NVIDIA Network Operator v26.1.1

Table of contents

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.

Networking Features

RDMA Support

Remote Direct Memory Access (RDMA) for memory-to-memory data transfers that bypass the CPU and kernel networking stack. Supports InfiniBand and RDMA over Converged Ethernet (RoCE) protocols.

SR-IOV Virtualization

Single Root I/O Virtualization technology that partitions network interface cards into multiple Virtual Functions (VFs) for hardware-level isolation and performance.

Secondary Networks

Multiple network interface types including host device networks, MacVLAN networks, IP over InfiniBand networks, and SR-IOV networks for specialized networking requirements.

Driver Management

Automated deployment and management of NVIDIA DOCA-OFED networking drivers across cluster nodes with version control and updates.

Supported Hardware

See the Platform Support page for supported hardware and software.

Use Cases

High-Performance Computing (HPC): Scientific simulations, modeling applications, and distributed computing workloads

Machine Learning: Distributed training and inference workloads across multiple GPU nodes

Data Processing: Database systems, analytics platforms, and storage applications requiring high network throughput

Legacy Applications: Existing applications that require direct access to networking hardware

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.

For detailed deployment instructions and examples:

- Quick Start Guide for Kubernetes: Quick deployment guide with common configurations
- NVIDIA Network Operator Deployment Guide with Kubernetes: Detailed deployment scenarios

Release Notes

Changes and New Features

Version	Description
26.1.1	- CVEs fixes

26.1.0	Added support for ConnectX-9 SuperNIC - Added support for Spectrum-X Ethernet Networking Platform with Kubernetes [GA] - Added support for Spectrum-X Single-plane and SW Multi-plane topologies in Spectrum-X Operator and NIC Configuration Operator - Added Spectrum-X support for NVIDIA Kubernetes Launch Kit - Added support for Azure AKS with Azure Linux - Added support for Government-Ready, STIG/FIPS hardening security requirements, in OpenShift deployments for NVIDIA AI Enterprise customers - Added support for Network Operator SOS report script, to collect system information for troubleshooting purposes - Added support for NIC multiport firmware parameter (esw_multiport) using devlinkParams in SR-IOV Network Operator - Added support for setting OVS bridges grouping configuration in SR-IOV Network Operator, grouping per NIC PFs to a single OVS bridge - [TECH PREVIEW] Added support for NicInterfaceNameTemplate allowing users to define custom naming schemes for network and RDMA interfaces on NIC devices in NIC Configuration Operator

25.10.0	- Added support for OpenShift v4.20 - Added support for RHEL v10.0 - Added support for SUSE Linux Enterprise Server 15 SP7 - Added support for Government-Ready, STIG/FIPS hardening security requirements, for NVIDIA AI Enterprise customers - Added support for Kubernetes node draining operation in Network Operator, to apply SR-IOV related configuration on host, by utilizing NVIDIA Maintenance Operator - Added support for enabling secured mode in created OVS bridges - Added support for OpenShift air-gapped environments - Added support for non-volatile parameters in NIC Configuration Operator - [TECH PREVIEW] Added NVIDIA Kubernetes Launch Kit - a CLI tool for deploying and managing NVIDIA cloud-native solutions on Kubernetes. The tool helps provide flexible deployment workflows for optimal network performance with SR-IOV, RDMA, and other networking technologies - Removed support for Whereabouts IPAM in Network Operator (Removed from NIC Cluster Policy API)
25.7.0	- Added support for OpenShift v4.19 - Added support for RHEL v9.6 - Added support for optimized DOCA driver container handling on pod termination, reducing cleanup time and avoiding unnecessary reloads - Added support for NIC Configuration Operator on Red Hat OpenShift - Added support for BlueField Firmware Bundle upgrades for BlueField-3 SuperNICs with NIC Configuration Operator

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 Driver 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-OFED 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-OFED 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-OFED 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
25.10.0	- Whereabouts IPAM plugin support removed - NIC Configuration Operator Helm chart support removed
25.7.0	- MOFED driver container older than 5.7-0.1.2.0 is not supported
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
26.1.1	- There is a known limitation in SR-IOV OVSNetwork, opposed of using SriovNetwork or SriovIBNetwork. When using OVSNetwork, setting VF link state to follow its PF link state is not supported

26.1.0	- There is a known limitation in OVS-CNI after applying SR-IOV Operator OVSNetwork and SriovNetworkNodePolicy with OVS bridge and switchdev mode configuration. The limitation is that after k8s node is rebooted, the VF representor that was attached to the OVS bridge will not be deleted. The workaround is to (manually/CD) delete all stale VF representors after the node is rebooted - Multus CNI does not use a service account token after the Kubernetes API rotates it. The recommended workaround is to edit the Multus CNI DaemonSet manually to remove `--skip-config-watch` CLI argument. - The DOCA driver container is validated with host kernel versions up to 6.8.0-100. Compatibility with newer kernel versions is not guaranteed and may require a newer DOCA driver container release
25.10.0	- NVIDIA Networking NIC Configuration Operator doesn't support Socket Direct Adapters
25.7.0	- NVIDIA DOCA-OFED Driver container can not start after kernel upgrade on OCP. To make Driver container work, please remove */var/opt/mofed-container/inventory* directory on the host - DOCA driver container deployment may fail if NVIDIA drivers are in use by third-party kernel modules or user-space applications. The recommended workaround is to use non-containerized DOCA drivers deployed via the DOCA-Host package - SR-IOV Network Operator doesn't support SR-IOV SystemD configuration mode on OCP

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 - In infiniband mode, in case of existing Intel NICs, loaded *irdma* module should be unloaded before deploying DOCA-OFED driver
24.10.0	- There is a known limitation when using NVIDIA NICs as **primary network interfaces**. If the NVIDIA DOCA-OFED 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-OFED 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-OFED Driver container and NVMe modules are loaded in the host system, NVIDA DOCA-OFED 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	>=1.31 and <=1.35	
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 ConnectX-9 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) / Red Hat OpenShift	GA
NVIDIA DGX/HGX B200	x86	NVIDIA Blackwell	BlueField-3 SuperNIC (NIC mode) / ConnectX-7	Ubuntu 22.04 / 24.04 (x86) / Red Hat OpenShift	GA
NVIDIA RTX PRO 6000 Blackwell Server	x86	NVIDIA Blackwell	BlueField-3 SuperNIC (NIC mode) / ConnectX-8	Ubuntu 22.04 / 24.04 (x86) / Red Hat OpenShift	GA
NVIDIA DGX/HGX B300	x86	NVIDIA Blackwell	ConnectX-8 SuperNIC	Ubuntu 22.04 / 24.04 (x86) / Red Hat OpenShift	GA

Supported Operating Systems and Kubernetes Platforms

NVIDIA Network Operator has been validated on the following OS and platform combinations for Ethernet (RoCE) and InfiniBand (IB RDMA):

Note

Kubernetes support for the NVIDIA Spectrum-X Reference Architecture (RA) is limited to a subset of OS and platform combinations. For details, refer to the NVIDIA Spectrum-X RA documentation.

Operating Systems	Upstream Kubernetes	Red Hat OpenShift	Ranch
Ubuntu 24.04 LTS	1.31–1.35	—	—
Ubuntu 22.04 LTS	1.31–1.35	—	—
Red Hat CoreOS	—	4.17–4.20	—
Red Hat Enterprise Linux 10.0 / 9.6 / 9.4	1.31–1.35	—	—
Red Hat Enterprise Linux 8.10 / 8.8	1.31–1.35	—	—
SUSE Linux Enterprise Server 15 SP7	1.31–1.35	—	1.31–1

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	

Red Hat Enterprise Linux 10	Yes	Yes	
SUSE Linux Enterprise Server 15 SP7	Yes	No	

NVIDIA Spectrum-X Ethernet Networking Platform

NVIDIA Network Operator has been validated with the following NVIDIA Spectrum-X Reference Architecture (RA) versions:

> **Note**
>
> For details on supported topologies, NIC hardware, software components, and version-specific notes, refer to the NVIDIA Spectrum-X RA documentation.

Spectrum-X RA Version	Topologies and NIC Hardware	Kubernetes Versions	Operating Systems	Notes
2.1	• Single-Plane (ConnectX-7 or BlueField-3 SuperNIC) • Dual- or Quad-plane (NVIDIA ConnectX-8 SuperNIC)	Upstream Kubernetes (1.31–1.35)	• Ubuntu 24.04 LTS • Ubuntu 22.04 LTS	Small RA, Software Multi-Plane

Supported Precompiled Container Images for DOCA-OFED Drivers

Overview

To save startup time and operational effort, precompiled DOCA-OFED 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-OFED 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-OFED 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 the Precompiled Container Build Instructions for NVIDIA DOCA-OFED 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	NVAIE	
NVIDIA Network Operator	NVIDIA (OSS)	nvcr.io/nvidia/cloud-native	network-operator	v26.1.1	Yes	
NVIDIA Network Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	network-operator-init-container	network-operator-v26.1.1	Yes	
DOCA-OFED Driver Container	NVIDIA (EULA)	nvcr.io/nvidia/mellanox	doca-driver	doca3.3.0-26.01-1.0.0.0-0	Yes	L d 2 4
RDMA Shared Device Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	k8s-rdma-shared-dev-plugin	network-operator-v26.1.1	Yes	
IB Kubernetes Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	ib-kubernetes	network-operator-v26.1.1	Yes	
IP Over Infiniband (IPoIB) CNI plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	ipoib-cni	network-operator-v26.1.1	Yes	
NVIDIA IPAM Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nvidia-k8s-ipam	network-operator-v26.1.1	Yes	
NVIDIA NIC Feature Discovery	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-feature-discovery	network-operator-v26.1.1	Yes	
DOCA Telemetry Service (DTS)	NVIDIA (EULA)	nvcr.io/nvidia/doca	doca_telemetry	1.23.4-doca3.2.0-host	Yes	

Node Feature Discovery	Community (OSS)	nvcr.io/nvidia/mellanox	node-feature-discovery	network-operator-v26.1.1	Yes	O d M b th w c
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-v26.1.1	Yes	
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-v26.1.1	Yes	
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-v26.1.1	Yes	
SR-IOV Network Device Plugin	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-device-plugin	network-operator-v26.1.1	Yes	
SR-IOV CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-cni	network-operator-v26.1.1	Yes	
InfiniBand SR-IOV CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	ib-sriov-cni	network-operator-v26.1.1	Yes	
K8s CNI network plugins	Community (OSS)	nvcr.io/nvidia/mellanox	plugins	network-operator-v26.1.1	Yes	
Multus CNI	Community (OSS)	nvcr.io/nvidia/mellanox	multus-cni	network-operator-v26.1.1	Yes	
RDMA CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	rdma-cni	network-operator-v26.1.1	Yes	
Open vSwitch CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	ovs-cni-plugin	network-operator-v26.1.1	No	

NVIDIA NIC Configuration Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-configuration-operator	network-operator-v26.1.1	No	
NVIDIA NIC Configuration Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon	network-operator-v26.1.1	No	
NVIDIA Maintenance Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	maintenance-operator	network-operator-v26.1.1	No	
NVIDIA Kubernetes Launch Kit	NVIDIA (OSS)	nvcr.io/nvidia/cloud-native	k8s-launch-kit	v26.1.0	No	

Additional Supported Tools and Integrations

Container management tools:

- Helm v3
- Red Hat Operator Lifecycle Manager (OLM)

Orchestration & resource scheduling:

- Run:ai

Note

Run:ai requires the NVIDIA Network Operator as a prerequisite. To configure NVIDIA Network Operator refer to the Run:ai cluster requirements documentation for more information.

Getting Started with Kubernetes

This section provides comprehensive guides to help you get started with the NVIDIA Network Operator on Kubernetes.

- Quick Start Guide
- Deployment Guide with Kubernetes

Quick Start Guide for Kubernetes

Before You Begin

Before deploying the NVIDIA Network Operator, ensure you have the following:

Prerequisites

1. **Kubernetes Cluster**: A running Kubernetes cluster (v1.19+) with nodes that have NVIDIA NICs.

2. **CLI Tools**: Install `kubectl` and `helm` on your client machine:

```
$ 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
```

3. **Container Runtime**: Nodes must be configured with a container engine such as CRI-O or containerd.

Install Network Operator Helm Chart

Add the NVIDIA NGC Helm repository:

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

Install the Network Operator:

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version v26.1.1 \
  --set sriovNetworkOperator.enabled=true \
  --wait
```

Verify the installation:

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

Overview of Quickstart Use Cases

This quick start guide covers five essential networking configurations for different computational requirements:

Use Case	Purpose	Performance Requirements	Applications

SR-IOV Network with RDMA	High-performance networking with hardware acceleration	• >10 Gbps throughput • <1µs latency • Dedicated VF resources	HPC simulations, distributed ML training, financial trading *Keywords: SR-IOV, RDMA, HPC, low-latency, VF isolation*
Host Device Network with RDMA	Direct hardware access for legacy applications	• Raw device control • Exclusive hardware access • Minimal CPU overhead	Legacy HPC codes, specialized protocols, DPDK applications *Keywords: host-device, PCI-passthrough, direct-access, exclusive-access*
IP over InfiniBand with RDMA Shared Device	InfiniBand networking with shared RDMA resources	• >50 Gbps bandwidth • Parallel I/O workloads • Shared device efficiency	Distributed storage, data analytics, scientific computing *Keywords: InfiniBand, IPoIB, shared-device, high-bandwidth*

MacVLAN Network with RDMA Shared Device	Network isolation with shared RDMA capabilities	• Multi-tenant segmentation • 10+ pods per node • Moderate throughput	Cloud-native HPC, microservices, multi-tenant ML *Keywords: MacVLAN, multi-tenant, network-segmentation, resource-sharing*
SR-IOV InfiniBand Network with RDMA	Virtualized InfiniBand with hardware acceleration	• >100 Gbps bandwidth • Hardware acceleration • Isolated IB partitions	Large-scale HPC clusters, AI/ML training, research computing *Keywords: SR-IOV, InfiniBand, hardware-acceleration, ultra-high-bandwidth*

NVIDIA Network Operator Deployment Guide with Kubernetes

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 v26.1.1 \
  --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 v26.1.1 >
values.yaml
```

Install with specifying the custom *values.yaml*

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version v26.1.1 \
  -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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
```

```
    version: network-operator-v26.1.1
    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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    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 and 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: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    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-OFED Driver, the following packages should be installed:
>
> - the linux-generic package on Ubuntu hosts
> - the kernel-modules-extra package on the RedHat-based hosts

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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    config: |
      {
        "resourceList": [
          {
            "resourcePrefix": "nvidia.com",
            "resourceName": "hostdev",
            "selectors": {
              "vendors": ["15b3"],
              "devices": [],
              "drivers": [],
              "pfNames": [],
              "pciAddresses": [],
              "rootDevices": [],
              "linkTypes": [],
              "isRdma": true
            }
          }
        ]
      }
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
```

```
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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": "nv-ipam",
      "poolName": "my-pool"
    }
```

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": "nv-ipam",
      "poolName": "myPool"
    }
```

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
- NVIDIA IPAM 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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
```

```
  config: |
    {
      "resourceList": [
        {
          "resourcePrefix": "nvidia.com",
          "resourceName": "hostdev",
          "selectors": {
            "vendors": [],
            "devices": [],
            "drivers": [],
            "pfNames": [],
            "pciAddresses": [],
            "rootDevices": [],
            "linkTypes": ["IB"],
            "isRdma": true
          }
        }
      ]
    }
nvIpam:
  image: nvidia-k8s-ipam
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
  imagePullSecrets: []
  enableWebhook: false
secondaryNetwork:
  cniPlugins:
    image: plugins
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    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-OFED driver
- RDMA shared device plugin
- Secondary network
- Multus CNI
- IPoIB CNI
- NVIDIA IPAM Plugin

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
```

```
    version: network-operator-v26.1.1
    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": []
            }
          }
        ]
      }
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
```

```
      version: network-operator-v26.1.1
      imagePullSecrets: []
    ipoib:
      image: ipoib-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
```

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": "nv-ipam",
      "poolName": "my-pool"
    }
```

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": "nv-ipam",
      "poolName": "my-pool"
    }
```

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-OFED 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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
```

```
  imagePullSecrets: []
  config: |
    {
      "resourceList": [
        {
          "resourcePrefix": "nvidia.com",
          "resourceName": "hostdev",
          "selectors": {
            "vendors": ["15b3"],
            "devices": [],
            "drivers": [],
            "pfNames": [],
            "pciAddresses": [],
            "rootDevices": [],
            "linkTypes": [],
            "isRdma": true
          }
        }
      ]
    }
nvIpam:
  image: nvidia-k8s-ipam
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
  imagePullSecrets: []
  enableWebhook: false
secondaryNetwork:
  cniPlugins:
    image: plugins
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
```

```
      imagePullSecrets: []
```

`host-device-net.yaml:`

```
apiVersion: mellanox.com/v1alpha1
kind: HostDeviceNetwork
metadata:
   name: hostdevice-net
spec:
  networkNamespace: "default"
  resourceName: "hostdev"
  ipam: |
    {
      "type": "nv-ipam",
      "poolName": "my-pool"
    }
```

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": "nv-ipam",
      "poolName": "my-pool"
    }
```

`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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
```

```
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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: |-
    {
      "type": "nv-ipam",
      "poolName": "my-pool"
    }
```

⚠ 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
nv-ipam-controller-865f479547-6dkkg                 1/1     Running
0          5d
nv-ipam-controller-865f479547-cbp4p                 1/1     Running
0          5d
nv-ipam-node-4ghkj                                  1/1     Running
0          5d
nv-ipam-node-p5kff                                  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-OFED Driver and OpenSM running. OpenSM runs on top of the NVIDIA DOCA-OFED 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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
```

```
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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": "nv-ipam",
      "poolName": "my-pool"
    }
  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-OFED Driver and OpenSM running. OpenSM runs on top of the NVIDIA DOCA-OFED 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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
```

```
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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-OFED 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-sriov-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-OFED 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: doca3.3.0-26.01-1.0.0.0-0
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
     config: |
      {
        "resourceList": [
            {
                "resourcePrefix": "nvidia.com",
                "resourceName": "rdma_host_dev",
                "selectors": {
                    "vendors": ["15b3"],
                    "devices": ["1018"],
                    "drivers": ["mlx5_core"]
                }
            }
        ]
      }
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
```

```
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
```

`host-device-net.yaml`

```
apiVersion: mellanox.com/v1alpha1
kind: HostDeviceNetwork
metadata:
  name: example-hostdev-net
spec:
  networkNamespace: "default"
  resourceName: "rdma_host_dev"
  ipam: |
    {
      "type": "nv-ipam",
      "poolName": "my-pool"
    }
```

`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-OFED 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.

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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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-OFED Driver container.

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

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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
      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:
    feature.node.kubernetes.io/pci-15b3.present: "true"
  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
```

Getting Started with OpenShift

This section provides comprehensive guides to help you get started with the NVIDIA Network Operator on Red Hat OpenShift.

- Deployment Guide with OpenShift
- Deployment in Disconnected OpenShift

NVIDIA Network Operator Deployment Guide with 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-OFED 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-OFED 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: stable
  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.

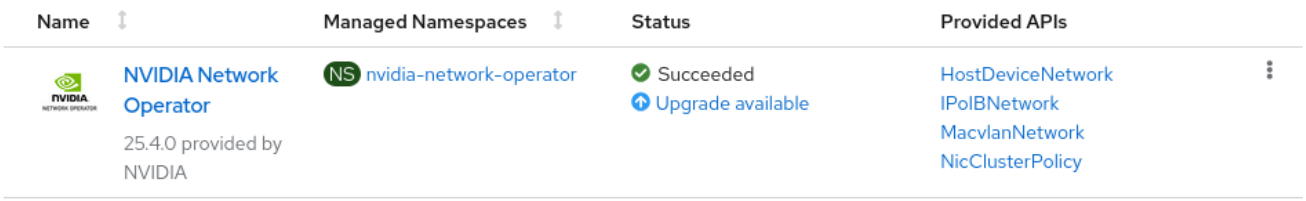
Network Operator Upgrade

This section describes how to upgrade the NVIDIA Network Operator on OpenShift Container Platform.

> **Note**
>
> Updating the NVIDIA Network Operator will not automatically update the NicClusterPolicy components. You will need to manually update the NicClusterPolicy components to the new version.

Upgrade Using OpenShift Web Console

- In the OpenShift Container Platform web console side menu, select Operators > Installed Operators, and search for the NVIDIA Network Operator.
- In case that the NVIDIA Network Operator has a pending update, it will display a status with Upgrade available like in the following image:



- Click on the *Upgrade Available* link, then click *Preview Install Plan* button.

- Review the install plan, and click *Approve* button to upgrade the NVIDIA Network Operator.
- Navigate back to the Operators -> Installed Operators page to monitor the progress of the update. When complete, the status changes to *Succeeded* and *Up to date*.
- For additional information, see the Red Hat OpenShift Container Platform Documentation upgrade guide.

Upgrade Using OpenShift OC CLI

1. Check the current subscription status to see if an upgrade is available:

```
oc get subscription nvidia-network-operator -n nvidia-
network-operator -o yaml
```

   Look for the following fields in the output:

   - *status.state*: Should show *UpgradePending* if an upgrade is available
   - *status.installedCSV*: Shows the currently installed version
   - *status.currentCSV*: Shows the available upgrade version
   - *status.installPlanRef.name*: The name of the install plan that requires approval

   Example output:

```
status:
  currentCSV: nvidia-network-operator.v25.7.0
  installedCSV: nvidia-network-operator.v25.4.0
  installPlanRef:
    name: install-r4pvj
  state: UpgradePending
```

2. List the install plans to identify the pending one:

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

Example output:

```
NAME            CSV                                APPROVAL
APPROVED
install-lrwp2   nvidia-network-operator.v25.4.0   Manual
true
install-r4pvj   nvidia-network-operator.v25.7.0   Manual
false
```

3. Review the install plan details before approving:

```
oc get installplan <install-plan-name> -n nvidia-network-
operator -o yaml
```

Replace *<install-plan-name>* with the name from the previous step (e.g., *install-r4pvj*).

4. Approve the install plan to proceed with the upgrade:

```
oc patch installplan <install-plan-name> -n nvidia-network-
operator \
  --type merge --patch '{"spec":{"approved":true}}'
```

5. Monitor the upgrade progress by checking the ClusterServiceVersion:

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

Wait until the new version shows *PHASE: Succeeded*:

```
NAME                                    DISPLAY
VERSION   REPLACES                                PHASE
nvidia-network-operator.v25.7.0   NVIDIA Network Operator
25.7.0    nvidia-network-operator.v25.4.0   Succeeded
```

6. Verify the operator pods are running with the new version:

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

Example output:

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

Note

After the upgrade is complete, remember to update the NicClusterPolicy components to match the new operator version if needed.

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 CNI Plugins, 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.

> **Note**
>
> In OCP, Multus is configured with *namespaceIsolation* enabled by default. This means that Pods using secondary networks should be deployed in the same namespace as the *network-attachment-definition* CR unless the NAD is in one of the following namespaces: *default*, *openshift-multus*, *openshift-sriov-network-operator* and *openshift-cnv*. The namespace of the NAD can be set in the *networkNamespace* field in *HostDeviceNetwork*, *MacvlanNetwork*, *IPoIBNetwork*, *OVSNetwork* and *SriovNetwork*.

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: doca3.3.0-26.01-1.0.0.0-0
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    config: |
      {
        "resourceList": [
            {
                "resourcePrefix": "nvidia.com",
                "resourceName": "hostdev",
                "selectors": {
                    "vendors": ["15b3"],
                    "isRdma": true
                }
            }
        ]
      }
```

```
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
```

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

To create an NV-IPAM IPPool, apply:

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

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": "nv-ipam",
       "poolName": "my-pool"
    }
```

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: doca3.3.0-26.01-1.0.0.0-0
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
```

SriovNetworkNodePolicy and K8s networking should be deployed.

NV-IPAM IPPool should be created before SriovNetwork:

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

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: |
    {
       "type": "nv-ipam",
       "poolName": "my-pool"
    }
```

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: doca3.3.0-26.01-1.0.0.0-0
     startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
  rdmaSharedDevicePlugin:
    config: |
      {
        "configList": [
          {
            "resourceName": "rdmashared",
            "rdmaHcaMax": 1000,
            "selectors": {
              "ifNames": ["enp4s0f0np0"]
            }
          }
        ]
      }
    image: k8s-rdma-shared-dev-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
  nvIpam:
```

```
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    imagePullSecrets: []
    enableWebhook: false
```

To create an NV-IPAM IPPool, apply:

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

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": "nv-ipam",
       "poolName": "my-pool"
    }
```

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.

NVIDIA Network Operator Deployment on Disconnected OpenShift

Deploying OpenShift operators in an environment with internet access is typically straightforward. However, in industries like cyber security, military sector or

Telco/communications, where security concerns often prohibit internet access, the process becomes more complex. In a disconnected or air-gapped environment, internet access is usually restricted or unavailable.

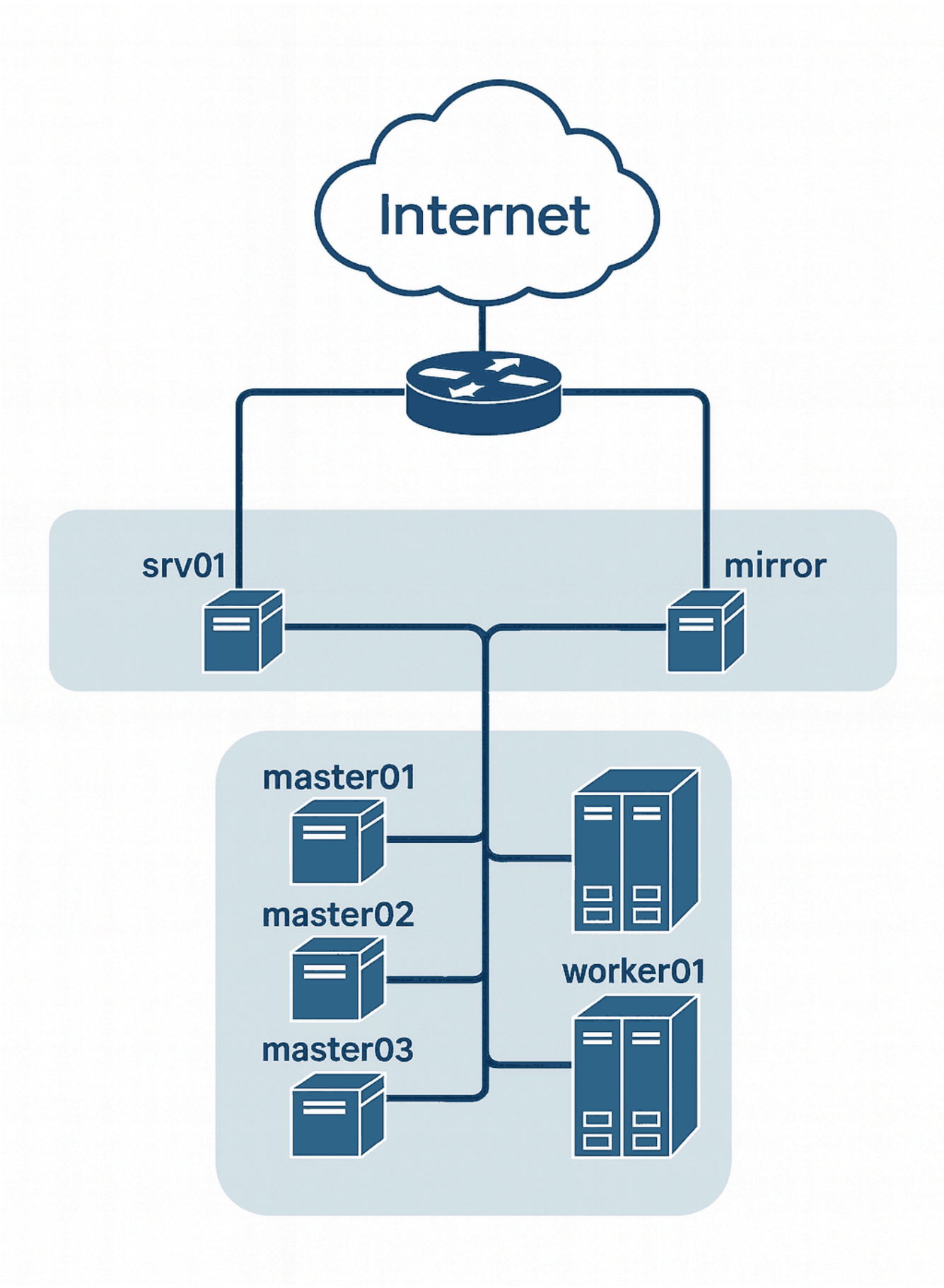
Prerequisites

The following requirements must be met:

- OpenShift is installed in a disconnected environment
- A private registry is deployed in the disconnected environment

Network Layout and Server Roles

The following network layout is used in this guide:


Internet
srv01
mirror
master01
master02
master03
worker01

Server Name	Services running on	Access to Internet
srv01	NTP, DHCP, DNS, HTTP	Yes
mirror	Mirroring tools, Local Registry	Yes
master01, master02, master03	Control Plane servers in a cluster	No
worker01, worker02	Workers in a cluster	No

Mirroring images from public registries

This section requires an internet connection, for fetching images from public registries. In our scenario we are using the *mirror* server for this operation.

Install OpenShift CLI

Follow the instructions in the OpenShift documentation to install the OpenShift CLI (*oc*).

Install oc-mirror plugin

To download the *oc-mirror* CLI plugin, navigate to the downloads page of the OpenShift Cluster Manager. Under the “OpenShift disconnected installation tools” section, click “Download” for “OpenShift Client (oc) mirror plugin” and save the file.

Extract the archive:

```
tar -xvf oc-mirror.tar.gz
chmod +x oc-mirror
```

Note

Do not rename the oc-mirror file

To install the *oc-mirror* CLI plugin, place the file in your PATH. For example:

```
sudo mv oc-mirror /usr/local/bin/
```

Verify that the plugin for *oc-mirror* is installed:

```
oc mirror help
```

Note

There is no dash (-) between oc and mirror.

If the command returns help options, then the *oc-mirror* CLI plugin is installed.

Configure the oc-mirror credentials

Download your *registry.redhat.io* pull secret from Red Hat OpenShift Cluster Manager.

Convert the pull secret to JSON format:

```
cat ./pull-secret | jq . > config.json
```

Copy the JSON file to the *.docker* directory:

```
mkdir ~/.docker
cp config.json ~/.docker/config.json
```

Log in to *registry.redhat.io*:

```
podman login registry.redhat.io
```

Create the oc-mirror ISC (ImageSourceConfiguration)

The oc-mirror tool uses an ISC configuration file to define what needs to be mirrored. This configuration allows you to specify the catalog source name and the versions of the operators you want to mirror.

Find operator channel and release information for the required OpenShift version (example: 4.19):

```
oc mirror list operators --catalogs --version=4.19
Available OpenShift OperatorHub catalogs:
OpenShift 4.19:
registry.redhat.io/redhat/redhat-operator-index:v4.19
registry.redhat.io/redhat/certified-operator-index:v4.19
registry.redhat.io/redhat/community-operator-index:v4.19
registry.redhat.io/redhat/redhat-marketplace-index:v4.19
```

In order to mirror NFD and NVIDIA Network-Operator, we will use the following catalogs:

- *redhat-operator-index*
- *certified-operator-index*

Find the channel for NFD operator:

```
oc mirror list operators --
catalog=registry.redhat.io/redhat/redhat-operator-index:v4.19 --
package=nfd
NAME  DISPLAY NAME  DEFAULT CHANNEL
nfd                 stable
PACKAGE  CHANNEL  HEAD
nfd      stable   nfd.4.19.0-202508120121
```

Find the channel for NVIDIA Network Operator:

```
oc mirror list operators --
catalog=registry.redhat.io/redhat/certified-operator-index:v4.19
--package=nvidia-network-operator
NAME                     DISPLAY NAME  DEFAULT CHANNEL
nvidia-network-operator                v25.7
PACKAGE                  CHANNEL  HEAD
nvidia-network-operator  stable   nvidia-network-operator.v25.7.0
nvidia-network-operator  v25.7    nvidia-network-operator.v25.7.0
```

Use the *oc mirror* command to create a template for the ISC configuration file.

```
oc mirror init > imagesset-config.yaml
```

Edit the new ISC file and update it using the right catalog and the operators release information you gathered in the previous steps. Here's an example of an ISC configuration file for our scenario:

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
storageConfig:
  local:
    path: /path/to/disk-mirror-dir/metadata
mirror:
  platform:
    channels:
    - name: stable-4.19
      minVersion: 4.19.0
      maxVersion: 4.19.0
      type: ocp
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-
index:v4.19
    packages:
    - name: nfd
      channels:
      - name: stable
  - catalog: registry.redhat.io/redhat/certified-operator-
index:v4.19
    packages:
    - name: nvidia-network-operator
      channels:
      - name: stable
  additionalImages:
  - name: registry.redhat.io/ubi8/ubi:latest
  helm: {}
```

Note

Please keep in mind that you can decide what will be the max and min version of any channel in any catalog according to your requirements. In our case, to save space on storage, we minimized it to one version only.

Create the operators' images TAR file

Create a directory for the image's TAR file, and then run the command:

```
mkdir data
oc mirror --verbose 3 --config=imagesset-config.yaml file://data
```

Once the operation is completed, the TAR file will appear in created directory:

```
ls -ltrh data/
total 35G
drwxr-xr-x. 3 root root  17 Jan 27 07:36 oc-mirror-workspace
-rw-r--r--. 1 root root 35G Jan 27 07:40 mirror_seq1_000000.tar
```

Upload the container images to the private registry

Move the TAR file to the disconnected environment

The operational procedures and security requirements can vary significantly from one organization to another. For example, in certain highly restricted environments, you must copy the TAR file to a portable disk, which you take to your disconnected environment's security team for review.

Mirroring the operator from disk to private registry

Once the data has been moved to the disconnected environment, you can begin pushing the images in your private registry. Because you've used the *oc-mirror* plugin to push all necessary container images, you must configure the tools accordingly, with the credential information pointing to your private registry instead of the Red Hat public registry.

This is an example of *config.json* file for local registry:

```
cat /mirror-data/config.json
{
  "auths": {
    "mirror.air-gapped.local:5000": {
    "auth": "aW5pdDpxYXdzMTI="
    }
  }
}
```

Create a directory to copy the TAR file, and to serve as the destination for *oc mirror* output:

```
mkdir data
```

Copy the TAR file to the destination directory:

```
mkdir data
cp mirror_seq1_000000.tar data/
cd data
```

Execute the *oc mirror* command to upload the container's images to your private registry:

```
oc mirror --verbose 3 \
--from=./mirror_seq1_000000.tar \
docker://mirror.air-gapped.local:5000/ocp/openshift4
```

Once complete, you get the following directories with similar files:

```
ls -ltr data/oc-mirror-workspace/results-1737982710/
total 64
drwxr-xr-x. 2 root root       6 Jan 27 07:58 charts
drwxr-xr-x. 2 root root       52 Jan 27 08:03 release-signatures
-rw-r--r--. 1 root root 49619 Jan 27 08:03 mapping.txt
-rwxr-xr-x. 1 root root   253 Jan 27 08:03 catalogSource-cs-
redhat-operator-index.yaml
-rwxr-xr-x. 1 root root   259 Jan 27 08:03 catalogSource-cs-
certified-operator-index.yaml
-rwxr-xr-x. 1 root root  1565 Jan 27 08:03
imageContentSourcePolicy.yaml
```

Configure the Openshift catalog

You must be logged in to your OpenShift instance as a user with cluster-admin privileges. In our setup we are using kubeconfig file created with OCP deployment in isolated network:

```
export KUBECONFIG=/mirror-
data/iso_build/brm_static/auth/kubeconfig
```

Keeping the Openshift default catalogs in a disconnected environment would just trigger unnecessary errors, so disable the default OperatorHub catalog sources:

```
$ oc patch OperatorHub cluster --type json -p '[{"op": "add",
"path": "/spec/disableAllDefaultSources", "value": true}]'
```

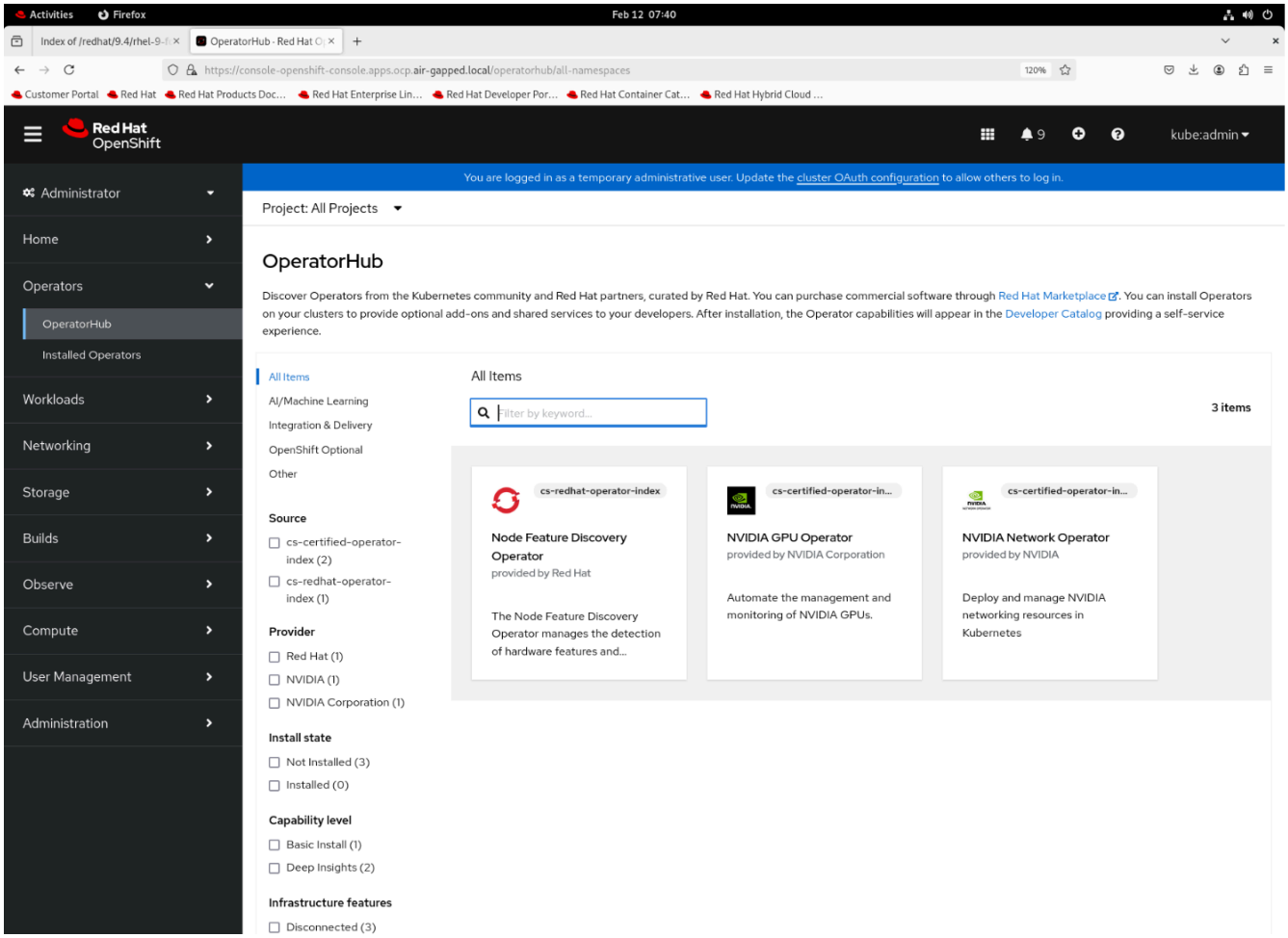
Apply the ICSP yaml file:

```
oc apply -f /mirror-data/data/oc-mirror-workspace/results-
1737982710/imageContentSourcePolicy.yaml
```

Next, create the Red Hat catalog, you could run it for each catalog workspace that you have. In our scenario it's: *certified-operator-index* and *redhat-operator-index*:

```
oc apply -f  data/oc-mirror-workspace/results-
1737982710/catalogSource-cs-redhat-operator-index.yaml
oc apply -f data/oc-mirror-workspace/results-
1737982710/catalogSource-cs-certified-operator-index.yaml
```

Verify that you can see the catalog in the mirrored operators in UI "Operators > OperatorHub":



Installing Operators in the disconnected environment

The easiest way to install the operators in OpenShift it's using the OpenShift UI. Once you have the OpenShift cluster UI open, follow these steps:

1. Click on Operators > OperatorHub
2. Select the operator to install
3. Keep the default settings and click Install

Create the instances

After successful installation of all operators you must create the instances or polices depending on your use case.

Create the NFD instance

In the web console, click “Operators > Installed Operators”, and then “Node Feature Discovery Operator” In Details > Provided API’s look for (NFD) NodeFeatureDiscovery “Create instance”

Once the instance creation will completed you can find it in “Operators > Installed Operators > Node Feature Discovery Operator > All instances”

NVIDIA DOCA OFED driver container in disconnected environment

In case you want to use the NVIDIA DOCA OFED driver container in the disconnected environment, there are two options:

- Option 1: Use the NVIDIA DOCA OFED driver container from NGC
    - This container builds the DOCA OFED driver from source code dynamically, therefore requires mirroring needed dependencies.
- Option 2: Create a precompiled container for the DOCA OFED driver
    - With this option it is not required to mirror dependencies, but it will support only a specific kernel version.

Option 1: Use the NVIDIA DOCA OFED driver container from NGC

Mirroring DOCA OFED Driver Container

Navigate to the NVIDIA catalog and looking for the right <os-version>-<architecture> suffix tag, such as *doca3.1.0-25.07-0.9.7.0-0-rhel9.6-amd64*.

The mirrored image must be tagged *<driver-version>-<os-version>-<architecture>*, such as *doca3.1.0-25.07-0.9.7.0-0-rhel9.6-amd64* for example.

Note that since OCP 4.19, the os version is now *rhel9.6* instead of *rhcos4.x*.

Create Local Package Repositories

The DOCA-OFED Driver container requires certain packages to be available for the driver installation. The following packages are required:

```
kernel-headers-${KERNEL_VERSION}
kernel-devel-${KERNEL_VERSION}
kernel-core-${KERNEL_VERSION}
createrepo
elfutils-libelf-devel
kernel-rpm-macros
umactl-libs
lsof
rpm-build
patch
hostname
```

For RT kernels following packages should be available:

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

Create the Local Package Repository required:

- redhat.repo
- ubi.repo
- cuda.repo

The detailed instructions and examples about how to create local repositories is available in this article . Instructions on how to create a repo file can be found here.

redhat.repo:

```
[baseos]
name=rhel-9-for-x86_64-baseos-rpms
baseurl=http://srv01.air-gapped.local/redhat/9.4/el-9-for-x86_64-
baseos-rpms
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
enabled=1
[apstream]
name=rhel-9-for-x86_64-appstream-rpms
baseurl=http://srv01.air-gapped.local/redhat/rhel-9-for-x86_64-
appstream-rpms
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
enabled=1
```

ubi.repo:

```
[ubi-9-baseos]
name = Red Hat Universal Base Image 9 (RPMs) - BaseOS
baseurl = http://srv01.air-gapped.local/redhat/ubi-9-baseos-rpms
enabled = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
[ubi-9-appstream]
name = Red Hat Universal Base Image 9 (RPMs) - AppStream
baseurl = http://srv01.air-gapped.local/redhat/ubi-9-appstream-
rpms
enabled = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
```

cuda.repo:

```
[cuda]
name=cuda
baseurl=http://srv01.air-gapped.local/nvidia/cuda
priority=0
gpgcheck=1
gpgkey=http://srv01.air-gapped.local/nvidia/cuda/D42D0685.pub
enabled=1
```

Create the ConfigMap for the repos files:

```
oc create configmap repo-config -n nvidia-network-operator  --
from-file=redhat.repo --from-file=ubi.repo --from-file=cuda.repo
```

If self-signed certificates are used for an HTTPS based local repository, a ConfigMap must be created for those certificates:

```
oc create configmap cert-config -n nvidia-network-operator --
from-file=<path-to-pem-file>
```

Create the NIC Cluster Policy instance

In the web console, click “Operators > Installed Operators”, and then “NVIDIA Network Operator > NicClusterPolicy > Create NicClusterPolicy”

You need to provide required parameters depending on your setup. After editing and overriding our nic-cluster-policy yaml looks like this:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    certConfig:
      name: cert-config
    env:
    - name: RESTORE_DRIVER_ON_POD_TERMINATION
      value: "true"
    - name: UNLOAD_STORAGE_MODULES
      value: "true"
    - name: CREATE_IFNAMES_UDEV
      value: "true"
    forcePrecompiled: false
    image: doca-driver
    imagePullSecrets:
    - mirror-registry-ps
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    repoConfig:
      name: repo-config
    repository: mirror.air-gapped.local:5000/mellanox
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    terminationGracePeriodSeconds: 300
    upgradePolicy:
      autoUpgrade: true
      drain:
```

```
      deleteEmptyDir: true
      enable: true
      force: true
      podSelector: ""
      timeoutSeconds: 300
    maxParallelUpgrades: 1
    safeLoad: false
    waitForCompletion:
      timeoutSeconds: 0
  version: doca3.1.0-25.07-0.9.7.0-0
```

Note: Please be sure to provide configured ConfigMaps: *repo-config* and *cert-config*.

Option 2: Create a precompiled container for the DOCA OFED driver

Please verify that you have the following:

- Podman (RH) installed on your build system
- Web access to NVIDIA NIC drivers sources

Follow the instructions in the Precompiled Container Build Instructions for NVIDIA DOCA-OFED Driver Container section to build the precompiled container.

> **Note**
>
> You need to make the created image accessible in your environment (on the local registry server). Working with container images is described in this page.

In order to get the sha256 of the image, you can use the following command:

```
skopeo inspect docker://mirror.air-
gapped.local:5000/mellanox/doca-driver:your-tag | jq -r '.Digest'
```

Create the NIC Cluster Policy instance

In the web console, click “Operators > Installed Operators”, and then “NVIDIA Network Operator > NicClusterPolicy > Create NicClusterPolicy”

You need to provide required parameters depending on your setup. After editing and overriding our nic-cluster-policy yaml looks like this:

```
apiVersion: mellanox.com/v1alpha1
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"
    forcePrecompiled: true
    image: doca-driver
    imagePullSecrets:
    - mirror-registry-ps
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 30
    readinessProbe:
      initialDelaySeconds: 10
      periodSeconds: 30
    repository: mirror.air-gapped.local:5000/mellanox
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    terminationGracePeriodSeconds: 300
    upgradePolicy:
      autoUpgrade: true
      drain:
        deleteEmptyDir: true
        enable: true
        force: true
        podSelector: ""
```

```
        timeoutSeconds: 300
      maxParallelUpgrades: 1
      safeLoad: false
      waitForCompletion:
        timeoutSeconds: 0
    version:
sha256:9a831bfdf85f313b1f5749b7c9b2673bb8fff18b4ff768c9242dabaa4468
```

Image Pull Secrets

If your local repository requires username and password for access you need to create imagePullSecrets and provide this parameter in nic-cluster-policy.yaml:

```
imagePullSecrets:
- mirror-registry-ps
```

How to create imagePullSecrets:

```
oc -n nvidia-network-operator create secret docker-registry \
--docker-server=mirror.air-gapped.local:5000 \
--docker-username=init \
--docker-password=qaws12 \
mirror-registry-ps
```

NVIDIA Network Operator Government Ready

The NVIDIA Network Operator now offers government-ready components for NVIDIA AI Enterprise customers. Government ready is NVIDIA's designation for software that meets applicable security requirements for deployment in your FedRAMP High or equivalent sovereign use case. For more information on NVIDIA's government-ready support, refer to the white paper AI Software for Regulated Environments.

Government-Ready Components Requiring NGC Access

Most Network Operator components are available as government-ready containers in the public container registry. However, the following STIG-FIPS certified components require an NGC API key and are available from a separate NGC repository:

Component	Repository	Image Name	Version
DOCA-OFED Driver Container	nvcr.io/nvidia/mellanox	doca-driver-stig-fips	doca3.3.0-26.01-1.0.0.0-4
SR-IOV Network Operator Config Daemon	nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon-stig-fips	network-operator-v26.1.1-stig-fips
NIC Configuration Operator	nvcr.io/nvidia/mellanox	nic-configuration-operator-stig-fips	network-operator-v26.1.1-stig-fips-ubuntu / network-operator-v26.1.1-stig-fips-rhel

NIC Configuration Operator Daemon	nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon-stig-fips	network-operator-v26.1.1-stig-fips-ubuntu / network-operator-v26.1.1-stig-fips-rhel
Spectrum X Operator	nvcr.io/nvidia/mellanox	spectrumx-operator-stig-fips	network-operator-v26.1.1-stig-fips-ubuntu / network-operator-v26.1.1-stig-fips-rhel

Note

All other Network Operator components are government-ready and available in the standard public container registry.

Validated Kubernetes Distributions

The government-ready NVIDIA Network Operator has been validated on the following Kubernetes distributions:

- Canonical Kubernetes 1.35 with Ubuntu Pro 24.04 amd64 and FIPS-compliant kernel
- Red Hat OpenShift Container Platform (OCP) 4.20 or newer with FIPS mode enabled

Common Prerequisites

The following prerequisites apply to all supported platforms:

- An active NVIDIA AI Enterprise subscription and NGC API token to access Network Operator government-ready containers. Refer to Generating Your NGC API Key in the NVIDIA NGC User Guide for more information on NGC API tokens.

- A namespace to deploy the NVIDIA Network Operator. The example install commands below use `nvidia-network-operator` as the namespace.

- Optionally, Service Mesh for intra-cluster traffic encryption. By default, the NVIDIA Network Operator does not encrypt traffic between its controller (and operands) and the Kubernetes API server. If you wish to encrypt this communication, you should deploy and maintain a service mesh application within the Kubernetes cluster to enable secure traffic.

Ubuntu/Canonical Kubernetes

Prerequisites

- The `helm` CLI installed 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
```

- An Ubuntu Pro token for Canonical Kubernetes deployments. This token is required for the driver container to download kernel headers and other necessary packages from the Canonical repository when using the FIPS-enabled kernel on Ubuntu 24.04. Refer to the Ubuntu Pro documentation for more information on accessing Ubuntu Pro tokens.

Create NGC API Pull Secret

Add a Docker registry secret for downloading the Network Operator artifacts from NVIDIA NGC in the same namespace where you are planning to deploy the NVIDIA Network Operator. Update `ngc-api-key` in the command below with your NGC API key.

```
$ kubectl create secret -n nvidia-network-operator docker-
registry ngc-secret \
    --docker-server=nvcr.io \
    --docker-username='$oauthtoken' \
    --docker-password=<ngc-api-key>
```

Install Network Operator Using Helm

1. Label your `nvidia-network-operator` namespace for the Operator to set the enforcement policy to privilege.

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

2. Add the NVIDIA Helm repository:

   ```
   $ helm repo add nvidia https://helm.ngc.nvidia.com/nvidia \
       && helm repo update
   ```

3. Install the NVIDIA Network Operator with SR-IOV Network Operator.

```
$  helm install network-operator nvidia/network-operator \
     --namespace nvidia-network-operator \
     --set sriov-network-
operator.images.sriovConfigDaemon=nvcr.io/nvidia/mellanox/sriov
network-operator-config-daemon-stig-fips:network-operator-
v26.1.1-stig-fips \
     --set sriov-network-operator.imagePullSecrets={ngc-
secret} \
     --set sriovNetworkOperator.enabled=true \
     --set nfd.enabled=true
```

Configure NicClusterPolicy

For the NVIDIA DOCA OFED Driver, the `UBUNTU_PRO_TOKEN` environment variable in the NicClusterPolicy should be configured. This token is required for the driver container to download kernel headers and other necessary packages from the Canonical repository when using the FIPS-enabled kernel on Ubuntu 24.04.

> **Note**
>
> The following example demonstrates how to configure the government-ready components that require NGC access (STIG-FIPS variants not available in the public registry). This example should be adapted to your specific environment. You can add other components as needed from the standard Network Operator configuration.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver-stig-fips
    repository: nvcr.io/nvidia/mellanox
    version: doca3.3.0-26.01-1.0.0.0-4
    imagePullSecrets:
      - ngc-secret
    env:
      - name: UBUNTU_PRO_TOKEN
        value: "<YOUR_UBUNTU_PRO_TOKEN>"
  spectrumXOperator:
    image: spectrum-x-operator-stig-fips
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1-stig-fips-ubuntu
    imagePullSecrets:
      - ngc-secret
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1-stig-fips-ubuntu
      imagePullSecrets:
        - ngc-secret
    configurationDaemon:
      image: nic-configuration-operator-daemon-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1-stig-fips-ubuntu
      imagePullSecrets:
        - ngc-secret
    nicFirmwareStorage:
      create: true
```

```
    pvcName: nic-fw-storage-pvc
    storageClassName: nic-fw-storage-class
    availableStorageSize: 1Gi
  logLevel: info
```

Red Hat OpenShift Container Platform

Prerequisites

- An OpenShift Container Platform cluster installed in FIPS mode. Refer to the OpenShift FIPS Installation Guide for detailed instructions on installing OpenShift Container Platform in FIPS mode.

> **Note**
>
> To enable FIPS mode for your OpenShift cluster, you must run the installation program from a RHEL 9 computer that is configured to operate in FIPS mode. Use a FIPS-capable version of the installation program and set `fips: true` in the `install-config.yaml` file before cluster deployment.

Verify FIPS Mode

Before proceeding with the Network Operator configuration, verify that your OpenShift cluster is running in FIPS mode.

You can check FIPS mode by running the following command on any node:

```
$ oc debug node/<node-name> -- chroot /host cat
/proc/sys/crypto/fips_enabled
```

The output should be `1` if FIPS mode is enabled.

All nodes should report `1` when checking `/proc/sys/crypto/fips_enabled`.

Install Network Operator

For OpenShift Container Platform, follow the standard OpenShift Operator installation process using the OpenShift Catalog or OC CLI.

Refer to the NVIDIA Network Operator Deployment Guide with OpenShift for detailed instructions on installing the operator via:

- OpenShift Web Console (OperatorHub)
- OpenShift OC CLI

Create NGC API Pull Secret

Add a Docker registry secret for downloading the Network Operator government-ready artifacts from NVIDIA NGC:

```
$ oc create secret -n nvidia-network-operator docker-registry
ngc-secret \
    --docker-server=nvcr.io \
    --docker-username='$oauthtoken' \
    --docker-password=<ngc-api-key>
```

Replace `<ngc-api-key>` with your NGC API key.

Configure NicClusterPolicy

For OpenShift deployments, create or update the NicClusterPolicy to use the government-ready FIPS images.

Note

The following example demonstrates how to configure the government-ready components that require NGC access (STIG-FIPS variants not available in the public registry). This example should be adapted to your specific environment. You can add other components as needed from the standard Network Operator configuration.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver-stig-fips
    repository: nvcr.io/nvidia/mellanox
    version: doca3.3.0-26.01-1.0.0.0-4
    imagePullSecrets:
      - ngc-secret
  spectrumXOperator:
    image: spectrum-x-operator-stig-fips
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1-stig-fips-rhel
    imagePullSecrets:
      - ngc-secret
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1-stig-fips-rhel
      imagePullSecrets:
        - ngc-secret
    configurationDaemon:
      image: nic-configuration-operator-daemon-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1-stig-fips-rhel
      imagePullSecrets:
        - ngc-secret
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      storageClassName: nic-fw-storage-class
      availableStorageSize: 1Gi
```

```
    logLevel: info
```

Additional Considerations

- **Security Context Constraints (SCC)**: The Network Operator requires privileged access. OpenShift will automatically apply the appropriate SCCs when the operator is deployed in the `nvidia-network-operator` namespace.
- **etcd Encryption**: For enhanced security in FIPS mode, consider enabling etcd encryption using the FIPS-approved AES CBC cryptographic algorithm. Refer to the OpenShift documentation on encrypting etcd data.
- **Storage**: If using persistent storage, ensure it uses RHEL-provided disk encryption for data at rest protection in FIPS environments.
- **Network Policy**: OpenShift's OVN-Kubernetes network plugin is FIPS-compliant. Ensure any additional network policies are compatible with your FIPS requirements.

NIC Configuration Operator

NIC Firmware Configuration

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.

> ⚠ **Warning**
>
> NVIDIA Networking NIC Configuration Operator doesn't support Socket Direct Adapters.

For more information about the CRD API, refer to CRD API Reference.

Use of NIC Configuration Operator together with SR-IOV Network Operator

NIC Configuration Operator can be used together with SR-IOV Network Operator to configure SR-IOV VFs on NVIDIA NICs. In this scenario, NIC Configuration Operator takes on the NIC FW Configuration, while SR-IOV Network Operator configures the SR-IOV VFs.

There are two requirements for the SR-IOV Network Operator to work together with NIC Configuration Operator:

1. NodeSelector for the SR-IOV Config Daemon should include the `network.nvidia.com/operator.nic-configuration.wait: "false"` label. It's managed by the NIC Configuration Operator and ensures that the SR-IOV Config Daemon is not started before the NIC Configuration is complete and ready.

> **Note**
>
> When the SR-IOV Network Operator is deployed via the Network Operator Helm chart, the Node Selector should be configured via the Network Operator Helm chart values.

`values.yaml`:

```
nfd:
  enabled: true
maintenanceOperator:
  enabled: true
sriovNetworkOperator:
  enabled: true
sriov-network-operator:
  sriovOperatorConfig:
    configDaemonNodeSelector:
      beta.kubernetes.io/os: "linux"
      network.nvidia.com/operator.mofed.wait: "false"
      # Enable when using together with NIC Configuration
Operator to wait until
      # all required FW parameters are successfully applied
before configuring SR-IOV
      network.nvidia.com/operator.nic-configuration.wait: "false"
```

2. `mellanox` plugin should be disabled in the *SriovOperatorConfig* CR.

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io -n
nvidia-network-operator default --patch '{ "spec": {
"disablePlugins": ["mellanox"]} }' --type='merge'
```

> ⚠ **Warning**
>
> SR-IOV Network Operator can work together with the NIC Configuration Operator only in `daemon` configuration mode. `systemd` configuration mode is not supported with this scenario.

Install the NIC Configuration Operator and observe NIC devices in the cluster

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. To disable the Firmware upgrade and validation logic, do not define the `nicFirmwareStorage` section in the NicClusterPolicy CR.

Note

On some DGX servers, the configuration update is not successfully applied after the warm reboot. In this case, it is recommended to explicitly reset the NIC's Firmware before the reboot and after updating its non-volatile configuration. This can be achieved by specifying the `FW_RESET_AFTER_CONFIG_UPDATE` environment variable in the NicClusterPolicy CR. Please see the commented section in the example below.

First install the Network Operator helm chart with the Maintenance Operator enabled and deploy a NIC Cluster Policy CRD with NIC Configuration Operator and DOCA-OFED Driver 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: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.1.1
    # Uncomment to explicitely reset the NIC's Firmware before
the reboot and after updating its non-volatile configuration.
    # Might be required on DGX servers where configuration update
is not successfully applied after the warm reboot.
    # env:
    # - name: "FW_RESET_AFTER_CONFIG_UPDATE"
    # value: "true"
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      # Name of the storage class is provided by the user
      storageClassName: nfs-csi
      availableStorageSize: 1Gi
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: doca3.3.0-26.01-1.0.0.0-0
    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
```

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
```

Update NIC Firmware using the NIC Configuration Operator

Configure and apply the NICFirmwareSource CR

Deploy the NICFirmwareSource CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareSource
metadata:
  name: connectx6-dx-firmware-22-44-1036
  namespace: network-operator
  finalizers:
    - configuration.net.nvidia.com/nic-configuration-operator
spec:
  # a list of firmware binaries from mlnx website if they are
zipped try to unzip before placing
  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
```

Note

The ConnectX firmware binaries can be downloaded from the NVIDIA Networking Firmware Downloads page. The URLs of the firmware binaries from the website can be directly provided in the binUrlSources field of the NicFirmwareSource CR.

Note

BlueField Bundle (BFB) can be downloaded from the NVIDIA DOCA Downloads page. The file should first be made available in the cluster and then its URL should be provided in the bfbUrlSource field of the NicFirmwareSource CR.

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

Configure and apply the NicFirmwareTemplate CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareTemplate
metadata:
  name: connectx6dx-config
  namespace: network-operator
spec:
  nodeSelector:
    kubernetes.io/hostname: cloud-dev-41
  nicSelector:
    nicType: "101d"
  template:
    nicFirmwareSourceRef: connectx6dx-firmware-22-44-1036
    updatePolicy: Update
```

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
```

Status conditions of the NicDevice CR reflect the status of the firmware 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"
```

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-OFED Driver is not installed otherwise it notifies if current NIC firmware is recommended or not recommended by DOCA-OFED 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"
```

Configure NIC Firmware using the NIC Configuration Operator

Configure and apply the NicConfigurationTemplate CR

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
  name: connectx6dx-config
  namespace: 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: 101b
    pciAddresses:
      - "0000:03:00.0"
      - “0000:04:00.0”
    serialNumbers:
      - "MT2116X09299"
  resetToDefault: false # if set, template is ignored, device
configuration should reset
  template:
    numVfs: 2
    linkType: Ethernet
    pciPerformanceOptimized:
      enabled: true
      # default values for maxAccOutRead and maxReadRequest
listed below, can be omitted
      maxAccOutRead: 44
      maxReadRequest: 4096
    roceOptimized:
      enabled: true
      # default values for qos listed below, can be omitted
      qos:
        trust: dscp
        pfc: "0,0,0,1,0,0,0,0"
    gpuDirectOptimized:
```

```
    enabled: true
    env: Baremetal
  rawNvConfig:
    - name: THIS_IS_A_SPECIAL_NVCONFIG_PARAM
      value: "55"
    - name: SOME_ADVANCED_NVCONFIG_PARAM
      value: "true"
```

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.

For detailed information about firmware parameters and configuration settings, refer to Configuration Details.

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
```

Observe the status of the configuration update

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"
```

Note

If both Firmware update and configuration are applied to a single device, the firmware update should be performed first. The configuration update will be applied after the firmware update is completed.

Reset NIC Configuration to Default

The NIC Configuration Operator supports resetting NIC non-volatile configuration to factory defaults using the `resetToDefault` field in the `NicConfigurationTemplate` CR. When enabled, the operator performs the following operations:

- Resets all non-volatile configurations (`mstconfig -d <device> reset` for each PF)
- Sets `ADVANCED_PCI_SETTINGS=1`

- Reboots the node to apply the new NIC NV configuration and undo any runtime configuration previously performed for the device or driver

> **⚠ Warning**
>
> A configuration reset triggers a node reboot. Ensure that workloads are drained or that the Maintenance Operator is configured to handle the node maintenance automatically.

To reset the NIC configuration, create a `NicConfigurationTemplate` CR with `resetToDefault: true`:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
  name: reset-nic-config
  namespace: nvidia-network-operator
spec:
  nicSelector:
    nicType: "1023"
  resetToDefault: true
  template:
    numVfs: 0
    linkType: Ethernet
```

> **ⓘ Note**
>
> When `resetToDefault` is set to `true`, the `template` section is ignored. The device configuration is reset to factory defaults regardless of the template contents.

After the reset is complete and the node has rebooted, you can remove the reset CR and apply the desired configuration template.

Configure custom interface names (NicInterfaceNameTemplate)

The `NicInterfaceNameTemplate` CRD allows you to define custom naming patterns for RDMA and network device interfaces on NVIDIA NICs. This is useful in Spectrum-X multiplane and multi-rail deployments where predictable interface naming is required.

The operator deploys udev rules to the host to rename network and RDMA interfaces according to the specified naming template. The template uses placeholders (`%nic_id%`, `%plane_id%`, `%rail_id%`) to construct device names based on the NIC topology.

For full details on NicInterfaceNameTemplate configuration, including multiplane modes and example udev rules, refer to Spectrum-X Configuration.

Configuration Details

Configuration details

- `numVFs`: if provided, configure SR-IOV VFs via nvconfig.
  - This is a mandatory parameter.
  - E.g: if `numVFs=2` then `SRIOV_EN=1` and `SRIOV_NUM_OF_VFS=2`.
  - If `numVFs=0` then `SRIOV_EN=0` and `SRIOV_NUM_OF_VFS=0`.
- `linkType`: if provided configure `linkType` for the NIC for all NIC ports.
  - This is a mandatory parameter.
  - E.g `linkType = Infiniband` then set `LINK_TYPE_P1=IB` and `LINK_TYPE_P2=IB` if second PCI function is present
- `pciPerformanceOptimized`: performs PCI performance optimizations. If enabled then by default the following will happen:

- Set nvconfig `MAX_ACC_OUT_READ` nvconfig parameter to `0` (use device defaults)
- Set PCI max read request size for each PF to `4096` (note: this is a runtime config and is not persistent)
- Users can override values via `maxAccOutRead` and `maxReadRequest`
- **IMPORTANT** :
  - According to the PRM, setting `MAX_ACC_OUT_READ` to zero enables the auto mode, which applies the best suitable optimizations. However, there is a bug in certain FW versions, where the zero value is not available.
  - In this case, until the fix is available, `MAX_ACC_OUT_READ` will not be set and a warning event will be emitted for this device's CR.

- `roceOptimized`: performs RoCE related optimizations. If enabled performs the following by default:
  - Nvconfig set for both ports (can be applied from PF0)
    - Conditionally applied for second port if present
      - `ROCE_CC_PRIO_MASK_P1=255`, `ROCE_CC_PRIO_MASK_P2=255`
      - `CNP_DSCP_P1=4`, `CNP_DSCP_P2=4`
      - `CNP_802P_PRIO_P1=6`, `CNP_802P_PRIO_P2=6`
  - Configure pfc (Priority Flow Control) for priority 3, set trust to dscp on each PF, set ToS (Type of Service) to 0.
    - Non-persistent (need to be applied after each boot)
    - Users can override values via `trust`, `pfc` and `tos` parameters
  - Can only be enabled with `linkType=Ethernet`

- `gpuDirectOptimized`: performs gpu direct optimizations. ATM only optimizations for Baremetal environment are supported. If enabled perform the following:

    - Set nvconfig `ATS_ENABLED=0`
    - Can only be enabled when `pciPerformanceOptimized` is enabled
    - Both the numeric values and their string aliases, supported by NVConfig, are allowed (e.g. `REAL_TIME_CLOCK_ENABLE=False`, `REAL_TIME_CLOCK_ENABLE=0`).
    - For per port parameters (suffix `_P1`, `_P2`) parameters with `_P2` suffix are ignored if the device is single port.
- `spectrumXOptimized`: enables Spectrum-X specific NIC optimizations. When enabled:
    - Requires `linkType=Ethernet` and `numVfs=1`
    - Cannot be combined with `roceOptimized` (RoCE settings are included automatically)
    - Cannot be combined with `rawNvConfig`
    - Only supported on ConnectX-8 (`nicType: 1023`) and BlueField-3 SuperNIC (`nicType: a2dc`)
    - `version`: Required. Reference Architecture version (`RA1.3`, `RA2.0`, or `RA2.1`)
    - `overlay`: Optional, default `none`. Set to `l3` for L3 EVPN overlay
    - `multiplaneMode`: Optional, default `none`. Only available with RA2.1. Options: `none`, `swplb`, `hwplb`, `uniplane`
    - `numberOfPlanes`: Optional, default `1`. Only available with RA2.1. Options: `1`, `2`, or `4`
- If a configuration is not set in spec, its non-volatile configuration parameters (if any) should be set to device default.

Spectrum-X Configuration

The NIC Configuration Operator supports Spectrum-X-specific NIC configuration for different versions of the NVIDIA Spectrum-X Reference Architecture (RA1.3, RA2.0, and RA2.1).

Supported NIC types for Spectrum-X: * ConnectX-8 (device ID `1023`) – supports all multiplane modes * BlueField-3 SuperNIC (device ID `a2dc`) – supports all multiplane modes except `hwplb`

RA2.1 introduces multiplane mode support, allowing NICs to be configured with multiple data planes. Available modes:

Mode	Description	Supported NICs	Planes
`none`	Single plane (default)	ConnectX-8, BF3 SuperNIC	1
`swplb`	Software Packet Load Balancing	ConnectX-8, BF3 SuperNIC	2, 4
`hwplb`	Hardware Packet Load Balancing	ConnectX-8 only	2, 4
`uniplane`	Uniplane mode	ConnectX-8, BF3 SuperNIC	2

Note: Multiplane modes are only available with RA2.1. For RA1.3 and RA2.0, `multiplaneMode` must be `none` and `numberOfPlanes` must be `1`.

Example Spectrum-X NicConfigurationTemplate with multiplane:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
  name: spectrum-x-multiplane-configuration
  namespace: nvidia-network-operator
spec:
  nodeSelector:
      feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
      nicType: "1023" # ConnectX-8. Use "a2dc" for BlueField-3
SuperNIC (hwplb not supported on BF3)
  template:
      numVfs: 1
      linkType: Ethernet
      spectrumXOptimized:
          enabled: true
          version: "RA2.1"
          overlay: "none"
          multiplaneMode: "hwplb" # Hardware Packet Load
Balancing, ConnectX-8 only
          numberOfPlanes: 4
```

[TECH PREVIEW] NVIDIA Spectrum-X NIC Configuration

NVIDIA NIC Configuration Operator offers NVIDIA Spectrum-X-specific NIC configuration for different versions of the Reference Architecture (RA1.3, RA2.0, and RA2.1). RA2.1 introduces multiplane mode support for enhanced network performance with multiple data planes.

Note

Currently, only ConnectX-8 (device ID `1023`) and BlueField-3 SuperNIC (device ID `a2dc`) devices are supported for Spectrum-X configuration. Hardware Packet Load Balancing (`hwplb`) multiplane mode is only supported on ConnectX-8.

Install and Configure the NIC Configuration Operator

To install the operator and for more information on the CRDs, see NIC Firmware Configuration and Configuration Details.

Provision the DOCA SPC-X CC Algorithm Package

Note

For Spectrum-X RA2.1 and later, the DOCA SPC-X CC algorithm package is included in the operator image and does not need to be deployed separately. For RA2.0 and earlier, the package must be deployed manually using the example below.

To enable the DOCA SPC-X CC algorithm on NIC devices, the DOCA SPC-X CC .deb package for ubuntu 22.04 is required. This configuration step will be removed in the future, once the DOCA SPC-X CC algorithm is publicly available. To access the package, contact your NVIDIA CPM. The package should be available in the cluster and then its URL should be provided in the packageUrlSource field of the SpectrumXOperator CR.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareSource
metadata:
  name: spectrum-x-configuration
  namespace: nvidia-network-operator
spec:
  # should point to the URL of the DOCA SPC-X CC .deb package for
Ubuntu 22.04
  docaSpcXCCUrlSource: "https://example.com/doca-spcx-
cc_3.1.0105-1_amd64.deb"
```

Firmware Upgrade

If the firmware on the devices needs to be updated, extend the NicFirmwareSource CR with fields for ConnectX and BlueField firmware. Make sure to use the correct firmware for your devices.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareSource
metadata:
  name: spectrum-x-configuration
  namespace: nvidia-network-operator
spec:
  # should point to the URL of the DOCA SPC-X CC .deb package for
Ubuntu 22.04
  docaSpcXCCUrlSource: "https://example.com/doca-spcx-
cc_3.1.0105-1_amd64.deb"
  # 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-ConnectX8-
rel-40_46_3048-900-9X85E-00NX-MC0_Ax-UEFI-14.39.14-FlexBoot-
3.8.100.signed.bin.zip
  # a URL to the BlueField Bundle (BFB) file, can point to any
URL accessible from the cluster
  bfbUrlSource:
    - https://example.com/bf-fwbundle-3.1.0-77_25.07-prod.bfb
```

Configure and apply the NicFirmwareTemplate CR:

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicFirmwareTemplate
metadata:
  name: spectrum-x-configuration
  namespace: nvidia-network-operator
spec:
  nicSelector:
    nicType: "a2dc" # BlueField-3 SuperNIC, Can also be "1023"
for ConnectX-8
  template:
    nicFirmwareSourceRef: spectrum-x-configuration
    updatePolicy: Update
```

Enable SPC-X Optimizations for Devices

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
  name: spectrum-x-configuration
  namespace: nvidia-network-operator
spec:
  nodeSelector:
      feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
      nicType: a2dc # BlueField-3 SuperNIC, Can also be "1023"
for ConnectX-8
  template:
      numVfs: 1
      linkType: Ethernet
      spectrumXOptimized:
          enabled: true
          version: "RA2.0" # For Reference Architecture v1.3, use
"RA1.3" value for this field.
          overlay: "none" # For L3 overlay, use "l3" value for
this field.
```

RA2.1 configuration with multiplane support

Reference Architecture 2.1 introduces multiplane mode support, allowing NICs to be configured with multiple data planes for enhanced network performance.

> **Note**
>
> It is recommended to perform a NIC configuration reset before applying or switching between multiplane configurations to ensure a clean and consistent initial state. See Reset NIC Configuration to Default for details.

To enable multiplane support, set `spectrumXOptimized.version` to `RA2.1` and configure the `multiplaneMode` and `numberOfPlanes` fields.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicConfigurationTemplate
metadata:
  name: spectrum-x-multiplane-configuration
  namespace: nvidia-network-operator
spec:
  nodeSelector:
      feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
      nicType: "1023" # ConnectX-8. Use "a2dc" for BlueField-3
SuperNIC (hwplb not supported on BF3)
  template:
      numVfs: 1
      linkType: Ethernet
      spectrumXOptimized:
          enabled: true
          version: "RA2.1"
          overlay: "none"
          multiplaneMode: "hwplb" # Hardware Packet Load
Balancing, ConnectX-8 only
          numberOfPlanes: 4
```

Multiplane modes

The following multiplane modes are available with RA2.1:

Mode	Description	Supported NICs	Planes
`none`	Single plane mode (no multiplane). This is the default.	ConnectX-8, BF3 SuperNIC	1

`swplb`	Software Packet Load Balancing. The NIC port is split into multiple PFs, each assigned to a separate data plane.	ConnectX-8, BF3 SuperNIC	2, 4
`hwplb`	Hardware Packet Load Balancing. Uses hardware LAG resource allocation and NIC-level plane configuration for load balancing across planes.	ConnectX-8 only	2, 4
`uniplane`	Uniplane mode. Each port is configured as a separate plane without plane-level load balancing.	ConnectX-8, BF3 SuperNIC	2

Note

Multiplane modes (`swplb` , `hwplb` , `uniplane`) are only supported with RA2.1. For RA1.3 and RA2.0, `multiplaneMode` must be `none` and `numberOfPlanes` must be `1` .

NIC type constraints

NIC Type	Device ID	Supported Multiplane Modes
ConnectX-8	`1023`	`none` , `swplb` , `hwplb` , `uniplane`
BlueField-3 SuperNIC	`a2dc`	`none` , `swplb` , `uniplane`

> **⚠ Warning**
>
> The `hwplb` multiplane mode is only supported on ConnectX-8 (device ID `1023`). Attempting to configure `hwplb` on a BlueField-3 SuperNIC will be rejected by the API validation.

Configure custom interface names

The `NicInterfaceNameTemplate` CRD allows you to define custom naming patterns for RDMA and network device interfaces on Spectrum-X NICs. This is useful in multiplane and multi-rail deployments where predictable interface naming is required.

The operator deploys udev rules to the host to rename network and RDMA interfaces according to the specified naming template.

The template uses the following placeholders for device name construction:

- `%nic_id%`: The index of the NIC in the flattened list of NICs
- `%plane_id%`: The index of the plane of the specific NIC
- `%rail_id%`: The index of the rail where the given NIC belongs to

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicInterfaceNameTemplate
metadata:
  name: spectrum-x-interface-names
  namespace: nvidia-network-operator
spec:
  # Number of PFs per NIC, used to calculate the number of planes
per NIC
  pfsPerNic: 2
  # Template for RDMA device names. Placeholders: %nic_id%,
%plane_id%, %rail_id%
  rdmaDevicePrefix: "rdma_%nic_id%_%plane_id%_%rail_id%"
  # Template for net device names. Placeholders: %nic_id%,
%plane_id%, %rail_id%
  netDevicePrefix: "net_%nic_id%_%plane_id%_%rail_id%"
  # PCI address to rail mapping. First dimension is rail index,
second is NIC PCI addresses in the rail
  railPciAddresses:
    - ["0000:1a:00.0", "0000:2a:00.0"]
    - ["0000:3a:00.0", "0000:4a:00.0"]
```

The `railPciAddresses` field defines the PCI address to rail mapping. The first dimension is the rail index and the second dimension is the list of PCI addresses of the NICs in that rail.

Generated udev rules

The operator generates udev rules based on the template and writes them to the host. The rules are written to two separate files.

Example generated udev rules for net devices (`/etc/udev/rules.d/10-nic-net-interface-naming.rules`):

```
# Auto-generated by nic-configuration-operator
# Do not edit manually
SUBSYSTEM=="net", ACTION=="add", KERNELS=="0000:1a:00.0",
NAME="net_0_0_0"
SUBSYSTEM=="net", ACTION=="add", KERNELS=="0000:1a:00.1",
NAME="net_0_1_0"
SUBSYSTEM=="net", ACTION=="add", KERNELS=="0000:3a:00.0",
NAME="net_1_0_1"
SUBSYSTEM=="net", ACTION=="add", KERNELS=="0000:3a:00.1",
NAME="net_1_1_1"
```

Example generated udev rules for RDMA devices (`/etc/udev/rules.d/10-nic-rdma-interface-naming.rules`):

```
# Auto-generated by nic-configuration-operator
# Do not edit manually
ACTION=="add", KERNELS=="0000:1a:00.0", SUBSYSTEM=="infiniband",
RUN+="/usr/bin/rdma dev set %k name rdma_0_0_0"
ACTION=="add", KERNELS=="0000:1a:00.1", SUBSYSTEM=="infiniband",
RUN+="/usr/bin/rdma dev set %k name rdma_0_1_0"
ACTION=="add", KERNELS=="0000:3a:00.0", SUBSYSTEM=="infiniband",
RUN+="/usr/bin/rdma dev set %k name rdma_1_0_1"
ACTION=="add", KERNELS=="0000:3a:00.1", SUBSYSTEM=="infiniband",
RUN+="/usr/bin/rdma dev set %k name rdma_1_1_1"
```

Validation rules

The following validation rules are enforced by the API:

- Spectrum-X optimizations can only be enabled when `linkType` is `Ethernet` and `numVfs` is `1`.

- Spectrum-X optimizations can only be enabled for ConnectX-8 (`nicType: 1023`) or BlueField-3 SuperNIC (`nicType: a2dc`).
- When Spectrum-X optimizations are enabled, `roceOptimized` must not be enabled (RoCE settings are included in the Spectrum-X configuration).
- When Spectrum-X optimizations are enabled, `rawNvConfig` must be empty.
- When `multiplaneMode` is `none`, `numberOfPlanes` must be `1`.
- When `multiplaneMode` is not `none`, `numberOfPlanes` must not be `1`.
- When `version` is `RA1.3` or `RA2.0`, `multiplaneMode` must be `none` and `numberOfPlanes` must be `1`.
- The `hwplb` multiplane mode can only be enabled for ConnectX-8 (`nicType: 1023`).

Configuration Details

The Spectrum-X configuration parameters depend on the Reference Architecture version. The operator applies the following NVConfig and runtime parameters based on the selected version.

When spectrumXOptimized.enabled == true and spectrumXOptimized.version == “RA2.1” the following configuration parameters are applied:

```
  swplb:
    2:
      - name: Number of PFs
        value: 2
        dmsPath: /nvidia/physical-functions/config/num-of-pf
        valueType: int
      - name: Number of Planes
        value: 0
        mlxconfig: "NUM_OF_PLANES_P1"
      - name: LAG Resource Allocation
        value: 0
        mlxconfig: "LAG_RESOURCE_ALLOCATION"
      # CX8-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..3"
        alternativeValue: "[0,1,2,3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 2
        value: "4..7"
        alternativeValue: "[4,5,6,7]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 255
        value: "8..15"
        alternativeValue: "[8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "1023"
      # BF3-specific parameters
```

```
      - name: Lanes for Module 0 and Port 1
        value: "0..1"
        alternativeValue: "[0,1]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 2
        value: "2..3"
        alternativeValue: "[2,3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 255
        value: "4..15"
        alternativeValue: "[4,5,6,7,8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "a2dc"
    4:
      - name: Number of PFs
        value: 4
        dmsPath: /nvidia/physical-functions/config/num-of-pf
        valueType: int
      - name: Number of Planes
        value: 0
        mlxconfig: "NUM_OF_PLANES_P1"
      - name: LAG Resource Allocation
        value: 0
        mlxconfig: "LAG_RESOURCE_ALLOCATION"
      # CX8-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..1"
        alternativeValue: "[0,1]"
```

```
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 2
        value: "2..3"
        alternativeValue: "[2,3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 3
        value: "4..5"
        alternativeValue: "[4,5]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 4
        value: "6..7"
        alternativeValue: "[6,7]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 255
        value: "8..15"
        alternativeValue: "[8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "1023"
      # BF3-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..0"
        alternativeValue: "[0]"
```

```
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 2
        value: "1..1"
        alternativeValue: "[1]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 3
        value: "2..2"
        alternativeValue: "[2]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=3]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 4
        value: "3..3"
        alternativeValue: "[3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=4]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 255
        value: "4..15"
        alternativeValue: "[4,5,6,7,8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "a2dc"
  hwplb:
    2:
      - name: Number of PFs
        value: 2
```

```
        dmsPath: /nvidia/physical-functions/config/num-of-pf
        valueType: int
      - name: Number of Planes
        value: 2
        mlxconfig: "NUM_OF_PLANES_P1"
      - name: LAG Resource Allocation
        value: 1
        mlxconfig: "LAG_RESOURCE_ALLOCATION"
      # CX8-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..7"
        alternativeValue: "[0,1,2,3,4,5,6,7]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 255
        value: "8..15"
        alternativeValue: "[8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "1023"
    4:
      - name: Number of PFs
        value: 4
        dmsPath: /nvidia/physical-functions/config/num-of-pf
        valueType: int
      - name: Number of Planes
        value: 4
        mlxconfig: "NUM_OF_PLANES_P1"
      - name: LAG Resource Allocation
        value: 1
        mlxconfig: "LAG_RESOURCE_ALLOCATION"
      # CX8-specific parameters
      - name: Lanes for Module 0 and Port 1
```

```
        value: "0..7"
        alternativeValue: "[0,1,2,3,4,5,6,7]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 255
        value: "8..15"
        alternativeValue: "[8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "1023"
  uniplane:
    2:
      - name: Number of PFs
        value: 2
        dmsPath: /nvidia/physical-functions/config/num-of-pf
        valueType: int
      - name: Number of Planes P1
        value: 0
        mlxconfig: "NUM_OF_PLANES_P1"
      - name: Number of Planes P2
        value: 0
        mlxconfig: "NUM_OF_PLANES_P2"
      - name: LAG Resource Allocation
        value: 0
        mlxconfig: "LAG_RESOURCE_ALLOCATION"
      # CX8-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..3"
        alternativeValue: "[0,1,2,3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "1023"
```

```
      - name: Lanes for Module 0 and Port 2
        value: "4..7"
        alternativeValue: "[4,5,6,7]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "1023"
      - name: Lanes for Module 0 and Port 255
        value: "8..15"
        alternativeValue: "[8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "1023"
      # BF3-specific parameters
      - name: Lanes for Module 0 and Port 1
        value: "0..1"
        alternativeValue: "[0,1]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=1]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 2
        value: "2..3"
        alternativeValue: "[2,3]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=2]/lanes
        deviceId: "a2dc"
      - name: Lanes for Module 0 and Port 255
        value: "4..15"
        alternativeValue: "[4,5,6,7,8,9,10,11,12,13,14,15]"
        valueType: string
        dmsPath: /nvidia/device/config/module[module-
id=0]/port[port-id=255]/lanes
        deviceId: "a2dc"
```

```
nvConfig:
  - name: Ethernet mode
    value: 2
    mlxconfig: "LINK_TYPE_P1"
  - name: Enable SR-IOV
    value: 1
    mlxconfig: "SRIOV_EN"
  - name: Set NUM_OF_VFS
    value: 1
    mlxconfig: "NUM_OF_VFS"
  - name: NIC mode
    value: NIC
    dmsPath: /nvidia/mode/config/mode
    valueType: string
    deviceId: "a2dc"
  - name: RoCE Adaptive Routing
    value: true
    dmsPath: /nvidia/roce/config/adaptive-routing
    valueType: bool
  - name: Programmable Congestion Control
    value: true
    dmsPath: /nvidia/cc/config/user-programmable
    valueType: bool
  - name: RoCE TX Scheduling Locality Mode
    value: TX_SCHED_LOCALITY_ACCUMULATIVE
    dmsPath: /nvidia/roce/config/tx-sched-locality-mode
    valueType: string
  - name: RoCE CC Steering Ext
    value: ENABLED
    dmsPath: /nvidia/roce/config/cc-steering-ext
    valueType: string
  # swplb/uniplane/none-specific settings (via DMS)
  - name: CNP DSCP
    value: 0
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp
```

```
    valueType: int
    multiplane: swplb
  - name: CNP DSCP
    value: 0
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp
    valueType: int
    multiplane: uniplane
  - name: CNP DSCP
    value: 0
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp
    valueType: int
    multiplane: none
  - name: CNP DSCP mode
    value: RTT_RESP_DSCP_DEFAULT
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp-mode
    valueType: string
    multiplane: swplb
  - name: CNP DSCP mode
    value: RTT_RESP_DSCP_DEFAULT
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp-mode
    valueType: string
    multiplane: uniplane
  - name: CNP DSCP mode
    value: RTT_RESP_DSCP_DEFAULT
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp-mode
    valueType: string
    multiplane: none
  - name: RoCE Multipath DSCP
    value: MULTIPATH_DSCP_DEFAULT
    dmsPath: /nvidia/roce/config/multipath-dscp
    valueType: string
```

```
    alternativeValue: "unknown"
    ignoreError: true
    multiplane: swplb
  - name: RoCE Multipath DSCP
    value: MULTIPATH_DSCP_DEFAULT
    dmsPath: /nvidia/roce/config/multipath-dscp
    valueType: string
    alternativeValue: "unknown"
    ignoreError: true
    multiplane: uniplane
  - name: RoCE Multipath DSCP
    value: MULTIPATH_DSCP_DEFAULT
    dmsPath: /nvidia/roce/config/multipath-dscp
    valueType: string
    alternativeValue: "unknown"
    ignoreError: true
    multiplane: none
  # hwplb-specific settings (via mlxconfig)
  - name: CNP DSCP
    value: 48
    mlxconfig: "ROCE_RTT_RESP_DSCP_P1"
    multiplane: hwplb
  - name: CNP DSCP mode
    value: 1
    mlxconfig: "ROCE_RTT_RESP_DSCP_MODE_P1"
    multiplane: hwplb
  - name: Flex Parser Profile
    value: 10
    mlxconfig: "FLEX_PARSER_PROFILE_ENABLE"
    multiplane: hwplb
  - name: Disable RDE
    value: 1
    mlxconfig: "RDE_DISABLE"
    multiplane: hwplb
  - name: VF LOG BAR size
    value: 5
```

```
    mlxconfig: "VF_LOG_BAR_SIZE"
    multiplane: hwplb
runtimeConfig:
  roce:
    - name: Trust
      value: dscp
      dmsPath: /interfaces/interface/nvidia/qos/config/trust-mode
      valueType: string
      alternativeValue: QOS_TRUST_MODE_DSCP
    - name: PFC
      value: "00010000"
      dmsPath: /interfaces/interface/nvidia/qos/config/pfc
      valueType: string
    # TODO: figure out if NIC operator or RDMA cni needs to set
the tos
    # - name: Type of Service
    # value: 96
    # dmsPath: /interfaces/interface/nvidia/roce/config/tos
    # valueType: int
  adaptiveRouting:
    - name: Enable CC per plane
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/cc-per-
plane
      valueType: bool
      multiplane: hwplb
    - name: Adaptive Retransmission
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
retransmission
      valueType: bool
    - name: Tx Window
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/tx-window
      valueType: bool
    - name: Slow Restart
```

```
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart
      valueType: bool
    - name: Slow Restart Idle
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart-idle
      valueType: bool
    # Workaround: CC Probe MP mode is not configurable via DMS.
    # It is set via mlxreg in the nic-configuration-operator
instead.
    # Remove this workaround once the parameter is added to DMS.
    # - name: CC Probe MP mode
    # value: true
    # dmsPath: /interfaces/interface/nvidia/roce/config/cc-probe-
mp-mode
    # valueType: bool
    # multiplane: hwplb
    - name: Adaptive Routing Force
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
routing-force
      valueType: bool
  congestionControl:
    - name: Congestion Control on RP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/rp_enabled #
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name:  Congestion Control on NP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/np_enabled #
```

```
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name: Congestion Control
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/enabled
      valueType: bool
    - name: Congestion Control with Counters
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/counter_enable
      valueType: bool
    - name: DCQCN
      value: false
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=15]/config/enabled
      valueType: bool
    # Bandwidth configuration for different cases
    - name: Bandwidth
      value: 400
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
      deviceId: "1023"
      breakout: 2
    - name: Bandwidth
      value: 200
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
      deviceId: "1023"
      breakout: 4
    - name: Bandwidth
      value: 200
```

```
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
      deviceId: "a2dc"
      breakout: 2
    - name: Bandwidth
      value: 100
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
      deviceId: "a2dc"
      breakout: 4
    - name: Responsiveness Alpha Factor
      value: 6553
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=1]/config/value
      valueType: int
    - name: Maximum Decrease Factor
      value: 63570
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=2]/config/value
      valueType: int
    - name: Maximum Increase Factor
      value: 69468
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=3]/config/value
      valueType: int
    - name: Additive Increase Step Size
      value: 96
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=4]/config/value
      valueType: int
    - name: High Additive Increase Step Size
      value: 1700
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=5]/config/value
```

```
      valueType: int
    - name: High Additive Increase Interval Period
      value: 2000000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=6]/config/value
      valueType: int
    - name: ZTR_CC_CONGESTION_DELAY_THRESHOLD
      value: 13000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=7]/config/value
      valueType: int
    - name: Maximum Queuing Delay
      value: 250000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=8]/config/value
      valueType: int
    - name: Rate on First Congestion
      value: 524288
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=9]/config/value
      valueType: int
    - name: Delay Only
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=10]/config/valu
      valueType: int
    - name: CNP Validity
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=11]/config/valu
      valueType: int
    - name: Transmit Rate Decrement Step
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=12]/config/valu
      valueType: int
```

```
    - name: Fixed Transmission Rate
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=13]/config/valu
      valueType: int
    - name: Fast Scheduling Factor
      value: 2097152
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=14]/config/valu
      valueType: int
    - name: Topology Awareness
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=15]/config/valu
      valueType: int
    - name: Advanced Features
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=16]/config/valu
      valueType: int
    - name: Troubleshooting Capabilities
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=17]/config/valu
      valueType: int
    - name: CC_FIXED_CWND
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=18]/config/valu
      valueType: int
    - name: Enable CC Plane Failure Detection
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=19]/config/valu
      valueType: int
    - name: CC Plane Failure Threshold
```

```
      value: 3
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=20]/config/valu
      valueType: int
    - name: CC Plane Recovery Threshold
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=21]/config/valu
      valueType: int
  interPacketGap:
    pureL3:
      - name: Inter Packet Gap for no overlay
        value: 25
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
    l3EVPN:
      - name: Inter Packet Gap for L3 EVPN overlay
        value: 33
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
docaCCVersion: 3.3.0
useSoftwareCCAlgorithm: true
```

When spectrumXOptimized.enabled == true and spectrumXOptimized.version == “RA2.0” the following configuration parameters are applied:

```
  - name: Ethernet mode
    value: 2
    mlxconfig: "LINK_TYPE_P1"
  - name: Enable SR-IOV
    value: 1
    mlxconfig: "SRIOV_EN"
  - name: Set NUM_OF_VFS
    value: 1
    mlxconfig: "NUM_OF_VFS"
  - name: NIC mode
    value: NIC
    dmsPath: /nvidia/mode/config/mode
    valueType: string
    deviceId: "a2dc"
  - name: RoCE Adaptive Routing
    value: true
    dmsPath: /nvidia/roce/config/adaptive-routing
    valueType: bool
  - name: Programmable Congestion Control
    value: true
    dmsPath: /nvidia/cc/config/user-programmable
    valueType: bool
  - name: RoCE TX Scheduling Locality Mode
    value: TX_SCHED_LOCALITY_ACCUMULATIVE
    dmsPath: /nvidia/roce/config/tx-sched-locality-mode
    valueType: string
  - name: RoCE Multipath DSCP
    value: MULTIPATH_DSCP_DEFAULT
    dmsPath: /nvidia/roce/config/multipath-dscp
    valueType: string
  - name: CNP DSCP
    value: 0
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp
    valueType: int
```

```
  - name: CNP DSCP mode
    value: RTT_RESP_DSCP_DEFAULT
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp-mode
    valueType: string
  - name: RoCE CC Steering Ext
    value: ENABLED
    dmsPath: /nvidia/roce/config/cc-steering-ext
    valueType: string
runtimeConfig:
  roce:
    - name: Trust
      value: dscp
      dmsPath: /interfaces/interface/nvidia/qos/config/trust-mode
      valueType: string
      alternativeValue: QOS_TRUST_MODE_DSCP
    - name: PFC
      value: "00010000"
      dmsPath: /interfaces/interface/nvidia/qos/config/pfc
      valueType: string
    # TODO: figure out if NIC operator or RDMA cni needs to set
the tos
    # - name: Type of Service
    # value: 96
    # dmsPath: /interfaces/interface/nvidia/roce/config/tos
    # valueType: int
  adaptiveRouting:
    - name: Adaptive Retransmission
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
retransmission
      valueType: bool
    - name: Tx Window
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/tx-window
      valueType: bool
```

```
    - name: Slow Restart
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart
      valueType: bool
    - name: Slow Restart Idle
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart-idle
      valueType: bool
    - name: Adaptive Routing Force
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
routing-force
      valueType: bool
  congestionControl:
    - name: Congestion Control on RP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/rp_enabled #
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name:  Congestion Control on NP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/np_enabled #
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name: Congestion Control
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/enabled
      valueType: bool
    - name: Congestion Control with Counters
```

```
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/counter_enable
      valueType: bool
    - name: DCQCN
      value: false
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=15]/config/enabled
      valueType: bool
    - name: Bandwidth
      value: 400
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
    - name: Responsiveness Alpha Factor
      value: 6553
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=1]/config/value
      valueType: int
    - name: Maximum Decrease Factor
      value: 63570
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=2]/config/value
      valueType: int
    - name: Maximum Increase Factor
      value: 69468
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=3]/config/value
      valueType: int
    - name: Additive Increase Step Size
      value: 36
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=4]/config/value
      valueType: int
    - name: High Additive Increase Step Size
      value: 1200
```

```
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=5]/config/value
      valueType: int
    - name: High Additive Increase Interval Period
      value: 7000000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=6]/config/value
      valueType: int
    - name: Base Round Trip Time
      value: 15000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=7]/config/value
      valueType: int
    - name: Maximum Queuing Delay
      value: 250000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=8]/config/value
      valueType: int
    - name: Rate on First Congestion
      value: 524288
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=9]/config/value
      valueType: int
    - name: Delay Only
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=10]/config/valu
      valueType: int
    - name: CNP Validity
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=11]/config/valu
      valueType: int
    - name: Transmit Rate Decrement Step
      value: 0
```

```
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=12]/config/valu
      valueType: int
    - name: Fixed Transmission Rate
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=13]/config/valu
      valueType: int
    - name: Fast Scheduling Factor
      value: 2097152
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=14]/config/valu
      valueType: int
    - name: Topology Awareness
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=15]/config/valu
      valueType: int
    - name: Advanced Features
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=16]/config/valu
      valueType: int
    - name: Troubleshooting Capabilities
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=17]/config/valu
      valueType: int
  interPacketGap:
    pureL3:
      - name: Inter Packet Gap for no overlay
        value: 25
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
      - name: Shut down interface
```

```
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
      - name: Bring up interface to apply IPG settings
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
    l3EVPN:
      - name: Inter Packet Gap for L3 EVPN overlay
        value: 33
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
      - name: Shut down interface
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
      - name: Bring up interface to apply IPG settings
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
docaCCVersion: 3.1.0
useSoftwareCCAlgorithm: true
```

When spectrumXOptimized.enabled == true and spectrumXOptimized.version == “RA1.3” the following configuration parameters are applied:

```
  - name: Ethernet mode
    value: 2
    mlxconfig: "LINK_TYPE_P1"
  - name: Enable SR-IOV
    value: 1
    mlxconfig: "SRIOV_EN"
  - name: Set NUM_OF_VFS
    value: 1
    mlxconfig: "NUM_OF_VFS"
  - name: NIC mode
    value: NIC
    dmsPath: /nvidia/mode/config/mode
    valueType: string
    deviceId: "a2dc"
  - name: RoCE Adaptive Routing
    value: true
    dmsPath: /nvidia/roce/config/adaptive-routing
    valueType: bool
  - name: Programmable Congestion Control
    value: true
    dmsPath: /nvidia/cc/config/user-programmable
    valueType: bool
  - name: RoCE TX Scheduling Locality Mode
    value: TX_SCHED_LOCALITY_ACCUMULATIVE
    dmsPath: /nvidia/roce/config/tx-sched-locality-mode
    valueType: string
  - name: RoCE Multipath DSCP
    value: MULTIPATH_DSCP_DEFAULT
    dmsPath: /nvidia/roce/config/multipath-dscp
    valueType: string
  - name: CNP DSCP
    value: 0
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp
    valueType: int
```

```
  - name: CNP DSCP mode
    value: RTT_RESP_DSCP_DEFAULT
    dmsPath: /interfaces/interface/nvidia/roce/config/rtt-resp-
dscp-mode
    valueType: string
  - name: RoCE CC Steering Ext
    value: ENABLED
    dmsPath: /nvidia/roce/config/cc-steering-ext
    valueType: string
runtimeConfig:
  roce:
    - name: Trust
      value: dscp
      dmsPath: /interfaces/interface/nvidia/qos/config/trust-mode
      valueType: string
      alternativeValue: QOS_TRUST_MODE_DSCP
    - name: PFC
      value: "00010000"
      dmsPath: /interfaces/interface/nvidia/qos/config/pfc
      valueType: string
    # TODO: figure out if NIC operator or RDMA cni needs to set
the tos
    # - name: Type of Service
    # value: 96
    # dmsPath: /interfaces/interface/nvidia/roce/config/tos
    # valueType: int
  adaptiveRouting:
    - name: Adaptive Retransmission
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
retransmission
      valueType: bool
    - name: Tx Window
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/tx-window
      valueType: bool
```

```
    - name: Slow Restart
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart
      valueType: bool
    - name: Slow Restart Idle
      value: false
      dmsPath: /interfaces/interface/nvidia/roce/config/slow-
restart-idle
      valueType: bool
    - name: Adaptive Routing Force
      value: true
      dmsPath: /interfaces/interface/nvidia/roce/config/adaptive-
routing-force
      valueType: bool
  congestionControl:
    - name: Congestion Control on RP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/rp_enabled #
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name:  Congestion Control on NP points
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/config/priority/np_enabled #
priority[id=0..7]
      valueType: bool
      alternativeValue: "1"
    - name: Congestion Control
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/enabled
      valueType: bool
    - name: Congestion Control with Counters
```

```
      value: true
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/config/counter_enable
      valueType: bool
    - name: DCQCN
      value: false
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=15]/config/enabled
      valueType: bool
    - name: Bandwidth
      value: 400
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=0]/config/value
      valueType: int
    - name: Responsiveness Alpha Factor
      value: 6553
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=1]/config/value
      valueType: int
    - name: Maximum Decrease Factor
      value: 63570
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=2]/config/value
      valueType: int
    - name: Maximum Increase Factor
      value: 69468
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=3]/config/value
      valueType: int
    - name: Additive Increase Step Size
      value: 36
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=4]/config/value
      valueType: int
    - name: High Additive Increase Step Size
      value: 1200
```

```
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=5]/config/value
      valueType: int
    - name: High Additive Increase Interval Period
      value: 7000000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=6]/config/value
      valueType: int
    - name: Base Round Trip Time
      value: 15000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=7]/config/value
      valueType: int
    - name: Maximum Queuing Delay
      value: 250000
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=8]/config/value
      valueType: int
    - name: Rate on First Congestion
      value: 524288
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=9]/config/value
      valueType: int
    - name: Delay Only
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=10]/config/valu
      valueType: int
    - name: CNP Validity
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=11]/config/valu
      valueType: int
    - name: Transmit Rate Decrement Step
      value: 0
```

```
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=12]/config/valu
      valueType: int
    - name: Fixed Transmission Rate
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=13]/config/valu
      valueType: int
    - name: Fast Scheduling Factor
      value: 2097152
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=14]/config/valu
      valueType: int
    - name: Topology Awareness
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=15]/config/valu
      valueType: int
    - name: Advanced Features
      value: 1
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=16]/config/valu
      valueType: int
    - name: Troubleshooting Capabilities
      value: 0
      dmsPath:
/interfaces/interface/nvidia/cc/slot[id=0]/param[id=17]/config/valu
      valueType: int
  interPacketGap:
    pureL3:
      - name: Inter Packet Gap for no overlay
        value: 25
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
      - name: Shut down interface
```

```
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
      - name: Bring up interface to apply IPG settings
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
    l3EVPN:
      - name: Inter Packet Gap for L3 EVPN overlay
        value: 33
        dmsPath:
/interfaces/interface/ethernet/nvidia/config/inter-packet-gap
        valueType: int
      - name: Shut down interface
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
      - name: Bring up interface to apply IPG settings
        value: false
        dmsPath: /interfaces/interface/config/enabled
        valueType: bool
docaCCVersion: 3.1.0
useSoftwareCCAlgorithm: true
```

Network Operator API reference v1alpha1

Packages:

- configuration.net.nvidia.com/v1alpha1

configuration.net.nvidia.com/v1alpha1

Package v1alpha1 contains API Schema definitions for the configuration.net v1alpha1 API group

Resource Types:

ConfigurationTemplateSpec

(*Appears on:* NicConfigurationTemplateSpec, NicDeviceConfigurationSpec)

ConfigurationTemplateSpec is a set of configurations for the NICs

Field	Description
`numVfs` int	Number of VFs to be configured
`linkType` LinkTypeEnum	LinkType to be configured, Ethernet|Infiniband
`pciPerformanceOptimized` PciPerformanceOptimizedSpec	PCI performance optimization settings
`roceOptimized` RoceOptimizedSpec	RoCE optimization settings
`gpuDirectOptimized` GpuDirectOptimizedSpec	GPU Direct optimization settings
`spectrumXOptimized` SpectrumXOptimizedSpec	Spectrum-X optimization settings. Works only with linkType==Ethernet && numVfs==0. Other optimizations must be skipped or disabled. RawNvConfig must be empty.
`rawNvConfig` []NvConfigParam	List of arbitrary nv config parameters

FirmwareTemplateSpec

(*Appears on:* NicDeviceSpec, NicFirmwareTemplateSpec)

FirmwareTemplateSpec specifies a FW update policy for a given FW source ref

Field	Description

`nicFirmwareSourceRef` string	NicFirmwareSourceRef refers to existing NicFirmwareSource CR on where to get the FW from
`updatePolicy` string	UpdatePolicy indicates whether the operator needs to validate installed FW or upgrade it

GpuDirectOptimizedSpec

(*Appears on:* ConfigurationTemplateSpec)

GpuDirectOptimizedSpec specifies GPU Direct optimization settings

Field	Description
`enabled` bool	Optimize GPU Direct
`env` string	GPU direct environment, e.g. Baremetal

LinkTypeEnum (`string` alias)

(*Appears on:* ConfigurationTemplateSpec)

LinkTypeEnum described the link type (Ethernet / Infiniband)

NicConfigurationTemplate

NicConfigurationTemplate is the Schema for the nicconfigurationtemplates API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicConfigurationTemplateSpec	Defines the desired state of NICs
`status` NicTemplateStatus	Defines the observed state of NicConfigurationTemplate

NicConfigurationTemplateSpec

(*Appears on:* NicConfigurationTemplate)

NicConfigurationTemplateSpec defines the desired state of NicConfigurationTemplate

Field	Description
`nodeSelector` map[string]string	NodeSelector contains labels required on the node. When empty, the template will be applied to matching devices on all nodes.
`nicSelector` NicSelectorSpec	NIC selector configuration
`resetToDefault` bool	*(Optional)* ResetToDefault specifies whether node agent needs to perform a reset flow The following operations will be performed: * Nvconfig reset of all non-volatile configurations - Mstconfig -d reset for each PF - Mstconfig -d set ADVANCED_PCI_SETTINGS=1 * Node reboot - Applies new NIC NV config - Will undo any runtime configuration previously performed for the device/driver
`template` ConfigurationTemplateSpec	Configuration template to be applied to matching devices

NicDevice

NicDevice is the Schema for the nicdevices API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicDeviceSpec	
`status` NicDeviceStatus	

NicDeviceConfigurationSpec

(*Appears on:* NicDeviceSpec)

NicDeviceConfigurationSpec contains desired configuration of the NIC

Field	Description
`resetToDefault` bool	ResetToDefault specifies whether node agent needs to perform a reset flow. The following operations will be performed: * Nvconfig reset of all non-volatile configurations - Mstconfig -d reset for each PF - Mstconfig -d set ADVANCED_PCI_SETTINGS=1 * Node reboot - Applies new NIC NV config - Will undo any runtime configuration previously performed for the device/driver
`template` ConfigurationTemplateSpec	Configuration template applied from the NicConfigurationTemplate CR

NicDeviceInterfaceNameSpec

(*Appears on:* NicDeviceSpec)

Field	Description
`nicIndex` int	NicIndex is the index of the NIC in the flattened list of NICs based on the Template
`railIndex` int	RailIndex is the index of the rail where the given NIC belongs to based on the Template
`planeIndices` []int	PlaneIndices is the indices of the planes for the given NIC based on the Template

`rdmaDevicePrefix` string	— Parameters from the NicInterfaceNameTemplate CR — RdmaDevicePrefix specifies the prefix for the rdma device name
`netDevicePrefix` string	NetDevicePrefix specifies the prefix for the net device name

NicDevicePortSpec

(*Appears on:* NicDeviceStatus)

NicDevicePortSpec describes the ports of the NIC

Field	Description
`pci` string	PCI is a PCI address of the port, e.g. 0000:3b:00.0
`networkInterface` string	NetworkInterface is the name of the network interface for this port, e.g. eth1
`rdmaInterface` string	RdmaInterface is the name of the rdma interface for this port, e.g. mlx5_1

NicDeviceSpec

(*Appears on:* NicDevice)

NicDeviceSpec defines the desired state of NicDevice

Field	Description
`configuration` NicDeviceConfigurationSpec	Configuration specifies the configuration requested by NicConfigurationTemplate
`firmware` FirmwareTemplateSpec	Firmware specifies the fw upgrade policy requested by NicFirmwareTemplate
`interfaceNameTemplate` NicDeviceInterfaceNameSpec	InterfaceNameTemplate specifies the interface name template to be applied to the NIC

NicDeviceStatus

(*Appears on:* NicDevice)

NicDeviceStatus defines the observed state of NicDevice

Field	Description
`node` string	Node where the device is located
`type` string	Type of device, e.g. ConnectX7
`serialNumber` string	Serial number of the device, e.g. MT2116X09299
`partNumber` string	Part number of the device, e.g. MCX713106AEHEA_QP1
`psid` string	Product Serial ID of the device, e.g. MT_0000000221
`firmwareVersion` string	Firmware version currently installed on the device, e.g. 22.31.1014
`dpu` bool	DPU indicates if the device is a BlueField in DPU mode
`modelName` string	ModelName is the model name of the device, e.g. ConnectX-6 or BlueField-3
`superNIC` bool	SuperNIC indicates if the device is a SuperNIC
`ports` []NicDevicePortSpec	List of ports for the device
`conditions` []Kubernetes meta/v1.Condition	List of conditions observed for the device

NicFirmwareSource

NicFirmwareSource is the Schema for the nicfirmwaresources API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicFirmwareSourceSpec	
`status` NicFirmwareSourceStatus	

NicFirmwareSourceSpec

(*Appears on:* NicFirmwareSource)

NicFirmwareSourceSpec represents a list of url sources for FW

Field	Description
`binUrlSources` []string	*(Optional)* BinUrlSources represents a list of url sources for ConnectX Firmware
`bfbUrlSource` string	*(Optional)* BFBUrlSource represents a url source for BlueField Bundle
`docaSpcXCCUrlSource` string	*(Optional)* DocaSpcXCCUrlSource represents a url source for DOCA SPC-X CC .deb package for ubuntu 22.04 Will be removed in the future, once Doca SPC-X CC algorithm will be publicly available

NicFirmwareSourceStatus

(*Appears on:* NicFirmwareSource)

NicFirmwareSourceStatus represents the status of the FW from given sources, e.g. version available for PSIDs

Field	Description
`state` string	State represents the firmware processing state

`reason` string	Reason shows an error message if occurred
`binaryVersions` map[string][]string	Versions is a map of available FW binaries versions to PSIDs a PSID should have only a single FW version available for it
`bfbVersions` map[string]string	BFBVersions represents the FW versions available in the provided BFB bundle
`docaSpcXCCVersion` string	DocaSpcXCCVersion represents the FW versions available in the provided DOCA SPC-X CC .deb package for ubuntu 22.04

NicFirmwareTemplate

NicFirmwareTemplate is the Schema for the nicfirmwaretemplates API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicFirmwareTemplateSpec	
`status` NicTemplateStatus	

NicFirmwareTemplateSpec

(*Appears on:* NicFirmwareTemplate)

NicFirmwareTemplateSpec defines the FW templates and node/nic selectors for it

Field	Description
`nodeSelector` map[string]string	NodeSelector contains labels required on the node. When empty, the template will be applied to matching devices on all nodes.
`nicSelector` NicSelectorSpec	NIC selector configuration

`template` FirmwareTemplateSpec	Firmware update template

NicInterfaceNameTemplate

NicInterfaceNameTemplate is the Schema for the nicinterfacenametemplates API

Field	Description
`metadata` Kubernetes meta/v1.ObjectMeta	Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` NicInterfaceNameTemplateSpec	
`status` NicInterfaceNameTemplateStatus	

NicInterfaceNameTemplateSpec

(*Appears on:* NicInterfaceNameTemplate)

NicInterfaceNameTemplateSpec defines the desired state of NicInterfaceNameTemplate

Field	Description
`nodeSelector` map[string]string	NodeSelector contains labels required on the node. When empty, the template will be applied to matching devices on all nodes.
`pfsPerNic` int	PfsPerNic specifies the number of PFs per NIC Used to calculate the number of planes per NIC
`rdmaDevicePrefix` string	RdmaDevicePrefix specifies the prefix for the rdma device name %nic_id%, %plane_id% and %rail_id% placeholders can be used to construct the device name %nic_id% is the index of the NIC in the flattened list of NICs %plane_id% is the index of the plane of the specific NIC %rail_id% is the index of the rail where the given NIC belongs to

`netDevicePrefix` string	NetDevicePrefix specifies the prefix for the net device name %nic_id%, %plane_id% and %rail_id% placeholders can be used to construct the device name %nic_id% is the index of the NIC in the flattened list of NICs %plane_id% is the index of the plane of the specific NIC %rail_id% is the index of the rail where the given NIC belongs to
`railPciAddresses` []string	RailPciAddresses defines the PCI address to rail mapping and order The first dimension is the rail index, the second dimension is the PCI addresses of the NICs in the rail. The PCI addresses must be sorted in the order of the rails. Example: [[“0000:1a:00.0”, “0000:2a:00.0”], [“0000:3a:00.0”, “0000:4a:00.0”]] specifies 2 rails with 2 NICs each.

NicInterfaceNameTemplateStatus

(*Appears on:* NicInterfaceNameTemplate)

NicInterfaceNameTemplateStatus defines the observed state of NicInterfaceNameTemplate

NicSelectorSpec

(*Appears on:* NicConfigurationTemplateSpec, NicFirmwareTemplateSpec)

NicSelectorSpec is a desired configuration for NICs

Field	Description
`nicType` string	Type of the NIC to be selected, e.g. 101d,1015,a2d6 etc.
`pciAddresses` []string	Array of PCI addresses to be selected, e.g. “0000:03:00.0”
`serialNumbers` []string	Serial numbers of the NICs to be selected, e.g. MT2116X09299

NicTemplateStatus

(*Appears on:* NicConfigurationTemplate, NicFirmwareTemplate)

NicTemplateStatus defines the observed state of NicConfigurationTemplate and NicFirmwareTemplate

Field	Description
`nicDevices` []string	NicDevice CRs matching this configuration / firmware template

NvConfigParam

(*Appears on:* ConfigurationTemplateSpec)

Field	Description
`name` string	Name of the arbitrary nvconfig parameter
`value` string	Value of the arbitrary nvconfig parameter

PciPerformanceOptimizedSpec

(*Appears on:* ConfigurationTemplateSpec)

PciPerformanceOptimizedSpec specifies PCI performance optimization settings

Field	Description
`enabled` bool	Specifies whether to enable PCI performance optimization
`maxAccOutRead` int	Specifies the PCIe Max Accumulative Outstanding read bytes

`maxReadRequest` int	Specifies the size of a single PCI read request in bytes

QosSpec

(*Appears on:* RoceOptimizedSpec)

QosSpec specifies Quality of Service settings

Field	Description
`trust` string	Trust mode for QoS settings, e.g. trust-dscp
`pfc` string	Priority-based Flow Control configuration, e.g. “0,0,0,1,0,0,0,0”
`tos` int	8-bit value for type of service

RoceOptimizedSpec

(*Appears on:* ConfigurationTemplateSpec)

RoceOptimizedSpec specifies RoCE optimization settings

Field	Description
`enabled` bool	Optimize RoCE
`qos` QosSpec	Quality of Service settings

SpectrumXOptimizedSpec

(*Appears on:* ConfigurationTemplateSpec)

SpectrumXOptimizedSpec enables Spectrum-X specific optimizations

Field	Description
`enabled` bool	Optimize Spectrum X
`version` string	Version of the Spectrum-X architecture to optimize for
`overlay` string	*(Optional)* Overlay mode to be configured Can be “l3” or “none”
`multiplaneMode` string	*(Optional)* Multiplane mode to be configured Can be “none”, “swplb”, “hwplb”, or “uniplane”
`numberOfPlanes` int	*(Optional)* Number of planes to be configured

[TECH PREVIEW] Configuration Assistance with NVIDIA Kubernetes Launch Kit

NVIDIA Kubernetes Launch Kit (l8k) is a CLI tool for deploying and managing NVIDIA cloud-native solutions on Kubernetes. The tool helps provide flexible deployment workflows for optimal network performance with SR-IOV, RDMA, and other networking technologies.

Prerequisites

For prerequisites, please refer to the NVIDIA Network Operator Deployment Guide with Kubernetes page.

You will need a Kubernetes cluster with NVIDIA Network Operator helm chart installed.

Operation Phases

Discover Cluster Configuration

Deploy a minimal Network Operator profile to automatically discover your cluster's network capabilities and hardware configuration. This phase can be skipped if you provide your own configuration file.

Select the Deployment Profile

Specify the desired deployment profile via CLI flags or with the natural language prompt for the LLM.

Generate Deployment Files

Based on the discovered/provided configuration, generate a complete set of YAML deployment files tailored to your selected network profile.

Deploy to Cluster

Apply the generated deployment files to your Kubernetes cluster. This phase uses the `--deploy` flag and requires `--kubeconfig` to be specified. This phase is optional and can be skipped if `--deploy` is not provided.

Supported Deployment Profiles

Kubernetes Launch Kit supports the following deployment profiles:

Profile	Fabric	Deployment Type	Notes
SR-IOV Ethernet RDMA	ethernet	sriov	High-performance networking with hardware acceleration and dedicated VF resources.
SR-IOV InfiniBand RDMA	infiniband	sriov	Virtualized InfiniBand with hardware acceleration and isolated IB partitions.
IP over InfiniBand with RDMA Shared Device	infiniband	rdma_shared	InfiniBand networking with shared RDMA resources for parallel I/O workloads.

Host Device RDMA	any	host_device	Direct hardware access for legacy applications requiring exclusive device control.
MacVLAN with RDMA Shared Device	ethernet	rdma_shared	Network isolation with shared RDMA capabilities for multi-tenant environments.
Spectrum-X	ethernet	sriov	Spectrum-X networking with `--spectrum-x` flag. Supports `hwplb`, `swplb`, `uniplane`, and `none` multiplane modes.

Please refer to the Quick Start Guide for Kubernetes page for more details.

For Spectrum-X configuration, refer to the [TECH PREVIEW] NVIDIA Spectrum-X NIC Configuration page.

Heterogeneous Cluster Support

During cluster discovery, nodes are automatically grouped by their NIC configuration. Nodes with identical PCI addresses and device IDs (the same PF fingerprint) are placed in the same group. Each group receives:

- A unique `identifier` (empty string for single-group clusters, `group-0`, `group-1`, etc. for multi-group)
- An auto-computed `nodeSelector` based on labels that distinguish the group from others
- `machineType` and `productType` extracted from `nvidia.com/gpu.machine` and `nvidia.com/gpu.product` node labels

Templates that reference cluster configuration are rendered once per group, producing separate output files per group (e.g., `30-sriovnetworknodepolicy-group-0.yaml`, `30-sriovnetworknodepolicy-group-1.yaml`).

Use the `--group` flag to generate manifests for a single group:

```
l8k --user-config ./config.yaml \
    --fabric infiniband --deployment-type sriov --multirail \
    --group group-0 \
    --save-deployment-files ./deployments
```

North-South Traffic Detection

During cluster discovery, the tool automatically identifies BlueField DPU devices by matching each device's part number against a known list of DPU product codes. Devices matching a DPU product code are classified as **north-south** traffic (management/external), while all other devices (SuperNICs, ConnectX NICs) are classified as **east-west** traffic (GPU interconnect).

North-south PFs are included in the saved cluster configuration for visibility, but are automatically filtered out during template rendering so that only east-west PFs appear in the generated deployment manifests.

Each east-west PF is assigned a sequential rail number (`rail: 0`, `rail: 1`, `rail: 2`, ...) used for naming resources such as SriovNetworkNodePolicy and IPPool entries.

Usage

Kubernetes Launch Kit is available as a docker container:

```
mkdir ~/cluster-configuration
cp /etc/kubernetes/admin.conf ~/cluster-configuration/kubeconfig
docker run -v ~/cluster-configuration:/cluster-configuration \
    --net=host \
    nvcr.io/nvidia/cloud-native/k8s-launch-kit:v26.1.0 \
    --discover-cluster-config \
    --kubeconfig /cluster-configuration/kubeconfig \
    --save-cluster-config /cluster-configuration/config.yaml \
    --log-level debug \
    --save-deployment-files /cluster-configuration/deployments \
    --fabric infiniband --deployment-type rdma_shared --multirail
```

Note

You must enable `--net=host` and mount the necessary directories for input and output files with `-v`.

CLI Reference

General Flags

Flag	Description
`--enabled-plugins <string>`	Comma-separated list of plugins to enable (default: `network-operator`)
`--log-level <string>`	Log level: debug, info, warn, error
`--log-file <string>`	Write logs to file instead of stderr
`-h, --help`	Show help

Cluster Discovery Flags

Flag	Description
`--discover-cluster-config`	Deploy a thin Network Operator profile to discover cluster capabilities
`--save-cluster-config <string>`	Save discovered cluster configuration to the specified path (defaults to `--user-config` path if set, otherwise `/opt/nvidia/k8s-launch-kit/cluster-config.yaml`)
`--user-config <string>`	Use provided cluster configuration file (as base config for discovery or as full config without discovery)
`--kubeconfig <string>`	Path to kubeconfig file for cluster operations
`--label-selector <string>`	Filter nodes for discovery by label (default: `feature.node.kubernetes.io/pci-15b3.present=true`)
`--network-operator-namespace <string>`	Override the network operator namespace from the config file

Profile Selection Flags

Flag	Description
`--fabric <string>`	Select the fabric type to deploy (infiniband, ethernet)
`--deployment-type <string>`	Select the deployment type (sriov, rdma_shared, host_device)
`--multirail`	Enable multirail deployment
`--spectrum-x`	Enable Spectrum-X deployment
`--ai`	Enable AI deployment
`--group <string>`	Generate templates for a specific node group only (e.g., `group-0`). See Heterogeneous Cluster Support.

Note

If the configuration file provided via `--user-config` already contains a complete `profile:` section, the `--fabric`, `--deployment-type`, and other profile selection flags are not required. CLI flags override values from the configuration file when both are provided.

Spectrum-X Flags

Flag	Description
`--spcx-version <string>`	Spectrum-X version (default: `RA2.1`)
`--multiplane-mode <string>`	Multiplane mode: `swplb`, `hwplb`, `uniplane`, `none`
`--number-of-planes <int>`	Number of planes for multiplane deployment (required when mode is not `none`)

Note

When `--spectrum-x` is specified, `--fabric`, `--deployment-type`, and `--multirail` are automatically set to `ethernet`, `sriov`, and `true` respectively. The `--spcx-version` defaults to `RA2.1`.

LLM Flags

Flag	Description
`--prompt <string>`	Path to file with a prompt to use for LLM-assisted profile generation

`--llm-interactive`	Enable interactive chat mode with the LLM agent
`--llm-api-key <string>`	API key for the LLM API
`--llm-api-url <string>`	API URL for the LLM API
`--llm-vendor <string>`	Vendor of the LLM API (openai, openai-azure, anthropic, gemini). Default: `openai-azure`
`--llm-model <string>`	Custom model name for the LLM API

Deployment Flags

Flag	Description
`--save-deployment-files <string>`	Save generated deployment files to the specified directory (default: `/opt/nvidia/k8s-launch-kit/deployment`)
`--deploy`	Deploy the generated files to the Kubernetes cluster

Usage Examples

Complete Workflow

Discover cluster config, generate files, and deploy:

```
l8k --discover-cluster-config --save-cluster-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments \
    --deploy --kubeconfig ~/.kube/config
```

Discover Cluster Configuration

```
l8k --discover-cluster-config --save-cluster-config ./my-cluster-
config.yaml \
    --kubeconfig ~/.kube/config
```

Filter discovery to specific nodes using a label selector:

```
l8k --discover-cluster-config --save-cluster-config ./my-cluster-
config.yaml \
    --label-selector "feature.node.kubernetes.io/pci-
15b3.present=true" \
    --kubeconfig ~/.kube/config
```

Discovery with User-Provided Base Config

Use your own config file (with custom network operator version, subnets, etc.) as the base for discovery. Without `--save-cluster-config`, the file is rewritten in place with discovery results:

```
l8k --user-config ./my-config.yaml --discover-cluster-config \
    --kubeconfig ~/.kube/config
```

Save discovery results to a separate file instead:

```
l8k --user-config ./my-config.yaml --discover-cluster-config \
    --save-cluster-config ./discovered-config.yaml \
    --kubeconfig ~/.kube/config
```

Use Existing Configuration

Generate and deploy with pre-existing config:

```
l8k --user-config ./existing-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --deploy --kubeconfig ~/.kube/config
```

Generate Deployment Files

```
l8k --user-config ./config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments
```

Generate Deployment Files for a Specific Node Group

In heterogeneous clusters, discovery produces multiple node groups. Use `--group` to generate manifests for a single group:

```
l8k --user-config ./config.yaml \
    --fabric infiniband --deployment-type sriov --multirail \
    --group group-0 \
    --save-deployment-files ./deployments
```

Spectrum-X Deployment

The `--spectrum-x` flag automatically sets `--fabric` to `ethernet`, `--deployment-type` to `sriov`, and `--multirail` to `true`. You must specify `--multiplane-mode` to select the multiplane mode, and `--number-of-planes` when the mode is not `none`. The `--spcx-version` defaults to `RA2.1`.

For more information on Spectrum-X configuration, refer to the [TECH PREVIEW] NVIDIA Spectrum-X NIC Configuration page.

Note

NIC type constraints for multiplane modes:

- ConnectX-8 (device ID 1023): supports `swplb`, `hwplb`, and `uniplane` modes
- BlueField-3 SuperNIC (device ID a2dc): only supports `none` mode

HWPLB Mode

Hardware Plane Load Balancing for larger-scale clusters with 2-tier or 3-tier switch topologies.

```
l8k --user-config ./config.yaml --spectrum-x \
    --multiplane-mode hwplb --number-of-planes 4 \
    --save-deployment-files ./deployments
```

SWPLB Mode

Software Plane Load Balancing generates separate resources per-rail per-plane. Suitable for smaller-scale clusters.

```
l8k --user-config ./config.yaml --spectrum-x \
    --multiplane-mode swplb --number-of-planes 2 \
    --save-deployment-files ./deployments
```

Uniplane Mode

Unified plane mode with no plane separation. Simplest topology for ConnectX-8. Forces `--number-of-planes` to `1`.

```
l8k --user-config ./config.yaml --spectrum-x \
    --multiplane-mode uniplane \
    --save-deployment-files ./deployments
```

Single Plane (None)

No multiplane separation. Use with BlueField-3 SuperNIC or simple topologies.

```
l8k --user-config ./config.yaml --spectrum-x \
    --multiplane-mode none \
    --save-deployment-files ./deployments
```

LLM-Assisted Profile Selection

Kubernetes Launch Kit supports LLM-assisted profile generation. An AI agent analyzes your cluster configuration and requirements to recommend the optimal deployment profile.

Supported LLM vendors:

- `openai` — OpenAI API
- `openai-azure` — Azure OpenAI Service
- `anthropic` — Anthropic API
- `gemini` — Google Gemini API

Non-Interactive Mode

Provide a prompt file with `--prompt` for single-shot profile generation.

```
echo "I want to enable multirail networking in my AI cluster" >
requirements.txt
l8k --user-config ./config.yaml \
    --prompt requirements.txt --llm-vendor openai-azure \
    --llm-api-key <OPENAI_API_KEY> --llm-api-url <OPENAI_API_URL>
\
    --llm-model <OPENAI_MODEL> \
    --save-deployment-files ./deployments
```

Using Anthropic:

```
l8k --user-config ./config.yaml \
    --prompt requirements.txt --llm-vendor anthropic \
    --llm-api-key <ANTHROPIC_API_KEY> --llm-api-url
<ANTHROPIC_API_URL> \
    --llm-model <ANTHROPIC_MODEL> \
    --save-deployment-files ./deployments
```

Using Gemini:

```
l8k --user-config ./config.yaml \
    --prompt requirements.txt --llm-vendor gemini \
    --llm-api-key <GEMINI_API_KEY> --llm-api-url <GEMINI_API_URL>
\
    --llm-model <GEMINI_MODEL> \
    --save-deployment-files ./deployments
```

Interactive Chat Mode

Use `--llm-interactive` for a back-and-forth conversation with the AI agent. The agent will ask clarifying questions and help you select the optimal profile. Type

`generate` to confirm and generate manifests.

> **Note**
>
> `--prompt` and `--llm-interactive` cannot be used together.

```
l8k --user-config ./config.yaml \
    --llm-interactive --llm-vendor anthropic \
    --llm-api-key <KEY> --llm-api-url <API_URL> \
    --llm-model <LLM_MODEL> \
    --save-deployment-files ./deployments
```

HTML Overview Generation

An `overview.html` file is generated alongside the YAML deployment files. It contains the profile description, notes, collapsible file contents, and a link to the deployment guide. The path to the generated HTML file is automatically printed to the console.

Configuration File Format

After the cluster configuration is discovered, the tool will save the configuration to a file. You can use this file as a starting point for your own configuration. A custom configuration file can be provided to the tool using the `--user-config` flag — either as a standalone config (skipping discovery) or as a base config combined with `--discover-cluster-config` (discovery takes network operator parameters from the file and adds discovered cluster config).

```
 networkOperator:
   version: v26.1.0
   componentVersion: network-operator-v26.1.0
   repository: nvcr.io/nvidia/cloud-native
   namespace: nvidia-network-operator
   docsBaseURL:
https://docs.nvidia.com/networking/display/kubernetes2610
 docaDriver:
   version: doca3.3.0-26.01-1.0.0.0-0
   unloadStorageModules: false
   enableNFSRDMA: false
 nvIpam:
   poolName: nv-ipam-pool
   subnets:
   - subnet: 192.168.2.0/24
     gateway: 192.168.2.1
   - subnet: 192.168.3.0/24
     gateway: 192.168.3.1
   - subnet: 192.168.4.0/24
     gateway: 192.168.4.1
   - subnet: 192.168.5.0/24
     gateway: 192.168.5.1
 sriov:
   ethernetMtu: 9000
   infinibandMtu: 4000
   numVfs: 8
   priority: 90
   resourceName: sriov_resource
   networkName: sriov-network
 hostdev:
   resourceName: hostdev-resource
   networkName: hostdev-network
 rdmaShared:
   resourceName: rdma_shared_resource
   hcaMax: 63
```

```
 ipoib:
   networkName: ipoib-network
 macvlan:
   networkName: macvlan-network
 spectrumX:
   nicType: "1023"    # "1023" for ConnectX-8, "a2dc" for
BlueField-3 SuperNIC
   overlay: "none"
   rdmaPrefix: "roce_p%plane%_r%rail%"
   netdevPrefix: "eth_p%plane%_r%rail%"
 profile:
   fabric: ethernet   # infiniband, ethernet
   deployment: sriov  # rdma_shared, sriov, host_device
   multirail: false
   spectrumX:          # Spectrum-X configuration (set to null or
omit if not using Spectrum-X)
     spcxVersion: "RA2.1"        # CLI override: --spcx-version
     multiplaneMode: swplb       # CLI override: --multiplane-
mode (swplb, hwplb, uniplane, none)
     numberOfPlanes: 4           # CLI override: --number-of-
planes
   ai: false
 clusterConfig:
 - identifier: "group-0"
   machineType: "DGX-B200"
   productType: "NVIDIA-B200"
   labelSelector:
     feature.node.kubernetes.io/pci-15b3.present: "true"
   capabilities:
     nodes:
       sriov: true
       rdma: true
       ib: false
   workerNodes: ["worker-0", "worker-1"]
   nodeSelector:
     nvidia.com/gpu.machine: "DGX-B200"
```

```
    pfs:
    - deviceID: "1023"
      pciAddress: "0000:05:00.0"
      rdmaDevice: "mlx5_0"
      networkInterface: "net1"
      traffic: east-west
      rail: 0
    - deviceID: "1023"
      pciAddress: "0000:75:00.0"
      rdmaDevice: "mlx5_1"
      networkInterface: "net2"
      traffic: east-west
      rail: 1
    - deviceID: "1023"
      pciAddress: "0000:6a:00.0"
      rdmaDevice: "mlx5_4"
      networkInterface: "net5"
      traffic: north-south
  - identifier: "group-1"
    machineType: "PowerEdge-XE9680"
    productType: "NVIDIA-H100"
    labelSelector:
      feature.node.kubernetes.io/pci-15b3.present: "true"
    capabilities:
      nodes:
        sriov: true
        rdma: true
        ib: false
    workerNodes: ["worker-2", "worker-3"]
    nodeSelector:
      nvidia.com/gpu.machine: "PowerEdge-XE9680"
    pfs:
    - deviceID: "a2dc"
      pciAddress: "0000:1a:00.0"
      rdmaDevice: ""
      networkInterface: ""
```

```
    traffic: east-west
    rail: 0
  - deviceID: "a2dc"
    pciAddress: "0000:3c:00.0"
    rdmaDevice: ""
    networkInterface: ""
    traffic: east-west
    rail: 1
```

Note

The `clusterConfig` section is an array of node groups. In heterogeneous clusters, each group contains nodes with identical NIC configurations (same PCI addresses and device IDs). Each PF entry includes a `traffic` field (`east-west` or `north-south`) and a `rail` field (sequential index for east-west PFs). North-south PFs do not have a `rail` field. See Heterogeneous Cluster Support for details.

Customization Options

        - ResourceRequirements
        - SecondaryNetworkSpec
        - SpectrumXOperatorSpec
        - `string`
        - WaitForCompletionSpec
- NicClusterPolicy Custom Resource 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
- Maintenance Operator
- Helm customization file

General Parameters

Name	Type	
imagePullSecrets	list	*[]*
maintenanceOperator.enabled	bool	*false*

nfd.deployNodeFeatureRules	bool	*true*
nfd.enabled	bool	*true*
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 plane" operator: values: [</pre>

operator.cniBinDirectory	string	*"/opt/cni/bin"*
operator.cniNetworkDirectory	string	*"/etc/cni/net.d"*
operator.fullnameOverride	string	*""*
operator.image	string	*"network-operator"*
operator.maintenanceOperator	object	*{"drainControllerRequestorID":"nvi controller","nodeMaintenanceNam operator","nodeMaintenanceName operator-driver-upgrade-controller","useDrainControllerReq*
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	*"nvcr.io/nvidia/mellanox"*

operator.ofedDriver.initContainer.version	string	*“network-operator-v26.1.1”*
operator.repository	string	*“nvcr.io/nvidia/cloud-native”*
operator.resources	yaml	<pre>limits: cpu: 500m memory: 128Mi requests: cpu: 5m memory: 64Mi</pre>
operator.tolerations	yaml	<pre>- key: "node-role.kuber operator: "Equal" value: "" effect: "NoSchedule" - key: "node-role.kuber operator: "Equal" value: "" effect: "NoSchedule"</pre>
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.image.pullPolicy	string	*“IfNotPresent”*
node-feature-discovery.image.repository	string	*“nvcr.io/nvidia/mellanox/node-feature-discovery”*
node-feature-discovery.image.tag	string	*“network-operator-v26.1.1”*

node-feature-discovery.master	yaml	<pre>serviceAccount: name: node-feature-discovery create: true config: extraLabelNs: ["nvidia.com"]</pre>
node-feature-discovery.postDeleteCleanup	bool	*false*

node-feature-discovery.worker	yaml	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

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.rdmaCni	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

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
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	
maintenance-operator-chart.operatorConfig	object	
maintenance-operator-chart.operatorConfig.deploy	bool	

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)

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 inbox driver that comes with an OS. See https://network.nvidia.com/support/mlnx-ofed-matrix/
`rdmaSharedDevicePlugin` DevicePluginSpec	RdmaSharedDevicePlugin manages support IB and RoC Kubernetes device plugin framework. The config field is representation of the RDMA shared device plugin confi 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 device plugin configuration. See https://github.com/k8snetworkplumbingwg/sriov-netwo
`ibKubernetes` IBKubernetesSpec	IBKubernetes provides a daemon that works in conjunct Network Device Plugin. It acts on Kubernetes pod objec reads the pod's network annotation. From there it fetch corresponding network CRD and reads the PKey. This is add the newly generated GUID or the predefined GUID i the CRD. This is then passed in cni-args to that PKey fo mellanox.infiniband.app annotation. See: https://github.c kubernetes

`secondaryNetwork` SecondaryNetworkSpec	SecondaryNetwork Specifies components to deploy in c secondary network in Kubernetes. It consists of the foll deployed components: - Multus-CNI: Delegate CNI plugi secondary networks in Kubernetes - CNI plugins: Currer containernetworking-plugins is supported - IPoIB CNI: A create IPoIB child link and move it to the pod
`nvIpam` NVIPAMSpec	NvIpam is an IPAM provider that dynamically assigns IP speed and performance in mind. Note: NvIPam requires management e.g. cert-manager or OpenShift cert mana https://github.com/Mellanox/nvidia-k8s-ipam
`nicFeatureDiscovery` NICFeatureDiscoverySpec	NicFeatureDiscovery works with NodeFeatureDiscovery information about NVIDIA NICs. https://github.com/Mell discovery
`docaTelemetryService` DOCATelemetryServiceSpec	DOCATelemetryService exposes telemetry from NVIDIA components to prometheus. See: https://docs.nvidia.com/doca/sdk/doca+telemetry+servi
`nicConfigurationOperator` NicConfigurationOperatorSpec	NicConfigurationOperator provides Kubernetes CRD AP configuration on NVIDIA NICs in a coordinated manner S https://github.com/Mellanox/nic-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 by the operator
`tolerations` []Kubernetes core/v1.Toleration	Tolerations to inject to the DaemonSets objects that are operator
`deploymentNodeAffinity` Kubernetes core/v1.NodeAffinity	NodeAffinity rules to inject to the Deployments objects by the operator
`deploymentTolerations` []Kubernetes core/v1.Toleration	Tolerations to inject to the Deployments objects that ar operator

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
`env` []Kubernetes core/v1.EnvVar	List of environment variables to set in the NIC Configuration Operator and NIC Configuration Daemon containers.
`nicFirmwareStorage` NicFirmwareStorageSpec	NicFirmwareStorage contains configuration for the NIC firmware storage. If not provided, the NIC firmware storage will not be configured.
`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 with the name specified in pvcName 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-OFED Driver Container

Field	Description
`ImageSpec` ImageSpec	Image information for DOCA-OFED 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-OFED driver container.
`upgradePolicy` DriverUpgradePolicySpec	DOCA-OFED driver auto-upgrade settings

`certConfig` ConfigMapNameReference	Optional: Custom TLS certificates configuration for DOCA-OFED driver container
`repoConfig` ConfigMapNameReference	Optional: Custom package repository configuration for DOCA-OFED driver container
`terminationGracePeriodSeconds` int64	*(Optional)* TerminationGracePeriodSeconds specifies the length of time in seconds to wait before killing the DOCA-OFED driver container pod on termination
`forcePrecompiled` bool	*(Optional)* ForcePrecompiled specifies if only DOCA-OFED driver precompiled images are allowed If set to false and precompiled image does not exists, DOCA-OFED 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
`failureThreshold` int	Minimum consecutive failures for the probe to be considered failed after having succeeded
`timeoutSeconds` int	Number of seconds after which the probe times out

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

SpectrumXOperatorSpec

(*Appears on:* NicClusterPolicySpec)

SpectrumXOperatorSpec describes configuration options for NVIDIA Spectrum-X Operator

Field	Description
`ImageSpec` ImageSpec	Image information for NVIDIA Spectrum-X Operator

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: doca3.3.0-26.01-1.0.0.0-0
    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: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
    # 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: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
  config: |
    {
      "resourceList": [
        {
          "resourcePrefix": "nvidia.com",
          "resourceName": "hostdev",
          "selectors": {
            "vendors": ["15b3"],
            "isRdma": true
          }
        }
      ]
    }
 secondaryNetwork:
   cniPlugins:
     image: plugins
     repository: nvcr.io/nvidia/mellanox
     version: network-operator-v26.1.1
   ipoib:
     image: ipoib-cni
     repository: nvcr.io/nvidia/mellanox
```

```
    version: network-operator-v26.1.1
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
nvIpam:
  image: nvidia-k8s-ipam
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
  enableWebhook: false
ibKubernetes:
  image: ib-kubernetes
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
  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: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
docaTelemetryService:
  image: doca_telemetry
  repository: nvcr.io/nvidia/doca
  version: 1.23.4-doca3.2.0-host
spectrumXOperator:
  image: spectrum-x-operator
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.1.1
nicConfigurationOperator:
  operator:
    image: nic-configuration-operator
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
  configurationDaemon:
    image: nic-configuration-operator-daemon
```

```
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.1.1
  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 new major version of NVIDIA Network Operator is released, it becomes the supported release and the previous major version enters a deprecated (maintenance) state, receiving only patch updates for critical bug and CVE fixes. Once a subsequent major version is released, the deprecated version reaches End of Support (EOS) and no longer receives any updates.

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

> **Note**
>
> Upgrades are only supported within a major release or to the next major release.`

Support Status for Releases

Network Operator Version	Status	Comment
26.1.x	Supported	GA version (full support)

25.10.x	Deprecated	Maintenance version (critical bug and CVE fixes only)
25.7.x and lower	End of Support	Unsupported versions (no updates, including bug and CVE fixes)

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-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

Downloading a New Helm Chart

To obtain new releases, run:

```
 # Download Helm chart
 $ helm fetch \https://helm.ngc.nvidia.com/nvidia/charts/network-
operator-26.1.1.tgz
 $ ls network-operator-\*.tgz | xargs -n 1 tar xf
```

Applying the Helm Chart Update

Edit the *values-<VERSION>.yaml* file as required for your cluster.

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
```

Updating the NicClusterPolicy

Note

Helm upgrade does not update components version in the NicClusterPolicy. It should be done manually after the upgrade is done.

Note

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 the configuration for devicePlugin changed without image upgrade, manual restart of the devicePlugin may be required.

These limitations will be addressed in future releases.

Update the components version in the NicClusterPolicy. Refer to the NicClusterPolicy Custom Resource Example for more details and latest version of the components.

Automatic DOCA-OFED Driver Upgrade

To enable automatic DOCA-OFED Driver upgrade, define the UpgradePolicy section for the ofedDriver in the NicClusterPolicy spec, and change the DOCA-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    upgradePolicy:
      # autoUpgrade is a global switch for automatic upgrade
feature
      # if set to false all other options are ignored
      autoUpgrade: true
      # maxParallelUpgrades indicates how many nodes can be
upgraded in parallel
      # 0 means no limit, all nodes will be upgraded in parallel
      maxParallelUpgrades: 0
      # cordon and drain (if enabled) a node before loading the
driver on it
      safeLoad: false
      # describes the configuration for waiting on job
completions
      waitForCompletion:
        # specifies a label selector for the pods to wait for
completion
        podSelector: "app=myapp"
        # specify the length of time in seconds to wait before
giving up for workload to finish, zero means infinite
        # if not specified, the default is 300 seconds
        timeoutSeconds: 300
      # describes configuration for node drain during automatic
upgrade
      drain:
```

```
        # allow node draining during upgrade
        enable: true
        # allow force draining
        force: false
        # specify a label selector to filter pods on the node
that need to be drained
        podSelector: ""
        # specify the length of time in seconds to wait before
giving up drain, zero means infinite
        # if not specified, the default is 300 seconds
        timeoutSeconds: 300
        # specify if should continue even if there are pods using
emptyDir
        deleteEmptyDir: false
```

Apply NicClusterPolicy CR:

```
$ 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-OFED Driver POD is up-to-date and running on the node, the node is schedulable.
`upgrade-required`	Set when DOCA-OFED 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-OFED Driver POD on the node is scheduled for restart. After the restart, the state is changed to uncordon-required.
`uncordon-required`	Set when DOCA-OFED 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: doca3.3.0-26.01-1.0.0.0-0
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
      drain:
        enable: true
        force: false
        deleteEmptyDir: true
        podSelector: ""
```

Upgrade modes

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

When this feature is enabled, the initial DOCA-OFED 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-OFED 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-OFED Driver pod failed to start/ a new version of NVIDIA DOCA-OFED 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-OFED Driver version in the NicClusterPolicy to the previous version or to another working version.

DOCA-OFED Driver Manual Upgrade

Automatic DOCA-OFED Driver upgrade is the preferred method for upgrading the DOCA-OFED Driver. However, if you need to manually upgrade the DOCA-OFED Driver, you can follow the steps below.

Restarting Pods with a Containerized DOCA-OFED Driver

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

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

The Helm upgrade command will only upgrade the DaemonSet spec of the DOCA-OFED Driver to point to the new driver version. The DOCA-OFED Driver's DaemonSet will not automatically restart pods with the driver on the nodes, as it uses "OnDelete" updateStrategy. The old DOCA-OFED 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-OFED 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-OFED Driver pod.
3. Return the pods with a secondary network to the node.

When the DOCA-OFED Driver is ready, proceed with the same steps for other nodes.

Removing Pods with a Secondary Network from the Node

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

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

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

Restarting the DOCA-OFED Driver Pod

Find the DOCA-OFED Driver pod name for the node:

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

Example for Ubuntu 20.04:

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

Deleting the DOCA-OFED Driver Pod from the Node

To delete the DOCA-OFED 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-OFED Driver pods on all nodes at once.

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

Returning Pods with a Secondary Network to the Node

After the DOCA-OFED 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"
```

Network Operator Upgrade on OpenShift Container Platform

See instructions in the Network Operator Upgrade section.

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 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-OFED Driver container must download several OS packages prior to the driver installation.

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

Local Image Registry

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

Pulling and Pushing Container Images to a Local Registry

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

NicClusterPolicy supports use of image container digest in the *version* field.

For DOCA-OFED driver, the use of image container digest in the *version* field is supported only for precompiled images.

For non-precompiled DOCA-OFED driver images, the version field must be appended by the OS name and Architecture running on the worker node.

For example for DOCA-OFED driver version doca3.3.0-26.01-1.0.0.0-0, the tag for Ubuntu 24.04 with X86 architecture is “doca3.3.0-26.01-1.0.0.0-0-ubuntu24.04-amd64”. Available DOCA-OFED 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-OFED Driver in the NICClusterPolicy to list the available tags. Specifying pull secret is required in the NicClusterPolicy DOCA-OFED 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-OFED Driver in the NICClusterPolicy to list the available tags. This is required to verify the availability of precompiled DOCA-OFED 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-OFED Driver container deployed as part of the Network Operator requires certain packages to be available for the driver installation. In restricted internet access or air-gapped installations, users are required to create a local mirror repository for their OS distribution, and make the following packages available:

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

For RT kernels following packages should be available:

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

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

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

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

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

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

`custom-repo.list`:

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

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

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

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

`redhat.repo`:

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

`ubi.repo`:

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

Create the ConfigMap for Ubuntu:

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

Create the ConfigMap for RHCOS:

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

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

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

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

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

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

NVIDIA DOCA-OFED Driver Container

NVIDIA DOCA-OFED Driver Container Environment Variables

The following are special environment variables supported by the NVIDIA DOCA-OFED 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-OFED Driver modules: * ib_isert * nvme_rdma * nvmet_rdma * rpcrdma * xprtrdma * ib_srpt
ENABLE_NFSRDMA	“false”	Enable loading of NFS related storage modules from a NVIDIA DOCA-OFED Driver container
RESTORE_DRIVER_ON_POD_TERMINATION	“false”	Restore host drivers when a container

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

CREATE_IFNAMES_UDEV Environment Variable

> **⚠ 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).

ENABLE_NFSRDMA Environment Variable

In context of GPU Direct Storage (GDS) feature, only GDS with NFS over RDMA is supported. In this case, NVME over RDMA cannot be used. It is not possible to load inbox modules since they depend on *ib_core* which does not match (symbol error). Only NVME with local drive is supported.

Example of NicClusterPolicy

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: "false"
    - name: UNLOAD_STORAGE_MODULES
      value: "true"
    - name: CREATE_IFNAMES_UDEV
      value: "true"
```

Precompiled Container Build Instructions for NVIDIA DOCA-OFED 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/3.2.0/SOURCES/mlnx_ofed/MLNX_OFED_debian-25.10-1.2.8.0.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-OFED 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 *3.2.0*) you can find the corresponding MLNX_OFED version at the link: https://linux.mellanox.com/public/repo/doca/3.2.0/SOURCES/mlnx_ofed/ which is *25.10-1.2.8.0'*

Download Docker files and scripts:

```
git clone https://github.com/Mellanox/doca-driver-build.git
cd doca-driver-build
git checkout e03368b97b2a784e48ad62a11f2905a1f1a18100
```

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-OFED Driver sources to the specified path. It prepares the necessary files for the driver build process.

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

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

Common mandatory build parameters

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

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

2. *D_ARCH*: Compiled Architecture

3. *D_BASE_IMAGE*: Base container image (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: **doca<doca_version>-<driver_ver>-<container_ver>-<kernel_ver-os-arch>**. For example: doca2.9.1-24.10-1.1.4.0-0-5.15.0-25-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

NOTE: Since OCP 4.19, RHCOS is based on RHEL 9.6, therefore the tag should include the RHEL version instead of the OCP version.

```
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 doca2.9.1-24.10-1.1.4.0-0-5.14.0-427.22.1.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 doca2.9.1-24.10-1.1.4.0-0-5.15.0-25-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 doca2.9.1-24.10-1.1.4.0-0-5.14.21-150500.55.83-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	v26.1.1	sha256:de6dfa3b87bf66d3e2d4de2
nvcr.io/nvidia/mellanox	network-operator-init-container	network-operator-v26.1.1	sha256:1bb3d668df54d1b1d459b0
nvcr.io/nvidia/mellanox	k8s-rdma-shared-dev-plugin	network-operator-v26.1.1	sha256:3ae8700da2abbf7a5628cff6
nvcr.io/nvidia/mellanox	ib-kubernetes	network-operator-v26.1.1	sha256:8fb6f0a5abdd49664220dd4
nvcr.io/nvidia/mellanox	ipoib-cni	network-operator-v26.1.1	sha256:93e56c99cfe7aff10fbdcccc
nvcr.io/nvidia/mellanox	nvidia-k8s-ipam	network-operator-v26.1.1	sha256:ce16569571c0652487ad536
nvcr.io/nvidia/mellanox	nic-feature-discovery	network-operator-v26.1.1	sha256:9146591022028e490346fe6
nvcr.io/nvidia/doca	doca_telemetry	1.23.4-doca3.2.0-host	sha256:b628647c87715087888a68
nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-v26.1.1	sha256:d08a2392ea394f891c4864f
nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-v26.1.1	sha256:e29792364dc8ccc616d262c
nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-v26.1.1	sha256:275c334dba4d11e4d7ab74a

nvcr.io/nvidia/mellanox	sriov-network-device-plugin	network-operator-v26.1.1	sha256:540b05d020dd6cd464e830
nvcr.io/nvidia/mellanox	sriov-cni	network-operator-v26.1.1	sha256:691fe58d100b61f3bc1536a
nvcr.io/nvidia/mellanox	ib-sriov-cni	network-operator-v26.1.1	sha256:f37dc08c68e9baca9653338
nvcr.io/nvidia/mellanox	plugins	network-operator-v26.1.1	sha256:8ec35312756be0c6329adf8
nvcr.io/nvidia/mellanox	multus-cni	network-operator-v26.1.1	sha256:8167840a2a7f172694b2d2e
nvcr.io/nvidia/mellanox	ovs-cni-plugin	network-operator-v26.1.1	sha256:95e63fd7f77a71d02c2d699
nvcr.io/nvidia/mellanox	rdma-cni	network-operator-v26.1.1	sha256:a1b2b0de2bd0687a86fc49e
nvcr.io/nvidia/mellanox	nic-configuration-operator	network-operator-v26.1.1	sha256:efdf04d9d3f056c6880c1bb
nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon	network-operator-v26.1.1	sha256:862429c1cea7c0c9fa9d9ee
nvcr.io/nvidia/mellanox	maintenance-operator	network-operator-v26.1.1	sha256:65ad5f388e5e0b9af4844c8
nvcr.io/nvidia/mellanox	spectrum-x-operator	network-operator-v26.1.1	sha256:cd41564b6c1e5bba7c8160

DOCA-OFED Driver Container Images

Repository	Image Name	Version

nvcr.io/nvidia/mellanox	doca-driver	doca3.3.0-26.01-1.0.0.0-0

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

Ubuntu

Tags	Digest
doca3.3.0-26.01-1.0.0.0-0-5.15.0-170-generic-ubuntu22.04-amd64	sha256:1fa6419eb5a038aaa69733b4fcb7623dae5acde09bcfe4b84f1a19
doca3.3.0-26.01-1.0.0.0-0-5.15.0-170-generic-ubuntu22.04-arm64	sha256:bf4479de7f60c7c4d5db808ace2e0e2130e375dc9b0158fd989f4
doca3.3.0-26.01-1.0.0.0-0-6.14.0-1017-azure-ubuntu24.04-amd64	sha256:ada2baa666c3dbeccec342f8b13f8b576b806d422081d717aed3c
doca3.3.0-26.01-1.0.0.0-0-6.14.0-1017-azure-ubuntu24.04-arm64	sha256:fa6fcc19cec7f8d93f61e092a3891267cc76f83324e40a0899c4ef

doca3.3.0-26.01-1.0.0.0-0-6.17.0-1007-aws-ubuntu24.04-amd64	sha256:c0d7f186d97b40d437d5231b2021c6cdcc85bc31bedbff405d6d3
doca3.3.0-26.01-1.0.0.0-0-6.17.0-1007-aws-ubuntu24.04-arm64	sha256:8bd02bdc0b63903d701dbcbbe69beca115b9c5e45eb6794d4e64
doca3.3.0-26.01-1.0.0.0-0-6.17.0-1007-oracle-ubuntu24.04-amd64	sha256:d533f482954333888d09068a1715d3c3fb171df2a5d6c0a35c131
doca3.3.0-26.01-1.0.0.0-0-6.17.0-1007-oracle-ubuntu24.04-arm64	sha256:a9e21bf47aa86fc9a3cc280cc9f5523f9243f3bd06024c0d911244
doca3.3.0-26.01-1.0.0.0-0-6.17.0-1008-nvidia-ubuntu24.04-amd64	sha256:4d161686f5c8d936021517753c4cccf24b9cb9da655e3f05a58be
doca3.3.0-26.01-1.0.0.0-0-6.17.0-1008-nvidia-ubuntu24.04-arm64	sha256:12fd1ada02a888c782d7c4ecaa050e199064632 1a6c51aa62bbdb

doca3.3.0-26.01-1.0.0.0-0-6.8.0-100-generic-ubuntu24.04-amd64	sha256:3405071164a25e4ad72c4a49bf1b5ff71eee1239cbc053c19762f4
doca3.3.0-26.01-1.0.0.0-0-6.8.0-100-generic-ubuntu24.04-arm64	sha256:8d9c04d2c135d39f14db12c3b81cd66a1962ff37318f7f51d7366a
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1043-oracle-ubuntu22.04-amd64	sha256:9ea6a22fa950d438cc4bad5d696127c482db8078c0641e2eb6891
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1043-oracle-ubuntu22.04-arm64	sha256:e7b1460fb6fede194387feb017c95858feef40a1c00bed506f335a
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1044-azure-ubuntu22.04-amd64	sha256:4512cb56164c7c1fec5aa5e90d129316e8112da30aa02a1a0e7eb
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1044-azure-ubuntu22.04-arm64	sha256:3d5cc4db0979d56a53170aef9533c90b8d6e0bf67b77b95c709bb

doca3.3.0-26.01-1.0.0.0-0-6.8.0-1046-aws-ubuntu22.04-amd64	sha256:d38f03f5726789060f96f906f74757f3482e78a419e6b48ee0e6fc
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1046-aws-ubuntu22.04-arm64	sha256:358fc3032e679ffe2d013346c3515179be58717d8666516fe58a4
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1046-nvidia-ubuntu22.04-amd64	sha256:8831ff92d307923942c396f72153d38d24b39928e43bb61830c04
doca3.3.0-26.01-1.0.0.0-0-6.8.0-1046-nvidia-ubuntu22.04-arm64	sha256:b7a2932e3cc83c08d50b6c860c6d6db6c9669f5bb9dca08cdf4b4
doca3.3.0-26.01-1.0.0.0-0-ubuntu22.04-amd64	sha256:d21a60774040cfefcf834bec5dbfd4c1a1992ce8b5d9a9ff30a3392
doca3.3.0-26.01-1.0.0.0-0-ubuntu22.04-arm64	sha256:f7c3746b46348468ee8a46958a17bcbcb7a4dcd801199e9b5e57c

doca3.3.0-26.01-1.0.0.0-0-ubuntu24.04-amd64	sha256:ec3f771597a821c1ce2eb16e3e5be53df067d91429d22c1db34ee
doca3.3.0-26.01-1.0.0.0-0-ubuntu24.04-arm64	sha256:285d637e0d8f54c1da3891bb39748991b2bd1d43fb4de2a2bfb6c

RHCOS

Tags	Digest
doca3.3.0-26.01-1.0.0.0-0-rhcos4.17-amd64 doca3.3.0-26.01-1.0.0.0-0-rhcos4.18-amd64	sha256:ac7f091b20c9c7cd5af5bad8d5f78f8109b3f62bc748a846f2a377beb
doca3.3.0-26.01-1.0.0.0-0-rhcos4.17-arm64 doca3.3.0-26.01-1.0.0.0-0-rhcos4.18-arm64	sha256:8f49ba149d43e117f21ee753a65c4a735b43bb1c57f316bb08e2f0f6

RHEL

Tags	Digest

doca3.3.0-26.01-1.0.0.0-0-rhel10.0-amd64	sha256:10e0430eaca9ea1f8528f810641e6ef42901ba6e757e4ec2ba9854e9
doca3.3.0-26.01-1.0.0.0-0-rhel10.0-arm64	sha256:a8bc5110e3d5c32738446b53809deb220dc8d71b37f5bd11c87e5af
doca3.3.0-26.01-1.0.0.0-0-rhel8.10-amd64	sha256:5999eb7175e0804fe9b48a9156932e4a657aecaa9c4315591abcc29c
doca3.3.0-26.01-1.0.0.0-0-rhel8.10-arm64	sha256:04e8293e45d7d1f518952be3f6a902d53b5cd6ae14c684dd36f4acbb
doca3.3.0-26.01-1.0.0.0-0-rhel9.4-amd64 doca3.3.0-26.01-1.0.0.0-0-rhel9.6-amd64	sha256:3bfd0a24859c3e8c0e9697d6f128855e2ab6b3731eb79ef5603673fc

doca3.3.0-26.01-1.0.0.0-0-rhel9.4-arm64 doca3.3.0-26.01-1.0.0.0-0-rhel9.6-arm64	sha256:3a6629d09fdc431594d399e1a5106920ed5b9fc42cecd501fd69664b

SLES

Tags	Digest
doca3.3.0-26.01-1.0.0.0-0-sles15.7-amd64	sha256:990f81022eb0fd213d823ecedb821f83145a9db7e708b365d3f06325
doca3.3.0-26.01-1.0.0.0-0-sles15.7-arm64	sha256:633ec9b617770dab3eb779a167262a94326e2005aded41edc0ca706

STIG FIPS Compliant DOCA-OFED Driver Container Images

Repository	Image Name	Version
nvcr.io/nvidia/mellanox	doca-driver-stig-fips	doca3.3.0-26.01-1.0.0.0-4

The followings tags are available for the above STIG FIPS Compliant DOCA-OFED Driver container version:

Ubuntu

Tags	Digest

doca3.3.0-26.01-1.0.0.0-4-ubuntu24.04-amd64	sha256:44c3f0e17a2f8a476e73fd065436bb68c10af20adf5262e4591cbe

RHEL

Tags	Digest
doca3.3.0-26.01-1.0.0.0-4-rhel9.6-amd64	sha256:a56683db77e8d019aade4d5d97279efb9e249f3ce7d3b4b23fbbdcbc

Troubleshooting

SOS-Report Collection Script

Overview

The Network Operator SOS-report script collects comprehensive diagnostic data from a Kubernetes cluster running the NVIDIA Network Operator. It gathers all relevant configuration, logs, status information, and diagnostic output into a single archive, making it easier to troubleshoot issues and share context with support teams.

The script is fully backward compatible and is designed to work with any version of the Network Operator, including all previous releases. Components or resources that are not present in a given release are gracefully skipped without errors.

The script and full README are available on GitHub at scripts/sosreport.

Installation

As a kubectl Plugin (Recommended)

Copy `kubectl-netop_sosreport`, `generate-report.py`, and `report-template.html` to a directory in your `PATH`:

```
# Install system-wide
sudo cp kubectl-netop_sosreport generate-report.py report-
template.html /usr/local/bin/
# Or install for the current user only
mkdir -p ~/.local/bin
cp kubectl-netop_sosreport generate-report.py report-
template.html ~/.local/bin/
export PATH="$HOME/.local/bin:$PATH"
```

Once installed, the script is available as a kubectl subcommand:

```
kubectl netop-sosreport [OPTIONS]
```

Note

`generate-report.py` and `report-template.html` must be in the same directory as `kubectl-netop_sosreport` for HTML report generation. If these files are not present locally, the script will attempt to download them from GitHub. If download fails, the collection still works but the HTML report is skipped.

As a Standalone Script

Run the script directly from the repository:

```
./kubectl-netop_sosreport [OPTIONS]
# Or use the backward-compatible symlink
./network-operator-sosreport.sh [OPTIONS]
```

Requirements

- `kubectl` binary installed and in `PATH`
- Valid kubeconfig with cluster access
- Permissions to read cluster resources (`cluster-admin` recommended)
- Bash 4.0 or later
- Python 3.6+ (for HTML report generation)
- Standard Unix utilities (`tar`, `gzip`, `sha256sum`)

Usage

Basic Usage

```
# Run with auto-detection (recommended)
./network-operator-sosreport.sh
# Specify kubeconfig explicitly
./network-operator-sosreport.sh --kubeconfig /path/to/kubeconfig
# Specify operator namespace
./network-operator-sosreport.sh --namespace nvidia-network-
operator
```

The script automatically detects the Network Operator namespace and the cluster platform (Kubernetes or OpenShift).

Command-Line Options

Option	Description
`--kubeconfig PATH`	Path to kubeconfig file. Default: `$KUBECONFIG` or `~/.kube/config`.
`--namespace NAMESPACE`	Network Operator namespace. Default: auto-detect.
`--output-dir PATH`	Output directory. Default: `./network-operator-sosreport-<timestamp>` .
`--no-compress`	Do not create a tarball; leave output as a directory.
`--log-lines N`	Number of log lines to collect per pod. Default: `5000`.
`--skip-diagnostics`	Skip running diagnostic commands in OFED pods (`lsmod` , `ibstat` , `ibv_devinfo` , `mst status` , `dmesg` , `ip` commands). Use this for faster collection when driver-level diagnostics are not needed.
`--skip-report`	Skip HTML report generation.

`--kubectl-path PATH`	Path to the `kubectl` binary. Default: `kubectl` from `PATH`.
`--verbose`	Enable verbose output during collection.
`--help`	Show the help message.

What's Collected

Custom Resources

All custom resources managed by the Network Operator are collected, including their definitions and instances. For a full reference of available CRDs, see Customization Options and CRDs.

Operator Resources

- Deployment, Pods, ConfigMaps
- Secrets (metadata only, no secret data)
- RBAC resources (ServiceAccounts, Roles, RoleBindings)
- Events in the operator namespace
- Webhook configurations (validating and mutating)

Components

The script collects data from all Network Operator components. Components that are not deployed in the cluster are automatically skipped. For the full list of components, see Network Operator Component Matrix.

For each component, the script collects:

- DaemonSet or Deployment specifications
- All pod details and status
- Current and previous container logs (if the container has restarted)
- Related ConfigMaps and Services

Node Information

- All node details with labels and annotations
- Node conditions and status
- Allocatable resources (RDMA, SR-IOV, GPU)
- Node-specific feature discovery labels

Diagnostic Commands

The following commands are executed inside OFED driver pods on each node:

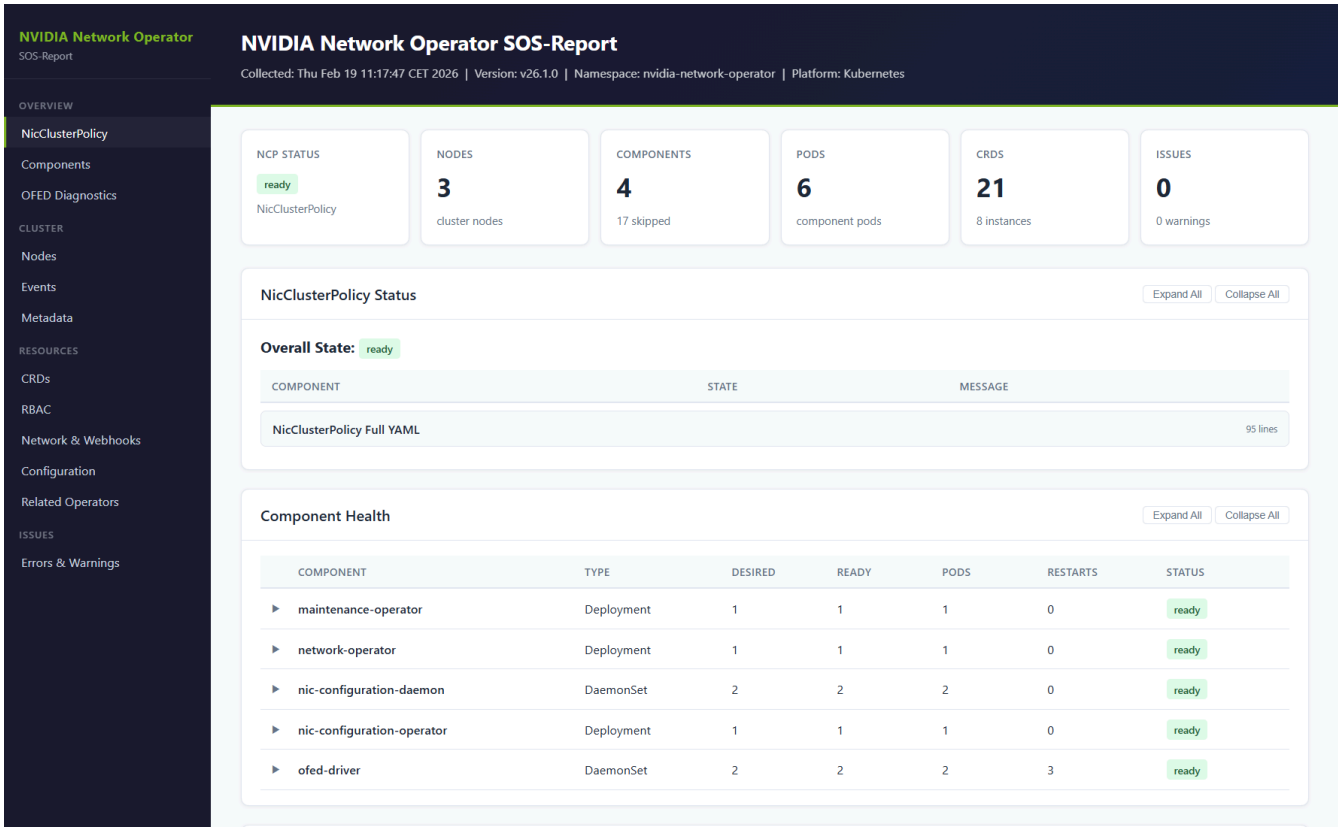
- `lsmod | grep mlx` — loaded Mellanox kernel modules
- `ibstat` — InfiniBand device status
- `ibv_devinfo` — RDMA device information
- `mst status` — Mellanox Software Tools status
- `uname -r` — kernel version
- `dmesg` (last 200 lines) — recent kernel messages
- `ip link` / `ip addr` — network interface information

> **Note**
>
> Use `--skip-diagnostics` to skip these commands for faster collection when driver-level diagnostics are not needed.

HTML Report

The collection script automatically generates a self-contained HTML report (`report.html`) that provides an interactive, navigable view of all collected data. The

report is included in the output archive alongside the raw files.



The report includes the following sections:

- Executive dashboard with overall NicClusterPolicy status, node count, pod health, and error summary
- NicClusterPolicy applied states with color-coded status badges
- Component health grid showing all components with desired/ready replicas, pod counts, and restart counts
- Per-component detail panels with workload YAML, pod status, and log viewers with error/warning highlighting
- OFED diagnostics per node
- Node overview with summary table, resource allocation, and labels
- Events timeline with warning highlighting
- CRD inventory with definitions and instances

- RBAC overview, network configuration, and webhook configuration
- Collection errors and warnings

The report can also be generated standalone from an existing sosreport directory:

```
python3 generate-report.py ./network-operator-sosreport-20260218-
143000/ --template report-template.html
# Custom output path
python3 generate-report.py ./network-operator-sosreport-20260218-
143000/ --template report-template.html --output /tmp/report.html
```

To skip report generation during collection, use the `--skip-report` flag.

Output Structure

The script creates a timestamped directory with the following structure:

```
network-operator-sosreport-<timestamp>/
    metadata/
        collection-info.txt
        cluster-version.yaml
        namespaces.txt
        api-resources.txt
    crds/
        definitions/
        instances/
    operator/
        namespace.yaml
        configmaps.yaml
        secrets-metadata.txt
        rbac/
        events.yaml
        validatingwebhookconfigurations.yaml
        mutatingwebhookconfigurations.yaml
        components/
            network-operator/
            ofed-driver/
                daemonset.yaml
                pods/
                diagnostics/
            ...
    nodes/
        all-nodes.yaml
        nodes-summary.txt
        node-labels.txt
        node-resources.txt
    network/
        services.yaml
    related-operators/
    diagnostic-summary.txt
    report.html
```

```
    collection-errors.log
```

By default, the output is compressed into a tarball with a SHA256 checksum:

- `network-operator-sosreport-<timestamp>.tar.gz`
- `network-operator-sosreport-<timestamp>.tar.gz.sha256`

Exit Codes

Code	Meaning
`0`	Success — all data collected.
`1`	Critical error — `kubectl` not found or no cluster access.
`2`	Partial success — some resources failed to collect.

Security Considerations

- **Secrets**: only metadata (names and types) is collected. Secret data is never included.
- **Logs**: may contain IP addresses, hostnames, and other environment-specific information.
- **Review**: always review the collected data before sharing it externally.

Example Workflows

Troubleshooting Pod Failures

```
# Collect full diagnostics with verbose output
./network-operator-sosreport.sh --verbose
# Extract and check the diagnostic summary
tar -xzf network-operator-sosreport-*.tar.gz
cat network-operator-sosreport-*/diagnostic-summary.txt
# Look at specific component logs
cat network-operator-sosreport-*/operator/components/ofed-
driver/pods/*.log
```

Quick Health Check

```
# Fast collection without driver diagnostics
./network-operator-sosreport.sh --skip-diagnostics --log-lines
1000
# Extract and check summary
tar -xzf network-operator-sosreport-*.tar.gz
less network-operator-sosreport-*/diagnostic-summary.txt
```

Preparing for a Support Case

```
# Comprehensive collection with verbose output
./network-operator-sosreport.sh --verbose --log-lines 10000
# Verify the archive integrity
sha256sum -c network-operator-sosreport-*.tar.gz.sha256
# The archive is ready to attach to a support case
```

Legal Notices and 3rd Party Licenses

Component	Version	Legal Notices and 3rd Party Licenses
NVIDIA Network Operator	v26.1.1	- License - 3rd Part Notice
NVIDIA Network Operator Init Container	network-operator-v26.1.1	- License - 3rd Part Notice
RDMA Shared Device Plugin	network-operator-v26.1.1	- License - 3rd Part Notice
IB Kubernetes Plugin	network-operator-v26.1.1	- License - 3rd Part Notice
IP Over Infiniband (IPoIB) CNI plugin	network-operator-v26.1.1	- License - 3rd Part Notice
NVIDIA IPAM Plugin	network-operator-v26.1.1	- License - 3rd Part Notice
NVIDIA NIC Feature Discovery	network-operator-v26.1.1	- License - 3rd Part Notice
Node Feature Discovery	network-operator-v26.1.1	- License - 3rd Part Notice
SRIOV Network Operator	network-operator-v26.1.1	- License - 3rd Part Notice

SR-IOV Network Device Plugin	network-operator-v26.1.1	- License - 3rd Part Notice
SR-IOV CNI plugin	network-operator-v26.1.1	- License - 3rd Part Notice
InfiniBand SR-IOV CNI plugin	network-operator-v26.1.1	- License - 3rd Part Notice
Multus CNI	network-operator-v26.1.1	- License - 3rd Part Notice
RDMA CNI plugin	network-operator-v26.1.1	- License - 3rd Part Notice
Open vSwitch CNI plugin	network-operator-v26.1.1	- License - 3rd Part Notice
NVIDIA NIC Configuration Operator	network-operator-v26.1.1	- License - 3rd Part Notice
NVIDIA Maintenance Operator	network-operator-v26.1.1	- License - 3rd Part Notice
NVIDIA SpectrumX Operator	network-operator-v26.1.1	- 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.