OpenShift Container Platform 4.19

硬件网络

在 OpenShift Container Platform 中配置特定于硬件的网络功能

Last Updated: 2025-12-12

OpenShift Container Platform 4.19 硬件网络

在 OpenShift Container Platform 中配置特定于硬件的网络功能

Legal Notice

Abstract

本文档论述了在 OpenShift Container Platform 中配置单一根 I/O 虚拟化(SR-IOV)和其他特定于硬件的网络优化。

Table of Contents

第 1 章 关于单根 I/O 虚拟化（SR-IOV）硬件网络

Single Root I/O 虚拟化 (SR-IOV) 规范是针对一类 PCI 设备分配的标准，可与多个 pod 共享单个设备。

您可以使用 SR-IOV Operator 在集群中配置单一根 I/O 虚拟化(SR-IOV)设备。

通过 SR-IOV，您可以将主机节点上识别为物理功能 (PF) 的兼容网络设备分段为多个虚拟功能 (VF)。VF 和其它网络设备一样使用。该设备的 SR-IOV 网络设备驱动程序决定了如何公开容器中的 VF：

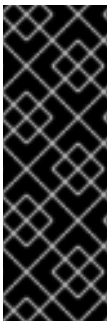
- **netdevice** 驱动程序： 容器 **netns** 中的常规内核网络设备
- **vfio-pci** 驱动程序： 挂载到容器中的字符设备

对于需要高带宽或低延迟的应用程序，您可以在裸机或 Red Hat OpenStack Platform(RHOSP)基础架构上安装 OpenShift Container Platform 集群上的额外网络使用 SR-IOV 网络设备。

您可以为 SR-IOV 网络配置多网络策略。对这个功能的支持是技术预览，SR-IOV 额外网络只支持内核 NIC。它们不支持 Data Plane Development Kit (DPDK) 应用程序。

注意

与 SR-IOV 网络相比，在 SR-IOV 网络中创建多网络策略可能无法为应用程序提供相同的性能。



重要

SR-IOV 网络的多网络策略只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅以下链接：

- 技术预览功能支持范围

您可以使用以下命令在节点上启用 SR-IOV：

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

1.1. 其他资源

- 安装 SR-IOV Network Operator

1.2. 负责管理 SR-IOV 网络设备的组件

SR-IOV Network Operator 会创建和管理 SR-IOV 堆栈的组件。Operator 执行以下功能：

- 编配 SR-IOV 网络设备的发现和管理
- 为 SR-IOV Container Network Interface（CNI）生成 **NetworkAttachmentDefinition** 自定义资源
- 创建和更新 SR-IOV 网络设备插件的配置

- 创建节点特定的 **SriovNetworkNodeState** 自定义资源
- 更新每个 **SriovNetworkNodeState** 自定义资源中的 **spec.interfaces** 字段

Operator 置备以下组件：

SR-IOV 网络配置守护进程

SR-IOV Network Operator 启动时部署在 worker 节点上的守护进程集。守护进程负责在集群中发现和初始化 SR-IOV 网络设备。

SR-IOV Network Operator Webhook

这是动态准入控制器 Webhook，用于验证 Operator 自定义资源，并为未设置的字段设置适当的默认值。

SR-IOV Network Resources Injector（网络资源注入器）

这是一个动态准入控制器 Webhook，它提供通过请求和限制为自定义网络资源（如 SR-IOV VF）应用 Kubernetes pod 规格的功能。SR-IOV 网络资源注入器只会将 **resource** 字段添加到 pod 中的第一个容器。

网络SR-IOV 网络设备插件

这个设备插件用于发现、公告并分配 SR-IOV 网络虚拟功能 (VF) 资源。在 Kubernetes 中使用设备插件能够利用有限的资源，这些资源通常为于物理设备中。设备插件可以使 Kubernetes 调度程序了解资源可用性，因此调度程序可以在具有足够资源的节点上调度 pod。

SR-IOV CNI 插件

SR-IOV CNI 插件会附加从 SR-IOV 网络设备插件中直接分配给 pod 的 VF 接口。

SR-IOV InfiniBand CNI 插件

附加从 SR-IOV 网络设备插件中直接分配给 pod 的 InfiniBand（IB）VF 接口的 CNI 插件。

注意

SR-IOV Network resources injector 和 SR-IOV Operator Webhook 会被默认启用，可通过编辑 **default SriovOperatorConfig** CR 来禁用。禁用 SR-IOV Network Operator Admission Controller Webhook 时要小心。您可以在特定情况下禁用 webhook，如故障排除，或者想要使用不支持的设备。

1.2.1. 支持的平台

在以下平台上支持 SR-IOV Network Operator：

- 裸机
- Red Hat OpenStack Platform (RHOSP)

1.2.2. 支持的设备

OpenShift Container Platform 支持以下网络接口控制器：

表 1.1. 支持的网络接口控制器

制造商	model	供应商 ID	设备 ID
Broadcom	BCM57414	14e4	16d7

制造商	model	供应商 ID	设备 ID
Broadcom	BCM57508	14e4	1750
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572
Intel	X710 Backplane	8086	1581
Intel	X710 基本 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
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 系列 [ConnectX-4]	15b3	1013
Mellanox	MT27710 系列 [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 系列 [ConnectX-5]	15b3	1017

制造商	model	供应商 ID	设备 ID
Mellanox	MT28880 系列 [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 系列 [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	Mellanox MT2910 系列 [ConnectX-7]	15b3	1021
Mellanox	ConnectX-6 NIC 模式中的 MT42822 BlueField-2	15b3	a2d6
Pensando [1]	用于 ionic 驱动程序的 DSC-25 双端口 25G 分布式服务卡	0x1dd8	0x1002
Pensando [1]	用于 ionic 驱动程序的 DSC-100 双端口 100G 分布式服务卡	0x1dd8	0x1003
Silicom	STS 系列	8086	1591

1. 支持 OpenShift SR-IOV，但在使用 SR-IOV 时，您必须使用 SR-IOV CNI 配置文件设置静态的虚拟功能(VF)介质访问控制(MAC)地址。

注意

有关支持的卡和兼容的 OpenShift Container Platform 版本的最新列表，请参阅 Openshift Single Root I/O Virtualization(SR-IOV)和 PTP 硬件网络支持列表。

1.3. 其他资源

- 配置多网络策略

1.4. 后续步骤

- 配置 SR-IOV Network Operator
- 配置 SR-IOV 网络设备
- 如果使用 OpenShift Virtualization将虚拟机连接到 SR-IOV 网络
- 配置 SR-IOV 网络附加
- 以太网网络附加：将 pod 添加到 SR-IOV 额外网络
- InfiniBand 网络附加：将 pod 添加到 SR-IOV 额外网络

第 2 章 配置 SR-IOV 网络设备

您可以在集群中配置单一根 I/O 虚拟化（SR-IOV）设备。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

2.1. SR-IOV 网络节点配置对象

您可以通过创建 SR-IOV 网络节点策略来为节点指定 SR-IOV 网络设备配置。策略的 API 对象是 **sriovnetwork.openshift.io** API 组的一部分。

以下 YAML 描述了 SR-IOV 网络节点策略：

```yaml
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. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。

   在指定名称时，请确保在 **resourceName** 中使用接受的语法表达式 **^[a-zA-Z0-9_]+$**。
4. 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。

重要

SR-IOV Network Operator 按顺序将节点网络配置策略应用到节点。在应用节点网络配置策略前，SR-IOV Network Operator 会检查节点的机器配置池(MCP)是否处于不健康状态，如 **Degraded** 或 **Updating**。如果节点处于不健康的 MCP，将节点网络配置策略应用到集群中的所有目标节点的过程会被暂停，直到 MCP 返回健康状态。

为了避免处于不健康的 MCP 的节点阻止将节点网络配置策略应用到其他节点，包括处于其他 MCP 的节点，您必须为每个 MCP 创建单独的节点网络配置策略。

5 可选： priority 是一个 **0** 到 **99** 之间的整数。较小的值具有更高的优先级。例如，优先级 **10** 是高于优先级 **99**。默认值为 **99**。

6 可选：物理功能及其所有虚拟功能的最大传输单元(MTU)。最大 MTU 值可能因不同的网络接口控制器（NIC）型号而有所不同。

重要

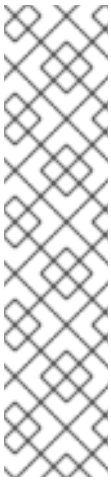
如果要在默认网络接口上创建虚拟功能，请确保将 MTU 设置为与集群 MTU 匹配的值。

如果要修改单个虚拟功能的 MTU，同时将功能分配给 pod，请将 MTU 值留在 SR-IOV 网络节点策略中。否则，SR-IOV Network Operator 会将虚拟功能的 MTU 恢复到 SR-IOV 网络节点策略中定义的 MTU 值，这可能会触发节点排空。

7 可选：将 **needVhostNet** 设置为 **true**，以在 pod 中挂载 **/dev/vhost-net** 设备。使用挂载的 **/dev/vhost-net** 设备及 Data Plane Development Kit (DPDK) 将流量转发到内核网络堆栈。

8 为 SR-IOV 物理网络设备创建的虚拟功能(（VF）的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 **127**。

9 **externallyManaged** 字段指示 SR-IOV Network Operator 是否管理所有，或只是虚拟功能(VF)的子集。将值设为 **false** 时，SR-IOV Network Operator 管理并配置 PF 上的所有 VF。



注意

当 **externalManaged** 设置为 **true** 时，您必须在应用 **SriovNetworkNodePolicy** 资源前在物理功能(PF)上手动创建虚拟功能(VF)。如果没有预先创建 VF，SR-IOV Network Operator 的 Webhook 将阻止策略请求。

当 **externallyManaged** 设为 **false** 时，SR-IOV Network Operator 会自动创建和管理 VF，包括重置 VF （如果需要）。

要在主机系统上使用 VF，您必须通过 NMState 创建它们，并将 **externallyManaged** 设置为 **true**。在这个模式中，SR-IOV Network Operator 不会修改 PF 或手动管理的 VF，策略 **nicSelector** 字段中明确定义的 VF 除外。但是，SR-IOV Network Operator 继续管理用作 pod 二级接口的 VF。

10 NIC 选择器标识此资源应用到的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。

如果指定了**rootDevices**，则必须同时为 **vendor**、 **deviceID** 或 **pfNames** 指定一个值。如果同时指定了 **pfNames** 和 **rootDevices**，请确保它们引用同一设备。如果您为 **netFilter** 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。

11 可选： SR-IOV 网络设备厂商的十六进制厂商代码。允许的值只能是 **8086** (Intel)和 **15b3** (Mellanox)。

12 可选： SR-IOV 网络设备的十六进制设备标识符。例如，**101b** 是 Mellanox ConnectX-6 设备的设备 ID。

13 可选：资源需要应用到的一个或多个物理功能(PF)名称的数组。

14 可选：资源需要应用到的一个或多个 PCI 总线地址的数组。例如 **0000:02:00.1**。

15 可选：特定平台的网络过滤器。唯一支持的平台是 Red Hat OpenStack Platform（RHOSP）。可接受的值具有以下格式： **openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**。将 **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** 替换为来自 **/var/config/openstack/latest/network_data.json** 元数据文件的值。此过滤器确保 VF 与特定的 OpenStack 网络关联。Operator 使用此过滤器根据 OpenStack 平台提供的元数据将 VF 映射到适当的网络。

16 可选：为从此资源创建的 VF 配置的驱动程序。允许的值只能是 **netdevice** 和 **vfio-pci**。默认值为 **netdevice**。

对于裸机节点上的 DPDK 模式的 Mellanox NIC，请使用 **netdevice** 驱动程序类型，并将 **isRdma** 设置为 **true**。

17 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 **false**。

如果 **isRdma** 参数设为 **true**，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。

将 **isRdma** 设置为 **true**，并将 **needVhostNet** 设置为 **true** 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。

注意

对于 intel NIC，您无法将 **isRdma** 参数设置为 **true**。

18 可选：VF 的链接类型。默认值为 **eth** （以太网）。在 InfiniBand 中将这个值改为 'ib'。

当将 **linkType** 设置为 **ib** 时，SR-IOV Network Operator Webhook 会自动将 **isRdma** 设置为 **true**。当将 **linkType** 设定为 **ib** 时，**deviceType** 不应该被设置为 **vfio-pci**。

不要为 SriovNetworkNodePolicy 将 linkType 设置为 **eth**，因为这可能会导致设备插件报告的可用设备数量不正确。

19 可选： 要启用硬件卸载，您必须将 **eSwitchMode** 字段设置为 **"switchdev"**。有关硬件卸载的更多信息，请参阅"配置硬件卸载"。

20 可选：要排除将一个 SR-IOV 网络资源的 NUMA 节点广告到拓扑管理器，将值设为 **true**。默认值为 **false**。

2.1.1. SR-IOV 网络节点配置示例

以下示例描述了 InfiniBand 设备的配置：

InfiniBand 设备的配置示例

```
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
# ...
```

以下示例描述了 RHOSP 虚拟机中的 SR-IOV 网络设备配置：

虚拟机中的 SR-IOV 设备配置示例

```
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. 为虚拟机配置节点网络策略时，**numVfs** 参数始终设置为 **1**。
2. 当虚拟机部署在 RHOSP 上时，**netFilter** 参数必须引用一个网络 ID。**netFilter** 的有效值来自 **SriovNetworkNodeState** 对象。

2.1.2. 自动发现 SR-IOV 网络设备

SR-IOV Network Operator 将搜索集群以获取 worker 节点上的 SR-IOV 功能网络设备。Operator 会为每个提供兼容 SR-IOV 网络设备的 worker 节点创建并更新一个 SriovNetworkNodeState 自定义资源 (CR)。

为 CR 分配了与 worker 节点相同的名称。**status.interfaces** 列表提供有关节点上网络设备的信息。

重要

不要修改 **SriovNetworkNodeState** 对象。Operator 会自动创建和管理这些资源。

2.1.2.1. SriovNetworkNodeState 对象示例

以下 YAML 是由 SR-IOV Network Operator 创建的 **SriovNetworkNodeState** 对象的示例：

一个 SriovNetworkNodeState 对象

```
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** 字段的值与 worker 节点的名称相同。
2. **interfaces** 小节包括 Operator 在 worker 节点上发现的所有 SR-IOV 设备列表。

2.1.3. 启用安全引导时，在 Mellanox 卡中配置 SR-IOV Network Operator

SR-IOV Network Operator 支持一个选项来跳过 Mellanox 设备的固件配置。这个选项允许您在系统启用了安全引导时使用 SR-IOV Network Operator 创建虚拟功能。在切换系统以安全引导前，您必须手动配置并分配固件中的虚拟功能数量。

注意

固件中的虚拟功能数量是策略中您可以请求的最大虚拟功能数。

流程

1. 当使用 sriov-config 守护进程时，当系统没有安全引导时，配置虚拟功能(VF)：

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

1. **SRIOV_EN** 环境变量在 Mellanox 卡中启用 SR-IOV Network Operator 支持。
2. **NUM_OF_VFS** 环境变量指定要在固件中启用的虚拟功能数。

2. 通过禁用 Mellanox 插件来配置 SR-IOV Network Operator。请参阅以下 **SriovOperatorConfig** 示例配置：

```
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. 重启系统以启用虚拟功能和配置设置。
4. 运行以下命令在重启系统后检查虚拟功能(VF)：

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

输出示例

```
- 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** 值与流程前面在 **mstconfig** 命令中使用的数字相同。

5. 启用安全引导以防止未经授权的操作系统和恶意软件在设备的引导过程中加载。
   a. 使用 BIOS （基本输入/输出系统）启用安全引导，以便为以下参数设置值：
      - **Secure Boot: Enabled**
      - **Secure Boot Policy: Standard**
      - **Secure Boot Mode: Mode Deployed**
   b. 重启系统。

2.1.4. SR-IOV 设备的虚拟功能 (VF) 分区

在某些情况下，您可能想要将同一个物理功能 (PF) 的虚拟功能 (VF) 分成多个资源池。例如：您可能想要某些 VF 使用默认驱动程序载入，而其他的 VF 负载使用 **vfio-pci** 驱动程序。

例如，以下 YAML 显示名为 **netpf0** 的、带有 VF **2** 到 **7** 的接口的选择器：

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

其中：

netpf0
PF 接口名称的名称。

2
包含在范围内的第一个 VF 索引（基于 0）。

7
包含在范围内的最后一个 VF 索引（基于 0）。

您可以使用满足以下要求的不同策略 CR 从同一 PF 中选择 VF：

- 对于选择相同 PF 的策略，**numVfs** 值必须是类似的。

- VF 索引范围是从 **0** 到 **<numVfs>-1** 之间。例如，如果您有一个策略，它的 **numVfs** 被设置为 **8**，则 **<first_vf>** 的值不能小于 **0**，**<last_vf>** 的值不能大于 **7**。
- 不同策略中的 VF 范围不得互相重叠。
- **<first_vf>** 不能大于 **<last_vf>**。

以下示例演示了 SR-IOV 设备的 NIC 分区。

策略 **policy-net-1** 定义了一个资源池 **net-1**，其中包含带有默认 VF 驱动的 PF **netpf0** 的 VF **0**。策略 **policy-net-1-dpdk** 定义了一个资源池 **net-1-dpdk**，其中包含带有 **vfio** VF 驱动程序的 PF **netpf0** 的 VF **8** 到 **15**。

策略 **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
```

策略 **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
```

验证接口是否已成功分区

- 运行以下命令，确认 SR-IOV 设备的接口分区到虚拟功能(VF)。

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

1 将 **<interface>** 替换为您在分区为 SR-IOV 设备的 VF 时指定的接口，如 **ens3f1**。

输出示例

```
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. 在 OpenStack 上使用 SR-IOV 的集群测试 pod 模板

以下 **testpmd** pod 演示了使用巨页、保留 CPU 和 SR-IOV 端口创建容器。

testpmd pod 示例

```
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) 本例假定性能配置集的名称为 **cnf-performance profile**。

2.1.6. 在 OpenStack 上使用 OVS 硬件卸载的集群测试 pod 模板

以下 **testpmd** pod 在 Red Hat OpenStack Platform (RHOSP) 上演示了 Open vSwitch (OVS) 硬件卸载。

testpmd pod 示例

```
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) 如果您的性能配置集没有命名为 **cnf-performance profile**，请将该字符串替换为正确的性能配置集名称。

2.1.7. Downward API 的巨页资源注入

当 pod 规格包含巨页的资源请求或限制时，Network Resources Injector 会自动在 pod 规格中添加 Downward API 字段，以便为容器提供巨页信息。

Network Resources Injector 添加一个名为 **podnetinfo** 的卷，并挂载到 pod 中的每个容器的 **/etc/podnetinfo**。卷使用 Downward API，并包含一个用于大页面请求和限制的文件。文件命名规则如下：

- **/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>**

上一个列表中指定的路径与 **app-netutil** 库兼容。默认情况下，该库配置为搜索 **/etc/podnetinfo** 目录中的资源信息。如果您选择自己手动指定 Downward API 路径项目，**app-netutil** 库除上一个列表中的路径外还会搜索以下路径。

- **/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**

与 Network Resources Injector 可以创建的路径一样，以上列表中的路径可以选择以一个 **_<container-name>** 后缀结尾。

2.2. 配置 SR-IOV 网络设备

SR-IOV Network Operator 把 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CRD 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。

注意

在应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。仅在以下情况下重启：

- 使用 Mellanox NIC (**mlx5** 驱动程序) 时，每次当在一个物理功能（PF）中的虚拟功能 (VF) 数量增加时，节点都会重启。
- 使用 Intel NIC 时，只有在内核参数不包含 **intel_iommu=on** 和 **iommu=pt** 时，才会重启。

它可能需要几分钟时间来应用配置更改。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。
- 集群中有足够的可用节点，用于处理从排空节点中驱除的工作负载。
- 您还没有为 SR-IOV 网络设备配置选择任何 control plane 节点。

流程

1. 创建一个 **SriovNetworkNodePolicy** 对象，然后在 **<name>-sriov-node-network.yaml** 文件中保存 YAML。使用配置的实际名称替换 **<name>**。
2. 可选：将 SR-IOV 功能的集群节点标记为 **SriovNetworkNodePolicy.Spec.NodeSelector**（如果它们还没有标记）。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。
3. 创建 **SriovNetworkNodePolicy** 对象：

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

   其中 **<name>** 指定这个配置的名称。

   在应用配置更新后，**sriov-network-operator** 命名空间中的所有 Pod 都会变为 **Running** 状态。

4. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 **<node_name>** 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

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

其他资源

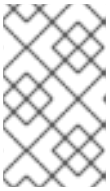
- 了解如何更新节点上的标签。

2.3. 创建与 SR-IOV POD 兼容的非统一内存访问 (NUMA)

您可以通过限制 SR-IOV 和从相同 NUMA 节点分配的 CPU 资源，使用 **restricted** 或 **single-numa-node** Topology Manager 来创建与 SR-IOV pod 兼容的 NUMA。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已将 CPU Manager 策略配置为 **static**。有关 CPU Manager 的详情请参考 "Additional resources" 部分。
- 您已将 Topology Manager 策略配置为 **single-numa-node**。



注意

当 **single-numa-node** 无法满足请求时，您可以将拓扑管理器策略配置为 **restricted**。有关更灵活的 SR-IOV 网络资源调度，请参阅 *附加资源*部分中的 *NUMA 感知调度过程中排除 SR-IOV 网络拓扑*。

流程

1. 创建以下 SR-IOV pod 规格，然后在 **<name>-sriov-pod.yaml** 文件中保存 YAML。将 **<name>** 替换为这个 pod 的名称。
   以下示例显示了 SR-IOV pod 规格：

```
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. 将 **<name>** 替换为 SR-IOV 网络附加定义 CR 的名称。
2. 将 **<image>** 替换为 **sample-pod** 镜像的名称。
3. 要创建带有保证 QoS 的 SR-IOV pod，将 **memory limits** 设置为与 **memory requests** 相同的值。
4. 要创建带有保证 QoS 的 SR-IOV pod，将 **cpu limits** 设置为与 **cpu requests** 相同。

2. 运行以下命令来创建 SR-IOV pod 示例：

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

1. 将 **<filename>** 替换为您在上一步中创建的文件的名称。

3. 确认 **sample-pod** 配置为带有保证 QoS。

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

4. 确认 **sample-pod** 被分配了独有的 CPU。

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

5. 确认为 **sample-pod** 分配的 SR-IOV 设备和 CPU 位于相同的 NUMA 节点上。

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

2.4. 为 NUMA 感知调度排除 SR-IOV 网络拓扑

在 NUMA 感知 pod 调度过程中，可以排除将 SR-IOV 网络的 Non-Uniform Memory Access (NUMA) 节点广告到拓扑管理器，以便实现更灵活的 SR-IOV 网络部署。

在某些情况下，为在单个 NUMA 节点上的一个 pod 最大化 CPU 和内存资源是一个优先操作。如果没有为 Topology Manager 提供有关 pod 的 SR-IOV 网络资源的 NUMA 节点的提示，拓扑管理器可能会将 SR-IOV 网络资源和 pod CPU 和内存资源部署到不同的 NUMA 节点。这可能会添加到网络延迟，因为需要在不同 NUMA 节点之间进行数据传输。但是，当工作负载需要最佳 CPU 和内存性能时，这是可以接受的。

例如，有一个计算节点 **compute-1**，它有两个 NUMA 节点：**numa0** 和 **numa1**。启用了 SR-IOV 的 NIC 存在于 **numa0** 上。可用于 pod 调度的 CPU 仅存在于 **numa1** 上。通过将 **excludeTopology** 规格设置为 **true**，拓扑管理器可将 pod 的 CPU 和内存资源分配给 **numa1**，并可将同一 pod 的 SR-IOV 网络资源分配给 **numa0**。只有将 **excludeTopology** 规格设置为 **true** 时，才能实现。否则，拓扑管理器会尝试将所有资源放在同一 NUMA 节点上。

2.5. SR-IOV 配置故障排除

在进行了配置 SR-IOV 网络设备的步骤后，以下部分会处理一些错误条件。

流程

- 要显示节点状态，请运行以下命令：

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

其中：

<node_name>
: 指定具有 SR-IOV 网络设备的节点名称。

如果命令的输出显示"cannot allocate memory"，请检查以下项目：

- 确认在 BIOS 中为节点启用了全局 SR-IOV 设置。
- 确认在 BIOS 中为该节点启用了 VT-d。

其他资源

- 使用 CPU Manager

2.6. 后续步骤

- 配置 SR-IOV 网络附加

第 3 章 配置 SR-IOV 以太网网络附加

您可以为集群中的单根 I/O 虚拟化（SR-IOV）设备配置以太网网络附加。

在执行以下文档中的任何任务前，请确保安装了 SR-IOV Network Operator。

3.1. 以太网设备配置对象

您可以通过定义 **SriovNetwork** 对象来配置以太网网络设备。

以下 YAML 描述了 **SriovNetwork** 对象：

```
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. 对象的名称。SR-IOV Network Operator 创建一个名称相同的 **NetworkAttachmentDefinition** 对象。
2. 安装 SR-IOV Network Operator 的命名空间。
3. 用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象中的 **spec.resourceName** 参数的值。
4. **SriovNetwork** 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。
5. 可选：指定要分配给额外网络的 VLAN ID。默认值为 **0**，代表额外网络没有 VLAN ID 标签。支持的 VLAN ID 值范围为 **1** 到 **4094**。
6. 可选：VF 的 spoof 检查模式。允许的值是字符串 **"on"** 和 **"off"**。

重要

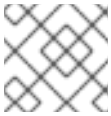
指定的值必须由引号包括，否则 SR-IOV Network Operator 将拒绝对象。

7. 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

8 可选：虚拟功能（VF）的链接状态。允许的值是 **enable**、**disable** 和 **auto**。

9 可选：VF 的最大传输率（以 Mbps 为单位）。

10 可选：VF 的最低传输率（以 Mbps 为单位）。这个值必须小于或等于最大传输率。



注意

Intel NIC 不支持 **minTxRate** 参数。如需更多信息，请参阅 BZ#1772847。

11 可选：VF 的 IEEE 802.1p 优先级级别。默认值为 **0**。

12 可选：VF 的信任模式。允许的值是字符串 **"on"** 和 **"off"**。

重要

您必须在引号中包含指定的值，或者 SR-IOV Network Operator 拒绝对象。

13 可选：为这个额外网络配置功能。您可以指定 **'{ "ips": true }'** 来启用 IP 地址支持，或指定 **'{ "mac": true }'** 来启用 MAC 地址支持。

3.1.1. 为动态分配双栈 IP 地址创建配置

您可以动态地将双栈 IP 地址分配给二级网络，以便 pod 可以通过 IPv4 和 IPv6 地址进行通信。

您可以在 **ipRanges** 参数中配置以下 IP 地址分配类型：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 **type** 设置为 **whereabouts**。
2. 使用 **ipRanges** 来分配 IP 地址，如下例所示：

```
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. 将二级网络附加到 pod。如需更多信息，请参阅"将 pod 添加到二级网络"。

验证

- 输入以下命令验证所有 IP 地址是否已分配给 pod 命名空间内的网络接口：

```
$ oc exec -it <pod_name> -- ip a
```

其中：

<podname>

pod 的名称。

3.1.2. 配置网络附加的 IP 地址分配

对于二级网络，可以使用 IP 地址管理 (IPAM) CNI 插件来分配 IP 地址，该插件支持多种分配方法，包括动态主机配置协议 (DHCP) 和静态分配。

负责动态分配 IP 地址的 DHCP IPAM CNI 插件与两个不同的组件一起运行：

- CNI 插件：负责与 Kubernetes 网络堆栈集成，以请求和释放 IP 地址。
- DHCP IPAM CNI 守护进程：用于 DHCP 事件的监听程序，该事件与环境中的现有 DHCP 服务器协调，以处理 IP 地址分配请求。这个守护进程本身并不是一个 DHCP 服务器。

对于在其 IPAM 配置中需要 **type: dhcp** 的网络，请确保 DHCP 服务器满足以下条件：

- DHCP 服务器可用并在环境中运行。
- DHCP 服务器位于集群外部，您希望该服务器是组成客户的现有网络基础架构的一部分。
- DHCP 服务器被正确配置为为节点提供 IP 地址。

如果在环境中 DHCP 服务器不可用，请考虑使用 Whereabouts IPAM CNI 插件。Whereabouts CNI 提供类似的 IP 地址管理功能，而无需外部 DHCP 服务器。

注意

当没有外部 DHCP 服务器或首选静态 IP 地址管理时，请使用 Whereabouts CNI 插件。Whereabouts 插件包含一个协调器守护进程来管理过时的 IP 地址分配。

通过包含单独的守护进程(DHCP IPAM CNI 守护进程)来确保在容器生命周期内定期续订 DHCP 租期。要部署 DHCP IPAM CNI 守护进程，请更改 Cluster Network Operator (CNO) 配置，以触发此守护进程的部署，作为二级网络设置的一部分。

3.1.2.1. 静态 IP 地址分配配置

下表描述了静态 IP 地址分配的配置：

表 3.1. ipam 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。
Routes	**数组**	指定要在 pod 中配置的路由的一组对象。
dns	**数组**	可选：指定 DNS 配置的对象数组。

address 数组需要带有以下字段的对象：

表 3.2. ipam.addresses[] array

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如：如果您指定 **10.10.21.10/24**，那么会为二级网络分配 IP 地址 **10.10.21.10**，网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 3.3. ipam.routes[] array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	路由网络流量的网关。

表 3.4. ipam.dns object

字段	类型	描述
nameservers	**数组**	发送 DNS 查询的一个或多个 IP 地址的数组。
domain	**数组**	要附加到主机名的默认域。例如，如果将域设置为 **example.com**，对 **example-host** 的 DNS 查找查询将被改写为 **example-host.example.com**。

字段	类型	描述
search	**数组**	在 DNS 查找查询过程中，附加到非限定主机名（如 **example-host**）的域名的数组。

静态 IP 地址分配配置示例

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

3.1.2.2. 动态 IP 地址(DHCP)分配配置

pod 在创建时获取其原始 DHCP 租期。该租期必须由集群中运行的一个小型的 DHCP 服务器部署定期续订。

重要

对于以太网网络附加，SR-IOV Network Operator 不会创建 DHCP 服务器部署。Cluster Network Operator 负责创建最小 DHCP 服务器部署。

要触发 DHCP 服务器的部署，您必须编辑 Cluster Network Operator 配置来创建 shim 网络附加，如下例所示：

shim 网络附加定义示例

```
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"
```

```
      }
    }
  # ...
```

其中：

type

为集群指定动态 IP 地址分配。

3.1.2.2.1. 使用 Whereabouts 进行动态 IP 地址分配配置

Whereabouts CNI 插件允许在不使用 DHCP 服务器的情况下动态地将 IP 地址分配给二级网络。

Whereabouts CNI 插件还支持在单独的 **NetworkAttachmentDefinition** CRD 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

3.1.2.2.1.1. 动态 IP 地址配置参数

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 3.5. ipam whereabouts 配置参数

字段	类型	描述
type	**string**	IPAM 地址类型。需要 **abouts** 的值。
range	**string**	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	**数组**	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	**string**	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

3.1.2.2.1.2. 使用 Whereabouts 排除 IP 地址范围的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的 NAD 文件中的动态地址分配配置：

Whereabouts 动态 IP 地址分配排除特定的 IP 地址范围

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

3.1.2.2.1.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

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

其中：

network_name

可选参数。如果设置，必须与 **NetworkAttachmentDefinition 2** 的 **network_name** 匹配。

NetworkAttachmentDefinition 2

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

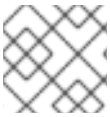
其中：

network_name

可选参数。如果设置，必须与 **NetworkAttachmentDefinition 1** 的 **network_name** 匹配。

3.1.2.3. 配置 SR-IOV 额外网络

您可以通过创建一个 **SriovNetwork** 对象来配置使用 SR-IOV 硬件的额外网络。创建 **SriovNetwork** 对象时，SR-IOV Network Operator 会自动创建一个 **NetworkAttachmentDefinition** 对象。



注意

如果一个 **SriovNetwork** 对象已被附加到状态为 **running** 的 pod，则不要修改或删除它。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建一个 **SriovNetwork** 对象，然后在 **<name>.yaml** 文件中保存 YAML，其中 **<name>** 是这个额外网络的名称。对象规格可能类似以下示例：

```
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. 运行以下命令来创建对象：

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

这里的 **<name>** 指定额外网络的名称。

3. 可选：要确认与您在上一步中创建的 **SriovNetwork** 对象关联的 **NetworkAttachmentDefinition** 对象是否存在，请输入以下命令。将 **<namespace>** 替换为您在 **SriovNetwork** 对象中指定的 networkNamespace。

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

3.1.2.4. 将 SR-IOV 网络分配给 VRF

作为集群管理员，您可以使用 CNI VRF 插件为 VRF 域分配 SR-IOV 网络接口。

要做到这一点，将 VRF 配置添加到 **SriovNetwork** 资源的可选 **metaPlugins** 参数中。

注意

使用 VRF 的应用程序需要绑定到特定设备。通常的用法是在套接字中使用 **SO_BINDTODEVICE** 选项。**SO_BINDTODEVICE** 将套接字绑定到在传递接口名称中指定的设备，例如 **eth1**。要使用 **SO_BINDTODEVICE**，应用程序必须具有 **CAP_NET_RAW** 功能。

OpenShift Container Platform pod 不支持通过 **ip vrf exec** 命令使用 VRF。要使用 VRF，将应用程序直接绑定到 VRF 接口。

3.1.2.4.1. 使用 CNI VRF 插件创建额外的 SR-IOV 网络附加

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 **NetworkAttachmentDefinition** 自定义资源（CR）。

注意

不要编辑 SR-IOV Network Operator 所管理的 **NetworkAttachmentDefinition** 自定义资源。这样做可能会破坏额外网络上的网络流量。

要使用 CNI 虚拟路由和转发(VRF)插件创建额外的 SR-IOV 网络附加，请执行以下步骤。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为额外 SR-IOV 网络附加创建 **SriovNetwork** 自定义资源 (CR) 并插入 **metaPlugins** 配置，如下例所示。将 YAML 保存为文件 **sriov-network-attachment.yaml**。

```
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"
    }
```

其中：

metaPlugins.type

将 **type** 参数设置为 **vrf**。

metaPlugins.vrfname

在 **vrfname** 参数中为 VRF 指定名称。为 VRF 分配接口。如果您没有在 pod 中为 VRF 指定名称，SR-IOV Network Operator 会自动为 VRF 生成一个名称。

2. 创建 **SriovNetwork** 资源：

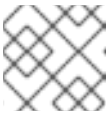
```
$ oc create -f sriov-network-attachment.yaml
```

验证

1. 运行以下命令，确认 SR-IOV Network Operator 创建了 **NetworkAttachmentDefinition** CR。预期输出显示 NAD CR 的名称和创建年龄（以分钟为单位）。

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

- **<namespace>**：将 **<namespace>** 替换为您在配置网络附加时指定的命名空间，如 **additional-sriov-network-1**。



注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

2. 要验证 VRF CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

   a. 创建使用 VRF CNI 的 SR-IOV 网络。

   b. 将网络分配给 pod。

   c. 验证 pod 网络附加是否已连接到 SR-IOV 额外网络。确保您远程登录 pod 并运行以下命令：预期输出显示 VRF 接口的名称及其在路由表中的唯一 ID。

```
$ ip vrf show
```

   d. 运行以下命令确认 VRF 接口是二级接口的 **master** 接口：输出示例显示 **5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red state UP mode**。

```
$ ip link
```

3.1.2.5. 基于以太网的 SR-IOV 附加的运行时配置

将 pod 附加到额外网络时，您可以指定运行时配置来为 pod 进行特定的自定义。例如,,您可以请求特定的 MAC 硬件地址。

您可以通过在 pod 规格中设置注解来指定运行时配置。注解键是 **k8s.v1.cni.cncf.io/network**,它接受一个 JSON 对象来描述运行时配置。

基于以太网的 SR-IOV 网络附加的运行时配置示例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network_attachment>",
          "mac": "<mac_address>",
```

```
          "ips": ["<cidr_range>"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

其中：

k8s.v1.cni.cncf.io/networks.name

SR-IOV 网络附加定义 CR 的名称。示例值为 **ibl**。

k8s.v1.cni.cncf.io/networks.mac

可选参数。从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 **SriovNetwork** 对象中指定 **{ "mac": true }**。示例值为 **c2:11:22:33:44:55:66:77**。

k8s.v1.cni.cncf.io/networks.ips

可选参数。从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 **SriovNetwork** 对象中指定 **{ "ips": true }**。示例值为 **192.168.10.1/24", "2001::1/64**。

3.1.2.6. 将 pod 添加到二级网络

您可以将 pod 添加到二级网络。pod 继续通过默认网络发送与集群相关的普通网络流量。

创建 pod 时，二级网络会附加到 pod。但是，如果 pod 已存在，则无法将二级网络附加到其中。

pod 必须与二级网络位于同一个命名空间中。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 登录到集群。

流程

1. 为 **Pod** 对象添加注解。只能使用以下注解格式之一：

   a. 要在没有自定义的情况下附加二级网络，请使用以下格式添加注解：

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

   其中：

   k8s.v1.cni.cncf.io/networks

   指定要与 pod 关联的二级网络的名称。要指定多个二级网络，请使用逗号分隔每个网络。逗号之间不可包括空格。如果您多次指定相同的二级网络，则该 pod 会将多个网络接口附加到该网络。

b. 要通过自定义来附加二级网络，请添加具有以下格式的注解：

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

其中：

name

指定 **NetworkAttachmentDefinition** 对象定义的二级网络的名称。

namespace

指定定义 **NetworkAttachmentDefinition** 对象的命名空间。

default-route

可选参数。为默认路由指定覆盖，如 **192.168.17.1**。

2. 运行以下命令来创建 pod。

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

将 **<name>** 替换为 pod 的名称。

3. 可选：输入以下命令确认 **pod** CR 中是否存在注解。将 **<name>** 替换为 pod 的名称。

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

在以下示例中，**example-pod** pod 附加到 **net1** 二级网络：

```
$ 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: |-
      [{
          "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:
  ...
```

其中：

k8s.v1.cni.cncf.io/network-status

指定对象的 JSON 数组。每个对象描述附加到 pod 的二级网络的状态。注解值保存为纯文本值。

3.1.2.6.1. 向 pod 公开 vfio-pci SR-IOV 设备的 MTU

将 pod 添加到额外网络后，您可以检查 MTU 可供 SR-IOV 网络使用。

流程

1. 运行以下命令，检查 pod 注解是否包含 MTU：

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

以下示例显示了输出示例：

```
"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. 运行以下命令，验证 pod 中的 **/etc/podnetinfo/** 中是否有 MTU：

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

以下示例显示了输出示例：

```
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\"
    }
}
}]"
```

3.1.2.7. 在 SR-IOV 网络策略更新过程中配置并行节点排空

默认情况下，SR-IOV Network Operator 会在每次策略更改前从节点排空工作负载。Operator 执行此操作（一次一个节点），以确保重新配置不会影响工作负载。

在大型集群中，按顺序排空节点可能非常耗时，需要几小时甚至几天。在时间敏感的环境中，您可以在 **SriovNetworkPoolConfig** 自定义资源 (CR) 中启用并行节点排空，以更快地推出 SR-IOV 网络配置。

要配置并行排空，请使用 **SriovNetworkPoolConfig** CR 创建节点池。然后，您可以在池中添加节点，并在 Operator 可以并行排空的池中定义最大节点数。使用这个方法，您可以启用并行排空来更快地重新配置，同时确保池中仍有足够的节点来处理任何正在运行的工作负载。

注意

节点只能属于一个 SR-IOV 网络池配置。如果节点不是池的一部分，则节点会添加到一个虚拟的默认池，它带有一次仅排空一个节点的配置。

节点可能会在排空过程中重启。

该流程需要您创建 SR-IOV 资源，然后并行排空节点。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 SR-IOV Network Operator。
- 节点具有支持 SR-IOV 的硬件。

流程

1. 创建一个定义 **SriovNetworkPoolConfig** 资源的 YAML 文件：

sriov-nw-pool.yaml 文件示例

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

其中：

name

指定 **SriovNetworkPoolConfig** 对象的名称。

namespace

指定安装 SR-IOV Network Operator 的命名空间。

maxUnavailable

为在更新过程中可用的节点指定一个整数或百分比值。例如，如果您有 10 个节点，并且将最大不可用设置为 2，那么可以随时并行排空 2 个节点，保留 8 个节点来处理工作负载。

nodeSelector

使用节点选择器指定要添加池的节点。本例将具有 **worker** 角色的所有节点添加到池中。

2. 运行以下命令来创建 **SriovNetworkPoolConfig** 资源：

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

3. 运行以下命令来创建 **sriov-test** 命名空间：

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

4. 创建一个用于定义 **SriovNetworkNodePolicy** 资源的 YAML 文件，以下是一个 YAML 文件示例：

```
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. 运行以下命令来创建 **SriovNetworkNodePolicy** 资源：

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

6. 创建一个定义 **SriovNetwork** 资源的 YAML 文件：

sriov-network.yaml 文件示例

```
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. 运行以下命令来创建 **SriovNetwork** 资源：

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

8. 运行以下命令，查看您创建的节点池：

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

预期的输出显示节点池的名称，如 **pool-1**，它包括具有 **worker** 角色的所有节点，以及节点池的年龄，以秒为单位，如 **67s**。

9. 更新 **SriovNetworkNodePolicy** 资源中的虚拟功能数量，以触发集群中的工作负载排空：

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

10. 运行以下命令，检查目标集群上的排空状态：

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

输出示例

```
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
```

当排空过程完成后，**SYNC STATUS** 变为 **Succeeded**，**DESIRED SYNC STATE** 和 **CURRENT SYNC STATE** 的值返回 **IDLE**。

3.1.2.8. 排除 NUMA 感知调度的 SR-IOV 网络拓扑

要将 SR-IOV 网络资源的 Non-Uniform Memory Access (NUMA)节点排除到拓扑管理器，您可以在 **SriovNetworkNodePolicy** 自定义资源中配置 **excludeTopology** 规格。在 NUMA 感知 pod 调度过程中，使用此配置来实现更灵活的 SR-IOV 网络部署。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已将 CPU Manager 策略配置为 **static**。有关 CPU Manager 的更多信息，请参阅 *附加资源*部分。
- 您已将 Topology Manager 策略配置为 **single-numa-node**。
- 已安装 SR-IOV Network Operator。

流程

1. 创建 **SriovNetworkNodePolicy** CR：
   a. 将以下 YAML 保存到 **sriov-network-node-policy.yaml** 文件中，替换 YAML 中的值以匹配您的环境：

```
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) (1) SR-IOV 网络设备插件的资源名称。此 YAML 使用示例 **resourceName** 值。

(2) 使用网络接口控制器(NIC 选择器)识别要配置的 Operator 的设备。

(3) 要将 SR-IOV 网络资源的 NUMA 节点排除到拓扑管理器，请将值设为 **true**。默认值为 **false**。

注意

如果多个 **SriovNetworkNodePolicy** 资源都以同一 SR-IOV 网络资源为目标，则 **SriovNetworkNodePolicy** 资源必须具有与 **excludeTopology** 规格相同的值。否则，冲突策略将被拒绝。

b. 运行以下命令来创建 **SriovNetworkNodePolicy** 资源。成功输出列出了 **SriovNetworkNodePolicy** 资源的名称，状态为 **created**。

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

2. 创建 **SriovNetwork** CR：

a. 将以下 YAML 保存到 **sriov-network.yaml** 文件中，替换 YAML 中的值以匹配您的环境：

```
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) 将 **sriov-numa-0-network** 替换为 SR-IOV 网络资源的名称。

(2) 指定上一步中的 **SriovNetworkNodePolicy** CR 的资源名称。此 YAML 使用示例 **resourceName** 值。

(3) 输入 SR-IOV 网络资源的命名空间。

(4) 输入 SR-IOV 网络的 IP 地址管理配置。

b. 运行以下命令来创建 **SriovNetwork** 资源。成功输出列出了 **SriovNetwork** 资源的名称，状态为 **created**。

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

3. 创建 pod 并从上一步中分配 SR-IOV 网络资源：

a. 将以下 YAML 保存到 **sriov-network-pod.yaml** 文件中，替换 YAML 中的值以匹配您的环境：

```
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 这是使用 **SriovNetworkNodePolicy** 资源的 **SriovNetwork** 资源的名称。

b. 运行以下命令来创建 **Pod** 资源。预期输出显示 **Pod** 资源的名称，状态为 **created**。

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

验证

1. 运行以下命令，将 **<pod_name>** 替换为 pod 的名称来验证 pod 的状态：

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

输出示例

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

2. 打开目标 pod 的 debug 会话，以验证 SR-IOV 网络资源是否已部署到与内存和 CPU 资源不同的节点上。

a. 运行以下命令，使用 pod 打开 debug 会话，将 <pod_name> 替换为目标 pod 名称。

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

b. 将 **/host** 设为 debug shell 中的根目录。debug pod 从 pod 中的 **/host** 中的主机挂载 root 文件系统。将根目录改为 **/host**，您可以从主机文件系统中运行二进制文件：

```
$ chroot /host
```

c. 运行以下命令，查看有关 CPU 分配的信息：

```
$ lscpu | grep NUMA
```

输出示例

```
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
```

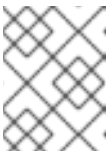
输出示例

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

预期输出显示 CPU (1, 3, 5, 和 7)，它们分配给一个 **NUMA** 节点，如 **NUMA node1**。SR-IOV 网络资源可以使用来自另一个 **NUMA** 节点的 NIC，如 **NUMA node0**。请注意，**ffff** 十六进制值代表运行一个进程的 CPU 内核。

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

预期输出显示 **NUMA** 节点的数，如 **0**。



注意

如果将 **excludeTopology** 规格设为 **True**，则所需资源可能存在于同一个 NUMA 节点上。

3.1.2.9. 其他资源

- 配置 SR-IOV 网络设备
- 使用 CPU Manager

第 4 章 配置 SR-IOV INFINIBAND 网络附加

您可以为集群中的单根 I/O 虚拟化（SR-IOV）设备配置 InfiniBand（IB）网络附加。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

4.1. INFINIBAND 设备配置对象

您可以通过定义 **SriovIBNetwork** 对象来配置 InfiniBand（IB）网络设备。

以下 YAML 描述了 **SriovIBNetwork** 对象：

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

其中：

name

对象的名称。SR-IOV Network Operator 创建一个名称相同的 **NetworkAttachmentDefinition** 对象。

namespace

安装 SR-IOV Operator 的命名空间。

resourceName

用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象中的 **spec.resourceName** 参数的值。

networkNamespace

SriovIBNetwork 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到网络设备。

ipam

可选参数。为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

linkState

可选参数。虚拟功能(VF)的链接状态。允许的值是 **enable**、**disable** 和 **auto**。

功能

可选参数。为此网络配置功能。您可以指定 **'{ "ips": true }'** 来启用 IP 地址支持，或 **'{ "infinibandGUID": true }'** 来启用 IB Global Unique Identifier (GUID)支持。

4.1.1. 为动态分配双栈 IP 地址创建配置

您可以动态地将双栈 IP 地址分配给二级网络，以便 pod 可以通过 IPv4 和 IPv6 地址进行通信。

您可以在 **ipRanges** 参数中配置以下 IP 地址分配类型：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 **type** 设置为 **whereabouts**。
2. 使用 **ipRanges** 来分配 IP 地址，如下例所示：

```
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. 将二级网络附加到 pod。如需更多信息，请参阅"将 pod 添加到二级网络"。

验证

- 输入以下命令验证所有 IP 地址是否已分配给 pod 命名空间内的网络接口：

```
$ oc exec -it <pod_name> -- ip a
```

其中：

<podname>

pod 的名称。

4.1.2. 配置网络附加的 IP 地址分配

对于二级网络，可以使用 IP 地址管理 (IPAM) CNI 插件来分配 IP 地址，该插件支持多种分配方法，包括动态主机配置协议 (DHCP) 和静态分配。

负责动态分配 IP 地址的 DHCP IPAM CNI 插件与两个不同的组件一起运行：

- CNI 插件：负责与 Kubernetes 网络堆栈集成，以请求和释放 IP 地址。
- DHCP IPAM CNI 守护进程：用于 DHCP 事件的监听程序，该事件与环境中的现有 DHCP 服务器协调，以处理 IP 地址分配请求。这个守护进程本身并不是一个 DHCP 服务器。

对于在其 IPAM 配置中需要 **type: dhcp** 的网络，请确保 DHCP 服务器满足以下条件：

- DHCP 服务器可用并在环境中运行。
- DHCP 服务器位于集群外部，您希望该服务器是组成客户的现有网络基础架构的一部分。
- DHCP 服务器被正确配置为为节点提供 IP 地址。

如果在环境中 DHCP 服务器不可用，请考虑使用 Whereabouts IPAM CNI 插件。Whereabouts CNI 提供类似的 IP 地址管理功能，而无需外部 DHCP 服务器。

注意

当没有外部 DHCP 服务器或首选静态 IP 地址管理时，请使用 Whereabouts CNI 插件。Whereabouts 插件包含一个协调器守护进程来管理过时的 IP 地址分配。

通过包含单独的守护进程(DHCP IPAM CNI 守护进程)来确保在容器生命周期内定期续订 DHCP 租期。要部署 DHCP IPAM CNI 守护进程，请更改 Cluster Network Operator (CNO) 配置，以触发此守护进程的部署，作为二级网络设置的一部分。

4.1.2.1. 静态 IP 地址分配配置

下表描述了静态 IP 地址分配的配置：

表 4.1. ipam 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。
Routes	**数组**	指定要在 pod 中配置的路由的一组对象。
dns	**数组**	可选：指定 DNS 配置的对象数组。

address 数组需要带有以下字段的对象：

表 4.2. ipam.addresses[] array

字段	类型	描述

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如：如果您指定 **10.10.21.10/24**，那么会为二级网络分配 IP 地址 **10.10.21.10**，网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 4.3. **ipam.routes[]** array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	路由网络流量的网关。

表 4.4. **ipam.dns** object

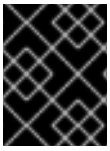
字段	类型	描述
nameservers	**数组**	发送 DNS 查询的一个或多个 IP 地址的数组。
domain	**数组**	要附加到主机名的默认域。例如，如果将域设置为 **example.com**，对 **example-host** 的 DNS 查找查询将被改写为 **example-host.example.com**。
search	**数组**	在 DNS 查找查询过程中，附加到非限定主机名（如 **example-host**）的域名的数组。

静态 IP 地址分配配置示例

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

4.1.2.2. 动态 IP 地址(DHCP)分配配置

pod 在创建时获取其原始 DHCP 租期。该租期必须由集群中运行的一个小型的 DHCP 服务器部署定期续订。



重要

对于以太网网络附加，SR-IOV Network Operator 不会创建 DHCP 服务器部署。Cluster Network Operator 负责创建最小 DHCP 服务器部署。

要触发 DHCP 服务器的部署，您必须编辑 Cluster Network Operator 配置来创建 shim 网络附加，如下例所示：

shim 网络附加定义示例

```
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"
        }
      }
  # ...
```

其中：

type

为集群指定动态 IP 地址分配。

4.1.2.2.1. 使用 Whereabouts 进行动态 IP 地址分配配置

Whereabouts CNI 插件允许在不使用 DHCP 服务器的情况下动态地将 IP 地址分配给二级网络。

Whereabouts CNI 插件还支持在单独的 **NetworkAttachmentDefinition** CRD 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

4.1.2.2.1.1. 动态 IP 地址配置参数

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 4.5. ipam whereabouts 配置参数

字段	类型	描述
type	**string**	IPAM 地址类型。需要 **abouts** 的值。

字段	类型	描述
range	**string**	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	**数组**	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	**string**	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

4.1.2.2.1.2. 使用 Whereabouts 排除 IP 地址范围的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的 NAD 文件中的动态地址分配配置：

Whereabouts 动态 IP 地址分配排除特定的 IP 地址范围

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

4.1.2.2.1.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

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

其中：

network_name

可选参数。如果设置，必须与 **NetworkAttachmentDefinition 2** 的 **network_name** 匹配。

NetworkAttachmentDefinition 2

```
{
```

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

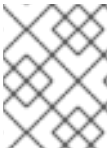
其中：

network_name

可选参数。如果设置，必须与 **NetworkAttachmentDefinition 1** 的 **network_name** 匹配。

4.1.2.3. 配置 SR-IOV 额外网络

您可以通过创建一个 **SriovIBNetwork** 对象来配置使用 SR-IOV 硬件的额外网络。创建 **SriovIBNetwork** 对象时，SR-IOV Network Operator 会自动创建一个 **NetworkAttachmentDefinition** 对象。



注意

如果一个 **SriovIBNetwork** 对象已被附加到状态为 **running** 的 pod，则不要修改或删除它。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建一个 **SriovIBNetwork** 对象，然后在 **<name>.yaml** 文件中保存 YAML，其中 **<name>** 是这个额外网络的名称。对象规格可能类似以下示例：

```
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. 运行以下命令来创建对象：

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

这里的 **<name>** 指定额外网络的名称。

3. 可选： 要确认与您在上一步中创建的 **SriovIBNetwork** 对象关联的 **NetworkAttachmentDefinition** 对象是否存在，请输入以下命令。将 **<namespace>** 替换为您在 **SriovIBNetwork** 对象中指定的 networkNamespace。

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

4.1.2.4. 基于 InfiniBand 的 SR-IOV 附加的运行时配置

将 pod 附加到额外网络时，您可以指定运行时配置来为 pod 进行特定的自定义。例如,,您可以请求特定的 MAC 硬件地址。

您可以通过在 pod 规格中设置注解来指定运行时配置。注解键是 **k8s.v1.cni.cncf.io/network**,它接受一个 JSON 对象来描述运行时配置。

以下 JSON 描述了基于 InfiniBand 的 SR-IOV 网络附加的运行时配置选项。

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

其中：

name

SR-IOV 网络附加定义 CR 的名称。

infiniband-guid

SR-IOV 设备的 InfiniBand GUID。要使用这个功能，还必须在 **SriovIBNetwork** 对象中指定 **{ "infinibandGUID": true }**。

ips

从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能,你还必须在 **SriovIBNetwork** 对象中指定 **{ "ips": true }**。

```
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"]
```

4.1.2.5. 将 pod 添加到二级网络

您可以将 pod 添加到二级网络。pod 继续通过默认网络发送与集群相关的普通网络流量。

创建 pod 时，二级网络会附加到 pod。但是，如果 pod 已存在，则无法将二级网络附加到其中。

pod 必须与二级网络位于同一个命名空间中。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 登录到集群。

流程

1. 为 **Pod** 对象添加注解。只能使用以下注解格式之一：

   a. 要在没有自定义的情况下附加二级网络，请使用以下格式添加注解：

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

   其中：

   k8s.v1.cni.cncf.io/networks

   指定要与 pod 关联的二级网络的名称。要指定多个二级网络，请使用逗号分隔每个网络。逗号之间不可包括空格。如果您多次指定相同的二级网络，则该 pod 会将多个网络接口附加到该网络。

   b. 要通过自定义来附加二级网络，请添加具有以下格式的注解：

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

   其中：

   name

   指定 **NetworkAttachmentDefinition** 对象定义的二级网络的名称。

   namespace

指定定义 **NetworkAttachmentDefinition** 对象的命名空间。

default-route

可选参数。为默认路由指定覆盖，如 **192.168.17.1**。

2. 运行以下命令来创建 pod。

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

将 **<name>** 替换为 pod 的名称。

3. 可选：输入以下命令确认 **pod** CR 中是否存在注解。将 **<name>** 替换为 pod 的名称。

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

在以下示例中，**example-pod** pod 附加到 **net1** 二级网络：

```
$ 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: |-
      [{
          "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:
  ...
```

其中：

k8s.v1.cni.cncf.io/network-status

指定对象的 JSON 数组。每个对象描述附加到 pod 的二级网络的状态。注解值保存为纯文本值。

4.1.2.5.1. 向 pod 公开 vfio-pci SR-IOV 设备的 MTU

将 pod 添加到额外网络后，您可以检查 MTU 可供 SR-IOV 网络使用。

流程

1. 运行以下命令，检查 pod 注解是否包含 MTU：

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

以下示例显示了输出示例：

```
"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. 运行以下命令，验证 pod 中的 **/etc/podnetinfo/** 中是否有 MTU：

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

以下示例显示了输出示例：

```
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\"
        }
    }
    }]"
```

4.1.2.6. 其他资源

- 配置 SR-IOV 网络设备
- 使用 CPU Manager
- 排除 NUMA 感知调度的 SR-IOV 网络拓扑

第 5 章 为 SR-IOV 配置 RDMA 子系统

远程直接内存访问(RDMA)允许在两个系统间直接访问内存，而无需涉及任一系统的操作系统。您可以在单根 I/O 虚拟化(SR-IOV)上配置 RDMA Container Network Interface (CNI)，以启用容器间的高性能、低延迟通信。当您将 RDMA 与 SR-IOV 相结合时，您可以提供一个机制来公开 Mellanox 以太网设备的硬件计数器，以便在 Data Plane Development Kit (DPDK)应用程序中使用。

5.1. 配置 SR-IOV RDMA CNI

在 SR-IOV 上配置 RDMA CNI。

注意

这个过程只适用于 Mellanox 设备。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。

流程

1. 创建一个 **SriovNetworkPoolConfig** CR，并将它保存为 **sriov-nw-pool.yaml**，如下例所示：

   SriovNetworkPoolConfig CR 示例

```
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) 将 RDMA 网络命名空间模式设置为 **exclusive**。

2. 运行以下命令来创建 **SriovNetworkPoolConfig** 资源：

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

3. 创建一个 **SriovNetworkNodePolicy** CR，并将它保存为 **sriov-node-policy.yaml**，如下例所示：

   SriovNetworkNodePolicy CR 示例

```
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 激活 RDMA 模式。

4. 运行以下命令来创建 **SriovNetworkNodePolicy** 资源：

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

5. 创建一个 **SriovNetwork** CR，并将它保存为 **sriov-network.yaml**，如下例所示：

SriovNetwork CR 示例

```
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 创建 RDMA 插件。

6. 运行以下命令来创建 **SriovNetwork** 资源：

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

验证

1. 创建 **Pod** CR，并将它保存为 **sriov-test-pod.yaml**，如下例所示：

运行时配置示例

```
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. 运行以下命令来创建测试 pod：

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

3. 运行以下命令登录到测试 pod：

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

4. 运行以下命令，验证 **hw-counters** 目录的路径是否存在：

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

输出示例

```
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 为 SR-IOV 网络配置接口级网络 SYSCTL 设置和 ALL-MULTICAST 模式

作为集群管理员，您可以使用连接到 SR-IOV 网络设备的 pod 的 tuning Container Network Interface (CNI) meta 插件更改接口级网络 sysctl 和几个接口属性，如 promiscuous 模式、all-multicast 模式、MTU 和 MAC 地址。

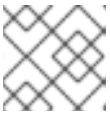
在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

6.1. 为启用了 SR-IOV 的 NIC 标记节点

如果您只想在 SR-IOV 功能的节点上启用 SR-IOV，请执行几种方法：

1. 安装 Node Feature Discovery (NFD) Operator。NFD 检测启用了 SR-IOV 的 NIC，并使用 **node.alpha.kubernetes-incubator.io/nfd-network-sriov.enabled = true** 标记节点。
2. 检查每个节点的 **SriovNetworkNodeState** CR。**interfaces** 小节包括 worker 节点上 SR-IOV Network Operator 发现的所有 SR-IOV 设备列表。使用以下命令，为每个节点标记 **feature.node.kubernetes.io/network-sriov.enabled: "true"** ：

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



注意

您可以使用您需要的任何名称标记节点。

6.2. 设置一个 SYSCTL 标记

您可以为连接到 SR-IOV 网络设备的 pod 设置接口级网络 **sysctl** 设置。

在本例中，**net.ipv4.conf.IFNAME.accept_redirects** 在创建的虚拟接口上设置为 **1**。

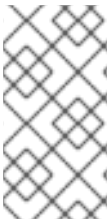
sysctl-tuning-test 是本例中使用的命名空间。

- 使用以下命令来创建 **sysctl-tuning-test** 命名空间：

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

6.2.1. 在使用 SR-IOV 网络设备的节点上设置一个 sysctl 标志

SR-IOV Network Operator 将 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** 自定义资源定义 (CRD) 添加到 OpenShift Container Platform。您可以通过创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR) 来配置 SR-IOV 网络设备。



注意

当应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空并重启节点。

它可能需要几分钟时间来应用配置更改。

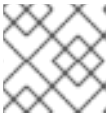
按照以下步骤创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR)。

流程

1. 创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR)。例如，将以下 YAML 保存为文件 **policyoneflag-sriov-node-network.yaml** ：

```
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. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。
4. 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。
5. 可选： priority 是一个 **0** 到 **99** 之间的整数。较小的值具有更高的优先级。例如，优先级 **10** 是高于优先级 **99**。默认值为 **99**。
6. 为 SR-IOV 物理网络设备创建的虚拟功能（VF）的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 **127**。
7. NIC 选择器标识要配置的 Operator 的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。如果指定了**rootDevices**，则必须同时为 **vendor**、**deviceID** 或 **pfNames** 指定一个值。如果同时指定了 **pfNames** 和 **rootDevices**，请确保它们引用同一设备。如果您为 **netFilter** 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。
8. 可选：该设备的一个或多个物理功能（PF）名称的数组。
9. 可选：虚拟功能的驱动程序类型。唯一允许的值是 **netdevice**。对于裸机节点上的 DPDK 模式的 Mellanox NIC，请将 **isRdma** 设置为 **true**。
10. 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 **false**。如果 **isRdma** 参数设为 **true**，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。将 **isRdma** 设置为 **true**，并将 **needVhostNet** 设置为 **true** 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。



注意

vfio-pci 驱动程序类型不被支持。

2. 创建 **SriovNetworkNodePolicy** 对象：

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

应用配置更新后，**sriov-network-operator** 命名空间中的所有 pod 将变为 **Running** 状态。

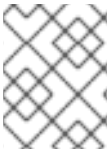
3. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 **<node_name>** 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。预期输出显示 **Succeeded**。

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

6.2.2. 在 SR-IOV 网络中配置 sysctl

您可以通过将调优配置添加到 **SriovNetwork** 资源的可选 **metaPlugins** 参数，在 SR-IOV 创建的虚拟接口上设置特定于接口的 **sysctl** 设置。

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 **NetworkAttachmentDefinition** 自定义资源（CR）。



注意

不要编辑 SR-IOV Network Operator 所管理的 **NetworkAttachmentDefinition** 自定义资源。这样做可能会破坏额外网络上的网络流量。

要更改接口级别网络 **net.ipv4.conf.IFNAME.accept_redirects sysctl** 设置，请使用 Container Network Interface (CNI) 调整插件创建额外的 SR-IOV 网络。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为额外 SR-IOV 网络附加创建 **SriovNetwork** 自定义资源 (CR) 并插入 **metaPlugins** 配置，如下例所示。将 YAML 保存为文件 **sriov-network-interface-sysctl.yaml**。

```
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. 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。
2. 安装 SR-IOV Network Operator 的命名空间。
3. 用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象中的 **spec.resourceName** 参数的值。
4. **SriovNetwork** 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。
5. 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。
6. 可选：为额外网络设置功能。您可以指定 **"{ "ips": true }"** 来启用 IP 地址支持，或指定 **"{ "mac": true }"** 来启用 MAC 地址支持。
7. 可选： metaPlugins 参数用于为该设备添加额外的功能。在这种情况下，将 **type** 字段设置为 **tuning**。指定在 **sysctl** 字段中设置的接口级网络 **sysctl**。

2. 创建 **SriovNetwork** 资源：

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

验证 NetworkAttachmentDefinition CR 是否已成功创建

- 运行以下命令，确认 SR-IOV Network Operator 创建了 **NetworkAttachmentDefinition** CR：

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

1. 将 **<namespace>** 替换为您在 **SriovNetwork** 对象中指定的 **networkNamespace** 的值。例如： **sysctl-tuning-test**。预期输出显示 NAD CRD 的名称和创建年龄（以分钟为单位）。

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

验证额外 SR-IOV 网络附加是否成功

要验证 tuning CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

1. 创建 **Pod** CR。将以下 YAML 保存为文件 **examplepod.yaml** ：

```
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 网络附加定义 CR 的名称。

(2) 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 **{ "mac": true }**。

(3) 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 **SriovNetwork** 对象中指定 **{ "ips": true }**。

2. 创建 **Pod** CR：

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

3. 运行以下命令验证 pod 是否已创建：

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

输出示例

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

4. 运行以下命令登录到 pod：

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

5. 验证配置的 sysctl 标记的值。运行以下命令，查找 **net.ipv4.conf.IFNAME.accept_redirects** 的值：

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

6.3. 为与绑定 SR-IOV 接口标记关联的 POD 配置 SYSCTL 设置

您可以为连接到绑定的 SR-IOV 网络设备的 pod 设置接口级网络 **sysctl** 设置。

在本例中，可以配置的特定网络接口级 **sysctl** 设置在绑定接口上设置。

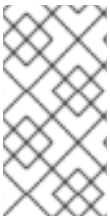
sysctl-tuning-test 是本例中使用的命名空间。

- 使用以下命令来创建 **sysctl-tuning-test** 命名空间：

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

6.3.1. 在带有绑定的 SR-IOV 网络设备的节点上设置所有 sysctl 标志

SR-IOV Network Operator 将 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** 自定义资源定义 (CRD) 添加到 OpenShift Container Platform。您可以通过创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR) 来配置 SR-IOV 网络设备。



注意

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。

它可能需要几分钟时间来应用配置更改。

按照以下步骤创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR)。

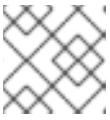
流程

1. 创建一个 **SriovNetworkNodePolicy** 自定义资源 (CR)。将以下 YAML 保存为文件 **policyallflags-sriov-node-network.yaml**。将 **policyallflags** 替换为配置的名称。

```
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. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。
4. 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。
5. 可选： priority 是一个 **0** 到 **99** 之间的整数。较小的值具有更高的优先级。例如，优先级 **10** 是高于优先级 **99**。默认值为 **99**。
6. 为 SR-IOV 物理网络设备创建的虚拟功能 (VF) 的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 **127**。
7. NIC 选择器标识要配置的 Operator 的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。如果指定了**rootDevices**，则必须同时为 **vendor**、**deviceID** 或 **pfNames** 指定一个值。如果同时指定了 **pfNames** 和 **rootDevices**，请确保它们引用同一设备。如果您为 **netFilter** 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。
8. 可选：该设备的一个或多个物理功能（PF）名称的数组。
9. 可选：虚拟功能的驱动程序类型。唯一允许的值是 **netdevice**。对于裸机节点上的 DPDK 模式的 Mellanox NIC，请将 **isRdma** 设置为 **true**。
10. 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 **false**。如果 **isRdma** 参数设为 **true**，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。将 **isRdma** 设置为 **true**，并将 **needVhostNet** 设置为 **true** 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。



注意

vfio-pci 驱动程序类型不被支持。

2. 创建 SriovNetworkNodePolicy 对象：

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

应用配置更新后，sriov-network-operator 命名空间中的所有 pod 将变为 **Running** 状态。

3. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 **<node_name>** 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。预期输出显示 **Succeeded**

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

6.3.2. 在绑定的 SR-IOV 网络中配置 sysctl

您可以在从两个 SR-IOV 接口创建的绑定接口上设置特定于接口的 **sysctl** 设置。为此，可将调优配置添加到绑定网络附加定义的可选 **Plugins** 参数中。

注意

不要编辑 SR-IOV Network Operator 所管理的 **NetworkAttachmentDefinition** 自定义资源。这样做可能会破坏额外网络上的网络流量。

要更改特定的接口级网络 **sysctl** 设置，请按照以下流程使用 Container Network Interface (CNI) 调优插件创建 **SriovNetwork** 自定义资源 (CR)。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为绑定接口创建 **SriovNetwork** 自定义资源 (CR)，如下例所示。将 YAML 保存为文件 **sriov-network-attachment.yaml**。

```
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) 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。

(2) 安装 SR-IOV Network Operator 的命名空间。

(3) 用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象中的 **spec.resourceName** 参数的值。

(4) **SriovNetwork** 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。

(5) 可选：为这个额外网络配置功能。您可以指定 **"{ "ips": true }"** 来启用 IP 地址支持，或指定 **"{ "mac": true }"** 来启用 MAC 地址支持。

2. 创建 **SriovNetwork** 资源：

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

3. 创建绑定网络附加定义，如下例所示。将 YAML 保存为文件 **sriov-bond-network-interface.yaml**。

```
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) 类型是 **bond**。

(2) **mode** 属性指定绑定模式。支持的绑定模式有：

- **balance-rr** - 0
- **active-backup** - 1

- **balance-xor** - 2
  对于 **balance-rr** 或 **balance-xor** 模式，您必须为 SR-IOV 虚拟功能将 **trust** 模式设置为 **on**。

3 对于 active-backup 模式，**failover** 属性是必需的。

4 **linksInContainer=true** 标志告知 Bond CNI 在容器内找到所需的接口。默认情况下，Bond CNI 会查找主机上的这些接口，该接口无法与 SRIOV 和 Multus 集成。

5 **links** 部分定义将用于创建绑定的接口。默认情况下，Multus 将附加的接口命名为 "net"，再加上一个连续的数字。

6 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。在这个 pod 示例 IP 地址中被手动配置，因此在本例中 **ipam** 被设置为 static。

7 为设备添加额外的功能。例如，将 **type** 字段设置为 **tuning**。指定在 **sysctl** 字段中设置的接口级网络 sysctl。这个示例设置可设置的所有接口级网络 **sysctl** 设置。

4. 创建绑定网络附加定义：

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

验证 NetworkAttachmentDefinition CR 是否已成功创建

- 运行以下命令，确认 SR-IOV Network Operator 创建了 **NetworkAttachmentDefinition** CR：

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

1 将 **<namespace>** 替换为您在配置网络附加时指定的 networkNamespace，如 **sysctl-tuning-test**。预期输出显示 NAD CRD 的名称以及创建年龄（以分钟为单位）。

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

验证额外的 SR-IOV 网络资源是否成功

要验证 tuning CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

1. 创建 **Pod** CR。例如，将以下 YAML 保存为文件 **examplepod.yaml** ：

```
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 网络附加定义 CR 的名称。

(2) 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 **{ "mac": true }**。

(3) 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 **SriovNetwork** 对象中指定 **{ "ips": true }**。

2. 应用 YAML：

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

3. 运行以下命令验证 pod 是否已创建：

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

输出示例

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

4. 运行以下命令登录到 pod：

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

5. 验证配置的 **sysctl** 标记的值。运行以下命令，查找 **net.ipv6.neigh.IFNAME.base_reachable_time_ms** 的值：

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

6.4. 关于 ALL-MULTICAST 模式

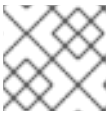
启用 all-multicast 模式（特别是在无根应用程序上下文中）非常重要。如果没有启用此模式，则需要为 Pod 的安全上下文约束(SCC)授予 **NET_ADMIN** 功能。如果您要允许 **NET_ADMIN** 功能授予 pod 权限，以便更改超出其特定要求的更改，您可能会暴露安全漏洞。

tuning CNI 插件支持更改几个接口属性，包括 all-multicast 模式。通过启用此模式，您可以在 SR-IOV 网络设备上配置的虚拟功能 (VF) 上运行的应用程序从其他 VF 上的应用程序接收多播流量，无论是附加到同一还是不同的物理功能。

6.4.1. 在 SR-IOV 网络中启用 all-multicast 模式

您可以通过以下方法在 SR-IOV 接口中启用 all-multicast 模式：

- 在 **SriovNetwork** 资源的 **metaPlugins** 参数中添加调优配置
- 在调优配置中将 **allmulti** 字段设置为 **true**



注意

确保创建启用了信任的虚拟功能 (VF)。

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 **NetworkAttachmentDefinition** 自定义资源（CR）。

注意

不要编辑 SR-IOV Network Operator 所管理的 **NetworkAttachmentDefinition** 自定义资源。这样做可能会破坏额外网络上的网络流量。

按照本指南，在 SR-IOV 网络中启用 all-multicast 模式。

先决条件

- 已安装 OpenShift Container Platform CLI (oc)。
- 以具有 **cluster-admin** 权限的用户身份登录 OpenShift Container Platform 集群。
- 已安装 SR-IOV Network Operator。
- 您已配置了适当的 **SriovNetworkNodePolicy** 对象。

流程

1. 使用以下设置创建一个 YAML 文件，为 Mellanox ConnectX-5 设备定义 **SriovNetworkNodePolicy** 对象。将 YAML 文件保存为 **sriovnetpolicy-mlx.yaml**。

```
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. 可选：如果支持 SR-IOV 的集群节点还没有标记，请添加 **SriovNetworkNodePolicy.Spec.NodeSelector** 标签。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。

3. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

   应用配置更新后，**sriov-network-operator** 命名空间中的所有 pod 会自动进入 **Running** 状态。

4. 运行以下命令来创建 **enable-allmulti-test** 命名空间：

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

5. 为额外 SR-IOV 网络附加创建 **SriovNetwork** 自定义资源 (CR) 并插入 **metaPlugins** 配置，如下例所示，并将该文件保存为 **sriov-enable-all-multicast.yaml**。

```
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) 指定对象的名称。SR-IOV Network Operator 创建一个名称相同的 **NetworkAttachmentDefinition** 对象。

2 指定 SR-IOV Network Operator 安装到的命名空间。

3 指定来自用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象的 **spec.resourceName** 参数的值。

4 为 **SriovNetwork** 对象指定目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。

5 为 IPAM CNI 插件指定一个配置对象作为 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

6 可选：为额外网络设置功能。您可以指定 **"{ "ips": true }"** 来启用 IP 地址支持，或指定 **"{ "mac": true }"** 来启用 MAC 地址支持。

7 指定虚拟功能的信任模式。这必须设置为 "on"。

8 使用 **metaPlugins** 参数为设备添加更多功能。在此用例中，将 **type** 字段设置为 **tuning**，并添加 **allmulti** 字段，并将它设为 **true**。

6. 运行以下命令来创建 **SriovNetwork** 资源：

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

验证 NetworkAttachmentDefinition CR

- 运行以下命令，确认 SR-IOV Network Operator 创建了 **NetworkAttachmentDefinition** CR：

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

1 将 **<namespace>** 替换为您在 **SriovNetwork** 对象中指定的 **networkNamespace** 的值。在本例中，是 **enable-allmulti-test**。预期输出显示 NAD CR 的名称和创建年龄（以分钟为单位）。

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

- 运行以下命令，显示 SR-IOV 网络资源的信息：

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

验证额外的 SR-IOV 网络附加

要验证 tuning CNI 是否已正确配置并附加了额外的 SR-IOV 网络附加，请按照以下步骤执行：

1. 创建 **Pod** CR。将以下 YAML 示例保存到名为 **examplepod.yaml** 的文件中：

```
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) 指定 SR-IOV 网络附加定义 CR 的名称。

(2) 可选：指定从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 **{"mac": true}**。

(3) 可选：指定从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 **SriovNetwork** 对象中指定 **{ "ips": true }**。

2. 运行以下命令来创建 **Pod** CR：

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

3. 运行以下命令验证 pod 是否已创建：

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

输出示例

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

4. 运行以下命令登录到 pod：

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

5. 运行以下命令，列出与 pod 关联的所有接口：

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

输出示例

```
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** 是主接口

(2) **net1@if24** 是配置了 network-attachment-definition 的二级接口，它支持 all-multicast 模式(**ALLMULTI** 标志)

第 7 章 为启用 SR-IOV 的工作负载配置 QINQ 支持

QinQ （正式称为 802.1Q-in-802.1Q）是由 IEEE 802.1ad 定义的联网技术。IEEE 802.1ad 扩展了 IEEE 802.1Q-1998 标准，并通过向已使用 802.1Q 标记的数据包引入额外的 802.1Q 标签增强 VLAN 功能。此方法也称为 VLAN 堆栈或双 VLAN。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

7.1. 关于 802.1Q-IN-802.1Q 支持

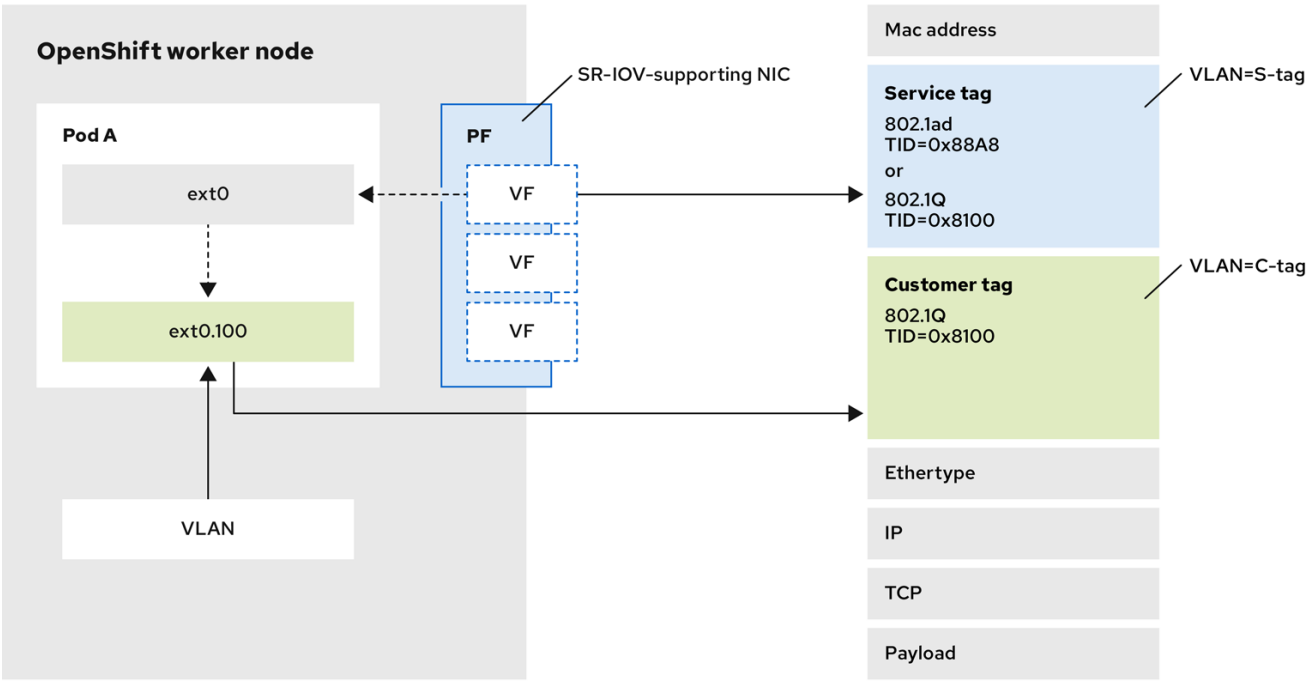
在传统的 VLAN 设置中，帧通常包含一个 VLAN 标签，如 VLAN-100，以及其他元数据，如服务质量 (QoS) 位和协议信息。QinQ 引入了第二个 VLAN 标签，其中服务提供商为其使用指定外部标签，提供它们的灵活性，而内部标签则保留给客户的 VLAN 专用。

QinQ 通过使用双 VLAN 标记促进嵌套 VLAN 的创建，从而在网络环境中实现更精细的分段和流量隔离。这种方法对于您需要通过通用基础架构向多个客户提供基于 VLAN 的服务的服务供应商网络特别有用，同时确保流量隔离和隔离。

下图演示了 OpenShift Container Platform 如何使用 SR-IOV 和 QinQ 实现容器化工作负载的高级网络分段和隔离。

图显示双 VLAN 标记 (QinQ) 如何在支持 SR-IOV 的 worker 节点上工作。位于 pod 命名空间中的 SR-IOV 虚拟功能 (VF)，**ext0** 由带有 VLAN ID 和 VLAN 协议的 SR-IOV Container Network Interface (CNI) 配置。这与 S-tag 对应。在 pod 中，VLAN CNI 使用主接口 **ext0** 创建一个子接口。此子接口使用 802.1Q 协议添加内部 VLAN ID，该协议对应于 C-tag。

这演示了 QinQ 如何在网络内启用更精细的流量分段和隔离。帧结构在右侧详细介绍，突出显示包含 VLAN 标签、EtherType、IP、TCP 和 Payload 部分的包含。QinQ 促进通过共享基础架构向多个客户提供基于 VLAN 的服务，同时确保流量隔离和隔离。



OpenShift Container Platform SR-IOV 解决方案支持在 **SriovNetwork** 自定义资源 (CR) 上设置 VLAN 协议。虚拟功能 (VF) 可以使用此协议来设置 VLAN 标签，也称为外部标签。然后，Pod 可以使用 VLAN CNI 插件来配置内部标签。

表 7.1. 支持的网络接口卡

NIC	802.1ad/802.1Q	802.1Q/802.1Q
Intel X710	否	支持
Intel E810	支持	支持
Mellanox	否	支持

其他资源

配置 VLAN 额外网络

7.2. 为启用 SR-IOV 的工作负载配置 QINQ 支持

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。

流程

1. 使用以下内容创建一个名为 **sriovnetpolicy-810-sriov-node-network.yaml** 的文件：

```
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. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

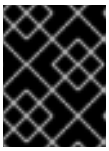
```
$ oc create -f sriovnetpolicy-810-sriov-node-network.yaml
```

3. 打开一个单独的终端窗口，运行以下命令来监控 **openshift-sriov-network-operator** 命名空间中指定节点的 SR-IOV 网络节点状态的同步状态：

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

同步状态表示从 **InProgress** 更改为 **Succeeded**。

4. 创建一个 **SriovNetwork** 对象，并设置名为 S-tag 或 **Service Tag** 的外部 VLAN，因为它属于基础架构。



重要

您必须在交换机的中继接口上配置 VLAN。另外，您可能需要进一步配置一些交换机来支持 QinQ 标记。

a. 使用以下内容创建一个名为 **nad-sriovnetwork-1ad-810.yaml** 的文件：

```
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) 将 S-tag VLAN 标签设置为 **171**。

(2) 指定要分配给虚拟功能 (VF) 的 VLAN 协议。支持的值是 **802.1ad** 和 **802.1q**。默认值为 **802.1q**。

b. 运行以下命令来创建对象：

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

5. 使用内部 VLAN 创建 **NetworkAttachmentDefinition** 对象。内部 VLAN 通常被称为 C-tag 或 **Customer Tag**，它属于 Network Function:

a. 使用以下内容，创建一个名为 **nad-cvlan100.yaml** 的文件：

```
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 指定 pod 中的 VF 接口。默认名称为 **net1**，因为 pod 注解中没有设置名称。

b. 运行以下命令来应用 YAML 文件：

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

验证

- 按照以下步骤，在节点上验证 QinQ 是否活跃：

1. 使用以下内容创建一个名为 **test-qinq-pod.yaml** 的文件：

```
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. 运行以下命令来创建测试 pod：

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

3. 在存在 pod 的目标节点上进入 debug 会话，运行以下命令显示网络接口 **ens5f0** 的信息：

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

输出示例

```
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
```

输出中的 **vlan protocol 802.1ad** ID 表示接口支持带有协议 802.1ad (QinQ)的 VLAN 标记。VLAN ID 为 171。

第 8 章 配置高性能多播

您可以在您的单根 I/O 虚拟化（SR-IOV）硬件网络中使用多播。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

8.1. 高性能多播

OVN-Kubernetes 网络插件支持默认网络上的 pod 间的多播。目前，多播最适用于低带宽协调或服务发现。它不适用于高带宽的应用程序。对于流传输介质应用程序，如 IPTV 和多方视频会议，可以使用 Single Root I/O Virtualization（SR-IOV）硬件来提供接近原生的性能。

使用额外的 SR-IOV 接口进行多播时：

- pod 必须通过额外的 SR-IOV 接口发送或接收多播软件包。
- 连接 SR-IOV 接口的物理网络决定了多播路由和拓扑结构，不受 OpenShift Container Platform 的控制。

8.2. 为多播配置 SR-IOV 接口

以下步骤为多播创建一个 SR-IOV 接口示例。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 您必须作为 **cluster-admin** 角色用户登录集群。

流程

1. 创建一个 **SriovNetworkNodePolicy** 对象：

```
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. 创建一个 **SriovNetwork** 对象：

```
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) 如果选择将 DHCP 配置为 IPAM，请确保通过 DHCP 服务器提供了以下默认路由：**224.0.0.0/5** 和 **232.0.0.0/5**。这会覆盖由默认网络供应商设置的静态多播路由。

3. 创建带有多播应用程序的 pod:

```
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) 只有在应用程序需要为 SR-IOV 接口分配多播 IP 地址时，才需要 **NET_ADMIN** 功能。否则，可以省略它。

第 9 章 使用 DPDK 和 RDMA

OpenShift Container Platform 支持容器化 Data Plane Development Kit (DPDK) 应用程序。您可以使用单一根 I/O 虚拟化（SR-IOV）网络硬件和 Data Plane Development Kit (DPDK) 以及远程直接内存访问 (RDMA) 。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

9.1. 在 POD 中使用虚拟功能的示例

您可以在附加了 SR-IOV VF 的 pod 中运行远程直接内存访问 (RDMA) 或 Data Plane Development Kit (DPDK) 应用程序。

本示例演示了在 RDMA 模式中使用虚拟功能 (VF) 的 pod：

使用 RDMA 模式的 Pod 规格

```
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"]
```

以下示例演示了在 DPDK 模式中使用 VF 的 pod:

使用 DPDK 模式的 Pod 规格

```
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. 在 DPDK 模式中使用 INTEL NIC 的虚拟功能

先决条件

- 安装 OpenShift CLI（**oc**）。
- 安装 SR-IOV Network Operator。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建以下 **SriovNetworkNodePolicy** 对象，然后在 **intel-dpdk-node-policy.yaml** 文件中保存 YAML。

```
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) 将虚拟功能（VF）的驱动器类型指定为 **vfio-pci**。

注意

如需了解 **inSriovNetworkNodePolicy** 的每个选项的详情，请参阅 **Configuring SR-IOV network devices** 部分。

当应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，**openshift-sriov-network-operator** 命名空间中的所有 pod 将变为 **Running** 状态。

2. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

3. 创建以下 **SriovNetwork** 对象，然后在 **intel-dpdk-network.yaml** 文件中保存 YAML。

```
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 为 ipam CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

注意

如需 **SriovNetwork** 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 app-netutil 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 **SriovNetwork** 对象：

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

5. 创建以下 **Pod** spec，然后在 **intel-dpdk-pod.yaml** 文件中保存 YAML。

```
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. 指定 **target_namespace**，它与 **SriovNetwork** 对象 **intel-dpdk-network** 创建于的命令空间相同。如果要在其他命名空间中创建 pod，在 **Pod** spec 和 **SriovNetwork** 对象中更改 **target_namespace**。
2. 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。
3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。
4. 在 **/mnt/huge** 下将巨页卷挂载到 DPDK pod。巨页卷由 emptyDir 卷类型支持，medium 为**Hugepages**。
5. 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果未明确指定，则此资源请求和限制将被 SR-IOV 网络资源注入程序自动添加。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 **SriovOperatorConfig** CR 中的 **enableInjector** 选项设置为 **false** 来禁用它。
6. 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 **static**，并创建带有**有保障的** QoS 的 pod 来实现。
7. 指定巨页大小 **hugepages-1Gi** 或 **hugepages-2Mi** 以及分配给 DPDK pod 的巨页数量。单独配置 **2Mi** 和 **1Gi** 巨页。配置 **1Gi** 巨页需要在节点中添加内核参数。例如：添加内核参数 **default_hugepagesz=1GB**，**hugepagesz=1G** 和 **hugepages=16** 将在系统引导时分配 **16*1Gi** 巨页。

6. 运行以下命令来创建 DPDK pod:

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

9.3. 在带有 MELLANOX NIC 的 DPDK 模式中使用虚拟功能

您可以创建一个网络节点策略，并在带有 Mellanox NIC 的 DPDK 模式中使用虚拟功能创建 Data Plane Development Kit (DPDK) pod。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已安装 Single Root I/O Virtualization (SR-IOV) Network Operator。
- 您已以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 将以下 **SriovNetworkNodePolicy** YAML 配置保存到 **mlx-dpdk-node-policy.yaml** 文件中：

```
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) 指定 SR-IOV 网络设备的设备十六进制代码。

(2) 指定到 **netdevice** 的虚拟功能（VF）的驱动器类型。Mellanox SR-IOV 虚拟功能 (VF) 可以在 DPDK 模式下工作，而无需使用 **vfio-pci** 设备类型。VF 设备作为容器内的内核网络接口显示。

(3) 启用远程直接内存访问 (RDMA) 模式。Mellanox 卡需要在 DPDK 模式下工作。

注意

如需了解 **SriovNetworkNodePolicy** 对象中的每个选项的详细说明，请参阅*配置 SR-IOV 网络设备*。

当应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，**openshift-sriov-network-operator** 命名空间中的所有 pod 将变为 **Running** 状态。

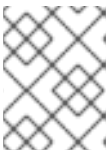
2. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

3. 将以下 **SriovNetwork** YAML 配置保存到 **mlx-dpdk-network.yaml** 文件中：

```
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 为 IP 地址管理 (IPAM) Container Network Interface (CNI) 插件指定一个配置对象作为 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。



注意

如需了解 **SriovNetwork** 对象中的每个选项的详细说明，请参阅*配置 SR-IOV 网络设备*。

app-netutil 选项库提供了几个 API 方法，用于收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 **SriovNetwork** 对象：

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

5. 将以下 **Pod** YAML 配置保存到 **mlx-dpdk-pod.yaml** 文件中：

```
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. 指定 **target_namespace**，它与 **SriovNetwork** 对象 **mlx-dpdk-network** 创建于的命令空间相同。要在不同命名空间中创建 pod，在 **Pod** spec 和 **SriovNetwork** 对象中更改 **target_namespace**。
2. 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。
3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。
4. 将巨页卷挂载到 **/mnt/huge** 下的 DPDK pod 中。巨页卷由 **emptyDir** 卷类型支持，介质是 **Hugepages**。
5. 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果没有明确指定，则 SR-IOV 网络资源注入程序会自动添加此资源请求和限制。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 **SriovOperatorConfig** CR 中的 **enableInjector** 选项设置为 **false** 来禁用它。
6. 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。要做到这一点，将 CPU Manager 策略设置为 **static**，并创建带有 **Guaranteed** 服务质量 (QoS) 的 pod。
7. 指定巨页大小 **hugepages-1Gi** 或 **hugepages-2Mi** 以及分配给 DPDK pod 的巨页数量。单独配置 **2Mi** 和 **1Gi** 巨页。配置 **1Gi** 巨页需要在节点中添加内核参数。

6. 运行以下命令来创建 DPDK pod:

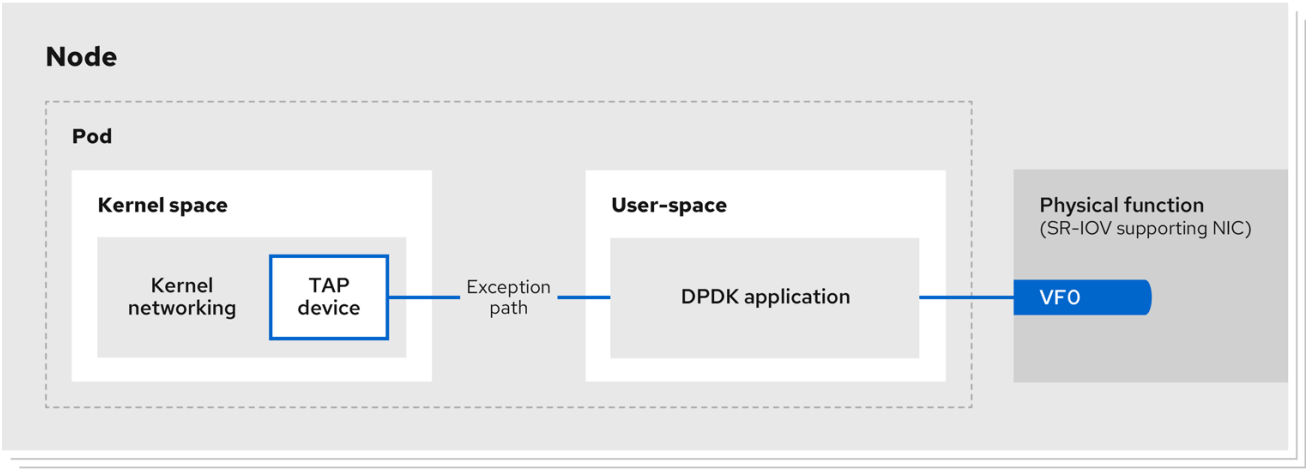
```
$ oc create -f mlx-dpdk-pod.yaml
```

9.4. 使用TAP CNI 运行具有内核访问权限的 ROOTLESS DPDK 工作负载

DPDK 应用程序可以使用 **virtio-user** 作为异常路径，将某些类型的数据包（如日志消息）注入内核进行处理。有关此功能的更多信息，请参阅 Virtio_user 作为例外路径。

在 OpenShift Container Platform 版本 4.14 及更高版本中，您可以使用非特权 pod 和 tap CNI 插件运行 DPDK 应用程序。要启用此功能，您需要在 **SriovNetworkNodePolicy** 对象中将 **needVhostNet** 参数设置为 **true** 来挂载 **vhost-net** 设备。

图 9.1. DPDK 和TAP 示例配置



先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已安装 SR-IOV Network Operator。
- 您以具有 **cluster-admin** 权限的用户身份登录。
- 确保在所有节点上将 **container_use_devices=on** 设置为 root。

注意

使用 Machine Config Operator 设置此 SELinux 布尔值。

流程

1. 创建一个文件，如 **test-namespace.yaml**，其内容类似以下示例：

```
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. 运行以下命令来创建新的 **Namespace** 对象：

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

3. 创建一个文件，如 **sriov-node-network-policy.yaml**，内容类似以下示例：

```
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) 这表示配置集是为 Mellanox Network Interface Controller (NIC) 量身定制的。

(2) 只有 Mellanox NIC 才需要将 **isRdma** 设置为 **true**。

(3) 这会将 **/dev/net/tun** 和 **/dev/vhost-net** 设备挂载到容器中，以便应用可以创建 tap 设备，并将 tap 设备连接到 DPDK 工作负载。

(4) SR-IOV 网络设备厂商的十六进制代码。值 15b3 代表与 Mellanox NIC 关联。

(5) SR-IOV 网络设备的设备十六进制代码。

4. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

5. 创建以下 **SriovNetwork** 对象，然后在 **sriov-network-attachment.yaml** 文件中保存 YAML：

```
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"
```

注意

如需 **SriovNetwork** 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 **app-netutil** 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

6. 运行以下命令来创建 **SriovNetwork** 对象：

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

7. 创建一个文件，如 **tap-example.yaml**，该文件定义网络附加定义，其内容类似以下示例：

```
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) 指定创建 **SriovNetwork** 对象的 **target_namespace**。

8. 运行以下命令来创建 **NetworkAttachmentDefinition** 对象：

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

9. 创建一个文件，如 **dpdk-pod-rootless.yaml**，其内容类似以下示例：

```
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. 指定 **target_namespace** 创建 **SriovNetwork** 对象。如果要在其他命名空间中创建 pod，在 **Pod** spec 和 **SriovNetwork** 对象中更改 **target_namespace**。
2. 设置在这些卷中创建的卷挂载目录和文件的组所有权。
3. 指定用于运行容器的主要组 ID。
4. 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。
5. 从容器的 securityContext 中删除所有功能 (**ALL**) 意味着容器在正常操作之外没有特殊的特权。

6 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。这些功能还必须使用 **setcap** 命令在二进制文件中设置。

7 Mellanox 网络接口控制器(NIC)需要 **NET_RAW** 功能。

8 指定用于运行容器的用户 ID。

9 此设置表示 pod 中的容器或容器不应被授予对主机系统的特权访问权限。

10 此设置允许容器在可能已分配的初始非 root 特权外升级其特权。

11 此设置可确保容器以非 root 用户身份运行。这有助于强制实施最小特权的原则，限制对容器造成潜在的影响，并减少攻击面。

12 在 **/mnt/huge** 下将巨页卷挂载到 DPDK pod。巨页卷由 emptyDir 卷类型支持，medium 为**Hugepages**。

13 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果没有明确指定，则 SR-IOV 网络资源注入程序会自动添加此资源请求和限制。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 **SriovOperatorConfig** CR 中的 **enableInjector** 选项设置为 **false** 来禁用它。

14 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 **static**，并创建带有**有保障的** QoS 的 pod 来实现。

15 指定巨页大小 **hugepages-1Gi** 或 **hugepages-2Mi** 以及分配给 DPDK pod 的巨页数量。单独配置 **2Mi** 和 **1Gi** 巨页。配置 **1Gi** 巨页需要在节点中添加内核参数。例如：添加内核参数 **default_hugepagesz=1GB**，**hugepagesz=1G** 和 **hugepages=16** 将在系统引导时分配 **16*1Gi** 巨页。

16 如果您的性能配置集没有命名为 **cnf-performance profile**，请将该字符串替换为正确的性能配置集名称。

10. 运行以下命令来创建 DPDK pod:

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

其他资源

- 创建性能配置集
- 配置 SR-IOV 网络设备

9.5. 实现特定 DPDK 行率概述

要实现特定的 Data Plane Development Kit (DPDK) 行率，部署 Node Tuning Operator 并配置单根 I/O 虚拟化 (SR-IOV)。您还必须为以下资源调整 DPDK 设置：

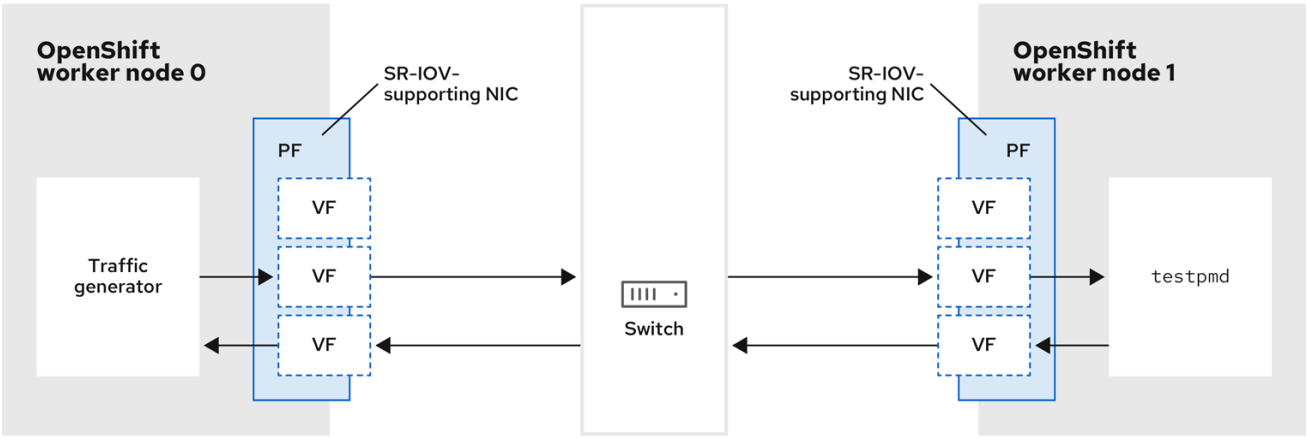
- 隔离的 CPU
- Hugepages
- 拓扑调度程序

注意

在早期版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift Container Platform 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

DPDK 测试环境

下图显示了流量测试环境的组件：



- **流量生成器**：可以生成高容量数据包流量的应用。
- **SR-IOV 支持 NIC**：与 SR-IOV 兼容的网络接口卡。该卡在物理接口上运行多个虚拟功能。
- **物理功能 (PF)**：支持 SR-IOV 接口的网络适配器的 PCI Express (PCIe) 功能。
- **虚拟功能 (VF)**：支持 SR-IOV 的网络适配器上的轻量级 PCIe 功能。VF 与网络适配器上的 PCIe PF 关联。VF 代表网络适配器的虚拟实例。
- **交换机**：网络交换机。节点也可以重新连接到回来。
- **testpmd**：DPDK 中包含的示例应用程序。**testpmd** 应用可用于在数据包转发模式下测试 DPDK。**testpmd** 应用程序也是如何使用 DPDK 软件开发套件 (SDK) 构建功能全面的应用程序的示例。
- **worker 0** 和 **worker 1**：OpenShift Container Platform 节点。

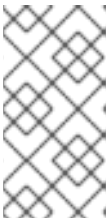
9.6. 使用 SR-IOV 和 NODE TUNING OPERATOR 实现 DPDK 行率

您可以使用 Node Tuning Operator 配置隔离的 CPU、巨页和拓扑调度程序。然后，您可以使用 Node Tuning Operator 和单根 I/O 虚拟化 (SR-IOV) 来实现特定的 Data Plane Development Kit (DPDK) 行率。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已安装 SR-IOV Network Operator。
- 您已以具有 **cluster-admin** 权限的用户身份登录。

- 您已部署了独立的 Node Tuning Operator。



注意

在以前版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

流程

1. 根据以下示例创建 **PerformanceProfile** 对象：

```
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) 如果系统上启用了超线程，请分配与 **isolated** 和 **reserved** CPU 组的相关符号链接。如果系统包含多个非统一内存访问节点 (NUMA)，请将两个 NUMA 中的 CPU 分配给这两个组。您还可以将 Performance Profile Creator 用于此任务。如需更多信息，请参阅*创建性能配置集*。

(2) 您还可以指定将队列设置为保留 CPU 数的设备列表。如需更多信息，请参阅*使用 Node Tuning Operator 缩减 NIC 队列*。

(3) 分配所需的巨页的数量和大小。您可以为巨页指定 NUMA 配置。默认情况下，系统会为系统上的每个 NUMA 节点分配一个偶数数量。如果需要，您可以请求对节点使用实时内核。如需更多信息，请参阅*置备具有实时功能的 worker*。

2. 将 **yaml** 文件保存为 **mlx-dpdk-perfprofile-policy.yaml**。

3. 使用以下命令应用性能配置集：

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

9.6.1. 用于容器应用程序的 DPDK 库

一个可选的库 **app-netutil** 提供了几个 API 方法，用于从该 pod 中运行的容器内收集 pod 的网络信息。

此库可以将 SR-IOV 虚拟功能（VF）集成到 Data Plane Development Kit（DPDK）模式中。该程序库提供 Golang API 和 C API。

当前，采用三个 API 方法：

GetCPUInfo()

此函数决定了哪些 CPU 可供容器使用并返回相关列表。

GetHugepages()

此函数决定了每个容器在 **Pod** spec 中请求的巨页内存量并返回相应的值。

GetInterfaces()

此函数决定了容器中的一组接口，并返回相关的列表。返回值包括每个接口的接口类型和特定于类型的数据。

库的存储库包括用于构建容器镜像 **dpdk-app-centos** 的示例 Dockerfile。根据容器集规格中的环境变量，容器镜像可以运行以下 DPDK 示例应用之一： **l2fwd**、**l3wd** 或 **testpmd**。容器镜像提供了一个将 **app-netutil** 库集成到容器镜像本身的示例。库也可以集成到 init 容器中。init 容器可以收集所需的数据，并将数据传递给现有的 DPDK 工作负载。

9.6.2. 用于虚拟功能的 SR-IOV Network Operator 示例

您可以使用单根 I/O 虚拟化 (SR-IOV) Network Operator 从节点上分配和配置虚拟功能 (VF) 的虚拟功能 (VF)。

如需有关部署 Operator 的更多信息，请参阅 *安装 SR-IOV Network Operator* 。有关配置 SR-IOV 网络设备的更多信息，请参阅 *配置 SR-IOV 网络设备*。

在 Intel VF 和 Mellanox VF 上运行 Data Plane Development Kit (DPDK) 工作负载之间存在一些区别。本节为这两个 VF 类型提供对象配置示例。以下是用于在 Intel NIC 上运行 DPDK 应用程序的 **sriovNetworkNodePolicy** 对象示例：

```
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 对于 Intel NIC，**deviceType** 必须是 **vfio-pci**。

2 如果需要内核与 DPDK 工作负载通信，请添加 **needVhostNet: true**。这会将 **/dev/net/tun** 和 **/dev/vhost-net** 设备挂载到容器中，以便应用可以创建 tap 设备，并将 tap 设备连接到 DPDK 工作负载。

以下是 Mellanox NIC 的 **sriovNetworkNodePolicy** 对象示例：

```
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 对于 Mellanox 设备，**deviceType** 必须是 **netdevice**。

2 对于 Mellanox 设备，**isRdma** 必须为 **true**。Mellanox 卡使用流 Bifurcation 连接到 DPDK 应用程序。这种机制在 Linux 用户空间和内核空间之间分割流量，并增强了行速率处理能力。

9.6.3. SR-IOV 网络 Operator 示例

以下是 **sriovNetwork** 对象的示例定义。在这种情况下，Intel 和 Mellanox 配置是相同的：

```
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 您可以使用不同的 IP 地址管理 (IPAM) 实现，如 Whereabouts。如需更多信息，请参阅*使用 Whereabouts 进行动态 IP 地址分配配置*。

2 您必须请求创建网络附加定义的 **networkNamespace**。您必须在 **openshift-sriov-network-operator** 命名空间内创建 **sriovNetwork** CR。

3 **resourceName** 值必须与在 **sriovNetworkNodePolicy** 下创建的 **resourceName** 相匹配。

9.6.4. DPDK 基础工作负载示例

以下是数据平面开发套件 (DPDK) 容器的示例：

```
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. 请求您需要的 SR-IOV 网络。设备的资源将自动注入。
2. 禁用 CPU 和 IRQ 负载均衡基础。如需更多信息，请参阅*禁用单个 pod 的中断处理*。
3. 将 **runtimeClass** 设置为 **performance-performance**。不要将 **runtimeClass** 设置为 **HostNetwork** 或 **privileged**。
4. 为请求和限值请求相同数量的资源，以启动具有 **Guaranteed** 服务质量 (QoS) 的 pod。

注意

不要使用 **SLEEP** 启动容器集，然后执行到容器集以启动 testpmd 或 DPDK 工作负载。这可添加额外的中断，因为 **exec** 进程没有固定到任何 CPU。

9.6.5. testpmd 脚本示例

以下是运行 **testpmd** 的示例脚本：

```
#!/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
```

这个示例使用两个不同的 **sriovNetwork** CR。环境变量包含分配给 pod 的虚拟功能 (VF) PCI 地址。如果您在 pod 定义中使用相同的网络，您必须分割 **pciAddress**。配置流量生成器的正确 MAC 地址非常重要。这个示例使用自定义 MAC 地址。

9.7. 在带有 MELLANOX NIC 的 RDMA 模式中使用虚拟功能

重要

RDMA over Converged Ethernet (RoCE) 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅以下链接：

- 技术预览功能支持范围

在 OpenShift Container Platform 上使用 RDMA 时，RDMA over Converged Ethernet (RoCE) 是唯一支持的模式。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 安装 SR-IOV Network Operator。

- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建以下 **SriovNetworkNodePolicy** 对象，然后在 **mlx-rdma-node-policy.yaml** 文件中保存 YAML。

```
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) 指定 SR-IOV 网络设备的设备十六进制代码。

(2) 指定到 **netdevice** 的虚拟功能（VF）的驱动器类型。

(3) 启用 RDMA 模式。

注意

如需了解 **inSriovNetworkNodePolicy** 的每个选项的详情，请参阅 **Configuring SR-IOV network devices** 部分。

当应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，**openshift-sriov-network-operator** 命名空间中的所有 pod 将变为 **Running** 状态。

2. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

3. 创建以下 **SriovNetwork** 对象，然后在 **mlx-rdma-network.yaml** 文件中保存 YAML。

```
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 为 ipam CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

注意

如需 **SriovNetwork** 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 app-netutil 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 **SriovNetworkNodePolicy** 对象：

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

5. 创建以下 **Pod** spec，然后在 **mlx-rdma-pod.yaml** 文件中保存 YAML。

```
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. 指定 **target_namespace**，它与 **SriovNetwork** 对象 **mlx-rdma-network** 创建于的命令空间相同。如果要在其他命名空间中创建 pod，在 **Pod** spec 和 **SriovNetwork** 对象中更改 **target_namespace**。
2. 指定包含应用程序和应用程序使用的 RDMA 库的 RDMA 镜像。
3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。
4. 在 **/mnt/huge** 下将巨页卷挂载到 RDMA pod。巨页卷由 emptyDir 卷类型支持，medium 为**Hugepages**。
5. 指定 CPU 数量。RDMA pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 **static**，并创建带有**有保障的** QoS 的 pod 来实现。
6. 指定巨页大小 **hugepages-1Gi** 或 **hugepages-2Mi** 以及分配给 RDMA pod 的巨页数量。单独配置 **2Mi** 和 **1Gi** 巨页。配置 **1Gi** 巨页需要在节点中添加内核参数。

6. 运行以下命令来创建 RDMA pod:

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

9.8. 在 OPENSTACK 上使用 OVS-DPDK 的集群测试 POD 模板

以下 **testpmd** pod 演示了使用巨页、保留 CPU 和 SR-IOV 端口创建容器。

testpmd pod 示例

```
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) 本例中名为 **dpdk1** 是一个用户创建的 **SriovNetworkNodePolicy** 资源。您可以为您创建的资源替换此名称。

(2) 如果您的性能配置集没有命名为 **cnf-performance profile**，请将该字符串替换为正确的性能配置集名称。

9.9. 其他资源

- 支持的设备
- 创建性能配置集
- 使用性能配置集调整 NIC 队列
- 置备实时和低延迟工作负载
- 安装 SR-IOV Network Operator
- 配置 SR-IOV 网络设备
- 使用 Whereabouts 进行动态 IP 地址分配配置
- 禁用单个 pod 的中断处理
- 配置 SR-IOV 以太网网络附加

第 10 章 使用 POD 级别绑定

在 pod 级别的绑定对于启用需要高可用性和更多吞吐量的 pod 内的工作负载至关重要。使用 pod 级别绑定，您可以在内核模式接口上从多个根 I/O 虚拟化(SR-IOV)虚拟功能接口创建绑定接口。SR-IOV 虚拟功能传递到 pod，并附加到内核驱动程序中。

需要 pod 级别绑定的一个场景是从不同物理功能的多个 SR-IOV 虚拟功能创建绑定接口。可以利用主机上的两个不同物理功能创建绑定接口，以便在 pod 级别上实现高可用性。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

有关创建 SR-IOV 网络、网络策略、网络附加定义和 pod 等任务的指导，请参阅配置 SR-IOV 网络设备。

10.1. 从两个 SR-IOV 接口配置绑定接口

绑定可让多个网络接口聚合到一个逻辑 "bonded" 接口。绑定 Container Network Interface (Bond-CNI) 将绑定功能引入容器中。

Bond-CNI 可使用单根 I/O 虚拟化 (SR-IOV) 虚拟功能创建，并将它们放在容器网络命名空间中。

OpenShift Container Platform 仅支持使用 SR-IOV 虚拟功能的 Bond-CNI。SR-IOV Network Operator 提供了管理虚拟功能所需的 SR-IOV CNI 插件。不支持其他 CNI 或接口类型。

先决条件

- 必须安装 SR-IOV Network Operator，并配置为获取容器中的虚拟功能。
- 要配置 SR-IOV 接口，必须为每个接口创建一个 SR-IOV 网络和策略。
- SR-IOV Network Operator 根据定义的 SR-IOV 网络和策略，为每个 SR-IOV 接口创建一个网络附加定义。
- **linkState** 设置为 SR-IOV 虚拟功能的默认值 **auto**。

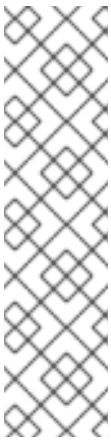
10.1.1. 创建绑定网络附加定义

现在，SR-IOV 虚拟功能可用，您可以创建一个绑定网络附加定义。

```
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 始终设置为 **bond**。
2. **mode** 属性指定绑定模式。



注意

支持的绑定模式有：

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2

对于 **balance-rr** 或 **balance-xor** 模式，您必须为 SR-IOV 虚拟功能将 **trust** 模式设置为 **on**。

3. active-backup 模式的 **failover** 属性是必需的，必须设为 1。
4. **linksInContainer=true** 标志告知 Bond CNI 在容器内找到所需的接口。默认情况下，Bond CNI 会查找主机上的这些接口，该接口无法与 SRIOV 和 Multus 集成。
5. **links** 部分定义将用于创建绑定的接口。默认情况下，Multus 将附加的接口命名为 "net"，再加上一个连续的数字。

10.1.2. 使用绑定接口创建 pod

1. 使用名为 example **podbonding.yaml** 的 YAML 文件创建 pod 来测试设置，其内容类似以下示例：

```
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 注意网络注解：它包含两个 SR-IOV 网络附加，以及一个绑定网络附加。绑定附加使用两个 SR-IOV 接口作为绑定的端口接口。

2. 运行以下命令来应用 yaml:

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

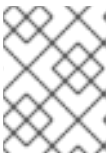
3. 使用以下命令检查 pod 接口：

```
$ 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 绑定接口自动命名为 **net3**。要设置特定的接口名称，请将 **@name** 后缀添加到 pod 的 **k8s.v1.cni.cncf.io/networks** 注解。

2 **net1** 接口基于 SR-IOV 虚拟功能。

3 **net2** 接口基于 SR-IOV 虚拟功能。



注意

如果在 pod 注解中没有配置接口名称，接口名称会自动分配为 **net<n>**，其中 **<n>** 以 **1** 开始。

4. 可选： 如果要为 example **bond0** 设置一个特定的接口名称，请编辑 **k8s.v1.cni.cncf.io/networks** 注解，并将 **bond0** 设为接口名称，如下所示：

```
annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0
```

-

第 11 章 配置硬件卸载 (OFFLOADING)

作为集群管理员，您可以在兼容节点上配置硬件卸载，以提高数据处理性能并减少主机 CPU 的负载。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

11.1. 关于硬件卸载

Open vSwitch 硬件卸载是一种处理网络任务的方法，方法是将它们从 CPU 中分离出来，并将它们卸载到网络接口控制器上的专用处理器。因此，集群可从更快的数据传输速度、CPU 工作负载减少并降低计算成本中受益。

此功能的关键元素是网络接口控制器的现代类，称为 SmartNIC。SmartNIC 是一个网络接口控制器，它可以处理计算密集型网络处理任务。与专用图形卡可提高图形性能的方式相同，T SmartNIC 可改进网络性能。在各个情形中，专用处理器提高了特定类型的处理任务的性能。

在 OpenShift Container Platform 中，您可以为具有兼容 SmartNIC 的裸机节点配置硬件卸载。SR-IOV Network Operator 配置并启用硬件卸载。

硬件卸载并不适用于所有工作负载或应用程序类型。只支持以下两种通信类型：

- pod 到 pod
- Pod 到服务，其中服务是一个由常规 pod 支持的 ClusterIP 服务

在所有情况下，只有在将 pod 和服务分配给具有兼容 SmartNIC 的节点时，硬件卸载才会发生。假设节点上带有硬件卸载的 pod 会尝试与常规节点上的服务进行通信。常规节点上，所有处理都会在内核中进行，因此 pod 到服务通信的整体性能仅限于该常规节点的最大性能。硬件卸载与 DPDK 应用程序不兼容。

在节点上启用硬件卸载，但没有配置 pod 使用，可能会导致 pod 流量的吞吐量性能降低。您无法为 OpenShift Container Platform 管理的 pod 配置硬件卸载。

11.2. 支持的设备

在以下网络接口控制器上支持硬件卸载：

表 11.1. 支持的网络接口控制器

制造商	model	供应商 ID	设备 ID
Mellanox	MT27800 系列 [ConnectX-5]	15b3	1017
Mellanox	MT28880 系列 [ConnectX-5 Ex]	15b3	1019
Mellanox	MT2892 系列 [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 系列 [ConnectX-6 Lx]	15b3	101f
Mellanox	ConnectX-6 NIC 模式中的 MT42822 BlueField-2	15b3	a2d6

11.3. 先决条件

- 集群至少有一个裸机带有网络接口控制器，支持进行硬件卸载。
- 已安装 SR-IOV Network Operator。
- 集群使用 OVN-Kubernetes 网络插件。
- 在 OVN-Kubernetes 网络插件配置中，**gatewayConfig.routingViaHost** 字段被设置为 **false**。

11.4. 将 SR-IOV NETWORK OPERATOR 设置为 SYSTEMD 模式

要支持硬件卸载，您必须首先将 SR-IOV Network Operator 设置为 **systemd** 模式。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建一个 **SriovOperatorConfig** 自定义资源 (CR) 以部署所有 SR-IOV Operator 组件：

   a. 创建名为 **sriovOperatorConfig.yaml** 的文件，其中包含以下 YAML：

```
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** 资源的唯一有效名称是 **default**，它必须位于部署 Operator 的命名空间中。

   (2) 将 SR-IOV Network Operator 设置为 **systemd** 模式仅与 Open vSwitch 硬件卸载相关。

   b. 运行以下命令来创建资源：

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

11.5. 为硬件卸载配置机器配置池

要启用硬件卸载，您可以创建一个专用的机器配置池，并将其配置为使用 SR-IOV Network Operator。

先决条件

- SR-IOV Network Operator 安装并设置为 **systemd** 模式。

流程

1. 为您要使用硬件卸载的机器创建机器配置池。

   a. 创建一个文件，如 **mcp-offloading.yaml**，其内容类似以下示例：

```
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 用于硬件卸载的机器配置池的名称。

   3 此节点角色标签用于添加节点到机器配置池。

   b. 应用机器配置池的配置：

```
$ oc create -f mcp-offloading.yaml
```

2. 将节点添加到机器配置池。使用池的节点角色标签标记每个节点：

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3. 可选：要验证是否创建了新池，请运行以下命令：

```
$ oc get nodes
```

   输出示例

```
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. 将此机器配置池添加到 **SriovNetworkPoolConfig** 自定义资源中：

   a. 创建一个文件，如 **sriov-pool-config.yaml**，其内容类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
```

```
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

1 用于硬件卸载的机器配置池的名称。

b. 应用配置：

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

注意

当您应用由 **SriovNetworkPoolConfig** 对象中指定的配置时，SR-IOV Operator 会排空并重启机器配置池中的节点。

它可能需要几分钟时间来应用配置更改。

11.6. 配置 SR-IOV 网络节点策略

您可以通过创建 SR-IOV 网络节点策略来为节点创建 SR-IOV 网络设备配置。要启用硬件卸载，您必须使用值 **"switchdev"** 定义 **.spec.eSwitchMode** 字段。

以下流程为带有硬件卸载的网络接口控制器创建 SR-IOV 接口。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建一个文件，如 **sriov-node-policy.yaml**，其内容类似以下示例：

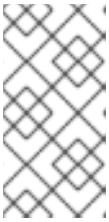
```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy 1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 2
  eSwitchMode: "switchdev" 3
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

```
numVfs: 6
priority: 5
resourceName: mlxnics
```

1. 自定义资源对象的名称。
2. 必需。**vfio-pci** 不支持硬件卸载。
3. 必需。

2. 应用策略的配置：

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



注意

当您应用由 **SriovNetworkPoolConfig** 对象中指定的配置时，SR-IOV Operator 会排空并重启机器配置池中的节点。

它可能需要几分钟时间来应用配置更改。

11.6.1. OpenStack 的 SR-IOV 网络节点策略示例

以下示例描述了在 Red Hat OpenStack Platform (RHOSP) 上使用硬件卸载的网络接口控制器 (NIC) 的 SR-IOV 接口。

用于 RHOSP 上带有硬件卸载的 NIC 的 SR-IOV 接口

```
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. 使用虚拟功能提高网络流量性能

按照以下步骤，为 OVN-Kubernetes 管理端口分配虚拟功能，并提高其网络流量性能。

此流程会导致创建两个池：第一个池具有 OVN-Kubernetes 使用的虚拟功能，第二个由剩余的虚拟功能组成。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 运行以下命令，将 **network.operator.openshift.io/smart-nic** 标签添加到带有 SmartNIC 的每个 worker 节点：

```
$ oc label node <node-name> network.operator.openshift.io/smart-nic=
```

使用 **oc get nodes** 命令获取可用节点的列表。

2. 为管理端口创建一个名为 **sriov-node-mgmt-vf-policy.yaml** 的策略，其内容如下：

```
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 根据您的用例，将此设备替换为适当的网络设备。**pfNames** 值的 **#0-0** 部分保留了 OVN-Kubernetes 使用的一个虚拟功能。

2 此处提供的值是一个示例。使用满足您的要求替换这个值。如需更多信息，请参阅*附加资源*部分中的 *SR-IOV 网络配置对象*。

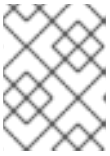
3. 创建名为 **sriov-node-policy.yaml** 的策略，其内容如下：

```
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 根据您的用例，将此设备替换为适当的网络设备。

2 此处提供的值是一个示例。使用 **sriov-node-mgmt-vf-policy.yaml** 文件中指定的值替换这个值。如需更多信息，请参阅*附加资源*部分中的 *SR-IOV 网络配置对象*。



注意

sriov-node-mgmt-vf-policy.yaml 文件具有与 **sriov-node-policy.yaml** 文件不同的 **pfNames** 和 **resourceName** 键的值。

4. 为这两个策略应用配置：

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

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

5. 在集群中创建 Cluster Network Operator (CNO) ConfigMap 以进行管理配置：

   a. 创建名为 **hardware-offload-config.yaml** 的 ConfigMap，其内容如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
   name: hardware-offload-config
   namespace: openshift-network-operator
data:
   mgmt-port-resource-name: openshift.io/mgmtvf
```

   b. 应用 ConfigMap 的配置：

```
$ oc create -f hardware-offload-config.yaml
```

其他资源

- SR-IOV 网络节点配置对象

11.8. 创建网络附加定义

在定义机器配置池和 SR-IOV 网络节点策略后，您可以为您指定的网络接口卡创建网络附加定义。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建一个文件，如 **net-attach-def.yaml**，其内容类似以下示例：

```
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. 网络附加定义的名称。
2. 网络附加定义的命名空间。
3. 这是您在 **SriovNetworkNodePolicy** 对象中指定的 **spec.resourceName** 字段的值。

2. 应用网络附加定义的配置：

```
$ oc create -f net-attach-def.yaml
```

验证

- 运行以下命令检查是否存在新定义：

```
$ oc get net-attach-def -A
```

输出显示了新定义的命名空间、名称和年龄。

11.9. 在 POD 中添加网络附加定义

创建机器配置池后，**SriovNetworkPoolConfig** 和 **SriovNetworkNodePolicy** 自定义资源以及网络附加定义后，您可以通过在 pod 规格中添加网络附加定义来将这些配置应用到 pod。

流程

- 在 pod 规格中，添加 **.metadata.annotations.k8s.v1.cni.cncf.io/networks** 字段，并为硬件卸载指定您创建的网络附加定义：

```
....
metadata:
  annotations:
```

```
v1.multus-cni.io/default-network: net-attach-def/net-attach-def (1)
```

(1) 该值必须是您为硬件卸载而创建的网络附加定义的名称和命名空间。

第 12 章 将 BLUEFIELD-2 从 DPU 切换到 NIC

您可以将 Bluefield-2 网络设备从数据处理单元 (DPU) 模式切换到网络接口控制器 (NIC) 模式。

在执行以下文档中的任何任务前，请确保 安装了 SR-IOV Network Operator。

12.1. 将 BLUEFIELD-2 从 DPU 模式切换到 NIC 模式

使用以下步骤将 Bluefield-2 从数据处理单元 (DPU) 模式切换到网络接口控制器 (NIC) 模式。

重要

目前，只支持将 Bluefield-2 从 DPU 切换到 NIC 模式。不支持从 NIC 模式切换到 DPU 模式。

先决条件

- 已安装 SR-IOV Network Operator。如需更多信息，请参阅"安装 SR-IOV Network Operator"。
- 您已将 Bluefield-2 更新至最新的固件。如需更多信息，请参阅 NVIDIA BlueField-2 的固件。

流程

1. 输入以下命令为每个计算节点添加以下标签。您必须为每个计算节点运行命令。

```
$ oc label node <node_name> node-role.kubernetes.io/sriov=
```

其中：

node_name

引用计算节点的名称。

1. 为 SR-IOV Network Operator 创建机器配置池，例如：

```
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. 将以下 **machineconfig.yaml** 文件应用到计算节点：

```
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
QgL2JpbmRhdGEvc2NyaXB0cy9iZjItc3dpdGNoLW1vZGUuc2ggliRAIgo=
        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) 可选：可以选择性地指定特定卡的 PCI 地址，如 **ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'**。默认情况下会选择第一个设备。如果有多个设备，您必须指定要使用的 PCI 地址。在将 Bluefield-2 从 DPU 模式切换到 NIC 模式的所有节点上，PCI 地址必须相同。

3. 等待计算节点重启。重启后，计算节点上的 Bluefield-2 网络设备切换到 NIC 模式。
4. 可选：您可能需要重启主机硬件，因为最新的 Bluefield-2 固件版本需要硬件重启来切换到 NIC 模式。

其他资源

- 安装 SR-IOV Network Operator