダ（IFTask）スレッドの数が 14 コアを超えている場合、IFTask はパケットをドロップするか、またはエラーを表示します。

以前の動作：現在、IFtask スレッド設定の最大数は 14 コアのみに限定されています。

新しい動作：リリース 21.8 以降、サポートされる IFtask スレッドの最大数が 22 コアに増加しました。

MTU サイズの設定

デフォルトでは、IFTASK プロセスは最大インターフェイス MTU を次のように設定します。

- サービスインターフェイス：2,100 バイト
- DI ネットワーク インターフェイス：7,100 バイト

これらのデフォルトを変更するには、param.cfg ファイルで次のパラメータを設定します。

表 *4 : MTU* サイズパラメータ

パラメータ名	範囲	デフォルト値
DI_INTERFACE_MTU=	576-9100	7100
SERVICE_INTERFACE_MTU=	576-9100	2100

ジャンボフレームをサポートしていないシステムの MTU サイズの設定については、サポート対象 MTU を超えるトラフィックのサポートの設定 （17 ページ）を参照してください。

サポート対象 MTU を超えるトラフィックのサポートの設定

デフォルトでは、システムが動作するにはジャンボフレームのサポートが必要です。インフラストラクチャがジャンボフレームをサポートしていない場合でも、システムを実行できます。ただし、起動パラメータファイルで DI 内部ネットワークの MTU を 1500 に指定する必要があります。これにより、IFTASK が、サポートされている MTU を超える DI ネットワークトラフィックを処理できるようになります。

始める前に

このタスクで説明されているすべての起動パラメータはオプションです。これらのパラメータが必要な場合は、起動パラメータの設定 （7 ページ） で説明されている必須のパラメータとともに、起動パラメータファイルに追加してください。

DI_INTERFACE_MTU=*1500*

ソフトウェアがジャンボフレームを適切に処理できるように、DI内部ネットワークがジャンボフレームをサポートしないことを指定します。

起動パラメータファイルの例

この例では、2 つの VIRTIO インターフェイスを備えたスロット 1 の CF の起動パラメータファイルを示しています。

```
CARDSLOT=1
```

```
CARDTYPE=0x40010100
DI_INTERFACE=TYPE:enic-1
MGMT_INTERFACE=TYPE:virtio_net-2
```

この例では、3 つの VIRTIO インターフェイスを備えたスロット 3 の SF の起動パラメータファイルを示しています。

```
CARDSLOT=3
CARDTYPE=0x42020100
DI_INTERFACE=TYPE:enic-1
SERVICE1_INTERFACE=TYPE:enic-3
SERVICE2_INTERFACE=TYPE:enic-4
```

この例では、パススルー NIC、ボンディングが設定された CF、および VLAN 上に DI 内部ネットワークがある CF の起動パラメータファイルを示しています。

```
CARDSLOT=1
CARDTYPE=0x40010100
DI_INTERFACE=BOND:TYPE:enic-1,TYPE:enic-2
MGMT_INTERFACE=BOND:TYPE:ixgbe-3,TYPE:ixgbe-4
DI_INTERNAL_VLANID=10
```

VPC-DI ESC を使用したオンボーディング

ESC を使用して VPC-DI のインスタンスを開始できます。

OpenStack での ESC を使用した VPC-DI のオンボーディング

この手順では、OpenStack 環境で ESC のインスタンスに VPC-DI をオンボードする方法について説明します。

始める前に

この手順では、ネットワークアクセスを使用してリリース Juno 以降を実行している動作中の OpenStack 環境内に ESC が作成されていることを前提としています。ESC の詳細なインストール手順については、『*Cisco Elastic Services Controller 2.3 Install and Upgrade Guide*』（http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-3/install/guide/Cisco-Elastic-Services-Controller-Install-Upgrade-Guide-2-3.html）を参照してください。ESC 設定の説明については、『*Cisco Elastic Services Controller 2.3 User Guide*』（http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-3/user/guide/Cisco-Elastic-Services-Controller-User-Guide-2-3.html）を参照してください。

このリリースでサポートされている Elastic Services Controller のバージョンを確認するには、『リリースノート』を参照してください。

ステップ 1 CF および SF の VPC-DI インスタンスの qcow イメージを取得します。

イメージを含むtarballファイルには、リリース番号に応じてproduction.xxxxx.qvpc-di.qcow2.tgzというような名前が付けられています。このアーカイブには、CF と SF それぞれの 2 つのイメージ（qvpc-di-cf.qcow2 と qvpc-di-xf.qcow2）が含まれています。

ステップ 2 VPC-DI コマンドを使用すると **glance image-create** イメージが瞬時に作成されます。

例：

```
$ glance image-create --file qvpc-di-cf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-cf

$ glance image-create --file qvpc-di-xf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-xf
```

ステップ 3 サンプルの VPC-DI の初期化 tarball（vpc_esc_sample.tgz）を取得します。

ステップ 4 サンプルの VPC-DI の初期化 tarball を管理者ホーム（/home/admin/）の ESC VM にコピーします。

ステップ 5 サンプルの VPC-DI 初期化 tarball を vnf ディレクトリ（opt/cisco/vnfs/cisco-qvpc/）に解凍します。

ステップ 6 コマンドライン API コマンド **esc_nc_cli edit-config** を使用して、アーティファクトを作成し ます。

例：

```
esc_nc_cli edit-config  /opt/cisco/vnfs/cisco-qvpc/dep/artifacts.xml
```

ステップ 7 **esc_nc_cli edit-config** コマンドを使用して VPC-DI を展開します。

（注） ESC を再展開する前に、既存の展開を必ず削除してください。手順 10 を参照してください。

VPC-DI は、vnfs/cisco-qvpc/dep/にある dep.xml ファイルを使用して展開されます。一般に、デフォルトの dep.xml ファイルを使用できます。展開をカスタマイズする必要がある場合は、このファイルに必要な変更を加えます。たとえば、VPC-DI のシャーシ ID の作成に使用するシャーシキーを編集するには、dep.xml ファイルの該当するセクションを編集します。

```
<property>
  <name>CHASSIS_KEY</name>
  <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
</property>
```

dep.xml ファイルの詳細については、次を参照してください。 http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-2/deployment/Cisco-Elastic-Services-Controller-2-2-Deployment-Attributes.pdf

例：

```
esc_nc_cli edit-config /opt/cisco/vnfs/cisco-qvpc/dep/dep.xml
```

このコマンドは、テナントコアがまだ存在していない場合に新しいテナントコアを作成できます。使用されている dep*.xml によっては、展開で「クォータ超過」エラーが発生する可能性があります。すべての SF と CF が起動しない場合は、OpenStack でデフォルトのテナントクォータとテナントコアのクォータを確認します。これを実行するコマンドは $ nova quota-defaults; nova quota-show --tenant Core です。

ステップ 8 ログファイルの var/log/esc/yangesc.logで展開ステータスを確認します。

ステップ 9 VPC-DI が統合されるまで待ちます。

ステップ 10 VPC-DI の展開を削除するには、ESC CLI コマンドの **esc_nc_cli delete-dep** *Tenant deployment-name* を使用します。

例：

```
$ ./esc_nc_cli delete-dep Core cisco-qvpc
```

ステップ 11 （オプション）カスタムモニタリングを使用するには、VPC-DI を展開する（手順 7）前に次の手順を実行します。

a) mib ファイルをコピーします。

例：

```
sudo cp /opt/cisco/vnfs/cisco-qvpc/config/starent.my /usr/share/snmp/mibs/
```

b) VPC-DI のダイナミック マッピング メトリックを既存のマッピングに追加します。ファイル /opt/cisco/vnfs/cisco-qvpc/config/dynamic_mappings_snippet.xml の内容を /opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml にマージします。この手順を実行するには、細心の注意が必要です。SF ごとに、opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml に 1 つのダイナミックマッピングが必要です。

ESC を使用した VPC-DI のオンボーディングのカスタマイズ

ESC を使用して VPC-DI をオンボードすると、VPC-DI ファイルは /opt/cisco/vnfs/cisco-qvpc/ ディレクトリに配置されます。VPC-DI システムのインストールにはさまざまな変更を加えることができます。次のファイルは必要に応じて変更できます。

VNF 展開ファイル

展開ファイルは、ディスクの /opt/cisco/vnfs/cisco-qvpc/dep/ にコピーされます。

必要に応じて、追加のファイルを作成できます。

起動パラメータファイル

ESC を使用した VPC-DI のオンボードの場合は、ディスク上に配置されたさまざまな CF VM と SF VM 用にいくつかの異なる起動ファイルが用意されています。これらのファイル /opt/cisco/vnfs/cisco-qvpc/config/ にコピーされます。

- param.cfg
- param_sf.cfg

これらは、必要に応じてカスタマイズできます。起動パラメータのファイル形式の詳細については、起動パラメータファイルの作成 （1 ページ）を参照してください。

次に、CF の起動パラメータファイルの例を示します。

```
CARDSLOT=$SLOT_CARD_NUMBER
CPUID=0
CARDTYPE=$CARD_TYPE_NUM
```

```
DI_INTERFACE=TYPE:enic-1
VNFM_INTERFACE=TYPE:virtio_net-2
MGMT_INTERFACE=TYPE:virtio_net-3

VNFM_IPV4_ENABLE=true
VNFM_IPV4_DHCP_ENABLE=true
```

次に、SF の起動パラメータファイルの例を示します。

```
CARDSLOT=$SLOT_CARD_NUMBER
CPUID=0
CARDTYPE=$CARD_TYPE_NUM

DI_INTERFACE=TYPE:enic-1
VNFM_INTERFACE=TYPE:enic-2
SERVICE1_INTERFACE=TYPE:enic-3

VNFM_IPV4_ENABLE=true
VNFM_IPV4_DHCP_ENABLE=true
```

設定ファイル

/cisco/images/system.cfg にある設定ファイルはカスタマイズできます。次に、標準設定ファイルの例を示します。

```
config
system hostname $VPC_HOSTNAME
clock timezone $TIMEZONE
context local
administrator admin password $ADMIN_PASS ftp

interface LOCAL1
ip address $CF_VIP_ADDR $CF_VIP_NETMASK
ip route 0.0.0.0 0.0.0.0 $NICID_1_GATEWAY LOCAL1
ip domain-lookup
ip domain-name $CF_DOMAIN_NAME
ip name-servers $CF_NAME_SERVER
ssh generate key
server sshd
subsystem sftp
port ethernet 1/1
bind interface LOCAL1 local
no shutdown
snmp community $SNMP_COMMUNITY read-only
end
```

詳細については、コンフィギュレーション ファイルの概要を参照してください。

OpenStack のパフォーマンスの最適化

Cisco ESC では、Non-Uniform Memory Access（NUMA）ノード設定などの OpenStack Kilo リリースを使用して、大きなページのサポートやゲスト vCPU のピン接続など、数多くのハイパーバイザ最適化を実行できます。

- vCPU ピン接続：ゲスト vCPU をホストの一連の物理 CPU に正確に固定する機能です。これにより、vCPU が物理リソースを使用できるようになるまで待機することを未然に防ぎます。
- 大規模ページ：仮想リソースに大きなメモリブロックを割り当てます。
- PCI ベースの NUMA のスケジューリング：OpenStack 内のインスタンスに PCI デバイスを割り当てる機能です。

（注） これらの OpenStack パフォーマンスの最適化は OpenStack Kilo バージョンのみを使用してサポートされます。

OpenStack Kilo はさまざまなメカニズムを使用してさまざまなハードウェア アクセラレーション機能を提供します。このようなメカニズムの 1 つには、フレーバオブジェクトのキー値属性の設定が含まれます。特定のフレーバを参照する VM インスタンス化要求は、対応するハードウェア アクセラレーション機能を効率的に要求します。必要な OpenStack 設定が OpenStack の制御ノードとコンピューティングノードに配置されている場合は、OpenStack コンピューティングサービス（「nova」）が適切なコンピューティングノードを選択し、対応するリソースを割り当てます。

VPC-DI でハードウェア アクセラレーションを使用するには、Cisco ESC に新しいフレーバを作成し、NETCONF インターフェイスか REST インターフェイスを使用して必要なメタデータ属性を追加する必要があります。フレーバの詳細については、『*Cisco Elastic Services Controller User Guide*』（http://www.cisco.com/c/en/us/support/cloud-systems-management/elastic-services-controller-2-1/model.html）の「*Managing Flavors*」を参照してください。また、多数の OpenStack 設定を変更する必要がある場合もあります。ハードウェア アクセラレーションを実装する手順については、次のタスクで説明します。

（注） メタデータの追加は、既存のものではなく、新しいフレーバでのみサポートされています。メタデータ属性を既存のフレーバに追加する必要がある場合は、OpenStack と直接連携する必要があります。

CPU ピニングの設定

ステップ 1 各 OpenStack 制御ノードで、スケジューラを設定します。

a) NUMAToplogy フィルタと AggregateInstanceExtraSpec フィルタを含めるようにスケジューラフィルタを設定します。

例：

```
$ sudo vim /etc/nova/nova.conf
...
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,
ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilter,NUMATopologyFilter,
```

```
AggregateInstanceExtraSpecsFilter
```

b) nova スケジューラサービスを再起動します。

例：

```
$ sudo systemctl restart openstack-nova-scheduler.service
```

ステップ 2 関連する各 OpenStack のコンピューティングノードで、どのハイパーバイザプロセスをゲストに使用し、どのハイパーバイザプロセスをゲストに使用しないようにするかを設定します。

a) ハイパーバイザプロセスがゲスト用に予約されているコアで実行されていないことを確認します。

たとえば、ゲストにコア 2、3、6、および 7 を予約するには、grub ブートローダーを更新して再起動します。

例：

```
$ sudo grubby --update-kernel=ALL --args="isolcpus=2,3,6,7"
$ sudo grub2-install /dev/sda
$ sudo reboot
```

b) カーネルのコマンドラインに変更が反映されていることを確認します。

例：

```
$ cat /proc/cmdline
...  isolcpus=2,3,6,7 ...
```

c) ゲスト仮想マシンのインスタンスを設定して、特定のコアでの実行のみが許可され、ハイパーバイザプロセス用の RAM を予約するようにします。たとえば、コア 2、3、6、および 7 を使用し、ハイパーバイザプロセスに 512 MB を予約するには、次の手順を実行します。

例：

```
$ sudo vim /etc/nova/nova.conf
...
vcpu_pin_set=2,3,6,7
...
reserved_host_memory_mb=512
...
```

d) nova コンピューティングサービスを再起動します。

例：

```
$ sudo systemctl restart openstack-nova-compute.service
```

ステップ 3 グローバルパラメータを設定します。

a) ピン接続要求を受信したホストに対して `performance-pinned` ホスト集約を作成し、その識別のために任意の属性 `pinned=true` を追加します。

例：

```
$ nova aggregate-create performance-pinned
$ nova aggregate-set-metadata performance-pinned pinned=true
```

b) 他のすべてのホストに対して通常の集約を作成し、同じ任意の属性を追加しますが、それを識別するには、pinned=false を設定します。

例：

```
$ nova aggregate-create normal
$ nova aggregate-set-metadata normal pinned=false
```

c) 以前に有効になっていたコンピューティングノードを、performance-pinned ホスト集約に追加し、他のすべてのコンピューティングノードを通常のホスト集約に追加します。

例：

```
$ nova aggregate-add-host normal             compute100.cloud.com
$ nova aggregate-add-host normal             compute101.cloud.com
$ nova aggregate-add-host normal             compute102.cloud.com
$ nova aggregate-add-host performance-pinned compute103.cloud.com
$ nova aggregate-add-host performance-pinned compute104.cloud.com
$ nova aggregate-add-host performance-pinned compute105.cloud.com
```

ステップ 4 Cisco ESC ノースバウンド API を使用してフレーバー属性を設定します。

a) フレーバー属性を設定します。

- hw:cpu_policy=dedicated
- aggregate_instance_extra_specs:pinned=true

このフレーバーを使用して作成されたすべてのインスタンスは、集約メタデータ内のピン接続された pinned=true を使用して、ホスト集約のホストに送信されます。

例：

```
 version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>testfl6</name>
      <vcpus>1</vcpus>
      <memory_mb>2048</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
      <ephemeral_disk_mb>0</ephemeral_disk_mb>
      <swap_disk_mb>0</swap_disk_mb>
      <properties>
        <property>
          <name>hw:cpu_policy</name><value>dedicated</value>
          <name>aggregate_instance_extra_specs:pinned</name><value>true</value>
        </property>
      </properties>
    </flavor>
  </flavors>
</esc_datamodel>
```

b) 追加の仕様が通常の集約のコンピューティングホストと一致するように、他のすべてのフレーバーを更新します。

例：

```
$ nova flavor-key <flavor_Id> set aggregate_instance_extra_specs:pinned=false
```

c) 確認するには、変更されたフレーバーを使用して VM インスタンスを起動し、VM インスタンスが開始されたコンピューティングノードを検索します。

例：

```
$ nova boot --image <test-image> --flavor <modified-flavor> test-instance
$ nova show test-instance | egrep
'OS-EXT-SRV-ATTR:hypervisor_hostname|OS-EXT-SRV-ATTR:instance_name'
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute3.cloud.com
| OS-EXT-SRV-ATTR:instance_name        | instance-00000cee
```

d) 返されたコンピューティングノードにログインし、**virsh** ツールを使用して、返されたインスタンスの XML を抽出します。

例：

```
$ ssh compute3.cloud.com
...
$ sudo virsh dumpxml instance-00000cee
...
  <vcpu placement='static' cpuset='2-3,6-7'>1</vcpu>
```

大きなページの設定

OpenStack 設定で大きなページを設定するには、次の手順を実行します。

ステップ 1 ページ数を設定するには、/etc/sysctrl.conf を編集します。

例：

```
vm.nr_hugepages = 32768
```

ステップ 2 ページサイズを設定するには、/proc/meminfo を編集します。

例：

```
Hugepagesize: 2048 kB
```

ステップ 3 Cisco ESC ノースバウンド API を使用して、hw: mem_page_size = 2048 のフレーバを作成します。

例：

```
<?xml version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>testfl6</name>
      <vcpus>1</vcpus>
      <memory_mb>2048</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
      <ephemeral_disk_mb>0</ephemeral_disk_mb>
      <swap_disk_mb>0</swap_disk_mb>
      <properties>
        <property>
          <name>hw:mem_page_size</name><value>2048</value>
```

```
        </property>
      </properties>
    </flavor>
  </flavors>
</esc_datamodel>
```

PCI パススルーの設定

Intel VT-d 拡張機能は、物理デバイスをゲストに直接割り当てるためのハードウェアサポートを提供します。これにより、仮想デバイスは、複数のスイッチまたはブリッジを通過して物理インターフェイスに到達する際に伴うスループットの損失とパケット転送容量の減少を回避できます。

VT-d 拡張機能は、Red Hat Enterprise Linux を使用した PCI パススルーに必要です。拡張機能は、BIOS で有効にする必要があります。システム メーカーによっては、これらの拡張機能がデフォルトで無効になっている場合があります。

この手順では、BIOS の観点から VT-d を有効化する方法については説明しません。VT-d を有効にするには、サーバ製造時の BIOS 設定ガイドを参照してください。Linux カーネルの観点から、「intel_iommu＝on」を grub 設定に追加することで、VT-d が有効になります。

始める前に

PCI パススルーをサポートするには、Intel チップセットで VT-d を有効にする必要があります。

ステップ 1 Red hat Enterprise Linux で VT-d サポートを有効にします。

a) 「/etc/default/grub」を編集し、行の末尾に「intel_iommu＝on」を追加します。GRUB_CMDLINE_LINUX

例：

```
GRUB_TIMEOUT=5
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto rd.lvm.lv=rhel/root rhgb quiet
intel_iommu=on"
GRUB_DISABLE_RECOVERY="true"
```

b) grub2-mkconfig コマンドを実行して、grub.conf を再生成し、サーバを再起動します。

例：

```
grub2-mkconfig -o /boot/grub2/grub.cfg on BIOS systems
```

または

```
grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg on UEFI systems
```

c) コマンドを実行して、IOMMU がアクティブになっていることを確認します。**dmesg | grep -iE "dmar|iommu"**

例：

IOMMU が有効になっている **dmesg** コマンドの出力例：

```
[ 0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.10.0-229.el7.x86_64
root=/dev/mapper/rhel-root
ro rd.lvm.lv=rhel/swap crashkernel=auto rd.lvm.lv=rhel/root rhgb quiet intel_iommu=on
[ 0.000000] Intel-IOMMU: enabled
```

ステップ 2 Linux カーネルからデバイスをアンバインドします。

a) PCI パススルーが機能するには、デバイスが Linux カーネルドライバからアンバインドされている必要があります。これを実現するには pci_stub モジュールを使用します。

例：

```
Load pci_stub module "modprobe pci_stub"
```

b) PCI パススルーに使用するネットワークアダプタを見つけます。**lspci** を実行し、目的のネットワークカードの PCI アドレスをメモします。

例：

この例では、PCI パススルーに PCI デバイス 15: 00.0 を使用する必要があります。

```
12:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
13:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
14:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
15:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
```

c) ベンダー ID とデバイス ID を特定します。**lspci -n**の実行

例：

この部分的な出力では、ベンダー ID 1137 とデバイス ID 0071 が識別されます。

```
11:00.0 0c04: 1137:0071 (rev a2)
12:00.0 0200: 1137:0071 (rev a2)
13:00.0 0200: 1137:0071 (rev a2)
14:00.0 0200: 1137:0071 (rev a2)
15:00.0 0200: 1137:0071 (rev a2)
```

d) この設定を使用して、目的のデバイスを Linux カーネルドライバからアンバインドします。

強調表示されたテキストは、デバイス情報に合わせて変更する必要があります。

例：

```
echo "1137 0071" > /sys/bus/pci/drivers/pci-stub/new_id
echo 0000:15:00.0 > /sys/bus/pci/devices/0000:15:00.0/driver/unbind
echo 0000:15:00.0 > /sys/bus/pci/drivers/pci-stub/bind
```

e) **dmesg | grep stub** を実行して、これらのコマンドが成功したことを確認します。

例：

```
[ 276.705315] pci-stub 0000:15:00.0: claimed by stub
```

f) 変更を永続的にするには、pci-stub.ids を grub CMDLINE に追加し、grub を更新して、ホストを再起動します。

（注） このコードは、指定されたベンダー/デバイス ID を持つすべての vNIC（この例では 1137:0071）に適用されます。

例：

```
edit /etc/default/grub
GRUB_CMDLINE_LINUX="..pci-stub.ids=1137:0071"
grub2-mkconfig -o /boot/grub2/grub.cfg
reboot
```

ステップ 3 OpenStack で nova を設定します。

a) （VendorID、productID）の組み合わせを使用して、（vendorID、productid）の組み合わせを使用した PCI デバイスの通過を許可するか、または通過を許可するデバイスの PCI アドレスを指定することにより、PCI パススルーに使用できる PCI デバイスを指定します。

```
pci_passthrough_whitelist={"vendor_id": "1137", "product_id": "0071"}
```

または

```
pci_passthrough_whitelist = [ {"address": "01:00.1"}, {"address": "02:00.1"} ]
```

b) PCI エイリアスと（製品 ID、ベンダー ID）の組み合わせマッピングを指定します。この設定では現在 PCI アドレスがサポートされていないため、ホワイトリストの PCI デバイスはすべて同じ名前になっています。

例：

```
pci_alias={"vendor_id":"1137", "product_id":"0071", "name":"nic1"}
```

c) nova.conf に次の追加の変更を加えます。

例：

```
scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_available_filters=nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter
scheduler_default_filters=RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter
```

d) nova を再起動します。

例：

```
openstack-service restart nova
```

ステップ 4 属性 pci_passthrough を使用してフレーバーを作成します。エイリアスは <PCI_DEVICE_ALIAS>:<NUM_DEVICES_REQUESTED> に設定されています。PCI_DEVICE_ALIAS は、/etc/nova/nova.conf の pci_alias 設定から値を参照します。

例：

```
$ cat fl.xml
<?xml version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <flavors>
        <flavor>
            <name>testfl6</name>
              <vcpus>1</vcpus>
              <memory_mb>2048</memory_mb>
              <root_disk_mb>10240</root_disk_mb>
              <ephemeral_disk_mb>0</ephemeral_disk_mb>
              <swap_disk_mb>0</swap_disk_mb>
              <properties><property>
                  <name>pci_passthrough:alias</name><value>nic1g:1</value>
              </property></properties>
        </flavor>
    </flavors>
</esc_datamodel>
$ sudo /opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli edit-config ./fl.xml
```

OpenStack で Heat オーケストレーション テンプレート（HOT）を使用した VPC-DI のオンボーディング

VPC-DI は、OpenStack 環境で仮想ネットワーク機能（VNF）として展開できます。VPC-DI は、仮想マシンの集合として実行され、VM には、ストレージ、ネットワーキング、および設定に関する固有の要件があります。OpenStack 環境では、オーケストレータは VPC-DI VM を起動するために必要なオブジェクトの作成を担当します。オーケストレータは、VM とそれらに関連付けられたオブジェクトを作成して終了する役割も担います。オーケストレータは OpenStack サービスを使用して、OpenStack にそのようなエンティティを作成します。

OpenStack は、テンプレートを介した VNF のネットワーク、コンピューティング、およびストレージトポロジを定義する、HEAT オーケストレーション テンプレート（HOT）と呼ばれるサービスを提供します。HEAT テンプレートは、VNF のインスタンスを展開するためのブループリントとして使用できます。

テンプレートの形式と ENV パラメータファイルの例については、この項の後半で説明します。

ステップ 1 CF および SF の VPC-DI インスタンスの qcow イメージを取得します。

イメージを含む tarball ファイルには、リリース番号に応じて production.xxxxx.qvpc-di.qcow2.tgz というような名前が付けられています。tarball ファイルが開いている場合は、CF と SF の 2 つのイメージ（qvpc-di-cf.qcow2 と qvpc-di-xf.qcow2）が存在する必要があります。

ステップ 2 **glance image-create** コマンドを使用するとすべての VPC-DI イメージが瞬時に作成されます。

例：

```
$ glance image-create --file qvpc-di-cf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-cf
```

```
$ glance image-create --file qvpc-di-sf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-sf
```

ステップ 3 VPC-DI サンプルの初期化 tarball（vpc_HOT_sample.tgz）を取得します。

ステップ 4 VPC-DI サンプルの初期化 tarball をローカルマシンにコピーします。

ステップ 5 VPC-DI サンプルの初期化 tarball を任意のディレクトリに解凍します。拡張子が **.yml** と **.env** の 2 つのファイルがあります。

ステップ 6 OpenStack の展開に応じて、ENV ファイルを編集します。

ネットワーク、可用性ゾーンなどの値を入力します。

ローカルディレクトリを参照して、テンプレートの送信元の **.yml** ファイルと環境の送信元の **.env** ファイルをクリックします。

ステップ 7 次のいずれかを実行します。

- [Project] > [Orchestration] > [Stacks] > [Launch Stack] に移動し、OpenStack ダッシュボードを使用して VPC-DI を展開します。
- CLI を使用して、**heat stack-create -f di.yml -e di.env** コマンドで HEAT を使用して VPC-DI を展開します。

ステップ 8 [Status] フィールドが [Complete] であることを確認します。これはエラーがあることを示しています。

ステップ 9 VPC-DI が統合されるまで待ちます。

ステップ 10 VPC-DI 展開を削除するには、スタック名の横のチェックボックスをオンにし、[Delete Stack] をクリックします。

VPC-DI Heat オーケストレーション テンプレート

この項では、Heat テンプレートの形式について説明します。VPC-DI HOT バージョンは 2013-05-23 です。テンプレートには、パラメータグループ、パラメータ、リソース、および出力の 4 つのセクションがあります。

VPC-DI HOT パラメータグループ

[parameter_groups] セクションでは、入力パラメータをグループ化する方法、およびパラメータを提供する順序を指定できます。これらのグループは、ダウンストリームユーザインターフェイスで予想される動作を説明するために使用されます。

表 *5 : HOT* パラメータ

テンプレートでのパラメータ定義	注記

`- label: images`
`  description: CF and SF images in qvpc-di`
`  parameters:`
`  - qvpc_image_cf`
`  - qvpc_image_sf`
	テンプレートで定義されるイメージのリスト

`- label: networks`
`  description: network configuration for DI`
`  parameters:`
`  - network_di_mgmt`
`  - network_di_internal`
`  - network_public`
`  - network_service1`
`  - network_service2`
`  - network_service3`
`  - network_service4`
	テンプレートで定義されるネットワークのリスト
	（注） この例では、4 個の SF サービスポートをリストしています。最大 12 個の SF サービスポートを定義できます。

VPC-DI HOT パラメータ

Heat テンプレートは、ENV ファイル内に値を指定する必要があるパラメータの数を定義します。これらのパラメータについては、ここで説明します。各パラメータ定義は、heat テンプレートの **parameters** セクションに記載されています。サンプルの ENV ファイルは、パラメータの記述に従っています。

表 *6 : HOT* パラメータ

テンプレートでのパラメータ定義	注記

`flavor_cf:`
`  type: string`
`  description: Flavor for Control Function VM`
`  default: m1.large`
	CF を作成するために使用されるフレーバの名前。これは、5 つのデフォルトのフレーバのいずれかか、または OpenStack で定義されているカスタムフレーバになります。

`flavor_sf:`
`  type: string`
`  description: Flavor for Service Function VM`
`  default: m1.large`
	SF を作成するために使用するフレーバの名前。

`availability_zone:`
`  type: string`
`  description: Availability_zone where the VNF should be created`
`  default: nova`
	VNF が作成されている OpenStack の場所。

テンプレートでのパラメータ定義	注記
qvpc_image_cf: type: string label: Active CF image file in glance description: Active CF image ID or file in glance default: qvpc-di-<*version*>-cf.qcow2 constraints: - custom_constraint: glance.image	CF の VPC-DI イメージファイルの名前。このファイルは一瞬でアップロードされているはずです。
qvpc_image_xf: type: string label: SF image file in glance description: SF image ID or file in glance default: qvpc-di-<*version*>-xf.qcow2 constraints: - custom_constraint: glance.image	サービス機能 VM の VPC-DI イメージファイルの名前。このファイルは一瞬でアップロードされているはずです。

テンプレートでのパラメータ定義	注記
network_public: type: string description: Network ID or Network Name of external network default: public constraints: - custom_constraint: neutron.network	外部ネットワークのネットワーク ID または名前
network_cf_mgmt: type: string description: Management Network ID or Name default: private constraints: - custom_constraint: neutron.network	VPC-DI 管理ネットワークの名前または識別子 。
network_di_internal: type: string description: Unique QVPC-DI internal Network associated with this VNF default: private constraints: - custom_constraint: neutron.network	DI 内部ネットワークの名前または識別子。これは、VPC-DI で VM を相互接続するプライベート L2 ネットワークです。
network_service#: type: string description: Network ID or Network Name of network to use for SF service ports default: cflocal constraints: - custom_constraint: neutron.network	サービスポートの名前または識別子。SF ごとに 1 ～ 12 個のサービスポートを定義できます。ここで、#はこの数値を表します。各サービスポートは異なるサービスを実行できます。
network_core: type: string description: core network for keepalives default: core constraints: - custom_constraint: neutron.network	キープアライブメッセージに使用されるコアネットワークの名前または識別子。

テンプレートでのパラメータ定義	注記
qvpc_vip_addr: type: string description: OAM IP Address shared between CF01 and CF02 default: <*value*> constraints: - custom_constraint: ip_addr	CF 間で使用される仮想 IP アドレス。
qvpc_vip_gateway: type: string description: IP Address of Default Gateway for OAM Network default: <*value*> constraints: - custom_constraint: ip_addr	管理ネットワークのデフォルトゲートウェイ。
vnf_name: type: string description: Unique name for this VNF instance default: qvpc_di	この VNF インスタンスの一意の名前。この名前は、VNF の識別に使用されます。
vnf_id: type: string description: Unique ID for this VNF instance default: 0	VNF インスタンスの ID。
admin_password: type: string description: Default Administrator password for DI Access default: Cisco123	
snmp_community: type: string description: READ SNMP string for this VPC instance default: public	
timezone: type: string description: TimeZone for this VF instance default: us-pacific	
cf_domain_name: type: string description: Domain for this VF instance default: localdomain	
az_cf<#>: type: string description: CF availability zone default: <*value*>	2 つの CF インスタンスそれぞれの可用性ゾーン。
az_sf<#>: type: string description: CF availability zone default:<*value*>	各 SF インスタンスの可用性ゾーン。

これらのパラメータはそれぞれ、ENV ファイルを使用して VNF インスタンス用に定義されています。次に、ENV ファイルの例を示します。

```
parameters:
  # flavor defined for CF and SF in AIC
  flavor_cf: vsaegw_cf
  flavor_sf: vsaegw_sf

  # availability zone where the VNF instance should be deployed
  availability_zone: avzone-kvm-az01

  # vPC-DI glance images in qcow2
  qvpc_image_cf01: QVPCCF
  qvpc_image_sf: QVPCSF

  # Neutron Networks attached to vSAEGW instancenetwork_di_mgmt: oam_protected_net
  network_di_internal: saegw_di_internal_active_net
  network_service1: saegw_gn_net
  network_service2: saegw_sgi_net
  network_service3: saegw_support_net
  network_service4: saegw_icsr_li_net

  # VNF Instance Name
  vnf_name: qvpcDI_vsaegw

  # VNF Instance ID
  vnf_id: 01

  # Administrator user password
  admin_password: cisco123
```

```
parameters:
  flavor_cf:
    type: string
    description: Flavor for Control Function VM
    default: cisco-qvpc-cf
  flavor_sf:
    type: string
    description: Flavor for Service Function VM
    default: cisco-qvpc-xf
  qvpc_image_cf:
    type: string
    label: CF image file in glance
    description: CF image ID or file in glance
    default: qvpc-di-68031-cf.qcow2
    constraints:
      - custom_constraint: glance.image
  qvpc_image_sf:
    type: string
    label: SF image file in glance
    description: SF image ID or file in glance
    default: qvpc-di-68031-xf.qcow2
    constraints:
      - custom_constraint: glance.image
  network_public:
    type: string
    description: Network ID or Network Name of external network
    default: public
    constraints:
      - custom_constraint: neutron.network
  network_cf_mgmt:
    type: string
    description: Management Network ID or Name
    default: cf-mgmt
    constraints:
      - custom_constraint: neutron.network
  network_di_internal:
```

```
    type: string
    description: Unique QVPC-DI internal Network associated with this VNF
    default: di-internal
    constraints:
      - custom_constraint: neutron.network
  network_service1:
    type: string
    description: Transport Interface (Gn/S11/S1-u/S5) in to SAEGW Context
    default: service1
    constraints:
      - custom_constraint: neutron.network
  network_service2:
    type: string
    description: Transport Interface (Data, Voice, LI VLANs) in SGi Context
    default: service2
    constraints:
      - custom_constraint: neutron.network
   network_core:
    type: string
    description: core network for keepalives
    default: core
    constraints:
      - custom_constraint: neutron.network

# vip_addr and vip_gateway are automatically retrieved from the management network
  qvpc_vip_addr:
    type: string
    description: OAM IP Address shared between CF01 and CF02
    default: 172.16.181.2
    constraints:
      - custom_constraint: ip_addr
  qvpc_vip_gateway:
   type: string
   description: IP Address of Default Gateway for OAM Network
   default: 172.16.181.1
   constraints:
      - custom_constraint: ip_addr
  vnf_name:
    type: string
    description: Unique name for this VNF instance
    default: qvpc_di
  vnf_id:
    type: string
    description: Unique ID for this VNF instance
    default: 0
  admin_password:
    type: string
    description: Default Administrator password for DI Access
    default: Cisco123
  snmp_community:
    type: string
    description: READ SNMP string for this VPC instance
    default: public
  timezone:
    type: string
    description: TimeZone for this VF instance
    default: us-pacific
  cf_domain_name:
    type: string
    description: Domain for this VF instance
    default: localdomain
  az_cf1:
    type: string
    description: CF availability zone
```

```
    default: conway1
  az_cf2:
    type: string
    description: CF availability zone
    default: conway2
  az_sf3:
    type: string
    description: SF3 availability zone
    default: conway3
  az_sf4:
    type: string
    description: SF6 availability zone
    default: conway4
```

VPC-DI HOT リソース

テンプレートのリソースセクションでは、制御機能（CF）およびサービス機能（SF）の VM と、それらのポートをそれぞれ定義します。

管理ネットワーク

```
# Create port on management network and reserve a virtual IP address
qvpc_vip_port:
  type: OS::Neutron::Port
  properties:
      network: {get_param: network_di_mgmt}
      fixed_ips:
        - subnet_id: {get_param: subnet_id_di_mgmt}

# Associate a floating IP address to the virutal port
qvpc_vip_floating_ip:
  type: OS::Neutron::FloatingIP
  properties:
      floating_network: {get_param: network_public}
      port_id: {get_resource: qvpc_vip_port}
```

VIP ポートは、VPC-DI へのアクセスに使用される仮想 IP ポートです。VIP ポートの IP アドレスは、Day 0 設定で設定できます。

CF の HOT リソース

Heat テンプレートでは、VNF を使用する 2 つの CF VM それぞれを定義する必要があります。この定義では、DI 内部ネットワークに接続するポートと CF 管理ネットワークに接続するポートを設定し、StarOS 起動パラメータファイルと StarOS Day 0 設定ファイルを指定します。次に最初の CF の定義と説明を示します。2 番目の CF も同様に定義されます。

CF DI 内部ネットワーク

このセクションでは、CF DI 内部ネットワークを作成します。設定する必要がある 2 つの CF のそれぞれに対して、このセクションを 2 回使用します。# は 1 または 2 のいずれかです。

```
# Port connected to unique DI-network
qvpc_cf_0#_port_int:
  type: OS::Neutron::Port
  properties:
```

```
    network: {get_param: network_di_internal}
    allowed_address_pairs:
      -ip_address: "172.16.0.0/18"
```

qvpc_cf_#_port_int は DI 内部ネットワークに接続されているポートです。ネットワークの値は、ENV ファイルから取得したパラメータ **network_di_internal** から抽出されます。

プロパティの allowed_address_pairs は、各 DI 内部ポートに存在する必要があります。di_internal ポートには、17.16.0.0/18 内の VPC-DI によって Neutron 内のアドレスとは異なる IP アドレスが割り当てられるため、allowed_address_pairs プロパティを設定してそれらのアドレスのすべてのトラフィックがそのポートを通過できるようにします。許可されたアドレスペアの拡張機能はポート属性を拡張し、ネットワークに関連付けられているサブネットに関係なく、ポートを通過できる任意の MAC アドレスまたは IP アドレス（CIDR）のペアを指定できるようにします。

CF 管理ネットワーク

このセクションでは、CF 管理ネットワークを作成します。設定する必要がある 2 つの CF のそれぞれに対して、このセクションを 2 回使用します。# は 1 または 2 のいずれかです。

```
# Port connected to the management network
qvpc_cf_0#_port_mgmt:
  type: OS::Neutron::Port
  properties:
    network: {get_param: network_di_mgmt}
    allowed_address_pairs:
      - ip_address: {get_param: qvpc_vip_addr}
```

qvpc_cf_#_port_mgmt は、OAM ネットワークに接続されているポートのポート定義を表します。値は、ENV ファイルから取得したパラメータ **network_di_mgmt** から抽出されます。

SSH キー

DI VM 間通信は、外部から提供された SSH キーによる認証によってのみ可能になりました。これらのキーは、Heat 展開の一部として渡されます。公開キーと秘密キーが必要です。

公開 SSH キーと秘密 SSH キーを生成します。公開キーを含む *user_key.pub* という名前のファイルを作成します。秘密キーを含む *user_key* という名前のファイルを作成します。これらのファイルの両方が設定ドライブに保存されていることを確認します。これらのファイルは、Heat によって参照されます。

```
      personality:
        "user_key.pub": |
                    ssh-rsa
<public_key>
        "user_key": |
                    -----BEGIN RSA PRIVATE KEY-----
<private_key>
                    -----END RSA PRIVATE KEY-----
```

CF VM の作成

このセクションでは、CF VM を作成します。このセクションは作成する必要がある 2 つの CF それぞれに 1 回ずつ、2 回使用します。# は 1 または 2 のいずれかです。

```
qvpc_cf_0#:
  type: OS::Nova::Server
  properties:
    # Create VM of format "<vnf_name>_cf_0#"
    name:
      str_replace:
        template: ${VF_NAME}_cf_0#
        params:
          ${VF_NAME}: {get_param: vnf_name}
    # Use active CF image and CF Flavor
    image: {get_param: qvpc_image_cf1 }
    flavor: {get_param: flavor_cf }
    networks:
      - port: {get_resource: qvpc_cf_0#_port_int}
      - port: {get_resource: qvpc_cf_0#_port_mgmt}
    config_drive: True
```

CF VM（**qvpc_cf_#**）は、以前に定義したパラメータで作成され、「<vnf_name>_cf_#」の表記法に従って名前が付けられます。**vnf_name** は、VNF の作成に使用されるイメージとフレーバと同様に、ENV ファイルから取得されます。

StarOS の Day 0 設定

ここで指定する Day 0 設定は、DI インターフェイス、システムホスト名を設定し、*personality* プロパティを使用して SSH アクセスと SFTP アクセスを有効にします。

```
    # Metadata to provide cloud-init capability to VPC-DI
      personality:
        "staros_param.cfg":
            str_replace:
                template: |
                    CARDSLOT=$CARD_NUMBER
                    CARDTYPE=$CARD_TYPE
                    CPUID=$UUID
                    DI_INTERFACE_MTU=1500
                    DI_INTERFACE=TYPE:virtio_net-1
                    MGMT_INTERFACE=TYPE:virtio_net-2
                    VNFM_INTERFACE=TYPE:virtio_net-3
                    VNFM_IPV4_ENABLE=true
                    VNFM_IPV4_DHCP_ENABLE=true
                    VNFM_PROXY_ADDRS=192.168.180.92,192.168.180.91,192.168.180.93
                params:
                    $CARD_NUMBER: 1
                    $CARD_TYPE: "0x40030100"
                    $UUID: 0
        "staros_config.txt":
            str_replace:
                template: |
                    config
                        system hostname $VF_NAME-cf-$CARD_NUMBER
                        clock timezone $TIMEZONE
                        ssh key-gen wait-time 0
                        context local
                            administrator admin password $ADMIN_PASS ftp
```

```
                                    interface LOCAL1
                                        ip address $CF_VIP_ADDR 255.255.255.0
                                #exit
                                ip route 0.0.0.0 0.0.0.0 $CF_VIP_GATEWAY LOCAL1
                                ip domain-lookup
                                ip domain-name $CF_DOMAIN_NAME
                                ip name-servers $CF_VIP_GATEWAY
                                ssh generate key
                                server sshd
                                    subsystem sftp
                                #exit
                                server confd
                                    confd-user admin
                            #exit
                            port ethernet 1/1
                                bind interface LOCAL1 local
                                no shutdown
                            #exit
                            snmp community $SNMP_COMMUNITY read-only
                        end
                    params:
                        $CARD_NUMBER: 1
                        $VF_NAME: {get_param: vnf_name}
                        $TIMEZONE: {get_param: timezone}
                        $ADMIN_PASS: {get_param: admin_password}
                        $SNMP_COMMUNITY: {get_param: snmp_community}
                        $CF_DOMAIN_NAME: {get_param: cf_domain_name}
                        $SLOT_CARD_NUMBER: 1
                       #$CF_VIP_ADDR: {get_attr: [qvpc_vip_port, fixed_ips, 0, ip_address]}

                        $CF_VIP_ADDR: 172.16.181.2
                      #$CF_VIP_GATEWAY: { get_attr: [qvpc_vip_port, subnets, 0, gateway_ip]
 }
                        $CF_VIP_GATEWAY: 172.16.181.1
        "user_key.pub": |
                        ssh-rsa
<public_key>
        "user_key": |
                        -----BEGIN RSA PRIVATE KEY-----
<private_key>
                        -----END RSA PRIVATE KEY-----
```

$CARD_NUMBER はスロットの番号を参照します。ここでは 1 ですが、2番目の CF では 2 です。

SF の HOT リソース

VPC-DI に展開する各サービス機能（SF）VM を定義するには、Heat テンプレートを使用します。SF ごとに、DI 内部ネットワークと、SF に必要なサービスポートのそれぞれに接続するようにポートを設定する必要があります。設定できるポートは最大 12 個です。この例では、4 つのサービスポートがある SAE ゲートウェイに使用する 1 つの SF を作成します。必要に応じて、SF ごとに同様の設定を繰り返す必要があります。

SF でのポートの定義

```
# Create port for DI-Internal Network
  qvpc_sf_03_port_int:
    type: OS::Neutron::Port
```

```
    properties:
        network: {get_param: network_di_internal}
        allowed_address_pairs:
          - ip_address: "172.16.0.0/18"
```

qvpc_sf_#_port_intは、内部 DI ネットワークに接続するポートです。# は SF の番号であり、3 から SF に許可されている最大数までの番号です。ネットワークの値は、ENV ファイルから取得したパラメータ **network_di_internal** から抽出されます。

```
# Create first service port (document as per your use)
  qvpc_sf_03_port_svc_01:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service1}
```

qvpc_sf_#_port_svc_01は、最初のサービスポートです。ポートには、1 から 12 までの連続番号が付けられます。ネットワークの値は、ENV ファイルから取得したパラメータ **network_service1** から抽出されます。

```
# Create second service port (document as per your use)
  qvpc_sf_03_port_svc_02:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service2}
      allowed_address_pairs:
        - ip_address: "192.168.10.0/24"
# Create third service port  (document as per your use)
  qvpc_sf_03_port_svc_03:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service3}
# Create forth service port  (document as per your use)
  qvpc_sf_03_port_svc_04:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service4}
```

残りの 3 つのサービスポートが作成されます。それぞれが ENV ファイルからネットワーク情報を取得します。必要に応じて追加のサービスポートを作成できます。

SF VM の作成

```
qvpc_sf_03:
  type: OS::Nova::Server
  properties:
# Create VM name of format "<vnf_name>_sf_0<num>"
    name:
      str_replace:
          template: ${VF_NAME}_sf_03
          params:
            ${VF_NAME}: {get_param: vnf_name}
# Use SF image and SF Flavor
    image: { get_param: qvpc_image_sf }
    flavor: { get_param: flavor_sf }
    networks:
      - port: {get_resource: qvpc_sf_03_port_int}
```

```
      - port: {get_resource: qvpc_sf_03_port_svc_01}
      - port: {get_resource: qvpc_sf_03_port_svc_02}
      - port: {get_resource: qvpc_sf_03_port_svc_03}
      - port: {get_resource: qvpc_sf_03_port_svc_04}
    config_drive: True
```

SF **qvpc_sf_#** は「vnf_name_sf_0#」形式の名前で作成されます。ここで、vnf_name は ENV ファイルから取得した VNF 名の値で、# は SF のスロットです。サービスポートの値は、Heat テンプレートに前もって定義されています。イメージとフレーバは、ENV ファイルからも取得されます。

各 SF はテンプレートと同様に定義されます。

パーソナリティの設定

Day 0 と Day 1 の設定は、パーソナリティプロパティを使用して VNF に挿入されます。VPC-DI は、システムにパーソナリティプロパティを適用し、次に示すように、 Heat テンプレートからのこのメタデータを予期します。

パーソナリティは、起動パラメータファイルを定義します。起動パラメータの詳細については、起動パラメータの設定 （7 ページ）を参照してください。

```
# Associate VM to unique slot (>2) and identify that its a SF
       config_drive: True
      personality:
        "staros_param.cfg":
            str_replace:
                template: |
                    CARDSLOT=$CARD_NUMBER
                    CARDTYPE=$CARD_TYPE
                    CPUID=$UUID
                    DI_INTERFACE_MTU=1500
                params:
                    $CARD_NUMBER: 3
                    $CARD_TYPE: "0x42070100"
                    $UUID: 0
        "user_key.pub": |
                    ssh-rsa
<public_key>
        "user_key": |
                    -----BEGIN RSA PRIVATE KEY-----
                    <private_key>
                    -----END RSA PRIVATE KEY-----
```

DI VM 間通信は、外部から提供された SSH キーによる認証によってのみ可能になりました。これらのキーは、Heat 展開の一部として渡されます。公開キーと秘密キーが必要です。

公開 SSH キーと秘密 SSH キーを生成します。公開キーを含む *user_key.pub* という名前のファイルを作成します。秘密キーを含む *user_key* という名前のファイルを作成します。これらのファイルの両方が設定ドライブに保存されていることを確認します。これらのファイルは、上記のように Heat によって参照されます。

VPC-DI HOT 出力

heeatテンプレートの[outputs]セクションには、テンプレートを使用した出力が定義されます。出力を表示するには、**[Project] > [Orchestration] > [Stacks]** に進み、展開した heat スタックを選択します。[Overview] タブには、heat スタックからの出力が表示されます。

コマンドラインで **heat stack-show $[stack_name]** コマンドを実行して、heat スタックからの出力を表示することもできます。

VPC-DI に定義される可能性のある出力のタイプの例を次に示します。

```
qvpc_floating_ip:
  description: Floating IP of qvpc-di VIP
  value: { get_attr: [qvpc_vip_floating_ip, floating_ip_address]}
CF1_networks:
  description: The networks of the deployed CF-1
  value: { get_attr: [qvpc_cf_01, networks] }
CF2_networks_2:
  description: The networks of the deployed CF-2
  value: { get_attr: [qvpc_cf_02, networks] }
port_1_int:
  description: The port of the deployed server 1, di-internal
  value: { get_attr: [qvpc_cf_01_port_int, mac_address] }
port_1_mgmt:
  description: The port of the deployed server 1, cf-mgmt
  value: { get_attr: [qvpc_cf_01_port_mgmt, mac_address] }
port_2_int:
    description: The port of the deployed server 2, di-internal
    value: { get_attr: [qvpc_cf_02_port_int, mac_address] }
port_2_mgmt:
    description: The port of the deployed server 2, cf-mgmt
    value: { get_attr: [qvpc_cf_02_port_mgmt, mac_address] }
```

VMware のインストールに関する注意事項

DI VM 間通信は、外部から提供された SSH キーによる認証によってのみ可能になりました。公開キーと秘密キーが必要です。これらのキーは、ISO の一部として VM を起動する前に指定する必要があります。

キーは外部ホストで生成され、その後 VM に接続する必要がある ISO 内にパッケージ化される必要があります。キーと ISO ファイルは、次のように生成されます。

```
$ mkdir iso
$ ssh-keygen -t rsa -N "" -C "root@localhost" -f iso/user_key
$ genisoimage -o vpcdi_keys.iso iso
```

ISO ファイルが生成されたら、VM の電源を投入し、CD-DVD ROM にマッピングします。これを行うには、vSphere 内でリストから VM（CF または SF）を選択し、上部の近くにあるオプションバーから CD/DVD アイコンをクリックします。次に、[Connect to ISO image on local disk] を選択し、ISO を選択します。すべての VM（CF と SF）に対してこの手順を繰り返します。

キーがマッピングされたら、適切な起動の優先順位を設定して VPC-DI をリロードすることによって VPC-DI の起動設定をイメージにポイントします。