Network Operator

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 K8s 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.

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

Network Operator Deployment from NGC:

To install the operator with chart default values, run:

```
# Download Helm chart
$ helm fetch https://helm.ngc.nvidia.com/nvidia/cloud-native/charts/network-operator-1.4.0.tgz
$ ls network-operator-*.tgz | xargs -n 1 tar xf

# Install Operator
$ helm install -n network-operator --create-namespace network-operator ./network-operator

# View deployed resources
$ kubectl -n network-operator get pods
$ kubectl get pod -n nvidia-network-operator-resources
```

Network Operator Deployment from GitHub:

To install the operator with chart default values, run:

```
# Add Repo
$ helm repo add NVIDIA https://mellanox.github.io/network-operator
$ helm repo update

# Install Operator
$ helm install -n network-operator --create-namespace --wait network-operator NVIDIA/network-operator

# View deployed resources
$ kubectl -n network-operator get pods
$ kubectl get pod -n nvidia-network-operator-resources
```

Helm Chart Customization Options

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

### General 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>
</tbody>
</table>

psp.enabled | Bool | False | Deploy Pod Security Policy.

operator.repository | String | nvcr.io/nvidia/cloud-native | Network Operator image repository.

operator.image | String | network-operator | Network Operator image name.

operator.tag | String | None | Network Operator image tag. If set to None, the chart’s appVersion will be used.

operator.imagePullSecrets | List | [] | An optional list of references to secrets to use for pulling any of the Network Operator image.

deployCR | Bool | false | Deploy NicClusterPolicy custom resource according to the provided parameters.

### NicClusterPolicy Custom Resource Parameters

#### NVIDIA 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>5.8-1.0.1.2</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.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>
</tbody>
</table>

#### 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 gpu-operator)</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>
</tbody>
</table>
RDMA Device Plugin Resource Configurations

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 RDMA Shared Device Plugin Selectors for supported selectors.

```
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

```
Name                  Type       Default       Description
sriovDevicePlugin.deploy  Bool      false        Deploy SR-IOV Network device plugin
sriovDevicePlugin.repository  String   ghcr.io/k8snetworkplumbingw g SR-IOV Network device plugin image repository
sriovDevicePlugin.image  String   sriov-network-device-plugin SR-IOV Network device plugin image name
sriovDevicePlugin.version  String   v3.5.1       SR-IOV Network device plugin version
sriovDevicePlugin.imagePullSecrets  List   []          An optional list of references to secrets to use for pulling any of the SR-IOV Network device plugin image
sriovDevicePlugin.resources  List   See below    SR-IOV Network device plugin resources
```

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.
### 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
kind: Secret
metadata:
  name: ib-kubernetes-ufm-secret
  namespace: kube-system
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

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/k8snetworkplumbingw</td>
<td>CNI Plugins image repository</td>
</tr>
<tr>
<td>secondaryNetwork.cniPlugins.version</td>
<td>String</td>
<td>v0.8.7-amd64</td>
<td>CNI Plugins image version</td>
</tr>
</tbody>
</table>
secondaryNetwork.cniPlugins.imagePullSecrets | List | [] | An optional list of references to secrets to use for pulling any of the CNI Plugins images

### 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>v3.8</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>
<tr>
<td>secondaryNetwork.multus.config</td>
<td>String</td>
<td><code>&quot;</code></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>
</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.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/k8snetworkplumbingwg</td>
<td>IPAM CNI Plugin image repository</td>
</tr>
<tr>
<td>secondaryNetwork.ipamPlugin.version</td>
<td>String</td>
<td>v0.5.2-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>

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 network-operator --create-namespace --wait NVIDIA/network-operator network-operator
```

By default, the Network Operator deploys the Node Feature Discovery (NFD), in order to perform node labeling in the cluster. This allows proper scheduling of Network Operator resources.

If the nodes have already been labeled by other means, it is possible to disable the deployment of the NFD by setting the `nfd.enabled=false` chart parameter:

```
$ helm install --set nfd.enabled=false -n network-operator --create-namespace --wait network-operator NVIDIA/network-operator
```

Currently, the following NFD labels are used:

<table>
<thead>
<tr>
<th>Label</th>
<th>Location</th>
</tr>
</thead>
</table>

```
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 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:

```yaml
privileged: true
hostIPC: false
hostNetwork: true
hostPID: false
allowPrivilegeEscalation: true
readOnlyRootFilesystem: false
allowedHostPaths: []
allowedCapabilities: - '*'
fsGroup:
  rule: RunAsAny
runAsUser:
  rule: RunAsAny
seLinux:
  rule: RunAsAny
supplementalGroups:
  rule: RunAsAny
volumes:
- configMap
- hostPath
- secret
- downwardAPI
```

PodSecurityPolicy is deprecated as of Kubernetes v1.21 and will be removed in v1.25.

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

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

```
$ kubectl label --overwrite ns network-operator pod-security.kubernetes.io/enforce=privileged
```
In case that baseline or restricted levels are being enforced on the Network Operator namespace, events for pods creation failures will be triggered:

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

Network Operator Deployment in 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.

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

Prerequisites

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

HTTP Proxy Configuration for Openshift

For Openshift, it is recommended to use the cluster-wide Proxy object to provide proxy information for the cluster. Please follow the procedure described in Configuring the Cluster-wide Proxy via the Red Hat Openshift public documentation. The 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: http_proxy
      value: http://<example.proxy.com:port>
    - name: https_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:

- **ubuntu:**
  - `linux-headers-${KERNEL_VERSION}`
  - `linux-modules-${KERNEL_VERSION}`

- **rhcos:**
  - `kernel-headers-${KERNEL_VERSION}`
  - `kernel-devel-${KERNEL_VERSION}`
  - `kernel-core-${KERNEL_VERSION}`
  - `createrepo`
  - `elfutils-libelf-devel`
  - `kernel-rpm-macros`
  - `numactl-libs`

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

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

```bash
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:

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

Create the ConfigMap for RHCOS:

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

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

```bash
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

Please follow the [GPU Operator Guide](#) to enable cluster-wide entitlement.

#### Node Feature Discovery

To enable Node Feature Discovery please follow the [Official Guide](#).

An example of Node Feature Discovery configuration:

```yaml
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:
              - vendor
      customConfig:
        configData: ""
```

Verify that the following label is present on the nodes containing NVIDIA networking hardware:

```bash
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 SriovOperatorConfig CR to work with the MOFED container, update the following values:

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
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 Using an OpenShift Container Platform Console

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 Using CLI

1. Create a namespace for the Network Operator.
   Create the following Namespace custom resource (CR) that defines the 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:
1. $ 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:

   $ 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: "v1.4.0"
     installPlanApproval: Manual
     name: nvidia-network-operator
     source: certified-operators
     sourceNamespace: openshift-marketplace
   ```

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

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

5. Change to the network-operator project:

   $ oc project nvidia-network-operator

Verification

To verify that the operator deployment is successful, run:

   $ oc get pods

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-zgfsq</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
</tbody>
</table>

A successful deployment shows a Running status.

**Network Operator Configuration in an OpenShift Container Platform**

In OCP, it is required to create the 'nvidia-network-operator-resources' namespace manually before creating the NicClusterPolicy CR.

Run:

   $ oc create ns nvidia-network-operator-resources
Uninstalling the Network Operator on an OpenShift Container Platform

Network Operator Uninstallation Using an OpenShift Container Platform 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.

Network Operator Uninstallation Using CLI in OpenShift Container Platform

- 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.v1.4.0

- 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 previous step:

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

  Example output:

  clusterserviceversion.operators.coreos.com "nvidia-network-operator.v1.2.1" deleted

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

Additional Steps

1. Remove namespaces:

   In OCP, it is required to delete the 'nvidia-network-operator-resources' and 'nvidia-network-operator' namespaces manually after uninstalling the Network Operator.

   Run:

   $ oc delete ns nvidia-network-operator-resources nvidia-network-operator

2. Run:

   $ oc delete ns nvidia-network-operator-resources nvidia-network-operator

3. Remove CRDs and CRs:

   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.

   Run:
Uninstalling the Network Operator

To uninstall the operator, run:

```bash
$ helm delete -n network-operator $(helm list -n network-operator | grep network-operator | awk '{print $1}')
$ kubectl -n network-operator delete daemonsets.apps sriov-device-plugin
```

You should now see all the pods being deleted:

```bash
$ kubectl get pods -n nvidia-network-operator-resources
No resources found.
$ kubectl get pods -n network-operator
No resources found.
```

In addition, make sure that the CRDs created during the operator installation have been removed:

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

When installing the Network Operator with MOFED in containers, it is required to reload the mlx5_core kernel module for Ethernet NICs, and the ib_ipoib for InfiniBand NICs after MOFED is uninstalled.

Network Operator Upgrade

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

Searching for Available Releases

To find available releases, run:

```bash
$ helm search repo NVIDIA/network-operator -l
```

Add the `--devel` option if you wish to list Beta releases as well.

Downloading 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.

```bash
$ helm pull NVIDIA/network-operator --version <VERSION> --untar --untardir network-operator-chart
```

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

**Restarting the OFED Driver Pod**

Find the OFED driver pod name for the node:

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

Example for Ubuntu 20.04:

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

**Deleting the OFED Driver Pod from the Node**

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

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

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.
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
  namespace: nvidia-network-operator
spec:
ofedDriver:
  image: mofed
  repository: mellanox
  version: 5.8-1.0.1.1.2
upgradePolicy:
  # autoUpgrade is a global switch for automatic upgrade feature
  # if set to false all other options are ignored
  autoUpgrade: true
  # maxParallelUpgrades indicates how many nodes can be upgraded in parallel
  # 0 means no limit, all nodes will be upgraded in parallel
  maxParallelUpgrades: 1
  # 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

⚠️ To be able to drain nodes, please make sure to fulfill PodDisruptionBudget for all the pods that use it.

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>This value is set when the upgrade flow is disabled or the node has not been processed yet.</td>
</tr>
<tr>
<td>upgrade-done</td>
<td>This value is set when the OFED pod is up to date and running on the node, and the node is schedulable - UpgradeStateDone = &quot;upgrade-done&quot;.</td>
</tr>
<tr>
<td>upgrade-required</td>
<td>This value is set when the OFED pod on the node is not up-to-date and requires upgrade. No actions are performed at this stage.</td>
</tr>
<tr>
<td>drain</td>
<td>This value is set when the node is scheduled for drain. Following the drain, the state is changed either to pod-restart or to drain-failed UpgradeStateDrain = &quot;drain&quot;.</td>
</tr>
<tr>
<td>pod-restart</td>
<td>This value is set when the OFED pod on the node is scheduled for restart. Following the restart, the state is changed to uncordon-required.</td>
</tr>
<tr>
<td>drain-failed</td>
<td>This value is set when the drain on the node has failed. A manual interaction is required at this stage. See the Troubleshooting section for more details.</td>
</tr>
<tr>
<td>uncordon-required</td>
<td>This value is set when the OFED pod on the node is up-to-date, and has a &quot;Ready&quot; status. After the uncordon command, the state is changed to upgrade-done.</td>
</tr>
</tbody>
</table>
Troubleshooting

<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: <code>kubectl delete pod -n </code>kubectl get pods --A --field-selector spec. nodeName=&lt;node name&gt; -l nvidia.com/ofed-driver --no-headers</td>
</tr>
<tr>
<td>The updated MOFED pod failed to start/ a new version of MOFED cannot be installed on the node.</td>
<td>Manually delete the pod by using <code>kubectl delete -n &lt;Network Operator Namespace&gt; &lt;pod name&gt;</code>. If following the restart the pod still fails, change the MOFED version in the NicClusterPolicy to the previous version or to other working version.</td>
</tr>
</tbody>
</table>

Deployment Examples

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 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:
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:

```
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 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:
Network Operator Deployment with a Host Device Network

Network operator deployment with:

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

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

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

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

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

The **pod.yaml** configuration file for such a deployment:
Network Operator Deployment with an IP over InfiniBand (IPoIB) Network

Network operator deployment with:

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

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

values.yaml:

```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
class: IPoIBNetwork
metadata:
  name: example-ipoibnetwork
spec:
  networkNamespace: "default"
  master: "ibs1f0"
ipam:
  - type: "whereabouts",
    datastore: "kubernetes",
    kubernetes:
      kubeconfig: "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
    range: "192.168.5.225/28",
    exclude:
      - 192.168.6.229/30
      - 192.168.6.236/32
    log_file: "/var/log/whereabouts.log",
    log_level: "info",
    gateway: "192.168.6.1"

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

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

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

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

host-net-gpudirect-pod.yaml:
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.

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

nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true

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

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

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

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

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: "example-sriov-network"
  namespace: 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"
```

The `ens2f0` network interface name has been chosen from the following command output:
kubectl -n network-operator get sriovnetworknodestates.sriovnetwork.openshift.io.
status:
  interfaces:
  - deviceID: 101d
    driver: mlx5_core
    linkSpeed: 100000 Mb/s
    linkType: ETH
    mac: 0c:42:a1:2b:74:ae
    mtu: 1500
    name: ens2f0
    pciAddress: "0000:07:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 101d
    driver: mlx5_core
    linkType: ETH
    mac: 0c:42:a1:2b:74:af
    mtu: 1500
    name: ens2f1
    pciAddress: "0000:07:00.1"
    totalvfs: 8
    vendor: 15b3

Wait for all required pods to be spawned:

```bash
# kubectl get pod -n network-operator | grep sriov
network-operator-sriov-network-operator-544c8d8b9b-vzkmc    1/1     Running   0          5d
sriov-cni-qgblf                                          2/2     Running   0          2d6h
sriov-device-plugin-vwpzn                                 1/1     Running   0          2d6h
sriov-network-config-daemon-qv467                         1/1     Running   0          5d

# kubectl get pod -n nvidia-network-operator-resources
NAME                                            READY   STATUS    RESTARTS   AGE
--                                             -------- -------- --------- -----
cni-plugins-ds-kbvnm                            1/1     Running   0          5d
cni-plugins-ds-pc1lg                             1/1     Running   0          5d
cube-multus-ds-5j6ns                             1/1     Running   0          5d
kube-multus-ds-mxqvl                             1/1     Running   0          5d
mofed-ubuntu20.04-ds-2zzf4                        1/1     Running   0          5d
mofed-ubuntu20.04-ds-rfnsw                        1/1     Running   0          5d
whereabouts-nw7hn                                 1/1     Running   0          5d
whereabouts-zvhrv                                 1/1     Running   0          5d
```

pod.yaml configuration file for such a deployment:
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 defmember=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: network-operator
spec:
  deviceType: netdevice
  mtu: 1500
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "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: 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:
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.

values.yaml:

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

# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: true
rdmaSharedDevicePlugin:
  deploy: false
sriovDevicePlugin:
  deploy: false
ibKubernetes:
  deploy: true
periodicUpdateSeconds: 5
pKeyGUIDPoolRangeStart: "02:00:00:00:00:00:00:00"
ufmSecret: ufm-secret

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

ufm-secret.yaml:

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

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

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

sriov-ib-network.yaml:

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ib-sriov-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: mlnxnics
spec:
  config: '{
    "type": "ib-sriov",
    "cniVersion": "0.3.1",
    "name": "ib-sriov-network",
    "pkey": "0x6",
    "link_state": "enable",
    "ibKubernetesEnabled": true,
    "ipam": {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "routes": [{
        "dst": "0.0.0.0/0"
      },
      "gateway": "10.56.217.1"
    }
  }'

sriov-ib-network-pod.yaml:
Network Operator Deployment for DPDK Workloads with NicClusterPolicy

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

Network Operator deployment with:

- Host Device Network, DPDK pod

```
nicclusterpolicy.yaml:
```
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: mofed
    repository: nvcr.io/nvidia/mellanox
    version: 5.8-1.0.1.1.2
  sriovDevicePlugin:
    image: sriov-network-device-plugin
    repository: ghcr.io/k8snetworkplumbingwg
    version: a765300344368efbf43f71016e9641c58ec1241b
    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: v0.8.7-amd64
    ipamPlugin:
      image: whereabouts
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.4.2-amd64
    multus:
      image: multus-cni
      repository: ghcr.io/k8snetworkplumbingwg
      version: v3.8
  secondaryNetwork:
    cniPlugins:
      image: plugins
      repository: ghcr.io/k8snetworkplumbingwg
      version: v0.8.7-amd64

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

pod.yaml:

apiVersion: v1
kind: Pod
metadata:
  name: testpod1
annotations:
  k8s.v1.cni.cncf.io/networks: example-hostdev-net
spec:
  containers:
  - name: appcntrl
    image: <dpdk image>
    imagePullPolicy: IfNotPresent
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
    volumeMounts:
      - mountPath: /dev/hugepages
        name: hugepage
    resources:
      requests:
        memory: 1Gi
        hugepages-1Gi: 2Gi
        nvidia.com/rdma_host_dev: '1'
    command: ["/bin/bash", "-c", "--"]
    args: ["while true; do sleep 300000; done;" ]
    volumes:
      - name: hugepage
        emptyDir:
          medium: HugePages

NicClusterPolicy CRD

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

MacVlanNetwork CRD

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

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: 5.8-1.0.1.1.2
startupProbe:
  initialDelaySeconds: 10
  periodSeconds: 20
livenessProbe:
  initialDelaySeconds: 30
  periodSeconds: 30
readinessProbe:
  initialDelaySeconds: 10
  periodSeconds: 30
sriovDevicePlugin:
  image: sriov-network-device-plugin
  repository: ghcr.io/k8snetworkplumbingwg
  version: a765300344368efbf43f71016e9641c58ec1241b
  config:
    "resourceList": [
      
        "resourcePrefix": "nvidia.com",
        "resourceName": "host_dev",
        
        "selectors": {
          "vendors": ["15b3"],
          "isRdma": true
        },
    ]
```
apiVersion: mellanox.com/v1alpha1
kind: HostDeviceNetwork
metadata:
    name: hostdev-net
spec:
    networkNamespace: "default"
    resourceName: "nvidia.com/hostdev"
    ipam: |
        
        |
        | "type": "whereabouts",
        | "datastore": "kubernetes",
        | "kubernetes": {  
        |     "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
        | },
        | "range": "192.168.3.225/28",
        | "exclude": [  
        |     "192.168.3.229/30",
        |     "192.168.3.236/32"
        | ],
        | "log_file": "*/var/log/whereabouts.log",
        | "log_level": "info"
    |

The pod.yaml configuration file for such a deployment:

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

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

This deployment mode supports SR-IOV in legacy mode.

Note that the SR-IOV Network Operator is required as described in the Deployment for OCP section.
Sriovnetwork node policy and K8s networking should be deployed.

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

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: network-operator
spec:
  deviceType: netdevice
  mtu: 1500
  nicSelector:
    vendor: "15b3"
    pfNames: ["ens2f0"]
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 5
  priority: 90
  isRdma: true
  resourceName: sriovlegacy
```

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

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: "sriov-network"
  namespace: 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`. 

```yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  ofedDriver:
    image: mofed
    repository: nvcr.io/nvidia/mellanox
    version: 5.8-1.0.1.1.2
  startupProbe:
    initialDelaySeconds: 10
    periodSeconds: 20
  livenessProbe:
    initialDelaySeconds: 30
    periodSeconds: 30
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
Sriovnetwork node policy and K8s networking should be deployed.
sriovnetwork-node-policy.yaml configuration file for such a deployment:
```
The pod.yaml configuration file for such a deployment:

```yaml
apiVersion: v1
class: Pod
metadata:
  name: testpod1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-network
spec:
  containers:
    - name: appcntr1
      image: <image>
      imagePullPolicy: IfNotPresent
      securityContext:
        capabilities:
          add: ["IPC_LOCK"]
      command:
        - sh
        - -c
        - sleep inf
      resources:
        requests:
          openshift.io/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: 5.8-1.0.1.1.2
  startupProbe:
    initialDelaySeconds: 10
    periodSeconds: 20
  livenessProbe:
    initialDelaySeconds: 30
    periodSeconds: 30
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
rdmaSharedDevicePlugin:
  config: |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
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.

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.

Example Status Field of a NICClusterPolicy Instance
An "Ignore" state indicates that the sub-state was not defined in the custom resource, and thus, it is ignored.

### 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/autorest/adal:v0.9.13</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/autorest/date:v0.3.0</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/Azure/go-autorest/autorest: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:v6fed771bf6517099ca07a961671fa8ed08723</td>
<td>chai2010-gettext-go20180126-snapshot-c6fed771</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/davecgh/go-speew:v1.1.1</td>
<td>go-speewv1.1.1</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-path:v4.12.0</td>
<td>evanphx/json-patchv4.12.0</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/exponent-io/jsonpath:d6023ce2651d8eaf5b575bb0c7167536102ec9f5</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-gov3.2.3</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>Package Name</td>
<td>Version</td>
<td>License</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>github.com/modern-go/concurrent</td>
<td>bacd9c7ef1d9b15be4a990b8ac7a4e313ee94</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/modern-go/reflect2</td>
<td>v1.0.2</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/monochromegane/go-gitignore</td>
<td>205db1a8001de79290d2f2dde994374</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/munnerz/goautoneg</td>
<td>a7dc8b61c225289f73ase47b272055c6db4e</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/nxadm/tail</td>
<td>v1.4.8</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/onsi/ginkgo</td>
<td>v1.16.5</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/onsi/gomega</td>
<td>v1.18.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/openstack/api</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>github.com/peterbourgon/diskv</td>
<td>v2.0.1</td>
<td>MIT</td>
</tr>
<tr>
<td>github.com/pkg/errors</td>
<td>v0.9.1</td>
<td>BSD-2-Clause</td>
</tr>
<tr>
<td>github.com/pmezard/go-difflib</td>
<td>v1.0.0</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>github.com/prometheus/client_golang</td>
<td>v1.12.1</td>
<td>Apache-2.0</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/procsfs</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/urlesc</td>
<td>de5bf2ad457846296e2031a34e2568304e35</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>MIT</td>
</tr>
<tr>
<td>github.com/xlab/treeprint</td>
<td>a009c3971eca89777614839eb7169abed3ea3959</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>go.uber.org/atomic</td>
<td>v1.7.0</td>
<td>MIT</td>
</tr>
<tr>
<td>go.uber.org/multierr</td>
<td>v1.6.0</td>
<td>MIT</td>
</tr>
<tr>
<td>go.uber.org/zap</td>
<td>v1.19.1</td>
<td>MIT</td>
</tr>
<tr>
<td>golang.org/x/crypto</td>
<td>86341886e292</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>golang.org/x/net</td>
<td>7d36cc0744dd69565798f1508446dc81e166fc</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>golang.org/x/oauth2</td>
<td>d3ed0bb246c8d3c7b5e3937da5e9c9c74d7fe</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>Project</td>
<td>Version</td>
<td>License</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>golang.org/x/sys</td>
<td>3681064d51587c1db0324b3d5c23c2ddbcff6e8f</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>golang.org/x/term</td>
<td>03fcf44c2211dcd5eb77510b57c1f6026d6ded50</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>golang.org/x/text</td>
<td>v0.3.7</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>golang.org/x/time</td>
<td>90d013bbcefe8e15b6f78023a0e3b996267153e7d</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>gomodules.xyz/jsonpatch/v2</td>
<td>v2.2.0</td>
<td>Apache-2.0</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>ddd329731e7218eb1089048e0798ec9ae7dceb8</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>496545a6307b2a7d7a710f516e5e16e8ab62dbc</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/api/v0.24.0</td>
<td>kubernetes/apiv0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/apiextensions-apiserver/v0.24.0</td>
<td>kubernetes/apiextensions-apiserverv0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/apimachinery/v0.24.0</td>
<td>kubernetes/apimachineryv0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/cli-runtime/v0.24.0</td>
<td>k8s.io/cli-runtimev0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/client-go/v0.24.0</td>
<td>client-go/v0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/component-base/v0.24.0</td>
<td>kubernetes/component-basev0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/klog/v2</td>
<td>k3s-io/klogv2.60.1</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/kube-openapi/v3ee0da9b0b42</td>
<td>N/A</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/kubectl/v0.24.0</td>
<td>kubectlv0.24.0</td>
<td>Apache-2.0</td>
</tr>
<tr>
<td>k8s.io/utils/3a6ce19f2f9</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-runtimev0.12.1</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/kyaml/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>