

NVIDIA Network Operator v25.4.0

Table of contents

Release Notes 3

Platform Support 15

Getting Started with Kubernetes 22

Getting Started with Red Hat OpenShift 125

Customization Options 142

- Helm Chart Customization Options 143
- Network Operator API reference v1alpha1 156
- NicClusterPolicy Custom Resource Example 173

Life Cycle Management 178

Advanced Configurations 197

- Proxy & Air-Gapped Environments 198
- NVIDIA DOCA Driver Container 208
- Advanced Configurations 197
- NVIDIA Network Operator Container Images 219

Legal Notices and 3rd Party Licenses 229

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

Changes and New Features

Version	Description
25.4.0	- Added support for NVIDIA NIC Configuration Operator deployment through NicClusterPolicy CR, since using Helm chart will be deprecated in future releases. - Integrate NVIDIA Network Operator with NVIDIA Maintenance Operator for DOCA OFED container upgrade. - Added support for OpenShift 4.18. - Added support for ConnectX-8 SuperNIC. - Added support for NVIDIA Spectrum-X Operator deployment - Tech Preview.
25.1.0	- Added support for OpenShift Container Platform v4.17. - Added support for SUSE Linux Enterprise Server 15 SP6 with Upstream K8s and Rancher. - Added support for RHEL v9.5. - Removed support for Ubuntu 20.04. - Added support for NVIDIA Maintenance Operator deployment.
24.10.1	- CVE fix (CVE-2024-45338).

24.10.0	- Added support for NVIDIA NIC Configuration Operator deployment. - Added support for Support Single-Node OpenShift (SNO). - NVIDIA Network Operator Helm chart does not create NicClusterPolicy CR anymore.
24.7.0	- Added support for OpenShift Container Platform v4.16. - Added support for NVIDIA Grace based ARM platforms with OpenShift Container Platform. - Added support for RHEL v8.9, v8.10, v9.3 and v9.4 with Upstream K8s and Containerd runtime. - Added support for Switchdev mode SR-IOV mode with OVS CNI with RHEL.
24.4.1	- Fixed NVIDIA Network Operator images in OpenShift Container Platform bundle.

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
25.4.0	- Whereabouts support is deprecated in Network Operator 25.4. It is advised to migrate to NV-IPAM as 'ipam' plugin.
24.10.0	- Dropped Multus CNI support for versions older than v4.1.0.
24.7.0	- Deploying NicClusterPolicy Custom Resource through helm is deprecated, support will be removed in Network Operator 24.10. It is advised to keep deployCR=false in your helm values and create/update NicClusterPolicy Custom Resource post helm install/update.
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
25.1.0	- In Infiniband mode, due to a kernel bug, there is a limitation on the number of Virtual Functions (VFs) on a single Physical Function (PF). The recommendation is to create up to 16 VFs per PF. Larger number will cause “ip link show dev ” to fail with a “Message too long” error. - SR-IOV switchdev mode is not supported on SLES. - In infiniband mode, in case of existing Intel NICs, loaded *irdma* module should be unloaded before deploying DOCA driver.

24.10.0	- There is a known limitation when using NVIDIA NICs as **primary network interfaces**. If the NVIDIA DOCA Driver container is configured to be deployed, we cannot guarantee that the inbox or pre-installed NVIDIA NIC driver will unload successfully if it remains in use. If the current driver does unload, it removes all NVIDIA NIC networking interfaces and netdevices. DOCA driver container then loads new drivers but only restores **basic configuration** (for example, IP addresses) on the primary network interface's Physical Function (PF) and its Virtual Functions (VFs). More advanced settings (such as VLANs, bonding, and OVS) will **not** be restored automatically. This limitation applies to **all** versions of the NVIDIA Network Operator.
24.10.0	- There is a known limitation when using *docker* on RHEL 8 and 9. If you encounter this issue, it is recommended to use "the preferred, maintained, and supported container runtime of choice for Red Hat Enterprise Linux". For more details, refer to the article Is the docker package available for Red Hat Enterprise Linux 8 and 9? in the Red Hat Knowledge Base. - In NIC Configuration Operator template v0.1.14 BF2/BF3 DPUs (not SuperNICs) FW reset flow isn't supported. - NVIDIA NIC Configuration Operator v0.1.14 Firmware Mismatch notification feature doesn't support NVIDIA BlueField-3 SuperNIC.

24.7.0	- In case ENABLE_NFSRDMA is enabled for DOCA Driver container and NVMe modules are loaded in the host system, NVIDA DOCA Driver Container will fail to load. User should blacklist NVMe modules to prevent them from loading on system boot. If this is not possible (e.g when the system uses NVMe SSD drives) then ENABLE_NFSRDMA must be set to *false*. Using features such as GPU Direct Storage is not supported in such case.
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

Prerequisites

Component	Version	Notes
Kubernetes	>=1.29 and <=1.32	
Helm	v3.5+	For information and methods of Helm installation, please refer to the official Helm Website.
Node Feature Discovery	>=0.15.6 and <=0.17.0	When deploying the Network Operator and GPU Operator on the same cluster, ensure only one instance of Node Feature Discovery (NFD) is installed. We recommend using the version included with the GPU Operator.

System Requirements

- **RDMA-capable NVIDIA network adapters**
    - NVIDIA ConnectX NICs and SuperNICs
    - NVIDIA BlueField Networking Platforms
- **NVIDIA GPU Operator v25.3.x or newer** – required for workloads that use NVIDIA GPUs and GPUDirect RDMA.

Supported NVIDIA Network Adapters

The following adapters have been tested and validated with **NVIDIA Network Operator**:

Product Family	Network Technology	Max Port Speed	Notes
NVIDIA ConnectX-6 NIC	Ethernet & InfiniBand	200 Gb/s	IB RDMA and RoCE
NVIDIA ConnectX-6 Dx NIC	Ethernet	200 Gb/s	RoCE
NVIDIA ConnectX-7 NIC	Ethernet & InfiniBand	400 Gb/s	IB RDMA and RoCE
NVIDIA ConnectX-8 SuperNIC	Ethernet & InfiniBand	800 Gb/s	IB RDMA and RoCE
NVIDIA BlueField-3 DPU	Ethernet	200 Gb/s	NIC mode only; RoCE
NVIDIA BlueField-3 SuperNIC	Ethernet	400 Gb/s	NIC mode only; RoCE

Supported NVIDIA Data Center Systems

The following NVIDIA Data Center systems have been tested and validated with **NVIDIA Network Operator**:

System	CPU Architecture	GPU Architecture	Network Adapter(s)	Operating System(s)	Notes
NVIDIA IGX Orin	Arm (NVIDIA Orin)	NVIDIA Ampere	ConnectX-7	Ubuntu 22.04 (ARM64)	GA (RoCE only, without GPUDirect RDMA)

NVIDIA Grace ARM Server	Arm (NVIDIA Grace)	NVIDIA Hopper	BlueField-3 (NIC Mode)	Ubuntu 22.04 (ARM64) / OCP 4.17 / SLES 15.6	GA (RoCE only, without GPUDirect RDMA)
NVIDIA DGX/HGX GB200 NVL72	Arm (NVIDIA Grace)	NVIDIA Blackwell	ConnectX-7	Ubuntu 24.04 (ARM64)	GA
NVIDIA DGX/HGX B200	x86	NVIDIA Blackwell	BlueField-3 SuperNIC (NIC mode) / ConnectX-7	Ubuntu 22.04 / 24.04 (x86)	GA

Supported Operating Systems and Kubernetes Platforms

NVIDIA Network Operator has been validated on the following OS / platform combinations:

Operating System	Upstream Kubernetes	Red Hat OpenShift	Rancher
Ubuntu 24.04 LTS	1.29–1.32	—	—
Ubuntu 22.04 LTS	1.29–1.32	—	—
Red Hat CoreOS	—	4.15–4.18	—
Red Hat Enterprise Linux 9.4 / 9.2	1.29–1.32	—	—
Red Hat Enterprise Linux 8.10 / 8.8	1.29–1.32	—	—
SUSE Linux Enterprise Server 15 SP6	1.29–1.32	—	1.29–1.3

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	
Red Hat Core OS	No	Yes	
Red Hat Enterprise Linux 9	Yes	Yes	
Red Hat Enterprise Linux 8	Yes	Yes	
SUSE Linux Enterprise Server 15 SP6	Yes	No	

Supported Precompiled Container Images for DOCA Drivers

Overview

To save startup time and operational effort, precompiled DOCA driver container images are available for common OS/flavor/kernel/architecture variants.

The container image tag pattern used for common variants is: **driver_ver-container_ver-kernel_ver-flavor-os-arch**. For example:
`24.07-0.6.1.0-0-6.8.0-49-generic-ubuntu24.04-amd64`

NOTE: For the `generic` flavor of Ubuntu, the default Kernel version is used for precompiling (e.g. *6.8.0-31* for Ubuntu 24.04). Whereas for all other flavors, their latest (at time of DOCA packaging/release) Kernel version is used.

Supported Operating Systems

Currently precompiled DOCA driver container images are provided for the following operating systems:

- Ubuntu 24.04 (amd64/arm64)
- Ubuntu 22.04 (amd64/arm64)

Limitations

- NVIDIA supports precompiled driver containers for the most recently released DOCA GA drivers.
- NVIDIA builds precompiled driver containers for `generic`, `nvidia`, `aws`, `azure`, and `oracle` kernel flavors.
- Precompiled driver containers are currently unsigned.
- If your hosts use a different kernel variant, you can create a custom precompiled driver container and host it in your own container registry. Please refer to Precompiled Container Build Instructions for NVIDIA DOCA Driver Container section.

⚠ Warning

- Only `generic` kernel variant is tested and supported as a GA.
- `nvidia`, `aws`, `azure`, and `oracle` kernel variants are supported as a Tech Preview and have limited testing.

Network Operator Component Matrix

The following component versions are deployed by **NVIDIA Network Operator**:

Component	Origin	Repository	Image Name	Tag
NVIDIA Network Operator	NVIDIA (OSS)	nvcr.io/nvidia/cloud-native	network-operator	v25.4.0
NVIDIA Network Operator	NVIDIA (OSS)	ghcr.io/mellanox	network-operator-init-container	v0.0.3
DOCA-OFED Driver Container	NVIDIA (EULA)	nvcr.io/nvidia/mellanox	doca-driver	25.04-0.6.1.0-2
RDMA Shared Device Plugin	NVIDIA (OSS)	ghcr.io/mellanox	k8s-rdma-shared-dev-plugin	v1.5.3

IB Kubernetes Plugin	NVIDIA (OSS)	ghcr.io/mellanox	ib-kubernetes	v1.1.1
IP Over Infiniband (IPoIB) CNI plugin	NVIDIA (OSS)	ghcr.io/mellanox	ipoib-cni	v1.2.2
NVIDIA IPAM Plugin	NVIDIA (OSS)	ghcr.io/mellanox	nvidia-k8s-ipam	v0.3.7
NVIDIA NIC Feature Discovery	NVIDIA (OSS)	ghcr.io/mellanox	nic-feature-discovery	v0.0.2
DOCA Telemetry Service (DTS)	NVIDIA (EULA)	nvcr.io/nvidia/doca	doca_telemetry	1.16.5-doca2.6.0-host
Node Feature Discovery	Community (OSS)	registry.k8s.io/nfd	node-feature-discovery	v0.15.6
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-25.4.0
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-25.4.0
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-25.4.0
SR-IOV Network Device Plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	sriov-network-device-plugin	v3.9.0
SR-IOV CNI plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	sriov-cni	v2.8.1

InfiniBand SR-IOV CNI plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	ib-sriov-cni	v1.2.1
K8s CNI network plugins	Community (OSS)	ghcr.io/k8snetworkplumbingwg	plugins	v1.6.2-update.1
Multus CNI	Community (OSS)	ghcr.io/k8snetworkplumbingwg	multus-cni	v4.1.0
Whereabouts IPAM Plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	whereabouts	v0.7.0
RDMA CNI plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	rdma-cni	v1.3.0
Open vSwitch CNI plugin	Community (OSS)	ghcr.io/k8snetworkplumbingwg	ovs-cni-plugin	v0.38.2
NVIDIA NIC Configuration Operator	NVIDIA (OSS)	ghcr.io/mellanox	nic-configuration-operator	v1.0.3
NVIDIA NIC Configuration Operator	NVIDIA (OSS)	ghcr.io/mellanox	nic-configuration-operator-daemon	v1.0.3
NVIDIA Maintenance Operator	NVIDIA (OSS)	ghcr.io/mellanox	maintenance-operator	v0.2.2

Getting Started with Kubernetes

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
- Kubernetes device plugins to provide hardware resources required for an accelerated network
- Kubernetes secondary network components for network intensive workloads

Prerequisites

1. You have the `kubectl` and `helm` CLIs available on a client machine.

   You can run the following commands to install the Helm CLI:

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get
helm-3 \
    && chmod 700 get_helm.sh \
    && ./get_helm.sh
```

2. Nodes must be configured with a container engine such CRI-O or containerd.

3. If your cluster uses Pod Security Admission (PSA) to restrict the behavior of pods, label the namespace for the Operator to set the enforcement policy to privileged:

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

4. Node Feature Discovery (NFD) is a dependency for the Operator on each node. By default, NFD master and worker are automatically deployed by the Operator. If NFD is already running in the cluster, then you must disable deploying NFD when you install the Operator. by setting `nfd.enabled=false` Helm value

   One way to determine if NFD is already running in the cluster is to check for a NFD label on your nodes:

```
$ kubectl get nodes -o json | jq '.items[].metadata.labels |
keys | any(startswith("feature.node.kubernetes.io"))'
```

   If the command output is `true`, then NFD is already running in the cluster.

Note

NFD needs to support NodeFeatureRules API or it should be configured to expose the needed NIC labels. Deploying NFD from either NVIDIA Network Operator or NVIDIA GPU Operator will have the correct configurations for both Operators.

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 v25.4.0 \
  --wait
```

View deployed resources

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

OR 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 v25.4.0 >
values.yaml
```

Install with specifying the custom *values.yaml*

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version v25.4.0 \
  -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

First install the Network Operator with NFD enabled:

`values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with

- DOCA driver
- RDMA Shared device plugin configured to a netdev with name ens1f0.

Note: You may need to change the interface names in the NicClusterPolicy to those used by your target nodes.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  rdmaSharedDevicePlugin:
    # [map[ifNames:[ens1f0] name:rdma_shared_device_a]]
    image: k8s-rdma-shared-dev-plugin
    repository: ghcr.io/mellanox
```

```
    version: v1.5.3
    imagePullSecrets: []
    # The config below directly propagates to k8s-rdma-shared-
device-plugin configuration.
    # Replace 'devices' with your (RDMA capable) netdevice name.
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": [],
              "deviceIDs": [],
              "drivers": [],
              "ifNames": ["ens1f0"],
              "linkTypes": []
            }
          }
        ]
      }
```

Network Operator Deployment with Multiple Resources in RDMA Shared Device Plugin

First install the Network Operator with NFD enabled:

`values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

- DOCA driver

- RDMA Shared Device pluging with two RDMA resources - the first mapped to ens1f0 and ens1f1 and the second mapped to ens2f0 and ens2f1.

Note: You may need to change the interface names in the NicClusterPolicy to those used by your target nodes.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  rdmaSharedDevicePlugin:
    # [map[ifNames:[ens1f0 ens1f1] name:rdma_shared_device_a]
map[ifNames:[ens2f0 ens2f1] name:rdma_shared_device_b]]
    image: k8s-rdma-shared-dev-plugin
```

```
    repository: ghcr.io/mellanox
    version: v1.5.3
    imagePullSecrets: []
    # The config below directly propagates to k8s-rdma-shared-
device-plugin configuration.
    # Replace 'devices' with your (RDMA capable) netdevice name.
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": [],
              "deviceIDs": [],
              "drivers": [],
              "ifNames": ["ens1f0","ens1f1"],
              "linkTypes": []
            }
          },
          {
            "resourceName": "rdma_shared_device_b",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": [],
              "deviceIDs": [],
              "drivers": [],
              "ifNames": ["ens2f0","ens2f1"],
              "linkTypes": []
            }
          }
        ]
      }
```

Network Operator Deployment with a Secondary Network

First install the Network Operator with NFD enabled:

`values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with the following enabled:

- Secondary network
- Multus CNI
- Container-networking-plugins CNI plugins
- IPAM Plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

Network Operator Deployment with NVIDIA-IPAM

First install the Network Operator with NFD enabled: `values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with the following enabled:

- Secondary network

- Multus CNI
- Container Networking plugins
- NVIDIA-IPAM plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
  nvIpam:
    image: nvidia-k8s-ipam
    repository: ghcr.io/mellanox
    version: v0.3.7
    imagePullSecrets: []
    enableWebhook: false
```

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

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 DOCA Driver, the following packages should be installed:
>
> - the linux-generic package on Ubuntu hosts
> - the kernel-modules-extra package on the RedHat-based hosts

First install the Network Operator with NFD enabled: `values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

- SR-IOV device plugin configured with a single SR-IOV resource pool
- Secondary network
- Multus CNI
- Container Networking plugins
- IPAM plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: v3.9.0
    imagePullSecrets: []
    config: |
      {
        "resourceList": [
          {
            "resourcePrefix": "nvidia.com",
            "resourceName": "hostdev",
            "selectors": {
              "vendors": ["15b3"],
              "devices": [],
              "drivers": [],
              "pfNames": [],
              "pciAddresses": [],
              "rootDevices": [],
              "linkTypes": [],
              "isRdma": true
            }
          }
        ]
      }
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
```

```
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

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: "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: "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: doca-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.

First install the Network Operator with NFD enabled: `values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed deploy a NicClusterPolicy with:

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

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  rdmaSharedDevicePlugin:
    # [map[linkTypes:[ether] name:rdma_shared_device_a]]
    image: k8s-rdma-shared-dev-plugin
    repository: ghcr.io/mellanox
    version: v1.5.3
    imagePullSecrets: []
    # The config below directly propagates to k8s-rdma-shared-
device-plugin configuration.
    # Replace 'devices' with your (RDMA capable) netdevice name.
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": [],
              "deviceIDs": [],
              "drivers": [],
              "ifNames": [],
              "linkTypes": ["ether"]
            }
          }
        ]
      }
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: v3.9.0
    imagePullSecrets: []
```

```
    config: |
      {
        "resourceList": [
          {
            "resourcePrefix": "nvidia.com",
            "resourceName": "hostdev",
            "selectors": {
              "vendors": [],
              "devices": [],
              "drivers": [],
              "pfNames": [],
              "pciAddresses": [],
              "rootDevices": [],
              "linkTypes": ["IB"],
              "isRdma": true
            }
          }
        ]
      }
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

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

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.

First install the Network Operator with NFD enabled: `values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

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

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  rdmaSharedDevicePlugin:
    # [map[ifNames:[ibs1f0] name:rdma_shared_device_a]]
    image: k8s-rdma-shared-dev-plugin
    repository: ghcr.io/mellanox
```

```
    version: v1.5.3
    imagePullSecrets: []
    # The config below directly propagates to k8s-rdma-shared-
device-plugin configuration.
    # Replace 'devices' with your (RDMA capable) netdevice name.
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": [],
              "deviceIDs": [],
              "drivers": [],
              "ifNames": ["ibs1f0"],
              "linkTypes": []
            }
          }
        ]
      }
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipoib:
      image: ipoib-cni
      repository: ghcr.io/mellanox
      version: 428715a57c0b633e48ec7620f6e3af6863149ccf
```

```
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

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: doca-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:

- NVIDIA DOCA Driver 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

First install the Network Operator with NFD enabled: `values.yaml`:

```
nfd:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

- DOCA driver
- SR-IOV Device Plugin
- Secondary network
- Multus CNI
- Container Networking plugins
- IPAM plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: v3.9.0
```

```
    imagePullSecrets: []
    config: |
      {
        "resourceList": [
          {
            "resourcePrefix": "nvidia.com",
            "resourceName": "hostdev",
            "selectors": {
              "vendors": ["15b3"],
              "devices": [],
              "drivers": [],
              "pfNames": [],
              "pciAddresses": [],
              "rootDevices": [],
              "linkTypes": [],
              "isRdma": true
            }
          }
        ]
      }
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

`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.

First install the Network Operator with NFD and SRIOV Network Operator enabled: `values.yaml`:

```
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

- DOCA driver
- Secondary network
- Multus CNI
- IPoIB CNI
- IPAM CNI plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
```

```
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

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: nvidia-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: nvidia-network-operator
spec:
  maxUnavailable: "10%"
  nodeSelector:
    - matchExpressions:
      - key: some-label
        operator: In
        values:
          - val-2
    - matchExpressions:
      - key: other-label
        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
nvidia-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
nvidia-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:

- NVIDIA DOCA Driver and OpenSM running. OpenSM runs on top of the NVIDIA DOCA Driver 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.
- 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.

First install the Network Operator with NFD and SR-IOV Network Operator enabled: `values.yaml`

```
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true
```

Once the Network Operator is installed create a NicClusterPolicy with:

- DOCA driver
- Secondary network
- Multus CNI
- Container Networking Plugins
- IPAM plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
```

```
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.7.0
      imagePullSecrets: []
```

`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: 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:

- NVIDIA DOCA Driver and OpenSM running. OpenSM runs on top of the NVIDIA DOCA Driver 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.

Note

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.

Warning

ib-kubernetes-ufm-secret should be created before NicClusterPolicy.

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:

`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: ""
```

First install the Network Operator with NFD enabled: `values.yaml`

```
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true
  resourcePrefix: "nvidia.com"
```

Once the Network Operator is installed create a NicClusterPolicy with:

- DOCA driver
- ibKubernetes
- Secondary network
- Multus CNI
- Container Networking plugins
- IPAM Plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    forcePrecompiled: false
    imagePullSecrets: []
    terminationGracePeriodSeconds: 300
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      safeLoad: false
      drain:
        enable: true
        force: true
        podSelector: ""
        timeoutSeconds: 300
        deleteEmptyDir: true
  ibKubernetes:
    image: ib-kubernetes
    repository: ghcr.io/mellanox
    version: v1.1.1
```

```
    imagePullSecrets: []
    pKeyGUIDPoolRangeStart: 02:00:00:00:00:00:00:00
    pKeyGUIDPoolRangeEnd: 02:FF:FF:FF:FF:FF:FF:FF
    ufmSecret: "ib-kubernetes-ufm-secret"
  nvIpam:
    image: nvidia-k8s-ipam
    repository: ghcr.io/mellanox
    version: v0.3.7
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
```

Create IPPool object for nv-ipam

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: IPPool
metadata:
  name: pool1
  namespace: nvidia-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
```

Wait for NVIDIA DOCA Driver 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":"nv-ipam",
      "poolName":"pool1"
    }
}'
```

Note

To use the IB network with Pkey management feature with RDMA isolation, use the following `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: '{
  "cniVersion":"0.3.1",
  "name":"ib-sriov-network",
  "plugins":[
  {
    "type":"ib-sriov",
    "pkey":"0x6",
    "link_state":"enable",
    "ibKubernetesEnabled":true,
    "ipam":{
      "type":"nv-ipam",
      "poolName":"pool1"
    }
  },
  {
    "type":"rdma"
  }
  ]
}'
```

`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, DOCA Driver 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: 25.04-0.6.1.0-2
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: v3.9.0
     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: v1.6.2-update.1-amd64
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
```

```
      version: v0.7.0-amd64
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
```

`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 - managed OpenvSwitch

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

> **⚠ Warning**
>
> To use DOCA Driver container with this mode of operation, set the *RESTORE_DRIVER_ON_POD_TERMINATION* environment variable to *false* in the driver configuration section in the NicClusterPolicy. Restoration to the inbox driver is not supported for this feature.

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

In this mode, the sriov-network-operator automatically creates and configures OpenvSwitch bridges. For more complex scenarios, such as VF lag, you must use the “externally managed OpenvSwitch” feature of the sriov-network-operator, which is detailed in a separate section of the documentation.

Network Operator Configuration

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

First install the Network Operator with NFD enabled: `values.yaml`

```
sriovNetworkOperator:
  enabled: true
```

Once the Network Operator has been installed create a NicClusterPolicy with nv-ipam:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nvIpam:
    image: nvidia-k8s-ipam
    repository: ghcr.io/mellanox
    version: v0.3.7
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
```

Enable `manageSoftwareBridges` featureGate for sriov-network-operator

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n
nvidia-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: nvidia-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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ovs-switchdev
  namespace: nvidia-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: nvidia-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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ovs-switchdev
  namespace: nvidia-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: nvidia-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

Please see the following DOCA documentation for OVS hardware offload verification and troubleshooting steps:

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

Network Operator Deployment and OpenvSwitch offload - externally managed OpenvSwitch with VF lag

> **Warning**
>
> This feature is not compatible with the DOCA Driver container.

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

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

In this mode, the sriov-network-operator is responsible for configuring the physical and virtual functions but will not manage the configuration of the software bridge. The VF LAG and Open vSwitch should be preconfigured on the host.

Network Operator Configuration

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

First install the Network Operator with NFD enabled: `values.yaml`

```
sriovNetworkOperator:
  enabled: true
```

Once the Network Operator has been installed create a NicClusterPolicy with nv-ipam:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nvIpam:
    image: nvidia-k8s-ipam
    repository: ghcr.io/mellanox
    version: v0.3.7
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v4.1.0
      imagePullSecrets: []
```

Switch sriov-network-operator to *systemd* configuration mode.

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

Create IPPool object for nv-ipam

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: IPPool
metadata:
  name: pool1
  namespace: nvidia-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.

Configure Bond interface with netplan

```
# content of /etc/netplan/01-uplink-bond.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    enp4s0f0np0:
      dhcp4: no
      dhcp6: no
    enp4s0f1np1:
      dhcp4: no
      dhcp6: no
  bonds:
    bond0:
      dhcp4: no
      dhcp6: no
      interfaces:
        - enp4s0f0np0
        - enp4s0f1np1
      parameters:
        mode: 802.3ad
```

Replace `enp4s0f0np0` and `enp4s0f1np1` with the right PF names for you node

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
```

Create bridge

Create OVS bridge

```
ovs-vsctl add-br mybr
# this commad may fail with "No such device" error
ovs-vsctl add-port mybr bond0
```

Note: the second command may fail with “No such device” error because bond0 interface is not exist yet.

Sriov Network Operator Configuration

Create SriovNetworkNodePolicy for selected NIC

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ovs-switchdev
  namespace: nvidia-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
```

Create OVSNetwork CR

```
apiVersion: sriovnetwork.openshift.io/v1
kind: OVSNetwork
metadata:
  name: ovs
  namespace: nvidia-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
```

Create OVS bridge

```
ovs-vsctl --no-wait add-br mybr -- set bridge mybr
datapath_type=netdev
# this commad may fail with "No such device" error
ovs-vsctl add-port mybr bond0 -- set Interface bond0 type=dpdk
options:dpdk-lsc-interrupt=true mtu_request=1450
```

Note: the second command may fail with “No such device” error because bond0 interface is not exist yet.

Sriov Network Operator Configuration

Create SriovNetworkNodePolicy for selected NIC

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ovs-switchdev
  namespace: nvidia-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
```

Create OVSNetwork CR

```
apiVersion: sriovnetwork.openshift.io/v1
kind: OVSNetwork
metadata:
  name: ovs
  namespace: nvidia-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

Please see the following DOCA documentation for OVS hardware offload verification and troubleshooting steps:

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

Network Operator Deployment and RDMA exclusive subsystem mode

When RDMA subsystem is in shared mode, RDMA device is accessible in all network namespace. When RDMA device isolation among multiple network namespaces is not needed, shared mode can be used. This mode is enabled by default.

> **Note**
>
> To use RDMA shared mode with MacVlanNetwork please check Network Operator Deployment with RDMA Shared Device Plugin section.

When user wants to assign dedicated RDMA device to a particular network namespace, exclusive mode should be configured.

SR-IOV Network Operator Configuration

First install the Network Operator with NFD and SR-IOV Operator enabled:

`values.yaml`:

```
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true
```

To configure RDMA exclusive mode apply `SriovNetworkPoolConfig` CR and specify `rdmaMode`:

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: rdma-exclusive-pool
  namespace: nvidia-network-operator
spec:
  nodeSelector:
    - matchExpressions:
      - key: feature.node.kubernetes.io/pci-15b3.present
        operator: "Exists"
  rdmaMode: exclusive
```

The `sriovnetwork-node-policy.yaml` configuration should be applied to configure SR-IOV and deploy RDMA CNI:

```
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
```

RDMA CNI plugin is intended to be run as a chained CNI plugin:

```
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: |-
    {
       "type": "nv-ipam",
       "poolName": "pool1"
    }
  metaPlugins: |
    {
      "type": "rdma"
    }
```

Test Workload

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
```

Configure NIC Firmware using the NIC Configuration Operator

NVIDIA NIC Configuration Operator provides Kubernetes API (Custom Resource Definition) to allow Firmware update and configuration on NVIDIA NICs in a coordinated manner. It deploys a configuration daemon on each of the desired nodes to configure NVIDIA NICs there. NVIDIA NIC Configuration Operator uses Maintenance Operator to prepare a node for maintenance before the actual configuration.

Warning

NVIDIA NIC Configuration Operator does not support FW reset flow for DPU mode. Check limitations

Note

To perform Firmware validation and update on NIC devices, NIC Configuration Operator requires a persistent storage set up in the cluster. To set up a persistent NFS storage in the cluster, the example from the CSI NFS Driver repository might be used. After deploying the NFS server and NFS CSI driver, the storage class should become available in the cluster. The name of the storage class should then be passed when configuring the NIC Configuration Operator.

First install the Network Operator helm chart with the Maintenance Operator enabled and deploy a NIC Cluster Policy CRD with NIC Configuration Operator enabled:

`values.yaml`:

```
maintenanceOperator:
  enabled: true
```

`nicclusterpolicy.yaml`:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator
      repository: ghcr.io/mellanox
      version: v1.0.3
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: ghcr.io/mellanox
      version: v1.0.3
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      # Name of the storage class is provided by the user
      storageClassName: nfs-csi
      availableStorageSize: 1Gi
```

Observe the NicDevice CRs detected in the cluster. The name of the CR is composed from the node name, NIC type and its serial number:

```
> kubectl get nicdevices -n nvidia-network-operator
NAME                       AGE
node1-1015-mt1627x08307    1m
node1-101d-mt1952x03330    1m
node2-1015-mt1627x08305    1m
node2-101d-mt1952x03327    1m
```

Discover more information about a specific device:

```
kubectl get nicdevice -n nvidia-network-operator node1-101d-
mt1952x03327 -o yaml
```

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicDevice
metadata:
  creationTimestamp: "2024-09-21T08:43:08Z"
  generation: 1
  name: node1-101d-mt1952x03327
  namespace: nvidia-network-operator
  ownerReferences:
  - apiVersion: v1
    kind: Node
    name: node1
    uid: 25c4f4e2-f7ba-4ba9-9a87-8056313ffc79
  resourceVersion: "1177095"
  uid: ac6763bf-67c6-4af5-81f8-1aad5da929bf
spec: {}
status:
  conditions:
  - type: FirmwareUpdateInProgress
    status: "False"
    reason: DeviceFirmwareSpecEmpty
    message: Device firmware spec is empty, cannot update or
validate firmware
    lastTransitionTime: "2024-09-21T08:43:04Z"
  - type: ConfigUpdateInProgress
    status: "False"
    reason: DeviceConfigSpecEmpty
    message: Device configuration spec is empty, cannot update
configuration
    lastTransitionTime: "2024-09-21T08:43:08Z"
firmwareVersion: 22.39.1015
node: cloud-dev-41
partNumber: mcx623106ac-cdat
ports:
- networkInterface: enp3s0f0np0
  pci: "0000:03:00.0"
```

```
  rdmaInterface: mlx5_0
- networkInterface: enp3s0f1np1
  pci: "0000:03:00.1"
  rdmaInterface: mlx5_1
psid: mt_0000000436
serialNumber: mt1952x03327
type: 101d
```

Configure and apply the NICFirmwareSource CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareSource
metadata:
  name: connectx6-dx-firmware-22-44-1036
  namespace: nvidia-network-operator
  finalizers:
    - configuration.net.nvidia.com/nic-configuration-operator
spec:
  # a list of firmware binaries zip archives from the Mellanox
website, can point to any url accessible from the cluster
  binUrlSources:
    - https://www.mellanox.com/downloads/firmware/fw-ConnectX6Dx-
rel-22_44_1036-MCX623106AC-CDA_Ax-UEFI-14.37.14-FlexBoot-
3.7.500.signed.bin.zip
```

Observe the NICFirmwareSource status:

```
> kubectl get nicfirmwaresource -n nvidia-network-operator
connectx6-dx-firmware-22-44-1036 -o yaml
...
status:
  state: Success
  versions:
    22.44.1036:
    - mt_0000000436
```

Configure and apply the NicFirmwareTemplate CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareTemplate
metadata:
  name: connectx6dx-config
  namespace: nvidia-network-operator
spec:
  nodeSelector:
    kubernetes.io/hostname: node1
  nicSelector:
    nicType: "101d"
  template:
    nicFirmwareSourceRef: connectx6dx-firmware-22-44-1036
    updatePolicy: Update
```

Configure and apply the NicConfigurationTemplate CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
   name: connectx6-config
   namespace: nvidia-network-operator
spec:
   nodeSelector:
      feature.node.kubernetes.io/network-sriov.capable: "true"
   nicSelector:
      # nicType selector is mandatory the rest are optional. Only
a single type can be specified.
      nicType: 101d
      pciAddresses:
         - "0000:03:00.0"
         - “0000:04:00.0”
      serialNumbers:
         - "mt1952x03327"
   resetToDefault: false # if set, template is ignored, device
configuration should reset
   template:
      # numVfs and linkType fields are mandatory, the rest are
optional
      numVfs: 2
      linkType: Ethernet
      pciPerformanceOptimized:
         enabled: true
         maxReadRequest: 4096
      roceOptimized:
         enabled: true
         qos:
            trust: dscp
            pfc: "0,0,0,1,0,0,0,0"
      gpuDirectOptimized:
         enabled: true
```

```
        env: Baremetal
```

Note

It's not possible to apply more than one template of each kind (NICFirmwareTemplate or NICConfigurationTemplate) to a single device. In this case, no template will be applied and an error event will be emitted for the corresponding NicDevice CR.

Note

To use the NIC Configuration Operator functionality together with SR-IOV Network Operator, "mellanox" plugin should be disabled in the SR-IOV Network Operator.

For more information about the CRD API, refer to API documentation. For more information, which FW parameter each settings corresponds to, refer to Configuration details doc section.

Spec of the NicDevice CR is updated in accordance with the NICFirmwareTemplate and NicConfigurationTemplate CRs matching the device

```
> kubectl get nicdevice -n nvidia-network-operator node1-101d-
mt1952x03327 -o jsonpath='{.spec}' | yq -P
template:
  firmware:
      nicFirmwareSourceRef: connectx6dx-firmware-22-44-1036
      updatePolicy: Update
  configuration:
      numVfs: 2
      linkType: Ethernet
      pciPerformanceOptimized:
        enabled: true
      roceOptimized:
        enabled: true
        qos:
            trust: dscp
            pfc: "0,0,0,1,0,0,0,0"
      gpuDirectOptimized:
        enabled: true
        env: Baremetal
```

Status conditions of the NicDevice CR reflect the status of the configuration update and indicate any errors that might occur during the process

```
> kubectl get nicdevice -n nvidia-network-operator node1-101d-
mt1952x03327 -o jsonpath='{.status.conditions}' | yq -P
- type: FirmwareUpdateInProgress
  status: "False"
  reason: DeviceFirmwareConfigMatch
  message: Firmware matches the requested version
  observedGeneration: 4
  lastTransitionTime: "2024-09-21T08:42:23Z"
- type: ConfigUpdateInProgress
  status: "True"
  reason: UpdateStarted
  message: ""
  lastTransitionTime: "2024-09-21T08:43:08Z"
```

NIC Firmware Mismatch Notification

NIC Configuration Operator updates status conditions of the NicDevice CR to set *FirmwareConfigMatch* condition based on a current NIC firmware:

```
> kubectl get nicdevice -n nvidia-network-operator node1-101d-
mt1952x03327 -o jsonpath='{.status.conditions}' | yq -P
- type: FirmwareConfigMatch
  status: "True"
  reason: DeviceFirmwareConfigMatch
  message: Device firmware '20.42.1000' matches to recommended
version '20.42.1000'
  lastTransitionTime: "2024-09-21T08:43:10Z"
```

FirmwareConfigMatch condition status is set to *Unknown* if DOCA Driver is not installed otherwise it notifies if current NIC firmware is recommended or not recommended by DOCA Driver. E.g.:

```
 > kubectl get nicdevice -n nvidia-network-operator node1-101d-
mt1952x03327 -o jsonpath='{.status.conditions}' | yq -P
- type: FirmwareConfigMatch
  status: "True"
  reason: DeviceFirmwareConfigMatch
  message: Device firmware '20.42.1000' matches to recommended
version '20.42.1000'
  lastTransitionTime: "2024-11-08T09:19:41Z"
```

Getting Started with Red Hat OpenShift

Network Operator Deployment on an OpenShift Container Platform

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

> ⚠ **Warning**
>
> Automatic DOCA Driver Upgrade doesn't work on Single Node OpenShift (SNO) deployments.

Node Feature Discovery

To enable Node Feature Discovery, please follow the official guide. A single instance of Node Feature Discovery is expected to be used in the cluster.

An example of Node Feature Discovery configuration:

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: nfd-instance
  namespace: openshift-nfd
spec:
  operand:
    image: registry.redhat.io/openshift4/ose-node-feature-
discovery-rhel9:v4.16
    imagePullPolicy: Always
  workerConfig:
    configData: |
      sources:
        pci:
          deviceClassWhitelist:
            - "02"
            - "03"
            - "0200"
            - "0207"
          deviceLabelFields:
            - vendor
```

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

For more details please read official NFD documentation.

```
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 NVIDIA DOCA Driver 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: "v25.4"
  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: 25.04-0.6.1.0-2
     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: v3.9.0
      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: "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: doca-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: 25.04-0.6.1.0-2
     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: 25.04-0.6.1.0-2
     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: v1.5.3
```

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
  - General Parameters
    - ImagePullSecrets customization
  - NFD labels
  - Node Feature Discovery
  - SR-IOV Network Operator
  - NIC Configuration Operator
  - Maintenance Operator
  - Helm customization file
- CRDs
  - mellanox.com/v1alpha1
    - AppliedState
    - ConfigMapNameReference
    - DOCATelemetryServiceConfig
    - DOCATelemetryServiceSpec
    - DevicePluginSpec
    - DrainSpec
    - DriverUpgradePolicySpec
    - HostDeviceNetwork
    - HostDeviceNetworkSpec
    - HostDeviceNetworkStatus
    - IBKubernetesSpec
    - IPoIBNetwork
    - IPoIBNetworkSpec
    - IPoIBNetworkStatus
    - ImageSpec
    - ImageSpecWithConfig
    - MacvlanNetwork
    - MacvlanNetworkSpec
    - MacvlanNetworkStatus
    - MultusSpec
    - NICFeatureDiscoverySpec
    - NVIPAMSpec
    - NicClusterPolicy
    - NicClusterPolicySpec
    - NicClusterPolicyStatus
    - NicConfigurationOperatorSpec
    - NicFirmwareStorageSpec
    - OFEDDriverSpec

        - PodProbeSpec
        - ResourceRequirements
        - SecondaryNetworkSpec
        - SpectrumXOperatorSpec
        - `string`
        - WaitForCompletionSpec
- NicClusterPolicy Reference Example

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
- Node Feature Discovery
- SR-IOV Network Operator
- NIC Configuration Operator
- Maintenance Operator
- Helm customization file

General Parameters

Name	Type	
imagePullSecrets	list	*[]*

maintenanceOperator.enabled	bool	*false*
nfd.deployNodeFeatureRules	bool	*true*
nfd.enabled	bool	*true*
nicConfigurationOperator	object	*{“enabled”:false}*
nicConfigurationOperator.enabled	bool	*false*
operator.admissionController.enabled	bool	*false*
operator.admissionController.useCertManager	bool	*true*

operator.affinity.nodeAffinity	yaml	<pre>preferredDuringScheduli - weight: 1 preference: matchExpression - key: "nod operator: values: [- weight: 1 preference: matchExpression - key: "nod operator: values: [</pre>
operator.cniBinDirectory	string	*“/opt/cni/bin”*
operator.fullnameOverride	string	*“”*
operator.image	string	*“network-operator”*
operator.maintenanceOperator	object	*{“nodeMaintenanceNamePrefix”:”r operator”,”nodeMaintenanceName*
operator.nameOverride	string	*“”*

operator.nodeSelector	object	*{}*
operator.ofedDriver.initContainer.enable	bool	*true*
operator.ofedDriver.initContainer.image	string	*“network-operator-init-container”*
operator.ofedDriver.initContainer.repository	string	*“ghcr.io/mellanox”*
operator.ofedDriver.initContainer.version	string	*“v0.0.3”*
operator.repository	string	*“nvcr.io/nvidia/cloud-native”*

operator.resources	yaml	`limits:`
		`  cpu: 500m`
		`  memory: 128Mi`
		`requests:`
		`  cpu: 5m`
		`  memory: 64Mi`

operator.tolerations	yaml	`- key: "node-role.kuber`
		`  operator: "Equal"`
		`  value: ""`
		`  effect: "NoSchedule"`
		`- key: "node-role.kuber`
		`  operator: "Equal"`
		`  value: ""`
		`  effect: "NoSchedule"`

operator.useDTK	bool	*true*
sriovNetworkOperator.enabled	bool	*false*

upgradeCRDs	bool	*true*

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

Node Feature Discovery

Node Feature Discovery Helm chart customization options can be found here. Following is a list of overriden values by NVIDIA Network Operator Helm Chart:

Name	Type	Default in NVIDIA Network Operator

node-feature-discovery.enableNodeFeatureApi	bool	*true*
node-feature-discovery.featureGates.NodeFeatureAPI	bool	*true*
node-feature-discovery.gc.enable	bool	*true*
node-feature-discovery.gc.replicaCount	int	*1*
node-feature-discovery.gc.serviceAccount.create	bool	*false*

node-feature-discovery.gc.serviceAccount.name	string	*“node-feature-discovery”*

node-feature-discovery.master	yaml	`serviceAccount:`
		`  name: node-feature-discovery`
		`  create: true`
		`config:`
		`  extraLabelNs: ["nvidia.com"]`

node-feature-discovery.postDeleteCleanup	bool	*false*

node-feature-discovery.worker	yaml	<pre>serviceAccount: *# disable creation to avoid duplicate serviceaccount creation by master spec* *# above* name: node-feature-discovery create: false tolerations: - key: "node-role.kubernetes.io/master" operator: "Exists" effect: "NoSchedule" - key: "node-role.kubernetes.io/control-plane" operator: "Exists" effect: "NoSchedule" - key: nvidia.com/gpu operator: Exists effect: NoSchedule config: sources: pci: deviceClassWhitelist: - "0300" - "0302" deviceLabelFields: - vendor</pre>

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 Network Operator Helm Chart:

Name	Ty
sriov-network-operator.images.ibSriovCni	stri
sriov-network-operator.images.operator	stri
sriov-network-operator.images.ovsCni	stri
sriov-network-operator.images.resourcesInjector	stri
sriov-network-operator.images.sriovCni	stri
sriov-network-operator.images.sriovConfigDaemon	stri
sriov-network-operator.images.sriovDevicePlugin	stri
sriov-network-operator.images.webhook	stri

sriov-network-operator.operator.admissionControllers	yam

sriov-network-operator.operator.admissionControllers.certificates.certManager.enabled	boo
sriov-network-operator.operator.admissionControllers.certificates.certManager.generateSelfSigned	boo

sriov-network-operator.operator.admissionControllers.certificates.custom	obj
sriov-network-operator.operator.resourcePrefix	stri
sriov-network-operator.sriovOperatorConfig.configDaemonNodeSelector	yam
sriov-network-operator.sriovOperatorConfig.deploy	boo

NIC Configuration Operator

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

Name	Type	Default in NVIDIA Network Operator	Notes
nic-configuration-operator-chart.configDaemon.image.name	string	*"nic-configuration-operator-daemon"*	
nic-configuration-operator-chart.configDaemon.image.repository	string	*"ghcr.io/mellanox"*	
nic-configuration-operator-chart.configDaemon.image.tag	string	*"v1.0.3"*	
nic-configuration-operator-chart.operator.image.name	string	*"nic-configuration-operator"*	
nic-configuration-operator-chart.operator.image.repository	string	*"ghcr.io/mellanox"*	

nic-configuration-operator-chart.operator.image.tag	string	*"v1.0.3"*	

Maintenance Operator

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

Name	Type	D N
maintenance-operator-chart.operator.admissionController.certificates.certManager.enable	bool	
maintenance-operator-chart.operator.admissionController.certificates.certManager.generateSelfSigned	bool	
maintenance-operator-chart.operator.admissionController.certificates.custom.enable	bool	
maintenance-operator-chart.operator.admissionController.certificates.secretNames.operator	string	
maintenance-operator-chart.operator.admissionController.enable	bool	
maintenance-operator-chart.operator.image.name	string	
maintenance-operator-chart.operator.image.repository	string	

maintenance-operator-chart.operator.image.tag	string	

Helm customization file

> ⚠ **Warning**
>
> 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
```

Network Operator API reference v1alpha1

Packages:

- mellanox.com/v1alpha1

mellanox.com/v1alpha1

Package v1alpha1 contains API Schema definitions for the mellanox.com v1alpha1 API group

Resource Types:

AppliedState

(*Appears on:* HostDeviceNetworkStatus, NicClusterPolicyStatus)

AppliedState defines a finer-grained view of the observed state of NicClusterPolicy

Field	Description
`name` string	Name of the deployed component this state refers to
`state` State	The state of the deployed component. (“ready”, “notReady”, “ignore”, “error”)
`message` string	Message is a human readable message indicating details about why the state is in this condition

ConfigMapNameReference

(*Appears on:* OFEDDriverSpec, SpectrumXOperatorSpec)

ConfigMapNameReference references a config map in a specific namespace. The namespace must be specified at the point of use.

Field	Description
`name` string	Name of the ConfigMap

DOCATelemetryServiceConfig

(*Appears on:* DOCATelemetryServiceSpec)

DOCATelemetryServiceConfig contains configuration for the DOCATelemetryService.

Field	Description
`fromConfigMap` string	*(Optional)* FromConfigMap sets the configMap the DOCATelemetryService gets its configuration from. The ConfigMap must be in the same namespace as the NICClusterPolicy.

DOCATelemetryServiceSpec

(*Appears on:* NicClusterPolicySpec)

DOCATelemetryServiceSpec is the configuration for DOCA Telemetry Service.

Field	Description
`ImageSpec` ImageSpec	Image information for DOCA Telemetry Service
`config` DOCATelemetryServiceConfig	*(Optional)* Config contains custom config for the DOCATelemetryService. If set no default config will be deployed.

DevicePluginSpec

(*Appears on:* NicClusterPolicySpec)

DevicePluginSpec describes configuration options for device plugin 1. Image information for device plugin 2. Device plugin configuration

Field	Description
`ImageSpecWithConfig` ImageSpecWithConfig	Image information for the device plugin and optional configuration
`useCdi` bool	Enables use of container device interface (CDI) NOTE: NVIDIA Network Operator does not configure container runtime to enable CDI.

DrainSpec

(*Appears on:* DriverUpgradePolicySpec)

DrainSpec describes configuration for node drain during automatic upgrade

Field	Description

`enable` bool	*(Optional)* Enable indicates if node draining is allowed during upgrade
`force` bool	*(Optional)* Force indicates if force draining is allowed
`podSelector` string	*(Optional)* PodSelector specifies a label selector to filter pods on the node that need to be drained For more details on label selectors, see: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors
`timeoutSeconds` int	*(Optional)* TimeoutSecond specifies the length of time in seconds to wait before giving up drain, zero means infinite
`deleteEmptyDir` bool	*(Optional)* DeleteEmptyDir indicates if should continue even if there are pods using emptyDir (local data that will be deleted when the node is drained)

DriverUpgradePolicySpec

(*Appears on:* OFEDDriverSpec)

DriverUpgradePolicySpec describes policy configuration for automatic upgrades

Field	Description
`autoUpgrade` bool	*(Optional)* AutoUpgrade is a global switch for automatic upgrade feature if set to false all other options are ignored
`maxParallelUpgrades` int	*(Optional)* MaxParallelUpgrades indicates how many nodes can be upgraded in parallel 0 means no limit, all nodes will be upgraded in parallel
`waitForCompletion` WaitForCompletionSpec	The configuration for waiting on pods completions
`drain` DrainSpec	The configuration for node drain during automatic upgrade
`safeLoad` bool	*(Optional)* SafeLoad turn on safe driver loading (cordon and drain the node before loading the driver)

HostDeviceNetwork

HostDeviceNetwork is the Schema for the hostdevicenetworks API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` HostDeviceNetworkSpec	Defines the desired state of HostDeviceNetwork
`status` HostDeviceNetworkStatus	Defines the observed state of HostDeviceNetwork

HostDeviceNetworkSpec

(*Appears on:* HostDeviceNetwork)

HostDeviceNetworkSpec defines the desired state of HostDeviceNetwork

Field	Description
`networkNamespace` string	Namespace of the NetworkAttachmentDefinition custom resource
`resourceName` string	Host device resource pool name
`ipam` string	IPAM configuration to be used for this network

HostDeviceNetworkStatus

(*Appears on:* HostDeviceNetwork)

HostDeviceNetworkStatus defines the observed state of HostDeviceNetwork

Field	Description
`state` State	Reflects the state of the HostDeviceNetwork
`hostDeviceNetworkAttachmentDef` string	Network attachment definition generated from HostDeviceNetworkSpec
`reason` string	Informative string in case the observed state is error
`appliedStates` []AppliedState	AppliedStates provide a finer view of the observed state

IBKubernetesSpec

(*Appears on:* NicClusterPolicySpec)

IBKubernetesSpec describes configuration options for ib-kubernetes

Field	Description
`ImageSpec` ImageSpec	Image information for ib-kubernetes
`periodicUpdateSeconds` int	*(Optional)* Interval of updates in seconds
`pKeyGUIDPoolRangeStart` string	The first guid in the pool
`pKeyGUIDPoolRangeEnd` string	The last guid in the pool
`ufmSecret` string	Secret containing credentials to UFM service

IPoIBNetwork

IPoIBNetwork is the Schema for the ipoibnetworks API

Field	Description

`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` IPoIBNetworkSpec	Defines the desired state of IPoIBNetwork
`status` IPoIBNetworkStatus	Defines the observed state of IPoIBNetwork

IPoIBNetworkSpec

(*Appears on:* IPoIBNetwork)

IPoIBNetworkSpec defines the desired state of IPoIBNetwork

Field	Description
`networkNamespace` string	Namespace of the NetworkAttachmentDefinition custom resource
`master` string	Name of the host interface to enslave. Defaults to default route interface
`ipam` string	IPAM configuration to be used for this network.

IPoIBNetworkStatus

(*Appears on:* IPoIBNetwork)

IPoIBNetworkStatus defines the observed state of IPoIBNetwork

Field	Description
`state` State	Reflects the state of the IPoIBNetwork
`ipoibNetworkAttachmentDef` string	Network attachment definition generated from IPoIBNetworkSpec
`reason` string	Informative string in case the observed state is error

ImageSpec

(*Appears on:* DOCATelemetryServiceSpec, IBKubernetesSpec, ImageSpecWithConfig, NICFeatureDiscoverySpec, NVIPAMSpec, NicConfigurationOperatorSpec, OFEDDriverSpec, SecondaryNetworkSpec, SpectrumXOperatorSpec)

ImageSpec Contains container image specifications

Field	Description
`image` string	Name of the image
`repository` string	Address of the registry that stores the image
`version` string	Version of the image to use
`imagePullSecrets` []string	*(Optional)* ImagePullSecrets is an optional list of references to secrets in the same namespace to use for pulling the image
`containerResources` []ResourceRequirements	ResourceRequirements describes the compute resource requirements

ImageSpecWithConfig

(*Appears on:* DevicePluginSpec, MultusSpec)

ImageSpecWithConfig Contains ImageSpec and optional configuration

Field	Description
`ImageSpec` ImageSpec	Image information for the component
`config` string	Configuration for the component as a string

MacvlanNetwork

MacvlanNetwork is the Schema for the macvlannetworks API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` MacvlanNetworkSpec	Defines the desired state of MacvlanNetworkSpec
`status` MacvlanNetworkStatus	Defines the observed state of MacvlanNetwork

MacvlanNetworkSpec

(*Appears on:* MacvlanNetwork)

MacvlanNetworkSpec defines the desired state of MacvlanNetwork

Field	Description
`networkNamespace` string	Namespace of the NetworkAttachmentDefinition custom resource
`master` string	Name of the host interface to enslave. Defaults to default route interface
`mode` string	Mode of interface one of “bridge”, “private”, “vepa”, “passthru”
`mtu` int	MTU of interface to the specified value. 0 for master’s MTU
`ipam` string	IPAM configuration to be used for this network.

MacvlanNetworkStatus

(*Appears on:* MacvlanNetwork)

MacvlanNetworkStatus defines the observed state of MacvlanNetwork

Field	Description
`state` State	Reflects the state of the MacvlanNetwork
`macvlanNetworkAttachmentDef` string	Network attachment definition generated from MacvlanNetworkSpec
`reason` string	Informative string in case the observed state is error

MultusSpec

(*Appears on:* SecondaryNetworkSpec)

MultusSpec describes configuration options for Multus CNI 1. Image information for Multus CNI 2. Multus CNI config if config is missing or empty then multus config will be automatically generated from the CNI configuration file of the master plugin (the first file in lexicographical order in cni-conf-dir)

Field	Description
`ImageSpecWithConfig` ImageSpecWithConfig	Image information for Multus and optional configuration

NICFeatureDiscoverySpec

(*Appears on:* NicClusterPolicySpec)

NICFeatureDiscoverySpec describes configuration options for nic-feature-discovery

Field	Description
`ImageSpec` ImageSpec	Image information for nic-feature-discovery

NVIPAMSpec

(*Appears on:* NicClusterPolicySpec)

NVIPAMSpec describes configuration options for nv-ipam 1. Image information for nv-ipam 2. Configuration for nv-ipam

Field	Description
`enableWebhook` bool	Enable deployment of the validation webhook
`ImageSpec` ImageSpec	Image information for nv-ipam

NicClusterPolicy

NicClusterPolicy is the Schema for the nicclusterpolicies API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicClusterPolicySpec	Defines the desired state of NicClusterPolicy
`status` NicClusterPolicyStatus	Defines the observed state of NicClusterPolicy

NicClusterPolicySpec

(*Appears on:* NicClusterPolicy)

NicClusterPolicySpec defines the desired state of NicClusterPolicy

Field	Description

`ofedDriver` OFEDDriverSpec	OFEDDriver is a specialized driver for NVIDIA NICs which driver that comes with an OS. See https://network.nvidia matrix/
`rdmaSharedDevicePlugin` DevicePluginSpec	RdmaSharedDevicePlugin manages support IB and RoC Kubernetes device plugin framework. The config field is the RDMA shared device plugin configuration. See https://github.com/Mellanox/k8s-rdma-shared-dev-plug
`sriovDevicePlugin` DevicePluginSpec	SriovDevicePlugin manages SRIOV through the Kuberne framework. The config field is a json representation of t plugin configuration. See https://github.com/k8snetwor network-device-plugin
`ibKubernetes` IBKubernetesSpec	IBKubernetes provides a daemon that works in conjunct Network Device Plugin. It acts on Kubernetes pod objec pod's network annotation. From there it fetches the cor and reads the PKey. This is done in order to add the new predefined GUID in the GUID field of the CRD. This is th that PKey for pods with mellanox.infiniband.app annota https://github.com/Mellanox/ib-kubernetes
`secondaryNetwork` SecondaryNetworkSpec	SecondaryNetwork Specifies components to deploy in c secondary network in Kubernetes. It consists of the foll components: - Multus-CNI: Delegate CNI plugin to supp Kubernetes - CNI plugins: Currently only containernetw - IPAM CNI: Currently only Whereabout IPAM CNI is supp secondaryNetwork section. - IPoIB CNI: Allows the user and move it to the pod
`nvIpam` NVIPAMSpec	NvIpam is an IPAM provider that dynamically assigns IP performance in mind. Note: NvIPam requires certificate manager or OpenShift cert management. See https://gi k8s-ipam
`nicFeatureDiscovery` NICFeatureDiscoverySpec	NicFeatureDiscovery works with NodeFeatureDiscovery about NVIDIA NICs. https://github.com/Mellanox/nic-fea
`docaTelemetryService` DOCATelemetryServiceSpec	DOCATelemetryService exposes telemetry from NVIDIA to prometheus. See: https://docs.nvidia.com/doca/sdk/nvidia+doca+telemetr
`nicConfigurationOperator` NicConfigurationOperatorSpec	NicConfigurationOperator provides Kubernetes CRD AP on NVIDIA NICs in a coordinated manner See: https://git configuration-operator
`spectrumXOperator` SpectrumXOperatorSpec	SpectrumXOperator exposes NVIDIA Spectrum-X Opera https://github.com/Mellanox/spectrum-x-operator/

`nodeAffinity` Kubernetes core/v1.NodeAffinity	NodeAffinity rules to inject to the DaemonSets objects operator
`tolerations` []Kubernetes core/v1.Toleration	Tolerations to inject to the DaemonSets objects that ar

NicClusterPolicyStatus

(*Appears on:* NicClusterPolicy)

NicClusterPolicyStatus defines the observed state of NicClusterPolicy

Field	Description
`state` State	Reflects the current state of the cluster policy
`reason` string	Informative string in case the observed state is error
`appliedStates` []AppliedState	AppliedStates provide a finer view of the observed state

NicConfigurationOperatorSpec

(*Appears on:* NicClusterPolicySpec)

NicConfigurationOperatorSpec is the configuration for NIC Configuration Operator

Field	Description
`operator` ImageSpec	Image information for nic-configuration-operator
`configurationDaemon` ImageSpec	Image information for nic-configuration-daemon
`nicFirmwareStorage` NicFirmwareStorageSpec	NicFirmwareStorage contains configuration for the NIC firmware storage
`logLevel` string	LogLevel sets the verbosity level of the logs. info|debug

NicFirmwareStorageSpec

(*Appears on:* NicConfigurationOperatorSpec)

NicFirmwareStorageSpec contains configuration for the NIC firmware storage

Field	Description
`create` bool	Create specifies whether to create a new PVC or use an existing one If create == false, the existing PVC should be located in the same namespace as the operator
`pvcName` string	PVCName is the name of the PVC to mount as NIC Firmware storage. Default value: “nic-fw-storage-pvc”
`storageClassName` string	StorageClassName is the name of a storage class to be used to store NIC FW binaries during NIC FW upgrade. If not provided, the cluster-default storage class will be used
`availableStorageSize` string	AvailableStorageSize is storage size for the NIC Configuration Operator to request. Only applies if nicFirmwareStorage.create == true. Default value: 1Gi

OFEDDriverSpec

(*Appears on:* NicClusterPolicySpec)

OFEDDriverSpec describes configuration options for DOCA Driver Container

Field	Description
`ImageSpec` ImageSpec	Image information for DOCA driver container
`startupProbe` PodProbeSpec	Pod startup probe settings

`livenessProbe` PodProbeSpec	Pod liveness probe settings
`readinessProbe` PodProbeSpec	Pod readiness probe settings
`env` []Kubernetes core/v1.EnvVar	List of environment variables to set in the DOCA driver container.
`upgradePolicy` DriverUpgradePolicySpec	DOCA driver auto-upgrade settings
`certConfig` ConfigMapNameReference	Optional: Custom TLS certificates configuration for DOCA driver container
`repoConfig` ConfigMapNameReference	Optional: Custom package repository configuration for DOCA driver container
`terminationGracePeriodSeconds` int64	*(Optional)* TerminationGracePeriodSeconds specifies the length of time in seconds to wait before killing the DOCA driver container pod on termination
`forcePrecompiled` bool	*(Optional)* ForcePrecompiled specifies if only DOCA driver precompiled images are allowed If set to false and precompiled image does not exists, DOCA driver will be compiled on Nodes If set to true and precompiled image does not exists, OFED state will be Error.

PodProbeSpec

(*Appears on:* OFEDDriverSpec)

PodProbeSpec describes a pod probe.

Field	Description
`initialDelaySeconds` int	Number of seconds after the container has started before the probe is initiated
`periodSeconds` int	How often (in seconds) to perform the probe

ResourceRequirements

(*Appears on:* ImageSpec)

ResourceRequirements describes the compute resource requirements.

Field	Description
`name` string	Name of the container the requirements are set for
`limits` Kubernetes core/v1.ResourceList	*(Optional)* Limits describes the maximum amount of compute resources allowed. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
`requests` Kubernetes core/v1.ResourceList	*(Optional)* Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

SecondaryNetworkSpec

(*Appears on:* NicClusterPolicySpec)

SecondaryNetworkSpec describes configuration options for secondary network

Field	Description
`multus` MultusSpec	Image and configuration information for multus
`cniPlugins` ImageSpec	Image information for CNI plugins
`ipoib` ImageSpec	Image information for IPoIB CNI
`ipamPlugin` ImageSpec	Image information for IPAM plugin

SpectrumXOperatorSpec

(*Appears on:* NicClusterPolicySpec)

SpectrumXOperatorSpec describes configuration options for NVIDIA Spectrum-X Operator

Field	Description
`ImageSpec` ImageSpec	Image information for NVIDIA Spectrum-X Operator
`spectrumXConfig` ConfigMapNameReference	Spectrum-X Operator ConfigMap name
`sriovObjNamespace` string	SR-IOV Network Operator and related CRDs namespace

State (`string` alias)

(*Appears on:* AppliedState, HostDeviceNetworkStatus, IPoIBNetworkStatus, MacvlanNetworkStatus, NicClusterPolicyStatus)

State represents reconcile state of the system.

WaitForCompletionSpec

(*Appears on:* DriverUpgradePolicySpec)

WaitForCompletionSpec describes the configuration for waiting on pods completions

Field	Description
`podSelector` string	*(Optional)* PodSelector specifies a label selector for the pods to wait for completion For more details on label selectors, see: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors
`timeoutSeconds` int	*(Optional)* TimeoutSecond specifies the length of time in seconds to wait before giving up on pod termination, zero means infinite

NicClusterPolicy Custom Resource Example

The following NIC Cluster Policy example contains all the sub-components that NVIDA Network Operator can deploy. This example should serve as a reference, it is not recommended to apply it as is to your cluster.

NOTE: Edit the example to contain only the required components for the target environment.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.04-0.6.1.0-2
    upgradePolicy:
      autoUpgrade: true
      drain:
        deleteEmptyDir: true
        enable: true
        force: true
        timeoutSeconds: 300
      maxParallelUpgrades: 1
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 10
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
  rdmaSharedDevicePlugin:
    image: k8s-rdma-shared-dev-plugin
    repository: ghcr.io/mellanox
    version: v1.5.3
    # The config below directly propagates to k8s-rdma-shared-
device-plugin configuration.
    # Replace 'devices' with your (RDMA capable) netdevice name.
    config: |
      {
```

```
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": ["15b3"],
              "deviceIDs": ["101b"]
            }
          }
        ]
      }
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: v3.9.0
    config: |
      {
        "resourceList": [
          {
            "resourcePrefix": "nvidia.com",
            "resourceName": "hostdev",
            "selectors": {
              "vendors": ["15b3"],
              "isRdma": true
            }
          }
        ]
      }
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v1.6.2-update.1
    ipoib:
      image: ipoib-cni
      repository: ghcr.io/mellanox
```

```
    version: v1.2.2
  multus:
    image: multus-cni
    repository: ghcr.io/k8snetworkplumbingwg
    version: v4.1.0
  ipamPlugin:
    image: whereabouts
    repository: ghcr.io/k8snetworkplumbingwg
    version: v0.7.0
nvIpam:
  image: nvidia-k8s-ipam
  repository: ghcr.io/mellanox
  version: v0.3.7
  enableWebhook: false
ibKubernetes:
  image: ib-kubernetes
  repository: ghcr.io/mellanox/ib-kubernetes
  version: v1.1.1
  pKeyGUIDPoolRangeStart: "02:00:00:00:00:00:00:00"
  pKeyGUIDPoolRangeEnd: "02:FF:FF:FF:FF:FF:FF:FF"
  ufmSecret: ufm-secret
nicFeatureDiscovery:
  image: nic-feature-discovery
  repository: ghcr.io/mellanox
  version: v0.0.2
docaTelemetryService:
  image: doca_telemetry
  repository: nvcr.io/nvidia/doca
  version: 1.16.5-doca2.6.0-host
nicConfigurationOperator:
  operator:
    image: nic-configuration-operator
    repository: ghcr.io/mellanox
    version: v1.0.3
  configurationDaemon:
    image: nic-configuration-operator-daemon
```

```
    repository: ghcr.io/mellanox
    version: v1.0.3
  nicFirmwareStorage:
    create: true
    pvcName: nic-fw-storage-pvc
    storageClassName: nic-fw-storage-class
    availableStorageSize: 1Gi
  logLevel: info
```

Life Cycle Management

Network Operator Versioning

NVIDIA Network Operator is versioned following the calendar versioning convention.

The version follows the pattern `YY.MM.PP`, such as 25.1.0, 24.10.0, and 24.10.1. The first two fields, `YY.MM` identify a major version and indicates when the major version was initially released. The third field, `PP`, identifies the patch version of the major version. Patch releases typically include critical bug and CVE fixes.

Network Operator Life Cycle

When a major version of NVIDIA Network Operator is released, the previous major version enters maintenance support and only receives patch release updates for critical bug and CVE fixes. Prior maintance release enters EOL (end-of-life) and are no longer supported and do not receive patch release updates.

The product life cycle and versioning are subject to change in the future.

Support Status for Releases

Network Operator Version	Status
25.1.x	Generally Available
24.10.x	Maintenance
24.7.x and lower	EOL

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 v24.10 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 DOCA 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-25.4.0.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.

DOCA Driver Manual Upgrade

Restarting Pods with a Containerized DOCA Driver

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

When a containerized DOCA 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 DOCA Driver to point to the new driver version. The DOCA Driver's DaemonSet will not automatically restart pods with the driver on the nodes, as it uses "OnDelete" updateStrategy. The old DOCA Driver 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 DOCA Driver 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 DOCA Driver pod.
3. Return the pods with a secondary network to the node.

When the DOCA 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 DOCA Driver Pod

Find the DOCA 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 DOCA Driver Pod from the Node

To delete the DOCA Driver pod from the node, run:

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

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

A new version of the DOCA Driver pod will automatically start.

Returning Pods with a Secondary Network to the Node

After the DOCA Driver 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 DOCA Driver Upgrade

To enable automatic DOCA Driver upgrade, define the UpgradePolicy section for the ofedDriver in the NicClusterPolicy spec, and change the DOCA Driver 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: 25.04-0.6.1.0-2
    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 DOCA Driver POD is up-to-date and running on the node, the node is schedulable.
`upgrade-required`	Set when DOCA Driver POD on the node is not up-to-date and requires upgrade. No actions are performed at this stage.
`node-maintenance-required`	Set when requestor mode upgrade is used, e.g. *MAINTENANCE_OPERATOR_ENABLED=true*, post *upgrade-required* state. Essentially it will create a matching nodeMaintenance object for dedicated node(s), utilizing maintenance operator to perform its node operations.
`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 DOCA Driver POD on the node is scheduled for restart. After the restart, the state is changed to uncordon-required.
`uncordon-required`	Set when DOCA Driver 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: 25.04-0.6.1.0-2
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      drain:
        enable: true
        force: false
        deleteEmptyDir: true
        podSelector: ""
```

Upgrade modes

DOCA Driver upgrade supports the following modes:

Mode	Description

In-place	In-place (legacy) mode is incorporates full driver upgrade lifecycle, including nodes operations e.g. cordon, pod eviction, drain, uncordon. It also maintains an internal scheduler for performing above node operations, according to provided `maxParallelUpgrades` under `UpgradePolicy` .
Requestor	New `requestor` upgrade mode uses NVIDIA maintenance operator (please refer to maintenance-operator repo) nodeMaintenance k8s API objects, to initiate the DOCA driver upgrade process. Essentially, it will retire current upgrade controller (in-place mode) from performing the following node operations: cordon, wait for pods completion, drain, uncordon. To enable requestor mode, the following environment variable should be enabled `MAINTENANCE_OPERATOR_ENABLED=true`.

Note

Enabling requestor mode will require deployment of NVIDIA maintenance operator on the cluster. By default, upgrade controller will use in-place mode. `nodeMaintenanceNamePrefix` is used to distinguish between different (operators) requestors, requesting node maintenance operations on the same node(s). Deploying maintenance operator, as well as enabling requestor mode, setting requestors env variables `MAINTENANCE_OPERATOR_REQUESTOR_ID`, `MAINTENANCE_OPERATOR_REQUESTOR_NAMESPACE`, `MAINTENANCE_OPERATOR_NODE_MAINTENANCE_PREFIX`, can be done through Network Operator helm `values.yaml`:

```
maintenanceOperator:
  enabled: true
maintenance-operator-chart:
  operatorConfig:
    maxParallelOperations: 2
    maxUnavailable: 2
operator:
  maintenanceOperator:
    useRequestor: true
    requestorID: "nvidia.network.operator"
    nodeMaintenanceNamePrefix: "network-operator"
    nodeMaintenanceNamespace: default
```

Safe Driver Loading

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

Upon node startup, the DOCA Driver container takes some time to compile and load the driver. During that time, workloads might get scheduled on that node. When DOCA Driver 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 DOCA Driver 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 DOCA Driver.

When this feature is enabled, the initial DOCA Driver 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 NVIDIA DOCA Driver pod on the node manually, by running the following command:
	`kubectl delete pod -n `kubectl get pods --A --field-selector spec.nodeName=<node name> -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 <node_name> nvidia.com/ofed-driver-upgrade.driver-wait-for-safe-load-`
	
	• Wait for the node to complete the upgrade.

The updated NVIDIA DOCA Driver pod failed to start/ a new version of NVIDIA DOCA Driver cannot be installed on the node.	Manually delete the pod by using `kubectl delete -n <Network Operator Namespace> <pod name>` . If following the restart the pod still fails, change the NVIDIA DOCA Driver 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

Delete the NicClusterPolicy:

```
kubectl delete -n nvidia-network-operator
nicclusterpolicies.mellanox.com nic-cluster-policy
```

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

- Proxy & Air-gapped
  - Network Operator Deployment in a Proxy Environment
    - Prerequisites
    - HTTP Proxy Configuration for Openshift
    - HTTP Proxy Configuration
  - Network Operator Deployment in an Air-gapped Environment
    - Local Image Registry
    - Pulling and Pushing Container Images to a Local Registry
    - Configuring Local Registry TLS Cert
    - Local Package Repository
- DOCA Driver Container
  - NVIDIA DOCA Driver Container Environment Variables
  - Precompiled Container Build Instructions for NVIDIA DOCA Driver Container
    - Prerequisites
      - Download Docker files and scripts:
    - Dockerfile Overview
    - Common mandatory build parameters
    - RHCOS
      - Prerequisites
      - Specific build parameters
      - RHCOS example
    - Ubuntu
      - Ubuntu example
    - SLES
      - Prerequisites
      - SLES example
- Other Advanced Configurations
  - Network Operator Deployment with Admission Controller
  - Network Operator Deployment with Pod Security Admission
  - Container Resources
- Container Images Digests
  - DOCA Driver Container Images
    - Ubuntu
    - RHCOS
    - RHEL
    - SLES

Proxy & Air-Gapped Environments

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 DOCA 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 or in the NVIDIA Network Operator Container Images section.

NicClusterPolicy supports use of image container digest in the *version* field, except for DOCA driver.

There is one caveat with regards to the DOCA driver image. The version field must be appended by the OS name and Architecture running on the worker node.

For example for DOCA driver version 25.04-0.6.1.0-2, the tag for Ubuntu 24.04 with X86 architecture is “25.04-0.6.1.0-2-ubuntu24.04-amd64”. Available DOCA driver image tags can be found at NGC.

In case of local registry required authentication, make sure to create a pull secret and configure in NicClusterPolicy accordingly.

Note

NVIDIA Network Operator communicates with the Image Registry configured for the DOCA Driver in the NICClusterPolicy to list the available tags. Specifying pull secret is required in the NicClusterPolicy DOCA Driver section, even if global container access credentials are configured on nodes.

Configuring Local Registry TLS Cert

NVIDIA Network Operator communicates with the Image Registry configured for the DOCA Driver in the NICClusterPolicy to list the available tags. This is required to verify the availability of precompiled DOCA Driver container images.

If the Image Registry uses a TLS certificate that is not issued by a well-known Certificate Authority (CA), it is required to configure the NVIDIA Network Operator with the Certificate.

Fetch the TLS Certificates from the Image Registry server and save them in the *ca.crt* file:

```
export REGISTRY_HOST=myregistry.example.com
export REGISTRY_PORT=5000
openssl s_client -showcerts -connect
${REGISTRY_HOST}:${REGISTRY_PORT} </dev/null 2>/dev/null | awk
'/-----BEGIN CERTIFICATE-----/,/-----END CERTIFICATE-----/' >
ca.crt
```

Create a ConfigMap containing the TLS Certificates from the *ca.crt file*:

```
kubectl create configmap custom-registry-cert --from-file=
<external_registry>=ca.crt -n nvidia-network-operator
```

In OpenShift, update the *nvidia-network-operator* Subscription by adding the following *config* under *spec*:

```
kubectl apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
name: nvidia-network-operator
namespace: nvidia-network-operator
spec:
config:
volumes:
- configMap:
name: custom-registry-cert
name: custom-registry-ca
volumeMounts:
- mountPath: /etc/pki/tls/certs/
name: custom-registry-ca
EOF
```

In Kubernetes, update the *network-operator* Deployment's volumes by running:

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
name: network-operator
namespace: nvidia-network-operator
spec:
template:
spec:
volumes:
- configMap:
name: custom-registry-cert
name: custom-registry-ca
containers:
- name: network-operator
volumeMounts:
- mountPath: /etc/pki/tls/certs/
name: custom-registry-ca
EOF
```

Local Package Repository

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

The DOCA 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
```

NVIDIA DOCA Driver Container

NVIDIA DOCA Driver Container Environment Variables

The following are special environment variables supported by the NVIDIA DOCA Driver 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 NVIDIA DOCA Driver modules: * ib_isert * nvme_rdma * nvmet_rdma * rpcrdma * xprtrdma * ib_srpt
ENABLE_NFSRDMA	“false”	Enable loading of NFS & NVME related storage modules from a NVIDIA DOCA Driver 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 NVIDIA DOCA Driver 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).

> **⚠ Warning**
>
> When ENABLE_NFSRDMA is set to *true*, it is not possible to load NVME related storage modules from NVIDIA DOCA Driver container when they are in use by the system (e.g the system has NVMe SSD drives in use). User should ensure the modules are not in use and blacklist them prior to the use of NVIDIA DOCA Driver container.

These variables can be set in the NicClusterPolicy. For example:

```
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    env:
    - name: RESTORE_DRIVER_ON_POD_TERMINATION
      value: "true"
    - name: UNLOAD_STORAGE_MODULES
      value: "true"
    - name: CREATE_IFNAMES_UDEV
      value: "true"
```

Precompiled Container Build Instructions for NVIDIA DOCA Driver Container

Prerequisites

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

- Docker (Ubuntu) / Podman (RH) installed on your build system.
- Web access to NVIDIA NIC drivers sources. Latest NIC drivers are published at NVIDIA DOCA Downloads, for example: https://linux.mellanox.com/public/repo/doca/2.10.0/SOURCES/MLNX_OFED/MLNX_OF debian-25.01-0.6.0.0.tgz

NOTE: NVIDIA NIC driver sources are bundled as part of NVIDIA DOCA package. Both the DOCA package version and its corresponding NIC driver (DOCA Driver) version need to be specified to fetch the correct driver sources when building the driver container. For example, given a DOCA package version (e.g *2.10.0*) you can find the corresponding MLNX_OFED version at the link: https://linux.mellanox.com/public/repo/doca/2.10.0/SOURCES/MLNX_OFED/ which is *25.01-0.6.0.0'*

Download Docker files and scripts:

```
wget https://raw.githubusercontent.com/Mellanox/doca-driver-
build/f5de72596d639bc369566676038ac251c9575ca3/RHEL_Dockerfile
wget https://raw.githubusercontent.com/Mellanox/doca-driver-
build/f5de72596d639bc369566676038ac251c9575ca3/Ubuntu_Dockerfile
wget https://raw.githubusercontent.com/Mellanox/doca-driver-
build/f5de72596d639bc369566676038ac251c9575ca3/SLES_Dockerfile
wget https://raw.githubusercontent.com/Mellanox/doca-driver-
build/f5de72596d639bc369566676038ac251c9575ca3/entrypoint.sh
wget https://raw.githubusercontent.com/Mellanox/doca-driver-
build/f5de72596d639bc369566676038ac251c9575ca3/dtk_nic_driver_build
chmod +x entrypoint.sh dtk_nic_driver_build.sh
```

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 NVIDIA DOCA 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 (e.g., ubuntu:22.04)

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_DOCA_VERSION*: NVIDIA DOCA version (e.g., 2.9.1)

6. *D_OFED_VERSION*: NVIDIA NIC drivers version (e.g., 24.10-1.1.4.0)

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

NOTE: For proper Network Operator functionality container tag name must be in following pattern: **driver_ver-container_ver-kernel_ver-os-arch**. For example: 24.10-1.1.4.0-0-5.15.0-25-generic-ubuntu22.04-amd64

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

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

RHCOS

Prerequisites

- Install oc CLI tool.
- Download OpenShift pull secret.

Specific build parameters

1. *D_BASE_IMAGE*: DriverToolKit container image

   NOTE: DTK (DriverToolKit) is tightly coupled with specific kernel version for an OpenShift release. In order to get the specific DTK container image for a specific OpenShift release, run:

```
oc adm release info <OCP_VERSION> --image-for=driver-toolkit
```

For example, for OpenShift 4.16.0:

```
oc adm release info 4.16.0 --image-for=driver-toolkit
quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:dde3cd6a75d865a476aa7e1cab6fa8d97742401e87e0d514f3042c3
```

Then pull the DTK image locally using your pull-secret:

```
podman pull --authfile=/path/to/pull-secret.txt
docker://quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:dde3cd6a75d865a476aa7e1cab6fa8d97742401e87e0d514f3042c3
```

2. *D_FINAL_BASE_IMAGE*: Final container image, to install compiled driver
3. *D_ARCH*: Target architecture: *x86_64* or *aarch64*.
4. *D_KERNEL_VER*: CoreOS kernel versions for OpenShift are listed here.

Kernel version can also be found with the DTK image using the following command:

```
podman run --rm -ti quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:47ba8a10d5938c41f907ee7f70d74aecb9c2dfd7afae4cea2942fc8
cat /etc/driver-toolkit-release.json | jq -r '.KERNEL_VERSION'
5.14.0-427.22.1.el9_4.x86_64
```

RHCOS example

```
podman build \
 --build-arg D_OS=rhcos4.16 \
 --build-arg D_ARCH=x86_64 \
 --build-arg D_KERNEL_VER=5.14.0-427.22.1.el9_4.x86_64 \
 --build-arg D_DOCA_VERSION=2.9.1 \
 --build-arg D_OFED_VERSION=24.10-1.1.4.0 \
 --build-arg D_BASE_IMAGE="quay.io/openshift-release-dev/ocp-
v4.0-art-
dev@sha256:dde3cd6a75d865a476aa7e1cab6fa8d97742401e87e0d514f3042c3
\
 --build-arg
D_FINAL_BASE_IMAGE=registry.access.redhat.com/ubi9/ubi:9.4 \
 --tag 24.10-1.1.4.0-0-5.14.0-427.22.1.el9_4.x86_64-rhcos4.16-
amd64 \
 -f RHEL_Dockerfile \
 --target precompiled .
```

Ubuntu

Ubuntu example

```
docker build \
 --build-arg D_OS=ubuntu22.04 \
 --build-arg D_ARCH=x86_64 \
 --build-arg D_BASE_IMAGE=ubuntu:22.04 \
 --build-arg D_KERNEL_VER=5.15.0-25-generic \
 --build-arg D_DOCA_VERSION=2.9.1 \
 --build-arg D_OFED_VERSION=24.10-1.1.4.0 \
 --tag 24.10-1.1.4.0-0-5.15.0-25-generic-ubuntu22.04-amd64 \
 -f Ubuntu_Dockerfile \
 --target precompiled .
```

SLES

Prerequisites

Active subscription. After registering, make sure to run *zypper refresh && zypper update -y*.

SLES example

```
docker build \
 --build-arg D_OS=sles15.5 \
 --build-arg D_ARCH=x86_64 \
 --build-arg D_BASE_IMAGE=registry.suse.com/suse/sle15:15.5 \
 --build-arg D_KERNEL_VER=5.14.21-150500.55.83-default \
 --build-arg D_DOCA_VERSION=2.9.1 \
 --build-arg D_OFED_VERSION=24.10-1.1.4.0 \
 --tag 24.10-1.1.4.0-0-5.14.21-150500.55.83-default-sles15.5-
amd64 \
 -f SLES_Dockerfile \
 --target precompiled .
```

Advanced Configurations

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 `<Release_Name>-webhook-service.<Release_Namespace>.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)
```

Container Resources

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

For example, for the SR-IOV Device Plugin:

```
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  sriovDevicePlugin:
    containerResources:
    - name: "sriov-device-plugin"
      requests:
        cpu: "200m"
        memory: "150Mi"
      limits:
        cpu: "300m"
        memory: "300Mi"
```

NVIDIA Network Operator Container Images

Repository	Image Name	Tag	
nvcr.io/nvidia/cloud-native	network-operator	v25.4.0	sha256:93c2063f4da6a93
ghcr.io/mellanox	network-operator-init-container	v0.0.3	sha256:67e93ccf3ecb61f1
ghcr.io/mellanox	k8s-rdma-shared-dev-plugin	v1.5.3	sha256:1eb5ab7f4b6f0cf8
ghcr.io/mellanox	ib-kubernetes	v1.1.1	sha256:9774f774df6ede9c
ghcr.io/mellanox	ipoib-cni	v1.2.2	sha256:513d88d639a01ef
ghcr.io/mellanox	nvidia-k8s-ipam	v0.3.7	sha256:1b20b78f8893398

ghcr.io/mellanox	nic-feature-discovery	v0.0.2	sha256:35b1820fe94a833a
nvcr.io/nvidia/doca	doca_telemetry	1.16.5-doca2.6.0-host	sha256:866971717cb2683
nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-25.4.0	sha256:fd4ef7f4a5342518
nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-25.4.0	sha256:cd350ea326bf311(
nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-25.4.0	sha256:747c2de5521389b
ghcr.io/k8snetworkplumbingwg	sriov-network-device-plugin	v3.9.0	sha256:cabce074d10a0f1c
ghcr.io/k8snetworkplumbingwg	sriov-cni	v2.8.1	sha256:749822bef050882a
ghcr.io/k8snetworkplumbingwg	ib-sriov-cni	v1.2.1	sha256:2e2c1f86a401a26(
ghcr.io/k8snetworkplumbingwg	plugins	v1.6.2-update.1	sha256:fd99966bdb5e7e7c
ghcr.io/k8snetworkplumbingwg	multus-cni	v4.1.0	sha256:aa59e65256324c83
ghcr.io/k8snetworkplumbingwg	whereabouts	v0.7.0	sha256:2d2f73ee5af0e7fd1
ghcr.io/k8snetworkplumbingwg	ovs-cni-plugin	v0.38.2	sha256:64baa43b7149add5
ghcr.io/k8snetworkplumbingwg	rdma-cni	v1.3.0	sha256:2500b2795e9afb8e
ghcr.io/mellanox	nic-configuration-operator	v1.0.3	sha256:353ab27f5f5fba08
ghcr.io/mellanox	nic-configuration-operator-daemon	v1.0.3	sha256:0e8478898e0b30a(
ghcr.io/mellanox	maintenance-operator	v0.2.2	sha256:951ff8b27a21109c
ghcr.io/mellanox	spectrum-x-operator	v0.0.3	sha256:d6a2546a8a65e103

DOCA Driver Container Images

Repository	Image Name	Version
nvcr.io/nvidia/mellanox	doca-driver	25.04-0.6.1.0-2

The followings tags are available for the above DOCA Driver container version:

Ubuntu

Tags	Digest
25.04-0.6.1.0-2-ubuntu22.04	sha256:b3561383f61423fc5e312fb0bfdd30345cbe32e8c404482c329ca
25.04-0.6.1.0-2-ubuntu22.04-amd64	sha256:cb436254179a539a33c31b97c3eb84f78011e5eba3b7a4978024
25.04-0.6.1.0-2-ubuntu22.04-arm64	sha256:b434c8d20f9fee8c590bb34375da751d25a0d9e249b9f67640264
25.04-0.6.1.0-2-ubuntu24.04	sha256:86122f68639756d7565b0f12368ffe872fa487d09aff874a736c10
25.04-0.6.1.0-2-ubuntu24.04-amd64	sha256:c9d12598ab699d29a04d6fadd89fdf0b2515dd5947f414baa704b
25.04-0.6.1.0-2-ubuntu24.04-arm64	sha256:97c311b6cf9b973b65767b5520cb0a76077837599ecd1b8581c7

25.04-0.6.1.0-21-ubuntu24.04-amd64	sha256:188eabded1db4f60051c58df8b162f39b66d96197790dc614ff0f
25.04-0.6.1.0-22-ubuntu24.04-amd64	sha256:607ae32ada98fc16c3d523829f81cdd99b27a759ba340133b735

RHCOS

Tags	Digest
25.04-0.6.1.0-2-rhcos4.13-amd64 25.04-0.6.1.0-2-rhcos4.14-amd64 25.04-0.6.1.0-2-rhcos4.15-amd64 25.04-0.6.1.0-2-rhcos4.16-amd64 25.04-0.6.1.0-2-rhcos4.17-amd64 25.04-0.6.1.0-2-rhcos4.18-amd64	sha256:69e6ab6a0249be918f60c1e04fc53a07f280b33563f8edc953806538

25.04-0.6.1.0-2-rhcos4.13-arm64 25.04-0.6.1.0-2-rhcos4.14-arm64 25.04-0.6.1.0-2-rhcos4.15-arm64 25.04-0.6.1.0-2-rhcos4.16-arm64 25.04-0.6.1.0-2-rhcos4.17-arm64 25.04-0.6.1.0-2-rhcos4.18-arm64	sha256:9bc09051aca763f83cde99ee8909cf989b1d6c5d6ab683fbb1ca441c
25.04-0.6.1.0-2-rhcos4.18	sha256:6bee03b26e7abf7e61b69522ece35e35648c6f4d52a89519bca2407
25.04-0.6.1.0-200-rhcos4.18-amd64	sha256:10a29c282d159a66e972bd2be12a8dce03c3b891deed07ad2b97c9e
25.04-0.6.1.0-210-rhcos4.16-amd64	sha256:710c72cca18b3625637d4828a612cc16fd3c6a7b37c4cd7c8c78a1bf

25.04-0.6.1.0-210-rhcos4.18-amd64	sha256:8b6992a0a96fd0a549e31553b449c6683de9d8c681cf9a4414e6fa2f
25.04-0.6.1.0-211-rhcos4.16-amd64	sha256:9178c2cb416396e22b6707089c3670aec32bec60221c4cba1db8e5f

RHEL

Tags	Digest
25.04-0.6.1.0-2-rhel8.10	sha256:9ffc93fa12782ea7bf064d2d3a199072510ceab5d0af18d34ec95b198(
25.04-0.6.1.0-2-rhel8.10-amd64 25.04-0.6.1.0-2-rhel8.6-amd64 25.04-0.6.1.0-2-rhel8.8-amd64 25.04-0.6.1.0-2-rhel8.9-amd64	sha256:35ff3b173b2694198e57b61720545be67e944c1da18ac004007a5b4e(

25.04-0.6.1.0-2-rhel8.10-arm64 25.04-0.6.1.0-2-rhel8.6-arm64 25.04-0.6.1.0-2-rhel8.8-arm64 25.04-0.6.1.0-2-rhel8.9-arm64	sha256:76529c24ea17e68b89a2c0d4823d66504fa6f53759760a449308fda39

25.04-0.6.1.0-2-rhel9.0-amd64 25.04-0.6.1.0-2-rhel9.2-amd64 25.04-0.6.1.0-2-rhel9.3-amd64 25.04-0.6.1.0-2-rhel9.4-amd64 25.04-0.6.1.0-2-rhel9.5-amd64	sha256:4a620bd39bc70928cd6e249855176ced0f4feb585c8c7624c0b9f5a3e

25.04-0.6.1.0-2-rhel9.0-arm64 25.04-0.6.1.0-2-rhel9.2-arm64 25.04-0.6.1.0-2-rhel9.3-arm64 25.04-0.6.1.0-2-rhel9.4-arm64 25.04-0.6.1.0-2-rhel9.5-arm64	sha256:56a41010b4dbee0dc31e8f9c1467a739241a2d248ed8b5054d4b418f
25.04-0.6.1.0-2-rhel9.5	sha256:861eb503c13cf4246aacefdd5deaa978eddce1cc6bf1d83beb1a7b38a4

SLES

Tags	Digest
25.04-0.6.1.0-2-sles15.6	sha256:18d789efefea306f42a87f08df8cc53075f274abb3ddd5d10da2974eaa

25.04-0.6.1.0-2-sles15.6-amd64	sha256:ea2b07942daafa8c08cacfcf8ba64f783a39fd6f8644458952fd582325
25.04-0.6.1.0-2-sles15.6-arm64	sha256:e3389a26293b3f2fa291b87c66f97819325359c61055fa3e360227bcd

Legal Notices and 3rd Party Licenses

Component	Version	Legal Notices and 3rd Party Licenses
NVIDIA Network Operator	v25.4.0	- License - 3rd Part Notice
NVIDIA Network Operator Init Container	v0.0.3	- License - 3rd Part Notice
RDMA Shared Device Plugin	v1.5.3	- License - 3rd Part Notice
IB Kubernetes Plugin	v1.1.1	- License - 3rd Part Notice
IP Over Infiniband (IPoIB) CNI plugin	v1.2.2	- License - 3rd Part Notice
NVIDIA IPAM Plugin	v0.3.7	- License - 3rd Part Notice
NVIDIA NIC Feature Discovery	v0.0.2	- License - 3rd Part Notice
Node Feature Discovery	v0.15.6	- License - 3rd Part Notice
SRIOV Network Operator	network-operator-25.4.0	- License - 3rd Part Notice

SR-IOV Network Device Plugin	v3.9.0	- License - 3rd Part Notice
SR-IOV CNI plugin	v2.8.1	- License - 3rd Part Notice
InfiniBand SR-IOV CNI plugin	v1.2.1	- License - 3rd Part Notice
Multus CNI	v4.1.0	- License - 3rd Part Notice
Whereabouts IPAM Plugin	v0.7.0	- License - 3rd Part Notice
RDMA CNI plugin	v1.3.0	- License - 3rd Part Notice
Open vSwitch CNI plugin	v0.38.2	- License - 3rd Part Notice
NVIDIA NIC Configuration Operator	v1.0.3	- License - 3rd Part Notice
NVIDIA Maintenance Operator	v0.2.2	- License - 3rd Part Notice
NVIDIA SpectrumX Operator	v0.0.3	- License - 3rd Part Notice

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF

ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2025, NVIDIA.. PDF Generated on 10/14/2026