OpenShift Container Platform 4.19

ハードウェアネットワーク

OpenShift Container Platform におけるハードウェア固有のネットワーク機能の設定

Last Updated: 2025-10-26

OpenShift Container Platform 4.19 ハードウェアネットワーク

OpenShift Container Platform におけるハードウェア固有のネットワーク機能の設定

Legal Notice

Abstract

このドキュメントでは、OpenShift Container Platform での Single Root I/O Virtualization (SR-IOV) およびその他のハードウェア固有のネットワーク最適化の設定を説明します。

Table of Contents

第1章 SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて

Single Root I/O Virtualization (SR-IOV) 仕様は、単一デバイスを複数の Pod で共有できる PCI デバイス割り当てタイプの標準です。

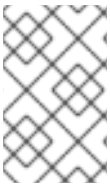
SR-IOV Operator を使用すると、クラスターに Single Root I/O Virtualization (SR-IOV) デバイスを設定できます。

SR-IOV を使用すると、準拠したネットワークデバイス (ホストノードで物理機能 (PF) として認識される) を複数の Virtual Function (VF) にセグメント化することができます。VF は他のネットワークデバイスと同様に使用されます。デバイスの SR-IOV ネットワークデバイスドライバーは、VF がコンテナーで公開される方法を判別します。

- **netdevice** ドライバー: コンテナーの **netns** 内の通常のカーネルネットワークデバイス
- **vfio-pci** ドライバー: コンテナーにマウントされるキャラクターデバイス

SR-IOV ネットワークデバイスは、ベアメタルまたは Red Hat OpenStack Platform (RHOSP) インフラ上にインストールされた OpenShift Container Platform クラスターにネットワークを追加して、高帯域または低遅延を確保する必要のあるアプリケーションに使用できます。

SR-IOV ネットワークのマルチネットワークポリシーを設定できます。これのサポートはテクノロジープレビューであり、SR-IOV 追加ネットワークはカーネル NIC でのみサポートされます。データプレーン開発キット (DPDK) アプリケーションではサポートされていません。



注記

SR-IOV ネットワークでマルチネットワークポリシーを作成しても、マルチネットワークポリシーが設定されていない SR-IOV ネットワークと比較して、アプリケーションに同じパフォーマンスが提供されない場合があります。

重要

SR-IOV ネットワークのマルチネットワークポリシーは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

次のコマンドを使用して、ノードで SR-IOV を有効にできます。

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

1.1. 関連情報

- SR-IOV Network Operator のインストール

1.2. SR-IOV ネットワークデバイスを管理するコンポーネント

SR-IOV Network Operator は SR-IOV スタックのコンポーネントを作成し、管理します。Operator は次の機能を実行します。

- SR-IOV ネットワークデバイスの検出および管理のオーケストレーション
- SR-IOV Container Network Interface (CNI) の **NetworkAttachmentDefinition** カスタムリソースの生成
- SR-IOV ネットワークデバイスプラグインの設定の作成および更新
- ノード固有の **SriovNetworkNodeState** カスタムリソースの作成
- 各 **SriovNetworkNodeState** カスタムリソースの **spec.interfaces** フィールドの更新

Operator は以下のコンポーネントをプロビジョニングします。

SR-IOV ネットワーク設定デーモン

SR-IOV Network Operator の起動時にワーカーノードにデプロイされるデーモンセット。デーモンは、クラスターで SR-IOV ネットワークデバイスを検出し、初期化します。

SR-IOV Network Operator Webhook

Operator カスタムリソースを検証し、未設定フィールドに適切なデフォルト値を設定する動的受付コントローラー Webhook。

SR-IOV Network Resources Injector

SR-IOV VF などのカスタムネットワークリソースの要求および制限のある Kubernetes Pod 仕様のパッチを適用するための機能を提供する動的受付コントローラー Webhook。SR-IOV Network Resources Injector は、Pod 内の最初のコンテナーのみに **resource** フィールドを自動的に追加します。

SR-IOV ネットワークデバイスプラグイン

SR-IOV ネットワーク Virtual Function (VF) リソースの検出、公開、割り当てを実行するデバイスプラグイン。デバイスプラグインは、とりわけ物理デバイスでの制限されたリソースの使用を有効にするために Kubernetes で使用されます。デバイスプラグインは Kubernetes スケジューラーにリソースの可用性を認識させるため、スケジューラーはリソースが十分にあるノードで Pod をスケジュールできます。

SR-IOV CNI プラグイン

SR-IOV ネットワークデバイスプラグインから割り当てられる VF インターフェイスを直接 Pod に割り当てる CNI プラグイン。

SR-IOV InfiniBand CNI プラグイン

SR-IOV ネットワークデバイスプラグインから割り当てられる InfiniBand (IB) VF インターフェイスを直接 Pod に割り当てる CNI プラグイン。

注記

SR-IOV Network Resources Injector および SR-IOV Network Operator Webhook は、デフォルトで有効にされ、**default** の **SriovOperatorConfig** CR を編集して無効にできます。SR-IOV Network Operator Admission Controller Webhook を無効にする場合は注意してください。トラブルシューティングなどの特定の状況下や、サポートされていないデバイスを使用する場合は、Webhook を無効にすることができます。

1.2.1. サポート対象のプラットフォーム

SR-IOV Network Operator は、以下のプラットフォームに対応しています。

- ベアメタル
- Red Hat OpenStack Platform (RHOSP)

1.2.2. サポートされるデバイス

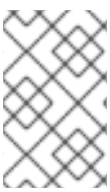
以下のネットワークインターフェイスコントローラーは、OpenShift Container Platform でサポートされています。

表1.1 サポート対象のネットワークインターフェイスコントローラー

製造元	モデル	ベンダー ID	デバイス ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572
Intel	X710 Backplane	8086	1581
Intel	X710 Base T	8086	15ff
Intel	XL710	8086	1583
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Intel	E810-XXVDA4T	8086	1593
Intel	E810-XXV Backplane	8086	1599
Intel	E823L Backplane	8086	124c
Intel	E823L SFP	8086	124d
Intel	E825-C Backplane	8086	579C

製造元	モデル	ベンダー ID	デバイス ID
Intel	E825-C QSFP	8086	579d
Intel	E825-C SFP	8086	579e
Marvell	OCTEON Fusion CNF105XX	177d	ba00
Marvell	OCTEON10 CN10XXX	177d	b900
Mellanox	MT27700 Family [ConnectX-4]	15b3	1013
Mellanox	MT27710 Family [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 Family [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	Mellanox MT2910 ファミリー [ConnectX-7]	15b3	1021
Mellanox	ConnectX-6 NIC モードの MT42822 BlueField-2	15b3	a2d6
Pensando [1]	ionic ドライバー用 DSC-25 デュアルポート 25G 分散サービスカード	0x1dd8	0x1002
Pensando [1]	ionic ドライバー用 DSC-100 デュアルポート 100G 分散サービスカード	0x1dd8	0x1003
Silicom	STS ファミリー	8086	1591

1. OpenShift SR-IOV はサポートされますが、SR-IOV を使用する際に SR-IOV CNI config ファイルを使用して静的な Virtual Function (VF) メディアアクセス制御 (MAC) アドレスを設定する必要があります。



注記

サポートされているカードの最新リストおよび利用可能な互換性のある OpenShift Container Platform バージョンは、Openshift Single Root I/O Virtualization (SR-IOV) and PTP hardware networks Support Matrix を参照してください。

1.3. 関連情報

- マルチネットワークポリシーの設定

1.4. 次のステップ

- SR-IOV Network Operator の設定
- SR-IOV ネットワークデバイスの設定
- OpenShift Virtualization を使用する場合: 仮想マシンの SR-IOV ネットワークへの接続
- SR-IOV ネットワーク割り当ての設定
- イーサネットネットワ割り当て: SR-IOV 追加ネットワークへの Pod の追加
- InfiniBand ネットワーク割り当て: SR-IOV 追加ネットワークへの Pod の追加

第2章 SR-IOV ネットワークデバイスの設定

クラスターで Single Root I/O Virtualization (SR-IOV) デバイスを設定できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

2.1. SR-IOV ネットワークノード設定オブジェクト

SR-IOV ネットワークノードポリシーを作成して、ノードの SR-IOV ネットワークデバイス設定を指定します。ポリシーの API オブジェクトは **sriovnetwork.openshift.io** API グループの一部です。

以下の YAML は SR-IOV ネットワークノードポリシーを説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  needVhostNet: false 7
  numVfs: <num> 8
  externallyManaged: false 9
  nicSelector: 10
    vendor: "<vendor_code>" 11
    deviceID: "<device_id>" 12
    pfNames: ["<pf_name>", ...] 13
    rootDevices: ["<pci_bus_id>", ...] 14
    netFilter: "<filter_string>" 15
  deviceType: <device_type> 16
  isRdma: false 17
  linkType: <link_type> 18
  eSwitchMode: "switchdev" 19
  excludeTopology: false 20
```

1. カスタムリソースオブジェクトの名前。
2. SR-IOV Network Operator がインストールされている namespace。
3. SR-IOV ネットワークデバイスプラグインのリソース名。1 つのリソース名に複数の SR-IOV ネットワークポリシーを作成できます。

   名前を指定するときは、**resourceName** で使用できる構文式 **^[a-zA-Z0-9_]+$** を必ず使用してください。

4.

ノードセレクターは設定するノードを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイス

重要

SR-IOV Network Operator は、ノードネットワーク設定ポリシーを順番にノードに適用します。ノードネットワーク設定ポリシーを適用する前に、SR-IOV Network Operator は、ノードのマシン設定プール (MCP) が **Degraded** または **Updating** などの正常ではない状態にないか確認します。ノード正常ではない MCP にある場合、ノードネットワーク設定ポリシーをクラスター内のすべての対象ノードに適用するプロセスは、MCP が正常な状態に戻るまで一時停止します。

正常でない MCP 内にあるノードが、他のノード (他の MCP 内のノードを含む) にノードネットワーク設定ポリシーを適用することを阻害しないようにするには、MCP ごとに別のノードネットワーク設定ポリシーを作成する必要があります。

5 オプション: 優先度は **0** から **99** までの整数値で指定されます。値が小さいほど優先度が高くなります。たとえば、**10** の優先度は **99** よりも高くなります。デフォルト値は **99** です。

6 オプション: Physical Function とそのす Virtual Function すべての最大転送単位 (MTU)。MTU の最大値は、複数の異なるネットワークインターフェイスコントローラー (NIC) に応じて異なります。

重要

デフォルトのネットワークインターフェイス上に仮想機能を作成する場合は、MTU がクラスター MTU と一致する値に設定されていることを確認してください。

Pod に割り当てられている 1 つの Virtual Function の MTU を変更する場合は、SR-IOV ネットワークノードポリシーで MTU 値を空白のままにします。そうしない場合、SR-IOV Network Operator により Virtual Function の MTU が SR-IOV ネットワークノードポリシーで定義された MTU 値に戻され、ノードドレインがトリガーされる可能性があります。

7 オプション: **/dev/vhost-net** デバイスを Pod にマウントするには、**needVhostNet** を **true** に設定します。Data Plane Development Kit(DPDK) と共にマウントされた **/dev/vhost-net** デバイスを使用して、トラフィックをカーネルネットワークスタックに転送します。

8 SR-IOV 物理ネットワークデバイス用に作成する Virtual Function (VF) の数。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。

9 **externallyManaged** フィールドは、SR-IOV Network Operator が Virtual Function (VF) のすべてを管理するか、またはサブセットのみを管理するかを示します。値を **false** に設定すると、SR-IOV Network Operator が PF 上のすべての VF を管理および設定します。

注記

externallyManaged を **true** に設定した場合、**SriovNetworkNodePolicy** リソースを適用する前に、物理機能 (PF) 上に仮想機能 (VF) を手動で作成する必要があります。VF が事前に作成されていないと、SR-IOV Network Operator の Webhook によってポリシー要求がブロックされます。

externallyManaged を **false** に設定した場合、SR-IOV Network Operator が、VF を自動的に作成および管理し、必要に応じて VF のリセットなどを実行します。

ホストシステムで VF を使用するには、NMState を通じて VF を作成し、**externallyManaged** を **true** に設定する必要があります。このモードでは、SR-IOV Network Operator は、ポリシーの **nicSelector** フィールドで明示的に定義されている VF を除き、PF または手動で管理される VF を変更しません。ただし、SR-IOV Network Operator は、Pod のセカンダリーインターフェイスとして使用される VF を引き続き管理します。

10 NIC セレクターにより、このリソースを適用するデバイスを指定します。すべてのパラメーターの値を指定する必要はありません。意図せずにデバイスを選択しないように、ネットワークデバイスを極めて正確に特定することが推奨されます。

rootDevices を指定する場合、**vendor**、**deviceID**、または **pfNames** の値も指定する必要があります。**pfNames** および **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスを参照していることを確認します。**netFilter** の値を指定する場合、ネットワーク ID は一意の ID であるためにその他のパラメーターを指定する必要はありません。

11 オプション: SR-IOV ネットワークデバイスの 16 進数ベンダー識別子。許可される値は **8086** (Intel) と **15b3** (Mellanox) のみです。

12 オプション: SR-IOV ネットワークデバイスの 16 進数デバイス識別子。たとえば、**101b** は Mellanox ConnectX-6 デバイスのデバイス ID です。

13 オプション: リソースを適用する必要がある 1 つ以上の物理機能 (PF) 名の配列。

14 オプション: リソースを適用する必要がある 1 つ以上の PCI バスアドレスの配列。たとえば、**0000:02:00.1** です。

15 オプション: プラットフォーム固有のネットワークフィルター。サポートされるプラットフォームは Red Hat OpenStack Platform (RHOSP) のみです。設定可能な値は、**openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** の形式を使用します。**xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** を、**/var/config/openstack/latest/network_data.json** メタデータファイルの値に置き換えます。このフィルターにより、VF が特定の OpenStack ネットワークに確実に関連付けられます。Operator はこのフィルターを使用して、OpenStack プラットフォームによって提供されるメタデータに基づき、VF を適切なネットワークにマッピングします。

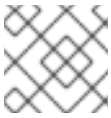
16 オプション: このリソースから作成される VF 用に設定するドライバー。許可される値は **netdevice** および **vfio-pci** のみです。デフォルト値は **netdevice** です。

Mellanox NIC をベアメタルノードの DPDK モードで機能させるには、**netdevice** ドライバータイプを使用し、**isRdma** を **true** に設定します。

17 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを設定します。デフォルト値は **false** です。

isRdma パラメーターが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

isRdma を **true** に設定し、追加の **needVhostNet** を **true** に設定して、Fast Datapath DPDK アプリケーションで使用する Mellanox NIC を設定します。



注記

Intel NIC の場合、**isRdma** パラメーターを **true** に設定することはできません。

18 オプション: VF のリンクタイプ。イーサネットのデフォルト値は **eth** です。InfiniBand の場合、この値を 'ib' に変更します。

linkType が **ib** に設定されている場合、SR-IOV Network Operator Webhook によって **isRdma** は **true** に自動的に設定されます。**linkType** が **ib** に設定されている場合、**deviceType** は **vfio-pci** に設定できません。

SriovNetworkNodePolicy の linkType を **eth** に設定しないでください。デバイスプラグインによって報告される使用可能なデバイスの数が正しくなくなる可能性があります。

19 オプション: ハードウェアオフロードを有効にするには、**eSwitchMode** フィールドを **"switchdev"** に設定する必要があります。ハードウェアオフロードの詳細は、「ハードウェアオフロードの設定」を参照してください。

20 オプション: SR-IOV ネットワークリソースの NUMA ノードを Topology Manager にアドバタイスする場合を除外するには、値を **true** に設定します。デフォルト値は **false** です。

2.1.1. SR-IOV ネットワークノードの設定例

以下の例は、InfiniBand デバイスの設定を示しています。

InfiniBand デバイスの設定例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: <sriov_resource_name>
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: <num>
  nicSelector:
    vendor: "<vendor_code>"
    deviceID: "<device_id>"
    rootDevices:
      - "<pci_bus_id>"
  linkType: <link_type>
  isRdma: true
# ...
```

以下の例は、RHOSP 仮想マシンの SR-IOV ネットワークデバイスの設定を示しています。

仮想マシンの SR-IOV デバイスの設定例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: <sriov_resource_name>
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 (1)
  nicSelector:
    vendor: "<vendor_code>"
    deviceID: "<device_id>"
    netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" (2)
# ...
```

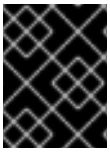
(1) 仮想マシンのノードネットワークポリシーを設定する場合、**numVfs** パラメーターは常に **1** に設定されます。

(2) 仮想マシンが RHOSP にデプロイされる場合、**netFilter** パラメーターはネットワーク ID を参照する必要があります。**netFilter** の有効な値は、**SriovNetworkNodeState** オブジェクトから選択できます。

2.1.2. SR-IOV ネットワークデバイスの自動検出

SR-IOV Network Operator は、クラスターでワーカーノード上の SR-IOV 対応ネットワークデバイスを検索します。Operator は、互換性のある SR-IOV ネットワークデバイスを提供する各ワーカーノードの SriovNetworkNodeState カスタムリソース (CR) を作成し、更新します。

CR にはワーカーノードと同じ名前が割り当てられます。**status.interfaces** リストは、ノード上のネットワークデバイスに関する情報を提供します。



重要

SriovNetworkNodeState オブジェクトは変更しないでください。Operator はこれらのリソースを自動的に作成し、管理します。

2.1.2.1. SriovNetworkNodeState オブジェクトの例

以下の YAML は、SR-IOV Network Operator によって作成される **SriovNetworkNodeState** オブジェクトの例です。

SriovNetworkNodeState オブジェクト

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 (1)
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
```

```
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded
```

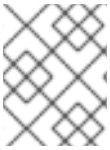
1 **name** フィールドの値はワーカーノードの名前と同じです。

2 **interfaces** スタンザには、ワーカーノード上の Operator によって検出されるすべての SR-IOV デバイスのリストが含まれます。

2.1.3. セキュアブートが有効な場合における Mellanox カードでの SR-IOV Network Operator の設定

SR-IOV Network Operator は、Mellanox デバイスのファームウェア設定をスキップするオプションを

サポートしています。このオプションを使用すると、システムでセキュアブートが有効になっていれば、SR-IOV Network Operator を使用して Virtual Function を作成できます。システムをセキュアブートに切り替える前に、ファームウェアで Virtual Function の数を手動で設定して割り当てる必要があります。



注記

ファームウェア内の Virtual Function の数は、ポリシーで要求できる Virtual Function の最大数です。

手順

1. sriov-config デーモンを使用するときにシステムにセキュアブートがない場合、次のコマンドを実行して Virtual Function (VF) を設定します。

```
$ mstconfig -d -0001:b1:00.1 set SRIOV_EN=1 NUM_OF_VFS=16 1 2
```

1 **SRIOV_EN** 環境変数は、Mellanox カードでの SR-IOV Network Operator サポートを有効にします。

2 **NUM_OF_VFS** 環境変数は、ファームウェアで有効にする Virtual Function の数を指定します。

2. Mellanox プラグインを無効にして SR-IOV Network Operator を設定します。次の **SriovOperatorConfig** 設定例を参照してください。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector: {}
  configurationMode: daemon
  disableDrain: false
  disablePlugins:
  - mellanox
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
```

3. システムを再起動して、Virtual Function と設定を有効にします。

4. 次のコマンドを実行して、システムの再起動後に Virtual Function (VF) を確認します。

```
$ oc -n openshift-sriov-network-operator get sriovnetworknodestate.sriovnetwork.openshift.io
worker-0 -oyaml
```

出力例

```
- deviceID: 101d
    driver: mlx5_core
    eSwitchMode: legacy
```

```
linkSpeed: -1 Mb/s
linkType: ETH
mac: 08:c0:eb:96:31:25
mtu: 1500
name: ens3f1np1
pciAddress: 0000:b1:00.1 (1)
totalvfs: 16
vendor: 15b3
```

1. **totalvfs** 値は、手順の前半の **mstconfig** コマンドで使用した数値と同じです。

5. セキュアブートを有効にすると、デバイスの起動プロセス中に不正なオペレーティングシステムや悪意のあるソフトウェアが読み込まれるのを防ぐことができます。

   a. BIOS (基本入出力システム) を使用して次のパラメーターの値を設定し、セキュアブートを有効にします。

      - **Secure Boot: Enabled**
      - **Secure Boot Policy: Standard**
      - **Secure Boot Mode: Mode Deployed**

   b. システムを再起動します。

2.1.4. SR-IOV デバイスの Virtual Function (VF) パーティション設定

場合によっては、同じ Physical Function (PF) の Virtual Function (VF) を複数のリソースプールに分割することが必要になる場合があります。たとえば、VF の一部をデフォルトドライバーで読み込み、残りの VF を **vfio-pci** ドライバーで読み込む必要がある場合などです。

たとえば、以下の YAML は、VF が **2** から **7** までである **netpf0** という名前のインターフェイスのセレクターを示します。

```
pfNames: ["netpf0#2-7"]
```

ここでは、以下のようになります。

netpf0
: PF インターフェイス名の名前。

2
: 範囲に含まれる最初の VF インデックス (0 ベース)。

7
: 範囲に含まれる最後の VF インデックス (0 ベース)。

次の要件を満たしている場合は、異なるポリシー CR を使用して同じ PF から VF を選択できます。

- 同じ PF を選択するポリシーでは、**numVfs** 値が同一である必要があります。
- VF インデックスは、**0** から **<numVfs>-1** の範囲にある必要があります。たとえば、**numVfs** が **8** に設定されているポリシーがある場合、**<first_vf>** の値は **0** よりも小さくすることはできず、**<last_vf>** は **7** よりも大きくすることはできません。

- 異なるポリシーの VF の範囲は重複しないようにしてください。
- **<first_vf>** は **<last_vf>** よりも大きくすることはできません。

以下の例は、SR-IOV デバイスの NIC パーティション設定を示しています。

policy-net-1 ポリシーは、リソースプール **net-1** を定義します。このリソースプールには、デフォルトの VF ドライバーを持つ PF **netpf0** の VF **0** が含まれます。**policy-net-1-dpdk** ポリシーは、**vfio** VF ドライバーを備えた PF **netpf0** の VF **8** - **15** を含む **net-1-dpdk** リソースプールを定義します。

ポリシー **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

ポリシー **policy-net-1-dpdk**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci
```

インターフェイスが正常にパーティショニングされていることを確認します

- 次のコマンドを実行して、インターフェイスが SR-IOV デバイスの Virtual Function (VF) にパーティショニングされていることを確認します。

```
$ ip link show <interface> 1
```

1 **<interface>** を、SR-IOV デバイスの VF にパーティショニングするときに指定したインターフェイス (例: **ens3f1**) に置き換えます。

出力例

```
5: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 3c:fd:fe:d1:bc:01 brd ff:ff:ff:ff:ff:ff

vf 0     link/ether 5a:e7:88:25:ea:a0 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 1     link/ether 3e:1d:36:d7:3d:49 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 2     link/ether ce:09:56:97:df:f9 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 3     link/ether 5e:91:cf:88:d1:38 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 4     link/ether e6:06:a1:96:2f:de brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
```

2.1.5. OpenStack で SR-IOV を使用するクラスター用のテスト Pod テンプレート

次の **testpmd** Pod では、ヒュージページ、予約済み CPU、および SR-IOV ポートを使用したコンテナーの作成を紹介します。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/sriov1: 1
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
        openshift.io/sriov1: 1
    volumeMounts:
      - mountPath: /dev/hugepages
        name: hugepage
        readOnly: False
  runtimeClassName: performance-cnf-performanceprofile (1)
  volumes:
```

```
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 この例では、パフォーマンスプロファイルの名前が **cnf-performance profile** であると想定しています。

2.1.6. OpenStack で OVS ハードウェアオフロードを使用するクラスター用のテスト Pod テンプレート

次の **testpmd** Pod は、Red Hat OpenStack Platform (RHOSP) での Open vSwitch (OVS) ハードウェアオフロードを示しています。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    k8s.v1.cni.cncf.io/networks: hwoffload1
spec:
  runtimeClassName: performance-cnf-performanceprofile 1
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

2.1.7. Downward API の huge page リソースの挿入

Pod 仕様に huge page のリソース要求または制限が含まれる場合、Network Resources Injector は Downward API フィールドを Pod 仕様に自動的に追加し、huge page 情報をコンテナーに提供します。

Network Resources Injector は、**podnetinfo** という名前のボリュームを追加し、Pod の各コンテナー用に **/etc/podnetinfo** にマウントされます。ボリュームは Downward API を使用し、huge page の要求および制限に関するファイルを追加します。ファイルの命名規則は以下のとおりです。

- **/etc/podnetinfo/hugepages_1G_request_<container-name>**
- **/etc/podnetinfo/hugepages_1G_limit_<container-name>**
- **/etc/podnetinfo/hugepages_2M_request_<container-name>**
- **/etc/podnetinfo/hugepages_2M_limit_<container-name>**

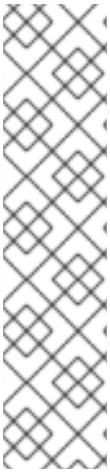
直前のリストで指定されているパスは、**app-netutil** ライブラリーと互換性があります。デフォルトで、ライブラリーは、**/etc/podnetinfo** ディレクトリーのリソース情報を検索するように設定されます。Downward API パス項目を手動で指定する選択をする場合、**app-netutil** ライブラリーは前述のリストのパスに加えて以下のパスを検索します。

- **/etc/podnetinfo/hugepages_request**
- **/etc/podnetinfo/hugepages_limit**
- **/etc/podnetinfo/hugepages_1G_request**
- **/etc/podnetinfo/hugepages_1G_limit**
- **/etc/podnetinfo/hugepages_2M_request**
- **/etc/podnetinfo/hugepages_2M_limit**

Network Resources Injector が作成できるパスと同様に、前述の一覧のパスの末尾にはオプションで **_<container-name>** 接尾辞を付けることができます。

2.2. SR-IOV ネットワークデバイスの設定

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、SriovNetworkNodePolicy カスタムリソース (CR) を作成して設定できます。



注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。再起動は次の場合にのみ行われます。

- Mellanox NIC (**mlx5** ドライバー) の場合、Physical Function (PF) 上の Virtual Function (VF) の数が増加するたびにノードの再起動が行われます。
- Intel NIC の場合、カーネルパラメーターに **intel_iommu=on** と **iommu=pt** が含まれていない場合にのみ再起動が行われます。

設定の変更が適用されるまでに数分かかる場合があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。
- ドレイン (解放) されたノードからエビクトされたワークロードを処理するために、クラスター内に利用可能な十分なノードがある。
- SR-IOV ネットワークデバイス設定にコントロールプレーンノードを選択していない。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **<name>-sriov-node-network.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。
2. オプション: SR-IOV 対応のクラスターノードにまだラベルが付いていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** でラベルを付けます。ノードのラベル付けの詳細は、「ノードのラベルを更新する方法について」を参照してください。
3. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f <name>-sriov-node-network.yaml
```

   ここで、**<name>** はこの設定の名前を指定します。

   設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

4. SR-IOV ネットワークデバイスが設定され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。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

関連情報

- ノードでラベルを更新する方法について

2.3. NON-UNIFORM MEMORY ACCESS (NUMA) に対応した SR-IOV POD の作成

restricted または **single-numa-node** Topology Manager ポリシーを使用して、同じ NUMA ノードから割り当てられる CPU リソースと SR-IOV を制限することで、NUMA に対応した SR-IOV Pod を作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- CPU マネージャーのポリシーを **static** に設定している。CPU マネージャーの詳細は、「関連情報」セクションを参照してください。
- Topology Manager ポリシーを **single-numa-node** に設定している。

注記

single-numa-node が要求を満たさない場合は、Topology Manager ポリシーを **restricted** にするように設定できます。より柔軟な SR-IOV ネットワークリソーススケジューリングは、**関連情報** セクションの **NUMA 対応スケジューリングにおける SR-IOV ネットワークトポロジーの除外** を参照してください。

手順

1. 以下の SR-IOV Pod 仕様を作成してから、YAML を **<name>-sriov-pod.yaml** ファイルに保存します。**<name>** をこの Pod の名前に置き換えます。
   以下の例は、SR-IOV Pod 仕様を示しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> 1
spec:
  containers:
  - name: sample-container
    image: <image> 2
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" 3
        cpu: "2" 4
      requests:
        memory: "1Gi"
        cpu: "2"
```

1. **<name>** を、SR-IOV ネットワーク割り当て定義 CR の名前に置き換えます。
2. **<image>** を **sample-pod** イメージの名前に置き換えます。
3. Guaranteed QoS を指定して SR-IOV Pod を作成するには、**memory requests** に等しい **memory limits** を設定します。

4 Guaranteed QoS を指定して SR-IOV Pod を作成するには、**cpu requests** に等しい **cpu limits** を設定します。

2. 以下のコマンドを実行して SR-IOV Pod のサンプルを作成します。

```
$ oc create -f <filename> 1
```

1 **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. **sample-pod** が Guaranteed QoS を指定して設定されていることを確認します。

```
$ oc describe pod sample-pod
```

4. **sample-pod** が排他的 CPU を指定して割り当てられていることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. **sample-pod** に割り当てられる SR-IOV デバイスと CPU が同じ NUMA ノード上にあることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

2.4. NUMA 対応スケジューリング用の SR-IOV ネットワークトポロジーを除外する場合

NUMA 対応 Pod のスケジューリングでより柔軟な SR-IOV ネットワークデプロイメントを実現するために、SR-IOV ネットワークの Non-Uniform Memory Access (NUMA) ノードを Topology Manager にアドバタイズする場合を除外できます。

一部のシナリオでは、シングル NUMA ノード上の Pod の CPU およびメモリーリソースを最大化することが優先されます。Topology Manager に Pod の SR-IOV ネットワークリソースの NUMA ノードに関するヒントを提供しないことで、Topology Manager は SR-IOV ネットワークリソースと Pod の CPU およびメモリーリソースを異なる NUMA ノードにデプロイできます。その場合、NUMA ノード間のデータ転送により、ネットワーク遅延が増加する可能性があります。ただし、ワークロードが最適な CPU とメモリーのパフォーマンスを必要とするシナリオでは許容されます。

たとえば、2 つの NUMA ノード (**numa0** と **numa1)** を備えたコンピュートノード **compute-1** があるとします。SR-IOV が有効な NIC は **numa0** にあります。Pod のスケジューリングに使用できる CPU は、**numa1** にしかありません。**excludeTopology** 仕様を **true** に設定すると、Topology Manager は Pod の CPU およびメモリーリソースを **numa1** に割り当て、同じ Pod の SR-IOV ネットワークリソースを **numa0** に割り当てることができます。これは、**excludeTopology** 仕様を **true** に設定した場合にのみ可能です。そうではない場合、Topology Manager はすべてのリソースを同じ NUMA ノードに配置しようとします。

2.5. SR-IOV 設定のトラブルシューティング

SR-IOV ネットワークデバイスの設定の手順を実行した後に、以下のセクションではエラー状態の一部に対応します。

手順

- ノードの状態を表示するには、以下のコマンドを実行します。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

ここでは、以下のようになります。

<node_name>
SR-IOV ネットワークデバイスを持つノードの名前を指定します。

コマンドの出力に "cannot allocate memory" と表示される場合は、次の項目を確認してください。

- ノードの BIOS でグローバル SR-IOV 設定が有効になっていることを確認します。
- ノードの BIOS で VT-d が有効であることを確認します。

関連情報

- CPU マネージャーの使用

2.6. 次のステップ

- SR-IOV ネットワーク割り当ての設定

第3章 SR-IOV イーサネットネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスのイーサネットネットワーク割り当てを設定できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

3.1. イーサネットデバイス設定オブジェクト

イーサネットネットワークデバイスは、**SriovNetwork** オブジェクトを定義して設定できます。

以下の YAML は **SriovNetwork** オブジェクトを説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  ipam: |- 7
    {}
  linkState: <link_state> 8
  maxTxRate: <max_tx_rate> 9
  minTxRate: <min_tx_rate> 10
  vlanQoS: <vlan_qos> 11
  trust: "<trust_vf>" 12
  capabilities: <capabilities> 13
```

1. オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
2. SR-IOV Network Operator がインストールされている namespace。
3. この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
4. **SriovNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。
5. オプション: 追加ネットワークに割り当てる VLAN ID を指定します。デフォルト値 **0** の場合、この追加ネットワークには VLAN ID タグが付与されません。サポートされる VLAN ID 値の範囲は **1** - **4094** です。
6. オプション: VF の spoof チェックモード。許可される値は、文字列の **"on"** および **"off"** です。

重要

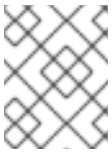
指定する値は引用符で囲む必要があります。引用符で囲まないと、オブジェクトが SR-IOV Network Operator によって拒否されます。

7 YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクトプラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。

8 オプション: Virtual Function (VF) のリンク状態。許可される値は、**enable**、**disable**、および **auto** です。

9 オプション: VF の最大伝送レート (Mbps)。

10 オプション: VF の最小伝送レート (Mbps)。この値は、最大伝送レート以下である必要があります。



注記

Intel NIC は **minTxRate** パラメーターをサポートしません。詳細は、BZ#1772847 を参照してください。

11 オプション: VF の IEEE 802.1p 優先度レベル。デフォルト値は **0** です。

12 オプション: VF の信頼モード。許可される値は、文字列の **"on"** および **"off"** です。

重要

指定する値を引用符で囲む必要があります。囲まないと、SR-IOV Network Operator はオブジェクトを拒否します。

13 オプション: この追加ネットワークに設定する機能。**'{ "ips": true }'** を指定して IP アドレスのサポートを有効にするか、**'{ "mac": true }'** を指定して MAC アドレスのサポートを有効にすることができます。

3.1.1. デュアルスタック IP アドレスを動的に割り当てる設定の作成

デュアルスタックの IP アドレスの割り当ては、**ipRanges** パラメーターで設定できます。

- IPv4 アドレス
- IPv6 アドレス
- 複数の IP アドレスの割り当て

手順

1. **type** を **whereabouts** に設定します。
2. 以下の例のように、**ipRanges** を使用して IP アドレスを割り当てます。

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
```

```
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. ネットワークを Pod にアタッチします。詳細は、「セカンダリーネットワークへの Pod の追加」を参照してください。

4. すべての IP アドレスが割り当てられていることを確認します。

5. 以下のコマンドを実行して、IP アドレスがメタデータとして割り当てられることを確認します。

```
$ oc exec -it mypod -- ip a
```

3.1.2. ネットワークアタッチメントの IP アドレス割り当ての設定

セカンダリーネットワークの場合、Dynamic Host Configuration Protocol (DHCP) や静的割り当てなどのさまざまな割り当て方法をサポートする IP アドレス管理 (IPAM) CNI プラグインを使用して IP アドレスを割り当てることができます。

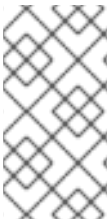
IP アドレスの動的割り当てを担当する DHCP IPAM CNI プラグインは、2 つの異なるコンポーネントを使用して動作します。

- **CNI プラグイン**: Kubernetes ネットワークスタックと統合して IP アドレスを要求および解放する役割を担います。
- **DHCP IPAM CNI デーモン**: 環境内の既存の DHCP サーバーと連携して IP アドレス割り当て要求を処理する DHCP イベントのリスナー。このデーモン自体は DHCP サーバーでは **ありません**。

IPAM 設定で **type: dhcp** を必要とするネットワークの場合は、次の点を確認してください。

- DHCP サーバーが環境内で利用可能かつ実行されている。
- DHCP サーバーはクラスターの外部にあり、そのサーバーがお客様の既存のネットワークインフラストラクチャーに含まれる必要があります。
- DHCP サーバーが、ノードに IP アドレスを提供するように適切に設定されている。

環境内で DHCP サーバーが利用できない場合は、代わりに Whereabouts IPAM CNI プラグインの使用を検討してください。Whereabouts CNI は、外部 DHCP サーバーを必要とせずに同様の IP アドレス管理機能を提供します。



注記

外部 DHCP サーバーが存在しない場合、または静的 IP アドレス管理が優先される場合は、Whereabouts CNI プラグインを使用します。Whereabouts プラグインには、古くなった IP アドレスの割り当てを管理するためのリコンサイラーデーモンが含まれています。

別のデーモンである DHCP IPAM CNI デーモンを組み込むことで、コンテナーの有効期間全体で DHCP リースが定期的に更新されるようにします。DHCP IPAM CNI デーモンをデプロイするには、セカンダリーネットワーク設定の一部としてこのデーモンのデプロイをトリガーするように Cluster Network Operator (CNO) 設定を変更します。

3.1.2.1. 静的 IP アドレス割り当ての設定

以下の表は、静的 IP アドレスの割り当ての設定を説明しています。

表3.1 ipam 静的設定オブジェクト

フィールド	型	説明
type	**string**	IPAM のアドレスタイプ。値 **static** が必要です。
addresses	**array**	仮想インターフェイスに割り当てる IP アドレスを指定するオブジェクトの配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
routes	**array**	Pod 内で設定するルートを指定するオブジェクトの配列です。
dns	**array**	オプション: DNS の設定を指定するオブジェクトの配列です。

addresses の配列には、以下のフィールドのあるオブジェクトが必要です。

表3.2 ipam.addresses[] 配列

フィールド	型	説明
address	**string**	指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、セカンダリーネットワークには IP アドレス **10.10.21.10** とネットマスク **255.255.255.0** が割り当てられます。
gateway	**string**	Egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

表3.3 ipam.routes[] 配列

フィールド	型	説明
dst	**string**	CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
gw	**string**	ネットワークトラフィックをルーティングするゲートウェイ。

表3.4 ipam.dns オブジェクト

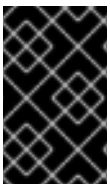
フィールド	型	説明
nameservers	**array**	DNS クエリーが送信される 1 つ以上の IP アドレスの配列。
domain	**array**	ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
search	**array**	DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

静的 IP アドレス割り当ての設定例

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
  }
}
```

3.1.2.2. 動的 IP アドレス (DHCP) 割り当ての設定

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。



重要

イーサネットネットワークアタッチメントの場合、SR-IOV Network Operator は DHCP サーバーデプロイメントを作成しません。Cluster Network Operator は最小限の DHCP サーバーデプロイメントを作成します。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp" 1
        }
      }
  # ...
```

1 クラスターの動的 IP アドレス (DHCP) の割り当てを指定します。

3.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインは、DHCP サーバーを使用せずに、セカンダリーネットワークに IP アドレスを動的に割り当てる場合に役立ちます。

また、Whereabouts CNI プラグインは、重複する IP アドレス範囲と、別々の **NetworkAttachmentDefinition** CRD 内で同じ CIDR 範囲を複数回設定することをサポートしています。これにより、マルチテナント環境での柔軟性と管理機能が向上します。

3.1.2.3.1. 動的 IP アドレス設定パラメーター

以下の表は、Whereabouts を使用した動的 IP アドレス割り当ての設定オブジェクトを説明しています。

表3.5 ipam の whereabouts 設定パラメーター

フィールド	型	説明
type	**string**	IPAM のアドレスタイプ。値 **whereabouts** が必要です。
range	**string**	IP アドレスと範囲を CIDR 表記。IP アドレスは、この範囲内のアドレスから割り当てられます。
exclude	**array**	オプション: CIDR 表記の IP アドレスと範囲 (0 個以上) のリスト。除外されたアドレス範囲内の IP アドレスは割り当てられません。

フィールド	型	説明
network_name	**string**	オプション: 同じ範囲の IP アドレスを共有する場合でも、Pod の各グループまたはドメインが独自の IP アドレスセットを取得するようにします。このフィールドを設定することは、特にマルチテナント環境でネットワークを分離して整理しておく場合に重要です。

3.1.2.3.2. IP アドレス範囲を除外する Whereabouts による動的 IP アドレス割り当て設定

次の例は、Whereabouts を使用する NAD ファイル内の動的アドレス割り当て設定を示しています。

特定の IP アドレス範囲を除外する Whereabouts 動的 IP アドレス割り当て

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

3.1.2.3.3. IP アドレス範囲が重複する場合に Whereabouts を使用した動的 IP アドレス割り当て

次の例は、マルチテナントネットワークで重複する IP アドレスの範囲を使用する、動的な IP アドレスの割り当てを示しています。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", (1)
  }
}
```

(1) オプション: 設定されている場合、**NetworkAttachmentDefinition 2** の **network_name** と一致する必要があります。

NetworkAttachmentDefinition 2

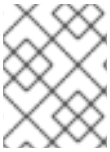
```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
```

```
    "network_name": "example_net_common", (1)
  }
}
```

(1) オプション: 設定されている場合、**NetworkAttachmentDefinition 1** の **network_name** と一致する必要があります。

3.2. SR-IOV の追加ネットワークの設定

SriovNetwork オブジェクトを作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。**SriovNetwork** オブジェクトの作成時に、SR-IOV Network Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。



注記

SriovNetwork オブジェクトが **running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SriovNetwork** オブジェクトを作成してから、YAML を **<name>.yaml** ファイルに保存します。**<name>** はこの追加ネットワークの名前になります。オブジェクト仕様は以下の例のようになります。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc create -f <name>.yaml
```

ここで、**<name>** は追加ネットワークの名前を指定します。

3. オプション: 以下のコマンドを実行して、直前の手順で作成した **SriovNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認するには、以下のコマンドを入力します。**<namespace>** を **SriovNetwork** オブジェクトで指定した networkNamespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

3.3. SR-IOV ネットワークの VRF への割り当て

クラスター管理者は、CNI VRF プラグインを使用して、SR-IOV ネットワークインターフェイスを VRF ドメインに割り当てることができます。

これを実行するには、VRF 設定を **SriovNetwork** リソースのオプションの **metaPlugins** パラメーターに追加します。

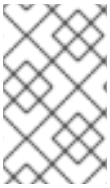
注記

VRF を使用するアプリケーションを特定のデバイスにバインドする必要があります。一般的な使用方法として、ソケットに **SO_BINDTODEVICE** オプションを使用できます。**SO_BINDTODEVICE** は、渡されるインターフェイス名で指定されているデバイスにソケットをバインドします (例: **eth1**)。**SO_BINDTODEVICE** を使用するには、アプリケーションに **CAP_NET_RAW** 機能がある必要があります。

ip vrf exec コマンドを使用した VRF の使用は、OpenShift Container Platform Pod ではサポートされません。VRF を使用するには、アプリケーションを VRF インターフェイスに直接バインドします。

3.3.1. CNI VRF プラグインを使用した追加 SR-IOV ネットワーク割り当ての作成

SR-IOV Network Operator は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、SR-IOV Network Operator は **NetworkAttachmentDefinition** カスタムリソース (CR) を自動的に作成します。



注記

SR-IOV Network Operator が管理する **NetworkAttachmentDefinition** カスタムリソースは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

CNI Virtual Routing and Forwarding (VRF) プラグインを使用して追加の SR-IOV ネットワーク割り当てを作成するには、次の手順を実行します。

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。

手順

1. 追加の SR-IOV ネットワーク割り当て用の **SriovNetwork** カスタムリソース (CR) を作成し、以下のサンプル CR のように **metaPlugins** 設定を挿入します。YAML を **sriov-network-attachment.yaml** ファイルとして保存します。

   SriovNetwork カスタムリソース (CR) の例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: example-network
  namespace: additional-sriov-network-1
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  vlan: 0
  resourceName: intelnics
  metaPlugins : |
    {
      "type": "vrf", (1)
      "vrfname": "example-vrf-name" (2)
    }
```

   (1) **type** パラメーターを **vrf** に設定します。

   (2) **vrfname** パラメーターで VRF の名前を指定します。インターフェイスが VRF に割り当てられます。Pod 内の VRF の名前を指定しない場合は、SR-IOV Network Operator によって VRF の名前が自動的に生成されます。

2. **SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network-attachment.yaml
```

NetworkAttachmentDefinition CR が正常に作成されることの確認

- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。予想される出力には、NAD CR の名前と作成後の経過時間 (分) が表示されます。

```
$ oc get network-attachment-definitions -n <namespace> (1)
```

   (1) **<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます (例: **additional-sriov-network-1**)。

注記

SR-IOV Network Operator が CR を作成するまでに遅延が生じる可能性があります。

追加の SR-IOV ネットワーク割り当てが正常であることの確認

VRF CNI が正しく設定され、追加の SR-IOV ネットワーク割り当てが接続されていることを確認するには、以下を実行します。

1. VRF CNI を使用する SR-IOV ネットワークを作成します。
2. ネットワークを Pod に割り当てます。
3. Pod のネットワーク割り当てが SR-IOV の追加ネットワークに接続されていることを確認します。Pod にリモートシェルログインし、次のコマンドを実行していることを確認します。予想される出力には、VRF インターフェイスの名前とルーティングテーブル内の一意の ID が表示されます。

```
$ ip vrf show
```

4. 次のコマンドを実行して、VRF インターフェイスがセカンダリーインターフェイスの **master** であることを確認します。

```
$ ip link
```

出力例: **5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red state UP mode**

3.4. イーサネットベースの SR-IOV 割り当てのランタイム設定

Pod を追加のネットワークに割り当てる場合、ランタイム設定を指定して Pod の特定のカスタマイズを行うことができます。たとえば、特定の MAC ハードウェアアドレスを要求できます。

Pod 仕様にアノテーションを設定して、ランタイム設定を指定します。アノテーションキーは **k8s.v1.cni.cncf.io/networks** で、ランタイム設定を記述する JSON オブジェクトを受け入れます。

イーサネットベースの SR-IOV ネットワーク割り当てのランタイム設定例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network_attachment>", 1
          "mac": "<mac_address>", 2
          "ips": ["<cidr_range>"] 3
        }
      ]
spec:
  containers:
```

```
- name: sample-container
  image: <image>
  imagePullPolicy: IfNotPresent
  command: ["sleep", "infinity"]
```

1. SR-IOV ネットワーク割り当て定義 CR の名前。値の例は **ibl** です。
2. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレス。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "mac": true }** も指定する必要があります。値の例は **c2:11:22:33:44:55:66:77** です。
3. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。値の例は **192.168.10.1/24", "2001::1/64** です。

3.5. セカンダリーネットワークに POD を追加する

セカンダリーネットワークに Pod を追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、セカンダリーネットワークが Pod にアタッチされます。ただし、Pod がすでに存在する場合は、セカンダリーネットワークをその Pod にアタッチすることはできません。

Pod はセカンダリーネットワークと同じ namespace に存在する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターにログインする。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。
   a. カスタマイズせずにセカンダリーネットワークをアタッチするには、次の形式でアノテーションを追加します。**<network>** が、Pod に関連付けるセカンダリーネットワークの名前に置き換えます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

1. 複数のセカンダリーネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じセカンダリーネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークにアタッチします。

   b. カスタマイズしてセカンダリーネットワークをアタッチするには、次の形式でアノテーションを追加します。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default_route>"] 3
        }
      ]
```

1. **NetworkAttachmentDefinition** オブジェクトによって定義されたセカンダリーネットワークの名前を指定します。
2. **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。
3. オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

```
$ oc create -f <name>.yaml
```

3. オプション: アノテーションが **Pod** CR に存在することを確認するには、**<name>** を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

次の例では、**example-pod** Pod が **net1** セカンダリーネットワークにアタッチされています。

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
          "name": "ovn-kubernetes",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
```

```
    }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1 **k8s.v1.cni.cncf.io/network-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod にアタッチされているセカンダリーネットワークのステータスを表します。アノテーションの値はプレーンテキストの値として保存されます。

3.5.1. vfio-pci SR-IOV デバイスの MTU を Pod に公開する

追加のネットワークに Pod を追加した後、SR-IOV ネットワークで MTU が使用可能であることを確認できます。

手順

1. 次のコマンドを実行して、Pod アノテーションに MTU が含まれていることを確認します。

```
$ oc describe pod example-pod
```

次の例はサンプル出力を示しています。

```
"mac": "20:04:0f:f1:88:01",
      "mtu": 1500,
      "dns": {},
      "device-info": {
        "type": "pci",
        "version": "1.1.0",
        "pci": {
          "pci-address": "0000:86:01.3"
    }
  }
```

2. 次のコマンドを実行して、Pod 内の **/etc/podnetinfo/** で MTU が使用可能であることを確認します。

```
$ oc exec example-pod -n sriov-tests -- cat /etc/podnetinfo/annotations | grep mtu
```

次の例はサンプル出力を示しています。

```
k8s.v1.cni.cncf.io/network-status="[{
    \"name\": \"ovn-kubernetes\",
    \"interface\": \"eth0\",
    \"ips\": [
        \"10.131.0.67\"
    ],
    \"mac\": \"0a:58:0a:83:00:43\",
    \"default\": true,
    \"dns\": {}
    },{
```

```
\"name\": \"sriov-tests/sriov-nic-1\",
\"interface\": \"net1\",
\"ips\": [
    \"192.168.10.1\"
],
\"mac\": \"20:04:0f:f1:88:01\",
\"mtu\": 1500,
\"dns\": {},
\"device-info\": {
    \"type\": \"pci\",
    \"version\": \"1.1.0\",
    \"pci\": {
        \"pci-address\": \"0000:86:01.3\"
    }
}
}]"
```

3.6. SR-IOV ネットワークポリシーの更新中に並列ノードドレインを設定する

デフォルトでは、SR-IOV Network Operator は、ポリシーを変更するたびに、ノードからワークロードをドレイン (解放) します。Operator は、再設定がワークロードに影響を与えないように、このアクションを一度に 1 つのノードで実行します。

大規模なクラスターでは、ノードを順番にドレインするには時間がかかり、数時間または数日かかることもあります。時間に敏感な環境では、**SriovNetworkPoolConfig** カスタムリソース (CR) で並列ノードドレインを有効にして、SR-IOV ネットワーク設定のロールアウトを高速化できます。

並列ドレインを設定するには、**SriovNetworkPoolConfig** CR を使用してノードプールを作成します。次に、プールにノードを追加し、Operator が並行してドレインできるプール内のノードの最大数を定義できます。このアプローチでは、実行中のワークロードを処理するために十分なノードがプール内に残っていることを確認しながら、並列ドレインを有効にして再設定を高速化できます。

注記

ノードは 1 つの SR-IOV ネットワークプール設定にのみ属することができます。ノードがプールに含まれていない場合、そのノードは、一度に 1 つのノードだけをドレインするように設定された仮想のデフォルトプールに追加されます。

ドレイン処理中にノードが再起動する可能性があります。

この手順では、SR-IOV リソースを作成し、ノードを並列ドレインする必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Network Operator がインストールされている。
- ノードには SR-IOV をサポートするハードウェアがある。

手順

1. **SriovNetworkPoolConfig** リソースを定義する YAML ファイルを作成します。

   sriov-nw-pool.yaml ファイルの例

```
apiVersion: v1
kind: SriovNetworkPoolConfig
metadata:
  name: pool-1 (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  maxUnavailable: 2 (3)
  nodeSelector: (4)
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

   (1) **SriovNetworkPoolConfig** オブジェクトの名前を指定します。

   (2) SR-IOV Network Operator がインストールされている namespace を指定します。

   (3) 更新中にプール内で使用できなくなるノードの整数値またはパーセンテージ値を指定します。たとえば、ノードが 10 個あり、使用不可の最大数を 2 に設定した場合は、一度に並列ドレインできるノードは 2 個だけとなり、ワークロードの処理には 8 個のノードが残ります。

   (4) ノードセレクターを使用して、プールを追加するノードを指定します。この例では、**worker** ロールを持つすべてのノードをプールに追加します。

2. 次のコマンドを実行して、**SriovNetworkPoolConfig** リソースを作成します。

```
$ oc create -f sriov-nw-pool.yaml
```

3. 次のコマンドを実行して、**sriov-test** namespace を作成します。

```
$ oc create namespace sriov-test
```

4. **SriovNetworkNodePolicy** リソースを定義する YAML ファイルを作成します。

   sriov-node-policy.yaml ファイルの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames: ["ens1"]
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  numVfs: 5
  priority: 99
  resourceName: sriov_nic_1
```

5. 次のコマンドを実行して、**SriovNetworkNodePolicy** リソースを作成します。

```
$ oc create -f sriov-node-policy.yaml
```

6. **SriovNetwork** リソースを定義する YAML ファイルを作成します。

sriov-network.yaml ファイルの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nic-1
  namespace: openshift-sriov-network-operator
spec:
  linkState: auto
  networkNamespace: sriov-test
  resourceName: sriov_nic_1
  capabilities: '{ "mac": true, "ips": true }'
  ipam: '{ "type": "static" }'
```

7. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network.yaml
```

8. 次のコマンドを実行して、作成したノードプールを表示します。

```
$ oc get sriovNetworkpoolConfig -n openshift-sriov-network-operator
```

予想される出力には、**worker** ロールを持つすべてのノードを含むノードプールの名前 (**pool-1** など) と、ノードプールの秒単位の経過時間 (**67s** など) が表示されます。

9. クラスター内のワークロードのドレインをトリガーするには、**SriovNetworkNodePolicy** リソース内の Virtual Function の数を更新します。

```
$ oc patch SriovNetworkNodePolicy sriov-nic-1 -n openshift-sriov-network-operator --type
merge -p '{"spec": {"numVfs": 4}}'
```

10. 次のコマンドを実行して、ターゲットクラスターのドレイン状態を確認します。

```
$ oc get sriovNetworkNodeState -n openshift-sriov-network-operator
```

出力例

```
NAMESPACE                      NAME       SYNC STATUS   DESIRED SYNC STATE
CURRENT SYNC STATE   AGE
openshift-sriov-network-operator   worker-0   InProgress    Drain_Required
DrainComplete        3d10h
openshift-sriov-network-operator   worker-1   InProgress    Drain_Required
DrainComplete        3d10h
```

ドレインプロセスが完了すると、**SYNC STATUS** が **Succeeded** に変わり、**DESIRED SYNC**

ドレイノノロセスが完了すると、**SYNC STATUS** が **Succeeded** に変わり、**DESIRED SYNC STATE** と **CURRENT SYNC STATE** の値が **IDLE** に戻ります。

3.7. NUMA 対応スケジューリングのための SR-IOV ネットワークトポロジーの除外

SR-IOV ネットワークリソースの Non-Uniform Memory Access (NUMA) ノードを Topology Manager にアドバタイズする場合を除外するには、**SriovNetworkNodePolicy** カスタムリソースで **excludeTopology** 仕様を設定できます。NUMA 対応 Pod のスケジューリングでより柔軟な SR-IOV ネットワークデプロイメントを行うには、この設定を使用します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- CPU マネージャーのポリシーを **static** に設定している。CPU マネージャーの詳細は、**関連情報** セクションを参照してください。
- Topology Manager ポリシーを **single-numa-node** に設定している。
- SR-IOV Network Operator がインストールされている。

手順

1. **SriovNetworkNodePolicy** CR を作成します。
   a. 次の YAML を **sriov-network-node-policy.yaml** ファイルに保存し、環境に合わせて YAML 内の値を置き換えます。

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <policy_name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 (1)
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  numVfs: <number_of_Vfs>
  nicSelector: (2)
    vendor: "<vendor_ID>"
    deviceID: "<device_ID>"
  deviceType: netdevice
  excludeTopology: true (3)
```

(1) SR-IOV ネットワークデバイスプラグインのリソース名。この YAML は、サンプルの **resourceName** 値を使用します。

(2) ネットワークインターフェイスコントローラー (NIC セレクター) を使用して、Operator が設定するデバイスを識別します。

(3) SR-IOV ネットワークリソースの NUMA ノードを Topology Manager にアドバタイスする場合を除外するには、値を **true** に設定します。デフォルト値は **false** です。

注記

多数の **SriovNetworkNodePolicy** リソースが同じ SR-IOV ネットワークリソースをターゲットとしている場合、**SriovNetworkNodePolicy** リソースは **excludeTopology** 仕様と値が同じである必要があります。そうでない場合、矛盾するポリシーは拒否されます。

b. 次のコマンドを実行して、**SriovNetworkNodePolicy** リソースを作成します。成功した出力には、**SriovNetworkNodePolicy** リソースの名前と **created** ステータスがリスト表示されます。

```
$ oc create -f sriov-network-node-policy.yaml
```

2. **SriovNetwork** CR を作成します。

a. 次の YAML を **sriov-network.yaml** ファイルに保存します。その場合、YAML 内の値は環境に合わせて置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-numa-0-network (1)
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 (2)
  networkNamespace: <namespace> (3)
  ipam: |- (4)
    {
      "type": "<ipam_type>",
    }
```

(1) **sriov-numa-0-network** は、SR-IOV ネットワークリソースの名前に置き換えます。

(2) 前の手順で作成した **SriovNetworkNodePolicy** CR のリソース名を指定します。この YAML は、サンプルの **resourceName** 値を使用します。

(3) SR-IOV ネットワークリソースの namespace を入力します。

(4) SR-IOV ネットワークの IP アドレス管理設定を入力します。

b. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。成功した出力には、**SriovNetwork** リソースの名前と **created** ステータスがリスト表示されます。

```
$ oc create -f sriov-network.yaml
```

3. Pod を作成し、前の手順で作成した SR-IOV ネットワークリソースを割り当てます。

a. 次の YAML を **sriov-network-pod.yaml** ファイルに保存します。その場合、YAML 内の値は環境に合わせて置き換えます。

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: <pod_name>
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "sriov-numa-0-network", (1)
        }
      ]
spec:
  containers:
  - name: <container_name>
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

(1) これは、**SriovNetworkNodePolicy** リソースを使用する **SriovNetwork** リソースの名前です。

b. 次のコマンドを実行して、**Pod** リソースを作成します。予想される出力には、**Pod** リソースの名前と **created** ステータスが表示されます。

```
$ oc create -f sriov-network-pod.yaml
```

検証

1. 次のコマンドを実行して、Pod のステータスを確認します。その場合、**<pod_name>** は Pod の名前に置き換えます。

```
$ oc get pod <pod_name>
```

出力例

```
NAME                                     READY   STATUS    RESTARTS   AGE
test-deployment-sriov-76cbbf4756-k9v72   1/1     Running   0          45h
```

2. ターゲット Pod とのデバッグセッションを開き、SR-IOV ネットワークリソースがメモリーおよび CPU リソースとは異なるノードにデプロイされていることを確認します。

a. 次のコマンドを実行して、Pod とのデバッグセッションを開きます。その場合、<pod_name> はターゲット Pod の名前に置き換えます。

```
$ oc debug pod/<pod_name>
```

b. **/host** をデバッグシェル内の root ディレクトリーとして設定します。デバッグ Pod は、Pod 内の **/host** にホストからのルートファイルシステムをマウントします。ルートディレクトリーを **/host** に変更すると、ホストファイルシステムからのバイナリーを実行できます。

```
$ chroot /host
```

c. 次のコマンドを実行して、CPU 割り当てに関する情報を表示します。

```
$ lscpu | grep NUMA
```

出力例

```
NUMA node(s):                2
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,...
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,...
```

```
$ cat /proc/self/status | grep Cpus
```

出力例

```
Cpus_allowed: ffff
Cpus_allowed_list: 1,3,5,7
```

出力には、**NUMA node1** などの **NUMA** ノードに割り当てられる CPU (1、3、5、および 7) が表示されるはずです。SR-IOV ネットワークリソースは、**NUMA node0** などの別の **NUMA** ノードの NIC を使用できます。**ffff** の 16 進値は、プロセスを実行する CPU コアを表すことに注意してください。

```
$ cat  /sys/class/net/net1/device/numa_node
```

出力には、**0** などの **NUMA** ノードの番号が表示されるはずです。

注記

excludeTopology 仕様を **True** に設定すると、必要なリソースが同じ NUMA ノード内に存在する可能性があります。

3.8. 関連情報

- SR-IOV ネットワークデバイスの設定
- CPU マネージャーの使用

第4章 SR-IOV INFINIBAND ネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスの InfiniBand (IB) ネットワーク割り当てを設定できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

4.1. INFINIBAND デバイス設定オブジェクト

SriovIBNetwork オブジェクトを定義することで、InfiniBand (IB) ネットワークデバイスを設定できます。

以下の YAML は、**SriovIBNetwork** オブジェクトを説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  ipam: |- 5
    {}
  linkState: <link_state> 6
  capabilities: <capabilities> 7
```

1. オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
2. SR-IOV Operator がインストールされている namespace。
3. この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
4. **SriovIBNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみをネットワークデバイスに割り当てることができます。
5. オプション: YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクト。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。
6. オプション: Virtual Function (VF) のリンク状態。許可される値は、**enable**、**disable**、および **auto** です。
7. オプション: このネットワークに設定する機能。**'{ "ips": true }'** を指定して IP アドレスのサポートを有効にするか、**'{ "infinibandGUID": true }'** を指定して IB Global Unique Identifier (GUID) のサポートを有効にすることができます。

4.1.1. デュアルスタック IP アドレスを動的に割り当てる設定の作成

デュアルスタックの IP アドレスの割り当ては、**ipRanges** パラメーターで設定できます。

- IPv4 アドレス
- IPv6 アドレス
- 複数の IP アドレスの割り当て

手順

1. **type** を **whereabouts** に設定します。
2. 以下の例のように、**ipRanges** を使用して IP アドレスを割り当てます。

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. ネットワークを Pod にアタッチします。詳細は、「セカンダリーネットワークへの Pod の追加」を参照してください。
4. すべての IP アドレスが割り当てられていることを確認します。
5. 以下のコマンドを実行して、IP アドレスがメタデータとして割り当てられることを確認します。

```
$ oc exec -it mypod -- ip a
```

4.1.2. ネットワークアタッチメントの IP アドレス割り当ての設定

セカンダリーネットワークの場合、Dynamic Host Configuration Protocol (DHCP) や静的割り当てなどのさまざまな割り当て方法をサポートする IP アドレス管理 (IPAM) CNI プラグインを使用して IP アドレスを割り当てることができます。

IP アドレスの動的割り当てを担当する DHCP IPAM CNI プラグインは、2 つの異なるコンポーネントを使用して動作します。

- **CNI プラグイン**: Kubernetes ネットワークスタックと統合して IP アドレスを要求および解放する役割を担います。
- **DHCP IPAM CNI デーモン**: 環境内の既存の DHCP サーバーと連携して IP アドレス割り当て要求を処理する DHCP イベントのリスナー。このデーモン自体は DHCP サーバーでは **ありません**。

IPAM 設定で **type: dhcp** を必要とするネットワークの場合は、次の点を確認してください。

- DHCP サーバーが環境内で利用可能かつ実行されている。
- DHCP サーバーはクラスターの外部にあり、そのサーバーがお客様の既存のネットワークインフラストラクチャーに含まれる必要があります。
- DHCP サーバーが、ノードに IP アドレスを提供するように適切に設定されている。

環境内で DHCP サーバーが利用できない場合は、代わりに Whereabouts IPAM CNI プラグインの使用を検討してください。Whereabouts CNI は、外部 DHCP サーバーを必要とせずに同様の IP アドレス管理機能を提供します。

注記

外部 DHCP サーバーが存在しない場合、または静的 IP アドレス管理が優先される場合は、Whereabouts CNI プラグインを使用します。Whereabouts プラグインには、古くなった IP アドレスの割り当てを管理するためのリコンサイラーデーモンが含まれています。

別のデーモンである DHCP IPAM CNI デーモンを組み込むことで、コンテナーの有効期間全体で DHCP リースが定期的に更新されるようにします。DHCP IPAM CNI デーモンをデプロイするには、セカンダリーネットワーク設定の一部としてこのデーモンのデプロイをトリガーするように Cluster Network Operator (CNO) 設定を変更します。

4.1.2.1. 静的 IP アドレス割り当ての設定

以下の表は、静的 IP アドレスの割り当ての設定を説明しています。

表4.1 ipam 静的設定オブジェクト

フィールド	型	説明
type	**string**	IPAM のアドレスタイプ。値 **static** が必要です。
addresses	**array**	仮想インターフェイスに割り当てる IP アドレスを指定するオブジェクトの配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
routes	**array**	Pod 内で設定するルートを指定するオブジェクトの配列です。
dns	**array**	オプション: DNS の設定を指定するオブジェクトの配列です。

addresses の配列には、以下のフィールドのあるオブジェクトが必要です。

表4.2 ipam.addresses[] 配列

フィールド	型	説明
address	**string**	指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、セカンダリーネットワークには IP アドレス **10.10.21.10** とネットマスク **255.255.255.0** が割り当てられます。
gateway	**string**	Egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

表4.3 ipam.routes[] 配列

フィールド	型	説明
dst	**string**	CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
gw	**string**	ネットワークトラフィックをルーティングするゲートウェイ。

表4.4 ipam.dns オブジェクト

フィールド	型	説明
nameservers	**array**	DNS クエリーが送信される 1 つ以上の IP アドレスの配列。
domain	**array**	ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
search	**array**	DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

静的 IP アドレス割り当ての設定例

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
  }
}
```

4.1.2.2. 動的 IP アドレス (DHCP) 割り当ての設定

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

重要

イーサネットネットワークアタッチメントの場合、SR-IOV Network Operator は DHCP サーバーデプロイメントを作成しません。Cluster Network Operator は最小限の DHCP サーバーデプロイメントを作成します。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp" 1
        }
      }
  # ...
```

1 クラスターの動的 IP アドレス (DHCP) の割り当てを指定します。

4.1.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインは、DHCP サーバーを使用せずに、セカンダリーネットワークに IP アドレスを動的に割り当てる場合に役立ちます。

また、Whereabouts CNI プラグインは、重複する IP アドレス範囲と、別々の **NetworkAttachmentDefinition** CRD 内で同じ CIDR 範囲を複数回設定することをサポートしています。これにより、マルチテナント環境での柔軟性と管理機能が向上します。

4.1.3.1. 動的 IP アドレス設定パラメーター

以下の表は、Whereabouts を使用した動的 IP アドレス割り当ての設定オブジェクトを説明しています。

表4.5 ipam の whereabouts 設定パラメーター

フィールド	型	説明
type	**string**	IPAM のアドレスタイプ。値 **whereabouts** が必要です。
range	**string**	IP アドレスと範囲を CIDR 表記。IP アドレスは、この範囲内のアドレスから割り当てられます。
exclude	**array**	オプション: CIDR 表記の IP アドレスと範囲 (0 個以上) のリスト。除外されたアドレス範囲内の IP アドレスは割り当てられません。
network_name	**string**	オプション: 同じ範囲の IP アドレスを共有する場合でも、Pod の各グループまたはドメインが独自の IP アドレスセットを取得するようにします。このフィールドを設定することは、特にマルチテナント環境でネットワークを分離して整理しておく場合に重要です。

4.1.3.2. IP アドレス範囲を除外する Whereabouts による動的 IP アドレス割り当て設定

次の例は、Whereabouts を使用する NAD ファイル内の動的アドレス割り当て設定を示しています。

特定の IP アドレス範囲を除外する Whereabouts 動的 IP アドレス割り当て

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

4.1.3.3. IP アドレス範囲が重複する場合に Whereabouts を使用した動的 IP アドレス割り当て

次の例は、マルチテナントネットワークで重複する IP アドレスの範囲を使用する、動的な IP アドレスの割り当てを示しています。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", (1)
  }
}
```

(1) オプション: 設定されている場合、**NetworkAttachmentDefinition 2** の **network_name** と一致する必要があります。

NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", (1)
  }
}
```

(1) オプション: 設定されている場合、**NetworkAttachmentDefinition 1** の **network_name** と一致する必要があります。

4.2. SR-IOV の追加ネットワークの設定

SriovIBNetwork オブジェクトを作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。**SriovIBNetwork** オブジェクトの作成時に、SR-IOV Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

注記

SriovIBNetwork オブジェクトが、**running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SriovIBNetwork** オブジェクトを作成し、YAML を **<name>.yaml** ファイルに保存します。**<name>** はこの追加ネットワークの名前です。オブジェクト仕様は以下の例のようになります。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc create -f <name>.yaml
```

ここで、**<name>** は追加ネットワークの名前を指定します。

3. オプション: 以下のコマンドを実行して、直前の手順で作成した **SriovIBNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認します。**<namespace>** を **SriovIBNetwork** オブジェクトで指定した networkNamespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

4.3. INFINIBAND ベースの SR-IOV 割り当てのランタイム設定

Pod を追加のネットワークに割り当てる場合、ランタイム設定を指定して Pod の特定のカスタマイズを行うことができます。たとえば、特定の MAC ハードウェアアドレスを要求できます。

Pod 仕様にアノテーションを設定して、ランタイム設定を指定します。アノテーションキーは **k8s.v1.cni.cncf.io/networks** で、ランタイム設定を記述する JSON オブジェクトを受け入れます。

以下の JSON は、InfiniBand ベースの SR-IOV ネットワーク割り当て用のランタイム設定オプションを説明しています。

```
[
  {
    "name": "<network_attachment>", 1
    "infiniband-guid": "<guid>", 2
    "ips": ["<cidr_range>"] 3
  }
]
```

1 SR-IOV ネットワーク割り当て定義 CR の名前。

2 SR-IOV デバイスの InfiniBand GUIDこの機能を使用するには、**SriovIBNetwork** オブジェクトで **{ "infinibandGUID": true }** も指定する必要があります。

3 SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovIBNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。

ランタイム設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
```

```
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

4.4. セカンダリーネットワークに POD を追加する

セカンダリーネットワークに Pod を追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、セカンダリーネットワークが Pod にアタッチされます。ただし、Pod がすでに存在する場合は、セカンダリーネットワークをその Pod にアタッチすることはできません。

Pod はセカンダリーネットワークと同じ namespace に存在する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターにログインする。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。

   a. カスタマイズせずにセカンダリーネットワークをアタッチするには、次の形式でアノテーションを追加します。**<network>** が、Pod に関連付けるセカンダリーネットワークの名前に置き換えます。

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
   ```

   1 複数のセカンダリーネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じセカンダリーネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークにアタッチします。

   b. カスタマイズしてセカンダリーネットワークをアタッチするには、次の形式でアノテーションを追加します。

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: |-
         [
           {
             "name": "<network>", 1
   ```

```
      "namespace": "<namespace>", 2
      "default-route": ["<default_route>"] 3
    }
  ]
```

1. **NetworkAttachmentDefinition** オブジェクトによって定義されたセカンダリーネットワークの名前を指定します。
2. **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。
3. オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

```
$ oc create -f <name>.yaml
```

3. オプション: アノテーションが **Pod** CR に存在することを確認するには、**<name>** を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

次の例では、**example-pod** Pod が **net1** セカンダリーネットワークにアタッチされています。

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
          "name": "ovn-kubernetes",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

■

1 **k8s.v1.cni.cncf.io/network-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod にアタッチされているセカンダリーネットワークのステータスを表します。アノテーションの値はプレーンテキストの値として保存されます。

4.4.1. vfio-pci SR-IOV デバイスの MTU を Pod に公開する

追加のネットワークに Pod を追加した後、SR-IOV ネットワークで MTU が使用可能であることを確認できます。

手順

1. 次のコマンドを実行して、Pod アノテーションに MTU が含まれていることを確認します。

```
$ oc describe pod example-pod
```

次の例はサンプル出力を示しています。

```
"mac": "20:04:0f:f1:88:01",
      "mtu": 1500,
      "dns": {},
      "device-info": {
        "type": "pci",
        "version": "1.1.0",
        "pci": {
          "pci-address": "0000:86:01.3"
    }
  }
```

2. 次のコマンドを実行して、Pod 内の **/etc/podnetinfo/** で MTU が使用可能であることを確認します。

```
$ oc exec example-pod -n sriov-tests -- cat /etc/podnetinfo/annotations | grep mtu
```

次の例はサンプル出力を示しています。

```
k8s.v1.cni.cncf.io/network-status="[{
    \"name\": \"ovn-kubernetes\",
    \"interface\": \"eth0\",
    \"ips\": [
        \"10.131.0.67\"
    ],
    \"mac\": \"0a:58:0a:83:00:43\",
    \"default\": true,
    \"dns\": {}
    },{
    \"name\": \"sriov-tests/sriov-nic-1\",
    \"interface\": \"net1\",
    \"ips\": [
        \"192.168.10.1\"
    ],
    \"mac\": \"20:04:0f:f1:88:01\",
    \"mtu\": 1500,
```

```
\"dns\": {},
\"device-info\": {
    \"type\": \"pci\",
    \"version\": \"1.1.0\",
    \"pci\": {
        \"pci-address\": \"0000:86:01.3\"
    }
}
}]"
```

4.5. 関連情報

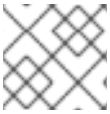
- SR-IOV ネットワークデバイスの設定
- CPU マネージャーの使用
- NUMA 対応スケジューリング用 SR-IOV ネットワークトポロジーの除外

第5章 SR-IOV 用の RDMA サブシステムの設定

Remote Direct Memory Access (RDMA) を使用すると、2 つのシステム間で、どちらのシステムのオペレーティングシステムも介さずに直接メモリーにアクセスできます。Single Root I/O Virtualization (SR-IOV) で RDMA Container Network Interface (CNI) を設定すると、コンテナー間の高性能で低遅延の通信が可能になります。RDMA と SR-IOV を組み合わせると、Data Plane Development Kit (DPDK) アプリケーション内で使用するために Mellanox Ethernet デバイスのハードウェアカウンターを公開するメカニズムが提供されます。

5.1. SR-IOV RDMA CNI の設定

SR-IOV 上で RDMA CNI を設定します。



注記

この手順は Mellanox デバイスにのみ適用されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。

手順

1. 次の例に示すように、**SriovNetworkPoolConfig** CR を作成し、**sriov-nw-pool.yaml** として保存します。

   SriovNetworkPoolConfig CR の例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: worker
  namespace: openshift-sriov-network-operator
spec:
  maxUnavailable: 1
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
  rdmaMode: exclusive (1)
```

   (1) RDMA ネットワーク namespace モードを **exclusive** に設定します。

2. 次のコマンドを実行して、**SriovNetworkPoolConfig** リソースを作成します。

```
$ oc create -f sriov-nw-pool.yaml
```

3. 次の例に示すように、**SriovNetworkNodePolicy** CR を作成し、**sriov-node-policy.yaml** として保存します。

SriovNetworkNodePolicy CR の例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nic-pf1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true (1)
  nicSelector:
    pfNames: ["ens3f0np0"]
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  numVfs: 4
  priority: 99
  resourceName: sriov_nic_pf1
```

(1) RDMA モードをアクティブ化します。

4. 次のコマンドを実行して、**SriovNetworkNodePolicy** リソースを作成します。

```
$ oc create -f sriov-node-policy.yaml
```

5. 次の例に示すように、**SriovNetwork** CR を作成し、**sriov-network.yaml** として保存します。

SriovNetwork CR の例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nic-pf1
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: sriov-tests
  resourceName: sriov_nic_pf1
    ipam: |-
  metaPlugins: |
    {
      "type": "rdma" (1)
    }
```

(1) RDMA プラグインを作成します。

6. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network.yaml
```

検証

1. 次の例に示すように、**Pod** CR を作成し、**sriov-test-pod.yaml** として保存します。

ランタイム設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

2. 次のコマンドを実行してテスト Pod を作成します。

```
$ oc create -f sriov-test-pod.yaml
```

3. 次のコマンドを実行してテスト Pod にログインします。

```
$ oc rsh testpod1 -n sriov-tests
```

4. 次のコマンドを実行して、**hw-counters** ディレクトリーへのパスが存在することを確認します。

```
$ ls
/sys/bus/pci/devices/${PCIDEVICE_OPENSHIFT_IO_SRIOV_NIC_PF1}/infiniband/*/ports/1/hw
_counters/
```

出力例

```
duplicate_request       out_of_buffer req_cqe_flush_error          resp_cqe_flush_error
roce_adp_retrans        roce_slow_restart_trans
implied_nak_seq_err     out_of_sequence req_remote_access_errors
resp_local_length_error     roce_adp_retrans_to     rx_atomic_requests
lifespan                packet_seq_err req_remote_invalid_request   resp_remote_access_errors
roce_slow_restart       rx_read_requests
local_ack_timeout_err  req_cqe_error resp_cqe_error                 rnr_nak_retry_err
roce_slow_restart_cnps  rx_write_requests
```

第6章 SR-IOV ネットワークのインターフェイスレベルのネットワーク SYSCTL 設定とオールマルチキャストモードを設定する

クラスター管理者は、SR-IOV ネットワークデバイスに接続されている Pod のチューニング Container Network Interface (CNI) メタプラグインを使用して、インターフェイスレベルのネットワーク sysctl と、プロミスキャスモード、オールマルチキャストモード、MTU、MAC アドレスなどのいくつかのインターフェイス属性を変更できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

6.1. SR-IOV 対応 NIC を使用したノードのラベル付け

SR-IOV 対応ノードのみで SR-IOV を有効にしたい場合は、いくつかの方法があります。

1. Node Feature Discovery (NFD) Operator をインストールします。NFD は SR-IOV 対応の NIC の存在を検出し、ノードに **node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true** ラベルを付けます。

2. 各ノードの **SriovNetworkNodeState** CR を調べます。**interfaces** スタンザには、ワーカーノード上の SR-IOV Network Operator によって検出されるすべての SR-IOV デバイスの一覧が含まれます。次のコマンドを使用して、各ノードに **feature.node.kubernetes.io/network-sriov.capable: "true"** というラベルを付けます。

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

注記

任意の名前でノードにラベルを付けることができます。

6.2. 1 つの SYSCTL フラグの設定

SR-IOV ネットワークデバイスに接続された Pod のインターフェイスレベルのネットワーク **sysctl** 設定を設定できます。

この例では、作成された仮想インターフェイスで **net.ipv4.conf.IFNAME.accept_redirects** が **1** に設定されます。

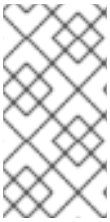
sysctl-tuning-test は、この例で使用される namespace です。

- 次のコマンドを使用して、**sysctl-tuning-test** namespace を作成します。

```
$ oc create namespace sysctl-tuning-test
```

6.2.1. SR-IOV ネットワークデバイスを持つノードで 1 つの sysctl フラグを設定する

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成して設定できます。



注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用すると、SR-IOV Operator がノードをドレインして再起動する場合があります。

設定の変更が適用されるまでに数分の時間がかかる場合があります。

この手順に従って、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。

手順

1. **SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。たとえば、次の YAML をファイル **policyoneflag-sriov-node-network.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyoneflag (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: policyoneflag (3)
  nodeSelector: (4)
    feature.node.kubernetes.io/network-sriov.capable="true"
  priority: 10 (5)
  numVfs: 5 (6)
  nicSelector: (7)
    pfNames: ["ens5"] (8)
  deviceType: "netdevice" (9)
  isRdma: false (10)
```

(1) カスタムリソースオブジェクトの名前。

(2) SR-IOV Network Operator がインストールされている namespace。

(3) SR-IOV ネットワークデバイスプラグインのリソース名。1 つのリソース名に複数の SR-IOV ネットワークポリシーを作成できます。

(4) ノードセレクターは設定するノードを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイスプラグインは、選択したノードにのみデプロイされます。

(5) オプション: 優先度は **0** から **99** までの整数値で指定されます。値が小さいほど優先度が高くなります。たとえば、**10** の優先度は **99** よりも高くなります。デフォルト値は **99** です。

(6) SR-IOV 物理ネットワークデバイス用に作成する Virtual Function (VF) の数。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。

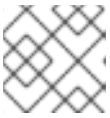
(7) NIC セレクターは、Operator が設定するデバイスを特定します。すべてのパラメーターの値を指定する必要はありません。意図せずにデバイスを選択しないように、ネットワークデバイスを極めて正確に特定することが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfNames** の値も指定する必要があります。**pfNames** お

よび **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスを参照していることを確認します。**netFilter** の値を指定する場合、ネットワーク ID は一意の ID であるためにその他のパラメーターを指定する必要はありません。

8 オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。

9 オプション: Virtual Function のドライバータイプ。許可される唯一の値は **netdevice** です。ベアメタルノードで Mellanox NIC を DPDK モードで動作させるには、**isRdma** を **true** に設定します。

10 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを設定します。デフォルト値は **false** です。**isRdma** パラメーターが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。**isRdma** を **true** に設定し、追加の **needVhostNet** を **true** に設定して、Fast Datapath DPDK アプリケーションで使用する Mellanox NIC を設定します。



注記

vfio-pci ドライバータイプはサポートされていません。

2. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

設定の更新が適用された後に、**sriov-network-operator** namespace 変更のすべての Pod が **Running** ステータスに移行します。

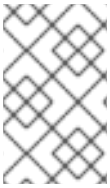
3. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。予想される出力には **Succeeded** が表示されます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

6.2.2. SR-IOV ネットワークでの sysctl の設定

SriovNetwork リソースのオプションの **metaPlugins** パラメーターにチューニング設定を追加することで、SR-IOV により作成された仮想インターフェイスにインターフェイス固有の **sysctl** 設定を設定できます。

SR-IOV Network Operator は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、SR-IOV Network Operator は **NetworkAttachmentDefinition** カスタムリソース (CR) を自動的に作成します。



注記

SR-IOV Network Operator が管理する **NetworkAttachmentDefinition** カスタムリソースは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

インターフェイスレベルのネットワーク **net.ipv4.conf.IFNAME.accept_redirects sysctl** 設定を変更するには、Container Network Interface (CNI) チューニングプラグインを使用して追加の SR-IOV ネットワークを作成します。

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。

手順

1. 追加の SR-IOV ネットワーク割り当て用の **SriovNetwork** カスタムリソース (CR) を作成し、以下のサンプル CR のように **metaPlugins** 設定を挿入します。YAML を **sriov-network-interface-sysctl.yaml** ファイルとして保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: onevalidflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  networkNamespace: sysctl-tuning-test 4
  ipam: '{ "type": "static" }' 5
  capabilities: '{ "mac": true, "ips": true }' 6
  metaPlugins : | 7
    {
      "type": "tuning",
      "capabilities":{
        "mac":true
      },
      "sysctl":{
        "net.ipv4.conf.IFNAME.accept_redirects": "1"
      }
    }
```

1. オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ NetworkAttachmentDefinition オブジェクトを作成します。
2. SR-IOV Network Operator がインストールされている namespace。
3. この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
4. **SriovNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。
5. YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクトプラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。
6. オプション: 追加のネットワークの機能を設定します。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

7 オプション: metaPlugins パラメーターは、デバイスに機能を追加するために使用されます。このユースケースでは、**type** フィールドを **tuning** に設定します。設定したいインターフェイスレベルのネットワーク **sysctl** を **sysctl** フィールドに指定します。

2. **SriovNetwork** リソースを作成します。

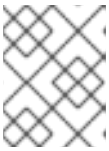
```
$ oc create -f sriov-network-interface-sysctl.yaml
```

NetworkAttachmentDefinition CR が正常に作成されることの確認

- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 **<namespace>** を、**SriovNetwork** オブジェクトで指定した **networkNamespace** の値に置き換えます。たとえば、**sysctl-tuning-test** です。予想される出力には、NAD CRD の名前と作成後の経過時間 (分) が表示されます。



注記

SR-IOV Network Operator が CR を作成するまでに遅延が生じる可能性があります。

追加の SR-IOV ネットワーク割り当てが正常であることの確認

チューニング CNI が正しく設定され、追加の SR-IOV ネットワーク割り当てが接続されていることを確認するには、以下を実行します。

1. **Pod** CR を作成します。次の YAML を **examplepod.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "onevalidflag", 1
          "mac": "0a:56:0a:83:04:0c", 2
          "ips": ["10.100.100.200/24"] 3
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
```

```
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1. SR-IOV ネットワーク割り当て定義 CR の名前。
2. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレス。この機能を使用するには、SriovNetwork オブジェクトで **{ "mac": true }** も指定する必要があります。
3. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。

2. **Pod** CR を作成します。

```
$ oc apply -f examplepod.yaml
```

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n sysctl-tuning-test
```

出力例

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 設定された sysctl フラグの値を確認します。次のコマンドを実行して、**net.ipv4.conf.IFNAME.accept_redirects** の値を見つけます。

```
$ sysctl net.ipv4.conf.net1.accept_redirects
```

6.3. ボンディングされた SR-IOV インターフェイスフラグに関連付けられた POD の SYSCTL 設定の設定

ボンディングされた SR-IOV ネットワークデバイスに接続された Pod のインターフェイスレベルのネットワーク **sysctl** 設定を設定できます。

この例では、設定可能な特定のネットワークインターフェイスレベルの **sysctl** 設定がボンドインターフェイスに設定されています。

sysctl-tuning-test は、この例で使用される namespace です。

- 次のコマンドを使用して、**sysctl-tuning-test** namespace を作成します。

```
$ oc create namespace sysctl-tuning-test
```

6.3.1. SR-IOV ネットワークデバイスがボンドされたノードですべての sysctl フラグを設定する

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成して設定できます。

注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。

設定の変更が適用されるまでに数分かかる場合があります。

この手順に従って、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。

手順

1. **SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。次の YAML を **policyallflags-sriov-node-network.yaml** ファイルとして保存します。**policyallflags** を設定の名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyallflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
  nodeSelector: 4
    node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = `true`
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens1f0"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

1. カスタムリソースオブジェクトの名前。
2. SR-IOV Network Operator がインストールされている namespace。
3. SR-IOV ネットワークデバイスプラグインのリソース名。1 つのリソース名に複数の SR-IOV ネットワークポリシーを作成できます。

4 ノードセレクターは設定するノードを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグイ

5 オプション: 優先度は **0** から **99** までの整数値で指定されます。値が小さいほど優先度が高くなります。たとえば、**10** の優先度は **99** よりも高くなります。デフォルト値は **99** です。

6 SR-IOV 物理ネットワークデバイス用に作成する Virtual Function (VF) の数。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。

7 NIC セレクターは、Operator が設定するデバイスを特定します。すべてのパラメーターの値を指定する必要はありません。意図せずにデバイスを選択しないように、ネットワークデバイスを極めて正確に特定することが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfNames** の値も指定する必要があります。**pfNames** および **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスを参照していることを確認します。**netFilter** の値を指定する場合、ネットワーク ID は一意の ID であるためにその他のパラメーターを指定する必要はありません。

8 オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。

9 オプション: Virtual Function のドライバータイプ。許可される唯一の値は **netdevice** です。ベアメタルノードで Mellanox NIC を DPDK モードで動作させるには、**isRdma** を **true** に設定します。

10 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを設定します。デフォルト値は **false** です。**isRdma** パラメーターが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。**isRdma** を **true** に設定し、追加の **needVhostNet** を **true** に設定して、Fast Datapath DPDK アプリケーションで使用する Mellanox NIC を設定します。

注記

vfio-pci ドライバータイプはサポートされていません。

2. SriovNetworkNodePolicy オブジェクトを作成します。

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

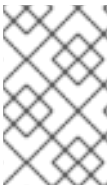
設定の更新が適用された後に、sriov-network-operator namespace のすべての Pod が **Running** ステータスに移行します。

3. SR-IOV ネットワークデバイスが設定され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。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。予想される出力には **Succeeded** が表示されます

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

6.3.2. ボンディングされた SR-IOV ネットワークでの sysctl の設定

2 つの SR-IOV インターフェイスから作成されたボンドインターフェイスで、インターフェイス固有の **sysctl** 設定を設定できます。これを行うには、ボンディングのネットワークアタッチメント定義のオプションの **Plugins** パラメーターにチューニング設定を追加します。



注記

SR-IOV Network Operator が管理する **NetworkAttachmentDefinition** カスタムリソースは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

特定のインターフェイスレベルのネットワーク **sysctl** 設定を変更するには、次の手順を使用して、Container Network Interface (CNI) チューニングプラグインを使用して、**SriovNetwork** カスタムリソース (CR) を作成します。

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。

手順

1. 次の例の CR のように、ボンドされたインターフェイスの **SriovNetwork** カスタムリソース (CR) を作成します。YAML を **sriov-network-attachment.yaml** ファイルとして保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: allvalidflags (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: policyallflags (3)
  networkNamespace: sysctl-tuning-test (4)
  capabilities: '{ "mac": true, "ips": true }' (5)
```

(1) オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ NetworkAttachmentDefinition オブジェクトを作成します。

(2) SR-IOV Network Operator がインストールされている namespace。

(3) この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。

(4) **SriovNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。

(5) オプション: この追加ネットワークに設定する機能。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

2. **SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network-attachment.yaml
```

3. 次の CR の例のように、ボンディングのネットワークアタッチメント定義を作成します。YAML を **sriov-bond-network-interface.yaml** ファイルとして保存します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test
spec:
  config: '{
  "cniVersion":"0.4.0",
  "name":"bound-net",
  "plugins":[
    {
      "type":"bond", (1)
      "mode": "active-backup", (2)
      "failOverMac": 1, (3)
      "linksInContainer": true, (4)
      "miimon": "100",
      "links": [ (5)
        {"name": "net1"},
        {"name": "net2"}
      ],
      "ipam":{ (6)
        "type":"static"
      }
    },
    {
      "type":"tuning", (7)
      "capabilities":{
        "mac":true
      },
      "sysctl":{
        "net.ipv4.conf.IFNAME.accept_redirects": "0",
        "net.ipv4.conf.IFNAME.accept_source_route": "0",
        "net.ipv4.conf.IFNAME.disable_policy": "1",
        "net.ipv4.conf.IFNAME.secure_redirects": "0",
        "net.ipv4.conf.IFNAME.send_redirects": "0",
        "net.ipv6.conf.IFNAME.accept_redirects": "0",
        "net.ipv6.conf.IFNAME.accept_source_route": "1",
        "net.ipv6.neigh.IFNAME.base_reachable_time_ms": "20000",
        "net.ipv6.neigh.IFNAME.retrans_time_ms": "2000"
      }
    }
  ]
}'
```

(1) タイプは **bond** です。

(2) **mode** 属性は、ボンドモードを指定します。サポートされているボンドモードは次のとおりです。

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2
  balance-rr または **balance-xor** モードの場合には、SR-IOV Virtual Function の **trust** モードを **on** に設定する必要があります。

3 **failover** 属性は、active-backup モードでは必須です。

4 **linksInContainer=true** フラグは、必要なインターフェイスがコンテナー内にあることを Bond CNI に通知します。デフォルトでは、Bond CNI はホスト上でこれらのインターフェイスを検索しますが、これは SRIOV および Multus との統合では機能しません。

5 **links** セクションでは、ボンディングの作成に使用するインターフェイスを定義します。デフォルトでは、Multus は接続されたインターフェイスに "net" と 1 から始まる連続した番号の名前を付けます。

6 YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクトプラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。この Pod の例では、IP アドレスは手動で設定されているため、この場合、**ipam** は static に設定されています。

7 デバイスに追加の機能を追加します。たとえば、**type** フィールドを **tuning** に設定します。設定したいインターフェイスレベルのネットワーク **sysctl** を sysctl フィールドに指定します。この例では、設定可能なすべてのインターフェイスレベルのネットワーク **sysctl** 設定を設定します。

4. ボンディングのネットワークアタッチメントリソースを作成します。

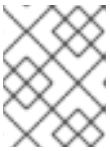
```
$ oc create -f sriov-bond-network-interface.yaml
```

NetworkAttachmentDefinition CR が正常に作成されることの確認

- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 **<namespace>** は、ネットワークアタッチメントの設定時に指定した networkNamespace (例: **sysctl-tuning-test**) に置き換えます。予想される出力には、NAD CRD の名前と作成語の経過時間 (分) が表示されます。



注記

SR-IOV Network Operator が CR を作成するまでに遅延が生じる可能性があります。

SR-IOV ネットワークリソースの追加が成功したことの確認

チューニング CNI が正しく設定され、追加の SR-IOV ネットワーク割り当てが接続されていることを確認するには、以下を実行します。

1. **Pod** CR を作成します。たとえば、次の YAML を **examplepod.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {"name": "allvalidflags"}, (1)
        {"name": "allvalidflags"},
        {
          "name": "bond-sysctl-network",
          "interface": "bond0",
          "mac": "0a:56:0a:83:04:0c", (2)
          "ips": ["10.100.100.200/24"] (3)
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

(1) SR-IOV ネットワーク割り当て定義 CR の名前。

(2) オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレス。この機能を使用するには、SriovNetwork オブジェクトで **{ "mac": true }** も指定する必要があります。

(3) オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。

2. YAML を適用します。

```
$ oc apply -f examplepod.yaml
```

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n sysctl-tuning-test
```

出力例

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 設定された **sysctl** フラグの値を確認します。次のコマンドを実行して、**net.ipv6.neigh.IFNAME.base_reachable_time_ms** の値を見つけます。

```
$ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
```

6.4. オールマルチキャストモード

特にルートレスアプリケーションのコンテキストでは、オールマルチキャストモードを有効にすることが重要です。このモードを有効にしない場合は、Pod のセキュリティーコンテキスト制約 (SCC) に **NET_ADMIN** ケイパビリティーを付与する必要があります。**NET_ADMIN** ケイパビリティーを使用して、特定の要件を超える変更を行う権限を Pod に付与すると、セキュリティーの脆弱性が露呈する可能性があります。

チューニング CNI プラグインは、オールマルチキャストモードを含め、いくつかのインターフェイス属性の変更をサポートしています。このモードを有効にすると、SR-IOV ネットワークデバイス上で設定された Virtual Function (VF) 上で実行されているアプリケーションは、接続されている物理機能が同じか異なるかにかかわらず、他の VF 上のアプリケーションからマルチキャストトラフィックを受信できます。

6.4.1. SR-IOV ネットワーク上でオールマルチキャストモードを有効にする

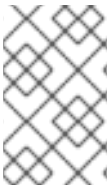
SR-IOV インターフェイスでオールマルチキャストモードを有効にするには、次の方法があります。

- **SriovNetwork** リソースの **metaPlugins** パラメーターにチューニング設定を追加します。
- チューニング設定で、**allmulti** フィールドを **true** に設定します。

注記

信頼を有効にして Virtual Function (VF) を作成していることを確認してください。

SR-IOV Network Operator は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、SR-IOV Network Operator は **NetworkAttachmentDefinition** カスタムリソース (CR) を自動的に作成します。



注記

SR-IOV Network Operator が管理する **NetworkAttachmentDefinition** カスタムリソースは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

このガイダンスに従って、SR-IOV ネットワーク上でオールマルチキャストモードを有効にします。

前提条件

- OpenShift Container Platform CLI (oc) がインストールされている。
- **cluster-admin** 権限を持つユーザーとして OpenShift Container Platform クラスターにログインしている。
- SR-IOV Network Operator がインストールされている。
- 適切な **SriovNetworkNodePolicy** オブジェクトを設定している。

手順

1. Mellanox ConnectX-5 デバイスの **SriovNetworkNodePolicy** オブジェクトを定義する次の設定を使用して、YAML ファイルを作成します。YAML ファイルを **sriovnetpolicy-mlx.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-mlx
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    deviceID: "1017"
    pfNames:
      - ens8f0np0#0-9
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 10
  priority: 99
  resourceName: resourcemlx
```

2. オプション: SR-IOV 対応クラスターノードにラベルが付けられていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** ラベルを追加します。ノードのラベル付けの詳細は、「ノードのラベルを更新する方法について」を参照してください。

3. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f sriovnetpolicy-mlx.yaml
```

   設定の更新を適用すると、**sriov-network-operator** namespace 内のすべての Pod が自動的に **Running** ステータスに移行します。

4. 次のコマンドを実行して、**enable-allmulti-test** namespace を作成します。

```
$ oc create namespace enable-allmulti-test
```

5. 追加の SR-IOV ネットワークアタッチメント用の **SriovNetwork** カスタムリソース (CR) を作成し、次の CR YAML の例のように **metaPlugins** 設定を挿入して、ファイルを **sriov-enable-all-multicast.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: enableallmulti (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: enableallmulti (3)
  networkNamespace: enable-allmulti-test (4)
  ipam: '{ "type": "static" }' (5)
  capabilities: '{ "mac": true, "ips": true }' (6)
  trust: "on" (7)
  metaPlugins : | (8)
   {
     "type": "tuning",
     "capabilities":{
       "mac":true
     },
     "allmulti": true
     }
   }
```

(1) オブジェクトの名前を指定します。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。

(2) SR-IOV Network Operator がインストールされている namespace を指定します。

(3) この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーター値を指定します。

(4) **SriovNetwork** オブジェクトのターゲット namespace を指定します。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。

(5) IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。

(6) オプション: 追加のネットワークの機能を設定します。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

(7) Virtual Function の信頼モードを指定します。これは "on" に設定する必要があります。

(8) **metaPlugins** パラメーターを使用して、デバイスにケイパビリティーをさらに追加します。このユースケースでは、**type** フィールドを **tuning** に設定し、**allmulti** フィールドを追加して **true** に設定します。

6. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。

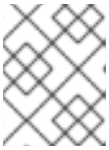
```
$ oc create -f sriov-enable-all-multicast.yaml
```

NetworkAttachmentDefinition CR の検証

- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 **<namespace>** を、**SriovNetwork** オブジェクトで指定した **networkNamespace** の値に置き換えます。この例では、**enable-allmulti-test** です。予想される出力には、NAD CR の名前と作成後の経過時間 (分) が表示されます。



注記

SR-IOV Network Operator が CR を作成するまでに遅延が生じる可能性があります。

- 次のコマンドを実行して、SR-IOV ネットワークリソースに関する情報を表示します。

```
$ oc get sriovnetwork -n openshift-sriov-network-operator
```

追加の SR-IOV ネットワークアタッチメントの検証

チューニング CNI が正しく設定されていること、および追加の SR-IOV ネットワークアタッチメントが割り当てられていることを確認するには、次の手順を実行します。

1. **Pod** CR を作成します。次のサンプル YAML を **examplepod.yaml** という名前のファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: enable-allmulti-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "enableallmulti", 1
          "mac": "0a:56:0a:83:04:0c", 2
          "ips": ["10.100.100.200/24"] 3
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1. SR-IOV network attachment definition CR の名前を指定します。
2. オプション: SR-IOV network attachment definition CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレスを指定します。この機能を使用するには、SriovNetwork オブジェクトで **{"mac": true}** も指定する必要があります。
3. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレスを指定します。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。

2. 以下のコマンドを実行して **Pod** を作成します。

```
$ oc apply -f examplepod.yaml
```

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n enable-allmulti-test
```

出力例

```
NAME      READY   STATUS    RESTARTS   AGE
samplepod  1/1     Running   0          47s
```

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n enable-allmulti-test samplepod
```

5. 次のコマンドを実行して、Pod に関連付けられているすべてのインターフェイスをリスト表示します。

```
sh-4.4# ip link
```

出力例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue state
UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0 (1)
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0 (2)
```

1. **eth0@if22** は、プライマリーインターフェイスです。
2. **net1@if24** は、オールマルチキャストモード (**ALLMULTI** フラグ) をサポートするネットワーク接続定義で設定されたセカンダリーインターフェイスです。

第7章 SR-IOV 対応ワークロードの QINQ サポートの設定

QinQ は正式には 802.1Q-in-802.1Q と呼ばれ、IEEE 802.1ad で定義されたネットワーク技術です。IEEE 802.1ad は IEEE 802.1Q-1998 標準を拡張し、すでに 802.1Q でタグ付けされているパケットに追加の 802.1Q タグを導入することで VLAN 機能を強化します。この方法は、VLAN スタッキングまたはダブル VLAN とも呼ばれます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

7.1. 802.1Q-IN-802.1Q サポートについて

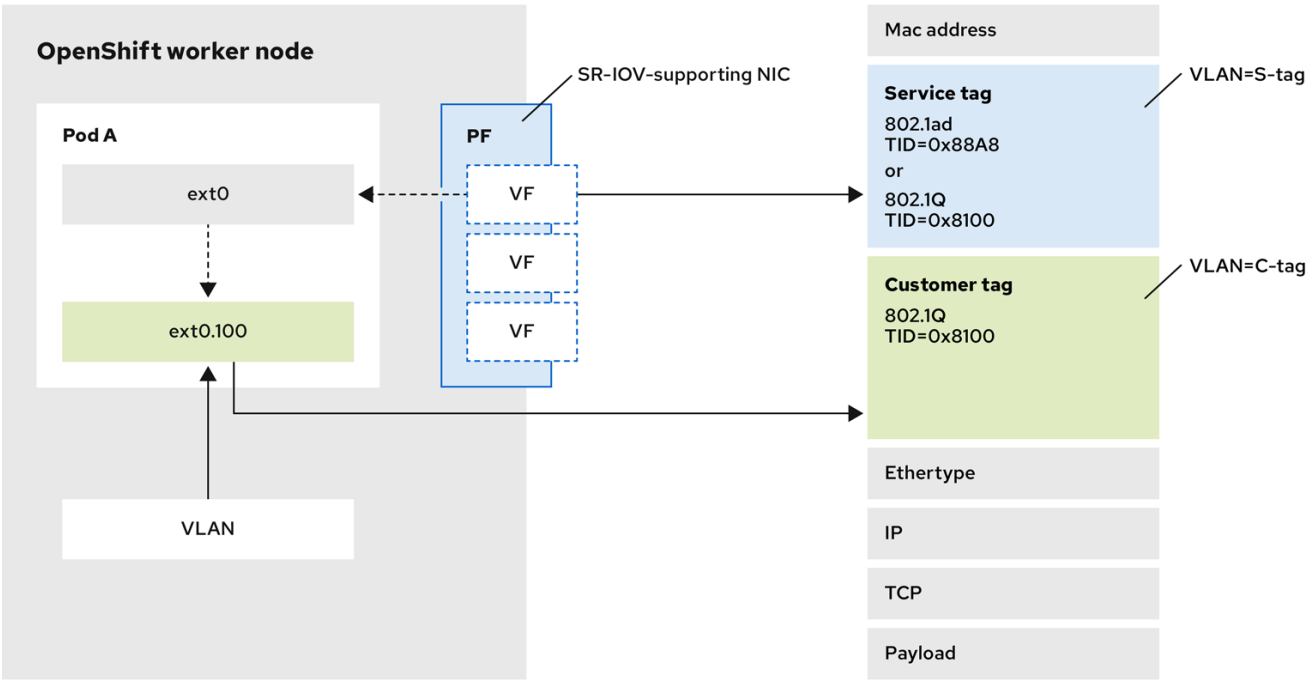
従来の VLAN セットアップでは、フレームには通常、VLAN-100 などの単一の VLAN タグと、Quality of Service (QoS) ビットやプロトコル情報などのその他のメタデータが含まれます。QinQ は 2 番目の VLAN タグを導入します。ここで、サービスプロバイダーは外部タグを自社用に指定して柔軟性を提供し、内部タグは顧客の VLAN 専用のままになります。

QinQ は、二重の VLAN タグ付けを使用してネストされた VLAN の作成を容易にし、ネットワーク環境内でのトラフィックのより細かいセグメンテーションと分離を可能にします。このアプローチは、トラフィックの分離と隔離を確保しながら、共通のインフラストラクチャーを介して複数の顧客に VLAN ベースのサービスを提供する必要があるサービスプロバイダーネットワークで特に役立ちます。

次の図は、OpenShift Container Platform が SR-IOV と QinQ を使用して、コンテナー化されたワークロードの高度なネットワークセグメンテーションと分離を実現する方法を示しています。

この図は、SR-IOV をサポートするワーカーノードでダブル VLAN タグ付け (QinQ) がどのように機能するかを示しています。Pod namespace **ext0** にある SR-IOV Virtual Function (VF) は、VLAN ID と VLAN プロトコルを使用して SR-IOV Container Network Interface (CNI) によって設定されます。これは S-tag に対応します。Pod 内では、VLAN CNI がプライマリーインターフェイス **ext0** を使用してサブインターフェイスを作成します。このサブインターフェイスは、C タグに対応する 802.1Q プロトコルを使用して内部 VLAN ID を追加します。

ここでは、QinQ がネットワーク内でより細かいトラフィックのセグメンテーションと分離を可能にする方法を示しています。右側にはイーサネットフレーム構造の詳細が示されており、VLAN タグ、EtherType、IP、TCP、およびペイロードセクションの両方が含まれていることが強調されています。QinQ は、トラフィックの分離と隔離を確保しながら、共有インフラストラクチャーを介して複数の顧客に VLAN ベースのサービスを提供することを容易にします。



OpenShift Container Platform SR-IOV ソリューションは、**SriovNetwork** カスタムリソース (CR) での VLAN プロトコルの設定をすでにサポートしています。Virtual Function (VF) はこのプロトコルを使用して、外部タグとも呼ばれる VLAN タグを設定できます。その後、Pod は VLAN CNI プラグインを使用して内部タグを設定できます。

表7.1 サポートされているネットワークインターフェイスカード

NIC	802.1ad/802.1Q	802.1Q/802.1Q
Intel X710	いいえ	サポート対象
Intel E810	サポート対象	サポート対象
Mellanox	いいえ	サポート対象

関連情報

VLAN 追加ネットワークの設定

7.2. SR-IOV 対応ワークロードの QINQ サポートの設定

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。

手順

1. 次の内容を使用して、**sriovnetpolicy-810-sriov-node-network.yaml** という名前のファイルを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-810
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames:
      - ens5f0#0-9
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: resource810
```

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f sriovnetpolicy-810-sriov-node-network.yaml
```

3. 別のターミナルウィンドウを開き、次のコマンドを実行して、**openshift-sriov-network-operator** namespace で指定されたノードの SR-IOV ネットワークノード状態の同期ステータスを監視します。

```
$ watch -n 1 'oc get sriovnetworknodestates -n openshift-sriov-network-operator
<node_name> -o jsonpath="{.status.syncStatus}"'
```

同期ステータスが **InProgress** から **Succeeded** に変更されたことを示します。

4. **SriovNetwork** オブジェクトを作成し、インフラストラクチャーに属する S-tag または **Service Tag** と呼ばれる外部 VLAN を設定します。

重要

スイッチのトランクインターフェイスで VLAN を設定する必要があります。さらに、QinQ タグ付けをサポートするために、一部のスイッチをさらに設定することを推奨します。

a. 次の内容を使用して、**nad-sriovnetwork-1ad-810.yaml** という名前のファイルを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriovnetwork-1ad-810
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{}'
  vlan: 171 (1)
```

```
vlanProto: "802.1ad" 2
networkNamespace: default
resourceName: resource810
```

1 S-tag VLAN タグを **171** に設定します。

2 Virtual Function (VF) に割り当てる VLAN プロトコルを指定します。サポートされる値は **802.1ad** と **802.1q** です。デフォルト値は **802.1q** です。

b. 以下のコマンドを実行してオブジェクトを作成します。

```
$ oc create -f nad-sriovnetwork-1ad-810.yaml
```

5. 内部 VLAN を使用して **NetworkAttachmentDefinition** オブジェクトを作成します。内部 VLAN はネットワーク機能に属しているため、多くの場合、C-tag または **Customer Tag** と呼ばれます。

a. 次の内容を使用して、**nad-cvlan100.yaml** という名前のファイルを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: nad-cvlan100
  namespace: default
spec:
  config: '{
    "name": "vlan-100",
    "cniVersion": "0.3.1",
    "type": "vlan",
    "linkInContainer": true,
    "master": "net1", 1
    "vlanId": 100,
    "ipam": {"type": "static"}
  }'
```

1 Pod 内の VF インターフェイスを指定します。Pod アノテーションに名前が設定されていないため、デフォルト名は **net1** になります。

b. 以下のコマンドを実行して、YAML ファイルを適用します。

```
$ oc apply -f nad-cvlan100.yaml
```

検証

- 次の手順に従って、ノード上で QinQ がアクティブであることを確認します。

1. 次の内容を使用して、**test-qinq-pod.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
```

```
    k8s.v1.cni.cncf.io/networks: sriovnetwork-1ad-810, nad-cvlan100
spec:
  containers:
    - name: test-container
      image: quay.io/ocp-edge-qe/cnf-gotests-client:v4.10
      imagePullPolicy: Always
      securityContext:
        privileged: true
```

2. 次のコマンドを実行してテスト Pod を作成します。

```
$ oc create -f test-qinq-pod.yaml
```

3. Pod が存在するターゲットノードでデバッグセッションに入り、次のコマンドを実行してネットワークインターフェイス **ens5f0** に関する情報を表示します。

```
$ oc debug node/my-cluster-node -- bash -c "ip link show ens5f0"
```

出力例

```
6: ens5f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
mode DEFAULT group default qlen 1000
link/ether b4:96:91:a5:22:10 brd ff:ff:ff:ff:ff:ff
vf 0 link/ether a2:81:ba:d0:6f:f3 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 1 link/ether 8a:bb:0a:36:f2:ed brd ff:ff:ff:ff:ff:ff, vlan 171, vlan protocol 802.1ad, spoof
checking on, link-state auto, trust off
vf 2 link/ether ca:0e:e1:5b:0c:d2 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 3 link/ether ee:6c:e2:f5:2c:70 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 4 link/ether 0a:d6:b7:66:5e:e8 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 5 link/ether da:d5:e7:14:4f:aa brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 6 link/ether d6:8e:85:75:12:5c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 7 link/ether d6:eb:ce:9c:ea:78 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 8 link/ether 5e:c5:cc:05:93:3c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
on
vf 9 link/ether a6:5a:7c:1c:2a:16 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
```

出力の **vlan protocol 802.1ad** ID は、インターフェイスがプロトコル 802.1ad (QinQ) による VLAN タグ付けをサポートしていることを示します。VLAN ID は 171 です。

第8章 高パフォーマンスのマルチキャストの使用

Single Root I/O Virtualization (SR-IOV) ハードウェアネットワーク上でマルチキャストを使用できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

8.1. 高パフォーマンスのマルチキャスト

OVN-Kubernetes ネットワークプラグインは、デフォルトネットワーク上の Pod 間のマルチキャストをサポートします。これは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のアプリケーションには適していません。インターネットプロトコルテレビ (IPTV) やマルチポイントビデオ会議など、ストリーミングメディアなどのアプリケーションでは、Single Root I/O Virtualization (SR-IOV) ハードウェアを使用してネイティブに近いパフォーマンスを提供できます。

マルチキャストに追加の SR-IOV インターフェイスを使用する場合:

- マルチキャストパッケージは、追加の SR-IOV インターフェイス経由で Pod によって送受信される必要があります。
- SR-IOV インターフェイスに接続する物理ネットワークは、OpenShift Container Platform で制御されないマルチキャストルーティングとトポロジーを判別します。

8.2. マルチキャストでの SR-IOV インターフェイスの設定

以下の手順では、サンプルのマルチキャスト用の SR-IOV インターフェイスを作成します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. **SriovNetwork** オブジェクトを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | (1)
    {
      "type": "host-local", (2)
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

(1) (2) DHCP を IPAM として設定する選択をした場合は、DHCP サーバー経由でデフォルトルート (**224.0.0.0/5** および **232.0.0.0/5**) をプロビジョニングするようにしてください。これにより、デフォルトのネットワークプロバイダーによって設定された静的なマルチキャストルートが上書きされます。

3. マルチキャストアプリケーションで Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] (1)
    command: [ "sleep", "infinity"]
```

(1) **NET_ADMIN** 機能は、アプリケーションがマルチキャスト IP アドレスを SR-IOV インターフェイスに割り当てる必要がある場合にのみ必要です。それ以外の場合は省略できます。

第9章 DPDK および RDMA の使用

コンテナー化された Data Plane Development Kit (DPDK) アプリケーションは OpenShift Container Platform でサポートされています。Single Root I/O Virtualization (SR-IOV) ネットワークハードウェアは、Data Plane Development Kit (DPDK) および Remote Direct Memory Access (RDMA) で利用できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

9.1. POD での VIRTUAL FUNCTION の使用例

SR-IOV VF が割り当てられている Pod で、Remote Direct Memory Access (RDMA) または Data Plane Development Kit (DPDK) アプリケーションを実行できます。

以下の例では、RDMA モードで Virtual Function (VF) を使用する Pod を示しています。

RDMA モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
    command: ["sleep", "infinity"]
```

以下の例は、DPDK モードの VF のある Pod を示しています。

DPDK モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
```

```
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
      requests:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

9.2. INTEL NIC を使用した DPDK モードでの VIRTUAL FUNCTION の使用

前提条件

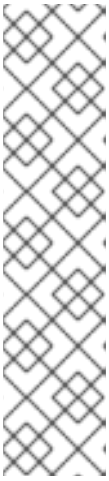
- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **intel-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci (1)
```

(1) Virtual Function のドライバータイプを **vfio-pci** に指定します。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**Configuring SR-IOV network devices** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f intel-dpdk-node-policy.yaml
```

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **intel-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |-
# ... 1
  vlan: <vlan>
  resourceName: intelnics
```

1 IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。

注記

SriovNetwork の各オプションに関する詳細は、「SR-IOV の追加ネットワークの設定」セクションを参照してください。

オプションのライブラリー app-netutil は、コンテナーの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

4. 以下のコマンドを実行して、**SriovNetwork** オブジェクトを作成します。

```
$ oc create -f intel-dpdk-network.yaml
```

5. 以下の **Pod** 仕様を作成してから、YAML を **intel-dpdk-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 **SriovNetwork** オブジェクトの **intel-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を **Pod** 仕様および **SriovNetwork** オブジェクトの両方で変更します。

2 アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3 hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナー内のアプリケーションに必要な追加機能を指定します。

4 hugepage ボリュームを **/mnt/huge** の下の DPDK Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5 オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS

要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。たとえば、カーネル引数 **default_hugepagesz=1GB**、**hugepagesz=1G** および **hugepages=16** を追加すると、**16*1Gi** hugepage がシステムの起動時に割り当てられます。

6. 以下のコマンドを実行して DPDK Pod を作成します。

```
$ oc create -f intel-dpdk-pod.yaml
```

9.3. MELLANOX NIC を使用した DPDK モードでの VIRTUAL FUNCTION の使用

Mellanox NIC で DPDK モードの Virtual Function を使用して、ネットワークノードポリシーを作成し、Data Plane Development Kit (DPDK) Pod を作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- Single Root I/O Virtualization (SR-IOV) Network Operator がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 次の **SriovNetworkNodePolicy** YAML 設定を **mlx-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

1 SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。

2 Virtual Function のドライバータイプを **netdevice** に指定します。Mellanox SR-IOV Virtual Function (VF) は、**vfio-pci** デバイスタイプを使用せずに DPDK モードで機能しま

3 リモートダイレクトメモリーアクセス (RDMA) モードを有効にします。これは、DPDK モードで機能させるために Mellanox カードで必要です。

注記

SriovNetworkNodePolicy オブジェクトの各オプションの詳細な説明は、**SR-IOV ネットワークデバイスの設定** を参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分かかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

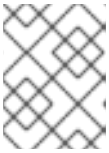
2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

3. 次の **SriovNetwork** YAML 設定を **mlx-dpdk-network.yaml** ファイルに保存します:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- 1
...
  vlan: <vlan>
  resourceName: mlxnics
```

1 IP アドレス管理 (IPAM) コンテナーネットワークインターフェイス (CNI) プラグインの設定オブジェクトを YAML ブロックスカラーとして指定します。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。



注記

SriovNetwork オブジェクトの各オプションの詳細な説明は、**SR-IOV ネットワークデバイスの設定** を参照してください。

app-netutil オプションライブラリーには、コンテナーの親 Pod に関するネットワーク情報を収集するための API メソッドが複数あります。

4. 以下のコマンドを実行して、**SriovNetwork** オブジェクトを作成します。

```
-
```

```
$ oc create -f mlx-dpdk-network.yaml
```

5. 次の **Pod** YAML 設定を **mlx-dpdk-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 **SriovNetwork** オブジェクトの **mlx-dpdk-network** が作成される同じ **target_namespace** を指定します。別の namespace で Pod を作成するには、**Pod** 仕様と **SriovNetwork** オブジェクトの両方で **target_namespace** を変更します。

2 アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3 hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナー内のアプリケーションに必要な追加機能を指定します。

4 hugepage ボリュームを **/mnt/huge** の下の DPDK Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている **emptyDir** ボリュームタイプでサポートされます。

5

オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースイ

6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これを行うには、CPU マネージャーポリシーを **static** に設定し、サービス品質 (QoS) が **Guaranteed** の Pod を作成します。

7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して DPDK Pod を作成します。

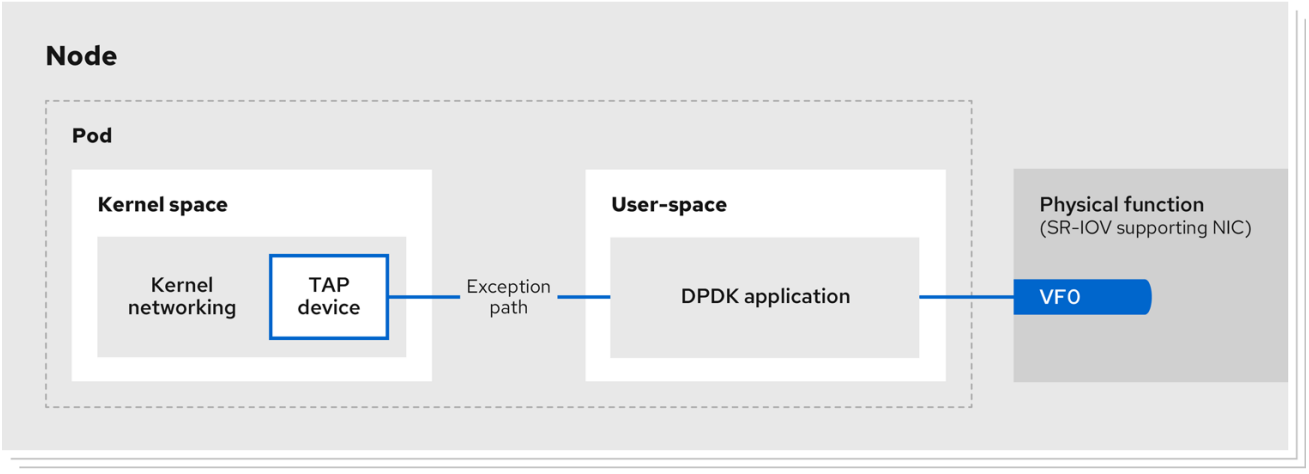
```
$ oc create -f mlx-dpdk-pod.yaml
```

9.4. TAP CNI を使用したカーネルアクセスでのルートレス DPDK ワークロード実行

DPDK アプリケーションは、ログメッセージなどの特定の種類のパケットを処理のためにカーネルに挿入するための例外パスとして **virtio-user** を使用できます。この機能の詳細は、例外パスとしての Virtio_user を参照してください。

OpenShift Container Platform バージョン 4.14 以降では、非権限 Pod を使用して、tap CNI プラグインと一緒に DPDK アプリケーションを実行できます。この機能を有効にするには、**SriovNetworkNodePolicy** オブジェクト内で **needVhostNet** パラメーターを **true** に設定して、**vhost-net** デバイスをマウントする必要があります。

図9.1 DPDK と TAP の設定例



前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

- すべてのノードで **setsebools container_use_devices=on** が root として設定されていることを確認します。

注記

Machine Config Operator を使用して、この SELinux ブール値を設定します。

手順

1. 次の例のような内容を含むファイル (**test-namespace.yaml** など) を作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
```

2. 次のコマンドを実行して、**Namespace** オブジェクトを新規作成します。

```
$ oc apply -f test-namespace.yaml
```

3. 次の例のようなコンテンツを含むファイル (**sriov-node-network-policy.yaml** など) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice (1)
 isRdma: true (2)
 needVhostNet: true (3)
 nicSelector:
   vendor: "15b3" (4)
   deviceID: "101b" (5)
   rootDevices: ["00:05.0"]
 numVfs: 10
 priority: 99
 resourceName: sriovnic
 nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

(1) これは、プロファイルが Mellanox ネットワークインターフェイスコントローラー (NIC) 専用に調整されていることを示します。

(2) **isRdma** を **true** に設定する必要があるのは、Mellanox NIC の場合のみです。

(3)

これにより、**/dev/net/tun** および **/dev/vhost-net** デバイスがコンテナーにマウントされ、アプリケーションがタップデバイスを作成し、タップデバイスを DPDK ワークロードに接

4 SR-IOV ネットワークデバイスのベンダーの 16 進数コード。値 15b3 は Mellanox NIC に関連付けられています。

5 SR-IOV ネットワークデバイスのデバイスの 16 進数コード。

4. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f sriov-node-network-policy.yaml
```

5. 次の **SriovNetwork** オブジェクトを作成し、YAML を **sriov-network-attachment.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
 resourceName: sriovnic
 spoofChk: "off"
 trust: "on"
```

注記

SriovNetwork の各オプションに関する詳細は、「SR-IOV の追加ネットワークの設定」セクションを参照してください。

オプションのライブラリー **app-netutil** は、コンテナーの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

6. 以下のコマンドを実行して、**SriovNetwork** オブジェクトを作成します。

```
$ oc create -f sriov-network-attachment.yaml
```

7. 次の例のような内容を含む、ネットワーク割り当て定義を指定するファイル (**tap-example.yaml** など) を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: tap-one
 namespace: test-namespace 1
spec:
 config: '{
   "cniVersion": "0.4.0",
   "name": "tap",
   "plugins": [
    {
       "type": "tap",
```

```
      "multiQueue": true,
      "selinuxcontext": "system_u:system_r:container_t:s0"
    },
    {
     "type":"tuning",
     "capabilities":{
      "mac":true
     }
    }
  ]
}'
```

1. **SriovNetwork** オブジェクトが作成されるのと同じ **target_namespace** を指定します。

8. 次のコマンドを実行して、**NetworkAttachmentDefinition** オブジェクトを作成します。

```
$ oc apply -f tap-example.yaml
```

9. 次の例のような内容を含むファイル (**dpdk-pod-rootless.yaml** など) を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: test-namespace (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {"name": "sriov-network", "namespace": "test-namespace"},
      {"name": "tap-one", "interface": "ext0", "namespace": "test-namespace"}]'
spec:
  nodeSelector:
    kubernetes.io/hostname: "worker-0"
  securityContext:
      fsGroup: 1001 (2)
      runAsGroup: 1001 (3)
      seccompProfile:
        type: RuntimeDefault
  containers:
  - name: testpmd
    image: <DPDK_image> (4)
    securityContext:
      capabilities:
        drop: ["ALL"] (5)
        add: (6)
          - IPC_LOCK
          - NET_RAW #for mlx only (7)
      runAsUser: 1001 (8)
      privileged: false (9)
      allowPrivilegeEscalation: true (10)
      runAsNonRoot: true (11)
    volumeMounts:
    - mountPath: /mnt/huge (12)
      name: hugepages
```

```
    resources:
      limits:
        openshift.io/sriovnic: "1" (13)
        memory: "1Gi"
        cpu: "4" (14)
        hugepages-1Gi: "4Gi" (15)
      requests:
        openshift.io/sriovnic: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  runtimeClassName: performance-cnf-performanceprofile (16)
  volumes:
  - name: hugepages
    emptyDir:
      medium: HugePages
```

(1) **SriovNetwork** オブジェクトが作成されるのと同じ **target_namespace** を指定します。Pod を別の namespace に作成する場合は、**target_namespace** を **Pod** 仕様と **SriovNetwork** オブジェクトの両方で変更します。

(2) ボリュームにマウントされたディレクトリーおよびそれらのボリューム内に作成されたファイルのグループ所有権を設定します。

(3) コンテナーの実行に使用するプライマリーグループ ID を指定します。

(4) アプリケーションを含む DPDK イメージとアプリケーションで使用される DPDK ライブラリーを指定します。

(5) コンテナーの securityContext からすべての機能 (**ALL**) を削除すると、通常の操作に必要とされる権限以上の権限がコンテナーからなくなります。

(6) hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナー内のアプリケーションに必要な追加機能を指定します。これらの機能は、**setcap** コマンドを使用してバイナリーファイルでも設定する必要があります。

(7) Mellanox ネットワークインターフェイスコントローラー (NIC) には、**NET_RAW** 機能が必要です。

(8) コンテナーの実行に使用するユーザー ID を指定します。

(9) この設定で、Pod 内のコンテナー (複数可) にホストシステムへの権限アクセスを許可しないように指定します。

(10) この設定を使用すると、コンテナーは、割り当てられている初期の root 以外の権限を超えて権限を昇格できます。

(11) また、この設定により、コンテナーは root 以外のユーザーで実行されます。これにより、最小権限の原則が適用され、コンテナーが不正アクセスされる可能性を最小限に抑えるとともに、攻撃対象領域を減少させます。

(12) hugepage ボリュームを **/mnt/huge** の下の DPDK Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

13 オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースイ

14 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

15 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。たとえば、カーネル引数 **default_hugepagesz=1GB**、**hugepagesz=1G** および **hugepages=16** を追加すると、**16*1Gi** hugepage がシステムの起動時に割り当てられます。

16 パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

10. 以下のコマンドを実行して DPDK Pod を作成します。

```
$ oc create -f dpdk-pod-rootless.yaml
```

関連情報

- パフォーマンスプロファイルの作成
- SR-IOV ネットワークデバイスの設定

9.5. 特定の DPDK ラインレート達成に関する概要

特定の Data Plane Development Kit (DPDK) ラインレートを実現するには、Node Tuning Operator をデプロイし、Single Root I/O Virtualization (SR-IOV) を設定します。次のリソースの DPDK 設定も調整する必要があります。

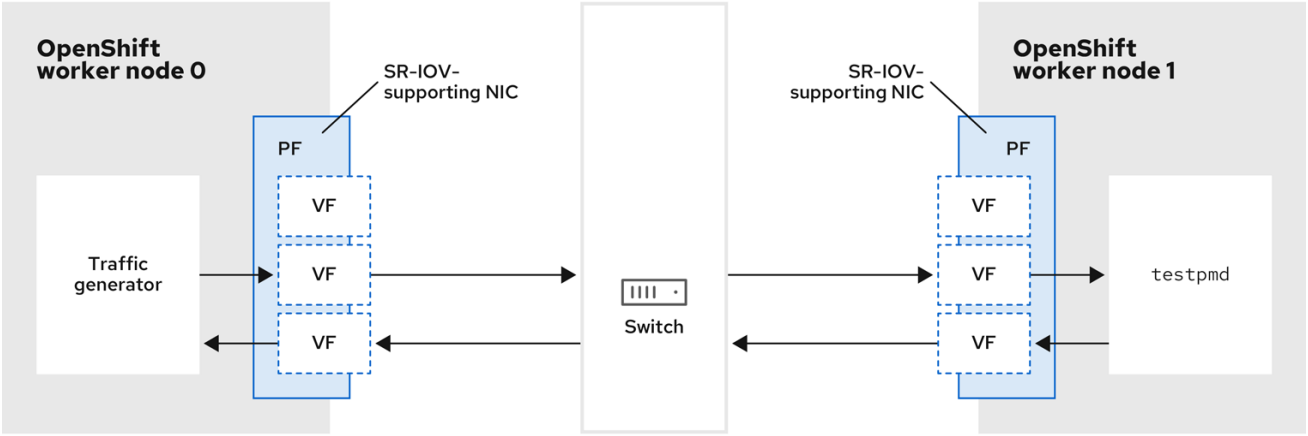
- 分離された CPU
- hugepage
- トポロジースケジューラー

注記

OpenShift Container Platform の以前のバージョンでは、パフォーマンスアドオン Operator を使用して自動チューニングを実装し、OpenShift Container Platform アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

DPDK テスト環境

次の図は、トラフィックテスト環境のコンポーネントを示しています。



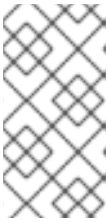
- **トラフィックジェネレーター**: 大量のパケットトラフィックを生成できるアプリケーション。
- **SR-IOV 対応 NIC**: SR-IOV に対応したネットワークインターフェイスカードです。カードは、物理インターフェイス上で多数の Virtual Function を実行します。
- **Physical Function (PF)**: SR-IOV インターフェイスをサポートするネットワークアダプターの PCI Express (PCIe) 機能。
- **Virtual Function (VF)**: SR-IOV をサポートするネットワークアダプター上の軽量の PCIe 機能。VF は、ネットワークアダプターの PCIe PF に関連付けられています。VF は、ネットワークアダプターの仮想化されたインスタンスを表します。
- **スイッチ**: ネットワークスイッチ。ノードは中断なしに接続することもできます。
- **testpmd**: DPDK に含まれるサンプルアプリケーション。**testpmd** アプリケーションを使用して、パケット転送モードで DPDK をテストできます。**testpmd** アプリケーションは、DPDK ソフトウェア開発キット (SDK) を使用して本格的なアプリケーションを構築する方法の例でもあります。
- **worker 0** および **worker 1**: OpenShift Container Platform ノード。

9.6. SR-IOV と NODE TUNING OPERATOR を使用した DPDK ラインレートの実現

Node Tuning Operator を使用して、分離された CPU、ヒュージページ、およびトポロジースケジューラーを設定できます。その後、Node Tuning Operator と Single Root I/O Virtualization (SR-IOV) を使用して、特定の Data Plane Development Kit (DPDK) ラインレートを実現できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- スタンドアロン Node Tuning Operator をデプロイしている。



注記

OpenShift Container Platform の以前のバージョンでは、パフォーマンスアドオン Operator を使用して自動チューニングを実装し、OpenShift アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

手順

1. 次の例に基づいて **PerformanceProfile** オブジェクトを作成します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  globallyDisableIrqLoadBalancing: true
  cpu:
    isolated: 21-51,73-103 (1)
    reserved: 0-20,52-72 (2)
  hugepages:
    defaultHugepagesSize: 1G (3)
    pages:
      - count: 32
        size: 1G
  net:
    userLevelNetworking: true
  numa:
    topologyPolicy: "single-numa-node"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

(1) システムでハイパースレッディングが有効になっている場合は、関連するシンボリックリンクを **isolated** および **reserved** の CPU グループに割り当てます。システムに複数の Non-Uniform Memory Access (NUMA) ノードが含まれている場合は、両方の NUMA から両方のグループに CPU を割り当てます。このタスクには Performance Profile Creator を使用することもできます。詳細は、**コントロールプレーンプロファイルの作成** を参照してください。

(2) キューが予約済みの CPU 数に設定されているデバイスのリストを指定することもできます。詳細は、**Node Tuning Operator を使用した NIC キューの削減**を参照してください。

(3) 必要なヒュージページの数とサイズを割り当てます。ヒュージページの NUMA 設定を指定できます。デフォルトでは、システムは、そのシステムにあるすべての NUMA ノードに偶数分を割り当てます。必要に応じて、ノードのリアルタイムカーネルの使用をリクエストできます。詳細は、**リアルタイム機能を備えたワーカーのプロビジョニング** を参照してください。

2. **yaml** ファイルを **mlx-dpdk-perfprofile-policy.yaml** として保存します。

3. 次のコマンドを使用して、パフォーマンスプロファイルを適用します。

```
$ oc create -f mlx-dpdk-perfprofile-policy.yaml
```

9.6.1. コンテナーアプリケーションで使用する DPDK ライブラリー

オプションライブラリー の **app-netutil** は、その Pod 内で実行されるコンテナーから Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

このライブラリーは、DPDK (Data Plane Development Kit) モードの SR-IOV Virtual Function (VF) のコンテナーへの統合を支援します。このライブラリーは Golang API と C API の両方を提供します。

現時点で 3 つの API メソッドが実装されています。

GetCPUInfo()

この機能は、コンテナーで利用可能な CPU を判別し、リストを返します。

GetHugepages()

この機能は、各コンテナーの **Pod** 仕様で要求される huge page メモリーの量を判別し、値を返します。

GetInterfaces()

この機能は、コンテナーのインターフェイスセットを判別し、インターフェイスタイプとタイプ固有のデータと共にリストを返します。戻り値には、インターフェイスのタイプと、各インターフェイスのタイプ固有のデータが含まれます。

ライブラリーのリポジトリーには、コンテナーイメージ **dpdk-app-centos** をビルドするためのサンプル Dockerfile が含まれます。コンテナーイメージは、Pod 仕様の環境変数に応じて、**l2fwd**、**l3wd** または **testpmd** の DPDK サンプルアプリケーションのいずれかを実行できます。コンテナーイメージは、**app-netutil** ライブラリーをコンテナーイメージ自体に統合する例を提供します。ライブラリーを init コンテナーに統合することもできます。init コンテナーは必要なデータを収集し、データを既存の DPDK ワークロードに渡すことができます。

9.6.2. Virtual Function の SR-IOV Network Operator の例

Single Root I/O Virtualization (SR-IOV) Network Operator を使用して、ノード上の SR-IOV をサポートする Physical Function NIC から Virtual Function (VF) を割り当てて設定できます。

Operator のデプロイの詳細は、**SR-IOV Network Operator のインストール** を参照してください。SR-IOV ネットワークデバイスの設定の詳細は、**SR-IOV ネットワークデバイスの設定** を参照してください。

Intel VF と Mellanox VF での Data Plane Development Kit (DPDK) ワークロードの実行にはいくつかの違いがあります。このセクションでは、両方の VF タイプのオブジェクト設定の例を示します。以下は、Intel NIC で DPDK アプリケーションを実行するために使用される **sriovNetworkNodePolicy** オブジェクトの例です。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci 1
  needVhostNet: true 2
  nicSelector:
    pfNames: ["ens3f0"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

```
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  needVhostNet: true
  nicSelector:
    pfNames: ["ens3f1"]
  nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_2
```

1 Intel NIC の場合、**deviceType** は **vfio-pci** である必要があります。

2 DPDK ワークロードとのカーネル通信が必要な場合は、**needVhostNet: true** を追加します。これにより、**/dev/net/tun** および **/dev/vhost-net** デバイスがコンテナーにマウントされ、アプリケーションがタップデバイスを作成し、タップデバイスを DPDK ワークロードに接続できるようになります。

以下は、Mellanox NIC の **sriovNetworkNodePolicy** オブジェクトの例です。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice (1)
  isRdma: true (2)
  nicSelector:
    rootDevices:
      - "0000:5e:00.1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-2
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
```

```
    rootDevices:
      - "0000:5e:00.0"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_2
```

1 Mellanox デバイスの場合、**deviceType** は **netdevice** である必要があります。

2 Mellanox デバイスの場合、**isRdma** は **true** である必要があります。Mellanox カードは、Flow Bifurcation を使用して DPDK アプリケーションに接続されます。このメカニズムは、Linux ユーザー空間とカーネル空間の間でトラフィックを分割し、ラインレートの処理能力を高めることができます。

9.6.3. SR-IOV Network Operator の例

以下は、**sriovNetwork** オブジェクトの定義例です。この場合、Intel と Mellanox の設定は同じです。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-1
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.1.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}' 1
  networkNamespace: dpdk-test 2
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_1 3
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-2
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.2.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'
  networkNamespace: dpdk-test
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_2
```

1 Whereabouts など、別の IP Address Management (IPAM) 実装を使用できます。詳細は、**Whereabouts を使用した動的 IP アドレス割り当ての設定** を参照してください。

2 ネットワークアタッチメント定義が作成される **networkNamespace** を要求する必要があります。**openshift-sriov-network-operator** namespace で **sriovNetwork** CR を作成する必要があります。

3 **resourceName** の値は、**sriovNetworkNodePolicy** で作成された **resourceName** の値と一致する必要があります。

9.6.4. DPDK ベースワークロードの例

以下は、Data Plane Development Kit (DPDK) コンテナーの例です。

```
apiVersion: v1
kind: Namespace
metadata:
  name: dpdk-test
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[ (1)
     {
      "name": "dpdk-network-1",
      "namespace": "dpdk-test"
     },
     {
      "name": "dpdk-network-2",
      "namespace": "dpdk-test"
     }
   ]'
    irq-load-balancing.crio.io: "disable" (2)
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  labels:
    app: dpdk
  name: testpmd
  namespace: dpdk-test
spec:
  runtimeClassName: performance-performance (3)
  containers:
    - command:
        - /bin/bash
        - -c
        - sleep INF
      image: registry.redhat.io/openshift4/dpdk-base-rhel8
      imagePullPolicy: Always
      name: dpdk
      resources: (4)
        limits:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
        requests:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
      securityContext:
        capabilities:
          add:
            - IPC_LOCK
            - SYS_RESOURCE
            - NET_RAW
```

```
          - NET_ADMIN
      runAsUser: 0
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepages
terminationGracePeriodSeconds: 5
volumes:
  - emptyDir:
      medium: HugePages
    name: hugepages
```

1. 必要な SR-IOV ネットワークをリクエストします。デバイスのリソースは自動挿入されます。
2. CPU と IRQ 負荷分散ベースを無効にします。詳細は、**個々の Pod の割り込み処理の無効化** を参照してください。
3. **runtimeClass** は **performance-performance** に設定します。**runtimeClass** は **HostNetwork** または **privileged** に設定しないでください。
4. サービスの品質 (QoS) が **Guaranteed** さの Pod を開始するには、要求と制限に対して同じ数のリソースを要求します。

注記

SLEEP 状態の Pod を起動し、その Pod で exec 操作を実行して testpmd または DPDK ワークロードを開始しないでください。これにより、**exec** プロセスがどの CPU にも固定されていないため、割り込みが追加される可能性があります。

9.6.5. testpmd スクリプトの例

以下は、**testpmd** を実行するスクリプトの例です。

```
#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -l ${CPU} -a ${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxd=4096 --txd=4096 --
rxq=7 --txq=7 --forward-mode=mac --eth-peer=0,50:00:00:00:00:01 --eth-peer=1,50:00:00:00:00:02
```

この例では、2 つの異なる **sriovNetwork** CR を使用しています。環境変数には、Pod に割り当てられた Virtual Function (VF) PCI アドレスが含まれています。Pod 定義で同じネットワークを使用する場合は、**pciAddress** を分割する必要があります。トラフィックジェネレータの正しい MAC アドレスを設定することが重要です。この例では、カスタム MAC アドレスを使用しています。

9.7. MELLANOX NIC を使用した RDMA モードでの VIRTUAL FUNCTION の使用

重要

RoCE (RDMA over Converged Ethernet) はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

RoCE (RDMA over Converged Ethernet) は、OpenShift Container Platform で RDMA を使用する場合に唯一サポートされているモードです。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

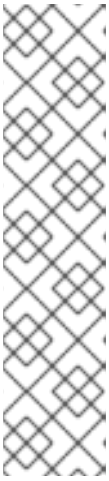
1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **mlx-rdma-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。

(2) Virtual Function のドライバータイプを **netdevice** に指定します。

(3) RDMA モードを有効にします。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**Configuring SR-IOV network devices** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

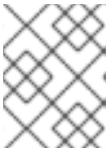
2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **mlx-rdma-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- 1
# ...
  vlan: <vlan>
  resourceName: mlxnics
```

1 IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。



注記

SriovNetwork の各オプションに関する詳細は、「SR-IOV の追加ネットワークの設定」セクションを参照してください。

オプションのライブラリー app-netutil は、コンテナーの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

4. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-rdma-network.yaml
```

5. 以下の **Pod** 仕様を作成してから、YAML を **mlx-rdma-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: rdma-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "4" 5
        hugepages-1Gi: "4Gi" 6
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. **SriovNetwork** オブジェクトの **mlx-rdma-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合は、**target_namespace** を **Pod** 仕様および **SriovNetwork** オブジェクトの両方で変更します。

2. アプリケーションとアプリケーションが使用する RDMA ライブラリーが含まれる RDMA イメージを指定します。

3. hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナー内のアプリケーションに必要な追加機能を指定します。

4. hugepage ボリュームを **/mnt/huge** の下の RDMA Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. CPU の数を指定します。RDMA Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

6. hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、RDMA Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して RDMA Pod を作成します。

```
$ oc create -f mlx-rdma-pod.yaml
```

9.8. OPENSTACK で OVS-DPDK を使用するクラスター用のテスト POD テンプレート

次の **testpmd** Pod では、ヒュージページ、予約済み CPU、および SR-IOV ポートを使用したコンテナーの作成を紹介します。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-dpdk
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/dpdk1: 1 (1)
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
        openshift.io/dpdk1: 1
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  runtimeClassName: performance-cnf-performanceprofile (2)
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1) この例の **dpdk1** という名前は、ユーザーが作成した **SriovNetworkNodePolicy** リソースです。この名前は、作成したリソースの名前に置き換えることができます。

2 パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

9.9. 関連情報

- サポートされるデバイス
- パフォーマンスプロファイルの作成
- パフォーマンスプロファイルによる NIC キューの調整
- リアルタイムおよび低レイテンシーワークロードのプロビジョニング
- SR-IOV Network Operator のインストール
- SR-IOV ネットワークデバイスの設定
- Whereabouts を使用した動的 IP アドレス割り当ての設定
- 個別の Pod の割り込み処理の無効化
- SR-IOV イーサネットネットワーク割り当ての設定

第10章 POD レベルのボンディングの使用

Pod レベルでのボンディングは、高可用性とスループットを必要とする Pod 内のワークロードを有効にするために不可欠です。Pod レベルのボンディングでは、カーネルモードインターフェイスで複数の Single Root I/O Virtualization (SR-IOV) Virtual Function インターフェイスからボンドインターフェイスを作成できます。SR-IOV Virtual Function は Pod に渡され、カーネルドライバーに割り当てられます。

Pod レベルのボンディングが必要なシナリオには、異なる Physical Function 上の複数の SR-IOV Virtual Function からのボンディングインターフェイスの作成が含まれます。ホストの 2 つの異なる Physical Function からボンディングインターフェイスを作成して、Pod レベルで高可用性およびスループットを実現するために使用できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

SR-IOV ネットワーク、ネットワークポリシー、ネットワークアタッチメント定義、Pod の作成などのタスクに関するガイダンスは、SR-IOV ネットワークデバイスの設定 を参照してください。

10.1. 2 つの SR-IOV インターフェイスからのボンドインターフェイスの設定

ボンディングを使用すると、複数のネットワークインターフェイスを 1 つの論理的な "ボンディングされた" インターフェイスに集約できます。Bond Container Network Interface (Bond-CNI) により、コンテナーでボンディング機能を使用できます。

Bond-CNI は、Single Root I/O Virtualization (SR-IOV) Virtual Function を使用して作成し、それらをコンテナーネットワーク namespace に配置できます。

OpenShift Container Platform は、SR-IOV Virtual Functions を使用する Bond-CNI のみをサポートします。SR-IOV Network Operator は、Virtual Function の管理に必要な SR-IOV CNI プラグインを提供します。他の CNI またはインターフェイスのタイプはサポートされていません。

前提条件

- コンテナー内の Virtual Function を取得するために、SR-IOV Network Operator をインストールして設定する。
- SR-IOV インターフェイスを設定するために、インターフェイスごとに SR-IOV ネットワークとポリシーを作成する。
- SR-IOV Network Operator により、定義された SR-IOV ネットワークとポリシーに基づいて、各 SR-IOV インターフェイスのネットワークアタッチメント定義が作成されている。
- SR-IOV Virtual Function に対して、**linkState** がデフォルト値である **auto** に設定されている。

10.1.1. ボンディングのネットワークアタッチメント定義の作成

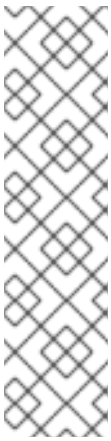
SR-IOV Virtual Function が使用可能になったら、ボンディングのネットワークアタッチメント定義を作成できます。

```
apiVersion: "k8s.cni.cncf.io/v1"
    kind: NetworkAttachmentDefinition
    metadata:
      name: bond-net1
      namespace: demo
```

```
spec:
  config: '{
  "type": "bond", (1)
  "cniVersion": "0.3.1",
  "name": "bond-net1",
  "mode": "active-backup", (2)
  "failOverMac": 1, (3)
  "linksInContainer": true, (4)
  "miimon": "100",
  "mtu": 1500,
  "links": [ (5)
        {"name": "net1"},
        {"name": "net2"}
    ],
  "ipam": {
        "type": "host-local",
        "subnet": "10.56.217.0/24",
        "routes": [{
        "dst": "0.0.0.0/0"
        }],
        "gateway": "10.56.217.1"
    }
  }'
```

(1) cni-type は常に **bond** に設定されます。

(2) **mode** 属性は、ボンドモードを指定します。



注記

サポートされているボンドモードは次のとおりです。

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2

balance-rr または **balance-xor** モードの場合には、SR-IOV Virtual Function の **trust** モードを **on** に設定する必要があります。

(3) **failover** 属性は、active-backup モードでは必須であり、1 に設定する必要があります。

(4) **linksInContainer=true** フラグは、必要なインターフェイスがコンテナー内にあることを Bond CNI に通知します。デフォルトでは、Bond CNI はホスト上でこれらのインターフェイスを検索しますが、これは SRIOV および Multus との統合では機能しません。

(5) **links** セクションでは、ボンディングの作成に使用するインターフェイスを定義します。デフォルトでは、Multus は接続されたインターフェイスに "net" と 1 から始まる連続した番号の名前を付けます。

10.1.2. ボンディングインターフェイスを使用した Pod の作成

1. **podbonding.yaml** などの名前の YAML ファイルに以下の内容を追加して Pod を作成し、この設定をテストします。

```
apiVersion: v1
  kind: Pod
  metadata:
    name: bondpod1
    namespace: demo
    annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1 (1)
  spec:
    containers:
    - name: podexample
      image: quay.io/openshift/origin-network-interface-bond-cni:4.11.0
      command: ["/bin/bash", "-c", "sleep INF"]
```

(1) ネットワークのアノテーションに注意してください。これには、SR-IOV ネットワーク割り当てが 2 つとボンドネットワーク割り当てが 1 つ含まれています。ボンド割り当ては、2 つの SR-IOV インターフェイスをボンドポートインターフェイスとして使用します。

2. 以下のコマンドを実行して yaml を適用します。

```
$ oc apply -f podbonding.yaml
```

3. 次のコマンドを使用して Pod インターフェイスを検査します。

```
$ oc rsh -n demo bondpod1
sh-4.4#
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
3: eth0@if150: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc
noqueue state UP
link/ether 62:b1:b5:c8:fb:7a brd ff:ff:ff:ff:ff:ff
inet 10.244.1.122/24 brd 10.244.1.255 scope global eth0
valid_lft forever preferred_lft forever
4: net3: <BROADCAST,MULTICAST,UP,LOWER_UP400> mtu 1500 qdisc noqueue state
UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (1)
inet 10.56.217.66/24 scope global bond0
valid_lft forever preferred_lft forever
43: net1: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (2)
44: net2: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (3)
```

(1) ボンディングインターフェイスには、自動的に **net3** という名前が付けられます。特定のインターフェイス名を設定するには、Pod の **k8s.v1.cni.cncf.io/networks** アノテーションに **@name** 接尾辞を追加します。

2 **net1** インターフェイスは、SR-IOV Virtual Function に基づいています。

3 **net2** インターフェイスは、SR-IOV Virtual Function に基づいています。

注記

Pod アノテーションでインターフェイス名が設定されていない場合、インターフェイス名は **net<n>** として自動的に割り当てられます (**<n>** は **1** から始まります)。

4. オプション: たとえば **bond0** などの特定のインターフェイス名を設定する場合は、次のように **k8s.v1.cni.cncf.io/networks** アノテーションを編集し、**bond0** をインターフェイス名として設定します。

```
annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0
```

第11章 ハードウェアオフロードの設定

クラスター管理者は、互換性のあるノードでハードウェアオフロードを設定して、データ処理パフォーマンスを向上させ、ホスト CPU の負荷を軽減できます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

11.1. ハードウェアのオフロードについて

Open vSwitch ハードウェアオフロードは、ネットワークタスクを CPU から迂回させ、ネットワークインターフェイスコントローラー上の専用プロセッサーにオフロードすることにより、ネットワークタスクを処理する方法です。その結果、クラスターは、データ転送速度の高速化、CPU ワークロードの削減、およびコンピューティングコストの削減の恩恵を受けることができます。

この機能の重要な要素は、SmartNIC と呼ばれる最新クラスのネットワークインターフェイスコントローラーです。SmartNIC は、計算量の多いネットワーク処理タスクを処理できるネットワークインターフェイスコントローラーです。専用のグラフィックカードがグラフィックパフォーマンスを向上させるのと同じように、SmartNIC はネットワークパフォーマンスを向上させることができます。いずれの場合も、専用プロセッサーにより、特定のタイプの処理タスクのパフォーマンスが向上します。

OpenShift Container Platform では、互換性のある SmartNIC を持つベアメタルノードのハードウェアオフロードを設定できます。ハードウェアオフロードは、SR-IOV Network Operator によって設定および有効化されます。

ハードウェアのオフロードは、すべてのワークロードまたはアプリケーションタイプと互換性があるわけではありません。次の 2 つの通信タイプのみがサポートされています。

- pod-to-pod
- pod-to-service。サービスは通常の Pod に基づく ClusterIP サービスです。

すべての場合において、ハードウェアのオフロードは、それらの Pod とサービスが互換性のある SmartNIC を持つノードに割り当てられている場合にのみ行われます。たとえば、ハードウェアをオフロードしているノードの Pod が、通常のノードのサービスと通信しようとしているとします。通常のノードでは、すべての処理がカーネルで行われるため、Pod からサービスへの通信の全体的なパフォーマンスは、その通常のノードの最大パフォーマンスに制限されます。ハードウェアオフロードは、DPDK アプリケーションと互換性がありません。

ノードでのハードウェアのオフロードを有効にし、使用する Pod を設定しないと、Pod トラフィックのスループットパフォーマンスが低下する可能性があります。OpenShift Container Platform で管理される Pod のハードウェアオフロードを設定することはできません。

11.2. サポートされるデバイス

ハードウェアオフロードは、次のネットワークインターフェイスコントローラーでサポートされています。

表11.1 サポート対象のネットワークインターフェイスコントローラー

製造元	モデル	ベンダー ID	デバイス ID
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017

製造元	モデル	ベンダー ID	デバイス ID
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 ファミリー [ConnectX-6 Lx]	15b3	101f
Mellanox	ConnectX-6 NIC モードの MT42822 BlueField-2	15b3	a2d6

11.3. 前提条件

- クラスターに、ハードウェアのオフロードがサポートされているネットワークインターフェイスコントローラーを備えたベアメタルマシンが少なくとも 1 台ある。
- SR-IOV Network Operator をインストール がインストールされている。
- クラスターが OVN-Kubernetes ネットワークプラグイン を使用している。
- OVN-Kubernetes ネットワークプラグイン設定 で、**gatewayConfig.routingViaHost** フィールドが **false** に設定されている。

11.4. SR-IOV NETWORK OPERATOR の SYSTEMD モードへの設定

ハードウェアオフロードをサポートするには、まず SR-IOV Network Operator を **systemd** モードに設定する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. すべての SR-IOV Operator コンポーネントをデプロイするには、**SriovOperatorConfig** カスタムリソース (CR) を作成します。
   a. 次の YAML を含む **sriovOperatorConfig.yaml** という名前のファイルを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default (1)
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: true
  enableOperatorWebhook: true
  configurationMode: "systemd" (2)
  logLevel: 2
```

1. **SriovOperatorConfig** リソースの有効な名前は **default** のみであり、Operator がデプロイされている namespace 内にある必要があります。
2. SR-IOV Network Operator の **systemd** モードへの設定は、Open vSwitch ハードウェアオフロードのみが対象です。

b. 次のコマンドを実行して、リソースを作成します。

```
$ oc apply -f sriovOperatorConfig.yaml
```

11.5. ハードウェアオフロード用のマシン設定プールの設定

ハードウェアオフロードを有効にするには、専用のマシン設定プールを作成し、SR-IOV Network Operator と連携するように設定する必要があります。

前提条件

- SR-IOV Network Operator がインストールされ、**systemd** モードに設定されている。

手順

1. ハードウェアオフロードを使用するマシンのマシン設定プールを作成します。

a. 次の例のようなコンテンツを含む **mcp-offloading.yaml** などのファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-offloading 1
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,mcp-offloading]} 2
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-offloading: "" 3
```

1 2 ハードウェアオフロード用のマシン設定プールの名前。

3. このノードロールラベルは、マシン設定プールにノードを追加するために使用されます。

b. マシン設定プールの設定を適用します。

```
$ oc create -f mcp-offloading.yaml
```

2. マシン設定プールにノードを追加します。プールのノードロールラベルで各ノードにラベルを付けます。

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3. オプション: 新しいプールが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get nodes
```

出力例

```
NAME       STATUS   ROLES                  AGE   VERSION
master-0   Ready    master                 2d    v1.32.3
master-1   Ready    master                 2d    v1.32.3
worker-0   Ready    worker                 2d    v1.32.3
worker-1   Ready    worker                 2d    v1.32.3
worker-2   Ready    mcp-offloading,worker  47h   v1.32.3
```

4. このマシン設定プールを **SriovNetworkPoolConfig** カスタムリソースに追加します。

   a. 次の例のようなコンテンツを含むファイル (**sriov-pool-config.yaml** など) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

   1 ハードウェアオフロード用のマシン設定プールの名前。

   b. 設定を適用します。

```
$ oc create -f <SriovNetworkPoolConfig_name>.yaml
```

注記

SriovNetworkPoolConfig オブジェクトで指定された設定を適用すると、SR-IOV Operator は、マシン設定プール内のノードをドレインして再起動します。

設定の変更が適用されるまでに数分かかる場合があります。

11.6. SR-IOV ネットワークノードポリシーの設定

SR-IOV ネットワークノードポリシーを作成することにより、ノードの SR-IOV ネットワークデバイス設定を作成できます。ハードウェアオフロードを有効にするには、値 **"switchdev"** を使用して **.spec.eSwitchMode** フィールドを定義する必要があります。

次の手順では、ハードウェアをオフロードするネットワークインターフェイスコントローラー用の SR-IOV インターフェイスを作成します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 次の例のようなコンテンツを含むファイル (**sriov-node-policy.yaml** など) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy 1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 2
  eSwitchMode: "switchdev" 3
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 6
  priority: 5
  resourceName: mlxnics
```

1. カスタムリソースオブジェクトの名前。
2. 必須。ハードウェアのオフロードは、**vfio-pci** ではサポートされていません。
3. 必須。

2. ポリシーの設定を適用します。

```
$ oc create -f sriov-node-policy.yaml
```

注記

SriovNetworkPoolConfig オブジェクトで指定された設定を適用すると、SR-IOV Operator は、マシン設定プール内のノードをドレインして再起動します。

設定の変更が適用されるまでに数分かかる場合があります。

11.6.1. OpenStack の SR-IOV ネットワークノードポリシーの例

次の例は、Red Hat OpenStack Platform (RHOSP) でハードウェアオフロードを使用するネットワークインターフェイスコントローラー (NIC) の SR-IOV インターフェイスを示しています。

RHOSP でのハードウェアオフロードを備えた NIC の SR-IOV インターフェイス

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
```

```
metadata:
  name: ${name}
  namespace: openshift-sriov-network-operator
spec:
  deviceType: switchdev
  isRdma: true
  nicSelector:
    netFilter: openstack/NetworkID:${net_id}
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: ${name}
```

11.7. VIRTUAL FUNCTION を使用したネットワークトラフィックのパフォーマンスの向上

この手順に従って、OVN-Kubernetes 管理ポートに Virtual Function を割り当て、そのネットワークトラフィックパフォーマンスを向上させます。

この手順により 2 つのプールが作成されます。1 つ目には OVN-Kubernetes によって使用される Virtual Function があり、2 つ目は残りの Virtual Function で構成されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 次のコマンドを実行して、SmartNIC が存在する各ワーカーノードに **network.operator.openshift.io/smart-nic** ラベルを追加します。

   ```
   $ oc label node <node-name> network.operator.openshift.io/smart-nic=
   ```

   oc get nodes コマンドを使用して、使用可能なノードのリストを取得します。

2. 次の例のような内容を含む、管理ポート用の **sriov-node-mgmt-vf-policy.yaml** という名前のポリシーを作成します。

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetworkNodePolicy
   metadata:
     name: sriov-node-mgmt-vf-policy
     namespace: openshift-sriov-network-operator
   spec:
     deviceType: netdevice
     eSwitchMode: "switchdev"
     nicSelector:
       deviceID: "1019"
       rootDevices:
       - 0000:d8:00.0
       vendor: "15b3"
   ```

```
    pfNames:
    - ens8f0#0-0 1
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 2
  priority: 5
  resourceName: mgmtvf
```

1 このデバイスは、ユースケースに適したネットワークデバイスに置き換えます。**pfNames** 値の **#0-0** の部分は、OVN-Kubernetes によって使用される単一の Virtual Function を予約します。

2 ここで提供される値は例です。この値は、要件を満たす値に置き換えます。詳細は、**関連情報** セクションの **SR-IOV ネットワークノード設定オブジェクト** を参照してください。

3. 次の例のような内容を含む **sriov-node-policy.yaml** という名前のポリシーを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0#1-5 1
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 2
  priority: 5
  resourceName: mlxnics
```

1 このデバイスは、ユースケースに適したネットワークデバイスに置き換えます。

2 ここで提供される値は例です。この値は **sriov-node-mgmt-vf-policy.yaml** ファイルで指定された値に置き換えます。詳細は、**関連情報** セクションの **SR-IOV ネットワークノード設定オブジェクト** を参照してください。

注記

sriov-node-mgmt-vf-policy.yaml ファイルには、**pfNames** キーと **resourceName** キーの値が **sriov-node-policy.yaml** ファイルとは異なります。

4. 両方のポリシーの設定を適用します。

```
$ oc create -f sriov-node-policy.yaml
```

```
$ oc create -f sriov-node-mgmt-vf-policy.yaml
```

5. 管理設定用にクラスター内に Cluster Network Operator (CNO) ConfigMap を作成します。
   a. 次の内容を含む **hardware-offload-config.yaml** という名前の ConfigMap を作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hardware-offload-config
  namespace: openshift-network-operator
data:
  mgmt-port-resource-name: openshift.io/mgmtvf
```

   b. ConfigMap の設定を適用します。

```
$ oc create -f hardware-offload-config.yaml
```

関連情報

- SR-IOV ネットワークノード設定オブジェクト

11.8. ネットワークアタッチメント定義の作成

マシン設定プールと SR-IOV ネットワークノードポリシーを定義した後、指定したネットワークインターフェイスカードのネットワークアタッチメント定義を作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 次の例のようなコンテンツを含むファイル (**net-attach-def.yaml** など) を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-attach-def 1
  namespace: net-attach-def 2
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnics 3
spec:
  config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":
{},"dns":{}}'
```

1 ネットワークアタッチメント定義の名前。

2 ネットワークアタッチメント定義の namespace。

3

これは、**SriovNetworkNodePolicy** オブジェクトで指定した **spec.resourceName** フィールドの値です。

2. ネットワークアタッチメント定義の設定を適用します。

```
$ oc create -f net-attach-def.yaml
```

検証

- 新しい定義が存在するかどうかを確認するには、次のコマンドを実行します。

```
$ oc get net-attach-def -A
```

出力には、新しい定義の名前空間、名前、および作成からの経過時間が表示されます。

11.9. ネットワークアタッチメント定義への POD の追加

マシン設定プール、**SriovNetworkPoolConfig** および **SriovNetworkNodePolicy** カスタムリソース、およびネットワークアタッチメント定義を作成した後、ネットワークアタッチメント定義を Pod 仕様に追加することで、これらの設定を Pod に適用できます。

手順

- Pod 仕様に **.metadata.annotations.k8s.v1.cni.cncf.io/networks** フィールドを追加し、ハードウェアオフロード用に作成したネットワークアタッチメント定義を指定します。

```
....
metadata:
  annotations:
    v1.multus-cni.io/default-network: net-attach-def/net-attach-def 1
```

1 この値は、ハードウェアオフロード用に作成したネットワークアタッチメント定義の名前と namespace である必要があります。

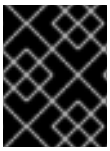
第12章 BLUEFIELD-2 を DPU から NIC に切り替える

Bluefield-2 ネットワークデバイスをデータ処理ユニット (DPU) モードからネットワークインターフェイスコントローラー (NIC) モードに切り替えることができます。

次のドキュメントのタスクを実行する前に、SR-IOV Network Operator がインストールされている ことを確認してください。

12.1. BLUEFIELD-2 を DPU モードから NIC モードに切り替える

以下の手順を使用して、Bluefield-2 をデータ処理ユニット (DPU) モードからネットワークインターフェイスコントローラー (NIC) モードに切り替えます。



重要

現在、DPU から NIC モードへの Bluefield-2 の切り替えのみがサポートされています。NIC モードから DPU モードへの切り替えはサポートされていません。

前提条件

- SR-IOV Network Operator がインストールされている。詳細は、「SR-IOV Network Operator のインストール」を参照してください。
- Bluefield-2 を最新のファームウェアに更新している。詳細は、Firmware for NVIDIA BlueField-2 を参照してください。

手順

1. 次のコマンドを入力して、各コンピュートノードに次のラベルを追加します。コマンドは、各コンピュートノードに対してそれぞれ実行する必要があります。

```
$ oc label node <node_name> node-role.kubernetes.io/sriov=
```

ここでは、以下のようになります。

node_name

コンピュートノードの名前を参照します。

1. SR-IOV Network Operator のマシン設定プールを作成します。次に例を示します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: sriov
spec:
  machineConfigSelector:
    matchExpressions:
    - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,sriov]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/sriov: ""
```

2. 次の **machineconfig.yaml** ファイルをコンピュートノードに適用します。

-

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: sriov
  name: 99-bf2-dpu
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ZmluZF9jb250YWluZXIoKSB7CiAgY3JpY3RsIHBzIC1vIGpzb24gfCBqcSAtciAnLmNv
bnRhaW5lcnNbXSB8IHNlbGVjdCgubWV0YWRhdGEubmFtZT09InNyaW92LW5ldHdvcmstY29
uZmlnLWRhZW1vbiIpIHwgLmlkJwp9CnVudGlsIG91dHB1dD0kKGZpbmRfY29udGFpbmVyKT
sgW1sgLW4gIiRvdXRwdXQiIF1dOyBkbwogIGVjaG8gIndhaXRpbmcgZm9yIGNvbnRhaW5lciB
0byBjb21lIHVwIgogIHNsZWVwIDE7CmRvbmUKISBzdWRvIGNyaWN0bCBleGVjICRvdXRwdX
QgL2JpbmRhdGEvc2NyaXB0cy9iZjItc3dpdGNoLW1vZGUuc2ggIiRAIgo=
        mode: 0755
        overwrite: true
        path: /etc/default/switch_in_sriov_config_daemon.sh
    systemd:
      units:
      - name: dpu-switch.service
        enabled: true
        contents: |
          [Unit]
          Description=Switch BlueField2 card to NIC/DPU mode
          RequiresMountsFor=%t/containers
          Wants=network.target
          After=network-online.target kubelet.service
          [Service]
          SuccessExitStatus=0 120
          RemainAfterExit=True
          ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic || shutdown
-r now' (1)
          Type=oneshot
          [Install]
          WantedBy=multi-user.target
```

(1) オプション: オプションで、特定のカードの PCI アドレスを指定することができます。例えば **ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'**。デフォルトでは、最初のデバイスが選択されています。複数のデバイスがある場合は、使用する PCI アドレスを指定する必要があります。Bluefield-2 を DPU モードから NIC モードに切り替えるすべてのノードで、PCI アドレスが同じである必要があります。

3. コンピュートノードが再起動するまで待ちます。再起動後、コンピュートノードの Bluefield-2 ネットワークデバイスは NIC モードに切り替わります。

4. オプション: 最新の Bluefield-2 ファームウェアリリースでは、NIC モードに切り替えるためにハードウェアの再起動が必要になるため、ホストハードウェアを再起動する必要がある場合があります。

関連情報

- SR-IOV Network Operator のインストール