NVIDIA Network Operator v24.4.0

Table of contents

List of Tables

The NVIDIA Network Operator simplifies the provisioning and management of NVIDIA networking resources in a Kubernetes cluster. The operator automatically installs the required host networking software - bringing together all the needed components to provide high-speed network connectivity. These components include the NVIDIA networking driver, Kubernetes device plugin, CNI plugins, IP address management (IPAM) plugin and others. The NVIDIA Network Operator works in conjunction with the NVIDIA GPU Operator to deliver high-throughput, low-latency networking for scale-out, GPU computing clusters.

A Helm chart is provided for easily deploying the Network operator in a cluster to provision the host software on NVIDIA-enabled nodes.

License Agreement

The NVIDIA Network Operator is licensed under Apache 2.0 and contributions are accepted with a DCO. See the contributing document for more information on how to contribute and the release artifacts.

Learn More

The Network Operator is open-source. For more information on contributions and release artifacts, see the GitHub repo.

Release Notes

On this page

Changes and New Features

Version	Description
24.4.0	- Added support for OpenShift Container Platform v4.15. - Added support for Ubuntu 24.04. - Added support for NVIDIA Grace based ARM platforms with Ubuntu 22.04 and Upstream K8s as a Tech Preview feature. - Added support for NVIDIA IGX Orin based ARM platforms with Ubuntu 22.04 and Upstream K8s as a GA feature. - Added support for Precompiled DOCA Driver containers for Ubuntu 22.04. - Added support for Switchdev SR-IOV mode with SR-IOV Network Operator and OVS CNI as a Tech Preview feature. - Added support for DOCA Telemetry Service (DTS) integration to expose network telemetry and NIC metrics in K8s. - Added support for network namespace isolation of RDMA devices with RDMA CNI.

	- Added support for RHEL and OpenShift deployments with Real-time kernels. - Enhanced DOCA Driver container deployment and significantly reduced compilation time after node reboots.
24.1.0	- Added support for Ubuntu 22.04 with Upstream K8s on ARM platforms (NVIDIA IGX Orin) - Tech Preview. - Added support for CNI bin directory configuration. - Added support for OpenShift MOFED/DOCA driver container build and deployment via driver toolkit (DTK). - Added support for Ubuntu 22.04 deployments with Real-time kernels. - Added the ability to disable SR-IOV VF for SR-IOV Network Operator (in systems with pre-configured SR-IOV). - Added the ability to set resource request and limits on the network operator and it components.
23.10.0	- Added support for OpenShift Container Platform v4.14. - Added support for RHEL v8.8. - Optimized SR-IOV NIC configuration time with Network Operator (vanilla Kubernetes only). - Added a validating admission controller for NVIDIA Network Operator. - Added support for NIC Feature Discovery (driver version discovery). - Added CDI support for SR-IOV Network Device Plugin and RDMA Shared Device Plugin for network device persistency. - Added support for NVIDIA BlueField-3 NIC mode. - Added High-Availability and Leader election support for NV-IPAM. - Added systemd mode support for SR-IOV Network Operator and MOFED container to optimize cluster/node startup time.
23.7.0	- Added support for OpenShift Container Platform 4.13. - Added support for RHEL 9.1 and 9.2 with CRI-O container runtime (Beta). - Added support for NodeFeatureApi in Node Feature Discovery.
23.5.0	- Added support for NVIDIA IPAM Plugin deployment. - Added support for CRDs upgrade during NVIDIA Network Operator installation or upgrade.
23.4.0	- Added support for Kubernetes >= 1.21 and <=1.27. - Added support for NicClusterPolicy update and removal.

	- Added support for OpenShift Container Platform 4.11 and 4.12.
23.4.0	- Added a calendar versioning schema for Network Operator releases to better align with the NVIDIA GPU Operator. - Added support for the following operating systems and Kubernetes environments: - RHEL 8.4 and 8.6 with CRI-O container runtime - Kubernetes >= 1.21 and <=1.26 - Added PKey configuration for IB networks with IB-Kubernetes. - Added the ability to gracefully terminate the OFED container on DGX systems running Red Hat OpenShift.
1.4.0	- Added support for Kubernetes >= 1.21 and <=1.25. - Added support for Ubuntu 22.04. - Added support for OpenShift Container Platform 4.11 including DGX platform. - Added Beta support for PKey configuration for IB networks with IB-Kubernetes.
1.3.0	- Added support for Kubernetes >= 1.17 and <=1.24. - Added the option to use a single namespace to deploy Network Operator components. - Added support for automatic MLNX OFED driver upgrade. - Added support for IPoIB CNI. - Added support for Air Gap deployment.
1.2.0	- Added support for OpenShift Container Platform 4.10. - Added extended selectors support for SR-IOV Device Plugin resources with Helm chart. - Added Whereabouts IP reconciler support. - Added BlueField2 NICs support for SR-IOV operator.
1.1.0	- Added support for OpenShift Container Platform 4.9. - Added support for Network Operator upgrade from v1.0.0. - Added support for Kubernetes POD Security Policy. - Added support for Kubernetes >= 1.17 and <=1.22. - Added the ability to propagate nodeAffinity property from the

	NicClusterPolicy to Network Operator dependencies.
1.0.0	- Added Node Feature Discovery that can be used to mark nodes with NVIDIA SR-IOV NICs. - Added support for different networking models: - Macvlan Network - HostDevice Network - SR-IOV Network - Added Kubernetes cluster scale-up support. - Published Network Operator image at NGC. - Added support for Kubernetes >= 1.17 and <=1.21.

General Support

Upgrade Notes

Version	Notes
23.10.0	- In NV-IPAM v0.1.1, the IP Pools configurations are read from IPPool CRs instead of using a ConfigMap. Existing ConfigMap configuration will be automatically migrated to IPPools CRs as part of the upgrade process.
23.7.0	- Dropped MLNX_OFED support for versions older than 5.7-0.1.2.0. - Removed nv-peer-mem support in favor of nvidia-peer-mem.
1.3.0	- The option of manual gradual upgrade is not supported when upgrading to Network Operator v1.3.0, since all pods are dropped/restarted in case components are deployed into the single namespace when the old namespace is deleted. This could lead to networking connectivity issues during the upgrade procedure.
1.2.0	- Network Operator 1.2.0 deploys the NVIDIA MLNX_OFED 5.6 driver container by default. When deployed, depending on your system kernel and OS configuration, the network device name may change, as it no longer installs an udev rule to force network device naming scheme. Instead, the default setting uses the name already configured in the system by either *systemd.network* or any pre-existing udev rules (e.g *enp3s0f0* netdev will

	change to *enp3s0f0np0*). If that is the case in your system, please make sure to update the following: - The *master* network device name in your MacvlanNetwork - The *ifNames* selector, if used in RDMA shared device plugin resource configuration - The *pfNames* selector, if used in SR-IOV device plugin configuration - If the sriov-network-operator is used, any instance of *SriovNetworkNodePolicy* which utilizes *NicSelector.PfNames* field should be updated to the new network device name. - When Network Operator 1.2.0 is installed via Helm, it no longer deploys both RDMA shared device plugin and SR-IOV network device plugin by default, as it may cause the same device to be registered to two different device plugins. This is an undesirable behavior. Instead, by default, only RDMA shared device plugin is deployed via Helm. If you wish to deploy both device plugins, set the *sriovDevicePlugin.deploy* Helm parameter to "true".
1.1.0	N/A
1.0.0	N/A

Bug Fixes

Version	Description
1.4.0	- Fixed a cluster scale-up issue. - Fixed an issue with IPoIB CNI deployment in OCP.
1.3.0	- N/A
1.2.0	- N/A
1.1.0	- Fixed the Whereabouts IPAM plugin to work with Kubernetes v1.22. - Fixed imagePullSecrets for Network Operator. - Enabled resource names for HostDeviceNetwork to be accepted both with and without a prefix.

Known Limitations

Version	Description
23.10.0	- IPoIB sub-interface creation does not work on RHEL 8.8 and RHEL 9.2 due to the kernel limitations in these distributions. This means that IPoIBNetwork cannot be used with these operating systems.
23.4.0	- In case that the UNLOAD_STORAGE_MODULES parameter is enabled for MOFED container deployment, it is required to make sure that the relevant storage modules are not in use in the OS.
23.1.0	- Only a single PKey can be configured per IPoIB workload pod.
1.4.0	- The operator upgrade procedure does not reflect configuration changes. The RDMA Shared Device Plugin or SR-IOV Device Plugin should be restarted manually in case of configuration changes. - The RDMA subsystem could be exclusive or shared only in one cluster. Mixed configuration is not supported. The RDMA Shared Device Plugin requires shared RDMA subsystem.
1.3.0	- MOFED container is not a supported configuration on the DGX platform. - MOFED container deletion may lead to the driver’s unloading: In this case, the mlx5_core kernel driver must be reloaded manually. Network connectivity could be affected if there are only NVIDIA NICs on the node.
1.2.0	- N/A
1.1.0	- NicClusterPolicy update is not supported at the moment. - Network Operator is compatible only with NVIDIA GPU Operator v1.9.0 and above. - GPUDirect could have performance degradation if it is used with servers which are not optimized. Please see official GPUDirect documentation here. - Persistent NICs configuration for netplan or ifupdown scripts is required for SR-IOV and Shared RDMA interfaces on the host. - POD Security Policy admission controller should be enabled to use PSP with Network Operator. Please see Deployment with Pod Security Policy in the

	Network Operator Documentation for details.
1.0.0	- Network Operator is only compatible with NVIDIA GPU Operator v1.5.2 and above. - Persistent NICs configuration for netplan or ifupdown scripts is required for SR-IOV and Shared RDMA interfaces on the host.

Platform Support

On this page

Prerequisites

Component	Version	Notes
Kubernetes	>=1.27 and <=1.29	
Helm	v3.5+	For information and methods of Helm installation, please refer to the official Helm Website.

Network Operator Component Matrix

The following component versions are deployed by the Network Operator:

Component	Version	Notes

Node Feature Discovery	v0.13.2	Optionally deployed. May already be present in the cluster with proper configuration.
NVIDIA MLNX_OFED driver container	24.04-0.6.6.0-0	
k8s-rdma-shared-device-plugin	1.4.0	
sriov-network-device-plugin	e6ead1e8f76a407783430ee2666b403db2d76f64	
containernetworking CNI plugins	v1.3.0	
whereabouts CNI	v0.7.0	
multus CNI	v3.9.3	
IPoIB CNI	428715a57c0b633e48ec7620f6e3af6863149ccf	
IB Kubernetes	v1.0.2	
NV IPAM Plugin	v0.1.2	

System Requirements

- NVIDIA RDMA-capable network adapters:
    - NVIDIA ConnectX NICs
        - ConnectX-5 or newer
    - NVIDIA BlueField Network Platforms
        - BlueField-2 DPU (NIC mode)
        - BlueField-3 DPU (NIC mode)
        - BlueField-3 SuperNIC (NIC mode)

- NVIDIA GPU Operator Version 24.3.x or newer (required for the workloads using NVIDIA GPUs and GPUDirect RDMA technology)

Tested Network Adapters

The following network adapters have been tested with the Network Operator:

- ConnectX-6 Dx
- ConnectX-7
- BlueField-2 NIC Mode
- BlueField-3 NIC Mode

Supported ARM Based Platforms

The following ARM based systems has been tested with Network Operator:

System	Network Adapters	OS	Notes
NVIDIA IGX Orin	ConnectX-7	Ubuntu 22.04 (ARM64)	GA (RoCE only, without GPUDirect RDMA)
NVIDIA Grace ARM Server	ConnectX-7	Ubuntu 22.04 (ARM64)	Tech Preview

Supported Operating Systems and Kubernetes Platforms

NVIDIA Network Operator has been validated in the following scenarios:

Operating System	Kubernetes	Red Hat OpenShift	Notes
Ubuntu 24.04 LTS	1.27-1.29		
Ubuntu 22.04 LTS	1.27-1.29		RT kernels support
Ubuntu 20.04 LTS	1.27-1.29		
Red Hat Core OS		4.12-4.15	RT kernels support

Red Hat Enterprise Linux 9.2, 9.0	1.27-1.29		
Red Hat Enterprise Linux 8.8, 8.6	1.27-1.29		RT kernels support

Supported Container Runtimes

NVIDIA Network Operator has been validated in the following scenarios:

Operating System	Containerd	CRI-O	Notes
Ubuntu 24.04 LTS	Yes	No	
Ubuntu 22.04 LTS	Yes	No	
Ubuntu 20.04 LTS	Yes	No	
Red Hat Core OS	No	Yes	
Red Hat Enterprise Linux 9	Yes	Yes	For containerd support DOCA/MOFED drivers must be pre-installed on host
Red Hat Enterprise Linux 8	Yes	Yes	For containerd support DOCA/MOFED drivers must be pre-installed on host

Getting Started with Kubernetes

On this page

Network Operator Deployment Guide

> ⚠ **Warning**
>
> The Network Operator Release Notes chapter is available here.

NVIDIA Network Operator leverages Kubernetes CRDs and Operator SDK to manage networking related components in order to enable fast networking, RDMA and GPUDirect for workloads in a Kubernetes cluster. The Network Operator works in conjunction with the GPU-Operator to enable GPU-Direct RDMA on compatible systems.

The goal of the Network Operator is to manage the networking related components, while enabling execution of RDMA and GPUDirect RDMA workloads in a Kubernetes cluster. This includes:

- NVIDIA Networking drivers to enable advanced features - enp1 netdcreate, an NV-IPAM IPPool
- Kubernetes device plugins to provide hardware resources required for an accelerated network
- Kubernetes secondary network components for network intensive workloads

Network Operator Deployment on Vanilla Kubernetes Cluster

⚠ **Warning**

It is recommended to have dedicated control plane nodes for Vanilla Kubernetes deployments with NVIDIA Network Operator.

The default installation via Helm as described below will deploy the Network Operator and related CRDs, after which an additional step is required to create a NicClusterPolicy custom resource with the configuration that is desired for the cluster.

For more information on NicClusterPolicy custom resource, please refer to the Network-Operator Project Sources.

The provided Helm chart contains various parameters to facilitate the creation of a NicClusterPolicy custom resource upon deployment.

⚠ **Warning**

Each Network Operator Release has a set of default version values for the various components it deploys. It is recommended that these values will not be changed. Testing and validation were performed with these values, and there is no guarantee of interoperability nor correctness when different versions are used.

Add NVIDIA NGC Helm repository

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
```

Update helm repositories

```
helm repo update
```

Install Network Operator from the NVIDIA NGC chart using the default values:

helm install network-operator nvidia/network-operator -n nvidia-network-operator --create-namespace --version v24.4.0 --wait *View deployed resources*

```
kubectl -n nvidia-network-operator get pods
```

Install the Network Operator from the NVIDIA NGC chart using custom values:

⚠ **Warning**

Since several parameters should be provided when creating custom resources during operator deployment, it is recommended to use a configuration file. While it is possible to override the parameters via CLI, we recommend to avoid the use of CLI arguments in favor of a configuration file.

helm show values nvidia/network-operator --version v24.4.0 > values.yaml helm install network-operator nvidia/network-operator -n nvidia-network-operator --create-namespace --version v24.4.0 -f ./values.yaml --wait

Deployment Examples

⚠ **Warning**

> Since several parameters should be provided when creating custom resources during operator deployment, it is recommended to use a configuration file. While it is possible to override the parameters via CLI, we recommend to avoid the use of CLI arguments in favor of a configuration file.

Below are deployment examples, which the `values.yaml` file provided to the Helm during the installation of the network operator. This was achieved by running:

```
helm install -f ./values.yaml -n nvidia-network-operator --create-namespace --wait
nvidia/network-operator network-operator
```

Network Operator Deployment with RDMA Shared Device Plugin

Network operator deployment with the default version of the OFED driver and a single RDMA resource mapped to ens1f0 netdev.:

`values.yaml` configuration file for such a deployment:

```
nfd: enabled: true sriovNetworkOperator: enabled: false # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: true rdmaSharedDevicePlugin: deploy: true
resources: - name: rdma_shared_device_a ifNames: [ens1f0] sriovDevicePlugin:
deploy: false
```

Network Operator Deployment with Multiple Resources in RDMA Shared Device Plugin

Network Operator deployment with the default version of OFED and an RDMA device plugin with two RDMA resources. The first is mapped to ens1f0 and ens1f1, and the second is mapped to ens2f0 and ens2f1.

`values.yaml` configuration file for such a deployment:

> nfd: enabled: true sriovNetworkOperator: enabled: false # *NicClusterPolicy CR values:* deployCR: true ofedDriver: deploy: true rdmaSharedDevicePlugin: deploy: true resources: - name: rdma_shared_device_a ifNames: [ens1f0, ens1f1] - name: rdma_shared_device_b ifNames: [ens2f0, ens2f1] sriovDevicePlugin: deploy: false

Network Operator Deployment with a Secondary Network

Network Operator deployment with:

- RDMA shared device plugin
- Secondary network
- Mutlus CNI
- Container-networking-plugins CNI plugins
- Whereabouts IPAM CNI Plugin

`values.yaml`:

> nfd: enabled: true sriovNetworkOperator: enabled: false # *NicClusterPolicy CR values:* deployCR: true ofedDriver: deploy: false rdmaSharedDevicePlugin: deploy: true resources: - name: rdma_shared_device_a ifNames: [ens1f0] secondaryNetwork: deploy: true multus: deploy: true cniPlugins: deploy: true ipamPlugin: deploy: true

Network Operator Deployment with NVIDIA-IPAM

Network Operator deployment with:

- RDMA shared device plugin
- Secondary network
- Multus CNI
- Container-networking-plugins
- CNI plugins

- NVIDIA-IPAM CNI Plugin

`values.yaml`:

```
nfd: enabled: true sriovNetworkOperator: enabled: false # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: false rdmaSharedDevicePlugin: deploy: true
resources: - name: rdma_shared_device_a ifNames: [ens1f0] secondaryNetwork:
deploy: true multus: deploy: true cniPlugins: deploy: true ipamPlugin: deploy: false
nvIpam: deploy: true
```

To create an NV-IPAM IPPool, apply:

```
apiVersion: nv-ipam.nvidia.com/v1alpha1 kind: IPPool metadata: name: my-pool
namespace: nvidia-network-operator spec: subnet: 192.168.0.0/24
perNodeBlockSize: 100 gateway: 192.168.0.1
```

Example of a MacvlanNetwork that uses NVIDIA-IPAM:

```
apiVersion: mellanox.com/v1alpha1 kind: MacvlanNetwork metadata: name:
example-macvlannetwork spec: networkNamespace: "default" master: "ens2f0"
mode: "bridge" mtu: 1500 ipam: | { "type": "nv-ipam", "poolName": "my-pool" }
```

Network Operator Deployment with a Host Device Network

Network Operator deployment with:

- SR-IOV device plugin, single SR-IOV resource pool
- Secondary network
- Multus CNI
- Container-networking-plugins CNI plugins
- Whereabouts IPAM CNI plugin

In this mode, the Network Operator could be deployed on virtualized deployments as well. It supports both Ethernet and InfiniBand modes. From the Network Operator perspective, there is no difference between the deployment procedures. To work on a VM (virtual machine), the PCI passthrough must be configured for SR-IOV devices. The Network Operator works both with VF (Virtual Function) and PF (Physical Function) inside the VMs.

> ⚠ **Warning**
>
> If the Host Device Network is used without the MLNX_OFED driver, the following packages should be installed:
>
> - the linux-generic package on Ubuntu hosts
> - the kernel-modules-extra package on the RedHat-based hosts

`values.yaml`:

```
nfd: enabled: true sriovNetworkOperator: enabled: false # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: false rdmaSharedDevicePlugin: deploy: false
sriovDevicePlugin: deploy: true resources: - name: hostdev vendors: [15b3]
secondaryNetwork: deploy: true multus: deploy: true cniPlugins: deploy: true
ipamPlugin: deploy: true
```

Following the deployment, the network operator should be configured, and K8s networking should be deployed to use it in pod configuration.

The `host-device-net.yaml` configuration file for such a deployment:

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
hostdev-net spec: networkNamespace: "default" resourceName:
"nvidia.com/hostdev" ipam: | { "type": "whereabouts", "datastore": "kubernetes",
"kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range":
```

```
"192.168.3.225/28", "exclude": [ "192.168.3.229/30", "192.168.3.236/32" ], "log_file":
"/var/log/whereabouts.log", "log_level": "info" }
```

The `host-device-net-ocp.yaml` configuration file for such a deployment in the OpenShift Platform:

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
hostdev-net spec: networkNamespace: "default" resourceName:
"nvidia.com/hostdev" ipam: | { "type": "whereabouts", "range": "192.168.3.225/28",
"exclude": [ "192.168.3.229/30", "192.168.3.236/32" ] }
```

The `pod.yaml` configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: hostdev-test-pod annotations:
k8s.v1.cni.cncf.io/networks: hostdev-net spec: restartPolicy: OnFailure containers: -
image: name: mofed-test-ctr securityContext: capabilities: add: [ "IPC_LOCK" ]
resources: requests: nvidia.com/hostdev: 1 limits: nvidia.com/hostdev: 1 command:
- sh - -c - sleep inf
```

Network Operator Deployment with a Host Device Network and Macvlan Network

In this combined deployment, different NVIDIA NICs are used for RDMA Shared Device Plugin and SR-IOV Network Device Plugin in order to work with a Host Device Network or a Macvlan Network on different NICs. It is impossible to combine different networking types on the same NICs. The same principle should be applied for other networking combinations.

`values.yaml`:

```
nfd: enabled: true # NicClusterPolicy CR values: deployCR: true ofedDriver: deploy:
false rdmaSharedDevicePlugin: deploy: true resources: - name:
rdma_shared_device_a linkTypes: [ether] sriovDevicePlugin: deploy: true resources: -
name: hostdev linkTypes: [“infiniband”] secondaryNetwork: deploy: true multus:
deploy: true cniPlugins: deploy: true ipamPlugin: deploy: true
```

For pods and network configuration examples please refer to the corresponding sections: Network Operator Deployment with the RDMA Shared Device Plugin and Network Operator Deployment with a Host Device Network.

Network Operator Deployment with an IP over InfiniBand (IPoIB) Network

Network Operator deployment with:

- RDMA shared device plugin
- Secondary network
- Multus CNI
- IPoIB CNI
- Whereabouts IPAM CNI plugin

In this mode, the Network Operator could be deployed on virtualized deployments as well. It supports both Ethernet and InfiniBand modes. From the Network Operator perspective, there is no difference between the deployment procedures. To work on a VM (virtual machine), the PCI passthrough must be configured for SR-IOV devices. The Network Operator works both with VF (Virtual Function) and PF (Physical Function) inside the VMs.

`values.yaml`:

```
nfd: enabled: true sriovNetworkOperator: enabled: false # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: true rdmaSharedDevicePlugin: deploy: true
resources: - name: rdma_shared_device_a ifNames: [ibs1f0] secondaryNetwork:
deploy: true multus: deploy: true ipoib: deploy: true ipamPlugin: deploy: true
```

Following the deployment, the network operator should be configured, and K8s networking deployed to use it in the pod configuration.

The `ipoib-net.yaml` configuration file for such a deployment:

```
apiVersion: mellanox.com/v1alpha1 kind: IPoIBNetwork metadata: name: example-ipoibnetwork spec: networkNamespace: "default" master: "ibs1f0" ipam: | { "type": "whereabouts", "datastore": "kubernetes", "kubernetes": { "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range": "192.168.5.225/28", "exclude": [ "192.168.6.229/30", "192.168.6.236/32" ], "log_file" : "/var/log/whereabouts.log", "log_level" : "info", "gateway": "192.168.6.1" }
```

The `ipoib-net-ocp.yaml` configuration file for such a deployment in the OpenShift Platform:

```
apiVersion: mellanox.com/v1alpha1 kind: IPoIBNetwork metadata: name: example-ipoibnetwork spec: networkNamespace: "default" master: "ibs1f0" ipam: | { "type": "whereabouts", "range": "192.168.5.225/28", "exclude": [ "192.168.6.229/30", "192.168.6.236/32" ] }
```

The `pod.yaml` configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: iboip-test-pod annotations: k8s.v1.cni.cncf.io/networks: example-ipoibnetwork spec: restartPolicy: OnFailure containers: - image: name: mofed-test-ctr securityContext: capabilities: add: [ "IPC_LOCK" ] resources: requests: rdma/rdma_shared_device_a: 1 limits: edma/rdma_shared_device_a: 1 command: - sh - -c - sleep inf
```

Network Operator Deployment for GPUDirect Workloads

GPUDirect requires the following:

- MLNX_OFED v5.5-1.0.3.2 or newer
- GPU Operator v1.9.0 or newer
- NVIDIA GPU and driver supporting GPUDirect e.g Quadro RTX 6000/8000 or NVIDIA T4/NVIDIA V100/NVIDIA A100

`values.yaml` example:

```
nfd: enabled: true sriovNetworkOperator: enabled: false # NicClusterPolicy CR values:
ofedDriver: deploy: true deployCR: true sriovDevicePlugin: deploy: true resources: -
name: hostdev vendors: [15b3] secondaryNetwork: deploy: true multus: deploy:
true cniPlugins: deploy: true ipamPlugin: deploy: true
```

`host-device-net.yaml:`

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
hostdevice-net spec: networkNamespace: "default" resourceName: "hostdev" ipam:
| { "type": "whereabouts", "datastore": "kubernetes", "kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range":
"192.168.3.225/28", "exclude": [ "192.168.3.229/30", "192.168.3.236/32" ], "log_file" :
"/var/log/whereabouts.log", "log_level" : "info" }
```

The `host-device-net-ocp.yaml` configuration file for such a deployment in the OpenShift Platform:

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
hostdevice-net spec: networkNamespace: "default" resourceName: "hostdev" ipam:
| { "type": "whereabouts", "range": "192.168.3.225/28", "exclude": [
"192.168.3.229/30", "192.168.3.236/32" ] }
```

`host-net-gpudirect-pod.yaml`

```
apiVersion: v1 kind: Pod metadata: name: testpod1 annotations:
k8s.v1.cni.cncf.io/networks: hostdevice-net spec: containers: - name: appcntr1
image: <image> imagePullPolicy: IfNotPresent securityContext: capabilities: add:
["IPC_LOCK"] command: - sh - -c - sleep inf resources: requests: nvidia.com/hostdev:
'1' nvidia.com/gpu: '1' limits: nvidia.com/hostdev: '1' nvidia.com/gpu: '1'
```

Network Operator Deployment in SR-IOV Legacy Mode

⚠ **Warning**

The SR-IOV Network Operator will be deployed with the default configuration. You can override these settings using a CLI argument, or the ‘sriov-network-operator’ section in the values.yaml file. For more information, refer to the Project Documentation.

⚠ **Warning**

This deployment mode supports SR-IOV in legacy mode.

`values.yaml` configuration for such a deployment:

```
nfd: enabled: true sriovNetworkOperator: enabled: true # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: true rdmaSharedDevicePlugin: deploy: false
sriovDevicePlugin: deploy: false secondaryNetwork: deploy: true multus: deploy:
true cniPlugins: deploy: true ipamPlugin: deploy: true
```

Following the deployment, the Network Operator should be configured, and sriovnetwork node policy and K8s networking should be deployed.

The `sriovnetwork-node-policy.yaml` configuration file for such a deployment:

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkNodePolicy metadata:
name: policy-1 namespace: nvidia-network-operator spec: deviceType: netdevice
mtu: 1500 nicSelector: vendor: "15b3" pfNames: ["ens2f0"] nodeSelector:
feature.node.kubernetes.io/pci-15b3.present: "true" numVfs: 8 priority: 90 isRdma:
true resourceName: sriov_resource
```

The `sriovnetwork.yaml` configuration file for such a deployment:

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetwork metadata: name:
"example-sriov-network" namespace: nvidia-network-operator spec: vlan: 0
```

```
networkNamespace: "default" resourceName: "sriov_resource" ipam: |- {
"datastore": "kubernetes", "kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "log_file":
"/tmp/whereabouts.log", "log_level": "debug", "type": "whereabouts", "range":
"192.168.101.0/24" }
```

⚠ Warning

The ens2f0 network interface name has been chosen from the following command output: `kubectl -n nvidia-network-operator get sriovnetworknodestates.sriovnetwork.openshift.io -o yaml`

.

```
... status: interfaces: - deviceID: 101d driver: mlx5_core linkSpeed: 100000 Mb/s
linkType: ETH mac: 0c:42:a1:2b:74:ae mtu: 1500 name: ens2f0 pciAddress:
"0000:07:00.0" totalvfs: 8 vendor: 15b3 - deviceID: 101d driver: mlx5_core linkType:
ETH mac: 0c:42:a1:2b:74:af mtu: 1500 name: ens2f1 pciAddress: "0000:07:00.1"
totalvfs: 8 vendor: 15b3 ...
```

Wait for all required pods to be spawned:

```
# kubectl get pod -n nvidia-network-operator | grep sriov network-operator-sriov-
network-operator-544c8dbbb9-vzkmc 1/1 Running 0 5d sriov-device-plugin-vwpzn
1/1 Running 0 2d6h sriov-network-config-daemon-qv467 3/3 Running 0 5d # kubectl
get pod -n nvidia-network-operator NAME READY STATUS RESTARTS AGE cni-plugins-
ds-kbvnm 1/1 Running 0 5d cni-plugins-ds-pcllg 1/1 Running 0 5d kube-multus-ds-
5j6ns 1/1 Running 0 5d kube-multus-ds-mxgvl 1/1 Running 0 5d mofed-
ubuntu20.04-ds-2zzf4 1/1 Running 0 5d mofed-ubuntu20.04-ds-rfnsw 1/1 Running 0
5d whereabouts-nw7hn 1/1 Running 0 5d whereabouts-zvhrv 1/1 Running 0 5d ...
```

The `pod.yaml` configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: testpod1 annotations:
k8s.v1.cni.cncf.io/networks: example-sriov-network spec: containers: - name:
appcntr1 image: <image> imagePullPolicy: IfNotPresent securityContext:
capabilities: add: ["IPC_LOCK"] resources: requests: nvidia.com/sriov_resource: '1'
limits: nvidia.com/sriov_resource: '1' command: - sh - -c - sleep inf
```

SR-IOV Network Operator Deployment – Parallel Node Configuration for SR-IOV

⚠ **Warning**

This feature is supported only for Vanilla Kubernetes deployments with SR-IOV Network Operator.

To apply SR-IOV configuration on several nodes in parallel, create a `SriovNetworkPoolConfig` CR and specify the maximum number or percentage of nodes that can be unavailable at the same time:

`sriov-network-pool-config-number.yaml`

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkPoolConfig metadata:
name: pool-1 namespace: network-operator spec: maxUnavailable: "20"
nodeSelector: - matchExpressions: - key: some-label operator: In values: - val-2 -
matchExpressions: - key: other-label operator: "Exists"
```

`sriov-network-pool-config-percent.yaml`

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkPoolConfig metadata:
name: pool-1 namespace: network-operator spec: maxUnavailable: "10%"
nodeSelector: - matchExpressions: - key: some-label operator: In values: - val-2 -
matchExpressions: - key: other-label operator: "Exists"
```

Upgrade from NVIDIA Network Operator v24.1.0

To upgrade SR-IOV Network operator you need to create `SriovNetworkPoolConfig` CR with the number of nodes to be configured in a parallel as we did in *SriovOperatorConfig*` in previous releases.

E.g.: old method to configure nodes in a parallel:

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n network-operator
default --patch '{ "spec": { "maxParallelNodeConfiguration": 5 } }' --type='merge'
```

New method to configure nodes in a parallel:

`sriov-network-pool-config-new.yaml`

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkPoolConfig metadata:
name: pool-1 namespace: network-operator spec: maxUnavailable: "5"
nodeSelector: - matchExpressions: - key: node-role.kubernetes.io/master operator:
Exists
```

SR-IOV Network Operator Deployment – Parallel NIC Configuration for SR-IOV

⚠ **Warning**

This feature is supported only for Vanilla Kubernetes deployments with SR-IOV Network Operator.

To apply SriovNetworkNodePolicy on several nodes in parallel, specify the `featureGates` option in the SriovOperatorConfig CRD:

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n network-operator
default --patch '{ "spec": { "featureGates": { "parallelNicConfig": true } } }' --
```

```
type='merge'
```

SR-IOV Network Operator Deployment – SR-IOV Using the systemd Service

To enable systemd SR-IOV configuration mode, specify the configurationMode option in the SriovOperatorConfig CRD:

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n network-operator
default --patch '{ "spec": { "configurationMode": "systemd"} }' --type='merge'
```

Network Operator Deployment with an SR-IOV InfiniBand Network

Network Operator deployment with InfiniBand network requires the following:

- MLNX_OFED and OpenSM running. OpenSM runs on top of the MLNX_OFED stack, so both the driver and the subnet manager should come from the same installation. Note that partitions that are configured by OpenSM should specify defmember=full to enable the SR-IOV functionality over InfiniBand. For more details, please refer to *this article <https://docs.mellanox.com/display/MLNXOFEDv51258060/OpenSM>*.

- InfiniBand device – Both the host device and switch ports must be enabled in InfiniBand mode.

- rdma-core package should be installed when an inbox driver is used.

values.yaml

```
nfd: enabled: true sriovNetworkOperator: enabled: true # NicClusterPolicy CR values:
deployCR: true ofedDriver: deploy: true rdmaSharedDevicePlugin: deploy: false
sriovDevicePlugin: deploy: false secondaryNetwork: deploy: true multus: deploy:
true cniPlugins: deploy: true ipamPlugin: deploy: true
```

sriov-ib-network-node-policy.yaml

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkNodePolicy metadata:
name: infiniband-sriov namespace: nvidia-network-operator spec: deviceType:
```

```
netdevice mtu: 1500 nodeSelector: feature.node.kubernetes.io/pci-15b3.present:
"true" nicSelector: vendor: "15b3" linkType: infiniband isRdma: true numVfs: 8
priority: 90 resourceName: mlnxnics
```

`sriov-ib-network.yaml`

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovIBNetwork metadata: name:
example-sriov-ib-network namespace: nvidia-network-operator spec: ipam: | {
"type": "whereabouts", "datastore": "kubernetes", "kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range":
"192.168.5.225/28", "exclude": [ "192.168.5.229/30", "192.168.5.236/32" ], "log_file":
"/var/log/whereabouts.log", "log_level": "info" } resourceName: mlnxnics linkState:
enable networkNamespace: default
```

`sriov-ib-network-pod.yaml`

```
apiVersion: v1 kind: Pod metadata: name: test-sriov-ib-pod annotations:
k8s.v1.cni.cncf.io/networks: example-sriov-ib-network spec: containers: - name: test-
sriov-ib-pod image: centos/tools imagePullPolicy: IfNotPresent command: - sh - -c -
sleep inf securityContext: capabilities: add: [ "IPC_LOCK" ] resources: requests:
nvidia.com/mlnxics: "1" limits: nvidia.com/mlnxics: "1"
```

Network Operator Deployment with an SR-IOV InfiniBand Network with PKey Management

Network Operator deployment with InfiniBand network requires the following:

- MLNX_OFED and OpenSM running. OpenSM runs on top of the MLNX_OFED stack, so both the driver and the subnet manager should come from the same installation. Note that partitions that are configured by OpenSM should specify defmember=full to enable the SR-IOV functionality over InfiniBand. For more details, please refer to this article.

- NVIDIA UFM running on top of OpenSM. For more details, please refer to the project documentation.

- InfiniBand device – Both the host device and the switch ports must be enabled in InfiniBand mode.
- rdma-core package should be installed when an inbox driver is used.

Current limitations:

- Only a single PKey can be configured per workload pod.
- When a single instance of NVIDIA UFM is used with several K8s clusters, different PKey GUID pools should be configured for each cluster.

> **Warning**
>
> *ib-kubernetes-ufm-secret* should be created before NicClusterPolicy.

ufm-secret.yaml

```
apiVersion: v1 kind: Secret metadata: name: ib-kubernetes-ufm-secret namespace:
nvidia-network-operator stringData: UFM_USERNAME: "admin" UFM_PASSWORD:
"123456" UFM_ADDRESS: "ufm-host" UFM_HTTP_SCHEMA: "" UFM_PORT: "" data:
UFM_CERTIFICATE: ""
```

values.yaml

```
nfd: enabled: true sriovNetworkOperator: enabled: true resourcePrefix:
"nvidia.com" # NicClusterPolicy CR values: deployCR: true ofedDriver: deploy: true
rdmaSharedDevicePlugin: deploy: false sriovDevicePlugin: deploy: false
ibKubernetes: deploy: true periodicUpdateSeconds: 5 pKeyGUIDPoolRangeStart:
"02:00:00:00:00:00:00:00" pKeyGUIDPoolRangeEnd: "02:FF:FF:FF:FF:FF:FF:FF"
ufmSecret: ufm-secret secondaryNetwork: deploy: true multus: deploy: true
cniPlugins: deploy: true ipamPlugin: deploy: true
```

Wait for MLNX_OFED to install and apply the following CRs:

sriov-ib-network-node-policy.yaml

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkNodePolicy metadata:
name: infiniband-sriov namespace: nvidia-network-operator spec: deviceType:
netdevice mtu: 1500 nodeSelector: feature.node.kubernetes.io/pci-15b3.present:
"true" nicSelector: vendor: "15b3" linkType: ib isRdma: true numVfs: 8 priority: 90
resourceName: mlnxnics
```

sriov-ib-network.yaml

```
apiVersion: "k8s.cni.cncf.io/v1" kind: NetworkAttachmentDefinition metadata: name:
ib-sriov-network annotations: k8s.v1.cni.cncf.io/resourceName: nvidia.com/mlnxnics
spec: config: '{ "type":"ib-sriov", "cniVersion":"0.3.1", "name":"ib-sriov-network",
"pkey":"0x6", "link_state":"enable", "ibKubernetesEnabled":true, "ipam":{
"type":"whereabouts", "datastore":"kubernetes", "kubernetes":{
"kubeconfig":"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" },
"range":"10.56.217.0/24", "log_file":"/var/log/whereabouts.log", "log_level":"info" } }'
```

sriov-ib-network-pod.yaml

```
apiVersion: v1 kind: Pod metadata: name: test-sriov-ib-pod annotations:
k8s.v1.cni.cncf.io/networks: ib-sriob-network spec: containers: - name: test-sriov-ib-
pod image: centos/tools imagePullPolicy: IfNotPresent command: - sh - -c - sleep inf
securityContext: capabilities: add: [ "IPC_LOCK" ] resources: requests:
nvidia.com/mlnxics: "1" limits: nvidia.com/mlnxics: "1"
```

Network Operator Deployment for DPDK Workloads with NicClusterPolicy

This deployment mode supports DPDK applications. In order to run DPDK applications, HUGEPAGE should be configured on the required K8s Worker Nodes. By default, the inbox operating system driver is used. For support of cases with specific requirements, OFED container should be deployed.

Network Operator deployment with:

- Host Device Network
- DPDK pod

nicclusterpolicy.yaml

```
apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata: name: nic-cluster-
policy spec: ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox version:
24.04-0.6.6.0-0 sriovDevicePlugin: image: sriov-network-device-plugin repository:
ghcr.io/k8snetworkplumbingwg version: e6ead1e8f76a407783430ee2666b403db2d76f64
config: | { "resourceList": [ { "resourcePrefix": "nvidia.com", "resourceName":
"rdma_host_dev", "selectors": { "vendors": ["15b3"], "devices": ["1018"], "drivers":
["mlx5_core"] } } ] } secondaryNetwork: cniPlugins: image: plugins repository:
ghcr.io/k8snetworkplumbingwg version: v1.3.0-amd64 ipamPlugin: image: whereabouts
repository: ghcr.io/k8snetworkplumbingwg version: v0.7.0-amd64 multus: image: multus-
cni repository: ghcr.io/k8snetworkplumbingwg version: v3.9.3
```

host-device-net.yaml

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
example-hostdev-net spec: networkNamespace: "default" resourceName:
"rdma_host_dev" ipam: | { "type": "whereabouts", "datastore": "kubernetes",
"kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range":
"192.168.3.225/28", "exclude": [ "192.168.3.229/30", "192.168.3.236/32" ], "log_file" :
"/var/log/whereabouts.log", "log_level" : "info" }
```

pod.yaml

```
apiVersion: v1 kind: Pod metadata: name: testpod1 annotations:
k8s.v1.cni.cncf.io/networks: example-hostdev-net spec: containers: - name:
appcntr1 image: <dpdk image> imagePullPolicy: IfNotPresent securityContext:
capabilities: add: ["IPC_LOCK"] volumeMounts: - mountPath: /dev/hugepages name:
hugepage resources: requests: memory: 1Gi hugepages-1Gi: 2Gi
nvidia.com/rdma_host_dev: '1' command: [ "/bin/bash", "-c", "--" ] args: [
"whiletrue;dosleep300000;done;" ] volumes: - name: hugepage emptyDir: medium:
HugePages
```

Network Operator Deployment and OpenvSwitch offload

> ⚠ **Warning**
>
> This feature is supported only for Vanilla Kubernetes deployments with SR-IOV Network Operator.

> ⚠ **Warning**
>
> This mode of operation is not compatible with OFED container.

> ⚠ **Warning**
>
> Tech Preview feature.

Network Operator Configuration

Deploy network-operator by Helm with sriov-network-operator and nv-ipam.

`values.yaml`

```
sriovNetworkOperator: enabled: true deployCR: true nvIpam: deploy: true
```

Enable `manageSoftwareBridges` featureGate for sriov-network-operator

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n network-operator
default --patch '{ "spec": { "featureGates": { "manageSoftwareBridges": true } } }' --
```

```
type='merge'
```

Create IPPool object for nv-ipam

```
apiVersion: nv-ipam.nvidia.com/v1alpha1 kind: IPPool metadata: name: pool1
namespace: network-operator spec: subnet: 192.168.0.0/16 perNodeBlockSize: 100
gateway: 192.168.0.1 nodeSelector: nodeSelectorTerms: - matchExpressions: - key:
node-role.kubernetes.io/worker operator: Exists
```

Prerequisites for Worker Nodes

Supported operating systems:

- Ubuntu 22.04

OpenvSwitch from the `NVIDIA DOCA for Host` package with `doca-all` or `doca-networking` profile should be installed on each worker node.

Check NVIDIA DOCA Official installation guide for details.

Supported OpenvSwitch dataplanes:

- OVS-kernel
- OVS-doca

Check OpenvSwitch Offload document to know about differences.

OVS-kernel

These steps are for OVS-kernel data plane, to use OVS-doca follow instructions from the relevant section.

Prepare Worker Nodes

Configure Open_vSwitch

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

Restart Open_vSwitch

```
systemctl restart openvswitch-switch.service
```

Sriov Network Operator Configuration

Create SriovNetworkNodePolicy for selected NIC

```
kind: SriovNetworkNodePolicy metadata: name: ovs-switchdev namespace:
network-operator spec: eSwitchMode: switchdev mtu: 1500 nicSelector: deviceID:
101d vendor: 15b3 nodeSelector: node-role.kubernetes.io/worker: "" numVfs: 4
isRdma: true linkType: ETH resourceName: switchdev bridge: ovs: {}
```

Create OVSNetwork CR

```
apiVersion: sriovnetwork.openshift.io/v1 kind: OVSNetwork metadata: name: ovs
namespace: network-operator spec: networkNamespace: default ipam: | { "type":
"nv-ipam", "poolName": "pool1" } resourceName: switchdev
```

OVS-doca

These steps are for OVS-doca data plane, to use OVS-kernel follow instructions from the relevant section.

Prepare Worker Nodes

Configure hugepages

```
mkdir -p /hugepages mount -t hugetlbfs hugetlbfs /hugepages echo 4096 >
/sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

Note: for multi CPU system hugepages should be created for each NUMA node: node0, node1, ...

Configure system to create hugepages on boot

```
echo "vm.nr_hugepages=8192" > /etc/sysctl.d/99-hugepages.conf
```

Note: this example is for a server with two CPU

Configure Open_vSwitch

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:doca-init=true ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

Restart Open_vSwitch

```
systemctl restart openvswitch-switch.service
```

Sriov Network Operator Configuration

Create SriovNetworkNodePolicy for selected NIC

```
kind: SriovNetworkNodePolicy metadata: name: ovs-switchdev namespace: network-operator spec: eSwitchMode: switchdev mtu: 1500 nicSelector: deviceID: 101d vendor: 15b3 nodeSelector: node-role.kubernetes.io/worker: "" numVfs: 4 isRdma: true linkType: ETH resourceName: switchdev bridge: ovs: bridge: datapathType: netdev uplink: interface: type: dpdk
```

Create OVSNetwork CR

```
apiVersion: sriovnetwork.openshift.io/v1 kind: OVSNetwork metadata: name: ovs namespace: network-operator spec: networkNamespace: default ipam: | { "type": "nv-ipam", "poolName": "pool1" } resourceName: switchdev interfaceType: dpdk
```

Test Workload

```
apiVersion: apps/v1 kind: Deployment metadata: name: ovs-offload labels: app: ovs-offload spec: replicas: 2 selector: matchLabels: app: ovs-offload template: metadata: labels: app: ovs-offload annotations: k8s.v1.cni.cncf.io/networks: ovs spec:
```

```
containers: - name: ovs-offload-container command: ["/bin/bash", "-c"] args: - |
while true; do sleep 1000; done image: mellanox/rping-test securityContext:
capabilities: add: ["IPC_LOCK"] resources: requests: nvidia.com/switchdev: 1 limits:
nvidia.com/switchdev: 1
```

Troubleshooting OVS

For OVS hardware offload verification and troubleshooting steps, please refer to the following DOCA documentation:

- OVS-Kernel Hardware Offloads
- OVS-DOCA Hardware Offloads

Getting Started with Red Hat OpenShift

On this page

Network Operator Deployment on an OpenShift Container Platform

> ⚠ **Warning**
>
> Currently, NVIDIA Network Operator does not support Single Node OpenShift (SNO) deployments.

> ⚠ **Warning**
>
> It is recommended to have dedicated control plane nodes for OpenShift deployments with NVIDIA Network Operator.

Node Feature Discovery

To enable Node Feature Discovery, please follow the official guide.

An example of Node Feature Discovery configuration:

```
apiVersion: nfd.openshift.io/v1 kind: NodeFeatureDiscovery metadata: name: nfd-
instance namespace: openshift-nfd spec: operand: namespace: openshift-nfd
image: registry.redhat.io/openshift4/ose-node-feature-discovery:v4.10
imagePullPolicy: Always workerConfig: configData: | sources: pci:
deviceClassWhitelist: - "02" - "03" - "0200" - "0207" deviceLabelFields: - vendor
customConfig: configData: ""
```

Verify that the following label is present on the nodes containing NVIDIA networking hardware: *feature.node.kubernetes.io/pci-15b3.present=true*

```
oc describe node | grep -E 'Roles|pci' | grep -v "control-plane" Roles: worker cpu-
feature.node.kubevirt.io/invpcid=true cpu-feature.node.kubevirt.io/pcid=true
feature.node.kubernetes.io/pci-102b.present=true feature.node.kubernetes.io/pci-
10de.present=true feature.node.kubernetes.io/pci-10de.sriov.capable=true
feature.node.kubernetes.io/pci-14e4.present=true feature.node.kubernetes.io/pci-
15b3.present=true feature.node.kubernetes.io/pci-15b3.sriov.capable=true Roles:
worker cpu-feature.node.kubevirt.io/invpcid=true cpu-
feature.node.kubevirt.io/pcid=true feature.node.kubernetes.io/pci-
102b.present=true feature.node.kubernetes.io/pci-10de.present=true
feature.node.kubernetes.io/pci-10de.sriov.capable=true
feature.node.kubernetes.io/pci-14e4.present=true feature.node.kubernetes.io/pci-
15b3.present=true feature.node.kubernetes.io/pci-15b3.sriov.capable=true
```

SR-IOV Network Operator

If you are planning to use SR-IOV, follow these instructions to install SR-IOV Network Operator on an OpenShift Container Platform.

⚠ Warning

The SR-IOV resources created will have the *openshift.io* prefix.

For the default SriovOperatorConfig CR to work with the MLNX_OFED container, please run this command to update the following values:

```
oc patch sriovoperatorconfig default \ --type=merge -n openshift-sriov-network-
operator \ --patch '{ "spec": { "configDaemonNodeSelector": {
"network.nvidia.com/operator.mofed.wait": "false", "node-
role.kubernetes.io/worker": "", "feature.node.kubernetes.io/pci-15b3.sriov.capable":
"true" } } }'
```

⚠ **Warning**

SR-IOV Network Operator configuration documentation can be found on the Official Website.

GPU Operator

If you plan to use GPUDirect, follow this to install GPU Operator on an OpenShift Container Platform.

Make sure to enable RDMA and disable useHostMofed in the driver section in the spec of the ClusterPolicy CR.

Network Operator Installation

Network Operator Installation Using OpenShift Catalog

- In the OpenShift Container Platform web console side menu, select Operators > OperatorHub, and search for the NVIDIA Network Operator.
- Select NVIDIA Network Operator, and click Install in the first screen and in the subsequent one.
- For additional information, see the Red Hat OpenShift Container Platform Documentation.

Network Operator Installation using OpenShift OC CLI

1. Create a namespace for the Network Operator.

```
oc create namespace nvidia-network-operator
```

2. Install the Network Operator in the namespace created in the previous step by creating the below objects. Run the following command to get the channel value required for the next step:

```
oc get packagemanifest nvidia-network-operator -n openshift-marketplace -o jsonpath='{.status.defaultChannel}'
```

Example output:

```
stable
```

3. Create the following Subscription CR, and save the YAML in the network-operator-sub.yaml file:

   apiVersion: operators.coreos.com/v1alpha1 kind: Subscription metadata: name: nvidia-network-operator namespace: nvidia-network-operator spec: channel: "v24.4.0" installPlanApproval: Manual name: nvidia-network-operator source: certified-operators sourceNamespace: openshift-marketplace

4. Create the subscription object by running the following command:

   ```
   oc create -f network-operator-sub.yaml
   ```

5. Change to the network-operator project:

   ```
   oc project nvidia-network-operator
   ```

Verification

To verify that the operator deployment is successful, run:

```
oc get pods -n nvidia-network-operator
```

Example output:

```
NAME READY STATUS RESTARTS AGE nvidia-network-operator-controller-manager-8f8ccf45c-zgfsq 2/2 Running 0 1m
```

A successful deployment shows a *Running* status.

Using Network Operator to Create NicClusterPolicy in OpenShift Container Platform

See Deployment Examples for OCP:

Deployment Examples For OpenShift Container Platform

In OCP, some components are deployed by default like Multus and WhereAbouts, whereas others, such as NFD and SR-IOV Network Operator must be deployed manually, as described in the Installation section.

In addition, since there is no use of the Helm chart, the configuration should be done via the NicClusterPolicy CRD.

Following are examples of NicClusterPolicy configuration for OCP.

Network Operator Deployment with a Host Device Network - OCP

Network Operator deployment with:

SR-IOV device plugin, single SR-IOV resource pool:

- There is no need for a secondary network configuration, as it is installed by default in OCP.

apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata: name: nic-cluster-policy spec: ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox version: 24.04-0.6.6.0-0 startupProbe: initialDelaySeconds: 10 periodSeconds: 20 livenessProbe: initialDelaySeconds: 30 periodSeconds: 30 readinessProbe: initialDelaySeconds: 10 periodSeconds: 30 sriovDevicePlugin: image: sriov-network-device-plugin repository: ghcr.io/k8snetworkplumbingwg version: e6ead1e8f76a407783430ee2666b403db2d76f64 config: | { "resourceList": [{ "resourcePrefix": "nvidia.com", "resourceName": "hostdev", "selectors": { "vendors": ["15b3"], "isRdma": true } }] }

Following the deployment, the Network Operator should be configured, and K8s networking deployed to use it in pod configuration. The *host-device-net.yaml`* configuration file for such a deployment:

```
apiVersion: mellanox.com/v1alpha1 kind: HostDeviceNetwork metadata: name:
hostdev-net spec: networkNamespace: "default" resourceName:
```

```
"nvidia.com/hostdev" ipam: | { "type": "whereabouts", "datastore": "kubernetes",
"kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "range":
"192.168.3.225/28", "exclude": [ "192.168.3.229/30", "192.168.3.236/32" ], "log_file" :
"/var/log/whereabouts.log", "log_level" : "info" }
```

The *pod.yaml* configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: hostdev-test-pod annotations:
k8s.v1.cni.cncf.io/networks: hostdev-net spec: restartPolicy: OnFailure containers: -
image: <rdma image> name: mofed-test-ctr securityContext: capabilities: add: [
"IPC_LOCK" ] resources: requests: nvidia.com/hostdev: 1 limits: nvidia.com/hostdev:
1 command: - sh - -c - sleep inf
```

Network Operator Deployment with SR-IOV Legacy Mode - OCP

This deployment mode supports SR-IOV in legacy mode. Note that the SR-IOV Network Operator is required as described in the Deployment for OCP section.

apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata: name: nic-cluster-policy spec: ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox version: 24.04-0.6.6.0-0 startupProbe: initialDelaySeconds: 10 periodSeconds: 20 livenessProbe: initialDelaySeconds: 30 periodSeconds: 30 readinessProbe: initialDelaySeconds: 10 periodSeconds: 30

Sriovnetwork node policy and K8s networking should be deployed. *sriovnetwork-node-policy.yaml* configuration file for such a deployment:

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetworkNodePolicy metadata:
name: policy-1 namespace: openshift-sriov-network-operator spec: deviceType:
netdevice mtu: 1500 nicSelector: vendor: "15b3" pfNames: ["ens2f0"] nodeSelector:
feature.node.kubernetes.io/pci-15b3.present: "true" numVfs: 8 priority: 90 isRdma:
true resourceName: sriovlegacy
```

The *sriovnetwork.yaml* configuration file for such a deployment:

```
apiVersion: sriovnetwork.openshift.io/v1 kind: SriovNetwork metadata: name: "sriov-
network" namespace: openshift-sriov-network-operator spec: vlan: 0
networkNamespace: "default" resourceName: "sriovlegacy" ipam: |- { "datastore":
"kubernetes", "kubernetes": { "kubeconfig":
"/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig" }, "log_file":
"/tmp/whereabouts.log", "log_level": "debug", "type": "whereabouts", "range":
"192.168.101.0/24" }
```

Note that the resource prefix in this case will be *openshift.io*. The *pod.yaml* configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: testpod1 annotations:
k8s.v1.cni.cncf.io/networks: sriov-network spec: containers: - name: appcntr1 image:
<image> imagePullPolicy: IfNotPresent securityContext: capabilities: add:
["IPC_LOCK"] command: - sh - -c - sleep inf resources: requests:
openshift.io/sriovlegacy: '1' limits: openshift.io/sriovlegacy: '1'
```

Network Operator Deployment with the RDMA Shared Device Plugin - OCP

The following is an example of RDMA Shared with MacVlanNetwork:

```
apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata: name: nic-cluster-
policy spec: ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox version:
24.04-0.6.6.0-0 startupProbe: initialDelaySeconds: 10 periodSeconds: 20 livenessProbe:
initialDelaySeconds: 30 periodSeconds: 30 readinessProbe: initialDelaySeconds: 10
periodSeconds: 30 rdmaSharedDevicePlugin: config: | { "configList": [ { "resourceName":
"rdmashared", "rdmaHcaMax": 1000, "selectors": { "ifNames": ["enp4s0f0np0"] } } ] }
image: k8s-rdma-shared-dev-plugin repository: nvcr.io/nvidia/cloud-native version: 1.4.0
```

The *macvlan-net-ocp.yaml* configuration file for such a deployment in an OpenShift Platform:

```
apiVersion: mellanox.com/v1alpha1 kind: MacvlanNetwork metadata: name:
rdmashared-net spec: networkNamespace: default master: enp4s0f0np0 mode:
bridge mtu: 1500 ipam:
'{"type":"whereabouts","range":"16.0.2.0/24","gateway":"16.0.2.1"}'
```

The *pod.yaml* configuration file for such a deployment:

```
apiVersion: v1 kind: Pod metadata: name: test-rdma-shared-1 annotations:
k8s.v1.cni.cncf.io/networks: rdmashared-net spec: containers: - image: myimage
name: rdma-shared-1 securityContext: capabilities: add: - IPC_LOCK resources:
limits: rdma/rdmashared: 1 requests: rdma/rdmashared: 1 restartPolicy: OnFailure
```

Network Operator Deployment for DPDK Workloads - OCP

In order to configure *HUGEPAGES* in OpenShift, refer to this steps.

For SR-IOV Network Operator configuration instructions, visit the Official Website.

Customization Options

Helm Chart Customization Options

There are various customizations you can do to tailor the deployment of the Network Operator to your cluster needs. You can find those below.

- General Parameters
    - ImagePullSecrets customization
- NFD labels
- SR-IOV Network Operator
- Container Resources
- MLNX_OFED Driver
- MLNX_OFED Driver Environment Variables
- RDMA Shared Device Plugin
- RDMA Device Plugin Resource Configurations
- SR-IOV Network Device Plugin
- SR-IOV Network Device Plugin Resource Configuration
- IB Kubernetes
- UFM Secret
- Secondary Network
- CNI Plugin
- Multus CNI
- IPoIB CNI
- IPAM CNI Plugin
- NVIDIA IPAM Plugin
- NVIDIA NIC Feature Discovery

- DOCA Telemetry Service
- Helm customization file

General Parameters

Name	Type	Default	Description
operator.admissionController.enabled	Bool	False	Deploy with admission controller
operator.admissionController.useCertManager	Bool	True	Use cert-manager for generating self-signed certificate
operator.admissionController.certificate.tlsCrt	String	“”	External TLS certificate. Ignored if cert-manager is used
operator.admissionController.certificate.tlsKey	String	“”	External TLS private key. Ignored if cert-manager is used
nfd.enabled	Bool	True	Deploy Node Feature Discovery
nfd.deployNodeFeatureRules	Bool	True	Deploy Node Feature Rules to label the nodes
sriovNetworkOperator.enabled	Bool	False	Deploy SR-IOV Network Operator
sriovNetworkOperator.configDaemonNodeSelectorExtra	List	node-role.kubernetes.io/worker: “”	Additional values for SR-IOV Config Daemon nodes selector
upgradeCRDs	Bool	True	Enable CRDs upgrade with helm pre-install and pre-upgrade hooks

operator.repository	String	nvcr.io/nvidia	Network Operator image repository
operator.image	String	network-operator	Network Operator image name
operator.tag	String	None	Network Operator image tag. If set to `None`, the chart’s `appVersion` will be used
operator.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling Network Operator image
operator.cniBinDirectory	String	/opt/cni/bin	Directory, where CNI binaries will be deployed on the nodes. Setting for the sriov-network-operator is set with `sriov-network-operator.cniBinPath` parameter. Note that the CNI bin directory should be aligned with the CNI bin directory in the container runtime.
operator.resources	Yaml	resources: limits: cpu: 500m memory: 128Mi requests: cpu: 5m memory: 64Mi	Optional resource requests and limits for the operator

imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the Network Operator image, if it is not overridden
deployCR	Bool	False	Deploy `NicClusterPolicy` custom resource according to the provided parameters
nodeAffinity	Yaml	requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms: - matchExpressions: - key: node-role.kubernetes.io/master operator: DoesNotExist - key: node-role.kubernetes.io/control-plane operator: DoesNotExist	Configure node affinity settings for Network Operator components
tolerations	Yaml	“”	Set additional tolerations for

			various Daemonsets deployed by the network operator, e.g. whereabouts, multus, cni-plugins.
useDTK	Bool	True	Enable the use of Driver ToolKit to compile OFED drivers (OpenShift only)

ImagePullSecrets customization

To provide *imagePullSecrets*` object references, you need to specify them using a following structure:

```
imagePullSecrets: - image-pull-secret1 - image-pull-secret2
```

NFD labels

The NFD labels required by the Network Operator and GPU Operator:

Label	Location
feature.node.kubernetes.io/pci-15b3.present	Nodes containing NVIDIA Networking hardware
feature.node.kubernetes.io/pci-10de.present	Nodes containing NVIDIA GPU hardware

SR-IOV Network Operator

SR-IOV Network Operator Helm chart customization options can be found here. Following is a list of overriden values by NVIDIA Operator Helm Chart:

Name	Type	Default in NVIDIA Network Operator	Notes
sriov-network-operator.operator.re	String	nvidia.com	

sourcePrefix			
sriov-network-operator.images.operator	String	nvcr.io/nvidia/mellanox/sriov-network-operator:network-operator-24.4.0	
sriov-network-operator.images.sriovConfigDaemon	String	nvcr.io/nvidia/mellanox/sriov-network-operator-config-daemon:network-operator-24.4.0	
sriov-network-operator.images.sriovCni	String	ghcr.io/k8snetworkplumbingwg/sriov-cni:3e6368077716f6b8368b0e036a1290d1c64cf1fb	For ARM-based deployments, it is recommended to use the `ghcr.io/k8snetworkplumbingwg/sriov-cni:v2.8.0-arm64` image
sriov-network-operator.images.ibSriovCni	String	ghcr.io/k8snetworkplumbingwg/ib-sriov-cni:fc002af57a81855542759d0f77d16dacd7e1aa38	For ARM-based deployments, it is recommended to use the `ghcr.io/k8snetworkplumbingwg/ib-sriov-cni:1.1.0-arm64` image
sriov-network-operator.images.sriovDevicePlugin	String	ghcr.io/k8snetworkplumbingwg/sriov-network-device-plugin:e6ead1e8f76a407783430ee2666b403db2d76f64	For ARM-based deployments, it is recommended to use the `ghcr.io/k8snetworkplumbingwg/sriov-network-device-plugin:v3.6.2-arm64` image

sriov-network-operator.images.webhook	String	nvcr.io/nvidia/mellanox/sriov-network-operator-webhook:network-operator-24.4.0	

Container Resources

Optional requests and limits can be configured for each container of the sub-resources deployed by the Network Operator by setting the parameter `containerResources`.

For example:

```
containerResources: - name: "mofed-container" requests: cpu: "200m" memory:
"150Mi" limits: cpu: "300m" memory: "300Mi"
```

MLNX_OFED Driver

Name	Type	Default	Description
ofedDriver.deploy	Bool	false	Deploy the MLNX_OFED driver container
ofedDriver.repository	String	nvcr.io/nvidia/mellanox	MLNX_OFED driver image repository
ofedDriver.image	String	doca-driver	MLNX_OFED driver image name
ofedDriver.version	String	24.04-0.6.6.0-0	MLNX_OFED driver version
ofedDriver.initContainer.enable	Bool	true	Deploy init container
ofedDriver.initContainer.repository	string	ghcr.io/mellanox	init container image repository
ofedDriver.initContainer.image	string	network-operator-init-container	init container image name

ofedDriver.initContainer.version	string	v0.0.2	init container image version
ofedDriver.certConfig.name	String	“”	Custom TLS key/certificate configuration configMap name
ofedDriver.repoConfig.name	String	“”	Private mirror repository configuration configMap name
ofedDriver.terminationGracePeriodSeconds	Int	300	NVIDIA OFED termination grace periods in seconds
ofedDriver.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the MLNX_OFED driver images
ofedDriver.env	List	[]	An optional list of environment variables passed to the NVIDIA OFED driver image
ofedDriver.startupProbe.initialDelaySeconds	Int	10	MLNX_OFED startup probe initial delay
ofedDriver.startupProbe.periodSeconds	Int	20	MLNX_OFED startup probe interval
ofedDriver.livenessProbe.initialDelaySeconds	Int	30	MLNX_OFED liveness probe initial delay
ofedDriver.livenessProbe.periodSeconds	Int	30	MLNX_OFED liveness probe interval

ofedDriver.readinessProbe.initialDelaySeconds	Int	10	MLNX_OFED readiness probe initial delay
ofedDriver.readinessProbe.periodSeconds	Int	30	MLNX_OFED readiness probe interval
ofedDriver.upgradePolicy.autoUpgrade	Bool	true	A global switch for the automatic upgrade feature. If set to false, all other options are ignored.
ofedDriver.upgradePolicy.maxParallelUpgrades	Int	1	The amount of nodes that can be upgraded in parallel. 0 means no limit. All nodes will be upgraded in parallel.
ofedDriver.upgradePolicy.safeLoad	Bool	false	Cordon and drain (if enabled) a node before loading the driver on it, requires `ofedDriver.initContainer` to be enabled and `ofedDriver.upgradePolicy.autoUpgrade` to be true
ofedDriver.upgradePolicy.drain.enable	Bool	true	Options for node drain (`kubectl drain`) before driver reload, if auto upgrade is enabled.
ofedDriver.upgradePolicy.drain.force	Bool	true	Use force drain of pods

ofedDriver.upgradePolicy.drain.podSelector	String	“”	Pod selector to specify which pods will be drained from the node. An empty selector means all pods.
ofedDriver.upgradePolicy.drain.timeoutSeconds	Int	300	Number of seconds to wait for pod eviction
ofedDriver.upgradePolicy.drain.deleteEmptyDir	Bool	false	Delete pods local storage
ofedDriver.upgradePolicy.waitForCompletion.podSelector	String	Not set	Specifies a label selector for the pods to wait for completion before starting the driver upgrade
ofedDriver.upgradePolicy.waitForCompletion.timeoutSeconds	int	Not set	Specify the length of time in seconds to wait before giving up for workload to finish. Zero means infinite
ofedDriver.containerResources	List	Not set	Optional resource requests and limits and limits for the `mofed-container`
ofedDriver.forcePrecompiled	Bool	false	Fail Mellanox OFED deployment if precompiled OFED driver container image does not exists

MLNX_OFED Driver Environment Variables

The following are special environment variables supported by the MLNX_OFED container to configure its behavior:

Name	Default	Description
CREATE_IFNAMES_UDEV	* “true” for Ubuntu 20.04, RHEL v8.x and OCP <= v4.13. * “false” for newer OS.	Create an udev rule to preserve “old-style” path based netdev names e.g enp3s0f0
UNLOAD_STORAGE_MODULES	“false”	Unload host storage modules prior to loading MLNX_OFED modules: * ib_isert * nvme_rdma * nvmet_rdma * rpcrdma * xprtrdma * ib_srpt
ENABLE_NFSRDMA	“false”	Enable loading of NFS related storage modules from a MLNX_OFED container
RESTORE_DRIVER_ON_POD_TERMINATION	“true”	Restore host drivers when a container

In addition, it is possible to specify any environment variables to be exposed to the MLNX_OFED container, such as the standard “HTTP_PROXY”, “HTTPS_PROXY”, “NO_PROXY”.

⚠ **Warning**

CREATE_IFNAMES_UDEV is set automatically by the Network Operator, depending on the Operating System of the worker nodes in the cluster (the cluster is assumed to be homogenous).

To set these variables, change them into Helm values. For example:

```
ofedDriver: env: - name: RESTORE_DRIVER_ON_POD_TERMINATION value: "true" -
name: UNLOAD_STORAGE_MODULES value: "true" - name: CREATE_IFNAMES_UDEV
value: "true"
```

The variables can also be configured directly via the NicClusterPolicy CRD.

RDMA Shared Device Plugin

Name	Type	Default	Description
rdmaSharedDevicePlugin.deploy	Bool	true	Deploy RDMA shared device plugin
rdmaSharedDevicePlugin.repository	String	nvcr.io/nvidia/cloud-native	RDMA shared device plugin image repository
rdmaSharedDevicePlugin.image	String	k8s-rdma-shared-dev-plugin	RDMA shared device plugin image name
rdmaSharedDevicePlugin.version	String	1.4.0	RDMA shared device plugin version
rdmaSharedDevicePlugin.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the RDMA Shared device plugin image
rdmaSharedDevicePlugin.resources	List	See below	RDMA shared device plugin resources

rdmaSharedDevicePlugin.useCdi	Bool	false	Enable Container Device Interface (CDI) mode. **NOTE**: NVIDIA Network Operator does not configure container runtime to enable CDI
rdmaSharedDevicePlugin.containerResources	List	Not set	Optional resource requests and limits for the `rdma-shared-dp` container

RDMA Device Plugin Resource Configurations

These configurations consist of a list of RDMA resources, each with a name and a selector of RDMA capable network devices to be associated with the resource. Refer to RDMA Shared Device Plugin Selectors for supported selectors.

```
resources: - name: rdma_shared_device_a vendors: [15b3] deviceIDs: [1017]
ifNames: [enp5s0f0] rdmaHcaMax: 63 - name: rdma_shared_device_b vendors:
[15b3] deviceIDs: [1017] ifNames: [ib0, ib1] rdmaHcaMax: 63
```

SR-IOV Network Device Plugin

Name	Type	Default	Description
sriovDevicePlugin.deploy	Bool	false	Deploy SR-IOV Network device plugin
sriovDevicePlugin.repository	String	ghcr.io/k8snetworkplumbingwg	SR-IOV Network device plugin image repository
sriovDevicePlugin.image	String	sriov-network-device-plugin	SR-IOV Network device plugin image

			name
sriovDevicePlugin.version	String	e6ead1e8f76a407783430ee2666b403db2d76f64	SR-IOV Network device plugin version For ARM-based deployments, it is recommended to use the `ghcr.io/k8snetworkplumbingwg/sriov-network-device-plugin:v3.6.2-amd64` image
sriovDevicePlugin.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the SR-IOV Network device plugin image
sriovDevicePlugin.resources	List	See below	SR-IOV Network device plugin resources
sriovDevicePlugin.useCdi	Bool	false	Enable Container Device Interface (CDI) mode. **NOTE**: NVIDIA Network Operator does not configure container runtime to enable CD.
sriovDevicePlugin.containerResources	List	Not set	Optional resource requests and limits for the `kube-sriovdp` container

SR-IOV Network Device Plugin Resource Configuration

Consists of a list of RDMA resources, each with a name and a selector of RDMA capable network devices to be associated with the resource. Refer to SR-IOV Network Device Plugin Selectors for supported selectors.

```
resources: - name: hostdev vendors: [15b3] - name: ethernet_rdma vendors: [15b3]
linkTypes: [ether] - name: sriov_rdma vendors: [15b3] devices: [1018] drivers:
[mlx5_ib]
```

IB Kubernetes

ib-kubernetes provides a daemon that works in conjunction with the SR-IOV Network Device Plugin. It acts on Kubernetes pod object changes (Create/Update/Delete), reading the pod's network annotation, fetching its corresponding network CRD and reading the PKey. This is done in order to add the newly generated GUID or the predefined GUID in the GUID field of the CRD cni-args to that PKey for pods with `mellanox.infiniband.app` annotation.

Name	Type	Default	Description
ibKubernetes.deploy	bool	false	Deploy IB Kubernetes
ibKubernetes.repository	string	ghcr.io/mellanox	IB Kubernetes image repository
ibKubernetes.image	string	ib-kubernetes	IB Kubernetes image name
ibKubernetes.version	string	v1.0.2	IB Kubernetes version
ibKubernetes.imagePullSecrets	list	[]	An optional list of references to secrets used for pulling any of the IB Kubernetes images
ibKubernetes.periodicUpdateSeconds	int	5	Interval of periodic update in seconds

ibKubernetes.pKeyGUIDPoolRangeStart	string	02:00:00:00:00:00:00:00	Minimal available GUID value to be allocated for the pod
ibKubernetes.pKeyGUIDPoolRangeEnd	string	02:FF:FF:FF:FF:FF:FF:FF	Maximal available GUID value to be allocated for the pod
ibKubernetes.ufmSecret	string	See below	Name of the Secret with the NVIDIA UFM access credentials, deployed in advance
ibKubernetes.containerResources	List	Not set	Optional resource requests and limits for the `ib-kubernetes` container

UFM Secret

IB Kubernetes must access NVIDIA UFM in order to manage pods' GUIDs. To provide its credentials, the secret of the following format should be deployed in advance:

```
apiVersion: v1 kind: Secret metadata: name: ib-kubernetes-ufm-secret namespace:
nvidia-network-operator stringData: UFM_USERNAME: "admin" UFM_PASSWORD:
"123456" UFM_ADDRESS: "ufm-hostname" UFM_HTTP_SCHEMA: "" UFM_PORT: ""
data: UFM_CERTIFICATE: ""
```

⚠ **Warning**

The InfiniBand Fabric manages a single pool of GUIDs. In order to use IB Kubernetes in different clusters, different GUID ranges must be specified to avoid collisions.

Secondary Network

Name	Type	Default	Description
secondaryNetwork.deploy	Bool	true	Deploy Secondary Network

Specifies components to deploy in order to facilitate a secondary network in Kubernetes. It consists of the following optionally deployed components:

- Multus-CNI: Delegate CNI plugin to support secondary networks in Kubernetes
- CNI plugins: Currently only containernetworking-plugins is supported
- IPAM CNI: Currently only Whereabout IPAM CNI is supported as a part of the secondaryNetwork section. NVIDIA-IPAM is configured separately.
- IPoIB CNI: Allows the user to create IPoIB child link and move it to the pod

CNI Plugin

Name	Type	Default	Description
secondaryNetwork.cniPlugins.deploy	Bool	true	Deploy CNI Plugins Secondary Network
secondaryNetwork.cniPlugins.image	String	plugins	CNI Plugins image name
secondaryNetwork.cniPlugins.repository	String	ghcr.io/k8snetworkplumbingwg	CNI Plugins image repository
secondaryNetwork.cniPlugins.version	String	v1.3.0-amd64	CNI Plugins image version
secondaryNetwork.cniPlugins.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the CNI Plugins images
secondaryNetwork.cniPlugins.containerResources	List	Not set	Optional resource requests and limits

			for the `cni-plugins` container

Multus CNI

Name	Type	Default	Description
secondaryNetwork.multus.deploy	Bool	true	Deploy Multus Secondary Network
secondaryNetwork.multus.image	String	multus-cni	Multus image name
secondaryNetwork.multus.repository	String	ghcr.io/k8snetworkplumbingwg	Multus image repository
secondaryNetwork.multus.version	String	v3.9.3	Multus image version
secondaryNetwork.multus.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the Multus images
secondaryNetwork.multus.config	String	“”	Multus CNI config. If empty, the config will be automatically generated from the CNI configuration file of the master plugin (the first file in lexicographical order in the cni-confg-dir).
secondaryNetwork.multus.containerResources	List	Not set	Optional resource requests and limits for the `kube-multus` container

IPoIB CNI

Name	Type	Default	Description
secondaryNetwork.ipoib.deploy	Bool	false	Deploy IPoIB CNI
secondaryNetwork.ipoib.image	String	ipoib-cni	IPoIB CNI image name
secondaryNetwork.ipoib.repository	String	“”	IPoIB CNI image repository
secondaryNetwork.ipoib.version	String	428715a57c0b633e48ec7620f6e3af6863149ccf	IPoIB CNI image version
secondaryNetwork.ipoib.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the IPoIB CNI images
secondaryNetwork.ipoib.containerResources	List	Not set	Optional resource requests and limits for the `ipoib-cni` container

IPAM CNI Plugin

Name	Type	Default	Description
secondaryNetwork.ipamPlugin.deploy	Bool	true	Deploy IPAM CNI Plugin Secondary Network
secondaryNetwork.ipamPlugin.image	String	whereabouts	IPAM CNI Plugin image name
secondaryNetwork.ipamPlugin.repository	String	ghcr.io/k8snetworkplumbingwg	IPAM CNI Plugin image repository
secondaryNetwork.ipamPlugin.version	String	v0.7.0-amd64	IPAM CNI Plugin image version
secondaryNetwork.ipamPlugin.imagePul	List	[]	An optional list of references to

lSecrets			secrets to use for pulling any of the IPAM CNI Plugin images
secondaryNetwork.ipamPlugin.containerResources	List	Not set	Optional resource requests and limits for the `whereabouts` container

NVIDIA IPAM Plugin

NVIDIA IPAM Plugin is recommended to be used on large-scale deployments of the NVIDIA Network Operator.

Name	Type	Default	Description
nvIpam.deploy	Bool	false	Deploy NVIDIA IPAM Plugin
nvIpam.image	String	nvidia-k8s-ipam	NVIDIA IPAM Plugin image name
nvIpam.repository	String	ghcr.io/mellanox	NVIDIA IPAM Plugin image repository
nvIpam.version	String	v0.1.2	NVIDIA IPAM Plugin image version
nvIpam.imagePullSecrets	List	[]	An optional list of references to secrets to use for pulling any of the Plugin images
nvIpam.enableWebhook	Bool	false	Enable deployment of the validataion webhook for IPPool CRD
nvIpam.containerResources	List	Not set	Optional resource requests and limits for the

			nv-ipam-node and nv-ipam-controller containers

⚠ **Warning**

Supported X.509 certificate management system should be available in the cluster to enable the validation webhook. Currently, the supported systems are certmanager and Openshift certificate management.

NVIDIA NIC Feature Discovery

NVIDIA NIC Feature Discovery leverages Node Feature Discovery to advertise NIC specific labels on K8s Node objects.

Name	Type	Default	Description
nicFeatureDiscovery.deploy	Bool	false	Deploy NVIDIA NIC Feature Discovery
nicFeatureDiscovery.image	String	nic-feature-discovery	NVIDIA NIC Feature Discovery image name
nicFeatureDiscovery.repository	String	ghcr.io/mellanox	NVIDIA NIC Feature Discovery repository
nicFeatureDiscovery.version	String	v0.0.1	NVIDIA NIC Feature Discovery image version
nicFeatureDiscovery.containerResources	List	Not set	Optional resource requests and limits for the nic-feature-discovery container

DOCA Telemetry Service

DOCA Telemetry Service exports metrics from NVIDIA NICs on K8s Nodes.

Name	Type	Default	Description
docaTelemetryService.deploy	Bool	false	Deploy DOCA Telemetry Service
docaTelemetryService.image	String	doca_telemetry	DOCA Telemetry Service image name
docaTelemetryService.repository	String	nvcr.io/nvidia/doca	DOCA Telemetry Service image repository
docaTelemetryService.version	String	1.16.5-doca2.6.0-host	DOCA Telemetry Service image version
docaTelemetryService.containerResources	List	Not set	Optional resource requests and limits for the `doca-telemetry-service` container

Helm customization file

⚠ **Warning**

Since several parameters should be provided when creating custom resources during operator deployment, it is recommended to use a configuration file. While it is possible to override the parameters via CLI, we recommend to avoid the use of CLI arguments in favor of a configuration file.

```
$ helm install -f ./values.yaml -n nvidia-network-operator --create-namespace --wait
nvidia/network-operator network-operator
```

CRDs

- NicClusterPolicy CRD
- MacVlanNetwork CRD
- HostDeviceNetwork CRD
- IPoIBNetwork CRD

NicClusterPolicy CRD

To change `NicClusterPolicy` CRD object manually without helm you need to change `nic-cluster-policy` CR like a regular Kubernetes resource. For more information on NicClusterPolicy custom resource, please refer to the Network-Operator Project Sources.

MacVlanNetwork CRD

For more information on *MacVlanNetwork* custom resource, please refer to the Network-Operator Project Sources.

HostDeviceNetwork CRD

For more information on *HostDeviceNetwork* custom resource, please refer to the Network-Operator Project Sources.

IPoIBNetwork CRD

For more information on *IPoIBNetwork* custom resource, please refer to the Network-Operator Project Sources.

Life Cycle Management

On this page

Ensuring Deployment Readiness

Once the Network Operator is deployed, and a NicClusterPolicy resource is created, the operator will reconcile the state of the cluster until it reaches the desired state, as defined in the resource.

Alignment of the cluster to the defined policy can be verified in the custom resource status.

a “Ready” state indicates that the required components were deployed, and that the policy is applied on the cluster.

Status Field Example of a NICClusterPolicy Instance

Get the NicClusterPolicy status:

```
kubectl get -n nvidia-network-operator nicclusterpolicies.mellanox.com nic-cluster-policy -o yaml
```

```
status: appliedStates: - name: state-pod-security-policy state: ignore - name: state-multus-cni state: ready - name: state-container-networking-plugins state: ignore - name: state-ipoib-cni state: ignore - name: state-whereabouts-cni state: ready - name: state-OFED state: ready - name: state-SRIOV-device-plugin state: ignore - name: state-RDMA-device-plugin state: ready - name: state-NV-Peer state: ignore - name: state-ib-kubernetes state: ignore - name: state-nv-ipam-cni state: ready state: ready
```

Note

An “Ignore” state indicates that the sub-state was not defined in the custom resource, and thus, it is ignored.

Network Operator Upgrade

Before upgrading to Network Operator v1.0 or newer with SR-IOV Network Operator enabled, the following manual actions are required:

```
$ kubectl -n nvidia-network-operator scale deployment network-operator-sriov-
network-operator --replicas 0 $ kubectl -n nvidia-network-operator delete
sriovnetworknodepolicies.sriovnetwork.openshift.io default
```

The network operator provides limited upgrade capabilities, which require additional manual actions if a containerized OFED driver is used. Future releases of the network operator will provide an automatic upgrade flow for the containerized driver.

Since Helm does not support auto-upgrade of existing CRDs, the user must follow a two-step process to upgrade the network-operator release:

- Upgrade the CRD to the latest version
- Apply Helm chart update

Downloading a New Helm Chart

To obtain new releases, run:

```
# Download Helm chart $ helm fetch https://helm.ngc.nvidia.com/nvidia/charts/network-
operator-24.4.0.tgz $ ls network-operator-*.tgz | xargs -n 1 tar xf
```

Upgrading CRDs for a Specific Release

It is possible to retrieve updated CRDs from the Helm chart or from the release branch on GitHub. The example below shows how to upgrade CRDs from the downloaded chart.

```
$ kubectl apply \ -f network-operator/crds \ -f network-operator/charts/sriov-network-operator/crds
```

Preparing the Helm Values for the New Release

Edit the values-<VERSION>.yaml file as required for your cluster. The network operator has some limitations as to which updates in the NicClusterPolicy it can handle automatically. If the configuration for the new release is different from the current configuration in the deployed release, some additional manual actions may be required.

Known limitations:

- If component configuration was removed from the NicClusterPolicy, manual clean up of the component's resources (DaemonSets, ConfigMaps, etc.) may be required.
- If the configuration for devicePlugin changed without image upgrade, manual restart of the devicePlugin may be required.

These limitations will be addressed in future releases.

> ⚠ **Warning**
>
> Changes that were made directly in the NicClusterPolicy CR (e.g. with kubectl edit) will be overwritten by the Helm upgrade due to the *force* flag.

Applying the Helm Chart Update

To apply the Helm chart update, run:

```
$ helm upgrade -n nvidia-network-operator network-operator nvidia/network-operator --version=<VERSION> -f values-<VERSION>.yaml --force
```

> ⚠ **Warning**
>
> The –devel option is required if you wish to use the Beta release.

OFED Driver Manual Upgrade

Restarting Pods with a Containerized OFED Driver

> ⚠ **Warning**
>
> This operation is required only if containerized OFED is in use.

When a containerized OFED driver is reloaded on the node, all pods that use a secondary network based on NVIDIA NICs will lose network interface in their containers. To prevent outage, remove all pods that use a secondary network from the node before you reload the driver pod on it.

The Helm upgrade command will only upgrade the DaemonSet spec of the OFED driver to point to the new driver version. The OFED driver's DaemonSet will not automatically restart pods with the driver on the nodes, as it uses "OnDelete" updateStrategy. The old OFED version will still run on the node until you explicitly remove the driver pod or reboot the node:

```
$ kubectl delete pod -l app=mofed-<OS_NAME> -n nvidia-network-operator
```

It is possible to remove all pods with secondary networks from all cluster nodes, and then restart the OFED pods on all nodes at once.

The alternative option is to perform an upgrade in a rolling manner to reduce the impact of the driver upgrade on the cluster. The driver pod restart can be done on each node individually. In this case, pods with secondary networks should be removed from the single node only. There is no need to stop pods on all nodes.

For each node, follow these steps to reload the driver on the node:

1. Remove pods with a secondary network from the node.
2. Restart the OFED driver pod.
3. Return the pods with a secondary network to the node.

When the OFED driver is ready, proceed with the same steps for other nodes.

Removing Pods with a Secondary Network from the Node

To remove pods with a secondary network from the node with node drain, run the following command:

```
$ kubectl drain <NODE_NAME> --pod-selector=<SELECTOR_FOR_PODS>
```

> ⚠ **Warning**
>
> Replace <NODE_NAME> with -l “network.nvidia.com/operator.mofed.wait=false” if you wish to drain all nodes at once.

Restarting the OFED Driver Pod

Find the OFED driver pod name for the node:

```
$ kubectl get pod -l app=mofed-<OS_NAME> -o wide -A
```

Example for Ubuntu 20.04:

```
kubectl get pod -l app=mofed-ubuntu20.04 -o wide -A
```

Deleting the OFED Driver Pod from the Node

To delete the OFED driver pod from the node, run:

```
$ kubectl delete pod -n <DRIVER_NAMESPACE> <OFED_POD_NAME>
```

> ⚠ **Warning**
>
> Replace <OFED_POD_NAME> with -l app=mofed-ubuntu20.04 if you wish to remove OFED pods on all nodes at once.

A new version of the OFED pod will automatically start.

Returning Pods with a Secondary Network to the Node

After the OFED pod is ready on the node, you can make the node schedulable again.

The command below will uncordon (remove node.kubernetes.io/unschedulable:NoSchedule taint) the node, and return the pods to it:

```
$ kubectl uncordon -l "network.nvidia.com/operator.mofed.wait=false"
```

Automatic OFED Driver Upgrade

To enable automatic OFED upgrade, define the UpgradePolicy section for the ofedDriver in the NicClusterPolicy spec, and change the OFED version.

`nicclusterpolicy.yaml`:

apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata: name: nic-cluster-policy namespace: nvidia-network-operator spec: ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox version: 24.04-0.6.6.0-0 upgradePolicy: # autoUpgrade is a global switch for automatic upgrade feature # if set to false all other options are ignored autoUpgrade: true # maxParallelUpgrades indicates how many nodes can be upgraded in parallel # 0 means no limit, all nodes will be upgraded in parallel maxParallelUpgrades: 0 # cordon and drain (if enabled) a node before loading the driver on it safeLoad: false # describes the configuration for waiting on job completions waitForCompletion: # specifies a label selector for the pods to wait for completion podSelector: "app=myapp" # specify the length of time in seconds to wait before giving up for workload to finish, zero means infinite # if not specified, the default is 300 seconds timeoutSeconds: 300 # describes configuration for node drain during automatic upgrade drain: # allow node draining during upgrade enable: true # allow force draining force: false # specify a label selector to filter pods on the node that need to be drained podSelector: "" # specify the length of time in seconds to wait before giving up drain, zero means infinite # if not specified, the default is 300 seconds timeoutSeconds: 300 # specify if should continue even if there are pods using emptyDir deleteEmptyDir: false

Apply NicClusterPolicy CRD:

```
$ kubectl apply -f nicclusterpolicy.yaml
```

⚠ **Warning**

To be able to drain nodes, make sure to fill the PodDisruptionBudget field for all the pods that use it. On some clusters (e.g. Openshift), many pods use PodDisruptionBudget, which makes draining multiple nodes at once impossible. Since evicting several pods that are controlled by the same deployment or replica set, violates their PodDisruptionBudget, those pods are not evicted and in drain failure.

To perform a driver upgrade, the network-operator must evict pods that are using network resources. Therefore, in order to ensure that the network-operator is evicting only the required pods, the upgradePolicy.drain.podSelector field must be configured.

Node Upgrade States

The status upgrade of each node is reflected in its nvidia.com/ofed-driver-upgrade-state label . This label can have the following values:

Name	Description

Unknown (empty)	The node has this state when the upgrade flow is disabled or the node has not been processed yet.
`upgrade-done`	Set when OFED POD is up-to-date and running on the node, the node is schedulable.
`upgrade-required`	Set when OFED POD on the node is not up-to-date and requires upgrade. No actions are performed at this stage.
`cordon-required`	Set when the node needs to be made unschedulable in preparation for driver upgrade.
`wait-for-jobs-required`	Set on the node when waiting is required for jobs to complete until the given timeout.
`drain-required`	Set when the node is scheduled for drain. After the drain, the state is changed either to pod-restart-required or upgrade-failed.
`pod-restart-required`	Set when the OFED POD on the node is scheduled for restart. After the restart, the state is changed to uncordon-required.
`uncordon-required`	Set when OFED POD on the node is up-to-date and has “Ready” status. After uncordone, the state is changed to upgrade-done
`upgrade-failed`	Set when the upgrade on the node has failed. Manual interaction is required at this stage. See Troubleshooting section for more details.

⚠ **Warning**

Depending on your cluster workloads and pod Disruption Budget, set the following values for auto upgrade:

```
apiVersion: mellanox.com/v1alpha1 kind: NicClusterPolicy metadata:
name: nic-cluster-policy namespace: nvidia-network-operator spec:
ofedDriver: image: doca-driver repository: nvcr.io/nvidia/mellanox
version: 24.04-0.6.6.0-0 upgradePolicy: autoUpgrade: true
maxParallelUpgrades: 1 drain: enable: true force: false
deleteEmptyDir: true podSelector: ""
```

Safe Driver Loading

> ⚠ **Warning**
>
> The state of this feature can be controlled with the ofedDriver.upgradePolicy.safeLoad option.

Upon node startup, the OFED container takes some time to compile and load the driver. During that time, workloads might get scheduled on that node. When OFED is loaded, all existing PODs that use NVIDIA NICs will lose their network interfaces. Some such PODs might silently fail or hang. To avoid this situation, before the OFED container is loaded, the node should get cordoned and drained to ensure all workloads are rescheduled. The node should be un-cordoned when the driver is ready on it.

The safe driver loading feature is implemented as a part of the upgrade flow, meaning safe driver loading is a special scenario of the upgrade procedure, where we upgrade from the inbox driver to the containerized OFED.

When this feature is enabled, the initial OFED driver rollout on the large cluster can take a while. To speed up the rollout, the initial deployment can be done with the safe driver loading feature disabled, and this feature can be enabled later by updating the NicClusterPolicy CRD.

Troubleshooting

Issue	Required Action
The node is in upgrade-failed state.	• Drain the node manually by running kubectl drain –ignore-daemonsets.

	- Delete the MLNX_OFED pod on the node manually, by running the following command: `kubectl delete pod -n ``kubectl get pods --A --field-selector spec.nodeName=&lt;node name&gt; -l nvidia.com/ofed-driver --no-headers | awk '{print $1 " "$2}'`` ` . **NOTE:** If the "Safe driver loading" feature is enabled, you may also need to remove the `nvidia.com/ofed-driver-upgrade.driver-wait-for-safe-load` annotation from the node object to unblock the loading of the driver `kubectl annotate node &lt;node_name&gt; nvidia.com/ofed-driver-upgrade.driver-wait-for-safe-load-` - Wait for the node to complete the upgrade.
The updated MLNX_OFED pod failed to start/ a new version of MLNX_OFED cannot be installed on the node.	Manually delete the pod by using `kubectl delete -n &lt;Network Operator Namespace&gt; &lt;pod name&gt;` . If following the restart the pod still fails, change the MLNX_OFED version in the NicClusterPolicy to the previous version or to another working version.

Uninstalling the Network Operator

Uninstalling Network Operator on a Vanilla Kubernetes Cluster

Uninstall the Network Operator:

```
helm uninstall network-operator -n nvidia-network-operator
```

You should now see all the pods being deleted:

```
kubectl get pods -n nvidia-network-operator
```

Make sure that the CRDs created during the operator installation have been removed:

```
kubectl get nicclusterpolicies.mellanox.com No resources found
```

Uninstalling the Network Operator on an OpenShift Cluster

From the console:

In the OpenShift Container Platform web console side menu, select **Operators >Installed Operators**, search for the **NVIDIA Network Operator**, and click on it.

On the right side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.

For additional information, see the Red Hat OpenShift Container Platform Documentation.

From the CLI:

- Check the current version of the Network Operator in the currentCSV field:

  ```
  oc get subscription -n nvidia-network-operator nvidia-network-operator -o
  yaml | grep currentCSV
  ```

  Example output:

  ```
  currentCSV: nvidia-network-operator.v24.1.0
  ```

- Delete the subscription:

  ```
  oc delete subscription -n nvidia-network-operator nvidia-network-operator
  ```

Example output:

```
subscription.operators.coreos.com "nvidia-network-operator" deleted
```

- Delete the CSV using the currentCSV value from the previous step:

```
subscription.operators.coreos.com "nvidia-network-operator" deleted
```

Example output:

```
clusterserviceversion.operators.coreos.com "nvidia-network-operator.v10.0"
deleted
```

The SR-IOV Network Operator uninstallation procedure is described in this document. For additional information, see the Red Hat OpenShift Container Platform Documentation.

Additional Steps

> ⚠ **Warning**
>
> In OCP, uninstalling an operator does not remove its managed resources, including CRDs and CRs. To remove them, you must manually delete the Operator CRDs following the operator uninstallation.

Delete the Network Operator CRDs:

```
oc delete crds hostdevicenetworks.mellanox.com macvlannetworks.mellanox.com
nicclusterpolicies.mellanox.com
```

NicClusterPolicy CRD Update

If the NicClusterPolicy manual update affects the device plugin configuration (e.g. NICs selectors), manual device plugin pods restart is required.

Advanced Configurations

On this page

Network Operator Deployment with Admission Controller

The Admission Controller can be optionally included as part of the Network Operator installation process. It has the capability to validate supported Custom Resource Definitions (CRDs), which currently include NicClusterPolicy and HostDeviceNetwork. By default, the deployment of the admission controller is disabled. To enable it, you must set `operator.admissionController.enabled` to `true`.

Enabling the admission controller provides you with two options for managing certificates. You can either utilize the cert-manager for generating a self-signed certificate automatically, or, alternatively, provide your own self-signed certificate.

To use cert-manager, ensure that `operator.admissionController.useCertManager` is set to `true`. Additionally, make sure that you deploy the cert-manager before initiating the Network Operator deployment.

If you prefer not to use the cert-manager, set `operator.admissionController.useCertManager` to `false`, and then provide your custom certificate and key using `operator.admissionController.certificate.tlsCrt` and `operator.admissionController.certificate.tlsKey`.

⚠ **Warning**

When using your own certificate, the certificate must be valid for

```
&lt;Release_Name&gt;-webhook-
service.&lt;Release_Namespace&gt;.svc
```

, e.g. `network-operator-webhook-service.nvidia-network-operator.svc`.

Network Operator Deployment with Pod Security Admission

The Pod Security admission controller replaces PodSecurityPolicy, enforcing predefined Pod Security Standards by adding a label to a namespace.

There are three levels defined by the Pod Security Standards : `privileged` , `baseline` and `restricted`.

⚠ **Warning**

In case you wish to enforce a PSA to the Network Operator namespace, the `privileged` level is required. Enforcing `baseline` or `restricted` levels will prevent the creation of required Network Operator pods.

If required, enforce PSA privileged level on the Network Operator namespace by running:

```
kubectl label --overwrite ns nvidia-network-operator pod-
security.kubernetes.io/enforce=privileged
```

In case that baseline or restricted levels are being enforced on the Network Operator namespace, events for pods creation failures will be triggered:

```
kubectl get events -n nvidia-network-operator --field-selector reason=FailedCreate
LAST SEEN TYPE REASON OBJECT MESSAGE 2m36s Warning FailedCreate
daemonset/mofed-ubuntu22.04-ds Error creating: pods "mofed-ubuntu22.04-ds-
rwmgs" is forbidden: violates PodSecurity "baseline:latest": host namespaces
(hostNetwork=true), hostPath volumes (volumes "run-mlnx-ofed", "etc-network",
"host-etc", "host-usr", "host-udev"), privileged (container "mofed-container" must
not set securityContext.privileged=true)
```

Network Operator Deployment in a Proxy Environment

This section describes how to successfully deploy the Network Operator in clusters behind an HTTP Proxy. By default, the Network Operator requires internet access for the following reasons:

- Container images must be pulled during the NVIDIA Network Operator installation.
- The driver container must download several OS packages prior to the driver installation.

To address these requirements, all Kubernetes nodes, as well as the driver container, must be properly configured in order to direct traffic through the proxy.

This section demonstrates how to configure the NVIDIA Network Operator, so that the driver container could successfully download packages behind an HTTP proxy. Since configuring Kubernetes/container runtime components for proxy use is not specific to the Network Operator, those instructions are not detailed here.

> ⚠ **Warning**
>
> If you are not running OpenShift, please skip the section titled HTTP Proxy Configuration for OpenShift, as Openshift configuration instructions are different.

Prerequisites

Kubernetes cluster is configured with HTTP proxy settings (container runtime should be enabled with HTTP proxy).

HTTP Proxy Configuration for Openshift

For Openshift, it is recommended to use the cluster-wide Proxy object to provide proxy information for the cluster. Please follow the procedure described in Configuring the Cluster-wide Proxy via the Red Hat Openshift public documentation. The NVIDIA Network Operator will automatically inject proxy related ENV into the driver container, based on the information present in the cluster-wide Proxy object.

HTTP Proxy Configuration

Specify the `ofedDriver.env` in your `values.yaml` file with appropriate `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables (in both uppercase and lowercase).

```
ofedDriver: env: - name: HTTPS_PROXY value: http://<example.proxy.com:port> -
name: HTTP_PROXY value: http://<example.proxy.com:port> - name: NO_PROXY
value: <example.com> - name: https_proxy value: http://<example.proxy.com:port>
- name: http_proxy value: http://<example.proxy.com:port> - name: no_proxy value:
<example.com>
```

Network Operator Deployment in an Air-gapped Environment

This section describes how to successfully deploy the Network Operator in clusters with restricted internet access. By default, the Network Operator requires internet access for the following reasons:

- The container images must be pulled during the Network Operator installation.
- The OFED driver container must download several OS packages prior to the driver installation.

To address these requirements, it may be necessary to create a local image registry and/or a local package repository, so that the necessary images and packages will be available for your cluster. Subsequent sections of this document detail how to configure the Network Operator to use local image registries and local package repositories. If your cluster is behind a proxy, follow the steps listed in Network Operator Deployment in Proxy Environments.

Local Image Registry

Without internet access, the Network Operator requires all images to be hosted in a local image registry that is accessible to all nodes in the cluster. To allow Network Operator to work with a local registry, users can specify local repository, image, tag along with pull secrets in the `values.yaml` file.

Pulling and Pushing Container Images to a Local Registry

To pull the correct images from the NVIDIA registry, you can leverage the fields `repository`, `image` and `version` specified in the `values.yaml` file.

Local Package Repository

> ⚠ **Warning**
>
> The instructions below are provided as reference examples to set up a local package repository for NVIDIA Network Operator.

The OFED driver container deployed as part of the Network Operator requires certain packages to be available for the driver installation. In restricted internet access or air-gapped installations, users are required to create a local mirror repository for their OS distribution, and make the following packages available:

```
ubuntu: linux-headers-${KERNEL_VERSION} linux-modules-${KERNEL_VERSION} pkg-
config rhel, rhcos: kernel-headers-${KERNEL_VERSION} kernel-
devel-${KERNEL_VERSION} kernel-core-${KERNEL_VERSION} createrepo elfutils-
libelf-devel kernel-rpm-macros umactl-libs lsof pm-build patch hostname
```

For RT kernels following packages should be available:

```
kernel-rt-devel-${KERNEL_VERSION} kernel-rt-modules-${KERNEL_VERSION}
```

For Ubuntu, these packages can be found at archive.ubuntu.com, and be used as the mirror that must be replicated locally for your cluster. By using apt-mirror or apt-get download, you can create a full or a partial mirror to your repository server.

For RHCOS, dnf reposync can be used to create the local mirror. This requires an active Red Hat subscription for the supported OpenShift version. For example:

```
dnf --releasever=8.4 reposync --repo rhel-8-for-x86_64-appstream-rpms --download-
metadata
```

Once all the above required packages are mirrored to the local repository, repo lists must be created following distribution specific documentation. A ConfigMap containing the repo list file should be created in the namespace where the NVIDIA Network Operator is deployed.

Following is an example of a repo list for Ubuntu 20.04 (access to a local package repository via HTTP):

`custom-repo.list`:

```
deb [arch=amd64 trusted=yes] http://<local pkg
repository>/ubuntu/mirror/archive.ubuntu.com/ubuntu focal main universe deb
[arch=amd64 trusted=yes] http://<local pkg
repository>/ubuntu/mirror/archive.ubuntu.com/ubuntu focal-updates main universe
deb [arch=amd64 trusted=yes] http://<local pkg
repository>/ubuntu/mirror/archive.ubuntu.com/ubuntu focal-security main universe
```

Following is an example of a repo list for RHCOS (access to a local package repository via HTTP):

`cuda.repo` (a mirror of https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64):

```
[cuda] name=cuda baseurl=http://<local pkg repository>/cuda priority=0 gpgcheck=0
enabled=1
```

`redhat.repo`:

```
[baseos] name=rhel-8-for-x86_64-baseos-rpms baseurl=http://<local pkg
repository>/rhel-8-for-x86_64-baseos-rpms gpgcheck=0 enabled=1 [baseoseus]
name=rhel-8-for-x86_64-baseos-eus-rpms baseurl=http://<local pkg repository>/rhel-
8-for-x86_64-baseos-eus-rpms gpgcheck=0 enabled=1 [rhocp] name=rhocp-4.10-for-
```

```
rhel-8-x86_64-rpms baseurl=http://<local pkg repository>/rhocp-4.10-for-rhel-8-x86_64-rpms gpgcheck=0 enabled=1 [apstream] name=rhel-8-for-x86_64-appstream-rpms baseurl=http://<local pkg repository>/rhel-8-for-x86_64-appstream-rpms gpgcheck=0 enabled=1
```

`ubi.repo`:

```
[ubi-8-baseos] name = Red Hat Universal Base Image 8 (RPMs) - BaseOS baseurl = http://<local pkg repository>/ubi-8-baseos enabled = 1 gpgcheck = 0 [ubi-8-baseos-source] name = Red Hat Universal Base Image 8 (Source RPMs) - BaseOS baseurl = http://<local pkg repository>/ubi-8-baseos-source enabled = 0 gpgcheck = 0 [ubi-8-appstream] name = Red Hat Universal Base Image 8 (RPMs) - AppStream baseurl = http://<local pkg repository>/ubi-8-appstream enabled = 1 gpgcheck = 0 [ubi-8-appstream-source] name = Red Hat Universal Base Image 8 (Source RPMs) - AppStream baseurl = http://<local pkg repository>/ubi-8-appstream-source enabled = 0 gpgcheck = 0
```

Create the ConfigMap for Ubuntu:

```
kubectl create configmap repo-config -n <Network Operator Namespace> --from-file=<path-to-repo-list-file>
```

Create the ConfigMap for RHCOS:

```
kubectl create configmap repo-config -n <Network Operator Namespace> --from-file=cuda.repo --from-file=redhat.repo --from-file=ubi.repo
```

Once the ConfigMap is created using the above command, update the `values.yaml` file with this information to let the Network Operator mount the repo configuration within the driver container and pull the required packages. Based on the OS distribution, the Network Operator will automatically mount this ConfigMap into the appropriate directory.

```
ofedDriver: deploy: true repoConfg: name: repo-config
```

If self-signed certificates are used for an HTTPS based internal repository, a ConfigMap must be created for those certifications and provided during the Network Operator installation. Based on the OS distribution, the Network Operator will automatically mount this ConfigMap into the appropriate directory.

```
kubectl create configmap cert-config -n <Network Operator Namespace> --from-file=<path-to-pem-file1> --from-file=<path-to-pem-file2>
```

```
ofedDriver: deploy: true certConfg: name: cert-config
```

Precompiled Container Build Instructions for DOCA Drivers

Prerequisites

Before you begin, ensure that you have the following prerequisites:

Common

- Docker (Ubuntu) / Podman (RH) installed on your build system.
- Web access to NVIDIA NIC drivers sources. Latest NIC drivers published at NIC drivers download center, for example: https://www.mellanox.com/downloads/ofed/MLNX_OFED-24.04-0.6.6.0/MLNX_OFED_SRC-debian-24.04-0.6.6.0-0.tgz

RHEL

- Active subscription and login credentials for registry.redhat.io. To build RHEL based container from official repository, you need to log in to registry.redhat.io, run the following command:

```
podman login registry.redhat.io --username=${RH_USERNAME} --password=${RH_PASSWORD}
```

Replace *RH_USERNAME* and *RH_PASSWORD* with your Red Hat account username and password.

Dockerfile Overview

To build the precompiled container, the Dockerfile is constructed in a multistage fashion. This approach is used to optimize the resulting container image size and reduce the number of dependencies included in the final image.

The Dockerfile consists of the following stages:

1. **Base Image Update**: The base image is updated and common requirements are installed. This stage sets up the basic environment for the subsequent stages.

2. **Download Driver Sources**: This stage downloads the Mellanox OFED driver sources to the specified path. It prepares the necessary files for the driver build process.

3. **Build Driver**: The driver is built using the downloaded sources and installed on the container. This stage ensures that the driver is compiled and configured correctly for the target system.

4. **Install precompiled driver**: Finally, the precompiled driver is installed on clean container. This stage sets up the environment to run the NVIDIA NIC drivers on the target system.

Common mandatory build parameters

Before building the container, you need to provide following parameters as *build-arg* for container build:

1. *D_OS*: The Linux distribution (e.g., ubuntu22.04 / rhel9.2)

2. *D_ARCH*: Compiled Architecture

3. *D_BASE_IMAGE*: Base container image

4. *D_KERNEL_VER*: The target kernel version (e.g., 5.15.0-25-generic / 5.14.0-284.32.1.el9_2.x86_64)

5. *D_OFED_VERSION*: NVIDIA NIC drivers version (e.g., 24.01-0.3.3.1)

NOTE: Check desired NVIDIA NIC drivers sources[^1] availability for designated container OS, only versions available on download page can be utilized

RHEL-specific build parameters

1. *D_BASE_IMAGE*: DriverToolKit container image

NOTE: DTK (DriverToolKit) is tightly coupled with specific kernel versions, verify match between kernel version to compile drivers for, versus DTK image.

2. *D_FINAL_BASE_IMAGE*: Final container image, to install compiled driver

For more details regarding DTK please read official documentation.

NOTE: For proper Network Operator functionality container tag name must be in following pattern: **driver_ver-container_ver-kernel_ver-os-arch**. For example: 24.01-0.3.3.1-0-5.15.0-25-generic-ubuntu22.04-amd64

RHEL example

To build RHEL-based image please use provided Dockerfile:

```
podman build \ --build-arg D_OS=rhel9.2 \ --build-arg D_ARCH=x86_64 \ --build-arg D_KERNEL_VER=5.14.0-284.32.1.el9_2.x86_64 \ --build-arg D_OFED_VERSION=24.01-0.3.3.1 \ --build-arg D_BASE_IMAGE="registry.redhat.io/openshift4/driver-toolkit-rhel9:v4.13.0-202309112001.p0.gd719bdc.assembly.stream" \ --build-arg D_FINAL_BASE_IMAGE=registry.access.redhat.com/ubi9/ubi:latest \ --tag 24.04-0.6.6.0-0-5.14.0-284.32.1.el9_2-rhel9.2-amd64 \ -f RHEL_Dockerfile \ --target precompiled .
```

Ubuntu example

To build RHEL-based image please use provided Dockerfile:.

```
docker build \ --build-arg D_OS=ubuntu22.04 \ --build-arg D_ARCH=x86_64 \ --build-arg D_BASE_IMAGE=ubuntu:24.04 \ --build-arg D_KERNEL_VER=5.15.0-25-generic \ --build-arg D_OFED_VERSION=24.01-0.3.3.1 \ --tag 24.01-0.3.3.1-0-5.15.0-25-generic-ubuntu22.04-amd64 \ -f Ubuntu_Dockerfile \ --target precompiled .
```

NOTE: Dockerfiles contain default build parameters, which may fail build proccess on your system if not overridden.

NOTE: Entrypoint script download **NOTE:** Driver build script download

⚠ **Warning**

Modification of *D_OFED_SRC_DOWNLOAD_PATH* must be tighdly coupled with corresponding update to entrypoint.sh script.