

NVIDIA Network Operator v26.4.1

Table of contents

Release Notes 7

Platform Support 24

Getting Started with Kubernetes 38

- Quick Start Guide for Kubernetes 38
- NVIDIA Network Operator Deployment Guide with Kubernetes 41
- KubeVirt SR-IOV Integration 129
- DRA SR-IOV Driver 140

Getting Started with OpenShift 154

- NVIDIA Network Operator Deployment Guide with OpenShift 154
- NVIDIA Network Operator Deployment on Disconnected OpenShift 177

NVIDIA Network Operator Government Ready 199

NIC Configuration Operator 210

- NIC Firmware Configuration 210
- Configuration Details 231
- Network Operator API reference v1alpha1 235

NVIDIA Spectrum-X Ethernet Networking Platform 249

- Spectrum-X Kubernetes Quick Start 252
  - Single-Plane Spectrum-X Quick Start 253
  - Software Multiplane Spectrum-X Quick Start 262
  - Hardware Multiplane Spectrum-X Quick Start (Tech Preview) 273
- Spectrum-X Kubernetes Architecture and Components 281
- NVIDIA Spectrum-X NIC Configuration 288

Spectrum-X CRDs and API Reference 301

NVIDIA Kubernetes Launch Kit 305

Overview 306

Installation 311

Quick Start 315

Deployment Profiles 317

Workflows 319

Discover Workflow 320

Generate Workflow 326

Deploy Workflow 333

Validate Workflow 337

Advanced Topics 345

Heterogeneous Clusters 345

Cluster Topology Presets 355

Automation and CI/CD 358

Troubleshooting 360

Reference 362

CLI Reference 363

Configuration Reference 371

AI Skills 381

Customization Options 390

Helm Chart Customization Options 391

Network Operator API reference v1alpha1 235

NicClusterPolicy Custom Resource Example 423

Heterogeneous Clusters with NicNodePolicy 427

Life Cycle Management 439

Advanced Configurations 457

Proxy & Air-Gapped Environments 458

NVIDIA DOCA-OFED Driver Container 468

Advanced Configurations 457

NVIDIA Network Operator Container Images 479

Troubleshooting 360

SOS-Report Collection Script 487

Legal Notices and 3rd Party Licenses 498

Enabling AI Networking in Kubernetes.

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.

Note

What's new in 26.4.0

- **KubeVirt SR-IOV passthrough** — accelerated East-West networking for VM workloads on vanilla Kubernetes. See KubeVirt SR-IOV Integration.
- **Spectrum-X RA 2.2 alignment** — multi-plane fabric with topology-aware pod placement via DRA, plus simplified config generation through the K8s LaunchKit.
- **Dynamic Resource Allocation (DRA) SR-IOV driver** — Kubernetes-native VF allocation for SR-IOV workloads. See DRA SR-IOV Driver.

For the full list of changes, see the Release Notes.

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 on NVIDIA

Quantum-X fabrics and RDMA over Converged Ethernet (RoCE) on NVIDIA Spectrum-X fabrics.

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.

Heterogeneous Cluster Support

Per-node-group NIC driver and device plugin configurations via NicNodePolicy, enabling different DOCA-OFED versions for different node roles.

Supported Hardware

See the Platform Support page for supported hardware and software.

Use Cases

AI Training at Scale: Distributed training across multi-GPU, multi-node clusters, with GPUDirect RDMA between GPUs and NICs for low-latency East-West traffic.

AI Inference and Generative AI: Multi-node serving of large language and multimodal models, where tensor and pipeline parallelism depend on accelerated GPU-to-GPU networking.

Telco CNFs and DPDK Workloads: Cloud-native network functions and DPDK-accelerated user-plane applications using SR-IOV for line-rate packet processing.

Media and Broadcast: Containerized live-broadcast pipelines on Kubernetes and OpenShift using NVIDIA Rivermax and uncompressed SMPTE ST 2110 over SR-IOV, with PTP-synchronized timing. Foundational to NVIDIA Holoscan for Media.

Edge AI and Sensor Streaming: Low-latency sensor ingest over SR-IOV and RDMA for real-time inference on NVIDIA IGX Orin and Grace ARM platforms.

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

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

License Agreements

The NVIDIA Network Operator source code 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.

NVIDIA Network Operator container images distributed via NVIDIA NGC are subject to the NVIDIA software license terms. Refer to the corresponding container listing on NGC for the applicable license; by pulling and using these containers you accept those terms.

Learn More

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

For detailed deployment, operational, and release information:

- Release Notes: What changed in this release, known issues, and upgrade considerations
- Platform Support: Supported hardware, operating systems, and Kubernetes platforms
- Quick Start Guide for Kubernetes: Quick deployment guide with common configurations
- NVIDIA Network Operator Deployment Guide with Kubernetes: Detailed deployment scenarios
- Life Cycle Management: Upgrade procedures and lifecycle guidance
- Troubleshooting: Troubleshooting common issues

Release Notes

Changes and New Features

Version	Description
26.4.1	- CVEs fixes - Added support for OpenShift 4.22 - Added support for Kubernetes 1.36 - Added support for RHEL 9.8 - Added support for RHEL 10.2

26.4.0	- Added support for Kubernetes 1.35 - Added support for Red Hat OpenShift 4.21 - Added support for NVIDIA GB300 NVL72 (DGX/HGX) systems - Added support for NVIDIA Spectrum-X Reference Architecture 2.2 (RA 2.2) in NVIDIA Spectrum-X Operator, with unified configuration and the new `SpectrumXRailPoolConfig` v1alpha2 CRD - Added IPv6 support for Spectrum-X on Kubernetes - Added consistent naming of RDMA and network devices in Pods - Added RDMA CNI host-to-Pod QoS propagation - Enhanced NVIDIA Kubernetes Launch Kit with simplified Spectrum-X configuration generation, new use-cases, and AI-assisted troubleshooting - Added KubeVirt support on vanilla Kubernetes, enabling Kubernetes-native VM deployments with accelerated East-West networking - Added heterogeneous cluster support through multiple NIC policies, with the new `NicNodePolicy` CRD enabling per-node-group DOCA-OFED driver management in NVIDIA Network Operator - Added global configuration support for `NicClusterPolicy` - Added preflight checks for the DOCA-OFED driver container (kernel module dependency validation prior to driver load) - Added interactive HTML report output for the SOS report collection script - [TECH PREVIEW] Added Dynamic Resource Allocation (DRA) support for SR-IOV in NVIDIA Network Operator - [TECH PREVIEW] Added support for multiple workloads (Pods) per node with topology awareness using DRA on Spectrum-X

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 - [TECH PREVIEW] Added support for NVIDIA Spectrum-X Operator deployment
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 SR-IOV mode with OVS CNI on 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 - [TECH PREVIEW] Added support for NVIDIA Grace based ARM platforms with Ubuntu 22.04 and Upstream K8s - 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 - [TECH PREVIEW] Added support for Switchdev SR-IOV mode with SR-IOV Network Operator and OVS CNI - 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	- [TECH PREVIEW] Added support for Ubuntu 22.04 with Upstream K8s on ARM platforms (NVIDIA IGX Orin) - 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
23.1.0	- 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
26.4.0	- Fixed stale VF representor deletion from the OVS bridge in OVS-CNI after a Kubernetes node reboot
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
26.1.0	- 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 - 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, NVIDIA 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

Overview

Use this page to confirm that your hardware, operating system, and Kubernetes platform are supported by **NVIDIA Network Operator** before deployment.

Support terms used on this page:

- **Supported** — A configuration that NVIDIA maintains and backs for this release.
- **Validated** — A hardware or software configuration that NVIDIA has tested with this release.
- **GA** (Generally Available) — Production-ready support tier.
- **Tech Preview** — Limited testing; not recommended for production deployments.

Versioning and Lifecycle

NVIDIA Network Operator uses calendar versioning in the form `YY.MM.PP` (for example, 26.4.0). The first two fields identify the major version and release timeframe; the third field identifies the patch version, used for critical bug and CVE fixes.

When a new major version is released, the previous major version enters a **Deprecated** state and receives only patch updates for critical fixes. Earlier major versions reach **End of Support** and no longer receive updates.

Version	Status
26.4.x	Supported
26.1.x	Deprecated
25.10.x and lower	End of Support

Upgrades are supported within a major release, or to the next major release. For upgrade procedures and operational guidance, see Life Cycle Management.

Note

The product lifecycle and versioning are subject to change in future releases.

Important Limitations

Review the following limitations before deployment:

- **Firmware reset on BMC-controlled platforms** — NVIDIA DGX/HGX GB200 NVL72, B200, and B300 systems require additional configuration so that firmware updates apply without a reboot loop. See Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB).
- **No GPUDirect RDMA on Arm-based validated configurations** — NVIDIA IGX Orin and NVIDIA Grace ARM Server are validated for RoCE only, without GPUDirect RDMA. See Support for GPUDirect RDMA.
- **Precompiled DOCA-OFED driver containers are currently unsigned.**
- **Precompiled DOCA-OFED kernel-flavor support** — Only the Ubuntu `generic` kernel flavor is GA. The `nvidia`, `aws`, `azure`, and `oracle` flavors are Tech Preview.
- For per-release known issues, see the Release Notes.

Prerequisites

Component	Version	Notes
Kubernetes	>=1.31 and <=1.36	
Helm	v3.5+	For installation methods, 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. See Support for GPUDirect RDMA.

Supported NVIDIA Network Adapters

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

Product Family	Ethernet	InfiniBand	Max Port Speed	Validated Protocols / Modes	Notes
NVIDIA ConnectX-6 NIC	Yes	Yes	200 Gb/s	IB RDMA, RoCE	—
NVIDIA ConnectX-6 Dx NIC	Yes	No	200 Gb/s	RoCE	—
NVIDIA ConnectX-7 NIC	Yes	Yes	400 Gb/s	IB RDMA, RoCE	—
NVIDIA ConnectX-8 SuperNIC	Yes	Yes	800 Gb/s	IB RDMA, RoCE	—

NVIDIA ConnectX-9 SuperNIC	Yes	Yes	800 Gb/s	IB RDMA, RoCE	—
NVIDIA BlueField-3 DPU	Yes	No	200 Gb/s	RoCE	NIC mode only
NVIDIA BlueField-3 SuperNIC	Yes	No	400 Gb/s	RoCE	NIC mode only

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 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 IGX Orin	Arm (NVIDIA Orin)	NVIDIA Ampere	ConnectX-7	Ubuntu 22.04 (Arm64)	GA (RoCE only, without GPUDirect RDMA)
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 B200	x86	NVIDIA Blackwell	BlueField-3 SuperNIC (NIC mode) / ConnectX-7	Ubuntu 22.04 / 24.04 (x86) / Red Hat OpenShift	GA. Firmware reset required (see Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB))

NVIDIA DGX/HGX B300	x86	NVIDIA Blackwell	ConnectX-8 SuperNIC	Ubuntu 22.04 / 24.04 (x86) / Red Hat OpenShift	GA. Firmware reset required (see Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB))
NVIDIA DGX/HGX GB200 NVL72	Arm (NVIDIA Grace)	NVIDIA Blackwell	ConnectX-7	Ubuntu 24.04 (Arm64) / Red Hat OpenShift	GA. Firmware reset required (see Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB))
NVIDIA DGX/HGX GB300 NVL72	Arm (NVIDIA Grace)	NVIDIA Blackwell	ConnectX-8	Ubuntu 24.04 (Arm64) / Red Hat OpenShift	GA. Firmware reset required (see Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB))

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 documentation.

Operating System	Upstream Kubernetes	Red Hat OpenShift	Rancher RKE2	Canonical MicroK8s	Nutanix NKP	Container Runtime	Notes

Ubuntu 24.04 LTS	1.31–1.36	—	—	1.31–1.36	2.12–2.15	Containerd	—
Ubuntu 22.04 LTS	1.31–1.36	—	—	1.31–1.36	2.12–2.15	Containerd	RT kernel support
Red Hat CoreOS	—	4.17–4.22	—	—	—	CRI-O	RT kernel support
Red Hat Enterprise Linux 10.0 / 10.2 / 9.8 / 9.6 / 9.4	1.31–1.36	—	—	—	—	Containerd, CRI-O	RT kernel support
Red Hat Enterprise Linux 8.10 / 8.8	1.31–1.36	—	—	—	—	Containerd, CRI-O	RT kernel support
SUSE Linux Enterprise Server 15 SP7	1.31–1.36	—	1.31–1.36	—	—	Containerd	Supported for Kubernetes and Rancher deployments

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 CoreOS	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 documentation.

Spectrum-X RA Version	Topologies and NIC Hardware	Kubernetes Versions	Operating Systems	Notes
2.2	• Single-Plane (ConnectX-7 or BlueField-3 SuperNIC) • Dual- or Quad-plane (NVIDIA ConnectX-8 SuperNIC)	Upstream Kubernetes (1.31–1.36)	• Ubuntu 24.04 LTS • Ubuntu 22.04 LTS	Small RA, Software Multi-Plane
2.1	• Single-Plane (ConnectX-7 or BlueField-3 SuperNIC) • Dual- or Quad-plane (NVIDIA ConnectX-8 SuperNIC)	Upstream Kubernetes (1.31–1.36)	• Ubuntu 24.04 LTS • Ubuntu 22.04 LTS	Small RA, Software Multi-Plane

Support for KubeVirt SR-IOV Passthrough

NVIDIA Network Operator supports attaching SR-IOV Virtual Functions (VFs) to KubeVirt virtual machines via VFIO PCI passthrough. For deployment instructions, see KubeVirt SR-IOV Integration.

Operating System	Kubernetes	KubeVirt	Red Hat OpenShift Virtualization
Ubuntu 24.04 LTS	1.31–1.36	v1.8.2+	Not supported
Ubuntu 22.04 LTS	1.31–1.36	v1.8.2+	Not supported

Key limitations:

- VFIO passthrough requires IOMMU-capable hardware (Intel VT-d or AMD-Vi) with IOMMU enabled at boot.
- Live migration is not supported for VMs that use SR-IOV VFIO passthrough.
- Host-side RDMA is not available for VFIO-passed VFs; RDMA inside the guest is provided by the `mlx5_core` driver shipped with the guest image.

Support for GPUDirect RDMA

NVIDIA Network Operator enables GPUDirect RDMA between NVIDIA GPUs and supported NICs/SuperNICs when deployed together with **NVIDIA GPU Operator**. For deployment instructions, see Network Operator Deployment for GPUDirect Workloads. For GPU Operator-side support details, see NVIDIA GPU Operator Platform Support — Support for GPUDirect RDMA.

Requirements:

- NVIDIA GPU Operator v25.3.x or newer
- NVIDIA DOCA-OFED Driver v5.5-1.0.3.2 or newer
- `nvidia_peermem` kernel module (auto-loaded by recent NVIDIA GPU drivers)
- Supported NVIDIA GPU (Ampere, Hopper, or Blackwell)
- Supported NVIDIA NICs (NVIDIA ConnectX or BlueField)

Operating System	Kubernetes
Ubuntu 24.04 LTS	1.31–1.36
Ubuntu 22.04 LTS	1.31–1.36

Red Hat Enterprise Linux 10.0 / 9.6 / 9.4	1.31–1.36
Red Hat Enterprise Linux 8.10 / 8.8	1.31–1.36
Red Hat CoreOS	OpenShift 4.17–4.22

For background on the underlying technology, see the NVIDIA GPUDirect RDMA documentation.

Support for GPUDirect Storage

NVIDIA Network Operator provides the RDMA networking fabric (RoCE and InfiniBand) required by **GPUDirect Storage (GDS)**. GDS-compatible storage clients operate over the RDMA fabric provisioned by Network Operator.

> ⚠ **Warning**
>
> With the DOCA-OFED **driver container** and `ENABLE_NFSRDMA=true`, host-side NVMe-over-RDMA cannot run concurrently — DOCA-OFED's `ib_core` conflicts with the inbox `nvme_rdma` module. The **DOCA-Host package** install path is unaffected, as is local-NVMe storage.

For configuration details (including the `ENABLE_NFSRDMA` setting and the NVMe inbox kernel module workaround), see DOCA-OFED Driver Container. For background on the underlying technology and the list of GDS-compatible storage clients and version requirements, see the NVIDIA GPUDirect Storage documentation.

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 (for example, `6.8.0-31` for Ubuntu 24.04). For all other flavors, the latest kernel version available at the time of DOCA packaging and release 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	v26.4.1	Yes	
NVIDIA Network Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	network-operator-init-container	network-operator-v26.4.1	Yes	
DOCA-OFED Driver Container	NVIDIA (EULA)	nvcr.io/nvidia/mellanox	doca-driver	doca3.4.1-26.04-1.1.0.0-1	Yes	L d 2 0
RDMA Shared Device Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	k8s-rdma-shared-dev-plugin	network-operator-v26.4.1	Yes	
IB Kubernetes Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	ib-kubernetes	network-operator-v26.4.1	Yes	
IP Over Infiniband (IPoIB) CNI plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	ipoib-cni	network-operator-v26.4.1	Yes	
NVIDIA IPAM Plugin	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nvidia-k8s-ipam	network-operator-v26.4.1	Yes	

NVIDIA NIC Feature Discovery	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-feature-discovery	network-operator-v26.4.1	Yes	
DOCA Telemetry Service (DTS)	NVIDIA (EULA)	nvcr.io/nvidia/doca	doca_telemetry	1.25.5-doca3.4.0-host	Yes	
Node Feature Discovery	Community (OSS)	nvcr.io/nvidia/mellanox	node-feature-discovery	network-operator-v26.4.1	Yes	O d M b th w c
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-v26.4.1	Yes	
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-v26.4.1	Yes	
SRIOV Network Operator	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-v26.4.1	Yes	
SR-IOV Network Device Plugin	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-network-device-plugin	network-operator-v26.4.1	Yes	
SR-IOV CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	sriov-cni	network-operator-v26.4.1	Yes	
InfiniBand SR-IOV CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	ib-sriov-cni	network-operator-v26.4.1	Yes	
K8s CNI network plugins	Community (OSS)	nvcr.io/nvidia/mellanox	plugins	network-operator-v26.4.1	Yes	
Multus CNI	Community (OSS)	nvcr.io/nvidia/mellanox	multus-cni	network-operator-v26.4.1	Yes	

RDMA CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	rdma-cni	network-operator-v26.4.1	Yes	
Open vSwitch CNI plugin	Community (OSS)	nvcr.io/nvidia/mellanox	ovs-cni-plugin	network-operator-v26.4.1	No	
NVIDIA NIC Configuration Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-configuration-operator	network-operator-v26.4.1	No	
NVIDIA NIC Configuration Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon	network-operator-v26.4.1	No	
NVIDIA Maintenance Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	maintenance-operator	network-operator-v26.4.1	No	
NVIDIA Kubernetes Launch Kit	NVIDIA (OSS)	nvcr.io/nvidia/cloud-native	k8s-launch-kit	v26.4.1	No	
NVIDIA Spectrum-X Operator	NVIDIA (OSS)	nvcr.io/nvidia/mellanox	spectrum-x-operator	network-operator-v26.4.1	No	
DRA Driver SR-IOV	Community (OSS)	nvcr.io/nvidia/mellanox	dra-driver-sriov	network-operator-v26.4.1	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
- KubeVirt SR-IOV Integration
- DRA SR-IOV Driver

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 v26.4.1 \
  --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 v26.4.1 \
  --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 v26.4.1 >
values.yaml
```

Install with specifying the custom *values.yaml*

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version v26.4.1 \
  -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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    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-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    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-v26.4.1
    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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
```

```
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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-v26.4.1
    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-v26.4.1
    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-v26.4.1
  imagePullSecrets: []
  enableWebhook: false
secondaryNetwork:
  cniPlugins:
    image: plugins
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    imagePullSecrets: []
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
```

```
      version: network-operator-v26.4.1
      imagePullSecrets: []
    ipoib:
      image: ipoib-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
```

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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
```

```
  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-v26.4.1
  imagePullSecrets: []
  enableWebhook: false
secondaryNetwork:
  cniPlugins:
    image: plugins
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    imagePullSecrets: []
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
```

```
      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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
```

```
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
```

```
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
```

```
    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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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: doca3.4.1-26.04-1.1.0.0-1
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
     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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
```

```
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
```

`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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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-v26.4.1
    imagePullSecrets: []
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      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
```

KubeVirt SR-IOV Integration

This guide explains how to attach SR-IOV Virtual Functions (VFs) to KubeVirt virtual machines using VFIO PCI passthrough.

With `deviceType: vfio-pci`, a VF's PCI device is passed directly into the guest VM via the VFIO userspace interface. The VM gets near-native network performance because the data path bypasses the host kernel entirely.

Prerequisites

- KubeVirt installed on the cluster:
  - Kubernetes: KubeVirt installation guide
- IOMMU enabled on worker nodes:
  - Intel: kernel parameter `intel_iommu=on iommu=pt`
  - AMD: kernel parameter `amd_iommu=on iommu=pt`
- `vfio-pci` kernel module available on worker nodes
- `virtctl` CLI tool installed:
  - Kubernetes: virtctl client tool

Install the Network Operator

Install the Network Operator with NFD and SR-IOV Network Operator enabled:

`values.yaml`:

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

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version v26.4.1 \
  -f values.yaml \
  --wait
```

Create a NicClusterPolicy

Once the Network Operator is installed, create a NicClusterPolicy with Multus CNI and CNI plugins:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
      imagePullSecrets: []
```

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

Verify that the NicClusterPolicy is ready:

```
kubectl get nicclusterpolicy nic-cluster-policy
```

The `state` should show `ready` before proceeding.

Deployment

Step 1: Create an SriovNetworkNodePolicy

Configure VFs with `deviceType: vfio-pci`. The operator creates the VFs and binds them to the `vfio-pci` driver, making them available as allocatable extended resources on the node.

Set `isRdma: false` (RDMA is not compatible with `vfio-pci`). The guest VM must have the `mlx5_core` kernel module available.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: kubevirt-policy
  namespace: nvidia-network-operator
spec:
  resourceName: kubevirt_sriov
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 8
  nicSelector:
    vendor: "15b3"
    pfNames:
      - ens1f0
  deviceType: vfio-pci
  isRdma: false
```

Wait for the policy to be applied:

```
kubectl get sriovnetworknodestates -n nvidia-network-operator -o
jsonpath='{.items[*].status.syncStatus}'
```

The output should show `Succeeded` for all nodes.

Step 2: Create an SriovNetwork

Create an `SriovNetwork` CR that references the `resourceName` from the policy. This generates a `NetworkAttachmentDefinition` that KubeVirt VMs can consume.

> **Note**
>
> With VFIO passthrough, the VF is passed directly into the guest VM. The host kernel does not see the network interface, so pod-level CNI IPAM cannot assign IPs to the VF. IP addresses must be configured inside the guest (e.g. via cloud-init or DHCP from an external server on the L2 network).

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-kubevirt-net
  namespace: nvidia-network-operator
spec:
  resourceName: kubevirt_sriov
  networkNamespace: default
  spoofChk: "off"
  trust: "on"
```

Verify the `NetworkAttachmentDefinition` was created:

```
kubectl get net-attach-def -n default sriov-kubevirt-net
```

Step 3: Create a VirtualMachine

Define a `VirtualMachine` with an `sriov: {}` interface pointing at the network attachment definition. Since IPAM is handled inside the guest, use cloud-init to configure a static IP on the SR-IOV interface.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-sriov
  namespace: default
spec:
  runStrategy: Always
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {}
            - name: sriov-net
              sriov: {}
          disks:
            - name: containerdisk
              disk: {bus: virtio}
            - name: cloudinit
              disk: {bus: virtio}
        resources:
          requests:
            memory: "4Gi"
      networks:
        - name: default
          pod: {}
        - name: sriov-net
          multus:
            networkName: sriov-kubevirt-net
      volumes:
        - name: containerdisk
          containerDisk:
            image: quay.io/containerdisks/fedora:latest
        - name: cloudinit
```

```
          cloudInitNoCloud:
            userData: |-
              #cloud-config
              password: password123
              chpasswd: {expire: false}
              ssh_pwauth: true
              runcmd:
                - |
                  for i in $(seq 1 30); do
                    SRIOV_IF=$(ls -1 /sys/class/net/ | grep -v
^lo$ | grep -v ^enp1s0$ | head -1)
                    [ -n "$SRIOV_IF" ] && break
                    sleep 1
                  done
                  if [ -n "$SRIOV_IF" ]; then
                    nmcli con add type ethernet ifname $SRIOV_IF
con-name sriov \
                      ipv4.addresses 192.168.0.1/24 ipv4.method
manual
                    nmcli con up sriov
                  fi
```

The `sriov: {}` interface type tells KubeVirt to pass the VF into the VM via VFIO. KubeVirt's resource injector automatically adds the extended resource request (e.g. `nvidia.com/kubevirt_sriov: "1"`) to the virt-launcher pod.

The cloud-init `runcmd` script waits for the SR-IOV interface to appear (the `mlx5_core` driver must load first), then configures a static IP using `nmcli`. Adjust the IP address for each VM accordingly.

Verification

Check the VMI is Running

```
kubectl get vmi vm-sriov
```

Verify the VF Inside the Guest

Connect to the VM console:

```
virtctl console vm-sriov
```

Inside the guest, verify the SR-IOV interface and IP configuration:

```
ip a
lspci | grep -i mellanox
```

> **Note**
>
> NVIDIA NICs require the `mlx5_core` driver inside the guest. If no network interface appears but `lspci` shows the device, run `modprobe mlx5_core`.

Guest Driver Note

NVIDIA VFs passed via VFIO require the `mlx5_core` driver inside the guest VM. If the guest image does not include it, you need to either:

- Use a guest image with NVIDIA DOCA-OFED or inbox `mlx5_core` drivers pre-installed
- Install the driver via cloud-init at first boot

> ⚠ **Warning**
>
> Without the driver, the VF PCI device appears in `lspci` output but no network interface is created.

Limitations

No live migration

VFIO passthrough gives the VM direct access to hardware PCI resources. Live migration is not possible because hardware state cannot be serialized.

Host-side RDMA not available

`deviceType: vfio-pci` is incompatible with `isRdma: true` on the `SriovNetworkNodePolicy`. RDMA works inside the guest VM because `mlx5_core` provides both ethernet and RDMA capabilities. To enable RDMA inside the guest, install the required packages and load the kernel modules:

```
sudo dnf install -y kernel-modules-extra-$(uname -r) rdma-core
sudo modprobe ib_uverbs mlx5_ib
```

IOMMU required

Nodes without IOMMU support cannot use VFIO passthrough. Run `virt-host-validate qemu` on the worker nodes to check hardware virtualization and IOMMU:

```
virt-host-validate qemu
```

All checks should show `PASS`, including `Checking for device assignment IOMMU support` and

`Checking if IOMMU is enabled by kernel`.

Confirm IOMMU groups are populated:

```
ls /sys/kernel/iommu_groups/
```

If IOMMU checks fail, enable it on the worker nodes via kernel parameter (`intel_iommu=on iommu=pt` or `amd_iommu=on iommu=pt`) and reboot.

Troubleshooting

VF Not Available on Node

Check node allocatable resources:

```
kubectl describe node <node> | grep kubevirt_sriov
```

Verify the VFs are bound to `vfio-pci`:

```
kubectl exec -n nvidia-network-operator <config-daemon-pod> --
lspci -k -s <vf-pci-addr>
```

VMI Fails to Start

Check virt-launcher pod events:

```
kubectl describe pod virt-launcher-vm-sriov-xxxxx
```

Check KubeVirt logs:

```
kubectl logs virt-launcher-vm-sriov-xxxxx -c compute
```

Common causes:

- IOMMU not enabled on the host
- `vfio-pci` module not loaded
- No VFs available (all allocated to other workloads)

No Network Interface in Guest

If `lspci` inside the guest shows the device but no interface appears in `ip link`:

```
modprobe mlx5_core
```

If the module is not available, install NVIDIA DOCA-OFED drivers in the guest image.

DRA SR-IOV Driver

- Overview
  - Limitations
- Deployment
  - **Step 1**: Create NicClusterPolicy
  - **Step 2**: Create IPPool for nv-ipam
  - **Step 3**: Configure SR-IOV
  - **Step 4**: Create SR-IOV Network
  - **Step 5**: Create ResourceClaimTemplate
  - **Step 6**: Deploy test workload

- Resource Alignment

Dynamic Resource Allocation (DRA) is a Kubernetes concept for flexibly requesting, configuring, and sharing specialized devices like SR-IOV network interfaces. DRA puts device configuration and scheduling into the hands of device vendors through drivers such as the DRA Driver for SR-IOV. This page outlines how to install the NVIDIA DRA Driver for SR-IOV with the NVIDIA Network Operator.

> **Note**
>
> The DRA Driver for SR-IOV is a **Tech Preview** feature: it has limited testing and is not recommended for production deployments. See Platform Support for support-tier definitions.

Before using the DRA Driver for SR-IOV, it is recommended that you are familiar with the following concepts:

- Upstream Kubernetes DRA documentation.
- DRA Driver repository documentation.

Overview

With DRA Driver for SR-IOV, your Kubernetes workload can allocate and consume SR-IOV Virtual Functions (VFs) from supported NVIDIA network adapters using the native Kubernetes DRA framework.

You can use the DRA Driver for SR-IOV with the SR-IOV Network Operator to deploy and manage your SR-IOV network resources.

Limitations

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

> **⚠ Warning**
>
> On GB300, Vera Rubin, and Fractal systems, the PCIe root used to match a NIC to a GPU is not the root of the NIC itself. Instead, it is the PCIe root of the NIC's Data Direct sub-interface. This applies to ConnectX-8 and later adapters. The DRA SR-IOV driver does not currently support this topology.

Deployment

> **⚠ Warning**
>
> Running the DRA driver and the SR-IOV device plugin on the same cluster at the same time is not supported. When DRA is enabled, the SR-IOV device plugin will not run. It is recommended to delete any existing `SriovNetworkNodePolicy` resources before enabling DRA.

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

```
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true
sriovOperatorConfig:
  featureGates:
    dynamicResourceAllocation: true
```

Disable the SR-IOV Resources Injector to avoid conflicts with the DRA Driver for SR-IOV:

```
kubectl patch sriovoperatorconfig default -n nvidia-network-
operator --type merge -p '{"spec":{"enableInjector":false}}'
```

Step 1: Create NicClusterPolicy

```
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-v26.4.1
    enableWebhook: false
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
```

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

Step 2: Create IPPool for nv-ipam

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: IPPool
metadata:
  name: sriov-pool
  namespace: nvidia-network-operator
spec:
  subnet: 192.168.2.0/24
  perNodeBlockSize: 50
  gateway: 192.168.2.1
```

```
kubectl apply -f ippool.yaml
```

Step 3: Configure SR-IOV

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

```
kubectl apply -f sriovnetworknodepolicy.yaml
```

Wait for the `SriovNetworkNodeState` CRs to reach the `Synced` state:

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

Verify that `ResourceSlices` are created:

```
kubectl get resourceslices
```

The following is an example of a `ResourceSlice` created by the DRA SR-IOV driver, showing a single Virtual Function with its attributes:

```
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
metadata:
  generateName: c-237-177-60-062-
sriovnetwork.k8snetworkplumbingwg.io-
  name: c-237-177-60-062-sriovnetwork.k8snetworkplumbingwg.io-
t4mc5
  ownerReferences:
  - apiVersion: v1
    controller: true
    kind: Node
    name: c-237-177-60-062
spec:
  devices:
  - attributes:
      dra.net/numaNode:
        int: 0
      resource.kubernetes.io/pciBusID:
        string: "0000:08:00.4"
      resource.kubernetes.io/pcieRoot:
        string: pci0000:00
      sriovnetwork.k8snetworkplumbingwg.io/EswitchMode:
        string: legacy
      sriovnetwork.k8snetworkplumbingwg.io/PFName:
        string: eth2
      sriovnetwork.k8snetworkplumbingwg.io/deviceID:
        string: 101e
      sriovnetwork.k8snetworkplumbingwg.io/linkType:
        string: ethernet
      sriovnetwork.k8snetworkplumbingwg.io/parentPciAddress:
        string: "0000:00:00.0"
      sriovnetwork.k8snetworkplumbingwg.io/pciAddress:
        string: "0000:08:00.4"
      sriovnetwork.k8snetworkplumbingwg.io/pfDeviceID:
        string: 101d
```

```
      sriovnetwork.k8snetworkplumbingwg.io/vendor:
        string: 15b3
      sriovnetwork.k8snetworkplumbingwg.io/vfID:
        int: 2
      k8s.cni.cncf.io/resourceName:
        string: nvidia.com/sriov_resource
      k8s.cni.cncf.io/deviceId:
        string: "0000:08:00.4"
    name: 0000-08-00-4
```

Step 4: Create SR-IOV Network

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-rdma-network
  namespace: nvidia-network-operator
spec:
  ipam: |
    {
      "type": "nv-ipam",
      "poolName": "sriov-pool"
    }
  networkNamespace: default
  resourceName: sriov_resource
```

```
kubectl apply -f sriovnetwork.yaml
```

Step 5: Create ResourceClaimTemplate

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: sriov-vf
spec:
  spec:
    devices:
      requests:
      - name: vf
        exactly:
          deviceClassName: sriovnetwork.k8snetworkplumbingwg.io
          count: 1
          selectors:
          - cel:
              expression: >
                device.attributes["k8s.cni.cncf.io"].resourceName
== "nvidia.com/sriov_resource"
```

```
kubectl apply -f resourceclaimtemplate.yaml
```

Step 6: Deploy test workload

```
---
apiVersion: v1
kind: Pod
metadata:
  name: sriov-rdma-server
  namespace: default
  labels:
    app: sriov-rdma
    role: server
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-network
spec:
  tolerations:
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Exists"
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/master"
    operator: "Exists"
    effect: "NoSchedule"
  restartPolicy: Never
  containers:
  - name: rdma-test
    image: nvcr.io/nvidia/doca/doca:3.1.0-full-rt-host
    command: ["/bin/bash", "-c", "sleepinfinity"]
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
      privileged: true
    resources:
      claims:
      - name: vf
  resourceClaims:
  - name: vf
    resourceClaimTemplateName: sriov-vf
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-rdma-client
  namespace: default
  labels:
    app: sriov-rdma
    role: client
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-network
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: role
            operator: In
            values:
            - server
        topologyKey: kubernetes.io/hostname
  restartPolicy: Never
  containers:
  - name: rdma-test
    image: nvcr.io/nvidia/doca/doca:3.1.0-full-rt-host
    command: ["/bin/bash", "-c", "sleepinfinity"]
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
      privileged: true
    resources:
      claims:
      - name: vf
  resourceClaims:
  - name: vf
    resourceClaimTemplateName: sriov-vf
```

```
kubectl apply -f pod.yaml
```

Resource Alignment

DRA enables end users to select resources from different DRA drivers with matching attributes to achieve maximum performance. By using constraints with `matchAttribute`, the Kubernetes scheduler ensures that allocated devices share a common topology, such as the same PCIe root complex.

The following example shows a `ResourceClaimTemplate` that requests both an SR-IOV VF and a GPU from the NVIDIA DRA Driver for GPUs, constrained to share the same PCIe root:

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: resource-alignment
spec:
  spec:
    devices:
      requests:
      - name: vf
        exactly:
          deviceClassName: sriovnetwork.k8snetworkplumbingwg.io
          selectors:
          - cel:
              expression: >
                device.attributes["k8s.cni.cncf.io"].resourceName
== "nvidia.com/sriov_resource"
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          count: 1
      constraints:
      - matchAttribute: "resource.kubernetes.io/pcieRoot"
        requests: [vf, gpu]
```

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.

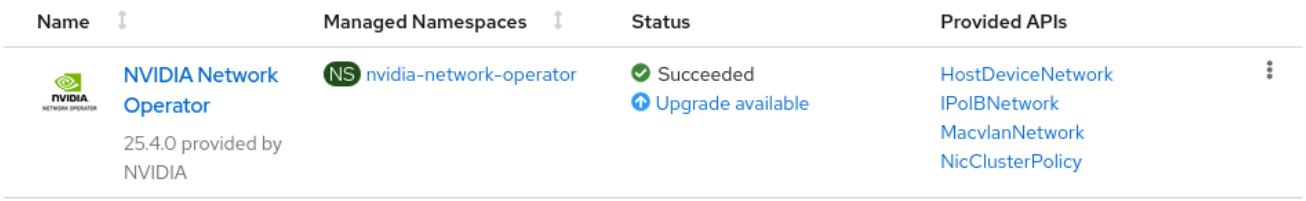
Network Operator Upgrade

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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    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-v26.4.1
    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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    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: doca3.4.1-26.04-1.1.0.0-1
     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-v26.4.1
  nvIpam:
```

```
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    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.

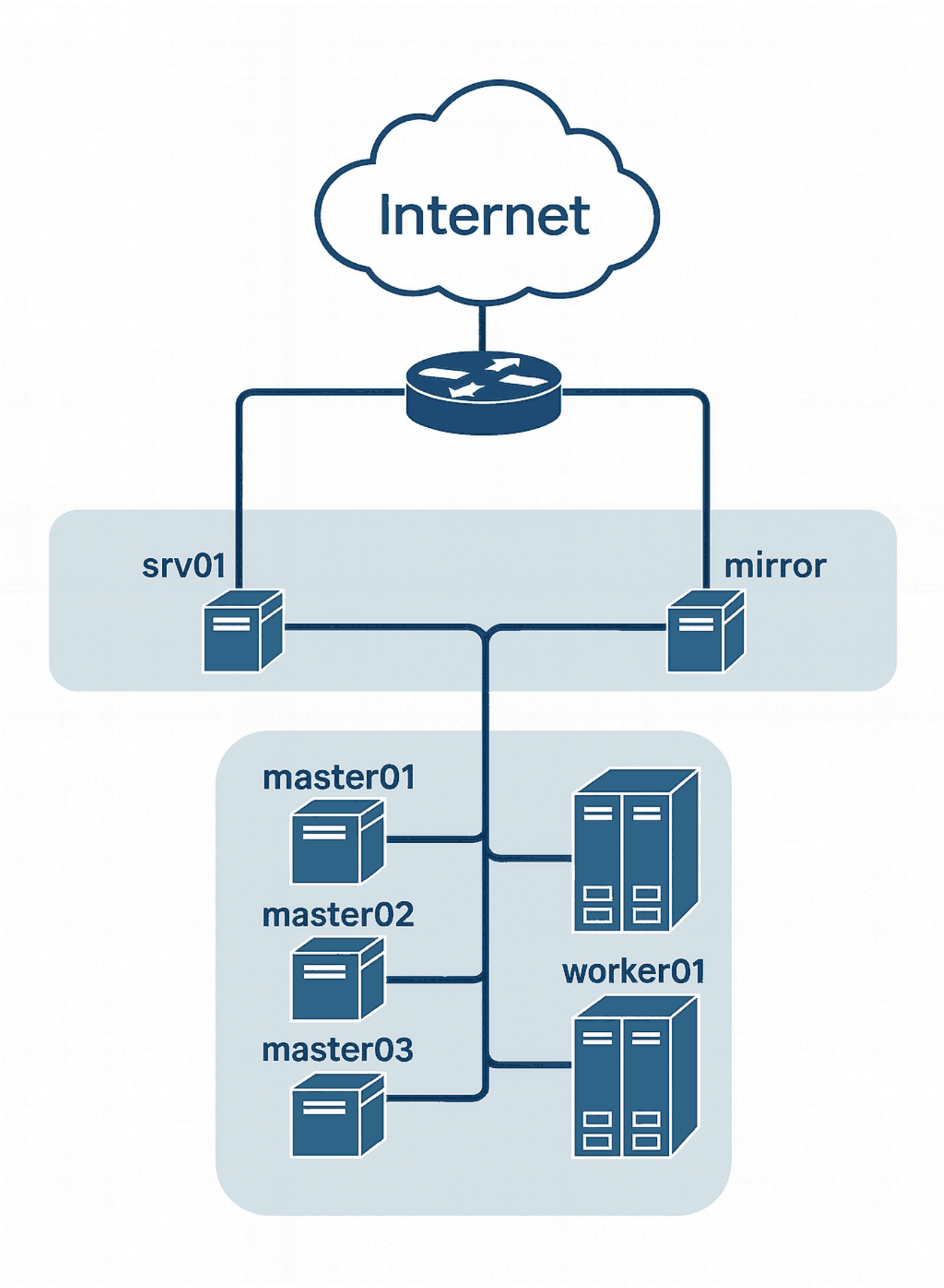
Prerequisites

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}]'
```

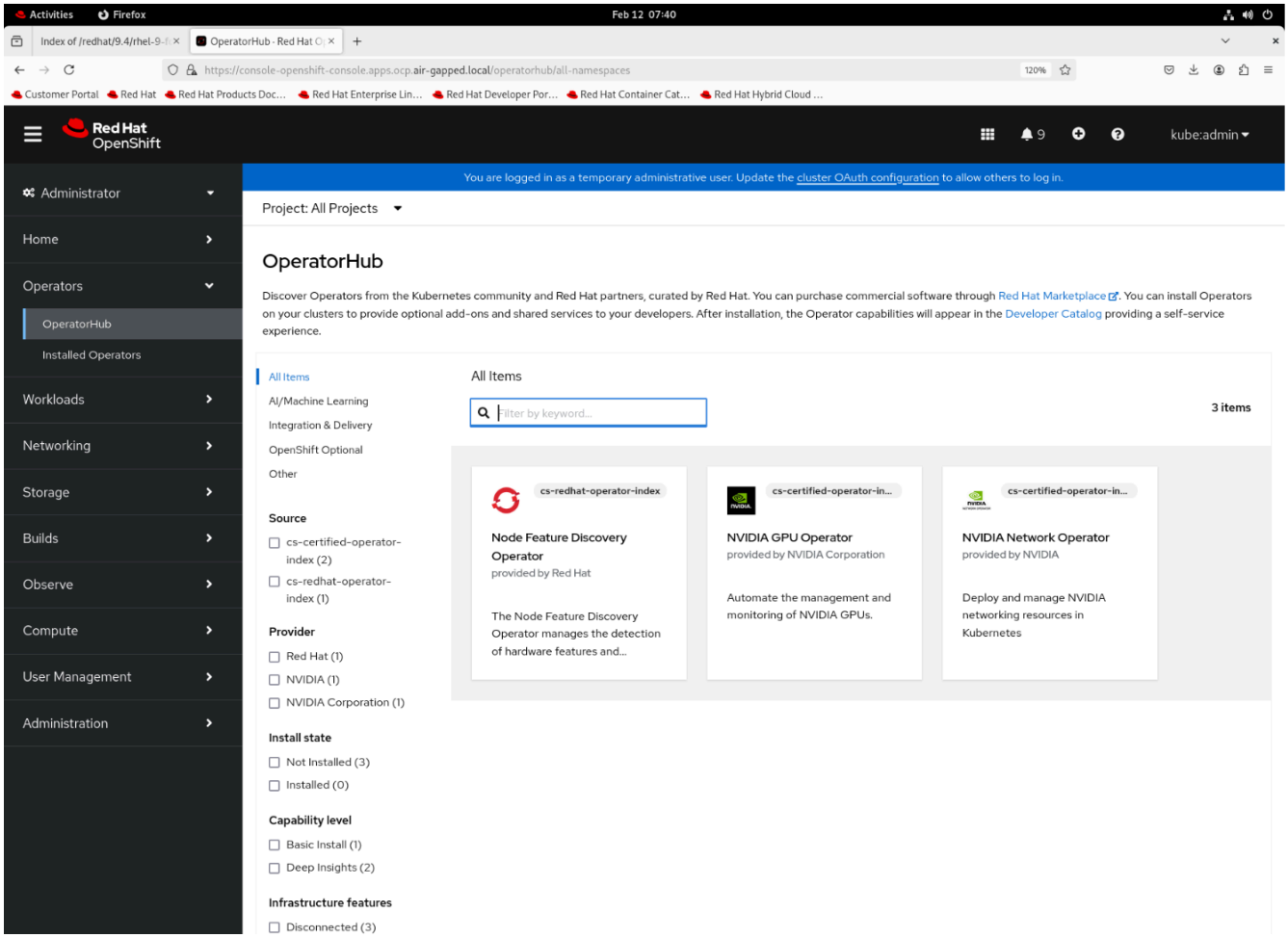
Apply the ICSP yaml file:

```
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	doca3.4.1-26.04-1.1.0.0-1
SR-IOV Network Operator Config Daemon	nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon-stig-fips	network-operator-v26.4.1-stig-fips
NIC Configuration Operator	nvcr.io/nvidia/mellanox	nic-configuration-operator-stig-fips	network-operator-v26.4.1-stig-fips-ubuntu / network-operator-v26.4.1-stig-fips-rhel

NIC Configuration Operator Daemon	nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon-stig-fips	network-operator-v26.4.1-stig-fips-ubuntu / network-operator-v26.4.1-stig-fips-rhel
Spectrum X Operator	nvcr.io/nvidia/mellanox	spectrumx-operator-stig-fips	network-operator-v26.4.1-stig-fips-ubuntu / network-operator-v26.4.1-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-
v26.4.1-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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1-stig-fips-ubuntu
    imagePullSecrets:
      - ngc-secret
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1-stig-fips-ubuntu
      imagePullSecrets:
        - ngc-secret
    configurationDaemon:
      image: nic-configuration-operator-daemon-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1-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: doca3.4.1-26.04-1.1.0.0-1
    imagePullSecrets:
      - ngc-secret
  spectrumXOperator:
    image: spectrum-x-operator-stig-fips
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1-stig-fips-rhel
    imagePullSecrets:
      - ngc-secret
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1-stig-fips-rhel
      imagePullSecrets:
        - ngc-secret
    configurationDaemon:
      image: nic-configuration-operator-daemon-stig-fips
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1-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
- Configuration Details
- CRD API Reference

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.

Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB)

On platforms where the NIC is controlled by an external BMC — including NVIDIA DGX and HGX systems (GB200 NVL72, B200, B300) and the Vera Rubin GB family — an OS-level node reboot does **not** reload NIC firmware: the BMC keeps the device powered across the reboot. As a result, persistent firmware parameters set with `mlxconfig` are not applied, and the operator stack can end up in a reboot loop trying to converge on the requested configuration.

To avoid this, configure both the NIC Configuration Operator and the SR-IOV Network Operator to perform an explicit firmware reset (`mlxfwreset` / `mstfwreset`) **before** the reboot. The two settings are independent — enable both when both operators are deployed, or just the one matching the operator you are running.

> **Note**
>
> `NicConfigurationTemplate` and `SriovNetworkNodePolicy` resources do not require any platform-specific changes. The firmware-reset behavior is controlled at the operator level only.

NIC Configuration Operator (`FW_RESET_AFTER_CONFIG_UPDATE`)

Set the `FW_RESET_AFTER_CONFIG_UPDATE` environment variable to `"true"` on the `nicConfigurationOperator` section of your `NicClusterPolicy`. The configuration daemon will then run `mlxfwreset` on each managed NIC after applying non-volatile configuration, before draining and rebooting the node.

```
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-v26.4.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
    env:
    - name: "FW_RESET_AFTER_CONFIG_UPDATE"
      value: "true"
```

SR-IOV Network Operator (`mellanoxFirmwareReset` feature gate)

Enable the `mellanoxFirmwareReset` feature gate on the `SriovOperatorConfig`. The SR-IOV config daemon will then invoke `mstfwreset` against the Mellanox NIC before triggering the reboot, so that firmware parameters changed by the SR-IOV plugin (for example `SRIOV_EN`, `NUM_OF_VFS`) take effect on external-BMC platforms.

When deploying via the Network Operator Helm chart, set the feature gate via Helm values:

`values.yaml`:

```
sriovNetworkOperator:
  enabled: true
sriov-network-operator:
  sriovOperatorConfig:
    featureGates:
      mellanoxFirmwareReset: true
```

For an already-installed cluster, patch the `SriovOperatorConfig` directly:

```
kubectl patch sriovoperatorconfigs.sriovnetwork.openshift.io \
  -n nvidia-network-operator default \
  --patch '{"spec":{"featureGates":
{"mellanoxFirmwareReset":true}}}' \
  --type=merge
```

The `mellanoxFirmwareReset` feature gate is documented as Beta in the upstream SR-IOV Network Operator project — see the SR-IOV Network Operator API documentation for details.

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 platforms where the NIC is controlled by an external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB), additional configuration is required so that firmware updates are applied without a reboot loop — see Platforms with external BMC (DGX, HGX GB200/B200/B300, Vera Rubin GB). The example below shows the relevant `FW_RESET_AFTER_CONFIG_UPDATE` knob commented out for reference.

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-v26.4.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
    # 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: doca3.4.1-26.04-1.1.0.0-1
    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"
    # serialNumbers:
    # - "MT2116X09299"
    # partNumbers:
    # - "MCX713106AEHEA_QP1"
  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”
    # Note: serial numbers are not guaranteed unique on systems
with embedded NICs
    # that share a flashed VPD image (e.g. HGX B300). Prefer
pciAddresses there.
    serialNumbers:
      - "MT2116X09299"
    # partNumbers:
    # - "MCX713106AEHEA_QP1"
  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)
    - Can be combined with `rawNvConfig` — raw params are merged as overrides on top of Spectrum-X calculated params
    - Only supported on ConnectX-8 (`nicType: 1023`), ConnectX-9 (`nicType: 1025`) and BlueField-3 SuperNIC (`nicType: a2dc`)
    - `version`: Required. Reference Architecture version (`RA1.3`, `RA2.0`, `RA2.1`, or `RA2.2`)
    - `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 * ConnectX-9 (device ID `1025`) – supports all multiplane modes (same configuration as ConnectX-8) * 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, ConnectX-9, BF3 SuperNIC	1
`swplb`	Software Packet Load Balancing	ConnectX-8, ConnectX-9, BF3 SuperNIC	2, 4
`hwplb`	Hardware Packet Load Balancing	ConnectX-8, ConnectX-9 only	2, 4
`uniplane`	Uniplane mode	ConnectX-8, ConnectX-9, 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 "1025" for ConnectX-9, or
"a2dc" for BlueField-3 SuperNIC (hwplb not supported on BF3)
      # partNumbers:
      # - "MCX713106AEHEA_QP1"
  template:
      numVfs: 1
      linkType: Ethernet
      spectrumXOptimized:
          enabled: true
          version: "RA2.1"
          overlay: "none"
          multiplaneMode: "hwplb" # Hardware Packet Load
Balancing, ConnectX-8 only
          numberOfPlanes: 4
```

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==1. RawNvConfig parameters, if provided, are merged as overrides on top of Spectrum-X calculated params.
`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. Empty means RDMA naming is skipped.
`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	SerialNumber of the device, e.g. MT2116X09299. Informational only — not guaranteed unique across all cards on a host: on systems with embedded NICs sharing a flashed VPD image (e.g. HGX B300) multiple cards will report the same serial number. The operator identifies NICs uniquely by their PCI device address (the `pci` field on the first entry in `ports`, with the function digit stripped).
`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	*(Optional)* RdmaDevicePrefix specifies the prefix for the rdma device name. When empty, no RDMA udev rules are generated and RDMA device naming is skipped. %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. Note: serial numbers are not guaranteed unique — on systems with embedded NICs that share a flashed VPD image (e.g. HGX B300), multiple physical cards report the same serial, and this selector will match all of them. Use `pciAddresses` for precise per-card selection on such systems.
`partNumbers` []string	Part numbers of the NICs to be selected, e.g. MCX713106AEHEA_QP1

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

NVIDIA Spectrum-X Ethernet Networking Platform

> **Note**
>
> This section covers **NVIDIA Network Operator configuration** to enable NVIDIA Spectrum-X NIC setup in Kubernetes deployments. For the full Spectrum-X platform documentation — supported topologies, NIC hardware, software components, and version-specific notes — refer to the NVIDIA Spectrum-X documentation.

NVIDIA Spectrum-X is an AI-optimized Ethernet networking platform that combines NVIDIA Spectrum switches with the BlueField-3 SuperNIC, ConnectX-7 NIC, and ConnectX-8 SuperNIC families to deliver high-bandwidth, lossless RoCE for the GPU-to-GPU compute (east-west) network. NVIDIA Network Operator provides the Kubernetes side: discovering the NICs, configuring rails, and exposing them to pods as schedulable resources.

Spectrum-X Multiplane. Spectrum-X Multiplane is the Spectrum-X capability that splits each SuperNIC across two or more independent network planes — enabling Ethernet to scale from thousands to hundreds of thousands of GPUs in a flat, two-tier topology, with improved performance and resiliency over single-plane networks. Network Operator exposes it through the `multiplaneMode` field on `NicConfigurationTemplate`, with **Software Multiplane** (`swplb`) and **Hardware Multiplane** (`hwplb`, tech preview) variants alongside the default single-plane mode (`none`).

Architecture and multiplane modes. Spectrum-X Kubernetes deployments fall into three network architectures, distinguished by the number of planes per rail and the load-balancing mechanism:

Architecture	NICs	GPU platforms	Multiplane mode	Status

Single-Plane	BlueField-3 SuperNIC, ConnectX-7 NIC, ConnectX-8 SuperNIC	H100/H200/B200, GB200	`none` (1 plane)	GA
Dual-Plane	ConnectX-8 SuperNIC	B300, GB300	`swplb` (2 planes) `hwplb` (2 planes)	`swplb` — GA `hwplb` — Tech preview
Quad-Plane	ConnectX-8 SuperNIC	B300	`swplb` (4 planes) `hwplb` (4 planes)	`swplb` — GA `hwplb` — Tech preview

> **Note**
>
> ConnectX-8 SuperNIC is listed in the Single-Plane row because it also supports single-plane (`none`) configuration. Typical Single-Plane deployments use BlueField-3 SuperNIC (HGX H100/H200/B200) or ConnectX-7 NIC (GB200).

Version compatibility. Each Spectrum-X Reference Architecture version is supported by a specific Network Operator release:

Spectrum-X RA Version	NVIDIA Network Operator Release
Spectrum-X RA 2.2	26.4.0
Spectrum-X RA 2.1	26.1.0

While each Network Operator release is **validated end-to-end** with a specific Spectrum-X RA version, individual components support a wider range of RAs in their configuration CRDs. For example, NIC Configuration Operator in 26.4.0 accepts

`spectrumXOptimized.version` values **RA1.3**, **RA2.0**, **RA2.1**, and **RA2.2**. See Spectrum-X NIC Configuration for details.

Configuration surface. Network Operator drives Spectrum-X setup through a small set of CRDs that work together:

- `NicClusterPolicy` — cluster-wide Network Operator configuration that enables the Spectrum-X Operator, SR-IOV Network Operator, NIC Configuration Operator, NV-IPAM, and Multus.
- `NicConfigurationTemplate` — NIC-level firmware/PF configuration for Spectrum-X (link type, `numVfs`, multiplane mode, RA version).
- `NicInterfaceNameTemplate` — predictable rail/plane-based netdev names driven by udev rules.
- `SpectrumXRailPoolConfig` (`spectrumx.nvidia.com/v1alpha2`) — rail topology, PF selection, IPAM binding, and DRA / SR-IOV resource exposure.
- `CIDRPool` (NV-IPAM) — IP allocation per rail (or per rail/plane in `swplb`).

For Dynamic Resource Allocation workflows (tech preview), the upstream Kubernetes `ResourceClaimTemplate` resource binds pod requests to specific GPU + VF combinations.

For the full operator / driver / CNI stack that backs these CRDs and how the components depend on each other, see Architecture and Components.

Further reading:

- **Supported platforms** (servers, NICs, switches, cables, OS combinations for each Spectrum-X release): see the NVIDIA Spectrum-X Solution Stack documentation.
- **Network Operator Kubernetes matrix** (operating systems, Kubernetes distribution versions, and the Spectrum-X RA support row): see Platform Support.
- **Component versions** for Network Operator, Spectrum-X Operator, NIC Configuration Operator, NV-IPAM, and the SR-IOV DRA driver: see the software components table in Platform Support.

When you're ready to deploy, continue to the Quick Start walkthroughs:

- Quick Start

- Architecture and Components
- Spectrum-X NIC Configuration
- CRD API Reference

Spectrum-X Kubernetes Quick Start

Note

These walkthroughs target **Spectrum-X RA 2.2** on **Network Operator 26.4.0**. For the RA-to-release mapping and other RAs, see NVIDIA Spectrum-X. For supported platforms (operating systems, Kubernetes distributions, NIC hardware), see Platform Support.

On Network Operator 26.4.0, **Single-Plane** and **Software Multiplane** (`swplb`) deployments on BlueField-3 SuperNICs, ConnectX-7 NICs, and ConnectX-8 SuperNICs are GA. **Hardware Multiplane** (`hwplb`) is tech preview only and is not part of the validated Spectrum-X Reference Architecture. For background on the operators and CNIs each walkthrough relies on, see Architecture and Components.

Tip

The configuration in each walkthrough can be automated end-to-end with NVIDIA Kubernetes Launch Kit — see Configuration Assistance with Kubernetes Launch Kit.

Pick the walkthrough that matches your hardware and target topology. Each one installs Network Operator via Helm, applies the Spectrum-X CRDs, and deploys a test pod — adapted to the chosen multiplane mode and NIC family.

Walkthrough	Multiplane mode	NICs	GPU platforms	Use when

Single Plane	`none`	BlueField-3 SuperNIC, ConnectX-7 NIC, ConnectX-8 SuperNIC	H100/H200/B200 (BlueField-3 SuperNIC), GB200 (ConnectX-7 NIC)	One PF per rail. Simplest setup. ConnectX-8 SuperNIC also supports single-plane configuration.
Software Multiplane	`swplb`	ConnectX-8 SuperNIC	B300, GB300	Software Plane Load Balancing across planes. Set `numberOfPlanes: 2` (Dual-Plane) or `4` (Quad-Plane, B300 only).
Hardware Multiplane *(tech preview)*	`hwplb`	ConnectX-8 SuperNIC	B300, GB300	Same as `swplb` but Plane Load Balancing happens in the NIC hardware. **Tech preview; not RA-validated.**

Single-Plane Spectrum-X Quick Start

> **Note**
>
> You can automate the configuration of this use case with NVIDIA Kubernetes Launch Kit. For more details, see Configuration Assistance with Kubernetes Launch Kit.

This walkthrough deploys a **Single-Plane** Spectrum-X cluster on Kubernetes: one PF per rail, one `CIDRPool` per rail, one network per rail. Used on **HGX H100/H200/B200** platforms (BlueField-3 SuperNIC, `nicType: a2dc`) and **GB200 NVL72** platforms

(ConnectX-7 NIC, `nicType: 1021`). **ConnectX-8 SuperNIC** (`nicType: 1023`) also supports single-plane configuration — useful if you want a single-plane setup on B300/GB300 hardware. The configuration uses RA 2.2 with `multiplaneMode: none` and `numberOfPlanes: 1`. The example below uses BlueField-3 SuperNIC; change `nicType` for other NICs. Replace `TODO_*` values with your cluster-specific values before applying.

Step 1: Install the 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. Spectrum-X Operator and NIC Configuration Operator are deployed via the same chart and enabled later through `NicClusterPolicy`.

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version 26.4.1 \
  --set sriovNetworkOperator.enabled=true \
  --wait
```

Verify the installation:

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

Step 2: Apply NicClusterPolicy

Enable the NIC Configuration Operator, NV-IPAM, Spectrum-X Operator (with XPlane), and the secondary network components.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      storageClassName: nic-fw-storage-class
      availableStorageSize: 1Gi
    logLevel: info
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    enableWebhook: false
  spectrumXOperator:
    image: spectrum-x-operator
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    # xPlane is only used when multiplaneMode=hwplb (Hardware
Multiplane).
    # Including it here lets you flip multiplaneMode without re-
applying NicClusterPolicy.
    xPlane:
      image: xplane
```

```
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
```

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

Step 3: NicInterfaceNameTemplate

Map PCI addresses to rails and define interface naming. With single-plane configuration there is one PF per NIC, so `pfsPerNic` is `1` and `%plane_id%` is always `0`. Replace `TODO_PCI_*` with the PCI addresses of the BlueField-3 SuperNICs on your nodes.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicInterfaceNameTemplate
metadata:
  name: spectrum-x-interface-names
  namespace: nvidia-network-operator
spec:
  pfsPerNic: 1
  rdmaDevicePrefix: "rdma_rail%rail_id%"
  netDevicePrefix: "net_rail%rail_id%"
  railPciAddresses:
    - ["TODO_PCI_RAIL0_NIC0"]
    - ["TODO_PCI_RAIL1_NIC0"]
```

```
kubectl apply -f nicinterfacenametemplate.yaml
```

Step 4: NicConfigurationTemplate

Configure the NICs for Spectrum-X RA 2.2 in single-plane mode. Use `nicType: a2dc` for BlueField-3 SuperNIC (HGX H100/H200/B200), `nicType: 1021` for ConnectX-7 NIC (GB200), or `nicType: 1023` for ConnectX-8 SuperNIC.

```
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 (HGX H100/H200/B200). Use "1021" for ConnectX-7 NIC (GB200) or "1023" for ConnectX-8 SuperNIC.
  template:
    numVfs: 1
    linkType: Ethernet
    spectrumXOptimized:
      enabled: true
      version: "RA2.2"
      overlay: "none"
      multiplaneMode: "none"
      numberOfPlanes: 1
```

```
kubectl apply -f nicconfigurationtemplate.yaml
```

Step 5: CIDRPool (per rail)

In single-plane mode each rail is a single network, so one `CIDRPool` per rail is sufficient. Replace `TODO_*` with subnets that match your cluster's east-west topology.

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-0
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL0_CIDR             # e.g., 10.0.0.0/15
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL0_SUBNET        # same as cidr
    - dst: TODO_EAST_WEST_SUBNET
---
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-1
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL1_CIDR
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL1_SUBNET
    - dst: TODO_EAST_WEST_SUBNET
```

```
kubectl apply -f cidrpool.yaml
```

Step 6: SpectrumXRailPoolConfig

One entry in `railTopology` per rail, selecting the single PF in that rail (matching `NicInterfaceNameTemplate`) and referencing the matching `CIDRPool`.

```
apiVersion: spectrumx.nvidia.com/v1alpha2
kind: SpectrumXRailPoolConfig
metadata:
  name: rails
  namespace: nvidia-network-operator
spec:
  draEnabled: true
  networkNamespace: default
  numVfs: 1
  railTopology:
    - name: rail0
      nicSelector:
        pfNames: ["net_rail0"]
      cidrPoolRef: rail-0
      mtu: 9216
    - name: rail1
      nicSelector:
        pfNames: ["net_rail1"]
      cidrPoolRef: rail-1
      mtu: 9216
```

```
kubectl apply -f spectrumxrailpoolconfig.yaml
```

Step 7: Deploy a Test Pod

Request one VF per rail. The network annotation references the rails created by `SpectrumXRailPoolConfig`.

```
apiVersion: v1
kind: Pod
metadata:
  name: spectrum-x-test
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: rail0,rail1
spec:
  containers:
    - name: spectrum-x-test
      image: nvcr.io/nvidia/doca/doca:3.3.0-full-rt-host
      command: ["/bin/bash", "-c", "sleepinfinity"]
      securityContext:
        capabilities:
          add: ["IPC_LOCK", "NET_RAW"]
      resources:
        requests:
          nvidia.com/rail_0: "1"
          nvidia.com/rail_1: "1"
        limits:
          nvidia.com/rail_0: "1"
          nvidia.com/rail_1: "1"
```

```
kubectl apply -f pod.yaml
kubectl -n default exec -it spectrum-x-test -- rdma link
```

Software Multiplane Spectrum-X Quick Start

> **Note**
>
> You can automate the configuration of this use case with NVIDIA Kubernetes Launch Kit. For more details, see Configuration Assistance with Kubernetes Launch Kit.

This walkthrough deploys a **Software Multiplane** Spectrum-X cluster on Kubernetes using **ConnectX-8 SuperNICs** (`nicType: 1023`). Each SuperNIC is split into multiple PFs, each assigned to a separate plane, and the software stack performs **Software Plane Load Balancing** (`swplb`) across them. Used on **B300** and **GB300** platforms — set `numberOfPlanes: 2` for Dual-Plane or `numberOfPlanes: 4` for Quad-Plane (B300 only). The configuration uses RA 2.2 with `multiplaneMode: swplb`. The example below uses `numberOfPlanes: 2`. Replace `TODO_*` values with your cluster-specific values before applying.

Step 1: Install the 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. Spectrum-X Operator and NIC Configuration Operator are deployed via the same chart and enabled later through `NicClusterPolicy`.

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version 26.4.1 \
  --set sriovNetworkOperator.enabled=true \
  --wait
```

Verify the installation:

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

Step 2: Apply NicClusterPolicy

Enable the NIC Configuration Operator, NV-IPAM, Spectrum-X Operator (with XPlane), and the secondary network components.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      storageClassName: nic-fw-storage-class
      availableStorageSize: 1Gi
    logLevel: info
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    enableWebhook: false
  spectrumXOperator:
    image: spectrum-x-operator
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    # xPlane is only used when multiplaneMode=hwplb (Hardware
Multiplane).
    # Including it here lets you flip multiplaneMode without re-
applying NicClusterPolicy.
    xPlane:
      image: xplane
```

```
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
```

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

Step 3: NicInterfaceNameTemplate

Map PCI addresses to rail/plane indices and define interface naming. Replace `TODO_PCI_*` with the PCI addresses of the Spectrum-X NICs on your nodes.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicInterfaceNameTemplate
metadata:
  name: spectrum-x-interface-names
  namespace: nvidia-network-operator
spec:
  pfsPerNic: 2
  rdmaDevicePrefix: "rdma_rail%rail_id%_plane%plane_id%"
  netDevicePrefix: "net_rail%rail_id%_plane%plane_id%"
  railPciAddresses:
    - ["TODO_PCI_RAIL0_NIC0", "TODO_PCI_RAIL0_NIC1"]
    - ["TODO_PCI_RAIL1_NIC0", "TODO_PCI_RAIL1_NIC1"]
```

```
kubectl apply -f nicinterfacenametemplate.yaml
```

Step 4: NicConfigurationTemplate

Configure the ConnectX-8 SuperNICs for Spectrum-X RA 2.2 with `swplb` multiplane mode. For Quad-Plane (B300 only), set `numberOfPlanes: 4`.

```
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: "1023"  # ConnectX-8 SuperNIC (B300, GB300)
  template:
    numVfs: 1
    linkType: Ethernet
    spectrumXOptimized:
      enabled: true
      version: "RA2.2"
      overlay: "none"
      multiplaneMode: "swplb"
      numberOfPlanes: 2
```

```
kubectl apply -f nicconfigurationtemplate.yaml
```

Step 5: CIDRPool (per rail-plane)

With `swplb`, each rail is split into multiple planes, and each plane requires its own `CIDRPool`. Create one pool per (rail, plane) combination. Replace `TODO_*` with subnets that match your cluster's east-west topology.

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-0-plane-0
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL0_PLANE0_CIDR      # e.g., 10.0.0.0/15
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL0_PLANE0_SUBNET # same as cidr
    - dst: TODO_EAST_WEST_SUBNET
---
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-0-plane-1
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL0_PLANE1_CIDR
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL0_PLANE1_SUBNET
    - dst: TODO_EAST_WEST_SUBNET
---
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
```

```
  name: rail-1-plane-0
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL1_PLANE0_CIDR
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL1_PLANE0_SUBNET
    - dst: TODO_EAST_WEST_SUBNET
---
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-1-plane-1
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL1_PLANE1_CIDR
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL1_PLANE1_SUBNET
    - dst: TODO_EAST_WEST_SUBNET
```

```
kubectl apply -f cidrpool.yaml
```

Step 6: SpectrumXRailPoolConfig

With `swplb`, `railTopology` has one entry per (rail, plane) combination. Each entry selects a single PF (netdev name from `NicInterfaceNameTemplate`) and references the matching per-plane `CIDRPool`.

```
apiVersion: spectrumx.nvidia.com/v1alpha2
kind: SpectrumXRailPoolConfig
metadata:
  name: rails
  namespace: nvidia-network-operator
spec:
  draEnabled: true
  networkNamespace: default
  numVfs: 1
  railTopology:
    - name: rail0p0
      nicSelector:
        pfNames: ["net_rail0_plane0"]
      cidrPoolRef: rail-0-plane-0
      mtu: 9216
    - name: rail0p1
      nicSelector:
        pfNames: ["net_rail0_plane1"]
      cidrPoolRef: rail-0-plane-1
      mtu: 9216
    - name: rail1p0
      nicSelector:
        pfNames: ["net_rail1_plane0"]
      cidrPoolRef: rail-1-plane-0
      mtu: 9216
    - name: rail1p1
      nicSelector:
        pfNames: ["net_rail1_plane1"]
      cidrPoolRef: rail-1-plane-1
      mtu: 9216
```

```
kubectl apply -f spectrumxrailpoolconfig.yaml
```

Step 7: Deploy a Test Pod

Request one VF per (rail, plane) combination. The network annotation lists each rail-plane and the resource request matches the corresponding DRA resource.

```
apiVersion: v1
kind: Pod
metadata:
  name: spectrum-x-test
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: rail0p0,rail0p1,rail1p0,rail1p1
spec:
  containers:
    - name: spectrum-x-test
      image: nvcr.io/nvidia/doca/doca:3.3.0-full-rt-host
      command: ["/bin/bash", "-c", "sleepinfinity"]
      securityContext:
        capabilities:
          add: ["IPC_LOCK", "NET_RAW"]
      resources:
        requests:
          nvidia.com/rail_0_plane_0: "1"
          nvidia.com/rail_0_plane_1: "1"
          nvidia.com/rail_1_plane_0: "1"
          nvidia.com/rail_1_plane_1: "1"
        limits:
          nvidia.com/rail_0_plane_0: "1"
          nvidia.com/rail_0_plane_1: "1"
          nvidia.com/rail_1_plane_0: "1"
          nvidia.com/rail_1_plane_1: "1"
```

```
kubectl apply -f pod.yaml
kubectl -n default exec -it spectrum-x-test -- rdma link
```

Hardware Multiplane Spectrum-X Quick Start (Tech Preview)

> ⚠ **Warning**
>
> Hardware Multiplane (`hwplb`) is **tech preview** in Network Operator 26.4.0 and is **not part of the validated Spectrum-X Reference Architecture**. Use only for evaluation purposes.

> ⓘ **Note**
>
> You can automate the configuration of this use case with NVIDIA Kubernetes Launch Kit. For more details, see Configuration Assistance with Kubernetes Launch Kit.

This walkthrough deploys a **Hardware Multiplane** Spectrum-X cluster on Kubernetes using **ConnectX-8 SuperNICs** (`nicType: 1023`). NIC LAG and **Hardware Plane Load Balancing** (`hwplb`) handle per-plane fan-out at the hardware layer, so each rail still uses a single `CIDRPool` while exposing multiple per-plane PFs. Used on **B300** and **GB300** platforms — set `numberOfPlanes: 2` for Dual-Plane or `numberOfPlanes: 4` for Quad-Plane (B300 only). The configuration uses RA 2.2 with `multiplaneMode: hwplb`. The example below uses `numberOfPlanes: 2`. Replace `TODO_*` values with your cluster-specific values before applying.

Step 1: Install the 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. Spectrum-X Operator and NIC Configuration Operator are deployed via the same chart and enabled later through `NicClusterPolicy`.

```
helm install network-operator nvidia/network-operator \
  -n nvidia-network-operator \
  --create-namespace \
  --version 26.4.1 \
  --set sriovNetworkOperator.enabled=true \
  --wait
```

Verify the installation:

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

Step 2: Apply NicClusterPolicy

Enable the NIC Configuration Operator, NV-IPAM, Spectrum-X Operator (with XPlane), and the secondary network components.

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nicConfigurationOperator:
    operator:
      image: nic-configuration-operator
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    configurationDaemon:
      image: nic-configuration-operator-daemon
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    nicFirmwareStorage:
      create: true
      pvcName: nic-fw-storage-pvc
      storageClassName: nic-fw-storage-class
      availableStorageSize: 1Gi
    logLevel: info
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    enableWebhook: false
  spectrumXOperator:
    image: spectrum-x-operator
    repository: nvcr.io/nvidia/cloud-native
    version: network-operator-v26.4.1
    xPlane:
      image: xplane
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
  secondaryNetwork:
    cniPlugins:
```

```
      image: plugins
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/cloud-native
      version: network-operator-v26.4.1
```

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

Step 3: NicInterfaceNameTemplate

Map PCI addresses to rail/plane indices and define interface naming. Replace `TODO_PCI_*` with the PCI addresses of the Spectrum-X NICs on your nodes.

```
apiVersion: configuration.net.nvidia.com/v1alpha1
kind: NicInterfaceNameTemplate
metadata:
  name: spectrum-x-interface-names
  namespace: nvidia-network-operator
spec:
  pfsPerNic: 2
  rdmaDevicePrefix: "rdma_rail%rail_id%_plane%plane_id%"
  netDevicePrefix: "net_rail%rail_id%_plane%plane_id%"
  railPciAddresses:
    - ["TODO_PCI_RAIL0_NIC0", "TODO_PCI_RAIL0_NIC1"]
    - ["TODO_PCI_RAIL1_NIC0", "TODO_PCI_RAIL1_NIC1"]
```

```
kubectl apply -f nicinterfacenametemplate.yaml
```

Step 4: NicConfigurationTemplate

Configure the ConnectX-8 SuperNICs for Spectrum-X RA 2.2 with `hwplb` multiplane mode. `hwplb` requires ConnectX-8 SuperNIC (`nicType: 1023`); it is not supported on BlueField-3 SuperNIC or ConnectX-7 NIC. For Quad-Plane (B300 only), set `numberOfPlanes: 4`.

```
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: "1023"  # ConnectX-8 SuperNIC (B300, GB300)
  template:
    numVfs: 1
    linkType: Ethernet
    spectrumXOptimized:
      enabled: true
      version: "RA2.2"
      overlay: "none"
      multiplaneMode: "hwplb"
      numberOfPlanes: 2
```

```
kubectl apply -f nicconfigurationtemplate.yaml
```

Step 5: CIDRPool (per rail)

With `hwplb`, load balancing happens at the NIC layer, so each rail uses a single `CIDRPool` covering all of its planes. Replace `TODO_*` with subnets that match your cluster's east-west topology.

```
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-0
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL0_CIDR             # e.g., 10.0.0.0/15
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL0_SUBNET        # same as cidr
    - dst: TODO_EAST_WEST_SUBNET
---
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: CIDRPool
metadata:
  name: rail-1
  namespace: nvidia-network-operator
spec:
  cidr: TODO_RAIL1_CIDR
  gatewayIndex: 0
  perNodeNetworkPrefix: 31
  perNodeExclusions:
    - startIndex: 1
      endIndex: 1
  routes:
    - dst: TODO_RAIL1_SUBNET
    - dst: TODO_EAST_WEST_SUBNET
```

```
kubectl apply -f cidrpool.yaml
```

Step 6: SpectrumXRailPoolConfig

With `hwplb`, `railTopology` has one entry per rail. Each entry lists all per-plane PF netdev names belonging to that rail and references a single per-rail `CIDRPool`.

```
apiVersion: spectrumx.nvidia.com/v1alpha2
kind: SpectrumXRailPoolConfig
metadata:
  name: rails
  namespace: nvidia-network-operator
spec:
  draEnabled: true
  networkNamespace: default
  numVfs: 1
  railTopology:
    - name: rail0
      nicSelector:
        pfNames: ["net_rail0_plane0", "net_rail0_plane1"]
      cidrPoolRef: rail-0
      mtu: 9216
    - name: rail1
      nicSelector:
        pfNames: ["net_rail1_plane0", "net_rail1_plane1"]
      cidrPoolRef: rail-1
      mtu: 9216
```

```
kubectl apply -f spectrumxrailpoolconfig.yaml
```

Step 7: Deploy a Test Pod

Request one VF per rail. The network annotation references the rails created by `SpectrumXRailPoolConfig`.

```
apiVersion: v1
kind: Pod
metadata:
  name: spectrum-x-test
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: rail0,rail1
spec:
  containers:
    - name: spectrum-x-test
      image: nvcr.io/nvidia/doca/doca:3.3.0-full-rt-host
      command: ["/bin/bash", "-c", "sleepinfinity"]
      securityContext:
        capabilities:
          add: ["IPC_LOCK", "NET_RAW"]
      resources:
        requests:
          nvidia.com/rail_0: "1"
          nvidia.com/rail_1: "1"
        limits:
          nvidia.com/rail_0: "1"
          nvidia.com/rail_1: "1"
```

```
kubectl apply -f pod.yaml
kubectl -n default exec -it spectrum-x-test -- rdma link
```

Spectrum-X Kubernetes Architecture and Components

A Spectrum-X Kubernetes deployment is a layered stack split across two distinct domains:

- **Host Layer** — operating-system prerequisites, firmware settings, kernel drivers, and the host virtual switch. Owned by an automated provisioning stack (or by manual / scripted operator action) and **out of Network Operator scope**.

- **Kubernetes Layer** — Kubernetes-native operators, CNIs, IPAM, and discovery components. Deployed and reconciled by **NVIDIA Network Operator** on top of a working Kubernetes cluster.

This page maps each component to its role, dependencies, and default-enabled state in Network Operator 26.4.0. For exact image versions, see the software components table in Platform Support. For end-to-end walkthroughs, see Spectrum-X Kubernetes Quick Start.

Deployment ownership

Spectrum-X K8s deployments are split across three phases. Host-side provisioning (Day 0) is owned outside Network Operator scope; Day 1 and Day 2 are reconciled by Network Operator.

Phase	Scope	Owner
Day 0 (pre-K8s)	Host OS bootstrap, PCIe / SR-IOV firmware, switch fabric, SuperNIC firmware floor, hugepages, RDMA exclusive mode, DOCA-Host / OVS-DOCA install.	Automated provisioning stack or manual / scripted.
Day 1 (cluster bring-up)	Driver, operator, and CNI deployment on top of a working Kubernetes cluster; Spectrum-X CRD reconciliation.	NVIDIA Network Operator.
Day 2 (workload)	Pod connectivity (secondary networks), rail / plane IP allocation, VF / representor lifecycle.	NVIDIA Network Operator (CNI + IPAM).

Stack overview

Host Layer *(Day 0 — outside Network Operator scope)*:

- **OS prerequisites** — SR-IOV BIOS + firmware; hugepages; RDMA namespace exclusive mode.
- **Driver** — DOCA-Host (NVIDIA OFED kernel modules + DOCA tooling).
- **Virtual switch** — OVS-DOCA (hardware-accelerated Open vSwitch, bundled with DOCA-Host).

Kubernetes Layer *(Day 1 / Day 2 — owned by Network Operator)*:

- **Discovery** — Node Feature Discovery (NFD); NIC Feature Discovery.
- **Driver** *(alternative to host-installed DOCA-Host)* — DOCA-OFED driver container.
- **Operator** — Network Operator (umbrella); SR-IOV Network Operator; NIC Configuration Operator; Spectrum-X Operator; Maintenance Operator.
- **CNI / data-plane** — Multus; OVS-CNI; SR-IOV Network Device Plugin; SR-IOV CNI; RDMA-CNI; NV-IPAM; Spectrum-X flow-controller DaemonSet.
- **Optional / tech preview** — DOCA xPlane (Hardware Multiplane only); SR-IOV DRA driver; NicNodePolicy CRD; DOCA Telemetry Service.

Host Layer

The Host Layer is the foundation: operating-system prerequisites, firmware settings, kernel drivers, and the host virtual switch that exist on each node **before** Kubernetes is installed. These components are typically owned by an automated provisioning stack or by manual / scripted operator action, and are **not** managed by Network Operator.

OS prerequisites

Component	Role
SR-IOV BIOS + firmware (`SRIOV_EN=True`, `NUM_OF_VFS=1`)	BIOS and NIC firmware settings that enable Virtual Function creation on each SuperNIC.
Hugepages	Memory backing for OVS-DOCA. 400 MB per SuperNIC from DOCA 3.4 onwards (1 GB on earlier DOCA).
RDMA namespace exclusive mode (`options ib_core netns_mode=0`)	Per-namespace RDMA device isolation, required to assign RDMA devices into pod network namespaces.

Driver

Component	Role
DOCA-Host	Host-installed NVIDIA OFED kernel modules and DOCA tooling. Includes OVS-DOCA (see Virtual switch below). Default driver path for Spectrum-X RA 2.2 validated deployments.

> **Important**
>
> **DOCA-Host (host-installed)** and the **containerized DOCA-OFED driver** (deployed via Network Operator — see Driver (alternative to host-installed DOCA-Host)) are **mutually exclusive**. Choose one per node; do not deploy both. Spectrum-X RA 2.2 validated deployments typically use host-installed DOCA-Host.

Virtual switch

Component	Role
OVS-DOCA	Hardware-accelerated Open vSwitch, bundled with DOCA-Host. Provides the per-rail OVS bridges into which the Kubernetes Layer's **OVS-CNI** plugs SR-IOV VFs at pod creation time.

Kubernetes Layer

The Kubernetes Layer is what NVIDIA Network Operator deploys and reconciles on top of a working Kubernetes cluster. It includes Kubernetes-native operators, CNIs, IPAM, and discovery components. Configuration is driven through the Spectrum-X CRDs documented in Spectrum-X CRDs and API Reference.

Discovery

Component	Role	Default

Node Feature Discovery (NFD)	Labels nodes with PCI / RDMA / GPU features so SR-IOV Network Operator and the scheduler can target the right hosts.	On by default.
NIC Feature Discovery	NFD extension that exposes NIC-specific capabilities (link type, device IDs) as node labels.	Opt-in.

Driver (alternative to host-installed DOCA-Host)

Component	Role	Default
DOCA-OFED driver container (`doca-driver`)	Containerized NVIDIA OFED kernel driver. Used **only** when the host does not have DOCA-Host installed. See the host Driver section for the mutual-exclusion rule.	Opt-in via `ofedDriver` on `NicClusterPolicy`.

Operator

The operator sub-layer is what Network Operator deploys and reconciles when you enable the Spectrum-X bits in `NicClusterPolicy`.

Component	Role	Default	Spectrum-X relevance
Network Operator (umbrella)	Reconciles `NicClusterPolicy` (and `NicNodePolicy`); orchestrates all sub-operators, CNIs, and driver lifecycle.	Always on.	Essential.
SR-IOV Network Operator (sub-chart)	Owns `SriovNetworkNodePolicy` and `OVSNetwork`; manages VF creation, switchdev mode, and ships **OVS-CNI**, **SR-IOV CNI**, **RDMA-CNI**, the SR-IOV Device Plugin, and the DRA driver.	Off by default; **on for Spectrum-X**.	Essential.

NIC Configuration Operator	Owns `NicConfigurationTemplate`, `NicFirmwareTemplate`, and `NicInterfaceNameTemplate`; applies firmware and rail-name templates to each SuperNIC.	Opt-in.	Recommended
Spectrum-X Operator	Owns `SpectrumXRailPoolConfig` and deploys the **flow-controller DaemonSet** that programs per-rail OVS flows on each host (writing into the OVS-DOCA bridges from the Host Layer). Essential for any Spectrum-X mode (single-plane, `swplb`, `hwplb`).	Opt-in via `spectrumXOperator`.	Essential.
Maintenance Operator	Orchestrates node-maintenance windows (drain / cordon) for safe firmware updates and OFED driver upgrades.	Opt-in.	Optional.

CNI / data-plane

The CNI / data-plane sub-layer is what actually plugs Virtual Functions into pods and routes traffic across rails and planes. Most components are deployed by the SR-IOV Network Operator sub-chart but are listed individually here because they play distinct roles.

Component	Role	Default	Spectrum-X relevance
Multus CNI	Meta-CNI that allows pods to attach to multiple networks (one per rail / plane).	Opt-in via `secondaryNetwork.multus`.	Essential.

OVS-CNI	Plugs SR-IOV VFs into the per-rail OVS bridge (provided by OVS-DOCA in the Host Layer) and chains with NV-IPAM for address allocation. Shipped under the SR-IOV Network Operator sub-chart.	With SR-IOV Op.	Essential.
SR-IOV Network Device Plugin	Advertises VFs to kubelet as schedulable `nvidia.com/...` resources.	With SR-IOV Op.	Essential.
SR-IOV CNI	Standard SR-IOV CNI binary. OVS-CNI is the typical Spectrum-X data path; SR-IOV CNI is available for non-OVS deployments.	With SR-IOV Op.	Alternative.
RDMA-CNI	Moves RDMA devices into the pod network namespace (requires RDMA exclusive mode on the host).	With SR-IOV Op.	Optional.
NV-IPAM	Rail / plane-aware IP allocation for pods. Consumes `CIDRPool` CRDs and assigns IPs to VFs on pod creation.	Opt-in via `nvIpam`.	Essential.
Spectrum-X flow-controller DaemonSet	Programs OVS flows on each host so packets are forwarded onto the correct rail / plane uplink. Not a separately versioned CNI; deployed by Spectrum-X Operator.	With Spectrum-X Op.	Essential.

Optional / tech preview

These components are either tech preview in 26.4.0 or strictly optional for a Spectrum-X K8s deployment.

Component	Role	Status
DOCA xPlane sidecar	DOCA control-plane container used by the Spectrum-X flow controller for **Hardware Multiplane** (`hwplb`) deployments. Not required for single-plane (`none`) or **Software Multiplane** (`swplb`). Deployed via the `spectrumXOperator.xPlane` sub-spec on `NicClusterPolicy`.	**Tech preview** in 26.4.0; planned GA in 26.7.0 alongside `hwplb` GA.
SR-IOV DRA driver (`dra-driver-sriov`)	Dynamic Resource Allocation driver for SR-IOV (Kubernetes 1.32+ DRA API). Enables fine-grained VF claims via `ResourceClaim` and `ResourceClaimTemplate` objects.	**Tech preview** in 26.4.0.
NicNodePolicy CRD	Per-node-group DOCA-OFED driver management — supersedes the cluster-wide `ofedDriver` setting for heterogeneous clusters.	New in 26.4.0.
DOCA Telemetry Service	Host telemetry exporter for OVS, RDMA, and SuperNIC statistics.	Opt-in.

Dependencies and ordering

The following dependencies are enforced by Network Operator's reconciliation loop and validated by CRD webhooks where applicable:

- **Spectrum-X Operator** requires SR-IOV Network Operator (for OVS-CNI and the SR-IOV Device Plugin), NIC Configuration Operator (for firmware and rail-name templates), NV-IPAM (for rail / plane IP allocation), and Multus (for secondary network attachment). It will not function standalone.

- **SR-IOV Network Operator** requires NFD labels on nodes and SR-IOV enabled in the host BIOS / firmware (Host Layer). It is off by default in `NicClusterPolicy` and must be explicitly enabled for Spectrum-X.
- **DOCA-OFED driver container** (Kubernetes Layer) and host-installed **DOCA-Host** (Host Layer) are alternatives, not additive. Choose one per node; do not deploy both.
- **OVS-CNI** (Kubernetes Layer) plugs VFs into OVS bridges provided by **OVS-DOCA** (Host Layer). OVS-DOCA must be present on each node before the Kubernetes Layer can program rail / plane flows.
- **DOCA xPlane** is required only when `multiplaneMode: hwplb` is set on the `NicConfigurationTemplate`. For single-plane and Software Multiplane deployments, the flow controller runs without xPlane.

Further reading

- **Image versions and supported platforms**: see the software components table in Platform Support.
- **Spectrum-X CRDs and API reference**: see Spectrum-X CRDs and API Reference.
- **NIC firmware and rail-name configuration**: see Spectrum-X NIC Configuration.
- **End-to-end deployment walkthroughs**: see Spectrum-X Kubernetes Quick Start.

NVIDIA Spectrum-X NIC Configuration

NVIDIA NIC Configuration Operator in Network Operator 26.4.0 supports **Spectrum-X RA 2.2** as the validated reference architecture. The `spectrumXOptimized.version` CRD field also accepts **RA 1.3**, **RA 2.0**, and **RA 2.1** for legacy or mixed-version configurations. RA 2.1 introduced multiplane mode support for enhanced network performance with multiple data planes; **RA 2.2 is the latest** and the validated default.

For the Network Operator Spectrum-X RA support matrix and validated hardware, operating system, and Kubernetes combinations, see Platform Support.

Note

Currently, ConnectX-7 NIC (device ID `1021`), ConnectX-8 SuperNIC (device ID `1023`), and BlueField-3 SuperNIC (device ID `a2dc`) devices are supported for Spectrum-X configuration. Hardware Plane Load Balancing (`hwplb`) is supported only on ConnectX-8 SuperNIC with RA 2.2, and is **tech preview** — it is not part of the validated Spectrum-X Reference Architecture.

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 or "1025" for ConnectX-9
    # partNumbers:
    # - "MCX713106AEHEA_QP1"
  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 or "1025" for ConnectX-9
      # partNumbers:
      # - "MCX713106AEHEA_QP1"
  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 / RA2.2 configuration with multiplane support

Reference Architecture 2.1 introduced multiplane mode support, allowing NICs to be configured with multiple data planes for enhanced network performance. RA2.2 extends this support and is the latest supported version.

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` or `RA2.2` 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 "1025" for ConnectX-9, or
"a2dc" for BlueField-3 SuperNIC (hwplb not supported on BF3)
      # partNumbers:
      # - "MCX713106AEHEA_QP1"
  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 and RA2.2:

Mode	Description	Supported NICs	Planes
`none`	Single plane mode (no multiplane). This is the default.	BlueField-3 SuperNIC, ConnectX-7 NIC, ConnectX-8 SuperNIC	1

`swplb`	Software Plane Load Balancing. The NIC port is split into multiple PFs, each assigned to a separate data plane; the software stack distributes packets across planes.	ConnectX-8 SuperNIC	2, 4
`hwplb` *(tech preview)*	Hardware Plane Load Balancing. Uses hardware LAG resource allocation and NIC-level plane configuration to distribute packets across planes in the NIC hardware. RA 2.2 only. **Tech preview; not part of the validated Reference Architecture.**	ConnectX-8 SuperNIC	2, 4
`uniplane`	Uniplane mode. Single-plane physical topology that uses the Spectrum-X multi-plane software stack and IP schema — multiple PFs all connect to the **same** ToR/plane (rather than separate planes as in `swplb` / `hwplb`). Specialized configuration for compatibility or regression scenarios; for production deployments, use `none` for Single-Plane or `swplb` / `hwplb` for Dual-Plane / Quad-Plane.	ConnectX-8 SuperNIC	2

Note

Multiplane modes (`swplb`, `hwplb`, `uniplane`) are only supported with RA2.1 and RA2.2. 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-7 NIC	`1021`	`none`
ConnectX-8 SuperNIC	`1023`	`none`, `swplb`, `hwplb`, `uniplane`
BlueField-3 SuperNIC	`a2dc`	`none`

Warning

The `hwplb` multiplane mode is supported only on ConnectX-8 SuperNIC (device ID `1023`) with RA 2.2, and is **tech preview** — it is not part of the validated Spectrum-X Reference Architecture. Attempting to configure `hwplb` on BlueField-3 SuperNIC or ConnectX-7 NIC, or with an RA version other than RA 2.2, 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"
```

Note

The actual generated names depend on the `netDevicePrefix` / `rdmaDevicePrefix` you set on the `NicInterfaceNameTemplate` — see the Quick-Start walkthroughs for the recommended rail / plane-based naming (for example, `net_rail0_plane0`).

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-7 NIC (`nicType: 1021`), ConnectX-8 SuperNIC (`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 SuperNIC (`nicType: 1023`), and only with `version: RA 2.2`. `hwplb` is tech preview and is not part of the validated Reference Architecture.

Spectrum-X CRDs and API Reference

A Spectrum-X Kubernetes deployment uses CRDs from several NVIDIA operators working together. This page documents the **Spectrum-X Operator** CRDs (v1alpha2 — `SpectrumXRailPoolConfig`). For the other CRDs, see the linked references. For the full stack of operators, drivers, and CNIs that own these CRDs and how they depend on each other, see Architecture and Components.

CRD	Owner	Role
`NicClusterPolicy`	NVIDIA Network Operator	Cluster-wide operator configuration. Enables NIC Configuration Operator, NV-IPAM, Spectrum-X Operator, and secondary network components.
`NicConfigurationTemplate`	NIC Configuration Operator	Per-NIC firmware and Spectrum-X settings (link type, `numVfs`, multiplane mode, RA version). See Spectrum-X NIC Configuration.
`NicInterfaceNameTemplate`	NIC Configuration Operator	Predictable rail / plane-based netdev and RDMA names driven by udev rules.
`CIDRPool`	NV-IPAM	Per-rail IP allocation pool consumed by `SpectrumXRailPoolConfig`.
SpectrumXRailPoolConfig	Spectrum-X Operator	Rail topology, PF selection, IPAM binding, and DRA / SR-IOV resource exposure (full API below).
`ResourceClaimTemplate`	Kubernetes DRA	Pod-to-GPU+VF binding via Dynamic Resource Allocation (Kubernetes upstream resource).

Packages:

- spectrumx.nvidia.com/v1alpha2

spectrumx.nvidia.com/v1alpha2

Package v1alpha2 contains API Schema definitions for the spectrumx v1alpha2 API group.

Resource Types:

- SpectrumXRailPoolConfig

SpectrumXRailPoolConfig

SpectrumXRailPoolConfig is the Schema for the spectrumxrailpoolconfigs API.

Field	Description
`apiVersion`	`spectrumx.nvidia.com/v1alpha2`
`kind`	`SpectrumXRailPoolConfig`
`metadata`	Standard object metadata. Refer to the Kubernetes API documentation for the fields of the `metadata` field.
`spec` SpectrumXRailPoolConfigSpec	Desired state of the SpectrumXRailPoolConfig.
`status` SpectrumXRailPoolConfigStatus	Observed state of the SpectrumXRailPoolConfig.

SpectrumXRailPoolConfigSpec

(*Appears on:* SpectrumXRailPoolConfig)

SpectrumXRailPoolConfigSpec defines the desired state of SpectrumXRailPoolConfig.

Field	Description
`draEnabled` bool	*(Optional)* Enables Dynamic Resource Allocation (DRA) for the configured Spectrum-X nodes. Defaults to `true`.
`nodeSelector` map[string]string	*(Optional)* Label selector that identifies the Spectrum-X nodes the configuration applies to.

`maxUnavailable` IntOrString	*(Optional)* Either an integer count or a percentage of nodes in the pool that can be configured in parallel. Defaults to `1`.
`networkNamespace` string	*(Optional)* Namespace of the NetworkAttachmentDefinition custom resource.
`numVfs` int	Number of VFs to configure on each PF. Must be `>= 1`.
`railTopology` []RailTopology	Rails topology list. Must contain at least one entry.

SpectrumXRailPoolConfigStatus

(*Appears on:* SpectrumXRailPoolConfig)

SpectrumXRailPoolConfigStatus defines the observed state of SpectrumXRailPoolConfig.

Field	Description
`syncStatus` string	Synchronization status. One of `Unknown`, `InProgress`, `Failed`, `Succeeded`.
`observedGeneration` int64	The most recent generation observed by the controller.

RailTopology

(*Appears on:* SpectrumXRailPoolConfigSpec)

RailTopology describes a single rail in the Spectrum-X pool. Only one of `cidrPoolRef` or `ipam` may be specified.

Field	Description
`name` string	Rail topology name. Must be non-empty.

`nicSelector` NicSelector	PF selector identifying the NICs that belong to this rail.
`cidrPoolRef` string	*(Optional)* Reference to a CIDR Pool resource. Mutually exclusive with `ipam`.
`ipam` string	*(Optional)* Advanced IPAM configuration. Mutually exclusive with `cidrPoolRef`.
`mtu` int	MTU for the rail. Must be `>= 0`.

NicSelector

(*Appears on:* RailTopology)

NicSelector selects the physical functions that participate in a rail.

Field	Description
`pfNames` []string	List of PF names. Must contain at least one entry.

NVIDIA Kubernetes Launch Kit

NVIDIA Kubernetes Launch Kit (l8k) is a CLI tool that automates the deployment of NVIDIA cloud-native networking on Kubernetes. It discovers cluster hardware, selects a deployment profile, generates Kubernetes YAML manifests, and optionally deploys them. Launch Kit supports SR-IOV, RDMA, InfiniBand, Host Device, MacVLAN, and Spectrum-X configurations, including heterogeneous clusters with mixed NIC and GPU hardware.

> **Note**
>
> **Who this is for:** operators and platform engineers deploying NVIDIA networking on a Kubernetes cluster. Launch Kit replaces hand-written `NicClusterPolicy`, `NicNodePolicy`, and `SriovNetworkNodePolicy` manifests with a discover-driven workflow.

Install

Install the `l8k` binary with the install script:

```
curl -fsSL https://raw.githubusercontent.com/nvidia/k8s-launch-kit/main/scripts/install.sh | sh
```

Or with Homebrew:

```
brew tap nvidia/l8k https://github.com/nvidia/k8s-launch-kit
brew install l8k
```

Where to Start

I want to...	Go to
See all installation options (container, build-from-source, version pinning)	Installation
Get the mental model and core vocabulary	Overview
Run an end-to-end walkthrough on a fresh cluster	Quick Start
Pick the right profile for my cluster	Deployment Profiles
Run a specific phase — discover, generate, or deploy	Workflows
Configure heterogeneous clusters, presets, CI/CD, AI agents, or troubleshoot	Advanced Topics
Look up flags or the config schema	Reference

Overview

This page explains the Launch Kit mental model. New users should read this once before working through the Quick Start. Returning users can skip it.

Workflow Phases

Launch Kit operates in three phases:

1. **Discover** — Probe the cluster's network hardware (NICs, PCI addresses, RDMA capability, OFED-dependent kernel modules, GPU topology) and write a `cluster-config.yaml` describing what was found.

2. **Generate** — Take a cluster configuration plus a deployment profile selection and render a complete set of numbered Kubernetes YAML manifests to a directory.

3. **Deploy** — Apply the generated manifests to the cluster in dependency order. Optional: `--dry-run` previews what would be applied without touching the cluster.

Each phase has a dedicated CLI subcommand — `l8k discover`, `l8k generate`, `l8k generate --deploy` — and a dedicated how-to page (Discover Workflow, Generate Workflow, Deploy Workflow).

The artifact pipeline:

flowchart LR KC[kubeconfig] BC[base config
optional] D([l8k discover]) CC[cluster-config.yaml] FLAGS[--fabric
--deployment-type
--network-operator-release] G([l8k generate]) MAN[./deployment/*.yaml] APPLY([deploy]) CLUSTER[(running cluster)] KC --> D BC -.-> D D --> CC CC --> G FLAGS -.-> G G --> MAN MAN --> APPLY APPLY --> CLUSTER classDef artifact fill:#fff,stroke:#888,stroke-dasharray: 4 3 classDef action fill:#79b8ff,stroke:#005cc5,color:#fff classDef state fill:#a2efb6,stroke:#28a745,color:#000 class KC,BC,CC,MAN,FLAGS artifact class D,G,APPLY action class CLUSTER state

Profiles

A **profile** is a named combination of fabric and deployment type that maps to a complete set of Kubernetes manifests. Launch Kit ships seven profiles:

- `sriov-ethernet` — SR-IOV with Ethernet
- `sriov-infiniband` — SR-IOV with InfiniBand
- `host-device` — direct passthrough to host network devices
- `macvlan-rdma-shared` — Ethernet with shared RDMA device plugin and MacVLAN
- `ipoib-rdma-shared` — InfiniBand with shared RDMA device plugin and IPoIB
- `spectrum-x` — Spectrum-X multi-rail AI interconnect (RA2.2 on Network Operator 26.4)
- `spectrum-x-ra2.1` — Spectrum-X for the RA2.1 reference architecture (Network Operator 26.1)

Profiles are selected by flag combinations on `l8k generate`: `--fabric`, `--deployment-type`, `--multirail`, and `--spectrum-x`. See Deployment Profiles for the decision matrix.

Network Operator Releases

Launch Kit pins to specific NVIDIA Network Operator releases. The `--network-operator-release` flag (and the `networkOperator.selectedRelease` config key) selects between supported release lines:

- `25.10`
- `26.1` — required for `spectrum-x-ra2.1`.
- `26.4` — required for `spectrum-x` (RA2.2).

The selected release auto-fills versions and image tags from an embedded catalog. Version mismatches (for example, requesting RA2.1 with release 26.4) error out with a specific message.

> **Note**
>
> Launch Kit is backward compatible with older Network Operator releases — a single `l8k` binary supports every release line in the catalog above. Always install the **latest** `l8k` release; pick the target Network Operator line via `--network-operator-release`. New releases of l8k add new lines to the catalog and pick up patch bumps for existing ones, so an older l8k binary will miss them.

Node Groups and Heterogeneity

During discovery, nodes are placed into **groups** by their PCI topology (the set of PCI addresses and device IDs across each node's PFs). Each group carries a `machineType` (e.g., `DGX-B200`), a `gpuType` (e.g., `NVIDIA-H100-NVL`), and a list of physical functions (PFs). Discovery writes a label

`nvidia.kubernetes-launch-kit.machine: <machineType>-<gpuType>` (sanitised) to every node in the group; the group's `identifier` and `nodeSelector` are both keyed by that label.

When the cluster has multiple groups, you have two strategies:

- **Automatic combination at generation** — groups stay separate in `cluster-config.yaml`, but `l8k generate` automatically combines groups sharing the same GPU type and east-west rail count into a single render group keyed by GPU type. One `l8k generate` run produces a single set of manifests covering all matching source groups.
- **Filter by group identifiers** — run `l8k generate --groups <a,b,...>` to restrict output to a named set of source groups. Use this when groups need different Network Operator releases, fabrics, deployment types, or driver versions per cohort.
- **Filter by GPU type** — run `l8k generate --gpu-type <X>` to restrict output to all source groups whose `gpuType` matches (case-insensitive). Best for declarative pipelines and CI/CD.

See Heterogeneous Clusters for the full picture.

Fabrics and Traffic Direction

The `--fabric` flag selects between `ethernet` and `infiniband`. Fabric is the physical transport; deployment type (`sriov` / `rdma_shared` / `host_device`) is the kubelet-facing networking model.

During discovery, Launch Kit also classifies each PF by **traffic direction**:

- **East-west** — GPU-to-GPU interconnect (ConnectX, BlueField-3 SuperNIC). These PFs appear in generated manifests.
- **North-south** — management / out-of-band (BlueField DPUs). These PFs are saved in `cluster-config.yaml` for visibility but **filtered out** of generated manifests.

Each east-west PF is assigned a sequential `rail` index (`rail: 0`, `rail: 1`, …) used in resource and network names.

Spectrum-X Multiplane Modes

Spectrum-X profiles add a **multiplane mode** dimension on top of profile selection:

- `hwplb` — hardware plane load balancing (large 2- or 3-tier switch topologies; tech preview, ConnectX-8 SuperNIC only)
- `swplb` — software plane load balancing (smaller-scale Spectrum-X clusters)
- `uniplane` — single unified plane (forces `--number-of-planes 1`)
- `none` — no plane separation (ConnectX-7 NIC, BlueField-3 SuperNIC; simple topologies)

NIC type constrains available modes: ConnectX-7 NIC and BlueField-3 SuperNIC support `none` only; ConnectX-8 SuperNIC supports `none`, `swplb`, `hwplb` (tech preview), and `uniplane`. See Spectrum-X.

Cluster Topology Presets

A **preset** is a pre-recorded hardware topology for a known machine type (e.g., `ThinkSystem-SR680a-V3`, `PowerEdge-XE9680`). Presets list expected PCI addresses, traffic class, NUMA affinity, and rail assignments.

Two ways presets are used:

- **Discovery overlay** — if the discovered machine type matches a known preset, the preset's topology is overlaid on discovery output for consistency.
- **Offline generation** — `l8k generate --for <preset>` skips discovery entirely and produces manifests for any cluster matching the preset's hardware.

See Cluster Topology Presets.

Glossary

Term	Definition
PF	Physical Function. A physical NIC interface, identified by PCI address.
VF	Virtual Function. An SR-IOV-virtualised slice of a PF.

Rail	A sequential index assigned to each east-west PF. Used in resource and network names.
Fabric	The physical transport layer: `ethernet` or `infiniband`.
Deployment type	The kubelet-facing networking model: `sriov`, `rdma_shared`, or `host_device`.
Profile	A named (fabric, deployment type, options) combination. See Deployment Profiles.
Group	A set of nodes sharing the same PCI topology. See Heterogeneous Clusters.
East-west	GPU-to-GPU interconnect traffic.
North-south	Management or out-of-band traffic. Filtered out of generated manifests.
Preset	A pre-recorded hardware topology for a known machine type. See Cluster Topology Presets.
Multiplane mode	Spectrum-X load-balancing mode: `hwplb`, `swplb`, `uniplane`, `none`.

See Also

- Quick Start — the canonical end-to-end walkthrough
- Deployment Profiles — decision matrix
- Heterogeneous Clusters — mixed-hardware deep dive
- Configuration Reference — `l8k-config.yaml` schema

Installation

Prerequisites

Before using Kubernetes Launch Kit, ensure the following:

- **Kubernetes cluster** with the NVIDIA Network Operator Helm chart installed, including the SR-IOV Network Operator, Maintenance Operator, and Node Feature Discovery (NFD). For installation instructions, see Getting Started with Kubernetes.
- `kubectl` configured with access to the target cluster.
- Docker or Podman on the machine where you will run `l8k` (only for the container scenario).

Install

Choose one of the methods below. The script and Homebrew methods install a binary plus profile templates and presets to a local prefix; the container method runs `l8k` from a published image and writes nothing outside your working directory.

Script

Downloads the latest release from GitHub and installs the `l8k` binary, profile templates, and presets to `/usr/local`:

```
curl -fsSL https://raw.githubusercontent.com/nvidia/k8s-launch-kit/main/scripts/install.sh | sh
```

Pin a specific version or install to a custom directory:

```
L8K_VERSION=v1.0.0 sh scripts/install.sh
curl -fsSL https://raw.githubusercontent.com/nvidia/k8s-launch-kit/main/scripts/install.sh | sh -s -- -d ~/local
```

Homebrew

For macOS and Linux systems with Homebrew:

```
brew tap nvidia/l8k https://github.com/nvidia/k8s-launch-kit
brew install l8k
```

Container

Run `l8k` directly from the published container image without copying any binary or assets to the host filesystem. The image stays in the local Docker cache; everything else (kubeconfig, output directory) is mounted at runtime.

Pull the image once:

```
docker pull nvcr.io/nvidia/cloud-native/k8s-launch-kit:v26.4.1
```

Define a shell alias so `l8k` invokes the container with kubeconfig + working directory mounted:

```
alias l8k='docker run --rm --net=host \
-v ~/.kube:/kube:ro \
-v $(pwd):/work -w /work \
nvcr.io/nvidia/cloud-native/k8s-launch-kit:v26.4.1'
```

Add the line to your `~/.bashrc` or `~/.zshrc` to make it persistent. With this alias in place, every `l8k <subcommand> ...` invocation creates a fresh container, runs the command, and removes the container — nothing is written outside the current working directory.

Note

`--net=host` is required so the container can reach the Kubernetes API server. The `--rm` flag ensures no stopped containers accumulate. Mount the kubeconfig directory read-only.

Verify

The same command works for all installation methods:

```
l8k version
```

Uninstall

Use the method that matches how you installed.

Script

```
curl -fsSL https://raw.githubusercontent.com/nvidia/k8s-launch-kit/main/scripts/install.sh | sh -s -- --uninstall
```

Homebrew

```
brew uninstall l8k
brew untap nvidia/l8k
```

Container

Drop the alias and remove the cached image:

```
unalias l8k
docker image rm nvcr.io/nvidia/cloud-native/k8s-launch-
kit:v26.4.1
```

Also remove the alias line from your shell rc file if you added it.

Layout (Script and Homebrew)

The script and Homebrew methods install the following under the install prefix (default: `/usr/local`):

- `<prefix>/bin/l8k` — CLI binary
- `<prefix>/share/l8k/profiles/` — profile templates
- `<prefix>/share/l8k/presets/` — cluster topology presets
- `<prefix>/share/l8k/l8k-config.yaml` — default configuration

The container method does not write anything outside your working directory; profiles, presets, and the default config are baked into the image.

See Also

- Quick Start — end-to-end walkthrough using `l8k`
- CLI Reference — full flag reference

Quick Start

This walkthrough takes you from a freshly prepared cluster to a running NVIDIA networking deployment using the **SR-IOV Ethernet** profile — the most common starting point. For other profiles, see the Deployment Profiles decision matrix.

> **Note**
>
> **Use this when:** you have the Network Operator Helm chart installed (per Installation), the `l8k` binary on your PATH, and `$KUBECONFIG` set or your default kubeconfig at `~/.kube/config`.

Step 1 — Discover

Probe the cluster's network hardware. Output is written to `./cluster-config.yaml` by default.

```
l8k discover
```

Discovery deploys a minimal probe profile, examines NICs and OFED-dependent kernel modules on each node, and groups nodes by PCI topology.

Step 2 — Generate

Generate Kubernetes manifests for an SR-IOV Ethernet deployment. Launch Kit auto-reads `./cluster-config.yaml` and writes manifests to `./deployment/`.

```
l8k generate --fabric ethernet --deployment-type sriov --
multirail
```

See Generate Workflow for the output file layout.

Step 3 — Deploy

Apply the generated manifests in dependency order (NicClusterPolicy first, then per-group NicNodePolicy, then networks and workloads):

```
l8k deploy
```

Step 4 — Verify

Check that operator pods, OFED driver pods, and example workloads are running:

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

Next Steps

- Picking a different profile? Deployment Profiles
- Cluster has mixed GPU/NIC types? Heterogeneous Clusters
- Generating manifests without cluster access? Cluster Topology Presets
- Wiring l8k into CI/CD? Automation and CI/CD
- Something didn't work? Troubleshooting

See Also

- Overview — the discover/generate/deploy mental model
- CLI Reference — every flag
- Configuration Reference — `cluster-config.yaml` schema

Deployment Profiles

A **profile** is a named combination of fabric and deployment type. Each profile maps to a complete set of Kubernetes manifests produced by `l8k generate`. Pick a profile from the decision tree or table below, then follow the link.

For the conceptual difference between fabric and deployment type, see Overview.

Decision Tree

Fabric is determined by the underlying hardware topology (Spectrum-X clusters are detected as a distinct fabric, not as a flavour of Ethernet). Pick the deployment type for each fabric:

flowchart TD Start{Fabric} Start -->|Ethernet| EthDT{Deployment type} Start -->|InfiniBand| IBDT{Deployment type} Start -->|Spectrum-X
HW topology| SPCXRA{RA version} EthDT -->|sriov| SRIOVE([SR-IOV Ethernet]) EthDT -->|host_device| HDE([Host Device]) EthDT -->|rdma_shared + MacVLAN| MV([MacVLAN RDMA Shared]) IBDT -->|sriov| SRIOVIB([SR-IOV InfiniBand]) IBDT -->|host_device| HDI([Host Device]) IBDT -->|rdma_shared + IPoIB| IPOIB([IPoIB RDMA Shared]) SPCXRA -->|RA2.1
Network Operator 26.1| SX1([Spectrum-X RA2.1]) SPCXRA -->|RA2.2
Network Operator 26.4| SX2([Spectrum-X RA2.2]) SX1 --> MPM[multiplane mode
swplb / hwplb / uniplane / none
+ number of planes] SX2 --> MPM classDef profile fill:#a2efb6,stroke:#28a745,color:#000 classDef params fill:#fff3cd,stroke:#ffc107,color:#000 class SRIOVE,HDE,MV,SRIOVIB,HDI,IPOIB,SX1,SX2 profile class MPM params

Decision Matrix

Profile	When to use	Fabric / Deployment type	Keywords
SR-IOV Ethernet	High-performance Ethernet networking with hardware acceleration. Per-pod dedicated VFs.	ethernet / sriov	SR-IOV, RDMA, low-latency, dedicated VFs
SR-IOV InfiniBand	Virtualized InfiniBand with hardware acceleration. Isolated IB partitions per pod.	infiniband / sriov	SR-IOV, InfiniBand, large-scale HPC

Host Device	Direct hardware passthrough. Minimal CPU overhead. Exclusive device access per pod.	ethernet or infiniband / host_device	host-device, PCI-passthrough, DPDK
IP over InfiniBand (RDMA Shared)	InfiniBand networking with shared RDMA resources. Parallel I/O workloads.	infiniband / rdma_shared	IPoIB, shared-device, high-bandwidth
MacVLAN (RDMA Shared)	Multi-tenant Ethernet with shared RDMA capabilities and network isolation.	ethernet / rdma_shared	MacVLAN, multi-tenant, network-segmentation
Spectrum-X	NVIDIA Spectrum-X multi-rail AI interconnect (RA2.1 or RA2.2). HWPLB (tech preview) / SWPLB / Uniplane / None modes.	ethernet / sriov / spectrum-x	Spectrum-X, multiplane, RA2.2, HWPLB

See Also

- Overview — profile selection vocabulary
- Generate Workflow — how profile flags are consumed
- Heterogeneous Clusters — mixing profiles per node group

Workflows

- Discover
- Generate
- Deploy
- Validate

Launch Kit operates in distinct phases. Each has a dedicated CLI subcommand and a focused how-to page.

Workflow	What it does
Discover	Probe the cluster's network hardware and write `cluster-config.yaml`.
Generate	Render Kubernetes manifests from a cluster config and a profile selection.
Deploy	Apply generated manifests to the cluster, with optional dry-run.
Validate	Verify the deployment matches the selected release: Helm chart version + manifest presence.

Discover Workflow

> **Note**
>
> **Use this when:** you want Launch Kit to inspect a live cluster's network hardware and produce a `cluster-config.yaml` describing it. Skip this workflow if you are generating manifests offline against a known machine type — see Cluster Topology Presets.

The `l8k discover` subcommand deploys a minimal Network Operator profile with the NIC Configuration Operator, then probes each node for NIC PCI addresses, device IDs, RDMA capability, InfiniBand support, OFED-dependent kernel modules, GPU topology, and machine type.

Basic Discovery

```
l8k discover
```

No flags required. `--kubeconfig` falls back to `$KUBECONFIG` (or `~/.kube/config`), and the output is written to `./cluster-config.yaml` by default. Discovery groups nodes by PCI topology (PCI addresses + device IDs across each node's PFs), labels each node with `nvidia.kubernetes-launch-kit.machine: <machineType>-<gpuType>`, and writes a `nodeSelector` keyed by that label per group.

Discovery with a Base Configuration

Provide your own configuration as a base. Discovery merges discovered hardware into your config, preserving custom settings (network operator version, subnets, MTU, etc.):

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

Without `--save-cluster-config`, the file specified by `--user-config` is updated in place. To save results to a separate file:

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

Filtering Nodes

Limit discovery to a subset of nodes using `--node-selector`:

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

The default selector targets nodes with Mellanox NICs.

PF Topology and Hardware Detection

For each NIC, Launch Kit derives PCI topology, NUMA affinity, and the connected GPU. The primary signal is `nvidia-smi` (queried via the NIC Configuration Daemon). When `nvidia-smi` is unavailable, Launch Kit falls back to sysfs and an embedded `pci.ids` database.

When GPU operator labels (`nvidia.com/gpu.machine`, `nvidia.com/gpu.product`) are not present, Launch Kit additionally execs into a NIC Configuration Daemon pod to read DMI data and `nvidia-smi` output directly, deriving the machine type and GPU product from hardware.

Group Labelling

After discovery resolves `machineType` and `gpuType` for a group, every node in the group is patched with two l8k-specific labels via a strategic-merge patch:

- `nvidia.kubernetes-launch-kit.machine: <machineType>-<gpuType>` — per-source-group identity. Used as the source group's `nodeSelector`.
- `nvidia.kubernetes-launch-kit.gpu: <gpuType>` — written alongside the machine label so auto-merged groups (different machineTypes sharing a GPU type) have a stable selector. Same value as `nvidia.com/gpu.product` by construction.

Label values keep their original case (e.g. `DGX-B200-NVIDIA-H100-NVL`, `NVIDIA-H100-NVL`); upstream parsing already trims whitespace and replaces spaces with hyphens. Values that would exceed Kubernetes' 63-char label limit are skipped (logged at debug).

The labels drive two pieces of downstream behaviour:

- **Group identity**: each group's `identifier` (lowercase, RFC 1123 resource-name form) and `nodeSelector` in `cluster-config.yaml` are keyed by the label. `l8k generate --groups <id1,id2,...>` targets exactly the nodes that carry the matching machine label(s); `--gpu-type <X>` selects every node whose `gpuType` matches.
- **Auto-merge selection**: when `l8k generate` merges groups sharing a GPU type, the merged group's `nodeSelector` keys on

`nvidia.kubernetes-launch-kit.gpu` so it covers every source machineType in the merged set.

Both labels are l8k state, not GPU operator state. Configs loaded from an earlier l8k version that used differential nodeSelectors keep working unchanged — the labels are only written on a fresh discovery run.

Groups whose `machineType` or `gpuType` could not be resolved keep a fallback `group-N` identifier. The machine label is skipped in that case, but the GPU label is still written when `gpuType` alone is resolved.

Fabric Type Detection

For each group's east-west PFs that have an RDMA device, Launch Kit reads `/sys/class/infiniband/<rdmaDevice>/ports/1/{state,phys_state,link_lay` from inside the NIC Configuration Daemon pod. A port contributes to the group's fabric verdict when:

- Port is `ACTIVE` and `link_layer=Ethernet` — contributes `Ethernet`.
- Port is `ACTIVE`, `link_layer=InfiniBand`, and a subnet manager is present (`sm_lid` non-zero) — contributes `InfiniBand`.

If every contributing port agrees on a single value, that value is recorded as `linkType` on the group in `cluster-config.yaml`. If no port produced a contribution (all down, IB without SM, probes failed), or different ports contribute different values, the field is **left empty** — discovery couldn't prove the cluster is using a specific fabric, and downstream code treats the absence as "unknown".

This is more reliable than reading `link_layer` alone: that file just reflects firmware config and may be a default rather than the cluster's actual fabric.

North-South Filtering

Each PF is classified by **traffic direction**:

- **East-west** — GPU interconnect (ConnectX, BlueField-3 SuperNIC). Included in generated manifests and assigned a sequential `rail` index.
- **North-south** — management or out-of-band (BlueField DPUs). Saved in `cluster-config.yaml` for visibility but **filtered out** of generated manifests.

BlueField-3 SuperNICs are explicitly classified as east-west even though they share a part-number prefix with BlueField DPUs — see the SuperNIC entry in the embedded DPU exclusion list.

For a traffic-directions diagram, see Heterogeneous Clusters.

Kernel Driver Dependencies Validation

During discovery, Launch Kit detects kernel modules that depend on OFED drivers. It execs into `nic-configuration-daemon` pods and builds a reverse dependency graph from `/sys/module/*/holders/` for the core MLX and OFED kernel modules.

The discovered modules are classified into two categories:

Category	Examples	Action
Storage-over-RDMA	`nvme_rdma`, `ib_isert`, `rpcrdma`	Auto-enables `docaDriver.unloadStorageModules: true`
Third-party RDMA	`rdma_rxe`, `qedr`, `bnxt_re`	Auto-enables `docaDriver.unloadThirdPartyRDMAModules: true`

Storage and third-party RDMA module lists are sourced from the `doca-driver-build` project to keep them in sync with the driver container itself. `mlx5`-prefixed modules (the OFED stack itself) are excluded from classification.

After discovery, the config reflects the auto-enabled flags and the discovered modules are saved per group as `storageModules` and `thirdPartyRDMAModules` lists.

To skip the kernel driver dependencies validation entirely (for environments where it's known-good), set `docaDriver.skipPreflightChecks: true` in your config.

> ⚠ **Warning**
>
> Verify that no running workloads depend on modules that will be unloaded. To disable automatic unloading, set `unloadStorageModules` and `unloadThirdPartyRDMAModules` back to `false` in your config after discovery.

Discovery Output

The output `cluster-config.yaml` contains all parameters needed for manifest generation. The typical workflow is:

1. Run `l8k discover` to produce `cluster-config.yaml`.
2. Edit the file to customize network parameters (subnets, MTU, image pull secrets, etc.).
3. Run `l8k generate --user-config ./cluster-config.yaml` to produce manifests.

You can also provide a pre-configured file directly to `l8k generate` without running discovery, or skip discovery entirely with `--for <preset>`.

For the full configuration schema, see Configuration Reference.

Debug Logging

Every probe and decision in discovery emits a structured-field debug line. To see them:

```
l8k discover --log-level debug
```

Logged probes include the GPU operator label read and hardware-fallback path, the DMI machine-type cat, the `nvidia-smi` and sysfs fallback, OFED-dependent module discovery and classification, preset matching with hit/miss, the PCI-fingerprint per-node bucketing, and the east-west / north-south traffic classification per PF. Phase summary

lines (`Discovery summary`, `Group merge complete`) are emitted at info level by default. The same convention applies to `l8k generate` for the group-merge logs.

See Also

- Generate Workflow — next step in the pipeline
- Heterogeneous Clusters — how groups are formed
- Cluster Topology Presets — offline alternative to discovery
- CLI Reference — `l8k discover` flags

Generate Workflow

> **Note**
>
> **Use this when:** you have a `cluster-config.yaml` (from Discover Workflow) or a known hardware preset and want to render Kubernetes manifests for a specific deployment profile.

Profile Selection via CLI Flags

Specify the deployment profile using `--fabric`, `--deployment-type`, and `--multirail`:

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

For Spectrum-X, the `--spectrum-x` flag automatically sets `--fabric ethernet`, `--deployment-type sriov`, and `--multirail true`.

> **Note**
>
> If the configuration file already contains a `profile` section, CLI flags are optional. CLI flags override configuration file values when both are provided.

Hardware-Derived Defaults

When a flag is unset on both the CLI and in the configuration file, `l8k generate` fills it from the discovered cluster:

Flag	Default	Trigger
`--fabric`	the cluster's fabric (`ethernet` or `infiniband`)	every group's `linkType` (Unit 5 fabric probe) agrees; skipped+warned otherwise
`--deployment-type`	`sriov`	always
`--multirail`	`true`	always; opt out via `--multirail=false` (YAML cannot express explicit-false)
`--multiplane-mode`	`uniplane` (CX7 / BF3 SuperNIC), `swplb` (CX8), `hwplb` (CX9)	only when `--spectrum-x` is set; per east-west PF deviceID, skipped+warned when groups have mixed deviceIDs
`--number-of-planes`	1 (CX7 / BF3 SuperNIC), 2 (CX8), 4 (CX9)	only when `--spectrum-x` is set
`--network-operator-release`	matching release for the chosen RA (RA2.1 → 26.1, RA2.2 → 26.4)	only when `--spectrum-x` is set

Each applied default is logged at info level (`Defaulted --multiplane-mode=swplb (ConnectX-8 (deviceID 1023))`); the full reasoning trail is at debug level (`--log-level debug`).

Resolution precedence (lowest → highest): hardware default < config-file < CLI flag. The Spectrum-X cohort rules (RA-to-release pairing, `multiplane-mode=none` requiring `number-of-planes=1`, etc.) are enforced after defaults run, so a partial CLI input plus defaults must still resolve to a consistent whole.

For the full profile decision matrix, see Deployment Profiles.

Network Operator Release

For Spectrum-X profiles — and recommended for all deployments — pin the Network Operator release line:

```
l8k generate --user-config ./cluster-config.yaml \
    --network-operator-release 26.4 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments
```

Supported release lines: `26.1` and `26.4`. The release auto-fills versions and image tags from an embedded catalog. RA2.1 Spectrum-X requires `--network-operator-release 26.1`; RA2.2 Spectrum-X requires `26.4`.

Generation without a Live Cluster

To generate manifests for a known machine type without running discovery, use `--for <preset>`:

```
l8k generate --for ThinkSystem-SR680a-V3 \
    --node-selector "nvidia.com/gpu.product=NVIDIA-H200" \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments
```

See Cluster Topology Presets for available presets and how to add new ones.

Output Directory Structure

Generated files follow a numbered naming convention that determines deployment order:

File pattern	Contents
`10-nicclusterpolicy.yaml`	Cluster-wide components: Multus, CNI plugins, NV-IPAM, NIC Configuration Operator, Spectrum-X Operator.
`11-nicnodepolicy.yaml`	Per-node-group components: OFED driver and device plugins. Rendered once per group with the group's `nodeSelector`.
`20-ippool.yaml`	NV-IPAM IP pools.
`30-*.yaml`	Network resources (`SriovNetworkNodePolicy`, `SriovNetwork`, `HostDeviceNetwork`, `MacvlanNetwork`, `IPoIBNetwork`).
`35-nicinterfacenametemplate.yaml`	NIC rename templates (only when needed — see Heterogeneous Clusters).
`40-*.yaml` / `50-*.yaml`	Example workload DaemonSets.
`overview.html`	Human-readable deployment summary.

In heterogeneous clusters, per-group files include the group identifier in the filename (e.g., `30-sriovnetworknodepolicy-group-0.yaml`).

Group-Specific Generation

In heterogeneous clusters, two flags scope the output to a subset of source groups:

```
# By identifier (comma-separated; case-sensitive match against
# cluster-config.yaml's clusterConfig[].identifier):
l8k generate --user-config ./cluster-config.yaml \
    --fabric infiniband --deployment-type sriov --multirail \
    --groups dgx-b200-nvidia-h100-nvl,poweredge-xe9680-nvidia-
h200 \
    --save-deployment-files ./deployments
# By GPU type (case-insensitive match against gpuType):
l8k generate --user-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --gpu-type NVIDIA-H200 \
    --save-deployment-files ./deployments
```

`--groups` and `--gpu-type` are mutually exclusive. An empty match is a validation error. `--gpu-type` is best for declarative pipelines; `--groups` is best for staged rollouts where the cohort is enumerated explicitly. See Heterogeneous Clusters for the per-Kind rendering rules under filter.

Custom Workload Manifests

By default, Launch Kit generates example DaemonSet workloads for each profile. To use your own workload manifest instead:

```
l8k generate --user-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov \
    --workload-manifest /path/to/my-workload.yaml \
    --save-deployment-files ./deployments
```

Launch Kit patches the workload manifest with the correct network annotations, resource requests, and node affinity based on the cluster configuration. Supported workload kinds: `Pod`, `Deployment`, `DaemonSet`, `StatefulSet`, `Job`, `ReplicaSet`.

Before — input manifest:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-rdma-workload
  namespace: default
spec:
  selector:
    matchLabels:
      app: my-rdma-workload
  template:
    metadata:
      labels:
        app: my-rdma-workload
    spec:
      containers:
      - name: rdma-app
        image: my-registry/my-rdma-app:latest
        securityContext:
          capabilities:
            add: ["IPC_LOCK"]
```

After — patched by Launch Kit for an SR-IOV deployment with 2 rails:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-rdma-workload
  namespace: default
spec:
  selector:
    matchLabels:
      app: my-rdma-workload
  template:
    metadata:
      labels:
        app: my-rdma-workload
      annotations:
        k8s.v1.cni.cncf.io/networks: sriov-network-rail-0,sriov-
network-rail-1
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: nvidia.com/gpu.machine
                operator: In
                values:
                - DGX-B200
      containers:
      - name: rdma-app
        image: my-registry/my-rdma-app:latest
        securityContext:
          capabilities:
            add: ["IPC_LOCK"]
        resources:
          requests:
            nvidia.com/sriov_resource_rail_0: "1"
```

```
            nvidia.com/sriov_resource_rail_1: "1"
        limits:
            nvidia.com/sriov_resource_rail_0: "1"
            nvidia.com/sriov_resource_rail_1: "1"
```

See Also

- Deploy Workflow — apply the generated manifests
- Deployment Profiles — profile decision matrix
- Heterogeneous Clusters — per-group manifests
- Cluster Topology Presets — offline generation with `--for`
- CLI Reference — `l8k generate` flags

Deploy Workflow

> **Note**
>
> **Use this when:** you have generated manifests (from Generate Workflow) and want to apply them to the cluster, or preview what would be applied.

Deploy to Cluster

Apply the generated manifests:

```
l8k deploy
```

By default, `l8k deploy` reads the manifests in `./deployment/` (the output of `l8k generate`) and uses `$KUBECONFIG` for cluster access.

You can also fold the deploy into the generate step by adding `--deploy` to `l8k generate`:

```
l8k generate --fabric ethernet --deployment-type sriov --
multirail --deploy
```

Deployment Ordering

Launch Kit applies manifests in four phases. The first two block on reconciliation because everything else depends on them; the last two batch the remaining manifests so controllers reconcile concurrently:

1. **NicClusterPolicy** — applied first, then awaited until the controller reports `READY`. Per-component `appliedStates[]` are surfaced in the progress output ("ready: 7/12; pending: state-OFED, state-multus-cni, …") so an operator sees exactly which component is the laggard.

2. **NicNodePolicy** — applied per group, awaited per policy (OFED driver, device plugins land here).

3. **Remaining manifests** — every other CR (`SriovNetwork`, `HostDeviceNetwork`, `CIDRPool`, `SpectrumXRailPoolConfig`, …) is applied in a single pass without per-manifest waits, so controllers reconcile concurrently.

4. **Verify** — each manifest from phase 3 is polled until it reaches a terminal state. The four-state classification (READY / IN-PROGRESS / ERROR / MISSING) and per-Kind cross-checks (e.g., the SR-IOV silent-failure case where `syncStatus=Succeeded` but `pfNames` matched zero interfaces) are shared with l8k validate.

Note

Example workload manifests (filenames matching `*example*`) are **not** applied by `l8k deploy`. They are fixtures consumed by the connectivity matrix in `l8k validate --connectivity` (or `l8k deploy --verify`, see below) and are deployed only as part of that ping-matrix phase.

Deploy-wide Timeout

`l8k deploy` has no per-manifest deadline. Pass `--deploy-timeout <duration>` to bound the whole apply + reconciliation phase end-to-end (e.g., `45m`, `2h`). Without the flag, deploy polls until every manifest reaches a terminal state or the user cancels — the right default for SR-IOV configuration on large clusters where a single `SriovNetworkNodePolicy` reconciliation can outlast any small per-manifest budget.

```
l8k deploy --deploy-timeout 90m
```

Dry-Run Mode

Preview what would be deployed without making changes:

```
l8k deploy --dry-run
```

Dry-run mode is recommended before any first deployment and before production changes.

End-to-End Verification (–verify)

Pass `--verify` to chain a full data-plane verification right after a successful apply: `l8k deploy --verify` applies the manifests, waits for every controller to reconcile,

then runs the connectivity ping matrix from l8k validate (apply the example DaemonSet, wait for every pod to be `Ready`, ping every rail across every pod pair, clean up).

```
l8k deploy --verify
```

The matrix's exit code propagates: a failed ping causes `l8k deploy --verify` to exit non-zero even though the manifests applied successfully. Combine with `--keep` (passed to validate) when debugging.

Manual Verification

You can also inspect the deployment manually:

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

Per-group `NicNodePolicy` resources have names matching the group identifier (e.g., `nicnodepolicy-group-0`). Each should reach the `Ready` state before workloads are applied.

If a deployment fails or stalls, see Troubleshooting.

See Also

- Generate Workflow — producing the manifests
- Heterogeneous Clusters — per-group rollout strategies
- Troubleshooting — diagnosing failed deployments
- Automation and CI/CD — exit codes and JSON output for CI

Validate Workflow

Note

Use this when: you've applied a deployment and want to confirm both the control plane (operator and CRs reconciled, component versions match the requested release) and the data plane (pods on different nodes can reach each other on every rail). `l8k validate` runs four checks back-to-back and emits an HTML report alongside the text/JSON output.

Run Validation

```
l8k validate
```

No flags required when defaults apply. `l8k validate` reads `./cluster-config.yaml` (or, when not in the working directory, `<deployment-files>/../cluster-config.yaml`) for `networkOperator.selectedRelease` and the operator namespace, then:

1. Classifies every YAML manifest under `--deployment-files` against the live cluster.
2. Runs the connectivity matrix (enabled by default; pass `--connectivity=false` to skip).
3. Writes an HTML validation report to `<deployment-files>/verify-report.html`.

```
l8k validate --user-config ./cluster-config.yaml \
    --deployment-files ./deployment \
    --kubeconfig ~/.kube/config
```

What It Checks

Network Operator Helm release version

Launch Kit reads any Helm release Secret in the operator namespace whose release name contains `network-operator` (Secret name format `sh.helm.release.v1.<release>.v<N>`, type `helm.sh/release.v1`), parses the chart's `appVersion`, and compares it against the version expected for `networkOperator.selectedRelease` in the user's config (looked up in the embedded release catalog).

The check is **skipped** (with a clear reason in the output) when:

- `cluster-config.yaml` is missing or doesn't declare `networkOperator.selectedRelease`.
- The selected release is not in the embedded catalog.
- No matching Helm release Secret is found in the operator namespace (e.g., the operator was installed via Argo CD or kubectl rather than Helm).

Component versions in NicClusterPolicy and NicNodePolicy

The Helm-release check confirms the *operator* matches the requested release line. Launch Kit also reads the live `NicClusterPolicy` and every `NicNodePolicy` and verifies that each component's `.version` field matches the catalog. This catches drift the Helm check can miss: out-of-band `kubectl edit` changes, partial upgrades, hand-rolled chart values that pinned an older image tag.

Sections checked:

- `NicClusterPolicy` — `nicConfigurationOperator.operator`, `nicConfigurationOperator.configurationDaemon`, `nvIpam`,

`secondaryNetwork.cniPlugins`, `secondaryNetwork.multus`, and `ofedDriver` (when the 26.1 model is used).

- `NicNodePolicy` — `ofedDriver`, `rdmaSharedDevicePlugin`, `sriovDevicePlugin`.

Each row is reported as `MATCH` or `MISMATCH` with the expected and actual versions side by side. A mismatch fails `l8k validate` with exit code 4, the same as a missing manifest or Helm-version mismatch. The check is **skipped** (no impact on exit code) when `selectedRelease` is empty, when the catalog doesn't carry that release line, or when the cluster has no `NicClusterPolicy` (`l8k deploy` hasn't been run yet).

Manifest state

Every YAML document under `--deployment-files` is routed through a per-Kind validator that classifies it as one of four states:

- `READY` — the controller reports the object is fully reconciled. For `NicClusterPolicy` and `NicNodePolicy`, the per-component `appliedStates[]` are folded into a "ready: 12/12; components: …" summary so an operator sees exactly what landed.

- `IN-PROGRESS` — the controller is still reconciling. For `SriovNetworkNodePolicy` and the `NicConfigurationTemplate` / `NicInterfaceNameTemplate` Kinds, per-node and per-device breakdowns are surfaced (the SR-IOV silent-failure case where `syncStatus=Succeeded` but `pfNames` matched zero interfaces is detected and reported as ERROR rather than READY).

- `ERROR` — the controller reported an error state, or a cross-check (expected PFs vs. discovered) failed.

- `MISSING` — the named object is not present in the cluster.

Files matching `*example*` (e.g., `50-example-daemonset.yaml`) are skipped here — those are demo workloads consumed by the connectivity matrix, not part of the network-operator surface `l8k validate` checks for static state.

Connectivity matrix (default ON)

When every manifest is `READY`, Launch Kit applies the example DaemonSet, waits for it to roll out completely (`numberReady == desiredNumberScheduled > 0` — a single `ContainerCreating`-stuck pod fails the wait), enumerates the test pods' rail IPs from each pod's `k8s.v1.cni.cncf.io/network-status` annotation, and runs a ping matrix:

- **Same-rail tests** — every ordered `(srcPod, dstPod)` pair on every rail both pods attach to. Verifies the rail's end-to-end data path.
- **Cross-rail canary** — one ping per ordered pod pair from rail-0 → rail-1 (when both pods have ⩾2 rails). Catches routing misconfigurations that would otherwise be invisible when every rail passes independently.

The DaemonSet is deleted on exit. Pass `--keep` to leave it running for follow-up debugging.

When fewer than two schedulable test pods exist (single-node clusters, pods stuck Pending), the matrix soft-skips with a reason in the report.

Preset deviations

If `cluster-config.yaml` records preset deviations, they're surfaced under each affected node group in the HTML report's **Node groups** section. Each entry shows the field (`pfCount` / `pciAddress` / `deviceID`), the expected value, the discovered value, and a short detail. Deviations are reported for visibility — they do **not** affect the exit code, since the deployment can still run correctly while diverging from the matched preset. See Cluster Topology Presets "Validation and Deviations".

HTML Validation Report

Every `l8k validate` run writes a self-contained HTML report to `<deployment-files>/verify-report.html` (override with `--report-path <file>`, disable with `--report-path=-`). The report is one HTML file with inline CSS — no JavaScript, no external assets — so it can be attached to a ticket, shared in chat, or opened offline.

Report sections:

- **Header** — Launch Kit version, generation timestamp, kubeconfig context, API-server version, operator namespace.

- **Profile** — fabric, deployment type, multirail, and Spectrum-X (version / multiplane mode / number of planes) when enabled. When `cluster-config.yaml` doesn't carry an explicit `profile:` block, the profile is inferred from the Kinds present in the rendered deployment manifests.

- **Node groups** — one card per `clusterConfig[]` entry: identifier, machine type, GPU type, link type, node selector, worker-node list, capability pills, and east-west / north-south PF tables (PCI, device ID, rail, netdev, RDMA device, PSID, part number, NUMA node, connected GPU, GPU proximity). Preset deviations are rendered as a sub-table when present.

- **Cluster nodes** — name, `nvidia.kubernetes-launch-kit.machine` and `.gpu` labels, role.

- **Network Operator release** — selected vs. deployed appVersion. When the component-version cross-check ran, a sub-table lists every `NicClusterPolicy` / `NicNodePolicy` section's `MATCH` / `MISMATCH` against the catalog (expected vs. actual side by side).

- **Manifest state** — one row per CR with a state badge and the validator's reason. Two expandable rows render full-width directly below each manifest row: **Details** (the SR-IOV per-node breakdown, the NCP appliedStates components, etc.) and **Live YAML** (the cluster's current view of the object, `managedFields` stripped, status kept). Manifests with nothing to expand render only the data row.

- **Connectivity matrix** — per-rail `src × dst` grids with pass/fail/skipped color-coded cells, plus the cross-rail canary list.

- **Warnings** — bulleted rollup ("connectivity matrix skipped because cluster has in-progress manifests", "SriovNetworkNodePolicy/rail-0 is in-progress on 2/3 nodes", etc.).

Flag Reference

Flag	Description
`--connectivity`	Run the data-plane ping matrix. Default `true`. Pass `--connectivity=false` to limit validate to the static manifest + Helm-release-version checks.

`--connectivity-timeout`	Wall-clock budget for the connectivity phase (DaemonSet rollout + ping execs). Default `5m`.
`--ping-count`	Number of ICMP echoes per `src dst` pair (`ping -c N`). Default `3`.
`--keep`	Leave the test DaemonSet running after `--connectivity` completes. Useful for follow-up kubectl exec / iperf3 sessions.
`--wait`	Block validate up to this duration waiting for in-progress manifests to reach a terminal state. `0` (default) returns immediately on the first snapshot. Re-polls the cluster every 10 s.
`--report-path`	Write the HTML report to this path. Empty (default) writes to `<deployment-files>/verify-report.html`. Pass `-` to skip the report entirely.
`--user-config`	Cluster config file. Lookup order: explicit path → `./cluster-config.yaml` → `<deployment-files>/../cluster-config.yaml` → `<deployment-files>/cluster-config.yaml`.
`--deployment-files`	Directory containing the manifests to verify (default: `./deployment`).
`--kubeconfig`	Path to kubeconfig (falls back to `$KUBECONFIG`).

Exit Codes

- **0** — every manifest is READY (or all-READY plus IN-PROGRESS without errors) and, when run, the connectivity matrix passed.

- **0 with warning** — at least one manifest is IN-PROGRESS but none are ERROR / MISSING. Connectivity is skipped because pinging an unready cluster would produce noise. Re-run validate later, or use `--wait <duration>` to block.
- **4** — at least one manifest is MISSING or ERROR, the deployed Helm release version doesn't match the selected release, or the connectivity matrix reported failures.
- Other codes — see CLI Reference "Exit Codes".

Sample Output

```
Network Operator release
  selectedRelease: 26.4
  expected version: v26.4.0-beta.9
  deployed: network-operator (chart=26.4.0-beta.9 app=v26.4.0-
beta.9 rev=3 status=deployed)
  result: MATCH
Manifests
  [READY      ] NicClusterPolicy/nic-cluster-policy in (cluster-
scoped) — ready — ready: 12/12; components: cni-plugins, ipoib,
multus, …
  [READY      ] NicNodePolicy/nicnodepolicy-h100 in (cluster-
scoped) — ready
  [READY      ] SriovNetworkNodePolicy/ethernet-sriov-rail-0 in
network-operator — 2/2 nodes ready
Summary: 12/12 ready, 0 in-progress, 0 error, 0 missing; version:
match; preset deviations: 0 group(s)
Connectivity matrix
Rail sriov-network-rail-0:
  src \ dst          c-220-166-240-241  c-220-166-240-242
  c-220-166-240-241  —                    0% 0.1ms
  c-220-166-240-242    0% 0.1ms         —
Rail sriov-network-rail-1:
  src \ dst          c-220-166-240-241  c-220-166-240-242
  c-220-166-240-241  —                    0% 0.1ms
  c-220-166-240-242    0% 0.3ms         —
Cross-rail canary:
  c-220-166-240-241 [sriov-network-rail-0]    c-220-166-240-242
[sriov-network-rail-1]    0% 0.1ms
  c-220-166-240-242 [sriov-network-rail-0]    c-220-166-240-241
[sriov-network-rail-1]    0% 0.1ms
  All 6 ping test(s) passed
HTML report written to /home/user/deployment/verify-report.html
```

For programmatic use, `--output json` emits a single object on stdout with `versionCheck`, `manifests`, `presetDeviations`, `summary`, `connectivity`, and `reportPath` fields. Logs go to stderr.

See Also

- Deploy Workflow — the writer side of the contract validate checks (also supports `--verify` to chain the matrix straight after a successful deploy)
- Troubleshooting — diagnose missing or failing manifests and read the HTML report
- CLI Reference — `l8k validate` flag reference

Advanced Topics

- Heterogeneous Clusters
- Cluster Topology Presets
- Automation and CI/CD
- Troubleshooting

Topic	When to use
Heterogeneous Clusters	Multiple node types — different GPU SKUs, NIC SKUs, or OFED requirements.
Cluster Topology Presets	Generate manifests offline against a known machine type, or use `l8k preset list` / `l8k preset update`.
Automation and CI/CD	Wire `l8k` into a CI/CD pipeline or AI agent — JSON output, exit codes, `l8k schema`.
Troubleshooting	Collect a sosreport and diagnose common failures.

For AI agent integration, see the top-level AI Skills page.

Heterogeneous Clusters

Note

Use this when: your cluster contains multiple node types — e.g., different GPU SKUs (H100 + H200), different NIC SKUs (ConnectX-7 NIC + BlueField-3 SuperNIC), or different OFED requirements.

Supported Configurations

Each configuration shows a discovered cluster shape, the `l8k generate` invocation, and which manifests get rendered.

Two Groups, Different GPUs (`--gpu-type`)

A cluster with two GPU types — H100 and H200, each on its own server SKU. `--gpu-type` filters generation to a single GPU type so a manifest set lands on exactly that subset.

flowchart LR subgraph Generate["l8k generate --gpu-type NVIDIA-H200"] direction TB NCP["NicClusterPolicy"] NNP["NicNodePolicy
nodeSelector: gpu=NVIDIA-H200"] SR["SriovNetwork
nodeSelector: gpu=NVIDIA-H200"] end subgraph Cluster["Kubernetes cluster"] direction TB G1["Group A: DGX-B200 + H100
(filtered out)"] G2["Group B: PowerEdge-XE9680 + H200
(selected)"] end NCP --> G2 NNP --> G2 SR --> G2 classDef included fill:#a2efb6,stroke:#28a745,color:#000 classDef excluded fill:#f0f0f0,stroke:#999,color:#666 class G2,NCP,NNP,SR included class G1 excluded

```
l8k generate --user-config ./cluster-config.yaml \
    --gpu-type NVIDIA-H200 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-h200
```

Drop `--gpu-type` to render manifests covering **both** GPU types: the generator emits a separate per-GPU bundle (each with its own `NicNodePolicy`,

`SriovNetworkNodePolicy`, and `SriovNetwork`) keyed off the `nvidia.kubernetes-launch-kit.gpu` label. The single `NicClusterPolicy` is shared.

Three Groups, Same GPU, Different Servers (`--groups`)

A cluster with three server SKUs all running H200s. Discovery places them in three source groups (one per machineType). They are auto-combined at generation time into a single bundle (see Strategy 1 — Automatic Combination at Generation below). `--groups` narrows the cohort when you need a staged rollout.

The diagram shows the per-source `NicNodePolicy` pairing — each NodePolicy carries its source group's `machine` label and targets exactly its source's nodes. `NicClusterPolicy`, `IPPool`, and `SriovNetwork` are bucket-shared (one CR each, covering all three sources) and are described in the prose below.

flowchart LR subgraph Manifests["Default run: one combined bundle"] direction TB NNP_A["NicNodePolicy A
machine=dgx-b200-..."] NNP_B["NicNodePolicy B
machine=thinksystem-..."] NNP_C["NicNodePolicy C
machine=poweredge-..."] end subgraph Cluster["Kubernetes cluster - all H200 nodes"] direction TB G1["Source A: DGX-B200 + H200"] G2["Source B: ThinkSystem-SR680a-V3 + H200"] G3["Source C: PowerEdge-XE9680 + H200"] end NNP_A --> G1 NNP_B --> G2 NNP_C --> G3

```
# Default: covers all three source groups (auto-combined into one
bundle)
l8k generate --user-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments
# Staged rollout: deploy to two of three first
l8k generate --user-config ./cluster-config.yaml \
    --groups dgx-b200-nvidia-h200,thinksystem-sr680a-v3-nvidia-
h200 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-stage1
```

Same GPU + identical east-west rail count = one bucket. Per-source `NicNodePolicy` CRs use the `nvidia.kubernetes-launch-kit.machine` label; the bucket-level

`IPPool` uses an `In` selector over the source labels (or the shared `.gpu` label when not filtered). The `SriovNetwork` references a single shared `resourceName` that all three NodePolicies register, so any pod scheduled on any of the six nodes gets a VF.

Mixed Cluster (`--gpu-type` and `--groups` Together)

A real-world cluster: two GPU types, multiple server SKUs per type. Discovery produces four source groups arranged as a 2×2 grid (GPU type × machine type). The three commands below show the three common ways to scope a generation.

flowchart TB subgraph Cluster["Kubernetes cluster - 4 source groups"] direction TB subgraph H100["H100 GPUs"] direction LR G1["DGX-B200 + H100"] G2["ThinkSystem + H100"] end subgraph H200["H200 GPUs"] direction LR G3["DGX-B200 + H200"] G4["PowerEdge-XE9680 + H200"] end end

```
# Default: render everything. Two render buckets (H100, H200),
each with
# per-source NicNodePolicy CRs and a bucket-level
SriovNetwork/IPPool.
l8k generate --user-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-all
# Just H200 nodes across both vendors -> selects G3 and G4
l8k generate --user-config ./cluster-config.yaml \
    --gpu-type NVIDIA-H200 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-h200-only
# Just DGX-B200 nodes (one source from each GPU type) -> selects
G1 and G3.
# Strict subset: per-source NodePolicies use machine labels;
bucket-level
# CRs use In: [machine-label-h100-dgx, machine-label-h200-dgx].
l8k generate --user-config ./cluster-config.yaml \
    --groups dgx-b200-nvidia-h100,dgx-b200-nvidia-h200 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-dgx-only
```

The default run is the common path. Use `--gpu-type` when “all nodes of GPU X” is the deployment unit. Use `--groups` when the cohort is enumerated explicitly — e.g., a vendor-specific rollout, or a CI matrix where each cell pins a fixed set of identifiers.

Discovery: Initial Grouping by PCI Topology

`l8k discover` places nodes into groups by their PCI topology — the set of PCI addresses and device IDs across each node’s PFs. Nodes with identical PCI topology land in the same group. Each group carries:

- `machineType` and `gpuType` (e.g., `ThinkSystem-SR680a-V3` / `NVIDIA-H100-NVL`), populated from GPU operator labels or probed from hardware.
- A machine label written to every node in the group: `nvidia.kubernetes-launch-kit.machine: <machineType>-<gpuType>` (sanitised — lowercase, spaces and slashes replaced with hyphens).
- An `identifier` equal to the sanitised label value (e.g., `thinksystem-sr680a-v3-nvidia-h100-nvl`). When `machineType` or `gpuType` couldn’t be resolved, a fallback `group-N` identifier is used and the label is **not** written.
- A `nodeSelector` keyed by the same label: `{nvidia.kubernetes-launch-kit.machine: <value>}`. This is the per-source-group selector — selecting on it picks exactly the nodes in that one group, even when several groups share a GPU type.
- The list of east-west PFs with their `rail` indices.

The resulting groups are written to `cluster-config.yaml` as separate entries. From there you have two strategies for handling more than one group.

Note

The label is l8k state, not GPU operator state. Discovery patches each node with a strategic-merge `client.RawPatch` after the group's `machineType` and `gpuType` are determined. Migrated configs (loaded with old-style differential nodeSelectors from previous l8k versions) keep working unchanged — the label is only written on a fresh discovery run.

Strategy 1 — Automatic Combination at Generation

The groups in `cluster-config.yaml` stay separate — they're written as distinct entries with their own machine-label `identifier`, `nodeSelector`, and PF lists. When `l8k generate` runs, groups that share the same **GPU product type** *and* the same **east-west rail count** are automatically combined into a single render group at generation time:

- The combined render group's `identifier` follows the resource-name convention (lowercase form of the gpuType, e.g., `nvidia-h100-nvl`). Source machine-labels can differ across the merged groups, so a single per-source label can't represent the merged set.
- Its effective `nodeSelector` becomes `{nvidia.kubernetes-launch-kit.gpu: <gpuType>}` — targeting every node of that GPU type, regardless of source machine type. The `.gpu` label is also written by `l8k discover` (alongside the per-source `.machine` label), and its value matches `nvidia.com/gpu.product` verbatim by construction.
- `workerNodes` and OFED-dependent module lists are taken as the union of the source groups.
- One set of manifests is produced that covers all source groups with that GPU type.

The source groups in `cluster-config.yaml` are not mutated; the combination happens in memory while rendering. This is the right strategy when your cluster has multiple node SKUs with the same GPU product type (e.g., two server vendors both running H100s) — the generator produces a single deployment driven by the GPU product label.

When source groups share a GPU type but have **conflicting PCI topologies** (the same PCI address appears at different rail positions across source groups, or rail-to-PCI mappings differ), the generator deploys `NicInterfaceNameTemplate` CRs to normalize interface names via udev rules. Pods then reference rail-stable names (e.g., `rdma_r0`, `eth_r1`) instead of PCI addresses, so a single manifest set works across all source groups despite the underlying PCI differences. See NIC Interface Name Templates below.

Combination happens automatically with no flags required. Groups with different GPU types remain separate; groups without a discoverable GPU type are never combined.

Strategy 2 — Separate Configs per Group

Use this strategy when groups need genuinely different configurations — different Network Operator releases, different fabric, different deployment types, different DOCA driver versions, or different profile selections.

A single `l8k generate` invocation produces one set of manifests with one Network Operator version, one fabric, one deployment type, etc. To deploy different configs to different groups, run `l8k generate` separately for each group (or compatible subset) with `--groups <comma-separated-identifiers>` and a distinct output directory:

```
# First group: SR-IOV Ethernet on Network Operator 26.4
l8k generate --user-config ./cluster-config.yaml \
    --groups dgx-b200-nvidia-h100-nvl \
    --network-operator-release 26.4 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-h100
# Second group: Host Device on Network Operator 26.1
l8k generate --user-config ./cluster-config.yaml \
    --groups poweredge-xe9680-nvidia-h200 \
    --network-operator-release 26.1 \
    --deployment-type host_device --multirail \
    --save-deployment-files ./deployments-h200
```

Each run produces an independent manifest set scoped to its source groups' `nodeSelector`. The two sets can be applied to the same cluster — they target disjoint nodes.

`--groups` accepts a comma-separated list of source-group identifiers (`--groups a,b,c`); identifiers come from `cluster-config.yaml`'s `clusterConfig[].identifier`. Match is case-sensitive; an empty match is a validation error.

Strategy 3 — Filter by GPU Type

For declarative pipelines and CI/CD, `--gpu-type <X>` filters source groups by their `gpuType` (matched case-insensitively):

```
l8k generate --user-config ./cluster-config.yaml \
    --gpu-type NVIDIA-H200 \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments-h200
```

`--gpu-type` and `--groups` are mutually exclusive. `--gpu-type` is the right choice when "deploy this manifest to every node of GPU type X regardless of vendor" is the intent. `--groups` is the right choice when the cohort of source groups must be enumerated explicitly.

What Renders Per Group vs. Per Bucket Under Filter

When `--groups` selects a strict subset of a compatible (gpuType, rail-count) bucket, l8k cannot use a single flat `nodeSelector` to identify exactly that subset (the bucket-level GPU label would over-select). The renderer dispatches each Kubernetes Kind by scope:

Scope	Behaviour under a strict-subset ``–groups`` filter
Cluster-wide (`NicClusterPolicy`)	One CR per cluster, unchanged.
Aggregate (`IPPool` , example `DaemonSet`)	One CR per bucket. `nodeSelectorTerms` emits `In: [machine-label-a, machine-label-b]` covering exactly the filtered sources.

Bucketed (`SriovNetwork`, `CIDRPool`, `HostDeviceNetwork`, `IPoIBNetwork`, `MacvlanNetwork`, `OVSNetwork`)	One CR per bucket, named with the merged-bucket identifier. No node selector — referenced by name from companion CRs and bound to nodes via the kubelet resource that the per-source NodePolicies register.
Simple-selector (`NicNodePolicy`, `SriovNetworkNodePolicy`, `SriovNetworkPoolConfig`, `SpectrumXRailPoolConfig`, `NicConfigurationTemplate`)	One CR per filtered source group, each with its own machine-label flat `nodeSelector`. All N CRs register the same bucket-shared kubelet resource so the Bucketed companions can reference one stable name.
Per-source (`NicInterfaceNameTemplate`)	One CR per source group, always (regardless of merge or filter). PCI addresses are machine-specific and never aggregate.

For default unfiltered runs and `--gpu-type` runs, every bucket selects exactly its full source set, so the Aggregate and Simple-selector CRs render once per bucket against a single flat `nodeSelector` keyed by the bucket's GPU label.

Per-Group Manifest Layout

Within a single `l8k generate` run, manifests split between cluster-wide and per-group resources:

- `10-nicclusterpolicy.yaml` — cluster-wide components (Multus, CNI plugins, NV-IPAM, NIC Configuration Operator, Spectrum-X Operator). Rendered once.
- `11-nicnodepolicy.yaml` — per-group components (OFED driver, device plugins). Rendered once per group with that group's `nodeSelector`.
- `30-*.yaml` — network resources, per-group when needed (e.g., `30-sriovnetworknodepolicy-group-0.yaml`).

The per-group split ensures node selectors are correctly scoped within a single deployment. `imagePullSecrets` set on the NicClusterPolicy propagate into per-group NicNodePolicy sub-specs.

This is a packaging mechanism inside one config — not a way to deploy different driver versions per group. To vary the OFED driver, Network Operator release, or fabric across groups, use Strategy 2.

The deployment ordering ensures dependency resolution: NicClusterPolicy first (wait for readiness), then each NicNodePolicy (wait for readiness), then the remaining manifests.

NIC Interface Name Templates

`NicInterfaceNameTemplate` CRs rename NIC interfaces to predictable, rail-based names using udev rules. Launch Kit deploys them by default for **every** profile and every cluster shape — not only when PCI addresses are ambiguous. The benefit is uniform interface names across server SKUs (`rdma_r0`, `eth_r1`, …): pod manifests, NetworkAttachmentDefinitions, and operator scripts can reference one stable name per rail regardless of which vendor chassis the pod lands on, and reboots that re-enumerate PCI no longer break selectors.

Naming conventions:

- Standard profiles: `rdma_r%rail%`, `eth_r%rail%` (e.g., `rdma_r0`, `eth_r1`).
- Spectrum-X profiles: `roce_p%plane%_r%rail%`, `eth_p%plane%_r%rail%` (e.g., `roce_p0_r2`).

To opt out and reference interfaces by their original PCI addresses instead, set `nicConfigurationOperator.deployNicInterfaceNameTemplate: false` in your `cluster-config.yaml` (or `l8k-config.yaml`). This is rarely needed — the renamed names are a strict superset of what PCI selectors can express.

North-South vs East-West Traffic

During discovery, each PF is classified by traffic direction:

- **East-west** — GPU-to-GPU interconnect (ConnectX, BlueField-3 SuperNIC). Included in generated manifests and assigned a sequential `rail` index.
- **North-south** — management or out-of-band (BlueField DPUs). Saved in `cluster-config.yaml` for visibility but **filtered out** of generated manifests.

The classification uses an embedded list of DPU product codes. BlueField-3 SuperNICs are explicitly classified as east-west even though they share a part-number prefix with

BlueField DPUs. PFs that match no known east-west or north-south signature are treated as out-of-band and excluded from generated manifests.

flowchart LR Mgmt[(Management /
OOB network)] Mesh[(GPU mesh
other cluster nodes)] subgraph Node[Compute node] direction TB BF3[BlueField-3 DPU
north-south] SUPER[ConnectX-7 NIC / BlueField-3 SuperNIC
east-west] GPU[GPUs] GPU --- SUPER end Mgmt <--> BF3 SUPER <-->|RDMA / GPUDirect| Mesh NS_NOTE[N-S PFs are filtered out
of generated manifests] NS_NOTE -.-> BF3 classDef ns fill:#ffd6a5,stroke:#fd7e14,color:#000 classDef ew fill:#a2efb6,stroke:#28a745,color:#000 class BF3 ns class SUPER ew

See Also

- Overview — the node-group / fabric / traffic vocabulary
- Discover Workflow — how groups are formed
- Generate Workflow — `--group` flag in context
- Configuration Reference — `clusterConfig` and `nicConfigurationOperator` schemas

Cluster Topology Presets

> **Note**
>
> **Use this when:** you want to generate manifests without running discovery against a live cluster — for offline manifest generation, CI pipelines, or known machine types where the topology is fixed.

A **preset** is a pre-recorded hardware topology for a known machine type (e.g., `ThinkSystem-SR680a-V3`, `PowerEdge-XE9680`). Each preset records expected PCI addresses, traffic class, NUMA affinity, and rail assignments for a specific machine + GPU pairing.

Launch Kit uses presets in two ways:

- **Discovery overlay** — when discovery detects a matching machine type, the preset's topology is overlaid for consistency.
- **Offline generation** — `l8k generate --for <preset>` skips discovery entirely.

Listing Presets

```
l8k preset list
```

Lists every preset known to the local installation.

Updating Presets

Download the latest presets from a Git repository:

```
l8k preset update
```

Optional flags:

- `--repo <url>` — override the repository URL.
- `--branch <name>` — fetch from a specific branch or tag.
- `--dir <path>` — subdirectory within the repo containing presets.

Presets are optional. Discovery and `--for` generation work without them; presets simply add machine-specific topology defaults.

Offline Manifest Generation

Generate manifests for a known preset without cluster access:

```
l8k generate --for ThinkSystem-SR680a-V3 \
    --node-selector "nvidia.com/gpu.product=NVIDIA-H200" \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments
```

The generated manifests will target nodes matching the supplied selector when applied. This is the recommended path for CI pipelines that produce deployment artefacts ahead of cluster availability.

Discovery Overlay

When `l8k discover` detects a machine type that matches a known preset, the preset's PCI topology, NUMA affinity, and rail assignments are overlaid on the discovery output. This ensures consistent rail numbering and PCI ordering across discovery runs.

If no preset matches, discovery proceeds with the values it derives directly from hardware — see Discover Workflow.

Validation and Deviations

When a matched preset's PFs don't exactly match the discovered hardware, Launch Kit **records the deviations** instead of refusing to apply the preset. There are no fatal validation errors — every discrepancy is soft:

- **PF count mismatch** — the preset and the cluster expose different numbers of PFs.
- **PCI address drift** — a PCI address present in the preset is missing on the cluster, or vice versa.
- **Device-ID drift** — the device ID at a matching PCI address differs from the preset.

Whenever any of these is detected, the matched preset is still applied on a best-effort basis (so rail/NUMA/GPU-affinity fields populate for whichever PFs do line up), and the discrepancies are written to `ClusterConfig.presetDeviation` in `cluster-config.yaml`. Whenever a config with non-empty `presetDeviation` is loaded by any l8k subcommand, a warning is re-emitted listing each deviation. The deployment proceeds, but the operator is reminded each run that the cluster differs from the matched preset.

Part numbers and PSIDs are not strict criteria (firmware variants are expected) so they're not checked.

Example `presetDeviation` block in `cluster-config.yaml`:

```
clusterConfig:
- identifier: "group-0"
  machineType: "PowerEdge-XE9680"
  gpuType: "NVIDIA-H200"
  presetDeviation:
  - field: pfCount
    expected: "10"
    got: "9"
    detail: "PFcountdiffersfrompreset"
  - field: pciAddress
    got: "0000:bd:00.0"
    detail: "discoveredPCIaddressnotpresentinpreset"
  - field: deviceID
    expected: "a2dc@0000:5e:00.0"
    got: "1023@0000:5e:00.0"
    detail: "deviceIDatPCIaddressdiffersfrompreset"
```

See Also

- Generate Workflow — `--for` flag in context
- Discover Workflow — discovery without presets
- CLI Reference — `l8k preset list` / `l8k preset update`

Automation and CI/CD

> **Note**
>
> **Use this when:** you are wiring Launch Kit into a CI/CD pipeline, an automation framework, or an AI agent that needs structured output and exit-code-driven retry logic.

Launch Kit is designed for both interactive use and automated pipelines. Every subcommand supports a JSON output mode, well-defined exit codes, and a programmatic capability discovery endpoint.

JSON Output Mode

Use `--output json` for machine-readable output. In JSON mode, structured data goes to stdout and human-readable logs go to stderr:

```
l8k generate --user-config ./cluster-config.yaml \
    --fabric ethernet --deployment-type sriov --multirail \
    --save-deployment-files ./deployments \
    --output json --yes 2>/dev/null | jq .
```

Non-Interactive Flags

- `--yes` / `-y` — auto-confirm all prompts (implied by `--output json`)
- `--quiet` / `-q` — suppress informational output

Exit Codes

Code	Meaning
0	Success.
1	General error.
2	Validation error (invalid flags or configuration).

3	Cluster error (API unreachable, discovery failed).
4	Deployment error (apply failed).
5	Partial success (discovery completed but deployment failed).

Structured Errors

In JSON mode, errors include structured fields (`code`, `category`, `transient`, `suggestion`) to help automation decide whether to retry or fix input. `transient: true` indicates a temporary condition (e.g., API unreachable) where retry is appropriate; `transient: false` indicates a permanent failure (e.g., invalid flag) requiring input changes.

Schema Discovery

AI agents and automation tools can programmatically discover Launch Kit's capabilities:

```
l8k schema
```

Outputs a JSON description of available commands, phases, fabrics, deployment types, flags, exit codes, and output formats. Tools that need to know what l8k can do (without parsing `--help` text) use this endpoint.

See Also

- AI Skills — driving `l8k` from Claude Code, Cursor, Codex CLI, or any agent
- CLI Reference — complete flag reference
- Troubleshooting — diagnosing failed pipelines

Troubleshooting

SOS-Report Collection

The `l8k sosreport` command collects diagnostic data from the cluster, including pod logs, CRD statuses, OFED diagnostics, and node information:

```
l8k sosreport --kubeconfig ~/.kube/config
```

The output is saved to a directory (default: `./sosreport`) that can be shared for offline analysis.

For the broader Network Operator sosreport workflow (parsing, web UI, what to look for), see Troubleshooting — SOS Report.

Troubleshooting with AI Skills

The `k8s-launch-kit-troubleshoot` skill (see AI Skills) can analyze sosreport data when invoked from any AI agent (Claude Code, Cursor, Codex CLI, or other agents that load Markdown context). Collect a sosreport and then ask the agent to investigate issues such as OFED driver crashes, SR-IOV VF allocation failures, pods stuck in `ContainerCreating`, or NIC configuration errors.

Common Failures

Symptom	Likely Cause	Where to look
`l8k discover` exits with code 3	API server unreachable or RBAC missing	`kubectl auth can-i` and the kubeconfig
Discovery completes with empty `clusterConfig`	Default `--node-selector` excludes all nodes	Pass `--node-selector` matching a label on your nodes (see Discover Workflow)
Generation fails with “RA2.1 requires –network-operator-release in [26.1]”	Spectrum-X version and Network Operator release mismatch	Set `--network-operator-release` to match the Spectrum-X version (see Spectrum-X)

`l8k generate --deploy` exits with code 4	Apply failed; an earlier resource is not Ready	Inspect `kubectl get nicclusterpolicy` and `kubectl get nicnodepolicy`; collect a sosreport
OFED driver pods CrashLoopBackOff after deploy	Storage or third-party RDMA modules block driver reload	Verify `unloadStorageModules` / `unloadThirdPartyRDMAModules` settings in your config (see Discover Workflow)
SR-IOV pods stuck in `ContainerCreating`	VF allocation failure or device plugin not ready	`kubectl describe pod` and SR-IOV operator logs

See Also

- Troubleshooting — SOS Report — the upstream sosreport workflow for the Network Operator
- AI Skills — the `k8s-launch-kit-troubleshoot` skill
- Automation and CI/CD — exit codes and structured errors

Reference

- CLI Reference
- Configuration Reference

Lookup-friendly references for flags and config schema.

Reference	Contents
CLI Reference	Every flag, every subcommand, every exit code.
Configuration Reference	The `cluster-config.yaml` / `l8k-config.yaml` schema, section by section.

CLI Reference

The Launch Kit binary is invoked as `l8k` and exposes a set of subcommands. `l8k --help` and `l8k <subcommand> --help` print labelled flag groups identical to the sections below.

Global Flags

Available on all subcommands:

Flag	Description
`--output <string>`	Output format: `text` (default, human-readable) or `json` (structured for automation, see Automation and CI/CD).
`-y, --yes`	Auto-confirm all prompts. Implied by `--output json`.
`-q, --quiet`	Suppress informational output.
`--log-level <string>`	Log level: `debug`, `info`, `warn`, `error`.
`--log-file <string>`	Write logs to file instead of stderr.
`-h, --help`	Show help.

l8k discover

Probes cluster hardware and writes a `cluster-config.yaml`.

Flag	Description
`--kubeconfig <string>`	Path to kubeconfig file (falls back to `$KUBECONFIG`).
`--user-config <string>`	Base config to merge with discovered hardware. If `--save-cluster-config` is omitted, the file is updated in place.

`--save-cluster-config <string>`	Output path for the discovered cluster configuration. Defaults to `--user-config` if set, otherwise `./cluster-config.yaml`.
`--network-operator-namespace <string>`	Override the Network Operator namespace (default: `nvidia-network-operator`).
`--network-operator-release <string>`	Pin discovery to a Network Operator release line. Supported: `25.10`, `26.1`, `26.4`.
`--node-selector <string>`	Filter nodes for discovery by label. Default: `feature.node.kubernetes.io/pci-15b3.present=true` (Mellanox NICs).
`--image-pull-secrets <strings>`	Image pull secret names for NicClusterPolicy (comma-separated).
`--enabled-plugins <string>`	Comma-separated list of plugins to enable (default: `network-operator`).

l8k generate

Renders Kubernetes manifests from a cluster configuration and a profile selection. Optionally deploys them.

Profile Selection

Flag	Description
`--user-config <string>`	Cluster configuration file. Auto-detected from `./cluster-config.yaml` (or the installed default) if omitted.
`--for <preset>`	Generate for a hardware preset without running discovery. Requires `--node-selector`. See Cluster Topology Presets.
`--fabric <string>`	Fabric type: `ethernet` or `infiniband`. Auto-defaults from the cluster's unanimous `linkType` (Unit 5 fabric probe) when not supplied.

`--deployment-type <string>`	Deployment type: `sriov`, `rdma_shared`, or `host_device`. Auto-defaults to `sriov` when not supplied.
`--multirail`	Enable multirail deployment. Auto-defaults to `true`. Opt out with `--multirail=false` (YAML cannot express explicit-false).
`--spectrum-x <RA-version>`	Enable Spectrum-X. Value is the SPC-X RA version: `RA2.1` or `RA2.2`. Implies ethernet fabric, sriov deployment, and multirail.
`--multiplane-mode <string>`	Multiplane mode: `swplb`, `hwplb`, `uniplane`, or `none`. Required with `--spectrum-x`; auto-defaulted from east-west PF deviceID when omitted (CX7/BF3 → `uniplane`, CX8 → `swplb`, CX9 → `hwplb`). `none` requires `--number-of-planes 1`.
`--number-of-planes <int>`	Number of planes: `1`, `2`, or `4`. Required with `--spectrum-x`; auto-defaulted from deviceID (CX7/BF3 → 1, CX8 → 2, CX9 → 4).
`--network-operator-release <string>`	Pin to a Network Operator release line. Supported: `25.10`, `26.1`, `26.4`. Auto-defaulted under `--spectrum-x` (`RA2.1` → `26.1`; `RA2.2` → `26.4`). See Overview.
`--groups <a,b,...>`	Restrict output to the named source groups (comma-separated, matched case-sensitively against `clusterConfig[].identifier`). Mutually exclusive with `--gpu-type`. Empty match is a validation error. See Heterogeneous Clusters.
`--gpu-type <string>`	Restrict output to source groups whose `gpuType` matches (case-insensitive). Mutually exclusive with `--groups`. See Heterogeneous Clusters.

`--node-selector <string>`	Node selector for the synthesized `clusterConfig` when `--for` is used (e.g., `key=value,key2=value2`). Required with `--for`.

Output and Deployment

Flag	Description
`--save-deployment-files <string>`	Output directory for generated YAML files (default: `./deployment`).
`--deploy`	Apply generated manifests to the cluster.
`--dry-run`	Preview deployment without applying (requires `--deploy`).
`--kubeconfig <string>`	Path to kubeconfig (required with `--deploy`; falls back to `$KUBECONFIG`).

Customization

Flag	Description
`--enable-doca-driver`	Enable DOCA driver deployment.
`--workload-manifest <string>`	Path to a custom workload manifest. Launch Kit patches network annotations, resource requests, and node affinity. See Generate Workflow.
`--image-pull-secrets <strings>`	Image pull secret names for NicClusterPolicy (comma-separated). Propagates to per-group NicNodePolicy sub-specs.
`--network-operator-namespace <string>`	Override the Network Operator namespace.
`--pod-namespace <string>`	Namespace for pods and network resources.
`--enabled-plugins <string>`	Comma-separated list of plugins to enable (default: `network-operator`).

l8k deploy

Applies previously generated manifests to the cluster in four phases: NicClusterPolicy first (await ready), per-group NicNodePolicy (await each), all remaining CRs in one batch (controllers reconcile concurrently), then verify every manifest reached a terminal state. Example workload manifests (`*example*`) are skipped here — they're applied by `l8k validate --connectivity` / `l8k deploy --verify` as part of the data-plane phase. See Deploy Workflow for the full deployment-ordering description.

Flag	Description
`--kubeconfig <string>`	Path to kubeconfig (falls back to `$KUBECONFIG`).
`--deployment-files <string>`	Directory containing the manifests to apply (default: `./deployment`).
`--deploy-timeout <duration>`	End-to-end wall-clock budget for the apply + reconciliation phase (e.g., `45m`, `2h`). `0` (default) means no deadline; polls until every manifest reaches a terminal state. Useful for matching a maintenance window when SR-IOV reconciliation on a large cluster can take an hour or more.
`--verify`	After a successful apply, chain the connectivity matrix (same flow as `l8k validate --connectivity`): apply the example DaemonSet, wait for it to roll out, run a ping matrix across every rail and pod pair, then clean up.
`--dry-run`	Preview what would be applied without changing the cluster.

l8k validate

Verifies a deployment by running four checks back-to-back: Network Operator Helm release version (compared against `networkOperator.selectedRelease` in `cluster-config.yaml`); per-component version cross-check that walks the live `NicClusterPolicy` + `NicNodePolicy` and confirms each section's `.version` field matches the catalog (catches out-of-band edits and partial upgrades the Helm check misses); per-manifest state classification (READY / IN-PROGRESS / ERROR / MISSING via the per-Kind validator registry); and a data-plane connectivity matrix (apply the example

DaemonSet, ping every rail across every pod pair). Writes a self-contained HTML report to `<deployment-files>/verify-report.html` by default. Exits 4 on any missing/error manifest, Helm-version mismatch, component-version mismatch, or connectivity-matrix failure. See Validate Workflow for full details.

Flag	Description
`--kubeconfig <string>`	Path to kubeconfig (falls back to `$KUBECONFIG`).
`--user-config <string>`	Cluster config file. Lookup order: explicit path → `./cluster-config.yaml` → `<deployment-files>/../cluster-config.yaml` → `<deployment-files>/cluster-config.yaml`. Read for `networkOperator.selectedRelease` and operator namespace.
`--deployment-files <string>`	Directory containing the manifests to verify (default: `./deployment`).
`--connectivity`	Run the data-plane ping matrix. Default `true`. Pass `--connectivity=false` to limit validate to the static manifest + Helm-release-version checks.
`--connectivity-timeout <duration>`	Wall-clock budget for the connectivity phase (default: `5m`).
`--ping-count <int>`	Number of ICMP echoes per `src dst` pair (`ping -c N`). Default `3`.
`--keep`	Leave the test DaemonSet running after `--connectivity` completes (useful for follow-up debugging).
`--wait <duration>`	Block validate up to this duration waiting for in-progress manifests to reach a terminal state. `0` (default) returns immediately on the first snapshot. Re-polls the cluster every 10 s.

`--report-path <string>`	Write the HTML report to this path. Empty (default) writes to `<deployment-files>/verify-report.html`. Pass `-` to skip the report file entirely.

l8k preset list

Lists locally available cluster topology presets.

```
l8k preset list
```

l8k preset update

Downloads the latest presets from a Git repository.

Flag	Description
`--repo <string>`	Git repository URL.
`--branch <string>`	Branch or tag to fetch from (default: `main`).
`--dir <string>`	Subdirectory within the repo containing presets.

See Cluster Topology Presets for usage.

l8k sosreport

Collects diagnostic data from the cluster.

Flag	Description
`--kubeconfig <string>`	Path to kubeconfig (falls back to `$KUBECONFIG`).
`--output-dir <string>`	Directory to save the sosreport (default: `./sosreport`).

See Troubleshooting for analysis guidance.

l8k schema

Outputs a JSON description of l8k's capabilities — supported commands, fabrics, deployment types, exit codes, and output formats. Used by AI agents and automation tooling to discover `l8k` capabilities programmatically. See Automation and CI/CD.

```
l8k schema
```

l8k version

Prints the version. Supports `--output json` for structured output.

Exit Codes

Code	Meaning
0	Success.
1	General error.
2	Validation error (invalid flags or configuration).
3	Cluster error (API unreachable, discovery failed).
4	Deployment error (apply failed).
5	Partial success (discovery completed but deployment failed).

See Also

- Configuration Reference — `l8k-config.yaml` schema
- Automation and CI/CD — JSON output and exit-code-driven retries
- Overview — the meaning behind the flags

Configuration Reference

The Launch Kit configuration file (typically `cluster-config.yaml`, produced by `l8k discover` and consumed by `l8k generate`) is YAML. This page documents every top-level section.

Full Schema

```
networkOperator:
  selectedRelease: "26.4"
  version: v26.4.1
  componentVersion: network-operator-v26.4.1
  repository: nvcr.io/nvidia/cloud-native
  namespace: nvidia-network-operator
  imagePullSecrets: []
  docsBaseURL:
https://docs.nvidia.com/networking/display/kubernetes2610
docaDriver:
  enable: true
  version: doca3.4.1-26.04-1.1.0.0-1
  unloadStorageModules: false
  enableNFSRDMA: false
  unloadThirdPartyRDMAModules: false
  skipPreflightChecks: false
nvIpam:
  poolName: nv-ipam-pool
  startingSubnet: "192.168.2.0"
  mask: 24
  offset: 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
nicConfigurationOperator:
  deployNicInterfaceNameTemplate: true
  rdmaPrefix: "rdma_r%rail%"
  netdevPrefix: "eth_r%rail%"
spectrumX:
  nicType: "1023"
  overlay: "none"
  rdmaPrefix: "roce_p%plane%_r%rail%"
  netdevPrefix: "eth_p%plane%_r%rail%"
workload:
  manifest: ""
profile:
  fabric: ethernet
  deployment: sriov
  multirail: false
  spectrumX:
    spcxVersion: "RA2.1"
    multiplaneMode: swplb
    numberOfPlanes: 4
  ai: false
clusterConfig:
- identifier: "dgx-b200-nvidia-b200"
  machineType: "DGX-B200"
  productType: "NVIDIA-B200"
  capabilities:
    nodes:
      sriov: true
      rdma: true
      ib: false
  workerNodes: ["worker-0", "worker-1"]
  nodeSelector:
    nvidia.kubernetes-launch-kit.machine: "DGX-B200-NVIDIA-B200"
  thirdPartyRDMAModules: []
```

```
  storageModules: []
  linkType: Ethernet
  pfs:
  - deviceID: "1023"
    pciAddress: "0000:05:00.0"
    rdmaDevice: "mlx5_0"
    networkInterface: "net1"
    traffic: east-west
    rail: 0
```

networkOperator

Network Operator version, image registry, namespace, and pull secrets.

Field	Description
`selectedRelease`	Pin to a release line. Supported: `25.10`, `26.1`, `26.4`. Auto-fills `version` and image tags from an embedded catalog. Equivalent to the `--network-operator-release` flag.
`version`	Explicit Network Operator version. Overrides the catalog when set.
`componentVersion`	Tag for component images (CNI, device plugins, etc.).
`repository`	Container registry (default: `nvcr.io/nvidia/mellanox`).
`namespace`	Operator namespace (default: `nvidia-network-operator`).
`imagePullSecrets`	List of secret names. Propagated to `NicClusterPolicy.spec.global.imagePullSecrets` and per-group `NicNodePolicy` sub-specs.
`docsBaseURL`	Documentation URL embedded in generated annotations.

docaDriver

OFED driver configuration and kernel driver dependencies validation.

Field	Description
`enable`	Include the OFED driver in generated manifests. Set to `false` to skip (or use `--enable-doca-driver` to flip).
`version`	DOCA driver version tag.
`unloadStorageModules`	Unload storage-over-RDMA modules (`nvme_rdma`, `ib_isert`, `rpcrdma`, ...). Auto-set to `true` during discovery if such modules are detected.
`unloadThirdPartyRDMAModules`	Unload third-party RDMA modules (`rdma_rxe`, `qedr`, `bnxt_re`, ...). Auto-set to `true` during discovery if such modules are detected. Storage and third-party module lists are sourced from the `doca-driver-build` project.
`enableNFSRDMA`	Enable NFS-over-RDMA support.
`skipPreflightChecks`	Skip the kernel driver dependencies validation. Useful for environments where it's known-good.

See Discover Workflow for how OFED-dependent modules are detected.

nvIpam

NV-IPAM configuration. Either provide an explicit `subnets` list or let Launch Kit auto-generate non-overlapping subnets per node group.

Field	Description
`poolName`	Pool name used in IPPool CRs.
`subnets`	Explicit list of `{subnet, gateway}` entries. Mutually exclusive with the auto-generation fields.
`startingSubnet`	First subnet for auto-generation (e.g., `192.168.2.0`).

`mask`	Prefix length for auto-generated subnets.
`offset`	Increment used between auto-generated subnets.

sriov / hostdev / rdmaShared / ipoib / macvlan

Profile-specific parameters — only the section for the selected profile is consumed.

Section	Field	Description
`sriov`	`ethernetMtu` / `infinibandMtu`	MTU values per fabric.
`sriov`	`numVfs`	Number of virtual functions per PF.
`sriov`	`priority`	SriovNetworkNodePolicy priority.
`sriov`	`resourceName` / `networkName`	Kubernetes resource and network names.
`hostdev`	`resourceName` / `networkName`	Kubernetes resource and network names for host-device.
`rdmaShared`	`resourceName`	Kubernetes resource name.
`rdmaShared`	`hcaMax`	Maximum HCAs per host (soft limit).
`ipoib`	`networkName`	IPoIB network name.
`macvlan`	`networkName`	MacVLAN network name.

nicConfigurationOperator

Controls when NIC interface names are templated by the NIC Configuration Operator.

Field	Description

`deployNicInterfaceNameTemplate`	“Enable when needed”. Templates are deployed when groups have cross-rail PCI conflicts or when names are otherwise ambiguous. See Heterogeneous Clusters.
`rdmaPrefix`	RDMA device naming template (default: `rdma_r%rail%`).
`netdevPrefix`	Netdev naming template (default: `eth_r%rail%`).

spectrumX

Spectrum-X-specific settings.

Field	Description
`nicType`	NIC type device ID. `1021` = ConnectX-7 NIC; `1023` = ConnectX-8 SuperNIC; `a2dc` = BlueField-3 SuperNIC.
`overlay`	Overlay mode.
`rdmaPrefix`	RDMA device naming template with `%plane%` and `%rail%` substitutions.
`netdevPrefix`	Netdev naming template with `%plane%` and `%rail%` substitutions.

workload

Field	Description
`manifest`	Path to a custom workload manifest. When set, Launch Kit patches it with network annotations, resource requests, and node affinity instead of generating an example DaemonSet. See Generate Workflow.

profile

Profile selection (also overridable via CLI flags).

Field	Description
`fabric`	`ethernet` or `infiniband`.
`deployment`	`sriov`, `rdma_shared`, or `host_device`.
`multirail`	Enable multirail.
`spectrumX.spcxVersion`	Spectrum-X reference architecture (`RA2.1` or `RA2.2`).
`spectrumX.multiplaneMode`	Multiplane mode: `hwplb`, `swplb`, `uniplane`, `none`.
`spectrumX.numberOfPlanes`	Number of planes.

clusterConfig

Discovered node groups, populated by `l8k discover`. Each entry describes one group.

Field	Description
`identifier`	Sanitised `<machineType>-<gpuType>` (e.g. `dgx-b200-nvidia-h100-nvl`) when both fields are resolved; `group-0` / `group-1` fallback when they aren't. Used as the NicNodePolicy / SriovNetworkNodePolicy name suffix.
`machineType` / `productType`	Hardware type strings (e.g., `DGX-B200` / `NVIDIA-B200`).
`capabilities.nodes.sriov` / `rdma` / `ib`	Boolean flags reflecting hardware capability.
`workerNodes`	List of node names in this group.

`nodeSelector`	Per-group selector. After `l8k discover`, this is `{nvidia.kubernetes-launch-kit.machine: <machineType>-<gpuType>}` — a label discovery writes onto every node in the group. When `l8k generate` auto-merges groups sharing a GPU type, the merged group falls back to `{nvidia.kubernetes-launch-kit.gpu: <gpuType>}` instead (different source machineTypes can't share a single machine label). Discovery writes both labels onto every node, so the merged selector has a value to bind to. Configs from earlier l8k versions with old-style differential nodeSelectors are preserved as-is.
`thirdPartyRDMAModules` / `storageModules`	OFED-dependent modules detected on the group.
`presetApplied`	`true` when a topology preset matched `(machineType, gpuType)` and was applied.
`presetDeviation`	List of field-level discrepancies between the matched preset and discovered hardware. Non-empty means the preset was applied but the cluster differs from the preset. Each entry has `field` (`pciAddress` / `deviceID`), `expected` , `got` , and `detail` . See Cluster Topology Presets "Validation and Deviations".

`linkType`	The discovered fabric for the group: `Ethernet` or `InfiniBand`. Populated by the fabric probe only when **every** east-west port produces a confirmed verdict (port ACTIVE plus, for IB, a subnet manager is present) and they agree. When omitted, discovery couldn't prove the cluster's fabric — downstream code should treat the absence as "unknown". See Discover Workflow "Fabric Type Detection".
`pfs`	List of physical functions. Each entry has `deviceID`, `pciAddress`, `rdmaDevice`, `networkInterface`, `traffic` (`east-west` or `north-south`), and `rail` (sequential index for east-west PFs).

North-south PFs are listed for visibility but filtered out of generated manifests. See Overview and Discover Workflow.

See Also

- CLI Reference — flag reference
- Discover Workflow — how this file is produced
- Generate Workflow — how this file is consumed

AI Skills

NVIDIA ships a set of **AI agent skills** that drive Network Operator configuration on Kubernetes through the deterministic `l8k` CLI (Kubernetes Launch Kit). The skills are plain Markdown files with light YAML frontmatter — agent-agnostic by design. They give an LLM agent the vocabulary, decision tree, and command idioms to discover cluster hardware, pick a deployment profile, render manifests, deploy them, and triage failures, without the operator having to memorize `l8k` flags or NVIDIA networking conventions.

Skills do not embed an LLM. `l8k` itself remains a self-contained binary — it is fully usable without any AI agent. The skills are an *interface layer* that lets agents call `l8k` correctly.

What Skills Provide

Each skill is a self-contained Markdown document in the k8s-launch-kit repository under `skills/`. The bundled set:

Skill	Purpose
`k8s-network-engineer`	Top-level persona. Embodies a senior NVIDIA networking engineer; activates on any Kubernetes networking question (SR-IOV, RDMA, Spectrum-X, BlueField, ConnectX, DOCA, multirail). Loads every utility skill below.
`k8s-launch-kit-shared`	Shared CLI patterns: install paths, global flags, output modes, exit-code semantics. Required by every other skill.
`k8s-launch-kit-discover`	Wraps `l8k discover`: cluster hardware probing, label patching, `cluster-config.yaml` interpretation.

`k8s-launch-kit-config`	Reads / explains / edits `cluster-config.yaml` and `l8k-config.yaml` — profile fields, group identifiers, OFED module lists.
`k8s-launch-kit-generate`	Wraps `l8k generate`: profile selection (fabric × deployment-type), Spectrum-X cohort flags, group filters, hardware-derived defaults.
`k8s-launch-kit-dryrun`	Wraps `l8k generate --dry-run`: previews what would land on the cluster without applying.
`k8s-launch-kit-deploy`	Wraps `l8k deploy`: applies pre-generated manifests in four phases (NicClusterPolicy → NicNodePolicy → remaining batch → verify reconciliation). Supports `--verify` to chain the data-plane connectivity matrix straight after a successful apply, and `--deploy-timeout` to bound the whole run end-to-end.
`k8s-launch-kit-validate`	Wraps `l8k validate`: classifies every manifest's cluster state (READY / IN-PROGRESS / ERROR / MISSING), runs a connectivity ping matrix between the example DaemonSet's pods on every rail (default ON; `--connectivity=false` to skip), and writes an HTML validation report to `<deployment-files>/verify-report.html`.
`k8s-launch-kit-pipeline`	End-to-end orchestration (discover → generate → deploy) for greenfield clusters.

`k8s-launch-kit-troubleshoot`	Wraps `l8k sosreport` plus a triage checklist for OFED crashes, SR-IOV VF allocation failures, `ContainerCreating`-stuck pods, and NIC firmware misconfiguration.

The skills know which `l8k` flags exist on which subcommand, when `--dry-run` is required, when `--multirail` should auto-default, which Spectrum-X RA pairs with which Network Operator release, and how to interpret a sosreport. They tell the agent to **start every deployment task with `l8k discover` and every troubleshooting task with `l8k sosreport`** — not with raw `kubectl`.

Installation

The skills are agent-agnostic Markdown. The right install location depends on which AI agent you're using.

All paths assume the repository has been cloned:

```
git clone https://github.com/NVIDIA/k8s-launch-kit.git
cd k8s-launch-kit
```

Claude Code

Claude Code discovers skills under `~/.claude/skills/` (user-scoped) or `<project>/.claude/skills/` (project-scoped):

```
# User-scoped (available in every Claude Code session)
mkdir -p ~/.claude/skills
ln -s "$(pwd)/skills/"k8s-launch-kit-* ~/.claude/skills/
ln -s "$(pwd)/skills/k8s-network-engineer" ~/.claude/skills/
# Or project-scoped (available only in this project)
mkdir -p .claude/skills
ln -s "$(pwd)/skills/"k8s-launch-kit-* .claude/skills/
ln -s "$(pwd)/skills/k8s-network-engineer" .claude/skills/
```

Verify by typing `/skills` in a Claude Code session — the `k8s-launch-kit-*` and `k8s-network-engineer` entries should appear.

Cursor

Cursor reads project-scoped rule files from `.cursor/rules/`. Symlink the skill directories or copy the `SKILL.md` files into `.mdc` rules:

```
# In your Kubernetes / l8k project
mkdir -p .cursor/rules
for skill in <path-to-k8s-launch-kit>/skills/*/SKILL.md; do
    name=$(basename "$(dirname "$skill")")
    cp "$skill" ".cursor/rules/${name}.mdc"
done
```

The YAML frontmatter on each `SKILL.md` is compatible with Cursor's rule metadata (`description`, `alwaysApply`-equivalent semantics handled by activation prompts). For user-wide rules, place them under `~/.cursor/rules/` instead.

OpenAI Codex CLI

Codex CLI loads project-level instructions from an `AGENTS.md` file at the repository root. Concatenate the persona skill plus its dependencies into `AGENTS.md`:

```
# In your Kubernetes / l8k project
{
    echo "# Agent Instructions"
    echo
    cat <path-to-k8s-launch-kit>/skills/k8s-network-
engineer/SKILL.md
    for skill in <path-to-k8s-launch-kit>/skills/k8s-launch-kit-
*/SKILL.md; do
        echo
        echo "---"
        echo
        cat "$skill"
    done
} > AGENTS.md
```

For machine-wide instructions, write to `~/.codex/AGENTS.md` instead. Codex CLI will pick up the file automatically on each `codex` invocation.

Other Agents

Any agent that supports loading external Markdown context (Continue.dev, Aider, custom MCP servers, etc.) can use the skills. Two integration patterns work:

- **As a system / project prompt**: concatenate `skills/k8s-network-engineer/SKILL.md` plus the `k8s-launch-kit-*/SKILL.md` files into the agent's persistent context.
- **As MCP server resources**: serve the `skills/` directory through any filesystem MCP server; the agent reads the relevant skill on demand.

The skill content is the contract. Frontmatter (`name`, `description`, `metadata.requires`) is metadata for skill-aware agents and is safely ignored by agents that don't parse it.

Real-World Scenarios

The examples below show prompts an operator would type in any of the agents above, plus the actions the skills would orchestrate.

Discovery on a Heterogeneous Cluster

Operator prompt:

"I have a new cluster with mixed DGX-B200 and Lenovo ThinkSystem nodes, both running H100 GPUs. The GPU operator is installed. Help me discover the network topology and tell me what I'm working with."

Skill-driven flow:

1. `k8s-network-engineer` activates and loads the discovery skills.
2. The agent runs `l8k discover --output json 2>/dev/null` and parses the result.
3. `k8s-launch-kit-config` interprets the resulting `cluster-config.yaml`: identifies two source groups (`dgx-b200-nvidia-h100-nvl` and `thinksystem-sr680a-v3-nvidia-h100-nvl`), confirms east-west PFs are ConnectX-7 (deviceID `1021`), counts 8 rails per node, and notes that both groups share a `railNumber` — so `--multirail` will auto-default to true.
4. The agent reports back: GPU type, NIC topology, fabric verdict (`linkType: Ethernet` confirmed via active port + `link_layer`), discovered OFED-dependent kernel modules (any storage modules trigger an "unloadStorageModules will be enabled" warning), and any preset deviations.
5. The agent suggests the next step: *"Both groups share GPU type and rail count, so a single ``l8k generate`` will produce one combined bundle covering both. Want me to render manifests for SR-IOV Ethernet?"*

The operator never reads `cluster-config.yaml` directly — the skill explains the contents in natural language and surfaces the decisions that matter.

Choosing a Deployment Profile (Spectrum-X)

Operator prompt:

“This cluster is going to run a Spectrum-X AI cloud. We have ConnectX-8 east-west NICs and we’re targeting Network Operator 26.4. Generate the manifests.”

Skill-driven flow:

1. `k8s-launch-kit-generate` recognizes Spectrum-X intent.
2. The skill verifies the cohort: ConnectX-8 → `--multiplane-mode swplb` and `--number-of-planes 2` are the hardware defaults; RA2.2 pairs with Network Operator 26.4.
3. Before running `l8k generate`, the skill asks the operator to confirm the **switch-side** Spectrum-X fabric setup is in place (Spectrum-4 switches with the matching configuration) — because `l8k` does not handle the switch side and a misconfigured fabric is the most common Spectrum-X failure mode.
4. Once confirmed, the agent runs:

```
l8k generate \
    --spectrum-x RA2.2 \
    --network-operator-release 26.4 \
    --save-deployment-files ./deployments-spectrum-x \
    --output json 2>/dev/null
```

5. The agent reports the auto-defaulted flags (“`Defaulted --multiplane-mode=swplb (ConnectX-8 deviceID 1023)`”, “`Defaulted --number-of-planes=2`”) and recommends `l8k generate --dry-run` before applying.

The skill knows the per-deviceID multiplane defaults, the RA-to-release pairing, and the switch-side prerequisite — the operator only needs to state intent.

Triaging a Stuck Deployment

Operator prompt:

“My OFED driver pods have been stuck in CrashLoopBackOff for the last hour. Some of my GPU workload pods are stuck in ContainerCreating. Figure out what’s wrong.”

Skill-driven flow:

1. `k8s-launch-kit-troubleshoot` activates.
2. The agent runs `l8k sosreport --output-dir ./sosreport` first — one command captures all CRDs, operator logs, per-node NIC info, and module status. The skill explicitly tells the agent **not** to start with raw `kubectl logs` until the sosreport bundle is read.
3. The skill walks the triage checklist against the bundle:
   - Is the right Network Operator release installed? (`l8k validate` against `cluster-config.yaml`)
   - Are storage / third-party RDMA modules loaded that would block MOFED reload? (`unloadStorageModules` / `unloadThirdPartyRDMAModules` flags)
   - Does the firmware preflight check pass on every node?
   - Are there NicNodePolicy resources for every group, with matching `nodeSelector`?
   - Are any pods scheduled on nodes that don’t carry the `nvidia.kubernetes-launch-kit.machine` label?
4. The agent finds the root cause — e.g. *“The OFED container is being killed by the kernel because ``nvme_rdma`` was loaded by an early-boot service and is holding ``mlx5_ib``. ``unloadStorageModules`` is set to ``false`` in the rendered NicNodePolicy. Re-run ``l8k discover`` to re-detect storage modules and apply, or set ``docaDriver.unloadStorageModules: true`` manually.”*
5. The agent offers to apply the fix and watch the rollout.

The skill turns “the cluster is broken” into a structured investigation that always starts from the same evidence baseline (the sosreport).

What Skills Don’t Do

- **No autonomous deploys.** Skills are designed to recommend `--dry-run` before any `kubectl apply` / `l8k deploy` and surface the exact command for operator approval. The operator stays in the loop.
- **No switch-side configuration.** Spectrum-X fabric setup, BGP / EVPN, and physical cabling are out of scope. The skills will warn before suggesting Spectrum-X profiles.
- **No replacement for review.** Skills produce the *most likely* configuration based on discovered hardware. For production rollouts, the operator should still inspect the rendered `./deployments/*.yaml` and `cluster-config.yaml` before applying.

See Also

- Configuration Assistance with Kubernetes Launch Kit — the underlying CLI
- Automation and CI/CD — using `l8k` from scripts and pipelines without an agent
- Troubleshooting — the underlying `l8k sosreport` workflow
- Deployment Profiles — the decision matrix the skills consult

Customization Options

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

            - NicNodePolicySpec
            - NicNodePolicyStatus
            - NicPolicyCR
            - OFEDDriverSpec
            - PodProbeSpec
            - ResourceRequirements
            - SecondaryNetworkSpec
            - SpectrumXOperatorSpec
            - `string`
            - WaitForCompletionSpec
- NicClusterPolicy Custom Resource Example
- Heterogeneous Clusters with NicNodePolicy
    - When to Use NicNodePolicy
    - How NicClusterPolicy and NicNodePolicy Work Together
        - Overview
        - NCP-Only Deployment Flow
        - NCP + NNP Deployment Flow
    - DOCA-OFED Driver Upgrades with NicNodePolicy
    - Deployment Rules and Restrictions
        - Section Exclusivity
        - Node Selector Overlap Prevention
        - Naming Requirements
        - Prerequisites
    - Verifying the Deployment

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 driver-upgrade”,”useDrainControlle*
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-v26.4.1”*
operator.repository	string	*“nvcr.io/nvidia/cloud-native”*

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

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-v26.4.1”*

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

node-feature-discovery.postDeleteCleanup	bool	*false*

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

SR-IOV Network Operator

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

Name	Ty
sriov-network-operator.images.ibSriovCni	stri
sriov-network-operator.images.operator	stri
sriov-network-operator.images.ovsCni	stri
sriov-network-operator.images.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.sriovDraDriver	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, NicNodePolicyStatus)

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, NicNodePolicySpec)

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)

GlobalConfig

(*Appears on:* NicClusterPolicySpec)

GlobalConfig contains global configuration for all components

Field	Description
`repository` string	*(Optional)* Repository is the default container image repository for all components
`version` string	*(Optional)* Version is the default version tag for all component images
`imagePullSecrets` []string	*(Optional)* ImagePullSecrets is a list of secret names for pulling component images

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	*(Optional)* Address of the registry that stores the image
`version` string	*(Optional)* 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

`global` GlobalConfig	*(Optional)* Global contains global configuration for all co
`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

NicNodePolicy

NicNodePolicy is the Schema for the NicNodePolicies API

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

NicNodePolicySpec

(*Appears on:* NicNodePolicy)

NicNodePolicySpec defines the desired state of NIC drivers and device plugin

Field	Description
`ofedDriver` OFEDDriverSpec	OFEDDriver is a specialized driver for NVIDIA NICs which can replace the inbox driver that comes with an OS. See https://network.nvidia.com/support/mlnx-ofed-matrix/
`rdmaSharedDevicePlugin` DevicePluginSpec	RdmaSharedDevicePlugin manages support IB and RoCE HCAs through the Kubernetes device plugin framework. The config field is a json representation of the RDMA shared device plugin configuration. See https://github.com/Mellanox/k8s-rdma-shared-dev-plugin
`sriovDevicePlugin` DevicePluginSpec	SriovDevicePlugin manages SRIOV through the Kubernetes device plugin framework. The config field is a json representation of the RDMA shared device plugin configuration. See https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin

`nodeSelector` map[string]string	NodeSelector specifies a selector for the nodes this policy applies to
`labels` map[string]string	Optional: Map of string keys and values that can be used to organize and categorize (scope and select) objects. May match selectors of replication controllers and services.
`annotations` map[string]string	Optional: Annotations is an unstructured key value map stored with a resource that may be set by external tools to store and retrieve arbitrary metadata. They are not queryable and should be preserved when modifying objects.
`tolerations` []Kubernetes core/v1.Toleration	Optional: Set tolerations

NicNodePolicyStatus

(*Appears on:* NicNodePolicy)

NicNodePolicyStatus defines the observed state of NicNodePolicy

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

NicPolicyCR

NicPolicyCR is the common interface satisfied by NicClusterPolicy and NicNodePolicy. State sync methods use it to access the desired NIC configuration and set owner references without depending on a concrete CRD type.

OFEDDriverSpec

(*Appears on:* NicClusterPolicySpec, NicNodePolicySpec)

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
`xPlane` ImageSpec	*(Optional)*

State (`string` alias)

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

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: doca3.4.1-26.04-1.1.0.0-1
    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-v26.4.1
    # 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-v26.4.1
  config: |
    {
      "resourceList": [
        {
          "resourcePrefix": "nvidia.com",
          "resourceName": "hostdev",
          "selectors": {
            "vendors": ["15b3"],
            "isRdma": true
          }
        }
      ]
    }
 secondaryNetwork:
   cniPlugins:
     image: plugins
     repository: nvcr.io/nvidia/mellanox
     version: network-operator-v26.4.1
   ipoib:
     image: ipoib-cni
     repository: nvcr.io/nvidia/mellanox
```

```
    version: network-operator-v26.4.1
  multus:
    image: multus-cni
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
nvIpam:
  image: nvidia-k8s-ipam
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.4.1
  enableWebhook: false
ibKubernetes:
  image: ib-kubernetes
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.4.1
  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-v26.4.1
docaTelemetryService:
  image: doca_telemetry
  repository: nvcr.io/nvidia/doca
  version: 1.25.5-doca3.4.0-host
spectrumXOperator:
  image: spectrum-x-operator
  repository: nvcr.io/nvidia/mellanox
  version: network-operator-v26.4.1
nicConfigurationOperator:
  operator:
    image: nic-configuration-operator
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
  configurationDaemon:
    image: nic-configuration-operator-daemon
```

```
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
  nicFirmwareStorage:
    create: true
    pvcName: nic-fw-storage-pvc
    storageClassName: nic-fw-storage-class
    availableStorageSize: 1Gi
  logLevel: info
```

Note

For heterogeneous clusters where different node groups require different DOCA-OFED driver versions or device plugin configurations, see Heterogeneous Clusters with NicNodePolicy.

Heterogeneous Clusters with NicNodePolicy

By default, the NVIDIA Network Operator uses a single NicClusterPolicy (NCP) resource to manage all NIC-related components cluster-wide. In heterogeneous clusters, where different groups of nodes require different DOCA-OFED driver versions or device plugin configurations, this single-policy model is not sufficient. NicNodePolicy (NNP) addresses this by allowing you to create multiple per-node-group policies, each targeting specific nodes via `nodeSelector` labels.

> **⚠ Warning**
>
> NicNodePolicy is recommended for **new deployments only**. There is no automated migration path to transition an existing NicClusterPolicy-only deployment to use NicNodePolicies. Migrating an existing cluster requires manually removing per-node sections from NicClusterPolicy and creating corresponding NicNodePolicy resources, which causes temporary disruption to affected components.

When to Use NicNodePolicy

For new deployments, NicNodePolicy is the **recommended** approach for managing per-node NIC components (DOCA-OFED driver, RDMA shared device plugin, SR-IOV device plugin), even in homogeneous clusters. Using NicNodePolicy from the start provides a consistent deployment model and makes it straightforward to support heterogeneous configurations in the future without re-architecting your policies.

NicNodePolicy is especially valuable when:

- Different groups of nodes need different DOCA-OFED driver versions (e.g., GPU nodes on the latest DOCA release, storage nodes on an LTS release).
- Different node groups need different SR-IOV device plugin configurations.
- Different node groups need different RDMA shared device plugin configurations.
- You need independent DOCA-OFED driver upgrade schedules per node group.

NicClusterPolicy alone is sufficient when you have an existing deployment that already manages per-node components through NCP and does not require heterogeneous configurations. Cluster-wide components such as Multus, NV-IPAM, and NIC Configuration Operator are always managed exclusively by NicClusterPolicy regardless of whether NicNodePolicy is used.

How NicClusterPolicy and NicNodePolicy Work Together

Overview

NicClusterPolicy is a singleton resource (always named `nic-cluster-policy`) that manages up to 12 component types cluster-wide. NicNodePolicy allows creating multiple instances, each targeting a specific set of nodes via `nodeSelector`, but only for a subset of 3 components.

The following table shows which components can be managed by each policy type. See the NicClusterPolicySpec and NicNodePolicySpec API references for full field details.

Component Ownership

Component	NCP	NNP
DOCA-OFED Driver	•	•
RDMA Shared Device Plugin	•	•
SR-IOV Device Plugin	•	•
Multus CNI	•	
CNI Plugins	•	
IPoIB CNI	•	
NV-IPAM	•	
NIC Configuration Operator	•	
DOCA Telemetry Service	•	
Spectrum-X Operator	•	
IB Kubernetes	•	
NIC Feature Discovery	•	

Cluster-wide infrastructure components (Multus, NV-IPAM, NIC Configuration Operator, etc.) always remain in NicClusterPolicy. Per-node components (DOCA-OFED driver and device plugins) can be moved to NicNodePolicy instances to support heterogeneous configurations.

NCP-Only Deployment Flow

In a homogeneous cluster, NicClusterPolicy manages everything. All nodes run the same DOCA-OFED driver version and device plugin configuration:

```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: doca3.4.1-26.04-1.1.0.0-1
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
  rdmaSharedDevicePlugin:
    image: k8s-rdma-shared-dev-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": ["15b3"]
            }
          }
        ]
      }
  secondaryNetwork:
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
  nvIpam:
    image: nvidia-k8s-ipam
```

```
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
```

NCP + NNP Deployment Flow

In a heterogeneous cluster, NicClusterPolicy retains only cluster-wide components. Per-node components move to NicNodePolicy instances, each targeting a specific node group:

```
# Cluster-wide components only
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  secondaryNetwork:
    multus:
      image: multus-cni
      repository: nvcr.io/nvidia/mellanox
      version: network-operator-v26.4.1
  nvIpam:
    image: nvidia-k8s-ipam
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
---
# GPU nodes: latest DOCA driver with RDMA
apiVersion: mellanox.com/v1alpha1
kind: NicNodePolicy
metadata:
  name: gpu-nodes
spec:
  nodeSelector:
    node-role.kubernetes.io/gpu: ""
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: doca3.4.1-26.04-1.1.0.0-1
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 1
  rdmaSharedDevicePlugin:
    image: k8s-rdma-shared-dev-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
```

```
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_a",
            "rdmaHcaMax": 63,
            "selectors": {
              "vendors": ["15b3"]
            }
          }
        ]
      }
---
# Storage nodes: LTS DOCA driver with SR-IOV
apiVersion: mellanox.com/v1alpha1
kind: NicNodePolicy
metadata:
  name: storage-nodes
spec:
  nodeSelector:
    node-role.kubernetes.io/storage: ""
  ofedDriver:
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: doca3.2.3-25.10-3.1.8.0-0
    upgradePolicy:
      autoUpgrade: true
      maxParallelUpgrades: 2
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: nvcr.io/nvidia/mellanox
    version: network-operator-v26.4.1
    config: |
      {
        "resourceList": [
          {
```

```
            "resourcePrefix": "nvidia.com",
            "resourceName": "sriov_rdma",
            "selectors": {
              "vendors": ["15b3"],
              "isRdma": true
            }
          }
        ]
      }
```

Notice that the `ofedDriver` section is absent from NicClusterPolicy and present in both NicNodePolicy instances. Each NNP targets a distinct set of nodes with different DOCA-OFED versions and device plugin types.

DOCA-OFED Driver Upgrades with NicNodePolicy

Each NicNodePolicy with an `ofedDriver` section gets its own independent upgrade state manager. Upgrading the DOCA-OFED driver on one node group does not affect other node groups.

Key behaviors:

- Each NicNodePolicy has a separate `maxParallelUpgrades` setting.
- In maintenance-operator mode, each NicNodePolicy creates its own `NodeMaintenance` custom resources with a policy-specific requestor ID.
- Upgrade progress is tracked independently per policy via the `nvidia.com/ofed-driver-upgrade-state` node label.

> **Note**
>
> For general DOCA-OFED driver upgrade configuration, see the Automatic DOCA-OFED Driver Upgrade section in the Life Cycle Management page.

Deployment Rules and Restrictions

Section Exclusivity

A given section (`ofedDriver`, `rdmaSharedDevicePlugin`, `sriovDevicePlugin`) can exist in **either** NicClusterPolicy or NicNodePolicy instances, but **not both simultaneously**. This rule is enforced by the admission webhook and prevents conflicting configurations.

For example:

```
VALID configuration:
  NicClusterPolicy:  secondaryNetwork, nvIpam
  NicNodePolicy "gpu-nodes":      ofedDriver,
rdmaSharedDevicePlugin
  NicNodePolicy "storage-nodes":  ofedDriver, sriovDevicePlugin
INVALID configuration (rejected by webhook):
  NicClusterPolicy:  ofedDriver            <-- conflict
  NicNodePolicy "gpu-nodes":  ofedDriver   <-- same section in
both
```

> **Note**
>
> Section exclusivity is enforced per section, not per NicNodePolicy instance. If **any** NicNodePolicy defines `ofedDriver`, then NicClusterPolicy must **not** define `ofedDriver`, and vice versa.

Node Selector Overlap Prevention

Two NicNodePolicy instances must not select overlapping sets of nodes. This is validated at two points:

1. **At admission time** – the validating webhook resolves node selectors against actual cluster nodes and checks for intersection. If overlap is found, the create or update

request is rejected.

2. **At runtime** – the NicNodePolicy controller re-checks for overlap on every reconciliation to catch cases where nodes are re-labeled after the initial admission check. If overlap is detected, the affected NicNodePolicy status is set to `Error` with a message describing which nodes overlap, and no DaemonSet changes are applied until the overlap is resolved.

```
# VALID: non-overlapping node selectors
---
# NicNodePolicy "gpu-nodes"
spec:
  nodeSelector:
    node-role.kubernetes.io/gpu: ""
---
# NicNodePolicy "storage-nodes"
spec:
  nodeSelector:
    node-role.kubernetes.io/storage: ""
```

```
# INVALID: if any node has BOTH labels, overlap is detected
---
# NicNodePolicy "pool-a"
spec:
  nodeSelector:
    datacenter: us-east
---
# NicNodePolicy "pool-b"
spec:
  nodeSelector:
    rack: row-1
```

> **Note**
>
> The overlap check resolves selectors against the actual nodes in the cluster. Two NicNodePolicies with different label keys can still overlap if any node happens to match both selectors.

Naming Requirements

NicNodePolicy names must be **at most 30 characters** long. This limit exists because derived resource names (such as DaemonSet names and label values) incorporate the policy name and must stay within the Kubernetes 63-character label value limit.

Kubernetes short names for NicNodePolicy: `nicnode`, `nnp`.

Each NicNodePolicy creates uniquely-named DaemonSets by appending the policy name as a suffix:

DaemonSet Naming

Policy	DaemonSet Name Pattern
NicClusterPolicy	`mofed-<os><version>-<hash>-ds`
NicNodePolicy `gpu-nodes`	`mofed-<os><version>-<hash>-gpu-nodes-ds`

Prerequisites

The admission controller must be enabled for the section exclusivity and node overlap validation rules to be enforced. Set `operator.admissionController.enabled` to `true` in the Helm chart values. See Advanced Configurations for admission controller deployment details.

Verifying the Deployment

List all NicNodePolicy resources:

```
kubectl get nicnodepolicies
```

Check the status of a specific NicNodePolicy:

```
kubectl get nnp gpu-nodes -o yaml
```

A `state: ready` status indicates that the NicNodePolicy has been successfully applied. An `state: error` status with a `reason` field indicates a problem, such as a node selector overlap.

Verify the DaemonSets created by a NicNodePolicy:

```
kubectl get ds -n nvidia-network-operator -l nvidia.com/ofed-
driver=
```

Check for policy errors across all NicNodePolicies:

```
kubectl get nnp -o custom-
columns=NAME:.metadata.name,STATE:.status.state,REASON:.status.reas
```

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
26.4.x	Supported	GA version (full support)

26.1.x	Deprecated	Maintenance version (critical bug and CVE fixes only)
25.10.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-26.4.1.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.

Note

When using NicNodePolicy for heterogeneous clusters, each NicNodePolicy with an `ofedDriver` section manages its own independent upgrade lifecycle with separate `maxParallelUpgrades` settings and upgrade state tracking. See Heterogeneous Clusters with NicNodePolicy for details.

`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: doca3.4.1-26.04-1.1.0.0-1
    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: doca3.4.1-26.04-1.1.0.0-1
    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
  - Network Operator Deployment in a Proxy Environment
    - Prerequisites
    - HTTP Proxy Configuration for Openshift
    - HTTP Proxy Configuration
  - Network Operator Deployment in an Air-gapped Environment
    - Local Image Registry
    - Pulling and Pushing Container Images to a Local Registry
    - Configuring Local Registry TLS Cert
    - Local Package Repository
- DOCA-OFED Driver Container
  - NVIDIA DOCA-OFED Driver Container Environment Variables
    - CREATE_IFNAMES_UDEV Environment Variable
    - ENABLE_NFSRDMA Environment Variable
    - Example of NicClusterPolicy
  - Precompiled Container Build Instructions for NVIDIA DOCA-OFED Driver Container
    - Prerequisites
      - Download Docker files and scripts:
    - Dockerfile Overview
    - Common mandatory build parameters
    - RHCOS
      - Prerequisites
      - Specific build parameters
      - RHCOS example
    - Ubuntu
      - Ubuntu example
    - SLES
      - Prerequisites
      - SLES example
- Other Advanced Configurations
  - Network Operator Deployment with Admission Controller
  - Network Operator Deployment with Pod Security Admission
  - Container Resources
- Container Images Digests
  - DOCA-OFED Driver Container Images
    - Ubuntu
    - RHCOS
    - RHEL

        - SLES
    - STIG FIPS Compliant DOCA-OFED Driver Container Images
        - Ubuntu
        - RHEL

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 doca3.4.1-26.04-1.1.0.0-1, the tag for Ubuntu 24.04 with X86 architecture is “doca3.4.1-26.04-1.1.0.0-1-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 7fa75defaf4487f80a0252a781b201093298cdaf
```

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, NicNodePolicy, and HostDeviceNetwork. By default, the deployment of the admission controller is disabled. To enable it, you must set `operator.admissionController.enabled` to `true`.

> **Note**
>
> The admission controller is **required** when using NicNodePolicy for heterogeneous cluster configurations. It enforces section exclusivity between NicClusterPolicy and NicNodePolicy, and prevents node selector overlap between NicNodePolicy instances. See Heterogeneous Clusters with NicNodePolicy for details.

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	v26.4.1	sha256:49d8bd43fa4559468ec93a4
nvcr.io/nvidia/mellanox	network-operator-init-container	network-operator-v26.4.1	sha256:922217cee504f080b2a76c5
nvcr.io/nvidia/mellanox	k8s-rdma-shared-dev-plugin	network-operator-v26.4.1	sha256:464970d3197fe982b779c48
nvcr.io/nvidia/mellanox	ib-kubernetes	network-operator-v26.4.1	sha256:b103b0a36693d6ab1a25ca6
nvcr.io/nvidia/mellanox	ipoib-cni	network-operator-v26.4.1	sha256:e5eb2d3c746a06a44bad7e8

nvcr.io/nvidia/mellanox	nvidia-k8s-ipam	network-operator-v26.4.1	sha256:88db2fa65846d68d9c4ec9
nvcr.io/nvidia/mellanox	nic-feature-discovery	network-operator-v26.4.1	sha256:2599a0d512af2f92c1fa6ac
nvcr.io/nvidia/doca	doca_telemetry	1.25.5-doca3.4.0-host	sha256:e728430bdde27bc0f2e57ce
nvcr.io/nvidia/mellanox	sriov-network-operator	network-operator-v26.4.1	sha256:d631df0a88e9088b6196d4
nvcr.io/nvidia/mellanox	sriov-network-operator-webhook	network-operator-v26.4.1	sha256:b4410df7cc3961fd7e059d8
nvcr.io/nvidia/mellanox	sriov-network-operator-config-daemon	network-operator-v26.4.1	sha256:96e50e3f5d754ac33bb0a80
nvcr.io/nvidia/mellanox	sriov-network-device-plugin	network-operator-v26.4.1	sha256:c24d0993feb97c0a1952bf3
nvcr.io/nvidia/mellanox	sriov-cni	network-operator-v26.4.1	sha256:9d627407b27ff817152e1c0
nvcr.io/nvidia/mellanox	ib-sriov-cni	network-operator-v26.4.1	sha256:ed91dfb15d5da90d9fad623
nvcr.io/nvidia/mellanox	dra-driver-sriov	network-operator-v26.4.1	sha256:ff43dae772403fa5f8988fa0
nvcr.io/nvidia/mellanox	plugins	network-operator-v26.4.1	sha256:7d96a2f3a4009f29c6cd799
nvcr.io/nvidia/mellanox	multus-cni	network-operator-v26.4.1	sha256:27e5fc5349625fd6d354230

nvcr.io/nvidia/mellanox	ovs-cni-plugin	network-operator-v26.4.1	sha256:d86c29504c0a4ff4bd2e412
nvcr.io/nvidia/mellanox	rdma-cni	network-operator-v26.4.1	sha256:03f9845060c952c7820836
nvcr.io/nvidia/mellanox	nic-configuration-operator	network-operator-v26.4.1	sha256:b5e9733a01ba6031ecbf71C
nvcr.io/nvidia/mellanox	nic-configuration-operator-daemon	network-operator-v26.4.1	sha256:ff80af3402473fba6251f36f
nvcr.io/nvidia/mellanox	maintenance-operator	network-operator-v26.4.1	sha256:cfd6cd87f077c1879d4c4e9
nvcr.io/nvidia/mellanox	spectrum-x-operator	network-operator-v26.4.1	sha256:0f5bb7c29d17b6c9c4957d2

DOCA-OFED Driver Container Images

Repository	Image Name	Version
nvcr.io/nvidia/mellanox	doca-driver	doca3.4.1-26.04-1.1.0.0-1

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

Ubuntu

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-ubuntu22.04-amd64	sha256:015ae0c39d5fe8ced57f66e369f0d1c0ba77e6f83817489b3d15a2

doca3.4.1-26.04-1.1.0.0-1-ubuntu22.04-arm64	sha256:f0a16e01d47cd928692092b500ce8b2e5401a73884bca8b4a8b8
doca3.4.1-26.04-1.1.0.0-1-ubuntu24.04-amd64	sha256:edce1f95a5778cd6a36c13ebb4b87fc43b61ede5d4dc0489e56cf
doca3.4.1-26.04-1.1.0.0-1-ubuntu24.04-arm64	sha256:c802335d23793c12347fae05adcadc4de5bff556d7c1d9f3eb1fa2

RHCOS

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-rhcos4.17-amd64 doca3.4.1-26.04-1.1.0.0-1-rhcos4.18-amd64	sha256:8c21373355eebfb0127fb6c19efb5a59ce878e636afddeaf186f238b

doca3.4.1-26.04-1.1.0.0-1-rhcos4.17-arm64 doca3.4.1-26.04-1.1.0.0-1-rhcos4.18-arm64	sha256:087d6ca78ff0ff1ca0446e80c093185cdf06976df757db74ff9f3453c

RHEL

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-rhel10.0-amd64	sha256:b15772d84e982ba4e83e772405d9ea71c80dc92bee7064fdc5725e8
doca3.4.1-26.04-1.1.0.0-1-rhel10.0-arm64	sha256:4bbb1726299e9df97fa43c2a067670f30ca9e13f301b78d426b1910b
doca3.4.1-26.04-1.1.0.0-1-rhel10.2-amd64	sha256:6bd61a20b45e22ceacac6da16fb04512c494ecf37878505d8cbd67a6
doca3.4.1-26.04-1.1.0.0-1-rhel10.2-arm64	sha256:092424661028fd9c1d8fcdc5fab7913695211a66d7f096c574f9797

doca3.4.1-26.04-1.1.0.0-1-rhel8.10-amd64	sha256:df9d3e99ca8ef8cb9188ae9c858d5eb32a480733f6b70d0fba1cb813
doca3.4.1-26.04-1.1.0.0-1-rhel8.10-arm64	sha256:32e2855ade034b7be6b04de7033c1c6764e5e6afb665a7e6ffa786b9
doca3.4.1-26.04-1.1.0.0-1-rhel9.4-amd64 doca3.4.1-26.04-1.1.0.0-1-rhel9.6-amd64	sha256:73e37500c6224b9807b3f7814b72d81ec2b5708cdb9e6264007910c
doca3.4.1-26.04-1.1.0.0-1-rhel9.4-arm64 doca3.4.1-26.04-1.1.0.0-1-rhel9.6-arm64	sha256:559ff08d3f0fbf72bb300f87e7782aacd1e90ab8b1d80c78235f6478c
doca3.4.1-26.04-1.1.0.0-1-rhel9.8-amd64	sha256:08946c3bc681506f5b08557bed9b19683d225569aa835dd6f2dace7

doca3.4.1-26.04-1.1.0.0-1-rhel9.8-arm64	sha256:0c5d7f07d8aa8331745739be21563e3faa67bd914bc055c63cc5aeb8

SLES

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-sles15.7-amd64	sha256:721656effacc772748693fd7087e5d313a1707d35c9b9ca14782ff748
doca3.4.1-26.04-1.1.0.0-1-sles15.7-arm64	sha256:048468221ba0664e1e8fba9daf0d9be0dca00952a80cc3dd3dbc022b

STIG FIPS Compliant DOCA-OFED Driver Container Images

Repository	Image Name	Version
nvcr.io/nvidia/mellanox	doca-driver-stig-fips	doca3.4.1-26.04-1.1.0.0-1

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

Ubuntu

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-ubuntu24.04-amd64	sha256:23d6bd67ae83f1d720a562dff3a94ca8e1c476d44ad3e3a0e88cba

RHEL

Tags	Digest
doca3.4.1-26.04-1.1.0.0-1-rhel9.6-amd64	sha256:648b0c3664c8111a7572d1996dbab2be2ea55f6609a75f0f9d78e64(

Troubleshooting

- SOS-Report Collection Script
  - Overview
  - Installation
    - As a kubectl Plugin (Recommended)
    - As a Standalone Script
  - Requirements
  - Usage
    - Basic Usage
    - Command-Line Options
  - What's Collected
    - Custom Resources
    - Operator Resources
    - Components
    - Node Information
    - Diagnostic Commands
  - HTML Report
  - Output Structure
  - Exit Codes
  - Security Considerations
  - Example Workflows
    - Troubleshooting Pod Failures
    - Quick Health Check
    - Preparing for a Support Case

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:

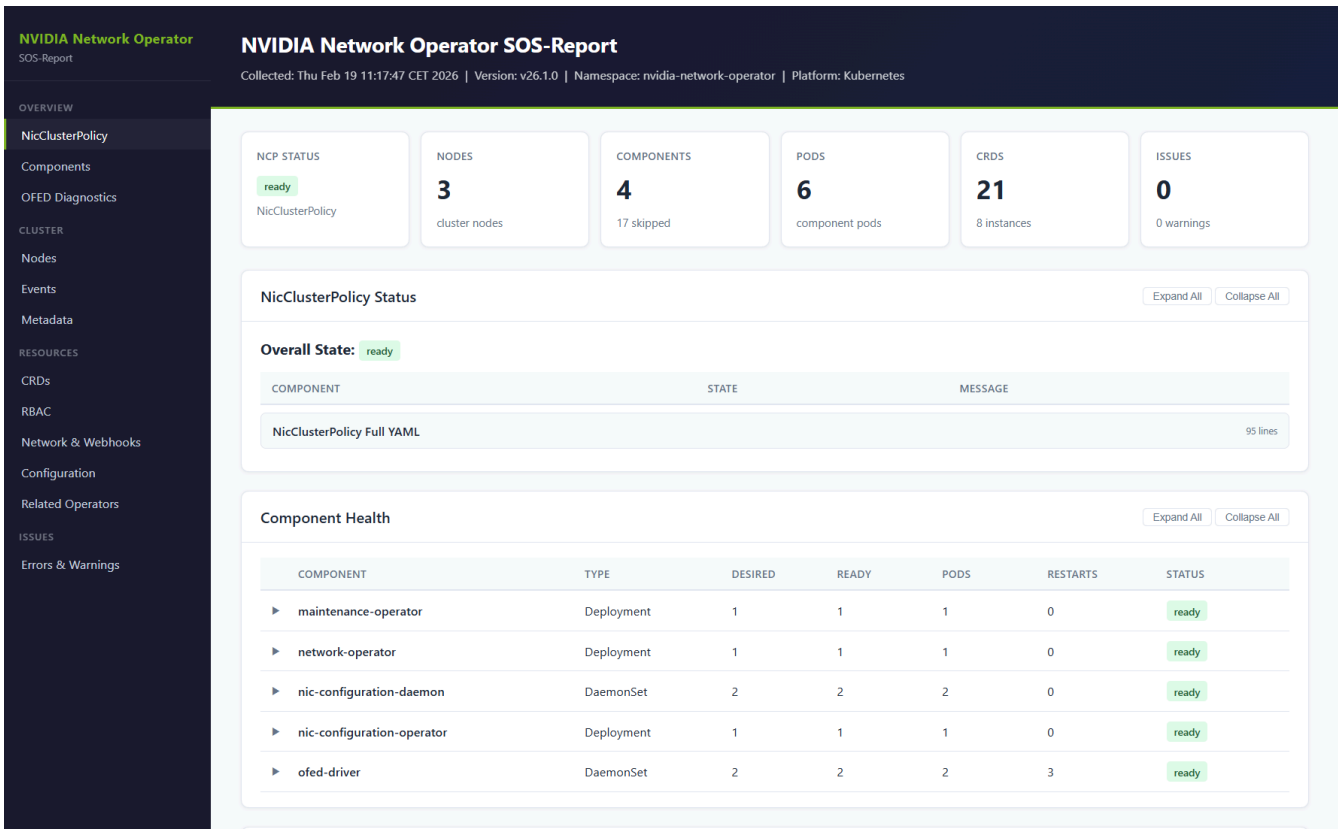
- `lsmod | grep mlx` — loaded Mellanox kernel modules
- `ibstat` — InfiniBand device status
- `ibv_devinfo` — RDMA device information
- `mst status` — Mellanox Software Tools status
- `uname -r` — kernel version
- `dmesg` (last 200 lines) — recent kernel messages
- `ip link` / `ip addr` — network interface 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	v26.4.1	- License - 3rd Part Notice
NVIDIA Network Operator Init Container	network-operator-v26.4.1	- License - 3rd Part Notice
RDMA Shared Device Plugin	network-operator-v26.4.1	- License - 3rd Part Notice
IB Kubernetes Plugin	network-operator-v26.4.1	- License - 3rd Part Notice
IP Over Infiniband (IPoIB) CNI plugin	network-operator-v26.4.1	- License - 3rd Part Notice
NVIDIA IPAM Plugin	network-operator-v26.4.1	- License - 3rd Part Notice
NVIDIA NIC Feature Discovery	network-operator-v26.4.1	- License - 3rd Part Notice
Node Feature Discovery	network-operator-v26.4.1	- License - 3rd Part Notice
SRIOV Network Operator	network-operator-v26.4.1	- License - 3rd Part Notice

SR-IOV Network Device Plugin	network-operator-v26.4.1	- License - 3rd Part Notice
SR-IOV CNI plugin	network-operator-v26.4.1	- License - 3rd Part Notice
InfiniBand SR-IOV CNI plugin	network-operator-v26.4.1	- License - 3rd Part Notice
Multus CNI	network-operator-v26.4.1	- License - 3rd Part Notice
RDMA CNI plugin	network-operator-v26.4.1	- License - 3rd Part Notice
Open vSwitch CNI plugin	network-operator-v26.4.1	- License - 3rd Part Notice
NVIDIA NIC Configuration Operator	network-operator-v26.4.1	- License - 3rd Part Notice
NVIDIA Maintenance Operator	network-operator-v26.4.1	- License - 3rd Part Notice
NVIDIA SpectrumX Operator	network-operator-v26.4.1	- License - 3rd Part Notice
CNI Plugins	network-operator-v26.4.1	- License - 3rd Part Notice
DOCA Driver Container	doca3.4.1-26.04-1.1.0.0-1	- License - 3rd Part Notice
DRA SR-IOV Driver	network-operator-v26.4.1	- License - 3rd Part Notice

Kubernetes Launch Kit	v26.4.1	- 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.

© Copyright 2025-2026, NVIDIA.. PDF Generated on 10/15/2026