OpenShift Container Platform 4.18

Hardware networks

Configuring hardware-specific networking features in OpenShift Container Platform

Last Updated: 2026-04-01

OpenShift Container Platform 4.18 Hardware networks

Configuring hardware-specific networking features in OpenShift Container Platform

Legal Notice

Abstract

This document covers configuring Single Root I/O Virtualization (SR-IOV) and other hardware-specific network optimizations in OpenShift Container Platform.

Table of Contents

CHAPTER 1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS

The Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device with multiple pods.

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster by using the SR-IOV Operator.

SR-IOV can segment a compliant network device, recognized on the host node as a physical function (PF), into multiple virtual functions (VFs). The VF is used like any other network device. The SR-IOV network device driver for the device determines how the VF is exposed in the container:

- **netdevice** driver: A regular kernel network device in the **netns** of the container
- **vfio-pci** driver: A character device mounted in the container

You can use SR-IOV network devices with additional networks on your OpenShift Container Platform cluster installed on bare metal or Red Hat OpenStack Platform (RHOSP) infrastructure for applications that require high bandwidth or low latency.

The SR-IOV Network Operator is supported on the following platforms:

- Bare metal
- Red Hat OpenStack Platform (RHOSP)

NOTE

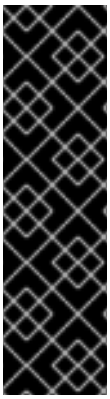
For a list of devices, such as network interface controllers (NICs) that OpenShift Container Platform supports, see Red Hat certified hardware on the Red Hat Ecosystem Catalog. The following example, finds the Intel X710 network adapter:

1. From the Red Hat certified hardware webpage, click **Explore** from the **Components** tile.
2. From the **Provider** drop-down menu, click the **Intel Corporation** checkbox.
3. From the **Platform** drop-down menu, click the **Red Hat OpenShift Container Platform** checkbox.
4. Find the **Intel® Ethernet Server Adapter X710** network adapter from the list and then click on its tile. A new webpage opens that shows information for the network adapter.

You can configure multi-network policies for SR-IOV networks. The support for this is technology preview and SR-IOV additional networks are only supported with kernel NICs. They are not supported for Data Plane Development Kit (DPDK) applications.

NOTE

Creating multi-network policies on SR-IOV networks might not deliver the same performance to applications compared to SR-IOV networks without a multi-network policy configured.



IMPORTANT

Multi-network policies for SR-IOV network is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can enable SR-IOV on a node by using the following command:

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

1.1. ADDITIONAL RESOURCES

- Installing the SR-IOV Network Operator

1.2. COMPONENTS THAT MANAGE SR-IOV NETWORK DEVICES

The SR-IOV Network Operator creates and manages the components of the SR-IOV stack. The Operator performs the following functions:

- Orchestrates discovery and management of SR-IOV network devices
- Generates **NetworkAttachmentDefinition** custom resources for the SR-IOV Container Network Interface (CNI)
- Creates and updates the configuration of the SR-IOV network device plugin
- Creates node specific **SriovNetworkNodeState** custom resources
- Updates the **spec.interfaces** field in each **SriovNetworkNodeState** custom resource

The Operator provisions the following components:

SR-IOV network configuration daemon

A daemon set that is deployed on worker nodes when the SR-IOV Network Operator starts. The daemon is responsible for discovering and initializing SR-IOV network devices in the cluster.

SR-IOV Network Operator webhook

A dynamic admission controller webhook that validates the Operator custom resource and sets appropriate default values for unset fields.

SR-IOV Network resources injector

A dynamic admission controller webhook that provides functionality for patching Kubernetes pod specifications with requests and limits for custom network resources such as SR-IOV VFs. The SR-IOV network resources injector adds the **resource** field to only the first container in a pod automatically.

SR-IOV network device plugin

A device plugin that discovers, advertises, and allocates SR-IOV network virtual function (VF) resources. Device plugins are used in Kubernetes to enable the use of limited resources, typically in

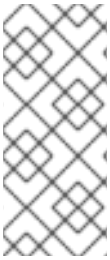
physical devices. Device plugins give the Kubernetes scheduler awareness of resource availability, so that the scheduler can schedule pods on nodes with sufficient resources.

SR-IOV CNI plugin

A CNI plugin that attaches VF interfaces allocated from the SR-IOV network device plugin directly into a pod.

SR-IOV InfiniBand CNI plugin

A CNI plugin that attaches InfiniBand (IB) VF interfaces allocated from the SR-IOV network device plugin directly into a pod.



NOTE

The SR-IOV Network resources injector and SR-IOV Network Operator webhook are enabled by default and can be disabled by editing the **default SriovOperatorConfig** CR. Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices.

1.3. ADDITIONAL RESOURCES

- Configuring multi-network policy

1.4. NEXT STEPS

- Configuring the SR-IOV Network Operator
- Configuring an SR-IOV network device
- If you use OpenShift Virtualization: Connecting a virtual machine to an SR-IOV network
- Configuring an SR-IOV network attachment
- Ethernet network attachement: Adding a pod to an SR-IOV additional network
- InfiniBand network attachement: Adding a pod to an SR-IOV additional network

CHAPTER 2. CONFIGURING AN SR-IOV NETWORK DEVICE

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

2.1. SR-IOV NETWORK NODE CONFIGURATION OBJECT

You specify the SR-IOV network device configuration for a node by creating an SR-IOV network node policy. The API object for the policy is part of the **sriovnetwork.openshift.io** API group.

The following YAML describes an SR-IOV network node policy:

```
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. The name for the custom resource object.
2. The namespace where the SR-IOV Network Operator is installed.
3. The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

   When specifying a name, be sure to use the accepted syntax expression **^[a-zA-Z0-9_]+$** in the **resourceName**.

4.

The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are

IMPORTANT

The SR-IOV Network Operator applies node network configuration policies to nodes in sequence. Before applying node network configuration policies, the SR-IOV Network Operator checks if the machine config pool (MCP) for a node is in an unhealthy state such as **Degraded** or **Updating**. If a node is in an unhealthy MCP, the process of applying node network configuration policies to all targeted nodes in the cluster pauses until the MCP returns to a healthy state.

To avoid a node in an unhealthy MCP from blocking the application of node network configuration policies to other nodes, including nodes in other MCPs, you must create a separate node network configuration policy for each MCP.

5 Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.

6 Optional: The maximum transmission unit (MTU) of the physical function and all its virtual functions. The maximum MTU value can vary for different network interface controller (NIC) models.

IMPORTANT

If you want to create virtual function on the default network interface, ensure that the MTU is set to a value that matches the cluster MTU.

If you want to modify the MTU of a single virtual function while the function is assigned to a pod, leave the MTU value blank in the SR-IOV network node policy. Otherwise, the SR-IOV Network Operator reverts the MTU of the virtual function to the MTU value defined in the SR-IOV network node policy, which might trigger a node drain.

7 Optional: Set **needVhostNet** to **true** to mount the **/dev/vhost-net** device in the pod. Use the mounted **/dev/vhost-net** device with Data Plane Development Kit (DPDK) to forward traffic to the kernel network stack.

8 The number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.

9 The **externallyManaged** field indicates whether the SR-IOV Network Operator manages all, or only a subset of virtual functions (VFs). With the value set to **false** the SR-IOV Network Operator manages and configures all VFs on the PF.

NOTE

When **externallyManaged** is set to **true**, you must manually create the Virtual Functions (VFs) on the physical function (PF) before applying the **SriovNetworkNodePolicy** resource. If the VFs are not pre-created, the SR-IOV Network Operator's webhook will block the policy request.

When **externallyManaged** is set to **false**, the SR-IOV Network Operator automatically creates and manages the VFs, including resetting them if necessary.

To use VFs on the host system, you must create them through NMState, and set **externallyManaged** to **true**. In this mode, the SR-IOV Network Operator does not modify the PF or the manually managed VFs, except for those explicitly defined in the **nicSelector** field of your policy. However, the SR-IOV Network Operator continues to manage VFs that are used as pod secondary interfaces.

10 The NIC selector identifies the device to which this resource applies. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally.

If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

11 Optional: The vendor hexadecimal vendor identifier of the SR-IOV network device. The only allowed values are **8086** (Intel) and **15b3** (Mellanox).

12 Optional: The device hexadecimal device identifier of the SR-IOV network device. For example, **101b** is the device ID for a Mellanox ConnectX-6 device.

13 Optional: An array of one or more physical function (PF) names the resource must apply to.

14 Optional: An array of one or more PCI bus addresses the resource must apply to. For example **0000:02:00.1**.

15 Optional: The platform-specific network filter. The only supported platform is Red Hat OpenStack Platform (RHOSP). Acceptable values use the following format: **openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**. Replace **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** with the value from the **/var/config/openstack/latest/network_data.json** metadata file. This filter ensures that VFs are associated with a specific OpenStack network. The operator uses this filter to map the VFs to the appropriate network based on metadata provided by the OpenStack platform.

16 Optional: The driver to configure for the VFs created from this resource. The only allowed values are **netdevice** and **vfio-pci**. The default value is **netdevice**.

For a Mellanox NIC to work in DPDK mode on bare metal nodes, use the **netdevice** driver type and set **isRdma** to **true**.

17 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**.

If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode.

Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

NOTE

You cannot set the **isRdma** parameter to **true** for intel NICs.

18 Optional: The link type for the VFs. The default value is **eth** for Ethernet. Change this value to 'ib' for InfiniBand.

When **linkType** is set to **ib**, **isRdma** is automatically set to **true** by the SR-IOV Network Operator webhook. When **linkType** is set to **ib**, **deviceType** should not be set to **vfio-pci**.

Do not set linkType to **eth** for SriovNetworkNodePolicy, because this can lead to an incorrect number of available devices reported by the device plugin.

19 Optional: To enable hardware offloading, you must set the **eSwitchMode** field to **"switchdev"**. For more information about hardware offloading, see "Configuring hardware offloading".

20 Optional: To exclude advertising an SR-IOV network resource's NUMA node to the Topology Manager, set the value to **true**. The default value is **false**.

2.1.1. SR-IOV network node configuration examples

The following example describes the configuration for an InfiniBand device:

Example configuration for an InfiniBand device

```
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
# ...
```

The following example describes the configuration for an SR-IOV network device in a RHOSP virtual machine:

Example configuration for an SR-IOV device in a virtual machine

```
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)
# ...
```

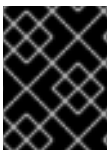
(1) When configuring the node network policy for a virtual machine, the **numVfs** parameter is always set to **1**.

(2) When the virtual machine is deployed on RHOSP, the **netFilter** parameter must refer to a network ID. Valid values for **netFilter** are available from an **SriovNetworkNodeState** object.

2.1.2. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator searches your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a SriovNetworkNodeState custom resource (CR) for each worker node that provides a compatible SR-IOV network device.

The CR is assigned the same name as the worker node. The **status.interfaces** list provides information about the network devices on a node.



IMPORTANT

Do not modify a **SriovNetworkNodeState** object. The Operator creates and manages these resources automatically.

2.1.2.1. Example SriovNetworkNodeState object

The following YAML is an example of a **SriovNetworkNodeState** object created by the SR-IOV Network Operator:

An SriovNetworkNodeState object

```
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
```

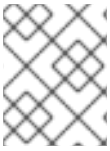
```
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    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 The value of the **name** field is the same as the name of the worker node.

2 The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

2.1.3. Configuring the SR-IOV Network Operator on Mellanox cards when Secure Boot is enabled

The SR-IOV Network Operator supports an option to skip the firmware configuration for Mellanox devices. This option allows you to create virtual functions by using the SR-IOV Network Operator when the system has secure boot enabled. You must manually configure and allocate the number of virtual functions in the firmware before switching the system to secure boot.



NOTE

The number of virtual functions in the firmware is the maximum number of virtual functions that you can request in the policy.

Procedure

1. Configure the virtual functions (VFs) by running the following command when the system is without a secure boot when using the sriov-config daemon:

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

1. The **SRIOV_EN** environment variable enables the SR-IOV Network Operator support on the Mellanox card.
2. The **NUM_OF_VFS** environment variable specifies the number of virtual functions to enable in the firmware.

2. Configure the SR-IOV Network Operator by disabling the Mellanox plugin. See the following **SriovOperatorConfig** example configuration:

```
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. Reboot the system to enable the virtual functions and the configuration settings.
4. Check the virtual functions (VFs) after rebooting the system by running the following command:

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

Example output

```
- 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. The **totalvfs** value is the same number used in the **mstconfig** command earlier in the procedure.

5. Enable secure boot to prevent unauthorized operating systems and malicious software from loading during the device's boot process.

   a. Enable secure boot by using the BIOS (Basic Input/Output System) to set values for the following parameters:

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

   b. Reboot the system.

2.1.4. Virtual function (VF) partitioning for SR-IOV devices

In some cases, you might want to split virtual functions (VFs) from the same physical function (PF) into many resource pools. For example, you might want some of the VFs to load with the default driver and the remaining VFs load with the **vfio-pci** driver.

For example, the following YAML shows the selector for an interface named **netpf0** with VF **2** through **7**:

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

where:

netpf0
: The name of the PF interface name.

2
: The first VF index (0-based) that gets included in the range.

7
: The last VF index (0-based) that gets included in the range.

You can select VFs from the same PF by using different policy CRs provided that you meet the following requirements:

- The **numVfs** value must be similar for policies that select the same PF.
- The VF index must be in the range of **0** to **<numVfs>-1**. For example, if you have a policy with **numVfs** set to **8**, then the **<first_vf>** value must not be smaller than **0**, and the **<last_vf>** must not be larger than **7**.
- The VFs ranges in different policies must not overlap.
- The **<first_vf>** must not be larger than the **<last_vf>**.

The following example illustrates NIC partitioning for an SR-IOV device.

The policy **policy-net-1** defines a resource pool **net-1** that includes the VF **0** of PF **netpf0** with the default VF driver. The policy **policy-net-1-dpdk** defines a resource pool **net-1-dpdk** that includes the VF **8** to **15** of PF **netpf0** with the **vfio** VF driver.

Policy **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
```

Policy **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
```

Verifying that the interface is successfully partitioned

- Confirm that the interface partitioned to virtual functions (VFs) for the SR-IOV device by running the following command.

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

1 Replace **<interface>** with the interface that you specified when partitioning to VFs for the SR-IOV device, for example, **ens3f1**.

Example output

```
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. A test pod template for clusters that use SR-IOV on OpenStack

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

An example testpmd pod

```
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 This example assumes that the name of the performance profile is **cnf-performance profile**.

2.1.6. A test pod template for clusters that use OVS hardware offloading on OpenStack

The following **testpmd** pod demonstrates Open vSwitch (OVS) hardware offloading on Red Hat OpenStack Platform (RHOSP).

An example **testpmd** pod

```
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 If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

2.1.7. Huge pages resource injection for Downward API

When a pod specification includes a resource request or limit for huge pages, the Network Resources Injector automatically adds Downward API fields to the pod specification to provide the huge pages information to the container.

The Network Resources Injector adds a volume that is named **podnetinfo** and is mounted at **/etc/podnetinfo** for each container in the pod. The volume uses the Downward API and includes a file for huge pages requests and limits. The file naming convention is as follows:

- **/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>**

The paths specified in the previous list are compatible with the **app-netutil** library. By default, the library is configured to search for resource information in the **/etc/podnetinfo** directory. If you choose to specify the Downward API path items yourself manually, the **app-netutil** library searches for the following paths in addition to the paths in the previous list.

- **/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**

As with the paths that the Network Resources Injector can create, the paths in the preceding list can optionally end with a **_<container-name>** suffix.

2.2. CONFIGURING SR-IOV NETWORK DEVICES

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).

NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** CR, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. Reboot only happens in the following cases:

- With Mellanox NICs (**mlx5** driver) a node reboot happens every time the number of virtual functions (VFs) increase on a physical function (PF).
- With Intel NICs, a reboot only happens if the kernel parameters do not include **intel_iommu=on** and **iommu=pt**.

It might take several minutes for a configuration change to apply.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.
- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.
- You have not selected any control plane nodes for SR-IOV network device configuration.

Procedure

1. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the **<name>-sriov-node-network.yaml** file. Replace **<name>** with the name for this configuration.
2. Optional: Label the SR-IOV capable cluster nodes with **SriovNetworkNodePolicy.Spec.NodeSelector** if they are not already labeled. For more information about labeling nodes, see "Understanding how to update labels on nodes".
3. Create the **SriovNetworkNodePolicy** object. When running the following command, replace **<name>** with the name for this configuration:

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

   After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.
4. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

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

Additional resources

- Understanding how to update labels on nodes

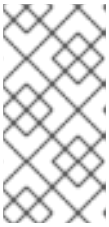
2.3. CREATING A NON-UNIFORM MEMORY ACCESS (NUMA) ALIGNED SR-IOV POD

You can create a NUMA aligned SR-IOV pod by restricting SR-IOV and the CPU resources allocated from the same NUMA node with **restricted** or **single-numa-node** Topology Manager policies.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have configured the CPU Manager policy to **static**. For more information on CPU Manager, see the "Additional resources" section.

- You have configured the Topology Manager policy to **single-numa-node**.



NOTE

When **single-numa-node** is unable to satisfy the request, you can configure the Topology Manager policy to **restricted**. For more flexible SR-IOV network resource scheduling, see *Excluding SR-IOV network topology during NUMA-aware scheduling* in the *Additional resources* section.

Procedure

1. Create the following SR-IOV pod spec, and then save the YAML in the **<name>-sriov-pod.yaml** file. Replace **<name>** with a name for this pod.
   The following example shows an SR-IOV pod spec:

```
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. Replace **<name>** with the name of the SR-IOV network attachment definition CR.
2. Replace **<image>** with the name of the **sample-pod** image.
3. To create the SR-IOV pod with guaranteed QoS, set **memory limits** equal to **memory requests**.
4. To create the SR-IOV pod with guaranteed QoS, set **cpu limits** equals to **cpu requests**.

2. Create the sample SR-IOV pod by running the following command:

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

1. Replace **<filename>** with the name of the file you created in the previous step.

3. Confirm that the **sample-pod** is configured with guaranteed QoS.

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

4. Confirm that the **sample-pod** is allocated with exclusive CPUs.

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

5. Confirm that the SR-IOV device and CPUs that are allocated for the **sample-pod** are on the same NUMA node.

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

2.4. EXCLUDE THE SR-IOV NETWORK TOPOLOGY FOR NUMA-AWARE SCHEDULING

You can exclude advertising the Non-Uniform Memory Access (NUMA) node for the SR-IOV network to the Topology Manager for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

In some scenarios, it is a priority to maximize CPU and memory resources for a pod on a single NUMA node. By not providing a hint to the Topology Manager about the NUMA node for the pod's SR-IOV network resource, the Topology Manager can deploy the SR-IOV network resource and the pod CPU and memory resources to different NUMA nodes. This can add to network latency because of the data transfer between NUMA nodes. However, it is acceptable in scenarios when workloads require optimal CPU and memory performance.

For example, consider a compute node, **compute-1**, that features two NUMA nodes: **numa0** and **numa1**. The SR-IOV-enabled NIC is present on **numa0**. The CPUs available for pod scheduling are present on **numa1** only. By setting the **excludeTopology** specification to **true**, the Topology Manager can assign CPU and memory resources for the pod to **numa1** and can assign the SR-IOV network resource for the same pod to **numa0**. This is only possible when you set the **excludeTopology** specification to **true**. Otherwise, the Topology Manager attempts to place all resources on the same NUMA node.

2.5. TROUBLESHOOTING SR-IOV CONFIGURATION

After following the procedure to configure an SR-IOV network device, the following sections address some error conditions.

Procedure

- To display the state of nodes, run the following command:

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

where:

<node_name>

Specifies the name of a node with an SR-IOV network device.

If the output from the command indicates "cannot allocate memory", check the following items:

- Confirm that global SR-IOV settings are enabled in the BIOS for the node.
- Confirm that VT-d is enabled in the BIOS for the node.

Additional resources

- Using CPU Manager

2.6. NEXT STEPS

- Configuring an SR-IOV network attachment

CHAPTER 3. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT

You can configure an Ethernet network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

3.1. ETHERNET DEVICE CONFIGURATION OBJECT

You can configure an Ethernet network device by defining an **SriovNetwork** object.

The following YAML describes an **SriovNetwork** object:

```
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. A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
2. The namespace where the SR-IOV Network Operator is installed.
3. The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
4. The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.
5. Optional: Specifies the VLAN ID to assign to an additional network. The default value of **0** means that an additional network has no VLAN ID tag. Supported VLAN ID values range from **1** to **4094**.
6. Optional: The spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.

IMPORTANT

You must enclose the value you specify in quotes or the object is rejected by the SR-IOV Network Operator.

7 A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

8 Optional: The link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.

9 Optional: A maximum transmission rate, in Mbps, for the VF.

10 Optional: A minimum transmission rate, in Mbps, for the VF. This value must be less than or equal to the maximum transmission rate.

NOTE

Intel NICs do not support the **minTxRate** parameter. For more information, see BZ#1772847.

11 Optional: An IEEE 802.1p priority level for the VF. The default value is **0**.

12 Optional: The trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.

IMPORTANT

You must enclose the value that you specify in quotes, or the SR-IOV Network Operator rejects the object.

13 Optional: The capabilities to configure for this additional network. You can specify **'{ "ips": true }'** to enable IP address support or **'{ "mac": true }'** to enable MAC address support.

3.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically

You can dynamically assign dual-stack IP addresses to a secondary network so that pods can communicate over both IPv4 and IPv6 addresses.

You can configure the following IP address assignment types in the **ipRanges** parameter:

- IPv4 addresses
- IPv6 addresses
- multiple IP address assignment

Procedure

1. Set **type** to **whereabouts**.
2. Use **ipRanges** to allocate IP addresses as shown in the following example:

```
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. Attach the secondary network to a pod. For more information, see "Adding a pod to a secondary network".

Verification

- Verify that all IP addresses got assigned to the network interfaces within the network namespace of a pod by entering the following command:

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

where:

<podname>

The name of the pod.

3.1.2. Configuration of IP address assignment for a network attachment

For secondary networks, you can assign IP addresses by using an IP Address Management (IPAM) CNI plugin, which supports various assignment methods, including Dynamic Host Configuration Protocol (DHCP) and static assignment.

The DHCP IPAM CNI plugin responsible for dynamic assignment of IP addresses operates with two distinct components:

- CNI Plugin: Responsible for integrating with the Kubernetes networking stack to request and release IP addresses.
- DHCP IPAM CNI Daemon: A listener for DHCP events that coordinates with existing DHCP servers in the environment to handle IP address assignment requests. This daemon is not a DHCP server itself.

For networks requiring **type: dhcp** in their IPAM configuration, ensure the DHCP server meets the following conditions:

- A DHCP server is available and running in the environment.
- The DHCP server is external to the cluster and you expect the server to form part of the existing network infrastructure for the customer.
- The DHCP server is appropriately configured to serve IP addresses to the nodes.

In cases where a DHCP server is unavailable in the environment, consider using the Whereabouts IPAM CNI plugin. The Whereabouts CNI provides similar IP address management capabilities without the need for an external DHCP server.

NOTE

Use the Whereabouts CNI plugin when no external DHCP server exists or where static IP address management is preferred. The Whereabouts plugin includes a reconciler daemon to manage stale IP address allocations.

Ensure the periodic renewal of a DHCP lease throughout the lifetime of a container by including a separate daemon, the DHCP IPAM CNI Daemon. To deploy the DHCP IPAM CNI daemon, change the Cluster Network Operator (CNO) configuration to trigger the deployment of this daemon as part of the secondary network setup.

3.1.2.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

Table 3.1. ipam static configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **static** is required.
addresses	**array**	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
routes	**array**	An array of objects specifying routes to configure inside the pod.
dns	**array**	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 3.2. ipam.addresses[] array

Field	Type	Description
address	**string**	An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, the secondary network gets assigned an IP address of **10.10.21.10** and the netmask of **255.255.255.0**.
gateway	**string**	The default gateway to route egress network traffic to.

Table 3.3. **ipam.routes[]** array

Field	Type	Description
dst	**string**	The IP address range in CIDR format, such as **192.168.17.0/24** or **0.0.0.0/0** for the default route.
gw	**string**	The gateway that routes network traffic.

Table 3.4. **ipam.dns** object

Field	Type	Description
nameservers	**array**	An array of one or more IP addresses where DNS queries get sent.
domain	**array**	The default domain to append to a hostname. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.
search	**array**	An array of domain names to append to an unqualified hostname, such as **example-host**, during a DNS lookup query.

Static IP address assignment configuration example

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

3.1.2.2. Dynamic IP address (DHCP) assignment configuration

A pod obtains its original DHCP lease when the pod gets created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

IMPORTANT

For an Ethernet network attachment, the SR-IOV Network Operator does not create a DHCP server deployment; the Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

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

where:

type

Specifies dynamic IP address assignment for the cluster.

The following table describes the configuration parameters for dynamic IP address address assignment with DHCP.

Table 3.5. ipam DHCP configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **dhcp** is required.

The following JSON example describes the configuration p for dynamic IP address address assignment with DHCP.

Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

3.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to a secondary network without the use of a DHCP server.

The Whereabouts CNI plugin also supports overlapping IP address ranges and configuration of the same CIDR range multiple times within separate **NetworkAttachmentDefinition** CRDs. This provides greater flexibility and management capabilities in multi-tenant environments.

3.1.2.3.1. Dynamic IP address configuration objects

The following table describes the configuration objects for dynamic IP address assignment with Whereabouts:

Table 3.6. ipam whereabouts configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **whereabouts** is required.
range	**string**	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
exclude	**array**	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.
network_name	**string**	Optional: Helps ensure that each group or domain of pods gets its own set of IP addresses, even if they share the same range of IP addresses. Setting this field is important for keeping networks separate and organized, notably in multi-tenant environments.

3.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts

The following example shows a dynamic address assignment configuration that uses Whereabouts:

Whereabouts dynamic IP address assignment

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

3.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges

The following example shows a dynamic IP address assignment that uses overlapping IP address ranges for multi-tenant networks.

NetworkAttachmentDefinition 1

```
{
```

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

(1) Optional. If set, must match the **network_name** of **NetworkAttachmentDefinition 2**.

NetworkAttachmentDefinition 2

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

(1) Optional. If set, must match the **network_name** of **NetworkAttachmentDefinition 1**.

3.2. CONFIGURING SR-IOV ADDITIONAL NETWORK

You can configure an additional network that uses SR-IOV hardware by creating an **SriovNetwork** object. When you create an **SriovNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.

NOTE

Do not modify or delete an **SriovNetwork** object if it is attached to any pods in a **running** state.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a **SriovNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

```
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. To create the object, enter the following command:

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

where:

<name>

Specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the **networkNamespace** value you specified in the **SriovNetwork** object.

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

3.3. ASSIGNING AN SR-IOV NETWORK TO A VRF

As a cluster administrator, you can assign an SR-IOV network interface to your VRF domain by using the CNI VRF plugin.

To do this, add the VRF configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.

NOTE

Applications that use VRFs need to bind to a specific device. The common usage is to use the **SO_BINDTODEVICE** option for a socket. **SO_BINDTODEVICE** binds the socket to a device that is specified in the passed interface name, for example, **eth1**. To use **SO_BINDTODEVICE**, the application must have **CAP_NET_RAW** capabilities.

Using a VRF through the **ip vrf exec** command is not supported in OpenShift Container Platform pods. To use VRF, bind applications directly to the VRF interface.

3.3.1. Creating an additional SR-IOV network attachment with the CNI VRF plugin

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To create an additional SR-IOV network attachment with the CNI virtual routing and forwarding (VRF) plugin, perform the following procedure.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **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"
    }
```

where:

metaPlugins.type

Set the **type** parameter to **vrf**.

metaPlugins.vrfname

Specify a name for the VRF in the **vrfname** parameter. An interface gets assigned to the VRF. If you do not specify a name for the VRF in a pod, the SR-IOV Network Operator automatically generates a name for the VRF.

2. Create the **SriovNetwork** resource:

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

Verification

1. Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command. The expected output shows the name of the NAD CR and the creation age in minutes.

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

- **<namespace>**: Replace **<namespace>** with the namespace that you specified when configuring the network attachment, for example, **additional-sriov-network-1**.

NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

2. To verify that the VRF CNI is correctly configured and that the additional SR-IOV network attachment is attached, do the following:

   a. Create an SR-IOV network that uses the VRF CNI.

   b. Assign the network to a pod.

   c. Verify that the pod network attachment connects to the SR-IOV additional network. Ensure that you remote shell login into the pod and run the following command. The expected output shows the name of the VRF interface and its unique ID in the routing table.

```
$ ip vrf show
```

   d. Confirm that the VRF interface is **master** for the secondary interface by running the following command. Example output shows **5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red state UP mode**.

```
$ ip link
```

3.4. RUNTIME CONFIGURATION FOR AN ETHERNET-BASED SR-IOV ATTACHMENT

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

Example runtime configuration for an Ethernet-based SR-IOV network attachment

```
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"]
```

where:

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

The name of the SR-IOV network attachment definition CR. Example value is **ibl**.

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

Optional parameter. The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the **SriovNetwork** object. Example value is **c2:11:22:33:44:55:66:77**.

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

Optional parameter. IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object. Example value is **192.168.10.1/24", "2001::1/64**.

3.5. ADDING A POD TO A SECONDARY NETWORK

To enable a pod to use additional network interfaces in OpenShift Container Platform, you can attach the pod to a secondary network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created, a secondary network is attached to the pod. However, if a pod already exists, you cannot attach a secondary network to it.

The pod must be in the same namespace as the secondary network.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

   a. To attach a secondary network without any customization, add an annotation with the following format:

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

   where:

   k8s.v1.cni.cncf.io/networks

   Specifies the name of the secondary network to associate with the pod. To specify more than one secondary network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same secondary network multiple times, that pod will have multiple network interfaces attached to that network.

   b. To attach a secondary network with customizations, add an annotation with the following format:

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

   where:

   <network>

   Specifies the name of the secondary network defined by a **NetworkAttachmentDefinition** object.

   <namespace>

   Specifies the namespace where the **NetworkAttachmentDefinition** object is defined.

   <default-route>

   Optional parameter. Specifies an override for the default route, such as **192.168.17.1**.

2. Create the pod by entering the following command.

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

   Replace **<name>** with the name of the pod.

3. Optional: Confirm that the annotation exists in the **pod** CR by entering the following command. Replace **<name>** with the name of the pod.

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

   In the following example, the **example-pod** pod is attached to the **net1** secondary network:

```
$ 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:
  ...
```

where:

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

Specifies a JSON array of objects. Each object describes the status of a secondary network attached to the pod. The annotation value is stored as a plain text value.

3.5.1. Exposing MTU for vfio-pci SR-IOV devices to pod

After adding a pod to an additional network, you can check that the MTU is available for the SR-IOV network.

Procedure

1. Check that the pod annotation includes MTU by running the following command:

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

The following example shows the sample output:

```
"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. Verify that the MTU is available in **/etc/podnetinfo/** inside the pod by running the following command:

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

The following example shows the sample output:

```
k8s.v1.cni.cncf.io/network-status="[{
    \"name\": \"ovn-kubernetes\",
    \"interface\": \"eth0\",
    \"ips\": [
        \"10.131.0.67\"
    ],
    \"mac\": \"0a:58:0a:83:00:43\",
    \"default\": true,
    \"dns\": {}
    },{
    \"name\": \"sriov-tests/sriov-nic-1\",
    \"interface\": \"net1\",
    \"ips\": [
        \"192.168.10.1\"
    ],
    \"mac\": \"20:04:0f:f1:88:01\",
    \"mtu\": 1500,
    \"dns\": {},
    \"device-info\": {
        \"type\": \"pci\",
        \"version\": \"1.1.0\",
        \"pci\": {
            \"pci-address\": \"0000:86:01.3\"
        }
    }
    }]"
```

3.6. CONFIGURING PARALLEL NODE DRAINING DURING SR-IOV NETWORK POLICY UPDATES

By default, the SR-IOV Network Operator drains workloads from a node before every policy change. The Operator performs this action, one node at a time, to ensure that the reconfiguration does not impact workloads.

In large clusters, draining nodes sequentially can be time-consuming, taking hours or even days. In time-sensitive environments, you can enable parallel node draining in an **SriovNetworkPoolConfig** custom resource (CR) for faster rollouts of SR-IOV network configurations.

To configure parallel draining, use the **SriovNetworkPoolConfig** CR to create a node pool. You can then add nodes to the pool and define the maximum number of nodes in the pool that the Operator can

drain in parallel. With this approach, you can enable parallel draining for faster reconfiguration while ensuring you still have enough nodes remaining in the pool to handle any running workloads.

NOTE

A node can only belong to one SR-IOV network pool configuration. If a node is not part of a pool, the node gets added to a virtual, default, pool that with a configuration for draining one node at a time only.

The node might restart during the draining process.

The procedure requires that you create SR-IOV resources and then parallel drain the nodes.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the SR-IOV Network Operator.
- Nodes have hardware that support SR-IOV.

Procedure

1. Create a YAML file that defines the **SriovNetworkPoolConfig** resource:

   Example sriov-nw-pool.yaml file

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

   where:

   name

   Specify the name of the **SriovNetworkPoolConfig** object.

   namespace

   Specify namespace where the SR-IOV Network Operator is installed.

   maxUnavailable

   Specify an integer number, or percentage value, for nodes that can be unavailable in the pool during an update. For example, if you have 10 nodes and you set the maximum unavailable to 2, then only 2 nodes can be drained in parallel at any time, leaving 8 nodes for handling workloads.

   nodeSelector

Specify the nodes to add the pool by using the node selector. This example adds all nodes with the **worker** role to the pool.

2. Create the **SriovNetworkPoolConfig** resource by running the following command:

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

3. Create the **sriov-test** namespace by running the following comand:

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

4. Create a YAML file that defines the **SriovNetworkNodePolicy** resource, as demonstrated in the following example YAML file:

```
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. Create the **SriovNetworkNodePolicy** resource by running the following command:

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

6. Create a YAML file that defines the **SriovNetwork** resource:

Example sriov-network.yaml file

```
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. Create the **SriovNetwork** resource by running the following command:

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

8. View the node pool you created by running the following command:

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

Expected output shows the name of the node pool, such as **pool-1**, that includes all the node that have the **worker** role and the age of the node pool in seconds, such as **67s**.

9. Update the number of virtual functions in the **SriovNetworkNodePolicy** resource to trigger workload draining in the cluster:

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

10. Check the draining status on the target cluster by running the following command:

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

Example output

```
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
```

When the draining process completes, the **SYNC STATUS** changes to **Succeeded**, and the **DESIRED SYNC STATE** and **CURRENT SYNC STATE** values return to **IDLE**.

3.7. EXCLUDING THE SR-IOV NETWORK TOPOLOGY FOR NUMA-AWARE SCHEDULING

To exclude advertising the SR-IOV network resource's Non-Uniform Memory Access (NUMA) node to the Topology Manager, you can configure the **excludeTopology** specification in the **SriovNetworkNodePolicy** custom resource. Use this configuration for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have configured the CPU Manager policy to **static**. For more information about CPU Manager, see the *Additional resources* section.
- You have configured the Topology Manager policy to **single-numa-node**.
- You have installed the SR-IOV Network Operator.

Procedure

1. Create the **SriovNetworkNodePolicy** CR:

a. Save the following YAML in the **sriov-network-node-policy.yaml** file, replacing values in the YAML to match your environment:

```
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) The resource name of the SR-IOV network device plugin. This YAML uses a sample **resourceName** value.

(2) Identify the device for the Operator to configure by using the network interface controller (NIC selector).

(3) To exclude advertising the NUMA node for the SR-IOV network resource to the Topology Manager, set the value to **true**. The default value is **false**.

NOTE

If many **SriovNetworkNodePolicy** resources target the same SR-IOV network resource, the **SriovNetworkNodePolicy** resources must have the same value as the **excludeTopology** specification. Otherwise, the conflicting policy is rejected.

b. Create the **SriovNetworkNodePolicy** resource by running the following command. Successful output lists the name of the **SriovNetworkNodePolicy** resource and the **created** status.

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

2. Create the **SriovNetwork** CR:

a. Save the following YAML in the **sriov-network.yaml** file, replacing values in the YAML to match your environment:

```
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. Replace **sriov-numa-0-network** with the name for the SR-IOV network resource.
2. Specify the resource name for the **SriovNetworkNodePolicy** CR from the previous step. This YAML uses a sample **resourceName** value.
3. Enter the namespace for your SR-IOV network resource.
4. Enter the IP address management configuration for the SR-IOV network.

b. Create the **SriovNetwork** resource by running the following command. Successful output lists the name of the **SriovNetwork** resource and the **created** status.

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

3. Create a pod and assign the SR-IOV network resource from the previous step:

a. Save the following YAML in the **sriov-network-pod.yaml** file, replacing values in the YAML to match your environment:

```
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. This is the name of the **SriovNetwork** resource that uses the **SriovNetworkNodePolicy** resource.

b. Create the **Pod** resource by running the following command. The expected output shows the name of the **Pod** resource and the **created** status.

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

Verification

1. Verify the status of the pod by running the following command, replacing **<pod_name>** with the name of the pod:

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

Example output

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

2. Open a debug session with the target pod to verify that the SR-IOV network resources are deployed to a different node than the memory and CPU resources.

   a. Open a debug session with the pod by running the following command, replacing <pod_name> with the target pod name.

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

   b. Set **/host** as the root directory within the debug shell. The debug pod mounts the root file system from the host in **/host** within the pod. By changing the root directory to **/host**, you can run binaries from the host file system:

```
$ chroot /host
```

   c. View information about the CPU allocation by running the following commands:

```
$ lscpu | grep NUMA
```

Example output

```
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
```

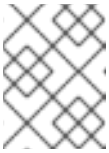
Example output

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

The expected output shows the CPUs (1, 3, 5, and 7) that get allocated to a **NUMA** node, such as **NUMA node1**. The SR-IOV network resource can use the NIC from another **NUMA** node, such as **NUMA node0**. Note that the **ffff** hexadecimal value represents the CPU cores that run a process.

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

Expected output shows the number for the **NUMA** node, such as **0**.



NOTE

If you set the **excludeTopology** specification to **True**, the required resources might exist in the same NUMA node.

3.8. ADDITIONAL RESOURCES

- Configuring an SR-IOV network device
- Using CPU Manager

CHAPTER 4. CONFIGURING AN SR-IOV INFINIBAND NETWORK ATTACHMENT

You can configure an InfiniBand (IB) network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

4.1. INFINIBAND DEVICE CONFIGURATION OBJECT

You can configure an InfiniBand (IB) network device by defining an **SriovIBNetwork** object.

The following YAML describes an **SriovIBNetwork** object:

```
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>
```

where:

name

A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.

namespace

The namespace where the SR-IOV Operator is installed.

resourceName

The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

networkNamespace

The target namespace for the **SriovIBNetwork** object. Only pods in the target namespace can attach to the network device.

ipam

Optional parameter. A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

linkState

Optional parameter. The link state of virtual function (VF). Allowed values are **enable**, **disable** and **auto**.

capabilities

Optional parameter. The capabilities to configure for this network. You can specify **'{ "ips": true }'** to enable IP address support or **'{ "infinibandGUID": true }'** to enable IB Global Unique Identifier (GUID) support.

4.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically

You can dynamically assign dual-stack IP addresses to a secondary network so that pods can communicate over both IPv4 and IPv6 addresses.

You can configure the following IP address assignment types in the **ipRanges** parameter:

- IPv4 addresses
- IPv6 addresses
- multiple IP address assignment

Procedure

1. Set **type** to **whereabouts**.
2. Use **ipRanges** to allocate IP addresses as shown in the following example:

```
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. Attach the secondary network to a pod. For more information, see "Adding a pod to a secondary network".

Verification

- Verify that all IP addresses got assigned to the network interfaces within the network namespace of a pod by entering the following command:

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

where:

<podname>

The name of the pod.

4.1.2. Configuration of IP address assignment for a network attachment

For secondary networks, you can assign IP addresses by using an IP Address Management (IPAM) CNI plugin, which supports various assignment methods, including Dynamic Host Configuration Protocol (DHCP) and static assignment.

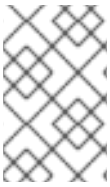
The DHCP IPAM CNI plugin responsible for dynamic assignment of IP addresses operates with two distinct components:

- CNI Plugin: Responsible for integrating with the Kubernetes networking stack to request and release IP addresses.
- DHCP IPAM CNI Daemon: A listener for DHCP events that coordinates with existing DHCP servers in the environment to handle IP address assignment requests. This daemon is not a DHCP server itself.

For networks requiring **type: dhcp** in their IPAM configuration, ensure the DHCP server meets the following conditions:

- A DHCP server is available and running in the environment.
- The DHCP server is external to the cluster and you expect the server to form part of the existing network infrastructure for the customer.
- The DHCP server is appropriately configured to serve IP addresses to the nodes.

In cases where a DHCP server is unavailable in the environment, consider using the Whereabouts IPAM CNI plugin. The Whereabouts CNI provides similar IP address management capabilities without the need for an external DHCP server.



NOTE

Use the Whereabouts CNI plugin when no external DHCP server exists or where static IP address management is preferred. The Whereabouts plugin includes a reconciler daemon to manage stale IP address allocations.

Ensure the periodic renewal of a DHCP lease throughout the lifetime of a container by including a separate daemon, the DHCP IPAM CNI Daemon. To deploy the DHCP IPAM CNI daemon, change the Cluster Network Operator (CNO) configuration to trigger the deployment of this daemon as part of the secondary network setup.

4.1.2.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

Table 4.1. ipam static configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **static** is required.
addresses	**array**	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
routes	**array**	An array of objects specifying routes to configure inside the pod.
dns	**array**	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 4.2. **ipam.addresses[]** array

Field	Type	Description
address	**string**	An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, the secondary network gets assigned an IP address of **10.10.21.10** and the netmask of **255.255.255.0**.
gateway	**string**	The default gateway to route egress network traffic to.

Table 4.3. **ipam.routes[]** array

Field	Type	Description
dst	**string**	The IP address range in CIDR format, such as **192.168.17.0/24** or **0.0.0.0/0** for the default route.
gw	**string**	The gateway that routes network traffic.

Table 4.4. **ipam.dns** object

Field	Type	Description
nameservers	**array**	An array of one or more IP addresses where DNS queries get sent.
domain	**array**	The default domain to append to a hostname. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.
search	**array**	An array of domain names to append to an unqualified hostname, such as **example-host**, during a DNS lookup query.

Static IP address assignment configuration example

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

4.1.2.2. Dynamic IP address (DHCP) assignment configuration

A pod obtains its original DHCP lease when the pod gets created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

IMPORTANT

For an Ethernet network attachment, the SR-IOV Network Operator does not create a DHCP server deployment; the Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

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

where:

type

Specifies dynamic IP address assignment for the cluster.

The following table describes the configuration parameters for dynamic IP address address assignment with DHCP.

Table 4.5. ipam DHCP configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **dhcp** is required.

The following JSON example describes the configuration p for dynamic IP address address assignment with DHCP.

Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

4.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to a secondary network without the use of a DHCP server.

The Whereabouts CNI plugin also supports overlapping IP address ranges and configuration of the same CIDR range multiple times within separate **NetworkAttachmentDefinition** CRDs. This provides greater flexibility and management capabilities in multi-tenant environments.

4.1.2.3.1. Dynamic IP address configuration objects

The following table describes the configuration objects for dynamic IP address assignment with Whereabouts:

Table 4.6. ipam whereabouts configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **whereabouts** is required.
range	**string**	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
exclude	**array**	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.
network_name	**string**	Optional: Helps ensure that each group or domain of pods gets its own set of IP addresses, even if they share the same range of IP addresses. Setting this field is important for keeping networks separate and organized, notably in multi-tenant environments.

4.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts

The following example shows a dynamic address assignment configuration that uses Whereabouts:

Whereabouts dynamic IP address assignment

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

4.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges

The following example shows a dynamic IP address assignment that uses overlapping IP address ranges for multi-tenant networks.

NetworkAttachmentDefinition 1

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

(1) Optional. If set, must match the **network_name** of **NetworkAttachmentDefinition 2**.

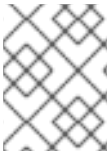
NetworkAttachmentDefinition 2

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

(1) Optional. If set, must match the **network_name** of **NetworkAttachmentDefinition 1**.

4.2. CONFIGURING SR-IOV ADDITIONAL NETWORK

You can configure an additional network that uses SR-IOV hardware by creating an **SriovIBNetwork** object. When you create an **SriovIBNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.



NOTE

Do not modify or delete an **SriovIBNetwork** object if it is attached to any pods in a **running** state.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a **SriovIBNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

```
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. To create the object, enter the following command:

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

where:

<name>

Specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovIBNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the **networkNamespace** value you specified in the **SriovIBNetwork** object.

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

4.3. RUNTIME CONFIGURATION FOR AN INFINIBAND-BASED SR-IOV ATTACHMENT

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

The following JSON describes the runtime configuration options for an InfiniBand-based SR-IOV network attachment.

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

where:

name

The name of the SR-IOV network attachment definition CR.

infiniband-guid

The InfiniBand GUID for the SR-IOV device. To use this feature, you also must specify **{ "infinibandGUID": true }** in the **SriovIBNetwork** object.

ips

The IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovIBNetwork** object.

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

4.4. ADDING A POD TO A SECONDARY NETWORK

To enable a pod to use additional network interfaces in OpenShift Container Platform, you can attach the pod to a secondary network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created, a secondary network is attached to the pod. However, if a pod already exists, you cannot attach a secondary network to it.

The pod must be in the same namespace as the secondary network.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

   a. To attach a secondary network without any customization, add an annotation with the following format:

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

   where:

   k8s.v1.cni.cncf.io/networks

   Specifies the name of the secondary network to associate with the pod. To specify more than one secondary network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same secondary network multiple times, that pod will have multiple network interfaces attached to that network.

   b. To attach a secondary network with customizations, add an annotation with the following format:

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

   where:

   <network>

   Specifies the name of the secondary network defined by a **NetworkAttachmentDefinition** object.

<namespace>

Specifies the namespace where the **NetworkAttachmentDefinition** object is defined.

<default-route>

Optional parameter. Specifies an override for the default route, such as **192.168.17.1**.

2. Create the pod by entering the following command.

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

Replace **<name>** with the name of the pod.

3. Optional: Confirm that the annotation exists in the **pod** CR by entering the following command. Replace **<name>** with the name of the pod.

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

In the following example, the **example-pod** pod is attached to the **net1** secondary network:

```
$ 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:
  ...
```

where:

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

Specifies a JSON array of objects. Each object describes the status of a secondary network attached to the pod. The annotation value is stored as a plain text value.

4.4.1. Exposing MTU for vfio-pci SR-IOV devices to pod

After adding a pod to an additional network, you can check that the MTU is available for the SR-IOV network.

Procedure

1. Check that the pod annotation includes MTU by running the following command:

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

The following example shows the sample output:

```
"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. Verify that the MTU is available in **/etc/podnetinfo/** inside the pod by running the following command:

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

The following example shows the sample output:

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

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

4.5. ADDITIONAL RESOURCES

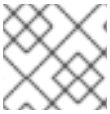
- Configuring an SR-IOV network device
- Using CPU Manager
- Exclude SR-IOV network topology for NUMA-aware scheduling

CHAPTER 5. CONFIGURING AN RDMA SUBSYSTEM FOR SR-IOV

Remote Direct Memory Access (RDMA) allows direct memory access between two systems without involving the operating system of either system. You can configure an RDMA Container Network Interface (CNI) on Single Root I/O Virtualization (SR-IOV) to enable high-performance, low-latency communication between containers. When you combine RDMA with SR-IOV, you provide a mechanism to expose hardware counters of Mellanox Ethernet devices for use inside Data Plane Development Kit (DPDK) applications.

5.1. CONFIGURING SR-IOV RDMA CNI

Configure an RDMA CNI on SR-IOV.



NOTE

This procedure applies only to Mellanox devices.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.

Procedure

1. Create an **SriovNetworkPoolConfig** CR and save it as **sriov-nw-pool.yaml**, as shown in the following example:

   Example SriovNetworkPoolConfig CR

```
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) Set RDMA network namespace mode to **exclusive**.

2. Create the **SriovNetworkPoolConfig** resource by running the following command:

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

3. Create an **SriovNetworkNodePolicy** CR and save it as **sriov-node-policy.yaml**, as shown in the following example:

   Example SriovNetworkNodePolicy CR

```
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) Activate RDMA mode.

4. Create the **SriovNetworkNodePolicy** resource by running the following command:

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

5. Create an **SriovNetwork** CR and save it as **sriov-network.yaml**, as shown in the following example:

   Example SriovNetwork CR

```
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) Create the RDMA plugin.

6. Create the **SriovNetwork** resource by running the following command:

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

Verification

1. Create a **Pod** CR and save it as **sriov-test-pod.yaml**, as shown in the following example:

Example runtime configuration

```
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. Create the test pod by running the following command:

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

3. Log in to the test pod by running the following command:

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

4. Verify that the path to the **hw-counters** directory exists by running the following command:

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

Example output

```
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
```

CHAPTER 6. CONFIGURING INTERFACE-LEVEL NETWORK SYSCTL SETTINGS AND ALL-MULTICAST MODE FOR SR-IOV NETWORKS

As a cluster administrator, you can change interface-level network sysctls and several interface attributes such as promiscuous mode, all-multicast mode, MTU, and MAC address by using the tuning Container Network Interface (CNI) meta plugin for a pod connected to a SR-IOV network device.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

6.1. LABELING NODES WITH AN SR-IOV ENABLED NIC

If you want to enable SR-IOV on only SR-IOV capable nodes there are a couple of ways to do this:

1. Install the Node Feature Discovery (NFD) Operator. NFD detects the presence of SR-IOV enabled NICs and labels the nodes with **node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true**.

2. Examine the **SriovNetworkNodeState** CR for each node. The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the SR-IOV Network Operator on the worker node. Label each node with **feature.node.kubernetes.io/network-sriov.capable: "true"** by using the following command:

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

NOTE

You can label the nodes with whatever name you want.

6.2. SETTING ONE SYSCTL FLAG

You can set interface-level network **sysctl** settings for a pod connected to a SR-IOV network device.

In this example, **net.ipv4.conf.IFNAME.accept_redirects** is set to **1** on the created virtual interfaces.

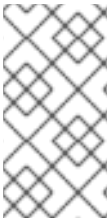
The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

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

6.2.1. Setting one sysctl flag on nodes with SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).



NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain and reboot the nodes.

It can take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

Procedure

1. Create an **SriovNetworkNodePolicy** custom resource (CR). For example, save the following YAML as the file **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) The name for the custom resource object.

(2) The namespace where the SR-IOV Network Operator is installed.

(3) The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

(4) The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.

(5) Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.

(6) The number of the virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.

(7) The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If

you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

8 Optional: An array of one or more physical function (PF) names for the device.

9 Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.

10 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

NOTE

The **vfio-pci** driver type is not supported.

2. Create the **SriovNetworkNodePolicy** object:

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

After applying the configuration update, all the pods in **sriov-network-operator** namespace change to the **Running** status.

3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured. Expected output shows **Succeeded**.

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

6.2.2. Configuring sysctl on a SR-IOV network

You can set interface specific **sysctl** settings on virtual interfaces created by SR-IOV by adding the tuning configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change the interface-level network **net.ipv4.conf.IFNAME.accept_redirects sysctl** settings, create an additional SR-IOV network with the Container Network Interface (CNI) tuning plugin.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **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) A name for the object. The SR-IOV Network Operator creates a NetworkAttachmentDefinition object with same name.

(2) The namespace where the SR-IOV Network Operator is installed.

(3) The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

(4) The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

(5) A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

(6) Optional: Set capabilities for the additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

(7) Optional: The metaPlugins parameter is used to add additional capabilities to the device. In this use case set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the **sysctl** field.

2. Create the **SriovNetwork** resource:

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

Verifying that the NetworkAttachmentDefinition CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

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

1 Replace **<namespace>** with the value for **networkNamespace** that you specified in the **SriovNetwork** object. For example, **sysctl-tuning-test**. The expected output shows the name of the NAD CRD and the creation age in minutes.

NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

Verifying that the additional SR-IOV network attachment is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

1. Create a **Pod** CR. Save the following YAML as the file **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. The name of the SR-IOV network attachment definition CR.
2. Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the SriovNetwork object.
3. Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

2. Create the **Pod** CR:

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

3. Verify that the pod is created by running the following command:

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

Example output

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

4. Log in to the pod by running the following command:

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

5. Verify the values of the configured sysctl flag. Find the value **net.ipv4.conf.IFNAME.accept_redirects** by running the following command::

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

6.3. CONFIGURING SYSCTL SETTINGS FOR PODS ASSOCIATED WITH BONDED SR-IOV INTERFACE FLAG

You can set interface-level network **sysctl** settings for a pod connected to a bonded SR-IOV network device.

In this example, the specific network interface-level **sysctl** settings that can be configured are set on the bonded interface.

The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

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

6.3.1. Setting all sysctl flag on nodes with bonded SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).

NOTE

When applying the configuration specified in a SriovNetworkNodePolicy object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes.

It might take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

Procedure

1. Create an **SriovNetworkNodePolicy** custom resource (CR). Save the following YAML as the file **policyallflags-sriov-node-network.yaml**. Replace **policyallflags** with the name for the configuration.

```
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
```

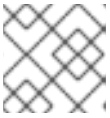
1. The name for the custom resource object.
2. The namespace where the SR-IOV Network Operator is installed.
3. The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.
4. The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.
5. Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.
6. The number of virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.

7 The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with

8 Optional: An array of one or more physical function (PF) names for the device.

9 Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.

10 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.



NOTE

The **vfio-pci** driver type is not supported.

2. Create the SriovNetworkNodePolicy object:

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

After applying the configuration update, all the pods in sriov-network-operator namespace change to the **Running** status.

3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured. Expected output shows **Succeeded**

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

6.3.2. Configuring sysctl on a bonded SR-IOV network

You can set interface specific **sysctl** settings on a bonded interface created from two SR-IOV interfaces. Do this by adding the tuning configuration to the optional **Plugins** parameter of the bond network attachment definition.

NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change specific interface-level network **sysctl** settings create the **SriovNetwork** custom resource (CR) with the Container Network Interface (CNI) tuning plugin by using the following procedure.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the bonded interface as in the following example CR. Save the YAML as the file **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) A name for the object. The SR-IOV Network Operator creates a NetworkAttachmentDefinition object with same name.

(2) The namespace where the SR-IOV Network Operator is installed.

(3) The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

(4) The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

(5) Optional: The capabilities to configure for this additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

2. Create the **SriovNetwork** resource:

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

3. Create a bond network attachment definition as in the following example CR. Save the YAML as the file **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. The type is **bond**.
2. The **mode** attribute specifies the bonding mode. The bonding modes supported are:
   - **balance-rr** - 0
   - **active-backup** - 1
   - **balance-xor** - 2
     For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.
3. The **failover** attribute is mandatory for active-backup mode.
4. The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.
5. The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.
6. A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition. In this pod example IP addresses are configured manually, so in this case,**ipam** is set to static.
7.

Add additional capabilities to the device. For example, set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the sysctl field. This example sets all

4. Create the bond network attachment resource:

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

Verifying that the NetworkAttachmentDefinition CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

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

1 Replace **<namespace>** with the networkNamespace that you specified when configuring the network attachment, for example, **sysctl-tuning-test**. Expected output shows the names of the NAD CRDs and the creation age in minutes.

NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

Verifying that the additional SR-IOV network resource is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

1. Create a **Pod** CR. For example, save the following YAML as the file **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. The name of the SR-IOV network attachment definition CR.
2. Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the SriovNetwork object.
3. Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

2. Apply the YAML:

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

3. Verify that the pod is created by running the following command:

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

Example output

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

4. Log in to the pod by running the following command:

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

5. Verify the values of the configured **sysctl** flag. Find the value **net.ipv6.neigh.IFNAME.base_reachable_time_ms** by running the following command:

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

6.4. ABOUT ALL-MULTICAST MODE

Enabling all-multicast mode, particularly in the context of rootless applications, is critical. If you do not enable this mode, you would be required to grant the **NET_ADMIN** capability to the pod's Security Context Constraints (SCC). If you were to allow the **NET_ADMIN** capability to grant the pod privileges to make changes that extend beyond its specific requirements, you could potentially expose security vulnerabilities.

The tuning CNI plugin supports changing several interface attributes, including all-multicast mode. By

enabling this mode, you can allow applications running on Virtual Functions (VFs) that are configured on a SR-IOV network device to receive multicast traffic from applications on other VFs, whether attached to the same or different physical functions.

6.4.1. Enabling the all-multicast mode on an SR-IOV network

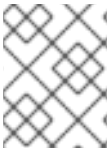
You can enable the all-multicast mode on an SR-IOV interface by:

- Adding the tuning configuration to the **metaPlugins** parameter of the **SriovNetwork** resource
- Setting the **allmulti** field to **true** in the tuning configuration

NOTE

Ensure that you create the virtual function (VF) with trust enabled.

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.



NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

Enable the all-multicast mode on a SR-IOV network by following this guidance.

Prerequisites

- You have installed the OpenShift Container Platform CLI (oc).
- You are logged in to the OpenShift Container Platform cluster as a user with **cluster-admin** privileges.
- You have installed the SR-IOV Network Operator.
- You have configured an appropriate **SriovNetworkNodePolicy** object.

Procedure

1. Create a YAML file with the following settings that defines a **SriovNetworkNodePolicy** object for a Mellanox ConnectX-5 device. Save the YAML file as **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. Optional: If the SR-IOV capable cluster nodes are not already labeled, add the **SriovNetworkNodePolicy.Spec.NodeSelector** label. For more information about labeling nodes, see "Understanding how to update labels on nodes".

3. Create the **SriovNetworkNodePolicy** object by running the following command:

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

   After applying the configuration update, all the pods in the **sriov-network-operator** namespace automatically move to a **Running** status.

4. Create the **enable-allmulti-test** namespace by running the following command:

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

5. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR YAML, and save the file as **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) Specify a name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with the same name.

(2) Specify the namespace where the SR-IOV Network Operator is installed.

3 Specify a value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional

4 Specify the target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

5 Specify a configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

6 Optional: Set capabilities for the additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

7 Specify the trust mode of the virtual function. This must be set to "on".

8 Add more capabilities to the device by using the **metaPlugins** parameter. In this use case, set the **type** field to **tuning**, and add the **allmulti** field and set it to **true**.

6. Create the **SriovNetwork** resource by running the following command:

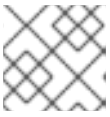
```
$ oc create -f sriov-enable-all-multicast.yaml
```

Verification of the NetworkAttachmentDefinition CR

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

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

1 Replace **<namespace>** with the value for **networkNamespace** that you specified in the **SriovNetwork** object. For this example, that is **enable-allmulti-test**. The expected output shows the name of the NAD CR and the creation age in minutes.



NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

- Display information about the SR-IOV network resources by running the following command:

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

Verification of the additional SR-IOV network attachment

To verify that the tuning CNI is correctly configured and that the additional SR-IOV network attachment is attached, follow these steps:

1. Create a **Pod** CR. Save the following sample YAML in a file named **examplepod.yaml**:

```
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) Specify the name of the SR-IOV network attachment definition CR.

(2) Optional: Specify the MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{"mac": true}** in the SriovNetwork object.

(3) Optional: Specify the IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

2. Create the **Pod** CR by running the following command:

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

3. Verify that the pod is created by running the following command:

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

Example output

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

4. Log in to the pod by running the following command:

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

5. List all the interfaces associated with the pod by running the following command:

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

Example output

```
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** is the primary interface

(2) **net1@if24** is the secondary interface configured with the network-attachment-definition that supports the all-multicast mode (**ALLMULTI** flag)

CHAPTER 7. CONFIGURING QINQ SUPPORT FOR SR-IOV ENABLED WORKLOADS

QinQ, formally known as 802.1Q-in-802.1Q, is a networking technique defined by IEEE 802.1ad. IEEE 802.1ad extends the IEEE 802.1Q-1998 standard and enriches VLAN capabilities by introducing an additional 802.1Q tag to packets already tagged with 802.1Q. This method is also referred to as VLAN stacking or double VLAN.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

7.1. ABOUT 802.1Q-IN-802.1Q SUPPORT

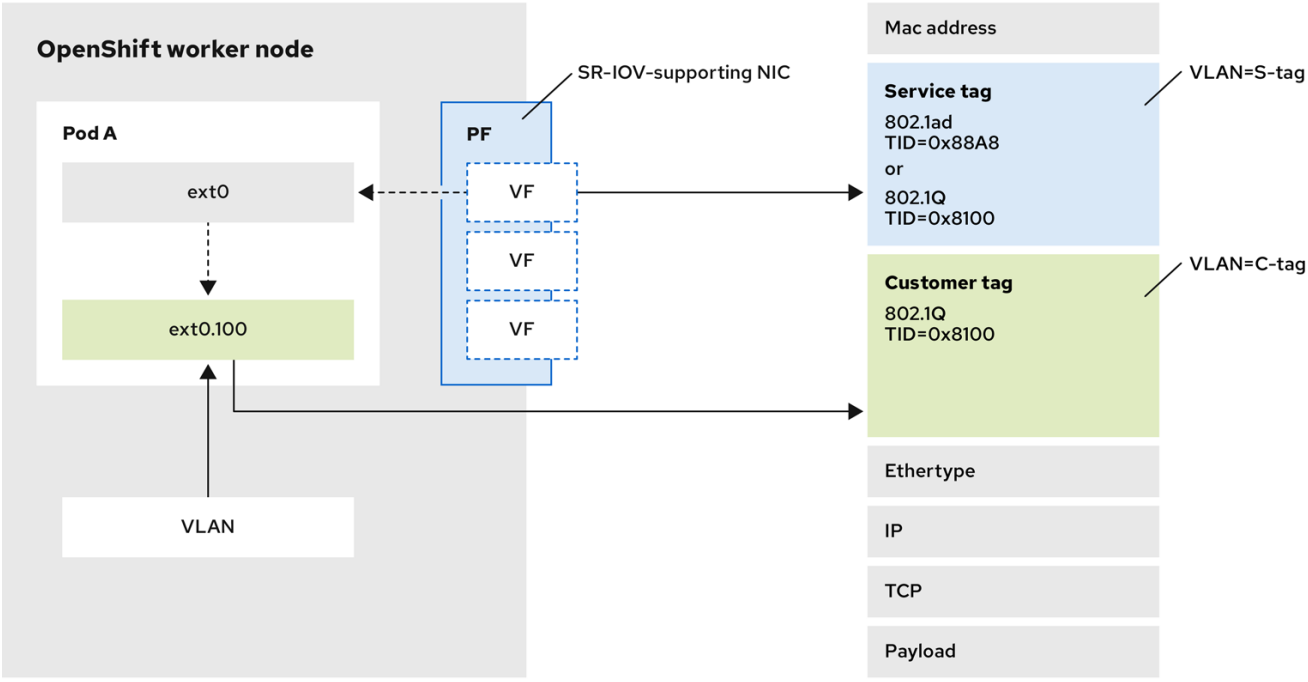
In traditional VLAN setups, frames typically contain a single VLAN tag, such as VLAN-100, as well as other metadata such as Quality of Service (QoS) bits and protocol information. QinQ introduces a second VLAN tag, where the service provider designates the outer tag for their use, offering them flexibility, while the inner tag remains dedicated to the customer's VLAN.

QinQ facilitates the creation of nested VLANs by using double VLAN tagging, enabling finer segmentation and isolation of traffic within a network environment. This approach is particularly valuable in service provider networks where you need to deliver VLAN-based services to multiple customers over a common infrastructure, while ensuring separation and isolation of traffic.

The following diagram illustrates how OpenShift Container Platform can use SR-IOV and QinQ to achieve advanced network segmentation and isolation for containerized workloads.

The diagram shows how double VLAN tagging (QinQ) works in a worker node with SR-IOV support. The SR-IOV virtual function (VF) located in the pod namespace, **ext0** is configured by the SR-IOV Container Network Interface (CNI) with a VLAN ID and VLAN protocol. This corresponds to the S-tag. Inside the pod, the VLAN CNI creates a subinterface using the primary interface **ext0**. This subinterface adds an internal VLAN ID using the 802.1Q protocol, which corresponds to the C-tag.

This demonstrates how QinQ enables finer traffic segmentation and isolation within the network. The Ethernet frame structure is detailed on the right, highlighting the inclusion of both VLAN tags, EtherType, IP, TCP, and Payload sections. QinQ facilitates the delivery of VLAN-based services to multiple customers over a shared infrastructure while ensuring traffic separation and isolation.



693_OpenShift_0624

The OpenShift Container Platform SR-IOV solution already supports setting the VLAN protocol on the **SriovNetwork** custom resource (CR). The virtual function (VF) can use this protocol to set the VLAN tag, also known as the outer tag. Pods can then use the VLAN CNI plugin to configure the inner tag.

Table 7.1. Supported network interface cards

NIC	802.1ad/802.1Q	802.1Q/802.1Q
Intel X710	No	Supported
Intel E810	Supported	Supported
Mellanox	No	Supported

Additional resources

- Configuration for an VLAN additional network

7.2. CONFIGURING QINQ SUPPORT FOR SR-IOV ENABLED WORKLOADS

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.

Procedure

Procedure

1. Create a file named **sriovnetpolicy-810-sriov-node-network.yaml** by using the following content:

```
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. Create the **SriovNetworkNodePolicy** object by running the following command:

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

3. Open a separate terminal window and monitor the synchronization status of the SR-IOV network node state for the node specified in the **openshift-sriov-network-operator** namespace by running the following command:

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

The synchronization status indicates a change from **InProgress** to **Succeeded**.

4. Create a **SriovNetwork** object, and set the outer VLAN called the S-tag, or **Service Tag**, as it belongs to the infrastructure.

IMPORTANT

You must configure the VLAN on the trunk interface of the switch. In addition, you might need to further configure some switches to support QinQ tagging.

a. Create a file named **nad-sriovnetwork-1ad-810.yaml** by using the following content:

```
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 Sets the S-tag VLAN tag to **171**.

2 Specifies the VLAN protocol to assign to the virtual function (VF). Supported values are **802.1ad** and **802.1q**. The default value is **802.1q**.

b. Create the object by running the following command:

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

5. Create a **NetworkAttachmentDefinition** object with an inner VLAN. The inner VLAN is often referred to as the C-tag, or **Customer Tag**, as it belongs to the Network Function:

a. Create a file named **nad-cvlan100.yaml** by using the following content:

```
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 Specifies the VF interface inside the pod. The default name is **net1** as the name is not set in the pod annotation.

b. Apply the YAML file by running the following command:

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

Verification

- Verify QinQ is active on the node by following this procedure:

1. Create a file named **test-qinq-pod.yaml** by using the following content:

```
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. Create the test pod by running the following command:

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

3. Enter into a debug session on the target node where the pod is present and display information about the network interface **ens5f0** by running the following command:

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

Example output

```
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
```

The **vlan protocol 802.1ad** ID in the output indicates that the interface supports VLAN tagging with protocol 802.1ad (QinQ). The VLAN ID is 171.

CHAPTER 8. USING HIGH PERFORMANCE MULTICAST

You can use multicast on your Single Root I/O Virtualization (SR-IOV) hardware network.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

8.1. HIGH PERFORMANCE MULTICAST

The OVN-Kubernetes network plugin supports multicast between pods on the default network. This is best used for low-bandwidth coordination or service discovery, and not high-bandwidth applications. For applications such as streaming media, like Internet Protocol television (IPTV) and multipoint videoconferencing, you can utilize Single Root I/O Virtualization (SR-IOV) hardware to provide near-native performance.

When using additional SR-IOV interfaces for multicast:

- Multicast packages must be sent or received by a pod through the additional SR-IOV interface.
- The physical network which connects the SR-IOV interfaces decides the multicast routing and topology, which is not controlled by OpenShift Container Platform.

8.2. CONFIGURING AN SR-IOV INTERFACE FOR MULTICAST

The follow procedure creates an example SR-IOV interface for multicast.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

1. Create a **SriovNetworkNodePolicy** object:

```
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. Create a **SriovNetwork** object:

```
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 If you choose to configure DHCP as IPAM, ensure that you provision the following default routes through your DHCP server: **224.0.0.0/5** and **232.0.0.0/5**. This is to override the static multicast route set by the default network provider.

3. Create a pod with multicast application:

```
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 The **NET_ADMIN** capability is required only if your application needs to assign the multicast IP address to the SR-IOV interface. Otherwise, it can be omitted.

CHAPTER 9. USING DPDK AND RDMA

The containerized Data Plane Development Kit (DPDK) application is supported on OpenShift Container Platform. You can use Single Root I/O Virtualization (SR-IOV) network hardware with the Data Plane Development Kit (DPDK) and with remote direct memory access (RDMA).

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

9.1. EXAMPLE USE OF A VIRTUAL FUNCTION IN A POD

You can run a remote direct memory access (RDMA) or a Data Plane Development Kit (DPDK) application in a pod with SR-IOV VF attached.

This example shows a pod using a virtual function (VF) in RDMA mode:

Pod spec that uses RDMA mode

```
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"]
```

The following example shows a pod with a VF in DPDK mode:

Pod spec that uses DPDK mode

```
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. USING A VIRTUAL FUNCTION IN DPDK MODE WITH AN INTEL NIC

Prerequisites

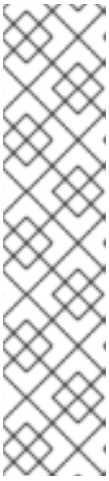
- Install the OpenShift CLI (**oc**).
- Install the SR-IOV Network Operator.
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **intel-dpdk-node-policy.yaml** file.

```
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) Specify the driver type for the virtual functions to **vfio-pci**.



NOTE

See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in **openshift-sriov-network-operator** namespace will change to a **Running** status.

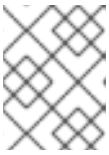
2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

3. Create the following **SriovNetwork** object, and then save the YAML in the **intel-dpdk-network.yaml** file.

```
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 Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.



NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriovNetwork** object by running the following command:

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

5. Create the following **Pod** spec, and then save the YAML in the **intel-dpdk-pod.yaml** file.

```
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. Specify the same **target_namespace** where the **SriovNetwork** object **intel-dpdk-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and the **SriovNetwork** object.

2. Specify the DPDK image which includes your application and the DPDK library used by application.

3. Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

4. Mount a hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

5. Optional: Specify the number of DPDK devices allocated to DPDK pod. This resource request and limit, if not explicitly specified, will be automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by the SR-IOV Operator. It is enabled by default and can be disabled by setting **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

6. Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.

7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately.

6. Create the DPDK pod by running the following command:

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

9.3. USING A VIRTUAL FUNCTION IN DPDK MODE WITH A MELLANOX NIC

You can create a network node policy and create a Data Plane Development Kit (DPDK) pod using a virtual function in DPDK mode with a Mellanox NIC.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the Single Root I/O Virtualization (SR-IOV) Network Operator.
- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Save the following **SriovNetworkNodePolicy** YAML configuration to an **mlx-dpdk-node-policy.yaml** file:

```
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 Specify the device hex code of the SR-IOV network device.

2 Specify the driver type for the virtual functions to **netdevice**. A Mellanox SR-IOV Virtual Function (VF) can work in DPDK mode without using the **vfio-pci** device type. The VF device appears as a kernel network interface inside a container.

3

Enable Remote Direct Memory Access (RDMA) mode. This is required for Mellanox cards to work in DPDK mode.

NOTE

See *Configuring an SR-IOV network device* for a detailed explanation of each option in the **SriovNetworkNodePolicy** object.

When applying the configuration specified in an **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. It might take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

3. Save the following **SriovNetwork** YAML configuration to an **mlx-dpdk-network.yaml** file:

```
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 Specify a configuration object for the IP Address Management (IPAM) Container Network Interface (CNI) plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

NOTE

See *Configuring an SR-IOV network device* for a detailed explanation on each option in the **SriovNetwork** object.

The **app-netutil** option library provides several API methods for gathering network information about the parent pod of a container.

4. Create the **SriovNetwork** object by running the following command:

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

5. Save the following **Pod** YAML configuration to an **mlx-dpdk-pod.yaml** file:

```
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. Specify the same **target_namespace** where **SriovNetwork** object **mlx-dpdk-network** is created. To create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and **SriovNetwork** object.

2. Specify the DPDK image which includes your application and the DPDK library used by the application.

3. Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

4. Mount the hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the **emptyDir** volume type with the medium being **Hugepages**.

5. Optional: Specify the number of DPDK devices allocated for the DPDK pod. If not explicitly specified, this resource request and limit is automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

6 Specify the number of CPUs. The DPDK pod usually requires that exclusive CPUs be allocated from the kubelet. To do this, set the CPU Manager policy to **static** and create a

7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepages requires adding kernel arguments to Nodes.

6. Create the DPDK pod by running the following command:

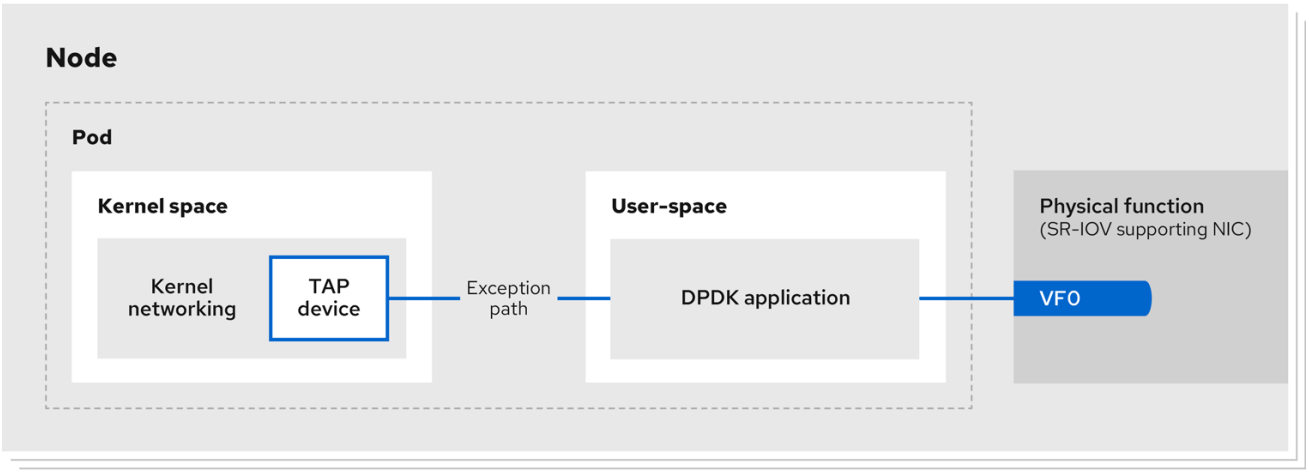
```
$ oc create -f mlx-dpdk-pod.yaml
```

9.4. USING THE TAP CNI TO RUN A ROOTLESS DPDK WORKLOAD WITH KERNEL ACCESS

DPDK applications can use **virtio-user** as an exception path to inject certain types of packets, such as log messages, into the kernel for processing. For more information about this feature, see Virtio_user as Exception Path.

In OpenShift Container Platform version 4.14 and later, you can use non-privileged pods to run DPDK applications alongside the tap CNI plugin. To enable this functionality, you need to mount the **vhost-net** device by setting the **needVhostNet** parameter to **true** within the **SriovNetworkNodePolicy** object.

Figure 9.1. DPDK and TAP example configuration



Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the SR-IOV Network Operator.
- You are logged in as a user with **cluster-admin** privileges.
- Ensure that **setsebools container_use_devices=on** is set as root on all nodes.

NOTE

Use the Machine Config Operator to set this SELinux boolean.

Procedure

1. Create a file, such as **test-namespace.yaml**, with content like the following example:

```
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. Create the new **Namespace** object by running the following command:

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

3. Create a file, such as **sriov-node-network-policy.yaml**, with content like the following example::

```
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) This indicates that the profile is tailored specifically for Mellanox Network Interface Controllers (NICs).

(2) Setting **isRdma** to **true** is only required for a Mellanox NIC.

(3) This mounts the **/dev/net/tun** and **/dev/vhost-net** devices into the container so the application can create a tap device and connect the tap device to the DPDK workload.

(4) The vendor hexadecimal code of the SR-IOV network device. The value 15b3 is associated with a Mellanox NIC.

5 The device hexadecimal code of the SR-IOV network device.

4. Create the **SriovNetworkNodePolicy** object by running the following command:

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

5. Create the following **SriovNetwork** object, and then save the YAML in the **sriov-network-attachment.yaml** file:

```
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"
```

NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, **app-netutil**, provides several API methods for gathering network information about a container's parent pod.

6. Create the **SriovNetwork** object by running the following command:

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

7. Create a file, such as **tap-example.yaml**, that defines a network attachment definition, with content like the following example:

```
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 Specify the same **target_namespace** where the **SriovNetwork** object is created.

8. Create the **NetworkAttachmentDefinition** object by running the following command:

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

9. Create a file, such as **dpdk-pod-rootless.yaml**, with content like the following example:

```
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. Specify the same **target_namespace** in which the **SriovNetwork** object is created. If you want to create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and the **SriovNetwork** object.

2. Sets the group ownership of volume-mounted directories and files created in those volumes.

3. Specify the primary group ID used for running the container.

4. Specify the DPDK image that contains your application and the DPDK library used by application.

5. Removing all capabilities (**ALL**) from the container's securityContext means that the container has no special privileges beyond what is necessary for normal operation.

6. Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access. These capabilities must also be set in the binary file by using the **setcap** command.

7. Mellanox network interface controller (NIC) requires the **NET_RAW** capability.

8. Specify the user ID used for running the container.

9. This setting indicates that the container or containers within the pod should not be granted privileged access to the host system.

10. This setting allows a container to escalate its privileges beyond the initial non-root privileges it might have been assigned.

11. This setting ensures that the container runs with a non-root user. This helps enforce the principle of least privilege, limiting the potential impact of compromising the container and reducing the attack surface.

12. Mount a hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

13. Optional: Specify the number of DPDK devices allocated for the DPDK pod. If not explicitly specified, this resource request and limit is automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

14.

Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and

15 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes. For example, adding kernel arguments **default_hugepagesz=1GB**, **hugepagesz=1G** and **hugepages=16** will result in **16*1Gi** hugepages be allocated during system boot.

16 If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

10. Create the DPDK pod by running the following command:

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

Additional resources

- Creating a performance profile
- Configuring an SR-IOV network device

9.5. OVERVIEW OF ACHIEVING A SPECIFIC DPDK LINE RATE

To achieve a specific Data Plane Development Kit (DPDK) line rate, deploy a Node Tuning Operator and configure Single Root I/O Virtualization (SR-IOV). You must also tune the DPDK settings for the following resources:

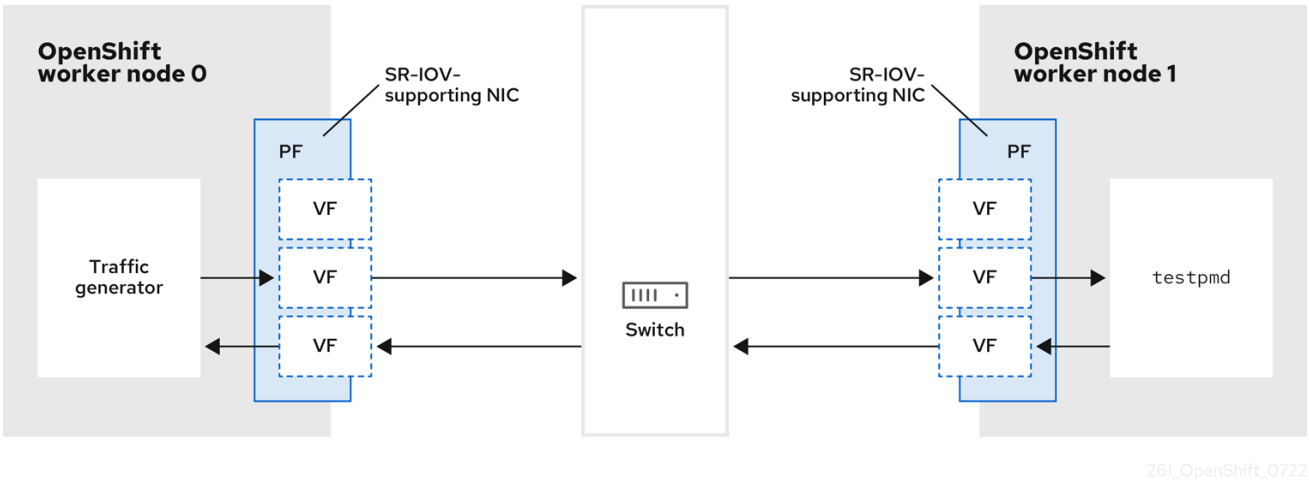
- Isolated CPUs
- Hugepages
- The topology scheduler

NOTE

In previous versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

DPDK test environment

The following diagram shows the components of a traffic-testing environment:



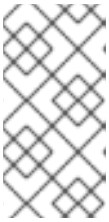
- **Traffic generator**: An application that can generate high-volume packet traffic.
- **SR-IOV-supporting NIC**: A network interface card compatible with SR-IOV. The card runs a number of virtual functions on a physical interface.
- **Physical Function (PF)**: A PCI Express (PCIe) function of a network adapter that supports the SR-IOV interface.
- **Virtual Function (VF)**: A lightweight PCIe function on a network adapter that supports SR-IOV. The VF is associated with the PCIe PF on the network adapter. The VF represents a virtualized instance of the network adapter.
- **Switch**: A network switch. Nodes can also be connected back-to-back.
- **testpmd**: An example application included with DPDK. The **testpmd** application can be used to test the DPDK in a packet-forwarding mode. The **testpmd** application is also an example of how to build a fully-fledged application using the DPDK Software Development Kit (SDK).
- **worker 0** and **worker 1**: OpenShift Container Platform nodes.

9.6. USING SR-IOV AND THE NODE TUNING OPERATOR TO ACHIEVE A DPDK LINE RATE

You can use the Node Tuning Operator to configure isolated CPUs, hugepages, and a topology scheduler. You can then use the Node Tuning Operator with Single Root I/O Virtualization (SR-IOV) to achieve a specific Data Plane Development Kit (DPDK) line rate.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the SR-IOV Network Operator.
- You have logged in as a user with **cluster-admin** privileges.
- You have deployed a standalone Node Tuning Operator.



NOTE

In previous versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

Procedure

1. Create a **PerformanceProfile** object based on the following example:

```
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) If hyperthreading is enabled on the system, allocate the relevant symbolic links to the **isolated** and **reserved** CPU groups. If the system contains multiple non-uniform memory access nodes (NUMAs), allocate CPUs from both NUMAs to both groups. You can also use the Performance Profile Creator for this task. For more information, see *Creating a performance profile*.

(2) You can also specify a list of devices that will have their queues set to the reserved CPU count. For more information, see *Reducing NIC queues using the Node Tuning Operator* .

(3) Allocate the number and size of hugepages needed. You can specify the NUMA configuration for the hugepages. By default, the system allocates an even number to every NUMA node on the system. If needed, you can request the use of a realtime kernel for the nodes. See *Provisioning a worker with real-time capabilities* for more information.

2. Save the **yaml** file as **mlx-dpdk-perfprofile-policy.yaml**.

3. Apply the performance profile using the following command:

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

9.6.1. DPDK library for use with container applications

An optional library, **app-netutil**, provides several API methods for gathering network information about a pod from within a container running within that pod.

This library can assist with integrating SR-IOV virtual functions (VFs) in Data Plane Development Kit (DPDK) mode into the container. The library provides both a Golang API and a C API.

Currently there are three API methods implemented:

GetCPUInfo()

This function determines which CPUs are available to the container and returns the list.

GetHugepages()

This function determines the amount of huge page memory requested in the **Pod** spec for each container and returns the values.

GetInterfaces()

This function determines the set of interfaces in the container and returns the list. The return value includes the interface type and type-specific data for each interface.

The repository for the library includes a sample Dockerfile to build a container image, **dpdk-app-centos**. The container image can run one of the following DPDK sample applications, depending on an environment variable in the pod specification: **l2fwd**, **l3wd** or **testpmd**. The container image provides an example of integrating the **app-netutil** library into the container image itself. The library can also integrate into an init container. The init container can collect the required data and pass the data to an existing DPDK workload.

9.6.2. Example SR-IOV Network Operator for virtual functions

You can use the Single Root I/O Virtualization (SR-IOV) Network Operator to allocate and configure Virtual Functions (VFs) from SR-IOV-supporting Physical Function NICs on the nodes.

For more information on deploying the Operator, see *Installing the SR-IOV Network Operator* . For more information on configuring an SR-IOV network device, see *Configuring an SR-IOV network device* .

There are some differences between running Data Plane Development Kit (DPDK) workloads on Intel VFs and Mellanox VFs. This section provides object configuration examples for both VF types. The following is an example of an **sriovNetworkNodePolicy** object used to run DPDK applications on Intel NICs:

```
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. For Intel NICs, **deviceType** must be **vfio-pci**.
2. If kernel communication with DPDK workloads is required, add **needVhostNet: true**. This mounts the **/dev/net/tun** and **/dev/vhost-net** devices into the container so the application can create a tap device and connect the tap device to the DPDK workload.

The following is an example of an **sriovNetworkNodePolicy** object for Mellanox NICs:

```
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. For Mellanox devices the **deviceType** must be **netdevice**.
2. For Mellanox devices **isRdma** must be **true**. Mellanox cards are connected to DPDK applications using Flow Bifurcation. This mechanism splits traffic between Linux user space and kernel space, and can enhance line rate processing capability.

9.6.3. Example SR-IOV network operator

The following is an example definition of an **sriovNetwork** object. In this case, Intel and Mellanox configurations are identical:

```
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. You can use a different IP Address Management (IPAM) implementation, such as Whereabouts. For more information, see *Dynamic IP address assignment configuration with Whereabouts* .
2. You must request the **networkNamespace** where the network attachment definition will be created. You must create the **sriovNetwork** CR under the **openshift-sriov-network-operator** namespace.
3. The **resourceName** value must match that of the **resourceName** created under the **sriovNetworkNodePolicy**.

9.6.4. Example DPDK base workload

The following is an example of a Data Plane Development Kit (DPDK) container:

```
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. Request the SR-IOV networks you need. Resources for the devices will be injected automatically.
2. Disable the CPU and IRQ load balancing base. See *Disabling interrupt processing for individual pods* for more information.
3. Set the **runtimeClass** to **performance-performance**. Do not set the **runtimeClass** to **HostNetwork** or **privileged**.
4. Request an equal number of resources for requests and limits to start the pod with **Guaranteed** Quality of Service (QoS).

NOTE

Do not start the pod with **SLEEP** and then exec into the pod to start the testpmd or the DPDK workload. This can add additional interrupts as the **exec** process is not pinned to any CPU.

9.6.5. Example testpmd script

The following is an example script for running **testpmd**:

```
#!/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
```

This example uses two different **sriovNetwork** CRs. The environment variable contains the Virtual Function (VF) PCI address that was allocated for the pod. If you use the same network in the pod definition, you must split the **pciAddress**. It is important to configure the correct MAC addresses of the traffic generator. This example uses custom MAC addresses.

9.7. USING A VIRTUAL FUNCTION IN RDMA MODE WITH A MELLANOX NIC

IMPORTANT

RDMA over Converged Ethernet (RoCE) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

RDMA over Converged Ethernet (RoCE) is the only supported mode when using RDMA on OpenShift Container Platform.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the SR-IOV Network Operator.
- Log in as a user with **cluster-admin** privileges.

Procedure

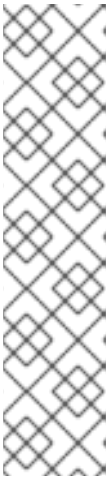
1. Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **mlx-rdma-node-policy.yaml** file.

```
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) Specify the device hex code of the SR-IOV network device.

(2) Specify the driver type for the virtual functions to **netdevice**.

(3) Enable RDMA mode.



NOTE

See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

3. Create the following **SriovNetwork** object, and then save the YAML in the **mlx-rdma-network.yaml** file.

```
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 Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriovNetworkNodePolicy** object by running the following command:

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

5. Create the following **Pod** spec, and then save the YAML in the **mlx-rdma-pod.yaml** file.

```
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) Specify the same **target_namespace** where **SriovNetwork** object **mlx-rdma-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both **Pod** spec and **SriovNetwork** object.

(2) Specify the RDMA image which includes your application and RDMA library used by application.

(3) Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

(4) Mount the hugepage volume to RDMA pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

(5) Specify number of CPUs. The RDMA pod usually requires exclusive CPUs be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and create pod with **Guaranteed** QoS.

(6) Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the RDMA pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

6. Create the RDMA pod by running the following command:

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

9.8. A TEST POD TEMPLATE FOR CLUSTERS THAT USE OVS-DPDK ON OPENSTACK

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

An example testpmd pod

```
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 The name **dpdk1** in this example is a user-created **SriovNetworkNodePolicy** resource. You can substitute this name for that of a resource that you create.

2 If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

9.9. ADDITIONAL RESOURCES

- Red Hat certified hardware (Red Hat Ecosystem Catalog)
- Creating a performance profile
- Adjusting the NIC queues with the performance profile
- Provisioning real-time and low latency workloads
- Installing the SR-IOV Network Operator
- Configuring an SR-IOV network device
- Dynamic IP address assignment configuration with Whereabouts
- Disabling interrupt processing for individual pods
- Configuring an SR-IOV Ethernet network attachment

CHAPTER 10. USING POD-LEVEL BONDING

Bonding at the pod level is vital to enable workloads inside pods that require high availability and more throughput. With pod-level bonding, you can create a bond interface from multiple single root I/O virtualization (SR-IOV) virtual function interfaces in a kernel mode interface. The SR-IOV virtual functions are passed into the pod and attached to a kernel driver.

One scenario where pod level bonding is required is creating a bond interface from multiple SR-IOV virtual functions on different physical functions. Creating a bond interface from two different physical functions on the host can be used to achieve high availability and throughput at pod level.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

For guidance on tasks such as creating a SR-IOV network, network policies, network attachment definitions and pods, see Configuring an SR-IOV network device .

10.1. CONFIGURING A BOND INTERFACE FROM TWO SR-IOV INTERFACES

Bonding enables multiple network interfaces to be aggregated into a single logical "bonded" interface. Bond Container Network Interface (Bond-CNI) brings bond capability into containers.

Bond-CNI can be created using Single Root I/O Virtualization (SR-IOV) virtual functions and placing them in the container network namespace.

OpenShift Container Platform only supports Bond-CNI using SR-IOV virtual functions. The SR-IOV Network Operator provides the SR-IOV CNI plugin needed to manage the virtual functions. Other CNIs or types of interfaces are not supported.

Prerequisites

- The SR-IOV Network Operator must be installed and configured to obtain virtual functions in a container.
- To configure SR-IOV interfaces, an SR-IOV network and policy must be created for each interface.
- The SR-IOV Network Operator creates a network attachment definition for each SR-IOV interface, based on the SR-IOV network and policy defined.
- The **linkState** is set to the default value **auto** for the SR-IOV virtual function.

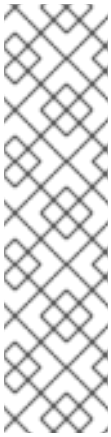
10.1.1. Creating a bond network attachment definition

Now that the SR-IOV virtual functions are available, you can create a bond network attachment definition.

```
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. The cni-type is always set to **bond**.
2. The **mode** attribute specifies the bonding mode.



NOTE

The bonding modes supported are:

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

For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.

3. The **failover** attribute is mandatory for active-backup mode and must be set to 1.
4. The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.
5. The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.

10.1.2. Creating a pod using a bond interface

1. Test the setup by creating a pod with a YAML file named for example **podbonding.yaml** with content similar to the following:

```
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) Note the network annotation: it contains two SR-IOV network attachments, and one bond network attachment. The bond attachment uses the two SR-IOV interfaces as bonded port interfaces.

2. Apply the yaml by running the following command:

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

3. Inspect the pod interfaces with the following command:

```
$ 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) The bond interface is automatically named **net3**. To set a specific interface name add **@name** suffix to the pod's **k8s.v1.cni.cncf.io/networks** annotation.

(2) The **net1** interface is based on an SR-IOV virtual function.

(3) The **net2** interface is based on an SR-IOV virtual function.

NOTE

If no interface names are configured in the pod annotation, interface names are assigned automatically as **net<n>**, with **<n>** starting at **1**.

4. Optional: If you want to set a specific interface name for example **bond0**, edit the **k8s.v1.cni.cncf.io/networks** annotation and set **bond0** as the interface name as follows:

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

CHAPTER 11. CONFIGURING HARDWARE OFFLOADING

As a cluster administrator, you can configure hardware offloading on compatible nodes to increase data processing performance and reduce load on host CPUs.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

11.1. ABOUT HARDWARE OFFLOADING

Open vSwitch hardware offloading is a method of processing network tasks by diverting them away from the CPU and offloading them to a dedicated processor on a network interface controller. As a result, clusters can benefit from faster data transfer speeds, reduced CPU workloads, and lower computing costs.

The key element for this feature is a modern class of network interface controllers known as SmartNICs. A SmartNIC is a network interface controller that is able to handle computationally-heavy network processing tasks. In the same way that a dedicated graphics card can improve graphics performance, a SmartNIC can improve network performance. In each case, a dedicated processor improves performance for a specific type of processing task.

In OpenShift Container Platform, you can configure hardware offloading for bare metal nodes that have a compatible SmartNIC. Hardware offloading is configured and enabled by the SR-IOV Network Operator.

Hardware offloading is not compatible with all workloads or application types. Only the following two communication types are supported:

- pod-to-pod
- pod-to-service, where the service is a ClusterIP service backed by a regular pod

In all cases, hardware offloading takes place only when those pods and services are assigned to nodes that have a compatible SmartNIC. Suppose, for example, that a pod on a node with hardware offloading tries to communicate with a service on a regular node. On the regular node, all the processing takes place in the kernel, so the overall performance of the pod-to-service communication is limited to the maximum performance of that regular node. Hardware offloading is not compatible with DPDK applications.

Enabling hardware offloading on a node, but not configuring pods to use, it can result in decreased throughput performance for pod traffic. You cannot configure hardware offloading for pods that are managed by OpenShift Container Platform.

11.2. PREREQUISITES

- Your cluster has at least one bare metal machine with a network interface controller that is supported for hardware offloading.
- You installed the SR-IOV Network Operator.
- Your cluster uses the OVN-Kubernetes network plugin.
- In your OVN-Kubernetes network plugin configuration, the **gatewayConfig.routingViaHost** field is set to **false**.

11.3. SETTING THE SR-IOV NETWORK OPERATOR INTO SYSTEMD MODE

To support hardware offloading, you must first set the SR-IOV Network Operator into **systemd** mode.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user that has the **cluster-admin** role.

Procedure

1. Create a **SriovOperatorConfig** custom resource (CR) to deploy all the SR-IOV Operator components:
   a. Create a file named **sriovOperatorConfig.yaml** that contains the following 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) The only valid name for the **SriovOperatorConfig** resource is **default** and it must be in the namespace where the Operator is deployed.

(2) Setting the SR-IOV Network Operator into **systemd** mode is only relevant for Open vSwitch hardware offloading.

   b. Create the resource by running the following command:

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

11.4. CONFIGURING A MACHINE CONFIG POOL FOR HARDWARE OFFLOADING

To enable hardware offloading, you now create a dedicated machine config pool and configure it to work with the SR-IOV Network Operator.

Prerequisites

- SR-IOV Network Operator installed and set into **systemd** mode.

Procedure

1. Create a machine config pool for machines you want to use hardware offloading on.

a. Create a file, such as **mcp-offloading.yaml**, with content like the following example:

```
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 The name of your machine config pool for hardware offloading.

3 This node role label is used to add nodes to the machine config pool.

b. Apply the configuration for the machine config pool:

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

2. Add nodes to the machine config pool. Label each node with the node role label of your pool:

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

3. Optional: To verify that the new pool is created, run the following command:

```
$ oc get nodes
```

Example output

```
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. Add this machine config pool to the **SriovNetworkPoolConfig** custom resource:

a. Create a file, such as **sriov-pool-config.yaml**, with content like the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

1 The name of your machine config pool for hardware offloading.

b. Apply the configuration:

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

NOTE

When you apply the configuration specified in a **SriovNetworkPoolConfig** object, the SR-IOV Operator drains and restarts the nodes in the machine config pool.

It might take several minutes for a configuration changes to apply.

11.5. CONFIGURING THE SR-IOV NETWORK NODE POLICY

You can create an SR-IOV network device configuration for a node by creating an SR-IOV network node policy. To enable hardware offloading, you must define the **.spec.eSwitchMode** field with the value **"switchdev"**.

The following procedure creates an SR-IOV interface for a network interface controller with hardware offloading.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a file, such as **sriov-node-policy.yaml**, with content like the following example:

```
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. The name for the custom resource object.
2. Required. Hardware offloading is not supported with **vfio-pci**.
3. Required.

2. Apply the configuration for the policy:

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

NOTE

When you apply the configuration specified in a **SriovNetworkPoolConfig** object, the SR-IOV Operator drains and restarts the nodes in the machine config pool.

It might take several minutes for a configuration change to apply.

11.5.1. An example SR-IOV network node policy for OpenStack

The following example describes an SR-IOV interface for a network interface controller (NIC) with hardware offloading on Red Hat OpenStack Platform (RHOSP).

An SR-IOV interface for a NIC with hardware offloading on RHOSP

```
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}
```

11.6. IMPROVING NETWORK TRAFFIC PERFORMANCE USING A VIRTUAL FUNCTION

Follow this procedure to assign a virtual function to the OVN-Kubernetes management port and increase its network traffic performance.

This procedure results in the creation of two pools: the first has a virtual function used by OVN-Kubernetes, and the second comprises the remaining virtual functions.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Add the **network.operator.openshift.io/smart-nic** label to each worker node with a SmartNIC present by running the following command:

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

   Use the **oc get nodes** command to get a list of the available nodes.

2. Create a policy named **sriov-node-mgmt-vf-policy.yaml** for the management port with content such as the following example:

```
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) Replace this device with the appropriate network device for your use case. The **#0-0** part of the **pfNames** value reserves a single virtual function used by OVN-Kubernetes.

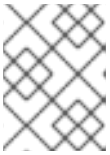
   (2) The value provided here is an example. Replace this value with one that meets your requirements. For more information, see *SR-IOV network node configuration object* in the *Additional resources* section.

3. Create a policy named **sriov-node-policy.yaml** with content such as the following example:

```
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. Replace this device with the appropriate network device for your use case.
2. The value provided here is an example. Replace this value with the value specified in the **sriov-node-mgmt-vf-policy.yaml** file. For more information, see *SR-IOV network node configuration object* in the *Additional resources* section.



NOTE

The **sriov-node-mgmt-vf-policy.yaml** file has different values for the **pfNames** and **resourceName** keys than the **sriov-node-policy.yaml** file.

4. Apply the configuration for both policies:

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

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

5. Create a Cluster Network Operator (CNO) ConfigMap in the cluster for the management configuration:
   a. Create a ConfigMap named **hardware-offload-config.yaml** with the following contents:

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

   b. Apply the configuration for the ConfigMap:

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

Additional resources

- SR-IOV network node configuration object

11.7. CREATING A NETWORK ATTACHMENT DEFINITION

After you define the machine config pool and the SR-IOV network node policy, you can create a network attachment definition for the network interface card you specified.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a file, such as **net-attach-def.yaml**, with content like the following example:

```
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. The name for your network attachment definition.
2. The namespace for your network attachment definition.
3. This is the value of the **spec.resourceName** field you specified in the **SriovNetworkNodePolicy** object.

2. Apply the configuration for the network attachment definition:

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

Verification

- Run the following command to check that the new definition exists:

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

The output shows the namespace, name, and age of the new definition.

11.8. ADDING THE NETWORK ATTACHMENT DEFINITION TO YOUR PODS

After you create the machine config pool, the **SriovNetworkPoolConfig** and **SriovNetworkNodePolicy** custom resources, and the network attachment definition, you can apply these configurations to your pods by adding the network attachment definition to your pod specifications.

Procedure

- In the pod specification, add the **.metadata.annotations.k8s.v1.cni.cncf.io/networks** field and specify the network attachment definition you created for hardware offloading:

```
....
metadata:
  annotations:
    v1.multus-cni.io/default-network: net-attach-def/net-attach-def 1
```

1 The value must be the name and namespace of the network attachment definition you created for hardware offloading.

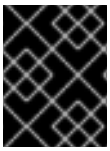
CHAPTER 12. SWITCHING BLUEFIELD-2 FROM DPU TO NIC

You can switch the Bluefield-2 network device from data processing unit (DPU) mode to network interface controller (NIC) mode.

Before you perform any tasks in the following documentation, ensure that you installed the SR-IOV Network Operator.

12.1. SWITCHING BLUEFIELD-2 FROM DPU MODE TO NIC MODE

Use the following procedure to switch Bluefield-2 from data processing units (DPU) mode to network interface controller (NIC) mode.



IMPORTANT

Currently, only switching Bluefield-2 from DPU to NIC mode is supported. Switching from NIC mode to DPU mode is unsupported.

Prerequisites

- You have installed the SR-IOV Network Operator. For more information, see "Installing SR-IOV Network Operator".
- You have updated Bluefield-2 to the latest firmware. For more information, see Firmware for NVIDIA BlueField-2.

Procedure

1. Add the following label to each of your compute nodes by entering the following command. You must run the command for each compute node.

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

where:

node_name

Refers to the name of a compute node.

1. Create a machine config pool for the SR-IOV Network Operator, for example:

```
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. Apply the following **machineconfig.yaml** file to the compute nodes:

-

```
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) Optional: The PCI address of a specific card can optionally be specified, for example **ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'**. By default, the first device is selected. If there is more than one device, you must specify which PCI address to be used. The PCI address must be the same on all nodes that are switching Bluefield-2 from DPU mode to NIC mode.

3. Wait for the compute nodes to restart. After restarting, the Bluefield-2 network device on the compute nodes is switched into NIC mode.

4. Optional: You might need to restart the host hardware because most recent Bluefield-2 firmware releases require a hardware restart to switch into NIC mode.

Additional resources

- Installing SR-IOV Network Operator