Network Operator

Network Operator Deployment Guide

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 a fast network
- Kubernetes secondary network components for network intensive workloads

Network Operator Deployment on Vanilla Kubernetes Cluster

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. Please refer to the NicClusterPolicy CRD Section for more information on manual Custom Resource creation.

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

Add NVIDIA NGC repository:

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 default values:

Install NVIDIA Network Operator Helm chart

helm install network-operator nvidia/network-operator
  -n nvidia-network-operator
  --create-namespace
  --version v23.4.0
  --wait

View deployed resources

kubectl -n nvidia-network-operator get pods

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

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.
Get chart values for customization

`helm show values nvidia/network-operator --version v23.4.0 > values.yaml`

Install NVIDIA Network Operator using customize values

`helm install network-operator nvidia/network-operator \
  --create-namespace \
  --version v23.4.0 \
  -f ./values.yaml \
  --wait`

Helm Chart Customization Options

In order to tailor the deployment of the Network Operator to your cluster needs, use the following parameters:

### General Parameters

In order to tailor the deployment of the Network Operator to your cluster needs, use the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfd.enabled</td>
<td>Bool</td>
<td>True</td>
<td>Deploy Node Feature Discovery</td>
</tr>
<tr>
<td>sriovNetworkOperator.enabled</td>
<td>Bool</td>
<td>False</td>
<td>Deploy SR-IOV Network Operator</td>
</tr>
<tr>
<td>sriovNetworkOperator.configDaemonNodeSelectorExtra</td>
<td>List</td>
<td>node-role.kubernetes.io/worker: &quot;&quot;</td>
<td>Additional values for SR-IOV Config Daemon nodes selector</td>
</tr>
<tr>
<td>upgradeCRDs</td>
<td>Bool</td>
<td>True</td>
<td>Enable CRDs upgrade with helm pre-install and pre-upgrade hooks</td>
</tr>
<tr>
<td>psp.enabled</td>
<td>Bool</td>
<td>False</td>
<td>Deploy Pod Security Policy</td>
</tr>
<tr>
<td>operator.repository</td>
<td>String</td>
<td>nvcr.io/nvidia</td>
<td>Network Operator image repository</td>
</tr>
<tr>
<td>operator.image</td>
<td>String</td>
<td>network-operator</td>
<td>Network Operator image name</td>
</tr>
<tr>
<td>operator.tag</td>
<td>String</td>
<td>None</td>
<td>Network Operator image tag. If set to None, the chart's appVersion will be used</td>
</tr>
<tr>
<td>operator.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the Network Operator images</td>
</tr>
<tr>
<td>deployCR</td>
<td>Bool</td>
<td>false</td>
<td>Deploy NicClusterPolicy custom resource according to the provided parameters</td>
</tr>
<tr>
<td>nodeAffinity</td>
<td>Object</td>
<td>requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms: - matchExpressions: key: node-role.kubernetes.io/master operator: DoesNotExist key: node-role.kubernetes.io/control-plane operator: DoesNotExist</td>
<td>Configure node affinity settings for Network Operator components</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Label</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature.node.kubernetes.io/pci-15b3.present</td>
<td>Nodes containing NVIDIA Networking hardware</td>
</tr>
<tr>
<td>feature.node.kubernetes.io/pci-10de.present</td>
<td>Nodes containing NVIDIA GPU hardware</td>
</tr>
</tbody>
</table>

**MLNX_OFED Driver**
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofedDriver.deploy</td>
<td>Bool</td>
<td>false</td>
<td>Deploy the NVIDIA MLNX_OFED driver container</td>
</tr>
<tr>
<td>ofedDriver.repository</td>
<td>String</td>
<td>nvcr.io/nvidia/mellanox</td>
<td>NVIDIA OFED driver image repository</td>
</tr>
<tr>
<td>ofedDriver.image</td>
<td>String</td>
<td>mofed</td>
<td>NVIDIA OFED driver image name</td>
</tr>
<tr>
<td>ofedDriver.version</td>
<td>String</td>
<td>23.04-0.5.3.1</td>
<td>NVIDIA OFED driver version</td>
</tr>
<tr>
<td>ofedDriver.env</td>
<td>List</td>
<td>[]</td>
<td>An optional list of environment variables passed to the Mellanox OFED driver image</td>
</tr>
<tr>
<td>ofedDriver.terminationGracePeriodSeconds</td>
<td>Int</td>
<td>300</td>
<td>NVIDIA OFED termination grace period in seconds</td>
</tr>
<tr>
<td>ofedDriver.repoConfig.name</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>Private mirror repository configuration configMap name</td>
</tr>
<tr>
<td>ofedDriver.certConfig.name</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>Custom TLS key/certificate configuration configMap name</td>
</tr>
<tr>
<td>ofedDriver.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the NVIDIA OFED driver images</td>
</tr>
<tr>
<td>ofedDriver.startupProbe.initialDelaySeconds</td>
<td>Int</td>
<td>10</td>
<td>NVIDIA OFED startup probe initial delay</td>
</tr>
<tr>
<td>ofedDriver.startupProbe.periodSeconds</td>
<td>Int</td>
<td>20</td>
<td>NVIDIA OFED startup probe interval</td>
</tr>
<tr>
<td>ofedDriver.livenessProbe.initialDelaySeconds</td>
<td>Int</td>
<td>30</td>
<td>NVIDIA OFED liveness probe initial delay</td>
</tr>
<tr>
<td>ofedDriver.livenessProbe.periodSeconds</td>
<td>Int</td>
<td>30</td>
<td>NVIDIA OFED liveness probe interval</td>
</tr>
<tr>
<td>ofedDriver.readinessProbe.initialDelaySeconds</td>
<td>Int</td>
<td>10</td>
<td>NVIDIA OFED readiness probe initial delay</td>
</tr>
<tr>
<td>ofedDriver.readinessProbe.periodSeconds</td>
<td>Int</td>
<td>30</td>
<td>NVIDIA OFED readiness probe interval</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.autoUpgrade</td>
<td>Bool</td>
<td>false</td>
<td>Global switch for the automatic upgrade feature. If set to false, all other options are ignored.</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.maxParallelUpgrades</td>
<td>Int</td>
<td>1</td>
<td>The amount of nodes that can be upgraded in parallel. 0 means no limit. All nodes will be upgraded in parallel.</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.drain.enable</td>
<td>Bool</td>
<td>true</td>
<td>Options for node drain ('kubectl drain') before the driver reload, if auto upgrade is enabled.</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.drain.force</td>
<td>Bool</td>
<td>false</td>
<td>Use force drain of pods</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.drain.podSelector</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>Pod selector to specify which pods will be drained from the node. An empty selector means all pods.</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.drain.timeoutSeconds</td>
<td>Int</td>
<td>300</td>
<td>Number of seconds to wait for pod eviction</td>
</tr>
<tr>
<td>ofedDriver.upgradePolicy.drain.deleteEmptyDir</td>
<td>Bool</td>
<td>false</td>
<td>Delete pods local storage</td>
</tr>
</tbody>
</table>

**MLNX_OFED Driver Environment Variables**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE_DRIVER_ON_POD_</td>
<td>true</td>
<td>• &quot;true&quot; = Restores Mellanox OFED Driver from the host upon pod termination.</td>
</tr>
<tr>
<td>TERMINATION</td>
<td></td>
<td>• &quot;false&quot; = Upon termination, keeps drivers loaded by the container in memory. On the next reboot, the kernel drivers will be loaded.</td>
</tr>
<tr>
<td>UNLOAD_STORAGE_MODULES</td>
<td>false</td>
<td>• &quot;true&quot; = Unload current storage modules (E.g. ib_isert nvme_rdma nvmet_rdma rpcrdma xprtdma ib_srpt) on pod startup to load updated versions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;false&quot; = Not handling storage modules unload. This might be the case when the operator wishes to manage storage drivers unload manually.</td>
</tr>
<tr>
<td>CREATE_IFNAMES_UDEV</td>
<td>false</td>
<td>• &quot;true&quot; = Create udev rules for NVIDIA network adapters to have persistent NIC names.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;false&quot; = Not handling udev rules. Operator assumed to configure manually persistent NIC names.</td>
</tr>
</tbody>
</table>
To set these variables, change them into Helm values. For example:

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

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

### NVIDIA Peer Memory Driver

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvPeerDriver.deploy</td>
<td>Bool</td>
<td>false</td>
<td>Deploy NVIDIA Peer memory driver container</td>
</tr>
<tr>
<td>nvPeerDriver.repository</td>
<td>String</td>
<td>mellanox</td>
<td>NVIDIA Peer memory driver image repository</td>
</tr>
<tr>
<td>nvPeerDriver.image</td>
<td>String</td>
<td>nv-peer-mem-driver</td>
<td>NVIDIA Peer memory driver image name</td>
</tr>
<tr>
<td>nvPeerDriver.version</td>
<td>String</td>
<td>1.1-0</td>
<td>NVIDIA Peer memory driver version</td>
</tr>
<tr>
<td>nvPeerDriver.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the NVIDIA Peer memory driver images</td>
</tr>
<tr>
<td>nvPeerDriver.gpuDriverSourcePath</td>
<td>String</td>
<td>/run/nvidia/driver</td>
<td>GPU driver sources root filesystem path (usually used in tandem with <code>gpu-operator</code>)</td>
</tr>
</tbody>
</table>

### RDMA Shared Device Plugin

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdmaSharedDevicePlugin.deploy</td>
<td>Bool</td>
<td>true</td>
<td>Deploy RDMA shared device plugin</td>
</tr>
<tr>
<td>rdmaSharedDevicePlugin.repository</td>
<td>String</td>
<td>nvcr.io/nvidia/cloud-native</td>
<td>RDMA shared device plugin image repository</td>
</tr>
<tr>
<td>rdmaSharedDevicePlugin.image</td>
<td>String</td>
<td>k8s-rdma-shared-dev-plugin</td>
<td>RDMA shared device plugin image name</td>
</tr>
<tr>
<td>rdmaSharedDevicePlugin.version</td>
<td>String</td>
<td>v1.3.2</td>
<td>RDMA shared device plugin version</td>
</tr>
<tr>
<td>rdmaSharedDevicePlugin.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the RDMA Shared device plugin images</td>
</tr>
<tr>
<td>rdmaSharedDevicePlugin.resources</td>
<td>List</td>
<td>See below</td>
<td>RDMA shared device plugin resources</td>
</tr>
</tbody>
</table>

### RDMA Device Plugin Resource Configurations

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

```yaml
resources:
- name: rdma_shared_device_a
  vendors: [15b3]
  deviceIDs: [1017]
  ifNames: [enp5s0f0]
- name: rdma_shared_device_b
  vendors: [15b3]
  deviceIDs: [1017]
  ifNames: [enp4s0f0, enp4s0f1]
```

### SR-IOV Network Device Plugin

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SR-IOV Network Device Plugin Resource Configuration

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

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

### IB Kubernetes

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibKubernetes.deploy</td>
<td>bool</td>
<td>false</td>
<td>Deploy IB Kubernetes</td>
</tr>
<tr>
<td>ibKubernetes.repository</td>
<td>string</td>
<td>ghcr.io/mellanox</td>
<td>IB Kubernetes image repository</td>
</tr>
<tr>
<td>ibKubernetes.image</td>
<td>string</td>
<td>ib-kubernetes</td>
<td>IB Kubernetes image name</td>
</tr>
<tr>
<td>ibKubernetes.version</td>
<td>string</td>
<td>v1.0.2</td>
<td>IB Kubernetes version</td>
</tr>
<tr>
<td>ibKubernetes.imagePullSecrets</td>
<td>list</td>
<td>[]</td>
<td>An optional list of references to secrets used for pulling any of the IB Kubernetes images</td>
</tr>
<tr>
<td>ibKubernetes.periodicUpdateSeconds</td>
<td>int</td>
<td>5</td>
<td>Interval of periodic update in seconds</td>
</tr>
<tr>
<td>ibKubernetes.pKeyGUIDPoolRangeStart</td>
<td>string</td>
<td>02:00:00:00:00:00:00:00:FF:00</td>
<td>Minimal available GUID value to be allocated for the pod</td>
</tr>
<tr>
<td>ibKubernetes.pKeyGUIDPoolRangeEnd</td>
<td>string</td>
<td>02:FF:FF:FF:FF:FF:FF:FF:FF</td>
<td>Maximal available GUID value to be allocated for the pod</td>
</tr>
<tr>
<td>ibKubernetes.ufmSecret</td>
<td>string</td>
<td>See below</td>
<td>Name of the Secret with the NVIDIA® UFM® access credentials, deployed in advance</td>
</tr>
</tbody>
</table>

### UFM Secret

IB Kubernetes must access NVIDIA® UFM® in order to manage pods' GUIDs. To provide its credentials, the secret of the following format should be deployed in advance:
```yaml
apiVersion: v1
class: Secret
metadata:
  name: ufm-secret
  namespace: nvidia-network-operator
stringData:
  UFM_USERNAME: "admin"
  UFM_PASSWORD: "123456"
  UFM_ADDRESS: "ufm-hostname"
  UFM_HTTP_SCHEMA: ""  
  UFM_PORT: ""
data:
  UFM_CERTIFICATE: ""

---

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

### Secondary Network

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondaryNetwork.deploy</td>
<td>Bool</td>
<td>true</td>
<td>Deploy Secondary Network</td>
</tr>
</tbody>
</table>

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

- **Multus-CNI**: Delegate CNI plugin to support secondary networks in Kubernetes
- **CNI plugins**: Currently only `containernetworking-plugins` is supported
- **IPAM CNI**: Currently only `Whereabout IPAM CNI` is supported
- **IPoIB CNI**: Allow the user to create IPoIB child link and move it to the pod

### CNI Plugin

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondaryNetwork.cniPlugins.deploy</td>
<td>Bool</td>
<td>true</td>
<td>Deploy CNI Plugins Secondary Network</td>
</tr>
<tr>
<td>secondaryNetwork.cniPlugins.image</td>
<td>String</td>
<td>plugins</td>
<td>CNI Plugins image name</td>
</tr>
<tr>
<td>secondaryNetwork.cniPlugins.repository</td>
<td>String</td>
<td>ghcr.io/k8snetworkplumbingwg</td>
<td>CNI Plugins image repository</td>
</tr>
<tr>
<td>secondaryNetwork.cniPlugins.version</td>
<td>String</td>
<td>v1.2.0-amd64</td>
<td>CNI Plugins image version</td>
</tr>
<tr>
<td>secondaryNetwork.cniPlugins.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the CNI Plugins images</td>
</tr>
</tbody>
</table>

### Multus CNI

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondaryNetwork.multus.deploy</td>
<td>Bool</td>
<td>true</td>
<td>Deploy Multus Secondary Network</td>
</tr>
<tr>
<td>secondaryNetwork.multus.image</td>
<td>String</td>
<td>multus-cni</td>
<td>Multus image name</td>
</tr>
<tr>
<td>secondaryNetwork.multus.repository</td>
<td>String</td>
<td>ghcr.io/k8snetworkplumbingwg</td>
<td>Multus image repository</td>
</tr>
<tr>
<td>secondaryNetwork.multus.version</td>
<td>String</td>
<td>v3.9.3</td>
<td>Multus image version</td>
</tr>
<tr>
<td>secondaryNetwork.multus.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the Multus images</td>
</tr>
</tbody>
</table>

### IPoIB CNI

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondaryNetwork.multus.config</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>Multus CNI config. If empty, the config will be automatically generated from the CNI configuration file of the master plugin (the first file in lexicographical order in the cni-config-dir).</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>---------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>secondaryNetwork.ipoib.deploy</td>
<td>Bool</td>
<td>false</td>
<td>Deploy IPoIB CNI</td>
</tr>
<tr>
<td>secondaryNetwork.ipoib.image</td>
<td>String</td>
<td>ipoib-cni</td>
<td>IPoIB CNI image name</td>
</tr>
<tr>
<td>secondaryNetwork.ipoib.repository</td>
<td>String</td>
<td></td>
<td>IPoIB CNI image repository</td>
</tr>
<tr>
<td>secondaryNetwork.ipoib.version</td>
<td>String</td>
<td>v1.1.0</td>
<td>IPoIB CNI image version</td>
</tr>
<tr>
<td>secondaryNetwork.ipoib.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the IPoIB CNI images</td>
</tr>
</tbody>
</table>

**IPAM CNI Plugin**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondaryNetwork.ipamPlugin.deploy</td>
<td>Bool</td>
<td>true</td>
<td>Deploy IPAM CNI Plugin Secondary Network</td>
</tr>
<tr>
<td>secondaryNetwork.ipamPlugin.image</td>
<td>String</td>
<td>whereabouts</td>
<td>IPAM CNI Plugin image name</td>
</tr>
<tr>
<td>secondaryNetwork.ipamPlugin.repository</td>
<td>String</td>
<td>ghcr.io/k8snetworkplumbingw</td>
<td>IPAM CNI Plugin image repository</td>
</tr>
<tr>
<td>secondaryNetwork.ipamPlugin.version</td>
<td>String</td>
<td>v0.6.1-amd64</td>
<td>IPAM CNI Plugin image version</td>
</tr>
<tr>
<td>secondaryNetwork.ipamPlugin.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the IPAM CNI Plugin images</td>
</tr>
</tbody>
</table>

**NVIDIA IPAM Plugin**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvIpam.deploy</td>
<td>Bool</td>
<td>false</td>
<td>Deploy NVIDIA IPAM Plugin</td>
</tr>
<tr>
<td>nvIpam.image</td>
<td>String</td>
<td>nvidia-k8s-ipam</td>
<td>NVIDIA IPAM Plugin image name</td>
</tr>
<tr>
<td>nvIpam.repository</td>
<td>String</td>
<td>ghcr.io/mellanox</td>
<td>NVIDIA IPAM Plugin image repository</td>
</tr>
<tr>
<td>nvIpam.version</td>
<td>String</td>
<td>v0.0.2</td>
<td>NVIDIA IPAM Plugin image version</td>
</tr>
<tr>
<td>nvIpam.imagePullSecrets</td>
<td>List</td>
<td>[]</td>
<td>An optional list of references to secrets to use for pulling any of the Plugin images</td>
</tr>
<tr>
<td>nvIpam.config</td>
<td>String</td>
<td>''</td>
<td>Network pool configuration as described in <a href="https://github.com/Mellanox/nvidia-k8s-ipam">https://github.com/Mellanox/nvidia-k8s-ipam</a></td>
</tr>
</tbody>
</table>

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

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

**Deployment with Pod Security Policy**

⚠️ This section applies to Kubernetes v1.24 or earlier versions only.

A Pod Security Policy is a cluster-level resource that controls security sensitive aspects of the pod specification. The PodSecurityPolicy objects define a set of conditions that a pod must run with in order to be accepted into the system, as well as defaults for the related fields.

By default, the NVIDIA Network Operator does not deploy pod Security Policy. To do that, override the PSP chart parameter:

```
$ helm install -n nvidia-network-operator --create-namespace --wait network-operator nvidia/network-operator --set psp.enabled=true
```

To enforce Pod Security Policies, PodSecurityPolicy admission controller must be enabled. For instructions, refer to this article in Kubernetes Documentation.

The NVIDIA Network Operator deploys a privileged Pod Security Policy, which provides the operator's pods the following permissions:
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.

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:

```bash
$ 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:

```bash
$ kubectl get events -n nvidia-network-operator --field-selector reason=FailedCreate
```

Network Operator Deployment in 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 GPU 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 GPU 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.
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 GPU 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).

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

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

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

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

Pulling and Pushing Container Images to a Local Registry
To pull the correct images from the NVIDIA registry, you can leverage the fields repository, image, and version specified in the values.yaml file.

Local Package Repository
The OFED driver container deployed as part of the Network Operator requires certain packages to be available as part of 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:
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-get` 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 GPU 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]
name=cuda
baseurl=http://<local pkg repository>/cuda
priority=0
gpgcheck=0
enabled=1
```

redhat.repo:
Create the ConfigMap for Ubuntu:

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

Create the ConfigMap for RHCOS:

```bash
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.
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
```

```yaml
ofedDriver:
  deploy: true
  certConfig:
    name: cert-config
```

### Network Operator Deployment on an OpenShift Container Platform

#### Cluster-wide Entitlement

**Introduction**

The NVIDIA Network Operator deploys MOFED pods used to deploy NVIDIA Network Adapter drivers in the OpenShift Container Platform. These Pods require packages that are not available by default in the Universal Base Image (UBI) that the OpenShift Container Platform uses. To make packages available to the MOFED driver container, enable the cluster-wide entitled container builds in OpenShift.

To enable a cluster-wide entitlement, perform the following three steps:

1. Download the Red Hat OpenShift Container Platform subscription certificates from the [Red Hat Customer Portal](https://access.redhat.com) (access requires login credentials).
2. Create a MachineConfig that enables the subscription manager and provides a valid subscription certificate. Wait for the MachineConfig Operator to reboot the node and finish applying the MachineConfig.
3. Validate that the cluster-wide entitlement is working properly.

These instructions assume you have downloaded an entitlement encoded in base64 from the [Red Hat Customer Portal](https://access.redhat.com), or extracted it from an existing node.

Creating entitled containers requires that assigning machine configuration that has a valid Red Hat entitlement certificate to your worker nodes. This step is necessary, since the Red Hat Enterprise Linux (RHEL) CoreOS nodes are not automatically entitled yet.

#### Obtaining an Entitlement Certificate

Follow the guidance below to obtain the entitlement certificate.

1. Navigate to the [Red Hat Customer Portal systems management page](https://access.redhat.com), and click **New**.

   ![New System Management Page](image)

   **Systems**

   Below is a list of systems for this account.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Last Check in</th>
<th>Errata</th>
</tr>
</thead>
</table>

   There are no systems on this account.

   ![Systems Table](image)

   **Show** 100 entries

   Showing 0 to 0 of 0 entries  First  Previous  Next  Last
2. Select Hypervisor, and populate the Name field with the OpenShift-Entitlement text.

3. Click CREATE.

4. Select the Subscriptions tab, and click Attach Subscriptions.

5. Search for the Red Hat Developer Subscription (the content here may vary according to the account), select the desired option, and click Attach Subscriptions.

6. Click Download Certificates.
7. Download and extract the file.
8. Extract the `<key>.pem` key, and test it with the following command:

```bash
curl -E <key>.pem -Sfs -k https://cdn.redhat.com/content/dist/rhel8/8/x86_64/baseos/os/repodata/repomd.xml | head -3
```

Adding a Cluster-wide Entitlement

Perform the following steps to add a cluster-wide entitlement:

1. Create an appropriately named local directory. Change to this directory.
2. Download the machine config YAML template for cluster-wide entitlements on the OpenShift Container Platform. Save the downloaded file to the directory created in Step 1:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
name: 50-rhsm-conf
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,
          IyBSZWQgSGF0IFN1YnNjcmlwdGlvbiBNYW5hZ2VyIENvbmZpZ3VyYXRpb24gRmlsZToKCiMgVW5pZmllZCBFbnRpdmVudCBQbGF0Zm9ybSBDb25maWd1cmF0aW9uCltzZXJ2ZXJdCiMgU2VydmVyIGhvc3RuYW1lOgpob3N0bmFtZSA9IHN1YnNjcmlwdGlvbi5yaHNtLnJlZGhhY2t5LnBocmFkaW5pdC4KCmF1dG9BdHRhY2hJbnRlcnZhbCA9IDE0NQKi
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<repomd xmlns="http://linux.duke.edu/metadata/repo" xmlns:rpm="http://linux.duke.edu/metadata/rpm">
  <revision>1631130504</revision>
</repomd>
```
3. Copy the selected pem file from your entitlement certificate to a local file named nvidia.pem:

```bash
cp <path/to/pem/file>/<certificate-file-name>.pem nvidia.pem
```

4. Generate the MachineConfig file by appending the entitlement certificate:

```bash
sed -i -f - 0003-cluster-wide-machineconfigs.yaml.template << EOF
s/BASE64_ENCODED_PEM_FILE/$(base64 -w0 nvidia.pem)/g
EOF
```

5. Apply the machine config to the OpenShift cluster:

```bash
oc apply -f 0003-cluster-wide-machineconfigs.yaml.template
```

⚠️ This step triggers an update driven by the OpenShift Machine Config Operator, and initiates a restart on all worker nodes, one by one.
6. Check the machineconfig:

```bash
oc get machineconfig | grep entitlement
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>50-entitlement-key-pem</td>
<td>2.2.0</td>
<td>45s</td>
</tr>
<tr>
<td>50-entitlement-pem</td>
<td>2.2.0</td>
<td>45s</td>
</tr>
</tbody>
</table>

7. Monitor the MachineConfigPool object:

```bash
oc get mcp/worker
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>CONFIG</th>
<th>UPDATED</th>
<th>UPDATING</th>
<th>DEGRADED</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker</td>
<td>rendered-worker-5f1eaf24c760fb389d47d3c37ef41c29</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>7h15m</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The status here indicates whether the MCP is updated, not updating or degraded. Make sure all the MachineConfig resources have been successfully applied to the nodes, so you can proceed to validate the cluster.

**Validating the Cluster-wide Entitlement**

Validate the cluster-wide entitlement with a test pod that queries a Red Hat subscription repo for the kernel-devel package.

1. Create a test pod:

```yaml
cat << EOF >> mypod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: cluster-entitled-build-pod
  namespace: default
spec:
  containers:
  - name: cluster-entitled-build
    image: registry.access.redhat.com/ubi8:latest
    command: [ "/bin/sh", "-c", "dnf search kernel-devel --showduplicates" ]
    restartPolicy: Never
EOF
```

2. Apply the test pod:

```bash
oc create -f mypod.yaml
```

```
pod/cluster-entitled-build-pod created
```

3. Verify that the test pod has been created:

```bash
oc get pods -n default
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster-entitled-build-pod</td>
<td>1/1</td>
<td>Completed</td>
<td>0</td>
<td>64m</td>
</tr>
</tbody>
</table>
4. Validate that the pod can locate the necessary kernel-devel packages:

```
 oc logs cluster-entitled-build-pod -n default
```

Any Pod based on RHEL can now run entitled builds.

**Node Feature Discovery**

To enable Node Feature Discovery please follow the Official Guide.

An example of Node Feature Discovery configuration:

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: nfd-instance
  namespace: openshift-nfd
spec:
  operand:
    namespace: openshift-nfd
    imagePullPolicy: Always
  workerConfig:
    configData:
      sources:
        pci:
          deviceClassWhitelist:
            - "02"
            - "03"
            - "0200"
            - "0207"
        deviceLabelFields:
```
Verify that the following label is present on the nodes containing NVIDIA networking hardware:

```
feature.node.kubernetes.io/pci-15b3.present=true
```

$ oc describe node | egrep 'Roles|pci' | grep -v master

```
Roles: worker
  feature.node.kubernetes.io/pci-10de.present=true
  feature.node.kubernetes.io/pci-14e4.present=true
  feature.node.kubernetes.io/pci-15b3.present=true
Roles: worker
  feature.node.kubernetes.io/pci-10de.present=true
  feature.node.kubernetes.io/pci-14e4.present=true
  feature.node.kubernetes.io/pci-15b3.present=true
Roles: worker
  feature.node.kubernetes.io/pci-10de.present=true
  feature.node.kubernetes.io/pci-14e4.present=true
  feature.node.kubernetes.io/pci-15b3.present=true
```

**SR-IOV Network Operator**

If you are planning to use SR-IOV, follow this guide to install SR-IOV Network Operator in OpenShift Container Platform.

Note that the SR-IOV resources created will have the `openshift.io` prefix.

For the default SrioOperatorConfig CR to work with the MOFED container, update the following values:

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: false
  enableOperatorWebhook: false
  configDaemonNodeSelector:
    node-role.kubernetes.io/worker: ""
    network.nvidia.com/operator.mofed.wait: "false"
```

SR-IOV Network Operator configuration documentation can be found on the [Official Website](#).

**GPU Operator**

If you plan to use GPUDirect, follow this guide to install GPU Operator in 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 on OpenShift Using a Catalog**

1. In the OpenShift Container Platform web console side menu, select Operators > OperatorHub, and search for the NVIDIA Network Operator.
2. Select the 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 on OpenShift Using OC CLI**

1. Create a namespace for the Network Operator.
Create the following Namespace custom resource (CR) that defines the nvidia-network-operator namespace, and then save the YAML in the `network-operator-namespace.yaml` file:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: nvidia-network-operator
```

Create the namespace by running the following command:

```bash
$ oc create -f network-operator-namespace.yaml
```

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:

```bash
$ 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:

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: nvidia-network-operator
  namespace: nvidia-network-operator
spec:
  channel: "v23.4.0"
  installPlanApproval: Manual
  name: nvidia-network-operator
  source: certified-operators
  sourceNamespace: openshift-marketplace
```

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

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

5. Change to the network-operator project:

```bash
$ oc project nvidia-network-operator
```

**Verification**

To verify that the operator deployment is successful, run:

```bash
$ oc get pods
```

**Example Output:**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvidia-network-operator-controller-manager-8f8ccf45c-zgfaq</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
</tbody>
</table>

A successful deployment shows a Running status.

**Using Network Operator to Create NicClusterPolicy in an OpenShift Container Platform**
Network Operator Upgrade

Before upgrading to Network Operator v23.1.0 or newer with SR-IOV Network Operator enabled, the following manual actions are required:

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

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

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

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

**Downloading a New Helm Chart**

To obtain new releases, run:

```
# Download Helm chart
$ ls network-operator-*.tgz | xargs -n 1 tar xf
```

**Upgrading CRDs for a Specific Release**

It is possible to retrieve updated CRDs from the Helm chart or from the release branch on GitHub. The example below shows how to download and unpack an Helm chart for a specified release, and apply CRDs update from it.

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

**Preparing the Helm Values for the New Release**

Download the Helm values for the specific release:

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

Known limitations:

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

These limitations will be addressed in future releases.

Changes that were made directly in the `NicClusterPolicy` CR (e.g. with kubectl edit) will be overwritten by the Helm upgrade.

**Temporarily Disabling the Network-operator**

This step is required to prevent the old network-operator version from handling the updated `NicClusterPolicy` CR. This limitation will be removed in future network-operator releases.

```
$ kubectl scale deployment --replicas=0 -n nvidia-network-operator network-operator
```

Please wait for the network-operator pod to be removed before proceeding.

The network-operator will be automatically enabled by the Helm upgrade command. There is no need to enable it manually.
### Applying the Helm Chart Update

To apply the Helm chart update, run:

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

The `--devel` option is required if you wish to use the beta release.

### OFED Driver Manual Upgrade

#### Restarting Pods with a Containerized OFED Driver

This operation is required only if containerized OFED is in use.

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

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

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

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

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

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

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

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

#### Removing Pods with a Secondary Network from the Node

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

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

Replace `<NODE_NAME>` with `-l "network.nvidia.com/operator.mofed.wait=false"` if you wish to drain all nodes at once.

#### Restoring the OFED Driver Pod

Find the OFED driver pod name for the node:

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

Example for Ubuntu 20.04:

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

### Deleting the OFED Driver Pod from the Node
To delete the OFED driver pod from the node, run:

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

A new version of the OFED pod will automatically start.

**Returning Pods with a Secondary Network to the Node**

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

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

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

**Automatic OFED Driver Upgrade**

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

```
nicclusterpolicy.yaml
```{.yaml}

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
  namespace: nvidia-network-operator
spec:
  ofedDriver:
    image: mofed
    repository: nvcr.io/nvidia/mellanox
    version: 23.04-0.5.3.3.1
  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
    # 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
      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
      timeoutSeconds: 300
      # specify if should continue even if there are pods using emptyDir
      deleteEmptyDir: false
```

Apply NicClusterPolicy CRD:

```
$ kubectl apply -f nicclusterpolicy.yaml
```
Node Upgrade States

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

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

Troubleshooting

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

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
  namespace: nvidia-network-operator
spec:
ofedDriver:
  image: mofed
  repository: nvcr.io/nvidia/mellanox
  version: 23.04-0.5.3.3.1
upgradePolicy:
  autoUpgrade: true
  maxParallelUpgrades: 1
  drain:
    enable: true
    force: false
    deleteEmptyDir: true
    podSelector: ""
```

<table>
<thead>
<tr>
<th>Issue</th>
<th>Required Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>The node is in drain-failed state.</td>
<td>Drain the node manually by running <code>kubectl drain &lt;node name&gt; --ignore-daemonsets</code>. Delete the MOFED pod on the node manually, by running the following command: `kubectl delete pod -n &lt;node name&gt; -l nvidia.com/ofed-driver --no-headers</td>
</tr>
<tr>
<td>Manual delete the pod by using <code>kubectl delete -n &lt;Network Operator Namespace&gt; &lt;pod name&gt;</code>.</td>
<td></td>
</tr>
</tbody>
</table>
The updated MOFED pod failed to start; a new version of MOFED cannot be installed on the node.

If following the restart the pod still fails, change the MOFED version in the NicClusterPolicy to the previous version or to other working version.

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 NicClusterPolicy status

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

```
status:
  appliedStates:
  - name: state-pod-security-policy
    state: ignore
  - name: state-multus-cni
    state: ready
  - name: state-container-networking-plugins
    state: ignore
  - name: state-ipoib-cni
    state: ignore
  - name: state-whereabouts-cni
    state: ready
  - name: state-OFED
    state: ready
  - name: state-SRIOV-device-plugin
    state: ignore
  - name: state-ROMA-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
```

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

Uninstalling the Network Operator

Uninstalling Network Operator on a Vanilla Kubernetes Cluster

```
Uninstall Network Operator

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

You should now see all the pods being deleted

```
kubectl get pods -n network-operator
```
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](https://docs.openshift.com/).

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.v23.4.0
  ```

- 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:

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

  Example output:

  ```
  clusterserviceversion.operators.coreos.com "nvidia-network-operator.v23.4.0" deleted
  ```

For additional information, see the [Red Hat OpenShift Container Platform Documentation](https://docs.openshift.com/).

Additional Steps

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 Network Operator CRDs**

```
$ oc delete crds hostdevicenetworks.mellanox.com macvlan networks.mellanox.com nicclusterpolicies.mellanox.com
```
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, it would be cumbersome, and therefore, not recommended.

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 the RDMA Shared Device Plugin

Network operator deployment with the default version of the OFED driver and a single RDMA resource mapped to enp1 netdev:

values.yaml configuration file for such a deployment:

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

### Network Operator Deployment with Multiple Resources in RDMA Shared Device Plugin

Network Operator deployment with the default version of OFED and an RDMA device plugin with two RDMA resources. The first is mapped to enp1 and enp2, and the second is mapped to enp3.

values.yaml configuration file for such a deployment:

```yaml
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: false
# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: true
nvPeerDriver:
  deploy: false
rdmaSharedDevicePlugin:
  deploy: true
  resources:
    - name: rdma_shared_device_a
      ifNames: [ens1f0, ens1f1]
    - name: rdma_shared_device_b
      ifNames: [ens2f0, ens2f1]
sriovDevicePlugin:
  deploy: false
```
Network Operator Deployment with a Secondary Network

Network Operator deployment with:

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

values.yaml:

```yaml
nfd:
  enabled: true
srivoNetworkOperator:
  enabled: false
# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: false

rdmaSharedDevicePlugin:
  deploy: true
  resources:
    - name: rdma_shared_device_a
      ifNames: [ens1f0]

secondaryNetwork:
  deploy: true
  multus:
    deploy: true
  cniPlugins:
    deploy: true
  ipamPlugin:
    deploy: true
```

Network Operator Deployment with NVIDIA-IPAM

Network Operator deployment with:

- RDMA shared device plugin
- Secondary network
- Mutlus CNI
- ContainerNetworking-plugins CNI plugins
- CNI plugins
- NVIDIA-IPAM CNI Plugin

values.yaml:

```yaml
nfd:
  enabled: true
srivoNetworkOperator:
  enabled: false
# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: false

rdmaSharedDevicePlugin:
  deploy: true
  resources:
    - name: rdma_shared_device_a
      ifNames: [ens1f0]

secondaryNetwork:
  deploy: true
  multus:
    deploy: true
  cniPlugins:
    deploy: true
  ipamPlugin:
    deploy: true
```
Example of the MacvlanNetwork that uses NVIDIA-IPAM:

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

**Network Operator Deployment with a Host Device Network**

Network operator deployment with:
- SR-IOV device plugin, single SR-IOV resource pool
- Secondary network
- Multius CNI
- Container networking plugins CNI plugins
- Whereabouts IPAM CNI plugin

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

```yaml
values.yaml:

nfd:
  enabled: true
sriovNetworkOperator:
  enabled: false
  # NicClusterPolicy CR values:
  deployCR: true
ofedDriver:
  deploy: false
rdmaSharedDevicePlugin:
  deploy: false
srivoDevicePlugin:
  deploy: true
resources:
  - name: hostdev
    vendors: [15b3]
secondaryNetwork:
  deploy: true
multus:
  deploy: true
cniPlugins:
  deploy: true
ipamPlugin:
  deploy: true
After deployment, the network operator should be configured, and K8s networking is deployed in order to use it in pod configuration.

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

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

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

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

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

```yaml
apiVersion: v1
description: Pod
metadata:
  name: hostdev-test-pod
annotations:
  k8s.v1.cni.cncf.io/networks: hostdev-net
description: restartPolicy: OnFailure
spec:
  descriptorPolicy: OnFailure
containers:
  name: mofed-test-ctr
  securityContext:
    capabilities:
      add: [ "IPC_LOCK" ]
  resources:
    requests:
      nvidia.com/hostdev: 1
    limits:
      nvidia.com/hostdev: 1
  command:
```
Network Operator Deployment with an IP over InfiniBand (IPoIB) Network

Network operator deployment with:

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

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

Values.yaml:

```
nfd:
  enabled: true

sriovNetworkOperator:
  enabled: false

# NicClusterPolicy CR values:
deployCR: true

ofedDriver:
  deploy: true

rdmaSharedDevicePlugin:
  deploy: true

resources:
  - name: rdma_shared_device_a
    ifNames: [ibs1f0]

secondaryNetwork:
  deploy: true

multus:
  deploy: true

ipoib:
  deploy: true

ipamPlugin:
  deploy: true
```

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

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

```
apiVersion: mellanox.com/v1alpha1
kind: IPoIBNetwork
metadata:
  name: example-ipoibnetwork
spec:
  networkNamespace: "default"
  master: "ibs1f0"
  ipam: |
    | "type": "whereabouts",
    | "datastore": "kubernetes",
    | "kubernetes": {
    |   "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
    | },
    | "range": "192.168.5.225/28",
    | "exclude": {
    |   "192.168.6.229/30",
    |   "192.168.6.236/32"
    | },
    | "log_file": "*/var/log/whereabouts.log",
    | "log_level": "info",
```
The `ipoib-net-ocp.yaml` configuration file for such a deployment in the OpenShift Platform:

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

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

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

Network Operator Deployment for GPUDirect Workloads

GPUDirect requires the following:

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

values.yaml example:

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

secondaryNetwork:
deploy: true
multus:
deploy: true
cniPlugins:
deploy: true
ipamPlugin:
deploy: true

---

host-device-net.yaml:

```yaml
apiVersion: mellanox.com/v1alpha1
deploy: true
kind: HostDeviceNetwork
metadata:
  name: hostdevice-net
spec:
  resourceName: "hostdev"
ipam: |
    
    "whereabouts": "whereabouts",
    "kubernetes": 
      "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig",
    "range": "192.168.3.225/28",
    "exclude": [
      "192.168.3.229/30",
      "192.168.3.236/32"
    ],
    "log_file": "/var/log/whereabouts.log",
    "log_level": "info"
```
Network Operator Deployment in SR-IOV Legacy Mode

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`.

This deployment mode supports SR-IOV in legacy mode.

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

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

Wait for all required pods to be spawned:

```
# kubectl get pod -n nvidia-network-operator | grep sriov
network-operator-sriov-network-operator-544c8d8bb9-vzkmc  1/1   Running   0   5d
```

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.
```
Network Operator Deployment with an SR-IOV InfiniBand Network

Network Operator deployment with InfiniBand network requires the following:

- MOFED and OpenSM running. OpenSM runs on top of the MOFED 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 `deftmember=full` to enable the SR-IOV functionality over InfiniBand.
- For more details, please refer to [this article](#).
- InfiniBand device – Both host device and switch ports must be enabled in InfiniBand mode.
- `rdma-core` package should be installed when an inbox driver is used.

values.yaml:

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

# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: true
rdmaSharedDevicePlugin:
  deploy: false
sriovDevicePlugin:
  deploy: false

secondaryNetwork:
```
deploy: true
multus:
  deploy: true
cniPlugins:
  deploy: true
ipamPlugin:
  deploy: true

sriov-ib-network-node-policy.yaml:

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

sriov-ib-network.yaml:

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

sriov-ib-network-pod.yaml:

apiVersion: v1
kind: Pod
metadata:
  name: test-sriov-ib-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: example-sriov-ib-network
spec:
  containers:
    - name: test-sriov-ib-pod
      image: centos/tools
Network Operator Deployment with an SR-IOV InfiniBand Network with PKey Management

Network Operator deployment with InfiniBand network requires the following:

- MOFED and OpenSM running. OpenSM runs on top of the MOFED 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's documentation.
- InfiniBand device – Both host device and 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.

Values.yaml:

```yaml
name: ib-kubernetes-ufm-secret
namespace: nvidia-network-operator
```

ufm-secret.yaml:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: ib-kubernetes-ufm-secret
  namespace: nvidia-network-operator
```

```yaml
imagePullPolicy: IfNotPresent
cmd: - sh -c sleep inf
securityContext:
capabilities: add: ['IPC_LOCK']
resources:
requests:
  nvidia.com/mlnxics: "1"
limits:
  nvidia.com/mlnxics: "1"
```

```yaml
networkOperator Deployment with an SR-IOV InfiniBand Network with PKey Management

Network Operator deployment with InfiniBand network requires the following:

- MOFED and OpenSM running. OpenSM runs on top of the MOFED 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's documentation.
- InfiniBand device – Both host device and 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.

Values.yaml:

```yaml
name: ib-kubernetes-ufm-secret
namespace: nvidia-network-operator
```

Ufm-secret.yaml:

```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: ""
  UFM_CERTIFICATE: ""

Wait for MOFED to install and apply the following CRs:

sriov-ib-network-node-policy.yaml:

```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: mlnxnic
```

sriov-ib-network.yaml:

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

sriov-ib-network-pod.yaml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-sriov-ib-pod
```
Network Operator Deployment for DPDK Workloads with NicClusterPolicy

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

Network Operator deployment with:

- Host Device Network, DPDK pod

**nicclusterpolicy.yaml**:

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
ofedDriver:
  image: mofed
  repository: nvcr.io/nvidia/mellanox
  version: 23.04-0.5.3.3.1
sriovDevicePlugin:
  image: sriov-network-device-plugin
  repository: ghcr.io/k8snetworkplumbingwg
  version: v3.5.1
config:
  "resourceList": [
    {"resourcePrefix": "nvidia.com",
     "resourceName": "rdma_host_dev",
     "selectors": {
      "vendors": ["15b3"],
      "devices": ["1018"],
      "drivers": ["mlx5_core"]
     }}]
psp:
  enabled: false
secondaryNetwork:
cniPlugins:
  image: plugins
  repository: ghcr.io/k8snetworkplumbingwg
  version: v1.2.0-amd64
ipamPlugin:
  image: whereabouts
  repository: ghcr.io/k8snetworkplumbingwg
  version: v0.6.1-amd64
multus:
```
Deployment Examples For OpenShift Container Platform

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

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

**Network Operator Deployment with a Host Device Network - OCP**

Network Operator deployment with:

- SR-IOV device plugin, single SR-IOV resource pool:
  - There is no need for a secondary network configuration, as it is installed by default in the OCP.

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
ofedDriver:
  image: mofed
  repository: nvcr.io/nvidia/mellanox
  version: 23.04-0.5.3.3.1
startupProbe:
  initialDelaySeconds: 10
  periodSeconds: 20
livenessProbe:
  initialDelaySeconds: 30
  periodSeconds: 30
readinessProbe:
  initialDelaySeconds: 10
  periodSeconds: 30
sriovDevicePlugin:
  image: sriov-network-device-plugin
  repository: ghcr.io/k8snetworkplumbingwg
  version: v3.5.1
config:

```

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

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

```json
"type": "whereabouts",
"datastore": "kubernetes",
"networks": [
  {"kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"},
  {"range": "192.168.3.225/28",
  "exclude": ["192.168.3.229/30",
  "192.168.3.236/32"],
```
The pod.yaml configuration file for such a deployment:

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

**Network Operator Deployment with SR-IOV Legacy Mode - OCP**

This deployment mode supports SR-IOV in legacy mode.

Note that the SR-IOV Network Operator is required as described in the Deployment for OCP section.

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: mofed
    repository: nvcr.io/nvidia/mellanox
    version: 23.04-0.5.3.3.1
  startupProbe:
    initialDelaySeconds: 10
    periodSeconds: 20
  livenessProbe:
    initialDelaySeconds: 30
    periodSeconds: 30
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
```

Sriovnetwork node policy and K8s networking should be deployed.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: nvidia-network-operator
spec:
  deviceType: netdevice
  mtu: 1500
  nicSelector:
```
The `sriovnetwork.yaml` configuration file for such a deployment:

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

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

Note that the resource prefix in this case will be `openshift.io`.

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

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-network
spec:
  containers:
    - name: appcntr1
      image: <image>
      imagePullPolicy: IfNotPresent
      securityContext:
        capabilities:
        - "IPC_LOCK"
      command:
        - sh
        - -c
        - sleep inf
      resources:
        requests:
          openshift.io/sriov_network: '1'
      limits:
        openshift.io/sriov_network: '1'
  nodeSelector:
    feature.node.kubernetes.io/pci-15b3.sriov.capable: "true"
```

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: mofed
  repository: nvcr.io/nvidia/mellanox
  version: 23.04-0.5.3.3.1
startupProbe:
  initialDelaySeconds: 10
  periodSeconds: 20
livenessProbe:
  initialDelaySeconds: 30
  periodSeconds: 30
readinessProbe:
  initialDelaySeconds: 10
  periodSeconds: 30
rdmaSharedDevicePlugin:
  config: |
    
    "configList": [
      
      { "resourceName": "rdma_shared_88", "rdmaHcaMax": 1000,
        "selectors": { "vendors": ["15b3"],
          "deviceIDs": ["101d"],
          "drivers": [],
          "ifNames": ["ens1f0", "ens2f0"],
          "linkTypes": []
        }
      }]
    
    image: k8s-rdma-shared-dev-plugin
    repository: nvcr.io/nvidia/cloud-native
    version: v1.3.2

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

```yaml
apiVersion: mellanox.com/v1alpha1
kind: MacvlanNetwork
metadata:
  name: rdma-shared-88
spec:
  networkNamespace: default
  master: enp4s0f0np0
  mode: bridge
  mtu: 1500
  ipam: '{"type": "whereabouts", "datastore": "kubernetes", "kubernetes": {"kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"}, "range": "16.0.2.0/24",
                "log_file": "/var/log/whereabouts.log", "log_level": "info", "gateway": "16.0.2.1"}'
```

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

```yaml
apiVersion: mellanox.com/v1alpha1
kind: MacvlanNetwork
metadata:
  name: rdma-shared-88
spec:
  networkNamespace: default
  master: enp4s0f0np0
  mode: bridge
  mtu: 1500
  ipam: '{"type": "whereabouts", "range": "16.0.2.0/24", "gateway": "16.0.2.1"}'}
```
SR-IOV Network Operator Deployment - Configuring Several Nodes at the Same Time

To apply SrivNetworkNodePolicy on several nodes in parallel, specify the `maxParallelConfiguration` option in the SrivOperatorConfig CRD:

```yaml
apiVersion: srovnetwork.openshift.io/v1
kind: SrivOperatorConfig
metadata:
  labels:
    app.kubernetes.io/managed-by: Helm
  name: default
  namespace: network-operator
spec:
  configDaemonNodeSelector:
    beta.kubernetes.io/os: linux
    network.nvidia.com/operator.mofed.wait: "false"
    node-role.kubernetes.io/worker: "" 
  enableInjector: false
  enableOperatorWebhook: false
  maxParallelNodeConfiguration: 1
```

Network Operator Deployment for DPDK Workloads - OCP

In order to configure HUGEPAGES in OpenShift, refer to this guide.

For Network Operator configuration instructions, see here.

NicClusterPolicy CRD

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

MacVlanNetwork CRD

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

HostDeviceNetwork CRD

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

IPoIBNetwork CRD

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

Open Source Dependencies
<table>
<thead>
<tr>
<th>Project and Version</th>
<th>Component Name and Branch/Tag</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud.google.com/go:v0.81.0</td>
<td>Google Cloud Client Libraries for Gov0.81.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-ansiterm:d185dfc1b5a126116ea5a19e148e29d16b4574c9</td>
<td>go-ansitermd185dfc1b5a126116ea5a19e148e29d16b4574c9</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/autores/v0.9.13</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/autores/date:v0.3.0</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/autores:v0.11.18</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/logger:v0.2.1</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/tracing:v0.6.0</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest:v14.2.0</td>
<td>go-autorestv14.2.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/beorn7/perks:v1.0.1</td>
<td>beorn7-perksv1.0.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/caarlos0/env/v6:v6.4.0</td>
<td>caarlos0/envv6.4.0</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/cespare/xxhash/v2:v2.1.2</td>
<td>cespare/xxhashv2.1.2</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/chai2010/gettext-go:c61d771bd517099ca0f7a961671fa8608723</td>
<td>chai2010-gettext-go20180126-snapshot-c61d771</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/davecgh/go-spew/v1.1.1</td>
<td>go-spewv1.1.1</td>
<td>ISC</td>
</tr>
<tr>
<td>github.com/emicklei/go-restful:v2.10.0</td>
<td>go-restfulv2.10.0</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/evanphx/json-patch:v4.12.0</td>
<td>evanphx/json-patchv4.12.0</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/exponent-io/jsonpath:d6023ce2651d8eaf5c75bb0cc7167536102ec9f5</td>
<td>exponent-io/jsonpath20151013-snapshot-d6023ce2</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/form3tech-oss/jwt-go:v3.2.3</td>
<td>form3tech-oss/jwt-govv3.2.3</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/fsnotify/fsnotify:v1.5.1</td>
<td>fsnotify/fsnotifyv1.5.1</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/go-errors/errors:v1.0.1</td>
<td>go-errors-errorsv1.0.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/go-logr/logr:v1.2.0</td>
<td>go-logr/logrv1.2.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/go-logr/zapr:v1.2.0</td>
<td>github.com/go-logr/zaprv1.2.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/go-openapi/jsonpointer:v0.19.5</td>
<td>go-openapi/jsonpointerv0.19.5</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/go-openapi/jsonreference:v0.19.5</td>
<td>jsonreferencev0.19.5</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/go-openapi/swag:v0.19.14</td>
<td>swagv0.19.14</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/gogo/protobuf:v1.3.2</td>
<td>gogo-protobufv1.3.2</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/golang/groupcache:41bb18bfe9da5321badc438f91158cd790a33aa3</td>
<td>groupcache20210321-snapshot-41bb18bf</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/golang/protobuf:v1.5.2</td>
<td>golang protobufv1.5.2</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/google/btree:v1.0.1</td>
<td>btreetev1.0.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/google/gnostic:v0.5.7-v3refs</td>
<td>google/gnosticv0.5.7-v3refs</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>Repository</td>
<td>Version</td>
<td>License</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>github.com/google/go-cmp:v0.5.5</td>
<td>google/go-cmpv0.5.5</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/google/gofuzz:v1.1.0</td>
<td>google-gofuzzv1.1.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/google/shlex:e7afc7bc51079733e9468cfd16fcdc7d196cd1d</td>
<td>google-shlex20191202-snapshot-e7afc7bc</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/google/uuid:v1.1.2</td>
<td>google/uuidv.1.1.2</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/gregjones/httpcache:9cad4c344b720d564000afef783878de563a38</td>
<td>gregjones/httpcache20180514-snapshot-9cad4c34</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/imdario/mergo:v0.3.12</td>
<td>merge0.3.12</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/inconshreveable/mousetrap:v1.0.0</td>
<td>inconshreveable/mousetrap1.0.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/josharian/intern:v1.0.0</td>
<td>josharian/internv1.0.0</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/json-iterator/go:v1.1.12</td>
<td>jsoniter-go-v1.1.12</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/k8snetworkplumbingwg/</td>
<td>network-attachment-definition-client:v1.4.0</td>
<td>k8snetworkplumbingwg/network-attachment-definition-clientv1.4.0</td>
</tr>
<tr>
<td>github.com/liggitt/tabwriter:89fcab3d43de07060e42c1547430ed57e87f2d</td>
<td>liggitt/tabwriter20181228-snapshot-89fcab3d</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/mailru/easyjson:v0.7.6</td>
<td>mailru/easyjsonv0.7.6</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/MakeNowJust/heredoc:</td>
<td>bb23615658ced5e105af4ce27de75b089cbe851</td>
<td>MakeNowJust-heredoc20180126-snapshot-bb236154</td>
</tr>
<tr>
<td>github.com/Masterminds/semver/v3.v3.1.1</td>
<td>Masterminds-semverv3.1.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/matttprogd/golang_protobuf_extensions:c182aff4396e30f2343e8b8a7858a75d7d</td>
<td>matttprogd-golang_protobuf_extensions20190325-snapshot-c182aff4396e30f2343e8b8a7858a75d7d</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/mitchell/go-wordwrap:v1.0.0</td>
<td>mitchell-go-wordwrapv1.0.0</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/moby/spdystream:v0.2.0</td>
<td>github.com/moby/spdystreamv0.2.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/moby/term:3f7f6695adc6a35abcb925370dd0a4da5db48ec64d</td>
<td>moby/term:3f7f6695adc6a35abcb925370dd0a4da5db48ec64d</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/monochromegane/go-gitignore:205db1a8ccc01de79237472da52edee4974d734</td>
<td>monochromegane/go-gitignore20200625-snapshot-205db1a8</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/munnerz/goautoneg:</td>
<td>a7dc8b61c822528f7937a5e4e7b27f5056fd6db43e</td>
<td>github.com/munnerz/goautoneg20191010-snapshot-a7dc8b61</td>
</tr>
<tr>
<td>github.com/nxadm/tail:v1.4.8</td>
<td>nxadm/tailv1.4.8</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/onsi/ginkgo:v1.16.5</td>
<td>onsi/ginkgov1.16.5</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/onsi/gomega:v1.18.1</td>
<td>gomega.v1.18.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/openshift/api:a8398931bee7</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/peterbourgon/diskv:v2.0.1</td>
<td>diskv:v2.0.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/pkg/errors:v0.9.1</td>
<td>pkg/errorsv0.9.1</td>
<td>BSD-2-Clause</td>
</tr>
<tr>
<td>github.com/pmezard/go-difflib:v1.0.0</td>
<td>pmezard-go-difflibv1.0.0</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/prometheus/client_golang:v1.12.1</td>
<td>client_golangv1.12.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>Package</td>
<td>Version</td>
<td>License</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>github.com/prometheus/client_model</td>
<td>v0.2.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/prometheus/common</td>
<td>v0.32.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/prometheus/procfs</td>
<td>v0.7.3</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/PuerkitoBio/purell</td>
<td>v1.1.1</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/PuerkitoBio/urlresc</td>
<td>de5bf2ad457846296e2031421a34e2568e304e35</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/russross/blackfriday</td>
<td>v1.5.2</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/spf13/cobra</td>
<td>v1.4.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/spf13/pflag</td>
<td>v1.0.5</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/stretchr/objx</td>
<td>v0.2.0</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/stretchr/testify</td>
<td>v1.7.0</td>
<td>Go Testify-1.7.0</td>
</tr>
<tr>
<td>github.com/xlab/treeprint</td>
<td>a009c3971eca89777614839eb7f69abed3ea3959</td>
<td>MIT</td>
</tr>
<tr>
<td>google.golang.org/appengine</td>
<td>v1.6.7</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>google.golang.org/protobuf</td>
<td>v1.27.1</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>gopkg.in/inf.v0</td>
<td>v0.9.1</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>gopkg.in/tomb.v1</td>
<td>d632973f1e7218eb1089048e0798ec9ae7dceb8</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>gopkg.in/yaml.v2</td>
<td>v2.4.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>gopkg.in/yaml.v3</td>
<td>496545a6307b2a7d7a710fd516e5e16e8ab62dbc</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>Package</td>
<td>Version</td>
<td>License</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>k8s.io/api:v0.24.0</td>
<td>kubernetes/api0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/apiregistration:v0.24.0</td>
<td>kubernetes/apiregistration0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/apimachinery:v0.24.0</td>
<td>kubernetes/apimachinery0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/cli-runtime:v0.24.0</td>
<td>k8s.io/cli-runtime0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/client-go:v0.24.0</td>
<td>client-go0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/component-base:v0.24.0</td>
<td>kubernetes/component-base0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/klog/v2:v2.60.1</td>
<td>k3s.io/klogv2.60.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/kube-openapi:3ee0da9b0b42</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/kubectl:v0.24.0</td>
<td>kubectl0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/utils:3a6ce19f1f9</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>sigs.k8s.io/controller-runtime:v0.12.1</td>
<td>sigs.k8s.io/controller-runtime0.12.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>sigs.k8s.io/kustomize/api:v0.11.4</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>sigs.k8s.io/kustomize/kyma:v0.13.6</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>sigs.k8s.io/structured-merge-diff/v4:v4.2.1</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>sigs.k8s.io/yaml/v1.4.0</td>
<td>sigs.k8s.io/yamlv1.4.0</td>
<td>MIT</td>
</tr>
</tbody>
</table>