Installing OpenShift on DGX

A100, H100, H200, B200

Version 1.2

Red Hat

Oct 2025

Red Hat

Contents

Version History

v1.1	Support for DGX A100, H100, and H200 Servers
v1.2	Support for B200 Servers

Introduction

OpenShift is an Enterprise-grade container-management solution based on Kubernetes for automating deployment, scaling, and management of containerized applications. It is developed and supported by Red Hat and includes additional security features and tooling for managing complex infrastructures on-premises as well as in hybrid cloud installations.

Red Hat OpenShift 4 is a major release upgrade from version 3 incorporating many technologies from the acquisition of CoreOS. This release follows a new paradigm where systems are always reimaged with the latest version with only minimal provisioning. At its core are the immutable Red Hat CoreOS (RHCOS) system images based on Red Hat Enterprise Linux 9. All additional software, drivers, and configuration are ephemeral and provided through kubernetes primitives, such as containers, deployments, and operators. This includes the NVIDIA GPU Operator for supporting NVIDIA GPUs and the NVIDIA Network Operator for the ConnectX network interfaces.

This document provides additional information for installing and configuring OpenShift 4 on clusters incorporating DGX worker nodes. It should be seen as a companion document to the official Red Hat OpenShift documentation. The following chapters describe additional configuration steps and best practices that are specific to NVIDIA DGX™ systems. Refer to the OpenShift Container Platform Documentation for generic information about OpenShift and installation instructions.

Customer Support

Customer support for running OpenShift on DGX systems is provided by Red Hat for OpenShift and NVIDIA for the DGX platform and drivers. For OpenShift support, visit the Red Hat Enterprise Support website: https://www.redhat.com/en/services/support. For DGX hardware, firmware / drivers, or NGC application issues, visit the NVIDIA Enterprise Support website: https://www.nvidia.com/en-us/support/enterprise/.

Additional Documentation

Refer to the following documents for additional Information:

- Red Hat OpenShift Product page
- OpenShift Container Platform Documentation
- GPU Operator on OpenShift
- Network Operator on OpenShift
- NVIDIA System Management Documentation

Prerequisites

To use this document it is expected that the following prerequisites are met:

Red Hat Subscription

Installing and running OpenShift requires a Red Hat account and additional subscriptions. Refer to Red Hat OpenShift for more information.

Helm Management Tool

NVIDIA System Management (NVSM) uses Helm for installing NVSM on DGX worker nodes. NVSM is a software framework for collecting health status information and helps users analyze hardware and software issues. Refer to Installing Helm for instructions on installing the Helm tool on the system you use to interact with the OpenShift cluster.

Image Registry

An image registry is required for disconnected environments.

DNS

DNS records are required for the installation of OpenShift.

Control Plane

Three nodes, whether virtual or physical, are required for an OpenShift control plane. These machines need to have the resource requirements for a control plane.

Support matrix

DGX B200

Red Hat OpenShift	GPU operator	Network Operator	NFD Operator	NVSM
4.18	25.3.4	25.7.0	4.18	1.0.1
4.17	25.3.4	25.7.0	4.17	1.0.1

DGX H100

Red Hat OpenShift	GPU operator	Network Operator	NFD Operator	NVSM
4.18	25.3	25.4	4.18	1.0.1
4.17	24.9	25.1	4.17	1.0.1
4.16	24.6	24.1	4.16	1.0.1
4.15	24.3	24.7	4.15	1.0.1

DGX A100

Red Hat OpenShift	GPU operator	Network Operator	NFD Operator	NVSM
4.18	25.3	25.4	4.18	1.0.1
4.17	24.9	25.1	4.17	1.0.1
4.16	24.6	24.1	4.16	1.0.1
4.15	24.3	24.7	4.15	1.0.1

Installation Workflow

The following list highlights the workflow for installing Red Hat OpenShift and the components required to enable all the necessary operators.

1. Installing Red Hat OpenShift
2. Installing the NFD Operator
3. Installing the SRIOV Operator (Optional)
4. Installing the NVIDIA Network Operator
5. Installing the NVIDIA GPU Operator
6. Installing NVSM Operator
7. Validating RDMA Networking
8. Using NVSM Operator

Installing Red Hat OpenShift

Note:
If using static networking on the DGX nodes make sure to have the MAC addresses from the active Ethernet interfaces and networking details like ip addresses per node as this information will be required for the discovery image.

This is specifically relevant for DGX systems with bonded configurations by default, where one port is often set to InfiniBand and the other to Ethernet.

You can deploy an OpenShift Container Platform cluster to on-premise hardware using virtual machines or non DGX baremetal servers for the control plane with DGX worker nodes using one of the following installation methods:

Connected: You can deploy a cluster with the web-based Assisted Installer. This is an ideal approach for clusters with networks connected to the internet. The Assisted Installer is the easiest way to install OpenShift Container Platform - it provides smart defaults, and it performs pre-flight validations before installing the cluster. It also provides a RESTful API for automation and advanced configuration scenarios. Start installing your cluster using the Hybrid Cloud Console.

Note:
Red Hat recommends that when installing on DGX nodes using a virtual media, it is preferable to use the *minimal installation* ISO to provide the best user experience. Using this method allows the local installer to manage the install and mitigates potential network interruptions that can occur.

Disconnected: You can deploy a cluster locally with the Agent-based Installer for disconnected environments or restricted networks. It provides many of the benefits of the Assisted Installer. Configuration is done with a command-line interface. This approach is ideal for disconnected environments.

Start installing a disconnected cluster downloading and the Agent-based Installer and follow the Agent-based Installer workflow. More specific details can be found in Installing in an air-gapped environment in the Advanced Installation Workflow.

Troubleshooting: For troubleshooting Assisted Installer deployments see the following Assisted Installer Troubleshooting Guide.

Installing the Red Hat NFD Operator

The Node Feature Discovery (NFD) Operator manages the detection of hardware features and configuration in an OpenShift Container Platform cluster by labeling the nodes with hardware specific information. NFD labels the host with node-specific attributes, such as PCI cards, kernel, operating system version, and so on.

Procedure

1. Install the NFD Operator using the procedure described here: Installing the NFD Operator using the CLI or Installing the NFD Operator via Web Console.
2. Configure the NFD Operator using the procedure described here: Configuring the NFD Operator.

Installing the SRIOV Operator

The Single Root I/O Virtualization (SR-IOV) Network Operator on the cluster manages SR-IOV network devices and network attachments. It allows the ability to create virtual functions from single network adapters. In the case of DGX, this is completely optional but will be needed if you

are planning on doing RDMA via SRIOV legacy as outlined in the Advanced Installation Workload section.

Note: You can also use the web console to install this Operator. For more information, refer to Web console: Installing the SR-IOV Network Operator.

Procedure

1. Install the SRIOV Operator using the procedure described here: Installing the SRIOV Network Operator.
2. Configure the SRIOV Operator using the procedure described here: Configuring the SRIOV Operator.

Installing the Network Operator

The NVIDIA Network Operator simplifies the provisioning and management of NVIDIA networking resources in a Kubernetes cluster. The Operator automatically installs the required host networking software , bringing together all the needed components to provide high-speed network connectivity. These components include the NVIDIA networking driver, Kubernetes device plugin, CNI plugins, IP address management (IPAM) plugin, and others. The NVIDIA Network Operator works in conjunction with the NVIDIA GPU Operator to deliver high-throughput, low-latency networking for scale-out, GPU computing clusters.

Procedure

1. Install the NVIDIA Network Operator using the procedure described here: NVIDIA Network Operator Installation
2. Configure the NVIDIA Network Operator using the procedure described here: Configuring the NVIDIA Network Operator.

TIP: Some modules, like the irdma kernel module, can prevent the Network Operator drivers from loading properly. There is more information on how to prevent certain modules from loading following the procedure here: Disabling the irdma Kernel Module.

TIP: Before installing the Network Operator it might make sense to set a core user password on the worker nodes in case we need to troubleshoot. The procedure is described here in the Red Hat OpenShift documentation.

Note: Ensure the NicClusterPolicy is created after the installation of the Network Operator is completed and before proceeding.

Installing the GPU Operator

The NVIDIA GPU Operator uses the operator framework within Kubernetes to automate the management of all NVIDIA software components needed to provision GPU. These components include the NVIDIA drivers (to enable CUDA), Kubernetes device plugin for GPUs, the NVIDIA Container Toolkit, automatic node labelling using GFD, DCGM based monitoring and others.

Procedure

1. Install the NVIDIA GPU Operator using the procedure described here: Installing the GPU Operator.
2. Configure the NVIDIA GPU Operator using the procedure described here: Configuring the GPU Operator.

Note: Ensure the GPU ClusterPolicy is created immediately after the GPU Operator is installed regardless of installation method.

Applying System Optimizations

This section describes additional kernel configurations and other system settings for providing optimal performance for workloads on the worker nodes.

They can be applied any time after completing the OpenShift installation as a Day-2 configuration change using the Machine Configuration Operator (MCO). DGX systems require the following configuration changes:

- Add transparent_hugepage=madvise and iommu=pt settings. The IOMMU setting enables the X2APIC interrupt controller on DGX CPUs
- Clear any enabled ACS flag on all PCIe devices on the DGX

The following instructions create an additional machine pool containing only DGX worker nodes. This step can be omitted if the cluster consists only of DGX systems or if these changes can be applied to all worker nodes in the cluster. It requires the NFD operator that was installed in Installing the NVIDIA GPU Operator to identify DGX worker nodes.

Procedure

Use the following procedure to create the additional machine pool `worker-dgx`.

1. Determine the DGX nodes in the cluster and then then label the nodes as worker-dgx node.:

```
None

$ oc get nodes -l nvidia.com/gpu.machine -o
jsonpath={'.items[].metadata.labels.nvidia\.com/gpu\.machine'} | grep -i dgx

DGXA100-920-23687-2530-000

$ oc label node p42-h03-000-dgx.rdu3.labs.perfscale.redhat.com
node-role.kubernetes.io/worker-dgx=
```

2. Create the machine pool for the DGX nodes. Note that the matchExpression filters for specific DGX nodes:

```
None

$ cat << EOF | oc apply -f -

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

  name: worker-dgx

spec:

  machineConfigSelector:

    matchExpressions:

      - {key: machineconfiguration.openshift.io/role, operator: In, values:
[worker,worker-dgx]}

  nodeSelector:

    matchLabels:

      node-role.kubernetes.io/worker-dgx: ""

EOF
```

3. Validate that the new machine pool has been created and the DGX worker nodes have been added. In this example, two DGX systems were added to the worker-dgx machine pool:

```
None

$ oc get mcp

NAME       ... UPDATED   UPDATING   DEGRADED   MACHINECOUNT   READYMACHINECOUNT
UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT   AGE

master     ... True      False      False      3              3
3                     0                     26d

worker     ... True      False      False      0              0
0                     0                     26d

worker-dgx ... True      False      False      2              2
2                     0                     20d
```

Use the following procedure to modify the kernel configurations across the selected DGX worker nodes in the cluster.

1. Create a machine configuration file for the additional kernel arguments. Replace `worker-dgx` with `worker` if you had chosen to apply these changes to all worker nodes and haven't created the `worker-dgx` machine pool. Note that IOMMU should also be enabled in hardware via BIOS settings as well.

```
None

$ cat << EOF > 99-openshift-machineconfig-dgx-kargs.yaml

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

  name: 99-worker-dgx-kargs

  labels:

    machineconfiguration.openshift.io/role: worker-dgx

spec:

  kernelArguments:

    - transparent_hugepage=madvise

    - iommu=pt
```

```
EOF
```

2. Apply the configuration to the cluster:

```
None
$ oc apply -f 99-openshift-machineconfig-dgx-kargs.yaml
```

3. The worker nodes will now reboot sequentially and start with the new configurations. This can take some time depending on the cluster size. Eventually, all nodes should display the the status `Ready`:

```
None
$ oc get nodes -o wide
NAME       STATUS                     ROLES    AGE    VERSION
worker-0   Ready                      worker   2d1h   v1.22.3+ffbb954
worker-1   Ready,SchedulingDisabled   worker   2d1h   v1.22.3+ffbb954
```

4. Get the status of the worker machine configuration pool. The number in `Ready Machine Count` should reflect the number of worker nodes in the systems when completed:

```
None
$ oc describe mcp worker
  ...
  Degraded Machine Count:     0
  Machine Count:              2
  Observed Generation:        3
  Ready Machine Count:        2
```

```
  Unavailable Machine Count:  0
  Updated Machine Count:      2
```

5. [optional] You can use the following command to validate that the kernel configurations have been updated. Note that this command randomly chooses one of the worker nodes and does not indicate that all nodes have been rebooted:

```
None
$ oc run -it --rm busybox --image=busybox  --restart=Never cat /proc/cmdline
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-80621dd090c35fcb990256491f0f27aab9117d0d67da
a5b351025d1d1186a53f/vmlinuz-4.18.0-305.28.1.el8_4.x86_64 random.trust_cpu=on
console=tty0 console=ttyS0,115200n8 ignition.platform.id=metal
ostree=/ostree/boot.1/rhcos/80621dd090c35fcb990256491f0f27aab9117d0d67daa5b3510
25d1d1186a53f/0 root=UUID=1c197505-c11e-460e-bc65-65ac0bb8f2e7 rw
rootflags=prjquota transparent_hugepage=madvise iommu=pt
```

ACS disable

The following machine configuration installs a script to clear the ACS flag on all PCI devices after boot. Disabling the ACS flag improves PCI-to-PCI communication. The script will also ignore other systems, so it can be safely installed on any cluster. Note that the script needs to be encoded in base64. The original version can be found in Appendix A. The `source:` line must be a single line and the encoded script prefixed with `data:text/plain;charset=utf-8;base64,`.

1. Apply the following configuration to deploy and run the disable-acs script. By default, it is added to the `dgx-worker` machine pool only. Change it to `worker` if you have chosen not to use a separate machine pool but to use the default worker machine pool.

```
None
$ cat << EOF | oc apply -f -
kind: MachineConfig
apiVersion: machineconfiguration.openshift.io/v1
```

```
metadata:
  name: 99-worker-dgx-disable-acs
  creationTimestamp:
  labels:
    machineconfiguration.openshift.io/role: worker-dgx
spec:
  osImageURL: ''
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - filesystem: root
        path: /usr/local/bin/nvidia-disable-acs.sh
        contents:
          source:
data:text/plain;charset=utf-8;base64,IyEvYmluL2Jhc2gKIwojIENvcHlyaWdodCAoYykgMj
AxOCwgTlZJRElBIENPUlBPUkFUSU9OLiAgQWxsIHJpZ2h0cyByZXNlcnZlZC4KIwojIE5WSURJQSBDT
1JQT1JBVElPTiBhbmQgaXRzIGxpY2Vuc29ycyByZXRhaW4gYWxsIGludGVsbGVjdHVhbCBwcm9wZXJ0
eQojIGFuZCBwcm9wcmlldGFyeSByaWdodHMgaW4gYW5kIHRvIHRoaXMgc29mdHdhcmUsIHJlbGF0ZWQ
gZG9jdW1lbnRhdGlvbgojIGFuZCBhbnkgbW9kaWZpY2F0aW9ucyB0aGVyZXRvLiAgQW55IHVzZSwgcm
Vwcm9kdWN0aW9uLCBkaXNjbG9zdXJlIG9yCiMgZGlzdHJpYnV0aW9uIG9mIHRoaXMgc29mdHdhcmUgY
W5kIHJlbGF0ZWQgZG9jdW1lbnRhdGlvbiB3aXRob3V0IGFuIGV4cHJlc3MKIyBsaWNlbnNlIGFncmVl
bWVudCBmcm9tIE5WSURJQSBDT1JQT1JBVElPTiBpcyBzdHJpY3RseSBwcm9oaWJpdGVkLgojCgojIER
pc2FibGUgQUNTIG9uIGV2ZXJ5IGRldmljZSB0aGF0IHN1cHBvcnRzIGl0CgojIG11c3QgYmUgcm9vdC
B0byBhY2Nlc3MgZXh0ZW5kZWQgUENJIGNvbmZpZyBzcGFjZQppZiBbICIkRVVJRCIgLW5lIDAgXTsgd
GhlbgogIGVjaG8gIkVSUk9SOiAkMCBtdXN0IGJlIHJ1biBhcyByb290IgogIGV4aXQgMQpmaQoKZm9y
IEJERiBpbiBgbHNwY2kgLWQgIio6KjoqIiB8IGF3ayAne3ByaW50ICQxfSdgOyBkbwoKICAgICMgc2t
pcCBpZiBpdCBkb2Vzbid0IHN1cHBvcnQgQUNTCiAgICBzZXRwY2kgLXYgLXMgJHtCREZ9IEVDQVBfQU
NTKzB4Ni53ID4gL2Rldi9udWxsIDI+JjEKICAgIGlmIFsgJD8gLW5lIDAgXTsgdGhlbgogICAgICAgI
CNlY2hvICIke0JERn0gZG9lcyBub3Qgc3VwcG9ydCBBQ1MsIHNraXBwaW5nIgogICAgICAgIGNvbnRp
bnVlCiAgICBmaQoKICAgIGxvZ2dlciAiRGlzYWJsaW5nIEFDUyBvbiBgbHNwY2kgLXMgJHtCREZ9YCI
KICAgIHNldHBjaSAtdiAtcyAke0JERn0gRUNBUF9BQ1MrMHg2Lnc9MDAwMAogICAgaWYgWyAkPyAtbm
UgMCBdOyB0aGVuCiAgICAgICAgbG9nZ2VyICJFcnJvciBkaXNhYmxpbmcgQUNTIG9uICR7QkRGfSIKI
CAgICAgICBjb250aW51ZQogICAgZmkKICAgIE5FV19WQUw9YHNldHBjaSAtdiAtcyAke0JERn0gRUNB
UF9BQ1MrMHg2LncgfCBhd2sgJ3twcmludCAkTkZ9J2AKICAgIG9uICE9IiR7TkVXX1ZBTH0iICE9ICI
wMDAwIiBdOyB0aGVuCiAgICAgICAgbG9nZ2VyICJGYWlsZWQgdG8gZGlzYWJsZSBBQ1Mgb24gJHtCRE
Z9IgogICAgICAgIGNvbnRpbnVlCiAgICBmaQpkb25lCmV4aXQgMAo=
```

```
          verification: {}
        mode: 0755
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Disable PCIe ACS capability
            DefaultDependencies=no
            After=sysinit.target local-fs.target
            Before=basic.target
            [Service]
            Type=oneshot
            ExecStart=/bin/sh -c '/usr/local/bin/nvidia-disable-acs.sh'
            [Install]
            WantedBy=basic.target
          name: nvidia-acs-disable.service
          enabled: true

EOF
```

2. The nodes reboot after applying the changes. This can take some time depending on the cluster size. You can monitor the progress using `oc get nodes --watch` to list the status of all nodes, and `oc get mcp` to display the status of the machine configuration pool. The nodes should all be in the `Ready` state, but will cycle through states, such as `Ready`,`SchedulingDisabled`, `NotReady` or `SchedulingDisabled`.

None

```
$ oc get nodes --watch
NAME       STATUS                     ROLES    AGE   VERSION
master-0   Ready                      master   27d   v1.22.3+ffbb954
```

```
master-1   Ready                      master   27d   v1.22.3+ffbb954
master-2   Ready                      master   27d   v1.22.3+ffbb954
worker-0   Ready,SchedulingDisabled   worker   27d   v1.22.3+ffbb954
worker-1   Ready                      worker   27d   v1.22.3+ffbb954
```

3. Get the status of the worker machine configuration pool. The number in `Ready Machine Count` should reflect the number of worker nodes in the systems when completed:

```
None
$ oc describe mcp worker
  ...
  Degraded Machine Count:     0
  Machine Count:              2
  Observed Generation:        3
  Ready Machine Count:        2
  Unavailable Machine Count:  0
  Updated Machine Count:      2
```

Troubleshooting

- Use the following command to display status information about the Machine Pool:

```
None
$ oc describe mcp worker-dgx
```

Example Output

```
None
Name:         worker-dgx
Namespace:
Labels:       <none>
...
Status:
  Conditions:
    Last Transition Time:  2022-01-04T19:33:30Z
    Message:
    Reason:
    Status:                False
    Type:                  Updated
    Last Transition Time:  2022-01-04T19:33:30Z
    Message:               All nodes are updating to
rendered-worker-dgx-fccf428c253ca52bbcf0c8573a172ee7
...
```

Installing NVSM

NVIDIA System Management (NVSM) is a software framework for monitoring NVIDIA DGX nodes in a data center. It includes active health monitoring, system alerts, and log generation, and also supports a stand-alone mode from the command line to get a quick health report of the system. Running NVSM is typically requested by the NVIDIA Enterprise Support team to resolve a reported problem.

NVSM can be deployed on the DGX nodes with the NVIDIA System Management NGC Container. It allows users to execute the NVSM tool remotely and on-demand in the containers that are deployed to the DGX nodes.

The installation uses the NVSM Helm Chart to create the necessary resources on the cluster. The deployment is limited to systems that are manually labeled with `nvidia.com/gpu.nvsm.deploy=true`.

Procedure

Use the following procedure to deploy NVSM on the DGX worker nodes.

1. [Optional] Use the following command to get a list of all DGX nodes in the cluster:

```
None
$ oc get nodes --show-labels|grep nvidia.com/gpu.machine=.*DGX[^,]*
```

2. Set the flag `nvidia.com/gpu.nvsm.deploy=true` on the DGX worker nodes on which you want to deploy NVSM (replace `WORKER1` etc. with the actual name of the nodes):

```
None
$ oc label node/WORKER1 nvidia.com/gpu.nvsm.deploy=true
$ oc label node/WORKER2 nvidia.com/gpu.nvsm.deploy=true
...
```

3. Get the Helm chart for deploying NVSM on the cluster:

```
None
$ helm fetch
https://helm.ngc.nvidia.com/nvidia/cloud-native/charts/nvsm-1.0.1.tgz
--username='$oauthtoken' --password=<NGC_API_KEY>"helm fetch
```

1. Ensure the file is in your local directory:

```
None
$ ls ./nvsm-1.0.1.tgz
```

2. To ensure the default settings are correct for your installation, inspect the contents of `values.yaml` in the above `tar` file. If the default settings are not correct, update the file as per the cluster configuration.

3. Deploy NVSM to the cluster. The cluster installs the container on the nodes that have been labeled in the previous steps. The following command creates the `nvidia-nvsm` namespace and deploys the resource in the namespace.

```
None

$ helm install --set platform.openshift=true --create-namespace -n nvidia-nvsm
nvidia-nvsm ./nvsm-1.0.1.tgz
```

4. Validate that NVSM has been deployed on all selected DGX nodes. You should see an nvidia-nvsm-XXXX pod instance for each node.

```
None

$ oc get pods -n nvidia-nvsm -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
...
nvidia-nvsm-d9d9t   1/1     Running   1          8h    10.128.2.11  worker-0
...
nvidia-nvsm-tt8g5   1/1     Running   1          8h    10.131.0.11  worker-1
...
```

5. Patching the nvidia-nvsm daemonset

```
None

$ oc patch ds nvidia-nvsm -n nvidia-nvsm --type=json -p='[{"op": "remove",
"path": "/spec/template/spec/containers/0/command"},{"op": "remove", "path":
"/spec/template/spec/containers/0/env"}]'
daemonset.apps/nvidia-nvsm patched

$ oc patch ds nvidia-nvsm -n nvidia-nvsm --type=json -p='[{"op": "add", "path":
"/spec/template/spec/containers/0/env", value: [{"name": "mode", "value":
"nvsm-start"},{"name":"NVSM_DCGM_HOSTENGINE_INFO","value":"nvidia-dcgm.nvidia-g
```

```
pu-operator.svc.cluster.local"}]}]'
daemonset.apps/nvidia-nvsm patched
```

Using NVSM

For a maintenance task or to complete a health analysis, you can now run NVSM remotely inside the deployed containers on one of the DGX worker nodes.

Procedure

The following procedure describes the general NVSM workflow.

1. List all NVSM pod instances and the corresponding worker nodes to find the pod name associated with a specific DGX:

```
None
$ oc get pods -n nvidia-nvsm -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
...
nvidia-nvsm-d9d9t   1/1     Running   1          8h    10.128.2.11  worker-0
...
nvidia-nvsm-tt8g5   1/1     Running   1          8h    10.131.0.11  worker-1
...
```

2. Use the `oc exec` command to start an interactive shell in the container that is running on that system:

```
None
$ oc rsh -n nvidia-nvsm <pod-name>
```

3. You can now use one of the main NVSM commands.

Note: When you execute NVSM, it can take a couple of minutes to collect system information.

- To print the software and firmware versions of the DGX system:

```
None
$ nvsm show version
```

- To provide a summary of the system health:

```
None
$ nvsm show health
```

- To run a stress test - this will run stress test across the GPUs, CPUs, Memory and storage for a duration of 30 seconds.

```
None
$ nvsm stress-test --force 30
```

- To create a snapshot of the system components for offline analysis and diagnosis. This command generates a tar-file in the /tmp directory. (see Retrieving Health Information for more information).

```
None
$ nvsm dump health
```

4. Exit the interactive shell:

```
None


$ exit
```

Retrieving Health Information

This section describes the steps to generate and retrieve health information to debug a system issue offline or when requested by the NVIDIA Enterprise Support organization.

Procedure

1. List all NVSM instances and corresponding worker nodes to find the pod name that is associated with a DGX system:

```
None
$ oc get pods -n nvidia-nvsm -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
...
nvidia-nvsm-d9d9t   1/1     Running   1          8h    10.128.2.11  worker-0
...
nvidia-nvsm-tt8g5   1/1     Running   1          8h    10.131.0.11  worker-1
...
```

2. Start an interactive shell in the NVSM pod of the corresponding DGX worker node. `<pod-name>` is the name of the pod from the list in Step 1.

```
None
$ oc rsh -n nvidia-nvsm <pod-name> bash
```

3. Create the NVSM snapshot file of all system components for offline analysis and diagnostics. The file is created under the `/tmp` directory inside the container.

```
None
$ nvsm dump health
```

Example Output

```
None


$ nvsm dump health


Health dump started


        This command will collect system configuration and diagnostic
information to help diagnose the system.

        The output may contain data considered sensitive and should be reviewed
before sending to any third party.


Collecting 100% |████████████████████████████████████████|


The output of this command is written to:
/tmp/nvsm-health-nvidia-nvsm-x2rhr-20250610172848.tar.xz
```

4. Exit the container:

```
None

$ exit
```

5. Copy the snapshot file from the container to the local host for additional analysis or to send it to NVIDIA Enterprise Support:

```
None

$ oc cp nvidia-nvsm/<pod-name>:tmp/<snapshot-file> <target-file>
```

Example

```
None

$ oc cp
nvidia-nvsm/nvidia-nvsm-x2rhr:/tmp/nvsm-health-nvidia-nvsm-x2rhr-20250610172848
.tar.xz 20250610172848.tar.xz
```

Note: `<snapshot-file>` describes the name of the generated file from Step 3, and `<target-file>` refers to the name for the file on the local host. Use the same file names.

6. Delete the generated snapshot file in the NVSM pod:

```
None

$ oc rsh -n nvidia-nvsm <pod-name> rm /tmp/<snapshot-file>
```

The generated file can now be used to debug and analyze system issues, or you can send the file to NVIDIA Enterprise Support.

NOTE:

The following sections are provided on a per-need basis. We will do more qualifications.

Advanced Installation Workflows

Advanced installation workflows are broken down into the following subsections to cover more advanced use cases around OpenShift on the DGX system.

1. Applying System Optimizations
2. Installing in Disconnected Air-Gapped Environments
3. Additional RDMA Deployment Configurations

Installing in Air-Gapped Environments

You can install an OpenShift Container Platform cluster in a disconnected environment using the Agent-based Installer. You must perform additional setups and configurations for your cluster to maintain full functionality in the disconnected environment. Also note that all operator images and dependency packages for any operators consumed in an air-gapped environment must also be mirrored.

To learn more about installing a cluster in a disconnected environment with the Agent-based installer, see the following:

- Installing a cluster in a disconnected environment
- Installing an OpenShift Container Platform cluster with the Agent-based Installer

Upgrade DGX and OpenShift

OpenShift Upgrade

Upgrading OpenShift ensures your cluster runs on the latest supported version, enhancing features, security, and performance. The process involves updating cluster components, Operators, and nodes, using the web console or the OpenShift CLI (oc).

For detailed guidance on upgrading OpenShift on bare-metal deployments, you can refer to Red Hat's official documentation.

General Guidelines for Firmware Upgrade

1. Distribute the upgrade package
2. Drain nodes, upgrade, and verify healthcheck
3. Remove the drain and verify healthcheck
4. Check versions

DGX Firmware Upgrade

This section describes how to upgrade the DGX firmware.

DGX H100 / H200 / B200 Firmware upgrade

Procedure

1. The following commands use the `oc` command (OpenShift CLI) to drain and unschedule the node from any workload. Alternatively, you can use your preferred oc authentication method.

```
None
$ oc get nodes # Will print the list of nodes and their names.
$ oc adm drain <NODE NAME> --delete-emptydir-data --ignore-daemonsets
```

2. Download the update package from Nvidia's enterprise support site at: https://enterprise-support.nvidia.com/s/downloads

Note: Step 2 requires a login to the NVIDIA Enterprise Support site.

3. Ensure you download and unzip the most recent firmware file that corresponds to your DGX model (DGXH100 firmware version 1.1.3 is an example.):

```
None
$ tar -xvf DGXH100_1.1.3.tar
```

4. List the installed firmware using BMC Authentication:

```
None
$ ./nvfwupd --target ip=<BMC Address> user=<BMC USER> password=<BMC PASSWORD>
show_version -p ./nvfw_DGXH100_240105.1.0.fwpkg
./nvfw_HGX_DGXH100_231101.1.0.fwpkg
```

5. Follow the official documentation at:

a. DGX H100/200: https://docs.nvidia.com/dgx/dgxh100-fw-update-guide/sequence.html

b. DGX B200 https://docs.nvidia.com/dgx/dgxb200-fw-update-guide/nvfwupd-reference.html

Upgrade all of the components as documented

6. After all of the components have been upgraded, you can resume operations on this node:

```
None
$ oc adm uncordon <NODE_NAME>
```

Note: Make sure to uncordon and verify the node is ready and schedulable before proceeding to the next node.

Intel NIC firmware upgrade

Procedure

1. Download the latest update package from Intel: Non-Volatile Memory (NVM) Update Utility for Intel® Ethernet Network Adapter 700 Series.

2. Use the following commands to extract the package:

```
None
$ unzip 700Series_NVMUpdatePackage_v9_52_.zip
Archive:  700Series_NVMUpdatePackage_v9_52_.zip
  inflating: 700Series_NVMUpdatePackage_v9_52_EFI.zip
  inflating: 700Series_NVMUpdatePackage_v9_52_ESX.tar.gz
  inflating: 700Series_NVMUpdatePackage_v9_52_FreeBSD.tar.gz
  inflating: 700Series_NVMUpdatePackage_v9_52_Linux.tar.gz
 extracting: 700Series_NVMUpdatePackage_v9_52_Windows.zip
```

```
$ tar -xf 700Series_NVMUpdatePackage_v9_52_Linux.tar.gz
```

3. Create the `Containerfile`:

```
cat > Containerfile << EOF
# Start from UBI9 image
FROM registry.access.redhat.com/ubi9/ubi:9.5-1730489303

# Set work directory

RUN dnf install procps-ng pciutils iputils ethtool net-tools -y && dnf
clean all

COPY 700Series/Linux_x64 /intel_700series
EOF
```

4. Build the container image and push it into a private registry:

```
None
$ export TAG=$(date +%Y%m%d_%H%M%S)
$ podman build -t intelnicfw:$TAG -f Containerfile
$ podman tag intelnicfw:$TAG <REGISTRY_ADDRESS>/intelnicfw:$TAG
$ podman push <REGISTRY_ADDRESS>/intelnicfw:$TAG
```

5. The following commands use `oc` (OpenShift CLI) to drain and unschedule the node from any workload. Alternatively, you can use your preferred oc authentication method.

```
None
$ oc get nodes # Will print the list of nodes and their names.
$ oc adm drain <NODE NAME> --delete-emptydir-data --ignore-daemonsets
```

6. Shell into the node using `oc debug`:

```
None
$ oc debug node/<NODE NAME>
# chroot /host
sh-5.1#:/#
```

7. Execute the update container:

```
None
sh-5.1#:/# podman run -it --privileged <REGISTRY_ADDRESS>/intelnicfw:<VERSION>
/bin/bash
[root@d74f7486ed25 /]# cd /intel_700series/
[root@d74f7486ed25 /]# ./nvmupdate64e
```

8. The `nvmupdate64e` tool will upgrade the firmware on those NIC adapters, when it's done we should now restore our workloads on this node.

```
None
[root@d74f7486ed25 /]# exit
sh-5.1# exit
sh-4.4# exit
$ oc adm uncordon <NODE NAME>
```

Note: Make sure to uncordon and verify the node is ready and schedulable before proceeding to the next node.

For more details, refer to Updating the Intel NIC Firmware.

NVMe firmware upgrade

Procedure

1. The following commands use `oc` (OpenShift CLI) to drain and unschedule the node from any workload. Alternatively, you can use your preferred oc authentication method.

```
None
$ oc get nodes # Will print the list of nodes and their names.
$ oc adm cordon <NODE NAME>
$ oc adm drain <NODE NAME> --delete-emptydir-data --ignore-daemonsets
```

2. Shell into the node using `oc debug`:

```
None
$ oc debug node/<NODE NAME>
# chroot /host
sh-5.1#:/#
```

3. List the NVMe devices.

   Note: In some cases, you might have to manually search for the firmware release on your site to find an update to your NVMe device.

```
None
sh-5.1# nvme list
Node                  Generic               SN                   Model
Namespace Usage                      Format           FW Rev
--------------------- --------------------- --------------------
---------------------------------------- --------- --------------------------
---------------- --------
/dev/nvme9n1          /dev/ng9n1            72M0A0XXXXXX         KCM6DRUL3T84
1           1.63  TB /   3.84  TB    512   B +  0 B   0107
/dev/nvme8n1          /dev/ng8n1            72L0A0XXXXXX         KCM6DRUL3T84
1          20.26  GB /   3.84  TB    512   B +  0 B   0107
...
/dev/nvme1n1          /dev/ng1n1            S666NS0TXXXXXX       SAMSUNG
MZ1L21T9HCLS-00A07               1           1.92  TB /   1.92  TB    512   B +
0 B   GDC7302Q
...
```

4. Download the firmware package from the NVIDIA Enterprise support downloads page of the current firmware release, extract, and `scp` it to the target node.

```
None
$ unzip CM6-SED-0107.zip
$ scp CM6-SED-0107.std core@<NODE NAME>:~
```

5. Shell into the node and escalate privilege:

```
None
$ ssh core@<NODE Address>
[core@node:/]$ sudo -s
[root@node:/]$
```

6. Flash the corresponding NVME firmware blob to your selected devices:

```
None
[root@node:/]# nvme fw-download /dev/nvme8n1 --fw=/home/core/CM6-SED-0107.std
Firmware download success
```

7. Commit and activate the new firmware:

```
None
[root@node:/]# nvme fw-commit /dev/nvme8n1 --action=3
Success committing firmware action:3 slot:0
Multiple Update Detected (MUD) Value: 0
```

8. Verify the utilization of the newly flashed firmware:

```
None
[root@node:/]# sh-5.1# nvme id-ctrl /dev/nvme8n1
NVME Identify Controller:
vid       : 0x1e0f
ssvid     : 0x1e0f
sn        : 72L0XXXXXXXX
mn        : KCM6DRUL3T84
fr        : 0107
...
```

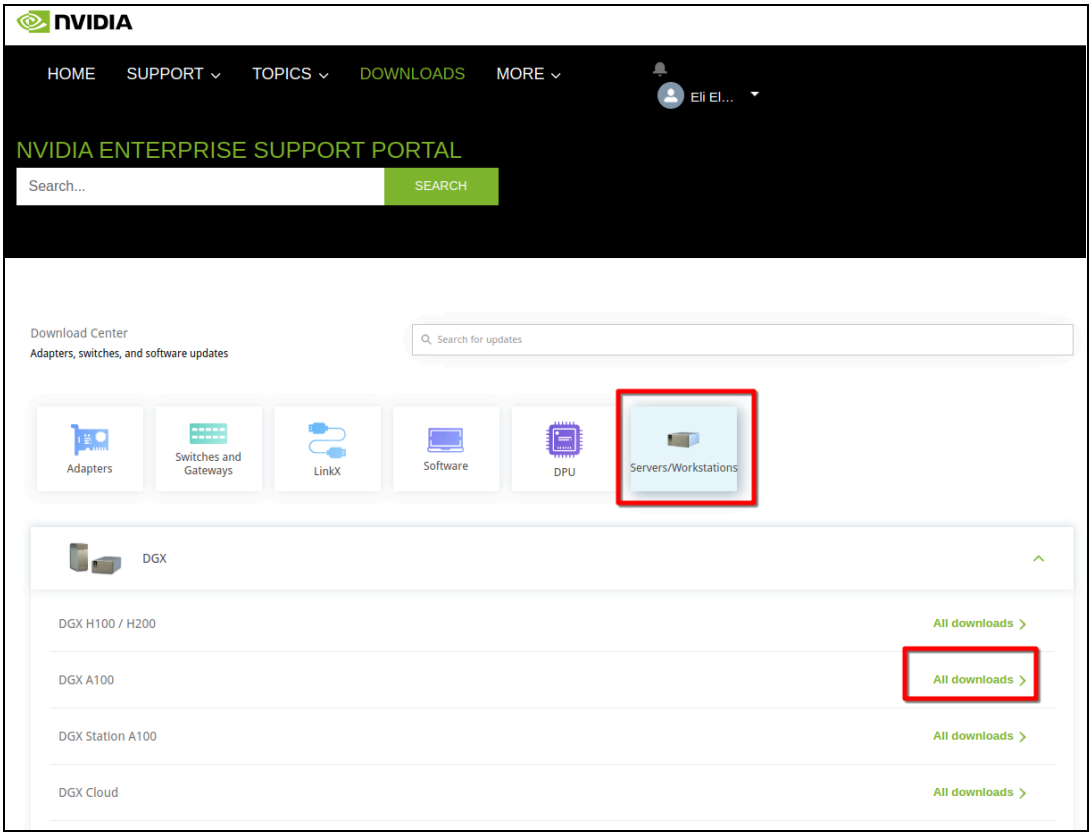
Note: Make sure to uncordon and verify the node is ready and schedulable before proceeding to the next node.

Follow the official instructions at Updating the NVMe Firmware.

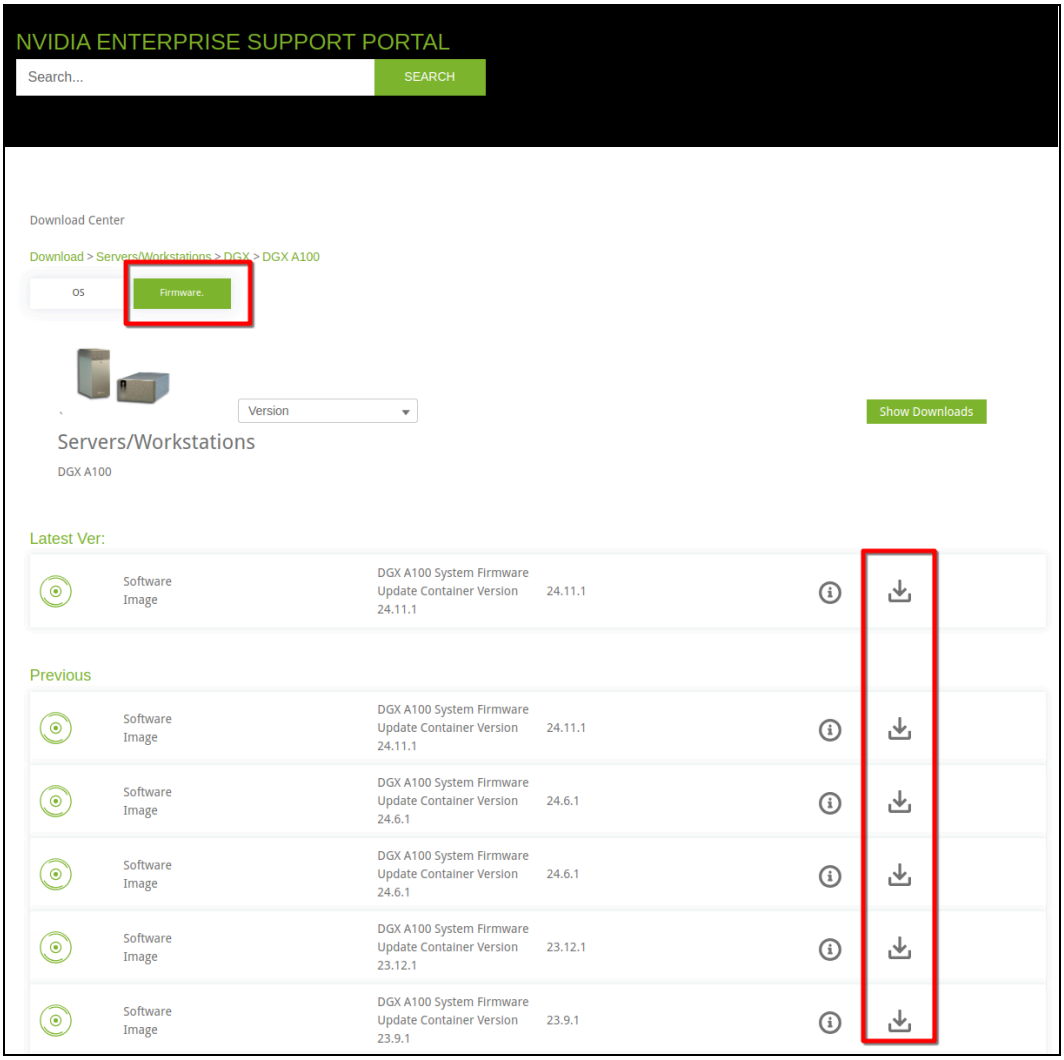
DGX A100 Firmware upgrade

Procedure

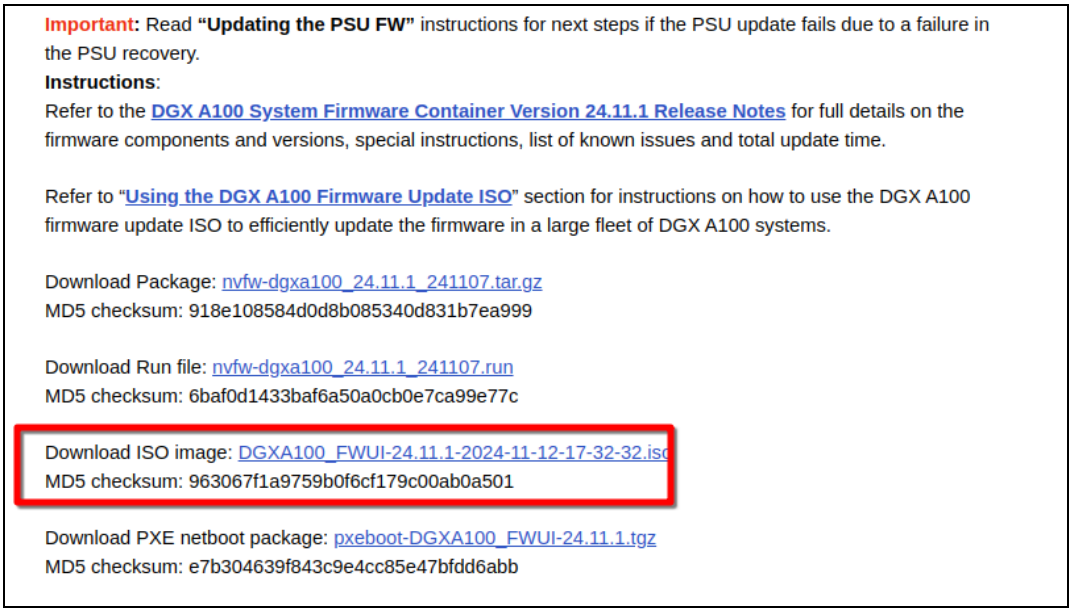
1. Download the Firmware update ISO. For more information, refer to https://enterprise-support.nvidia.com/s/downloader.



2. Click **Firmware**, then select the firmware suite:



3. Click **Download ISO image**:



4. Drain the target node:

```
None
$ oc get nodes # Will print the list of nodes and their names.
$ oc adm drain <NODE NAME> --delete-emptydir-data --ignore-daemonsets --force
```

5. Boot the ISO with your preferred method (BMC ISO mount, USB, PXE, etc.) and follow the official instructions at Using the DGX A100 Firmware Update ISO.

6. After firmware update, reboot into Red Hat Openshift and resume operations for this node:

```
None
$ oc adm uncordon <NODE NAME>
```

Note: Make sure to uncordon and verify the node is ready and schedulable before proceeding to the next node.

Mellanox Adapters Firmware Update

Procedure

1. Remove the workload from the target node:

```
None
$ oc get nodes # Will print the list of nodes and their names.
$ export TARGET_NODE=<NODE_NAME>
$ oc adm drain <NODE NAME> --delete-emptydir-data --ignore-daemonsets
```

Note: Only use `--force` if the `oc adm drain` command does not drain the node.

2. Enter prompt shell in the SRIOV container:

```
None
$ export SRIOV_CONTAINER=$(oc get pods -n openshift-sriov-network-operator
--field-selector spec.nodeName=$TARGET_NODE
--output=custom-columns=NAME:.metadata.name | grep -E 'sriov-network-config')
$ oc rsh -n openshift-sriov-network-operator $SRIOV_CONTAINER
```

3. List all Mellanox devices:

```
None
sh-5.1# lspci | grep Mellanox
```

Example Output

```
None

...
8d:00.0 Infiniband controller: Mellanox Technologies MT28908 Family
[ConnectX-6]
...
```

4. Query the current installed firmware version and the PSID (used to select the update file):

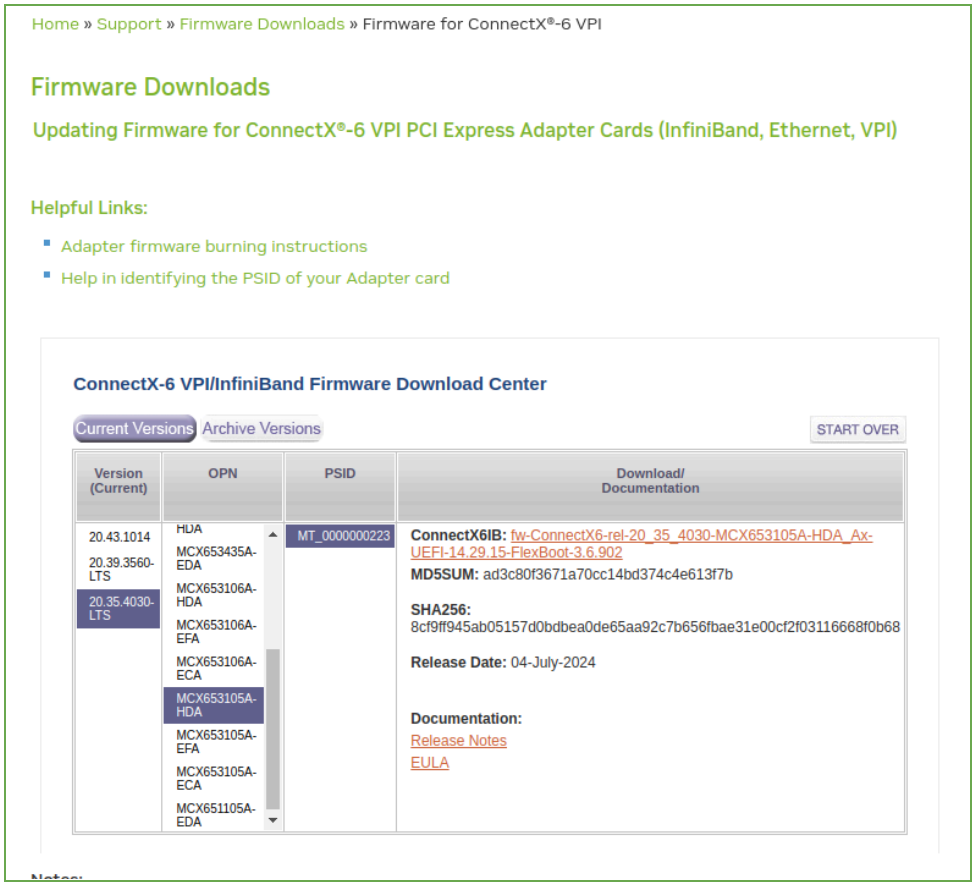
```
None

sh-5.1# mstflint -d 8d:00.0 query
Image type:            FS4
FW Version:            20.35.4030
FW Release Date:       27.6.2024
Product Version:       20.35.4030
Rom Info:              type=UEFI version=14.29.15 cpu=AMD64,AARCH64
                       type=PXE version=3.6.902 cpu=AMD64
Description:           UID                GuidsNumber
Base GUID:             0c42a103000c001c       4
Base MAC:              0c42a10c001c           4
Image VSD:             N/A
Device VSD:            N/A
PSID:                  MT_0000000223
Security Attributes:   N/A
```

5. To select a compatible firmware update you will need the part number or OPN. This example uses ConnectX6, P/N: MCX653105A-HDAT with the PSID: MT_0000000223 from the previous output.

None

```
sh-5.1# lspci -vv -s 8d:00.0 | grep "Part number" -A 3
                        [PN] Part number: MCX653105A-HDAT
                        [EC] Engineering changes: A8
                        [V2] Vendor specific: MCX653105A-HDAT
                        [SN] Serial number: MT2004X08955
```

6. Go to the Mellanox firmware downloads at https://network.nvidia.com/support/firmware/firmware-downloads/. You can view the available versions:



If the part number isn't listed, use the PSID to find the correct firmware.

7. Download the firmware zipped blob, unzip it, and copy it to the sr-iov container:

```
None
$ unzip
fw-ConnectX6-rel-20_39_3560-MCX653105A-HDA_Ax-UEFI-14.32.17-FlexBoot-3.7.300.bi
n.zip
$ oc cp
fw-ConnectX6-rel-20_39_3560-MCX653105A-HDA_Ax-UEFI-14.32.17-FlexBoot-3.7.300.bi
n openshift-sriov-network-operator/"$SRIOV_CONTAINER":/root
```

8. Login to the SR-IOV container and flash the firmware binary blob on the selected device:

```
None
$ oc rsh -n openshift-sriov-network-operator $SRIOV_CONTAINER
sh-5.1# mstflint -d 8d:00.0 -i
/root/fw-ConnectX6-rel-20_39_3560-MCX653105A-HDA_Ax-UEFI-14.32.17-FlexBoot-3.7.
300.bin burn

    Current FW version on flash:  20.35.4030
    New FW version:               20.39.3560

-E- Burning FS4 image failed: The firmware image was already updated on flash,
pending reset.
sh-5.1# mstfwreset -d 8d:00.0 reset

Minimal reset level for device, 8d:00.0:

0: Driver, PCI link, network link will remain up ("live-Patch")
Continue with reset?[y/N] y
-I- Sending Reset Command To Fw             -Done
-I- FW was loaded successfully.
sh-5.1# mstflint -d 8d:00.0 -i
/root/fw-ConnectX6-rel-20_39_3560-MCX653105A-HDA_Ax-UEFI-14.32.17-FlexBoot-3.7.
300.bin burn

    Current FW version on flash:  20.35.4030
    New FW version:               20.39.3560

FSMST_INITIALIZE -   OK
Writing Boot image component -   OK
Restoring signature                    - OK
-I- To load new FW run mstfwreset or reboot machine.
```

9. Verify the firmware flash was successful by querying the device:

```
None
sh-5.1# mstflint -d 8d:00.0 query
Image type:            FS4
FW Version:            20.39.3560
FW Version(Running):   20.35.4030
FW Release Date:       24.6.2024
Product Version:       20.35.4030
Rom Info:              type=UEFI version=14.29.15 cpu=AMD64,AARCH64
                       type=PXE version=3.6.902 cpu=AMD64
Description:           UID                GuidsNumber
Base GUID:             0c42a103000c001c       4
Base MAC:              0c42a10c001c           4
Image VSD:             N/A
Device VSD:            N/A
PSID:                  MT_0000000223
Security Attributes:   N/A
```

10. As you can see, the firmware was flashed into the adapter, but it is still running the previous version. Reboot the node, or optionally, attempt to restart the firmware using `mstfwreset.`

    Note: Make sure to uncordon and verify the node is ready and schedulable before proceeding to the next node if node was cordoned.

Appendix A – Disable PCIe ACS

Systems such as the DGX may have PCIe devices booted with the ACS flag enabled. This deteriorates performance on such systems for PCI to PCI communication. The ACS flag should, therefore, be cleared on reboot. The following script disables any enabled ACS flags on the DGX A100 and should be started as part of a `systemd` service.

- To generate the base64 encoding, use the following command followed by the actual script and EOF at the end.

```
None
$ cat << 'EOF' > disable-acs.sh
> #!/bin/bash
#
# Copyright (c) 2018, NVIDIA CORPORATION.  All rights reserved.
#
# NVIDIA CORPORATION and its licensors retain all intellectual property
# and proprietary rights in and to this software, related documentation
# and any modifications thereto.  Any use, reproduction, disclosure or
# distribution of this software and related documentation without an express
# license agreement from NVIDIA CORPORATION is strictly prohibited.
#

# Disable ACS on every device that supports it

# must be root to access extended PCI config space
if [ "$EUID" -ne 0 ]; then
  echo "ERROR: $0 must be run as root"
  exit 1
fi

for BDF in `lspci -d "*:*:*" | awk '{print $1}'`; do
```

```
        # skip if it doesn't support ACS
        setpci -v -s ${BDF} ECAP_ACS+0x6.w > /dev/null 2>&1
        if [ $? -ne 0 ]; then
        #echo "${BDF} does not support ACS, skipping"
        continue
        fi

        logger "Disabling ACS on `lspci -s ${BDF}`"
        setpci -v -s ${BDF} ECAP_ACS+0x6.w=0000
        if [ $? -ne 0 ]; then
        logger "Error disabling ACS on ${BDF}"
        continue
        fi
        NEW_VAL=`setpci -v -s ${BDF} ECAP_ACS+0x6.w | awk '{print $NF}'`
        if [ "${NEW_VAL}" != "0000" ]; then
        logger "Failed to disable ACS on ${BDF}"
        continue
        fi
done
exit 0
EOF

$ cat disable-acs.sh | base64 -w0
IyEvYmluL2Jhc2gKIwojIENvcHlyaWdodCAoYykgMjAxOCwgTlZJRElBIENPUlBPUkFUSU9OLiAgQWx
sIHJpZ2h0cyByZXNlcnZlZC4KIwojIE5WSURJQSBDT1JQT1JBVElPTiBhbmQgaXRzIGxpY2Vuc29ycy
ByZXRhaW4gYWxsIGludGVsbGVjdHVhbCBwcm9wZXJ0eQojIGFuZCBwcm9wcmlldGFyeSByaWdodHMga
W4gYW5kIHRvIHRoaXMgc29mdHdhcmUsIHJlbGF0ZWQgZG9jdW1lbnRhdGlvbgojIGFuZCBhbnkgbW9k
aWZpY2F0aW9ucyB0aGVyZXRvLiAgQW55IHVzZSwgcmVwcm9kdWN0aW9uLCBkaXNjbG9zdXJlIG9yCiM
gZGlzdHJpYnV0aW9uIG9mIHRoaXMgc29mdHdhcmUgYW5kIHJlbGF0ZWQgZG9jdW1lbnRhdGlvbiB3aX
Rob3V0IGFuIGV4cHJlc3MKIyBsaWNlbnNlIGFncmVlbWVudCBmcm9tIE5WSURJQSBDT1JQT1JBVElPT
iBpcyBzdHJpY3RseSBwcm9oaWJpdGVkLgojCgojIERpc2FibGUgQUNTIG9uIGV2ZXJ5IGRldmljZSB0
aGF0IHN1cHBvcnRzIGl0CgojIG11c3QgYmUgcm9vdCB0byBhY2Nlc3MgZXh0ZW5kZWQgUENJIGNvbmZ
pZyBzcGFjZQppZiBbICIkRVVJRCIgLW5lIDAgXTsgdGhlbgogIGVjaG8gIkVSUk9SOiAkMCBtdXN0IG
```

```
JlIHJ1biBhcyByb290IgogIGV4aXQgMQpmaQoKZm9yIEJERiBpbiBgbHNwY2kgLWQgIio6KjoqIiB8I
GF3ayAne3ByaW50ICQxfSdgOyBkbwoKICAgICMgc2tpcCBpZiBpdCBkb2Vzbid0IHN1cHBvcnQgQUNT
CiAgICBzZXRwY2kgLXYgLXMgJHtCREZ9IEVDQVBfQUNTKzB4Ni53ID4gL2Rldi9udWxsIDI+JjEKICA
gIGlmIFsgJD8gLW5lIDAgXTsgdGhlbgogICAgICAgICNlY2hvICIke0JERn0gZG9lcyBub3Qgc3VwcG
9ydCBBQ1MsIHNraXBwaW5nIgogICAgICAgIGNvbnRpbnVlCiAgICBmaQoKICAgIGxvZ2dlciAiRGlzY
WJsaW5nIEFDUyBvbiBgbHNwY2kgLXMgJHtCREZ9YCIKICAgIHNldHBjaSAtdiAtcyAke0JERn0gRUNB
UF9BQ1MrMHg2Lnc9MDAwMAogICAgaWYgWyAkPyAtbmUgMCBdOyB0aGVuCiAgICAgICAgbG9nZ2VyICJ
FcnJvciBkaXNhYmxpbmcgQUNTIG9uICR7QkRGfSIKICAgICAgICBjb250aW51ZQogICAgZmkKICAgIE
5FV19WQUw9YHNldHBjaSAtdiAtcyAke0JERn0gRUNBUF9BQ1MrMHg2LncgfCBhd2sgJ3twcmludCAkT
kZ9J2AKICAgIGlmIFsgIiR7TkVXX1ZBTH0iICE9ICIwMDAwIiBdOyB0aGVuCiAgICAgICAgbG9nZ2Vy
ICJGYWlsZWQgdG8gZGlzYWJsZSBBQ1Mgb24gJHtCREZ9IgogICAgICAgIGNvbnRpbnVlCiAgICBmaQp
kb25lCmV4aXQgMAonRU9GJwo=
```

Appendix B – Switching Mellanox ConnectX 6 Modes (Ethernet or Infiniband)

Procedure

1. Enter prompt shell in the SRIOV container:

```
None
$ export SRIOV_CONTAINER=$(oc get pods -n openshift-sriov-network-operator
--field-selector spec.nodeName=<NODE_NAME>
--output=custom-columns=NAME:.metadata.name | grep -E 'sriov-network-config')
$ oc rsh -n openshift-sriov-network-operator $SRIOV_CONTAINER
```

2. List the Mellanox devices:

```
None
sh-5.1# lspci | grep Mellanox
```

3. Copy the PCI Address of the device on which you want to switch its mode and use it with the following `mstconfig` instruction. In this example, the Infiniband mode to Ethernet mode is switched on this single port adapter. A value of 1 means Infiniband, A value of 2 means Ethernet.

```
None
sh-5.1# mstconfig -d cc:00.0 set LINK_TYPE_P1=2

Device #1:
----------

Device type:    ConnectX6
Name:           MCX653105A-HDA_Ax
```

```
Description:    ConnectX-6 VPI adapter card; HDR IB (200Gb/s) and 200GbE;
single-port QSFP56; PCIe4.0 x16; tall bracket; ROHS R6
Device:         cc:00.0

Configurations:                                      Next Boot       New
       LINK_TYPE_P1                                 IB(1)           ETH(2)

Apply new Configuration? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

4. Use `mstfwreset` to restart the firmware on the adapter:

```
None
sh-5.1# mstfwreset -d cc:00.0 reset

Minimal reset level for device, cc:00.0:


3: Driver restart and PCI reset

Continue with reset?[y/N] y

-I- Sending Reset Command To Fw             -Done

-I- Stopping Driver                         -Done

-I- Resetting PCI                           -Done

-I- Starting Driver                         -Done

-I- FW was loaded successfully.
```

Appendix C - Example YAML files

This appendix contains the YAML examples created in the installation procedures.

network-sharedrdma-nic-cluster-policy.yaml

```
None
$ cat <<EOF > network-sharedrdma-nic-cluster-policy.yaml
apiVersion: mellanox.com/v1alpha1
kind: NicClusterPolicy
metadata:
  name: nic-cluster-policy
spec:
  nicFeatureDiscovery:
    image: nic-feature-discovery
    repository: ghcr.io/mellanox
    version: v0.0.1
  docaTelemetryService:
    image: doca_telemetry
    repository: nvcr.io/nvidia/doca
    version: 1.16.5-doca2.6.0-host
  rdmaSharedDevicePlugin:
    config: |
      {
        "configList": [
          {
            "resourceName": "rdma_shared_device_ib",
            "rdmaHcaMax": 63,
            "selectors": {
              "ifNames": ["ibs2f0"]
```

```
          }
        },
        {
          "resourceName": "rdma_shared_device_eth",
          "rdmaHcaMax": 63,
          "selectors": {
            "ifNames": ["ens8f0np0"]
          }
        }
      ]
    }
  image: k8s-rdma-shared-dev-plugin
  repository: ghcr.io/mellanox
  version: v1.5.1
secondaryNetwork:
  ipoib:
    image: ipoib-cni
    repository: ghcr.io/mellanox
    version: v1.2.0
nvIpam:
  enableWebhook: false
  image: nvidia-k8s-ipam
  repository: ghcr.io/mellanox
  version: v0.2.0
ofedDriver:
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
  forcePrecompiled: false
  terminationGracePeriodSeconds: 300
  livenessProbe:
```

```
      initialDelaySeconds: 30
      periodSeconds: 30
    upgradePolicy:
      autoUpgrade: true
      drain:
        deleteEmptyDir: true
        enable: true
        force: true
        timeoutSeconds: 300
        podSelector: ''
      maxParallelUpgrades: 1
      safeLoad: false
      waitForCompletion:
        timeoutSeconds: 0
    startupProbe:
      initialDelaySeconds: 10
      periodSeconds: 20
    image: doca-driver
    repository: nvcr.io/nvidia/mellanox
    version: 25.01-0.6.0.0-0
    env:
    - name: UNLOAD_STORAGE_MODULES
      value: "true"
    - name: RESTORE_DRIVER_ON_POD_TERMINATION
      value: "true"
    - name: CREATE_IFNAMES_UDEV
      value: "true"
EOF
```

machine-config-blacklist-irdma.yaml

```
None
cat <<EOF > 99-machine-config-blacklist-irdma.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-blacklist-irdma
spec:
  kernelArguments:
    - "module_blacklist=irdma"
EOF
```

gpu-cluster-policy.yaml

```
None
$ cat <<EOF > gpu-cluster-policy.yaml
apiVersion: nvidia.com/v1
kind: ClusterPolicy
metadata:
  name: gpu-cluster-policy
spec:
  vgpuDeviceManager:
       config:
       default: default
       enabled: true
  migManager:
```

```
    config:
    default: all-disabled
    name: default-mig-parted-config
    enabled: true
operator:
    defaultRuntime: crio
    initContainer: {}
    runtimeClass: nvidia
    use_ocp_driver_toolkit: true
dcgm:
    enabled: true
gfd:
    enabled: true
dcgmExporter:
    config:
    name: ''
    serviceMonitor:
    enabled: true
    enabled: true
cdi:
    default: false
    enabled: false
driver:
    licensingConfig:
    nlsEnabled: true
    configMapName: ''
    certConfig:
    name: ''
    rdma:
    enabled: true
    kernelModuleConfig:
```

```
      name: ''
      upgradePolicy:
      autoUpgrade: true
      drain:
      deleteEmptyDir: false
      enable: false
      force: false
      timeoutSeconds: 300
      maxParallelUpgrades: 1
      maxUnavailable: 25%
      podDeletion:
      deleteEmptyDir: false
      force: false
      timeoutSeconds: 300
      waitForCompletion:
      timeoutSeconds: 0
      repoConfig:
      configMapName: ''
      virtualTopology:
      config: ''
      enabled: true
      useNvidiaDriverCRD: false
      useOpenKernelModules: true
  devicePlugin:
      config:
      name: ''
      default: ''
      mps:
      root: /run/nvidia/mps
      enabled: true
  gdrcopy:
```

```
     enabled: true
kataManager:
     config:
     artifactsDir: /opt/nvidia-gpu-operator/artifacts/runtimeclasses
mig:
     strategy: single
sandboxDevicePlugin:
     enabled: true
validator:
     plugin:
     env:
     - name: WITH_WORKLOAD
     value: 'false'
nodeStatusExporter:
     enabled: true
daemonsets:
     rollingUpdate:
     maxUnavailable: '1'
     updateStrategy: RollingUpdate
sandboxWorkloads:
     defaultWorkload: container
     enabled: false
gds:
     enabled: true
     image: nvidia-fs
     version: 2.20.5
     repository: nvcr.io/nvidia/cloud-native
vgpuManager:
     enabled: false
vfioManager:
     enabled: true
```

```
  toolkit:
       installDir: /usr/local/nvidia
       enabled: true
EOF
```

ipoib-network.yaml

Shell

```
$ cat <<EOF > ipoib-network.yaml
apiVersion: mellanox.com/v1alpha1
kind: IPoIBNetwork
metadata:
  name: example-ipoibnetwork
spec:
  ipam: |
    {
      "type": "whereabouts",
      "range": "192.168.6.225/28",
      "exclude": [
       "192.168.6.229/30",
       "192.168.6.236/32"
      ]
    }
  master: ibs2f0
  networkNamespace: default
EOF
```

macvlan-network.yaml

Shell

```
cat <<EOF > macvlan-network.yaml
apiVersion: mellanox.com/v1alpha1
kind: MacvlanNetwork
metadata:
  name: rdmashared-net
spec:
  networkNamespace: default
  master: ens8f0np0
  mode: bridge
  mtu: 1500
  ipam: '{"type": "whereabouts", "range": "192.168.2.0/24", "gateway":
"192.168.2.1"}'
EOF
```

Rdma-ib-workload.yaml - Infiniband example

None

```
$ cat <<EOF > rdma-ib-a-workload.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rdma-ib-a-workload
  namespace: default
  annotations:
```

```
      k8s.v1.cni.cncf.io/networks: example-ipoibnetwork
spec:
  nodeSelector:
      kubernetes.io/hostname: <node name>
  serviceAccountName: rdma
  containers:
  - image: quay.io/redhat_emp1/ecosys-nvidia/gpu-operator:tools
      name: rdma-ib-a-workload
      command:
      - sh
      - -c
      - sleep inf
      securityContext:
      privileged: true
      capabilities:
      add: [ "IPC_LOCK" ]
      resources:
      limits:
      nvidia.com/gpu: 1
      rdma/rdma_shared_device_ib: 1
      requests:
      nvidia.com/gpu: 1
      rdma/rdma_shared_device_ib: 1
EOF

$ cat <<EOF > rdma-ib-b-workload.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rdma-ib-b-workload
  namespace: default
```

```
  annotations:
       k8s.v1.cni.cncf.io/networks: example-ipoibnetwork
spec:
  nodeSelector:
       kubernetes.io/hostname: <node name>
  serviceAccountName: rdma
  containers:
  - image: quay.io/redhat_emp1/ecosys-nvidia/gpu-operator:tools
       name: rdma-ib-b-workload
       command:
       - sh
       - -c
       - sleep inf
       securityContext:
       privileged: true
       capabilities:
       add: [ "IPC_LOCK" ]
       resources:
       limits:
       nvidia.com/gpu: 1
       rdma/rdma_shared_device_ib: 1
       requests:
       nvidia.com/gpu: 1
       rdma/rdma_shared_device_ib: 1
EOF
```

Rdma-eth-a-workload.yaml - Ethernet example

Shell

```
$ cat <<EOF > rdma-eth-a-workload.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rdma-eth-a-workload
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: rdmashared-net
spec:
  nodeSelector:
    kubernetes.io/hostname: <node name>
  serviceAccountName: rdma
  containers:
  - image: quay.io/redhat_emp1/ecosys-nvidia/gpu-operator:tools
    name: rdma-eth-a-workload
    command:
      - sh
      - -c
      - sleep inf
    securityContext:
      privileged: true
      capabilities:
        add: [ "IPC_LOCK" ]
    resources:
      limits:
        nvidia.com/gpu: 1
        rdma/rdma_shared_device_eth: 1
      requests:
        nvidia.com/gpu: 1
        rdma/rdma_shared_device_eth: 1
EOF
```

Appendix D - Troubleshooting IB connectivity

This appendix tells you how to confirm the Infiniband connectivity is working between the systems.

Procedure

1. Use `rsh` to connect to each of the rdma workload pods:

```
None
$ oc rsh -n default rdma-ib-32-workload
sh-5.1#
```

2. Run the `ibhosts` command to show Infiniband host nodes in the topology:

```
None
sh-5.1# ibhosts
Ca     : 0x58a2e10300e14446 ports 1 "nvd-srv-33 mlx5_0"
Ca     : 0x58a2e10300dfe416 ports 1 "nvd-srv-32 mlx5_0"
```

3. You can also run the `ibnodes` command that will show not only the nodes but also the switches in the topology:

```
None
sh-5.1# ibnodes
Ca     : 0x58a2e10300e14446 ports 1 "nvd-srv-33 mlx5_0"
```

```
Ca     : 0x58a2e10300dfe416 ports 1 "nvd-srv-32 mlx5_0"
Switch : 0xfc6a1c0300e7ecc0 ports 129 "MF0;qm9700-ib:MQM9700/U1" enhanced port 0
lid 1 lmc 0
```

4. You can look deeper into an interface state by using the `ibstatus` command and pass an interface. If no interface is passed, all will display.

None

```
sh-5.1# ibstatus mlx5_0
Infiniband device 'mlx5_0' port 1 status:
       default gid:  fe80:0000:0000:0000:58a2:e103:00df:e416
       base lid:     0x4
       sm lid:       0x1
       state:        4: ACTIVE
       phys state:   5: LinkUp
       rate:         400 Gb/sec (4X NDR)
       link_layer:   Infiniband
```

5. Now that you have familiarized yourself with the environment, you can run `ibstat` and grep out only certain key elements of the output. These will be needed for the `ibping` test.

   The first `ibstat` output is from the first node, which acts as the server side for the `ibping` command.

None

```
sh-5.1# ibstat | egrep "Port|Base|Link"
       Port 1:
       Physical state: LinkUp
       Base lid: 5
```

```
        Port GUID: 0x58a2e10300e14446
        Link layer: Infiniband
        Port 1:
        Physical state: LinkUp
        Base lid: 0
        Port GUID: 0x0000000000000000
        Link layer: Ethernet
```

The output above shows both an Infiniband and Ethernet interface. You are only interested in the Infiniband in this use case. Make note of the `lid` number as the `ibping` command uses it on the client side.

6. Run `ibstat` on the client side and notice while some of the details are similar, the `lid` number is unique, along with the port GUID:

None

```
sh-5.1# ibstat | egrep "Port|Base|Link"
        Port 1:
        Physical state: LinkUp
        Base lid: 5
        Port GUID: 0x58a2e10300e14446
        Link layer: Infiniband
        Port 1:
        Physical state: LinkUp
        Base lid: 0
        Port GUID: 0x0000000000000000
        Link layer: Ethernet
```

7. Run `ibping` to confirm connectivity:

```
None
sh-5.1# ibping -S -P 1 -d
ibdebug: [114] ibping_serv: starting to serve...
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
ibwarn: [114] mad_respond_via: dest Lid 5
ibwarn: [114] mad_respond_via: qp 0x1 class 0x32 method 129 attr 0x0 mod 0x0
datasz 0 off 0 qkey 80010000
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
ibwarn: [114] mad_respond_via: dest Lid 5
ibwarn: [114] mad_respond_via: qp 0x1 class 0x32 method 129 attr 0x0 mod 0x0
datasz 0 off 0 qkey 80010000
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
ibwarn: [114] mad_respond_via: dest Lid 5
ibwarn: [114] mad_respond_via: qp 0x1 class 0x32 method 129 attr 0x0 mod 0x0
datasz 0 off 0 qkey 80010000
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
ibwarn: [114] mad_respond_via: dest Lid 5
ibwarn: [114] mad_respond_via: qp 0x1 class 0x32 method 129 attr 0x0 mod 0x0
datasz 0 off 0 qkey 80010000
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
ibwarn: [114] mad_respond_via: dest Lid 5
ibwarn: [114] mad_respond_via: qp 0x1 class 0x32 method 129 attr 0x0 mod 0x0
datasz 0 off 0 qkey 80010000
ibdebug: [114] ibping_serv: Pong: rdma-workload-client.(none)
```

8. Run `ibping` on the client side pod to confirm connectivity to the server pod:

```
None
sh-5.1# ibping -P 1 4
Pong from rdma-workload-client.(none) (Lid 4): time 0.013 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.013 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.011 ms
```

```
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.014 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.013 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.013 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.013 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
Pong from rdma-workload-client.(none) (Lid 4): time 0.012 ms
```

Appendix E - Validating RDMA Networking

This section describes running workload pods across the nodes in the environment and the given Infiniband and/or Ethernet shared rdma resources configured previously. You need to set up the required privileges, create the workload pod, validate connectivity between the two hosts on the infinband fabric, and then run a performance test.

Create a Service Account

Procedure

1. Generate a service account CRD to use in the `default` namespace:

```
None
$ cat <<EOF > default-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rdma
  namespace: default
EOF
```

2. Create the account on your cluster:

```
None
$ oc create -f default-serviceaccount.yaml
serviceaccount/rdma created
```

3. Add privileges to the service account:

```
None
$ oc -n default adm policy add-scc-to-user privileged -z rdma
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:privileged added:
"rdma"
```

Create Workload Pods for IB

After you have set up the service account, create a workload pod that contains all the tooling for testing across the Infiniband links.

Procedure

1. Generate two custom pod resource files.

   See rdma-ib-32-workload.yaml for an example of the YAML file.

2. Create the pods on the cluster:

```
None
$ oc create -f rdma-ib-32-workload.yaml
pod/rdma-ib-32-workload created

$ oc create -f rdma-ib-33-workload.yaml
pod/rdma-ib-33-workload created
```

3. Validate that the pods are running:

```
None
$ oc get pods
NAME                READY   STATUS      RESTARTS   AGE
rdma-ib-32-workload   1/1  Running   0          10s
rdma-ib-33-workload   1/1  Running   0          3s
```

Performance Test Across IB Link

This section describes how to run a test across two running pods using `rsh` to connect to each of the `rdma-workload-client` pods and the `ib_write_bw`` and `ib_write_bw <ipaddress>` commands.

Procedure

1. Validate that the pods are running:

```
None
$ oc get pods -n default
NAME                READY   STATUS      RESTARTS   AGE
rdma-ib-32-workload   1/1  Running   0          8m12s
rdma-ib-33-workload   1/1  Running   0          8m5s
```

2. Obtain the ipaddress of the first pod:

```
None
$ oc get pod rdma-ib-32-workload -o yaml | grep -E
'default/example-ipoibnetwork' -A3
       "name": "default/example-ipoibnetwork",
       "interface": "net1",
       "ips": [
             "192.168.6.225"
```

3. Use `rsh` to connect to the first pod and run the `ib_write_bw` command and leave that terminal open:

```
None
$ oc rsh -n default rdma-ib-32-workload
sh-5.1# ib_write_bw

************************************
* Waiting for client to connect... *
************************************
```

4. Open another terminal and `rsh` to the second pod and run `ib_write_bw 192.168.6.225`:

```
None
$ oc rsh -n default rdma-ib-33-workload
sh-5.1# ib_write_bw 192.168.6.225
---------------------------------------------------------------------------------------
--------
                    RDMA_Write BW Test
 Dual-port    : OFF         Device         : mlx5_0
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API  : ON
 TX depth     : 128
 CQ Moderation   : 1
 Mtu          : 4096[B]
 Link type    : IB
 Max inline data : 0[B]
 rdma_cm QPs  : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
--------
 local address: LID 0x05 QPN 0x0cb9 PSN 0xf5fbfc RKey 0x200000 VAddr
0x007fcbace2f000
```

```
 remote address: LID 0x04 QPN 0x0cb9 PSN 0xf5fbfc RKey 0x200000 VAddr
0x007f360e3d8000
---------------------------------------------------------------------------------
--------
 #bytes      #iterations   BW peak[MB/sec]     BW average[MB/sec]
MsgRate[Mpps]
Conflicting CPU frequency values detected: 2500.000000 != 3495.887000. CPU
Frequency is not max.
 65536       5000          44604.62            44576.86            0.713230
---------------------------------------------------------------------------------
--------
```

5. Go back to the first terminal on pod number one. You should see similar response results:

```
None
sh-5.1# ib_write_bw


************************************
* Waiting for client to connect... *
************************************
---------------------------------------------------------------------------------
--------
                    RDMA_Write BW Test
 Dual-port    : OFF         Device         : mlx5_0
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API  : ON
 CQ Moderation   : 1
 Mtu          : 4096[B]
 Link type    : IB
```

```
 Max inline data : 0[B]
 rdma_cm QPs  : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
--------
 local address: LID 0x04 QPN 0x0cb9 PSN 0xf5fbfc RKey 0x200000 VAddr
0x007f360e3d8000
 remote address: LID 0x05 QPN 0x0cb9 PSN 0xf5fbfc RKey 0x200000 VAddr
0x007fcbace2f000
---------------------------------------------------------------------------------
--------
 #bytes      #iterations   BW peak[MB/sec]     BW average[MB/sec]
MsgRate[Mpps]
 65536       5000          44604.62            44576.86            0.713230
---------------------------------------------------------------------------------
--------
```

6. When you have finished, you can clean up the pods since testing is complete. You can also remove the project if you are done with any networking testing.

Create Workload Pods for ETH

This section describes how to test rdma over Ethernet. You can generate a custom pods resource file as follows to meet that requirement.

Procedure

1. Create the contents of `rdma-eth-32-workload.yaml`:

None

```
$ cat <<EOF > rdma-eth-32-workload.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: rdma-eth-32-workload
  namespace: default
  annotations:
       k8s.v1.cni.cncf.io/networks: rdmashared-net
spec:
  nodeSelector:
       kubernetes.io/hostname: nvd-srv-32.nvidia.eng.rdu2.dc.redhat.com
  serviceAccountName: rdma
  containers:
  - image: quay.io/redhat_emp1/ecosys-nvidia/gpu-operator:tools
       name: rdma-eth-32-workload
       command:
       - sh
       - -c
       - sleep inf
       securityContext:
       privileged: true
       capabilities:
       add: [ "IPC_LOCK" ]
       resources:
       limits:
       nvidia.com/gpu: 1
       rdma/rdma_shared_device_eth: 1
       requests:
       nvidia.com/gpu: 1
       rdma/rdma_shared_device_eth: 1
EOF
```

2. Create the contents of `rdma-eth-33-workload.yaml`:

```
None
$ cat <<EOF > rdma-eth-33-workload.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rdma-eth-33-workload
  namespace: default
  annotations:
       k8s.v1.cni.cncf.io/networks: rdmashared-net
spec:
  nodeSelector:
       kubernetes.io/hostname: nvd-srv-33.nvidia.eng.rdu2.dc.redhat.com
  serviceAccountName: rdma
  containers:
  - image: quay.io/redhat_emp1/ecosys-nvidia/gpu-operator:tools
       name: rdma-eth-33-workload
       command:
       - sh
       - -c
       - sleep inf
       securityContext:
       privileged: true
       capabilities:
       add: [ "IPC_LOCK" ]
       resources:
       limits:
       nvidia.com/gpu: 1
       rdma/rdma_shared_device_eth: 1
       requests:
       nvidia.com/gpu: 1
```

```
      rdma/rdma_shared_device_eth: 1
EOF
```

3. Create the pods on the cluster:

```
None
$ oc create -f rdma-eth-32-workload.yaml
pod/rdma-eth-32-workload created

$ oc create -f rdma-eth-33-workload.yaml
pod/rdma-eth-33-workload created
```

4. Validate that the pods are running:

```
None
$ oc get pods -n default
NAME                 READY   STATUS      RESTARTS   AGE
rdma-eth-32-workload   1/1       Running   0          25s
rdma-eth-33-workload   1/1       Running   0          22s
```

Performance Test Across ETH Link

This section describes how to run a test across two running pods using `rsh` to connect to each of the `rdma-workload-client` pods and run inside the `ib_write_bw` and `ib_write_bw <ipaddress>` commands.

Procedure

1. Validate that the pods are running:

```
None
$ oc get pods -n default
NAME                 READY   STATUS      RESTARTS   AGE
rdma-eth-32-workload   1/1       Running   0          106s
rdma-eth-33-workload   1/1       Running   0          103s
```

2. Obtain the ipaddress of the first pod:

```
None
$ oc get pod rdma-eth-32-workload -o yaml | grep -E 'default/rdmashared' -A3
       "name": "default/rdmashared-net",
       "interface": "net1",
       "ips": [
              "192.168.2.1"
```

3. Use `rsh` to connect to the first pod and run the `ib_write_bw` command and leave that terminal open:

```
None
$ oc rsh -n default rdma-eth-32-workload
sh-5.1# ib_write_bw

************************************
* Waiting for client to connect... *
************************************
```

4. Open another terminal and `rsh` to the second pod and run `ib_write_bw 192.168.6.225`:

```
None
$ oc rsh -n default rdma-eth-33-workload
sh-5.1# ib_write_bw 192.168.2.1
---------------------------------------------------------------------------------
--------
                    RDMA_Write BW Test
 Dual-port    : OFF         Device         : mlx5_0
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API  : ON
 TX depth     : 128
 CQ Moderation   : 1
 Mtu          : 4096[B]
 Link type    : IB
 Max inline data : 0[B]
 rdma_cm QPs  : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
--------
 local address: LID 0x05 QPN 0x0ce2 PSN 0x5389f7 RKey 0x1fff00 VAddr
0x007f7368df3000
 remote address: LID 0x04 QPN 0x0ce2 PSN 0x81fa7f RKey 0x1fff00 VAddr
0x007f7e8c890000
---------------------------------------------------------------------------------
--------
 #bytes      #iterations   BW peak[MB/sec]     BW average[MB/sec]
MsgRate[Mpps]
Conflicting CPU frequency values detected: 2500.000000 != 3497.359000. CPU
Frequency is not max.
 65536       5000         44490.32            44467.35            0.711478
---------------------------------------------------------------------------------
--------
```

5. Go back to the first terminal on pod number one to see similar response results:

```
None
sh-5.1# ib_write_bw


************************************
* Waiting for client to connect... *
************************************
---------------------------------------------------------------------------------
--------
                    RDMA_Write BW Test
 Dual-port    : OFF        Device       : mlx5_0
 Number of qps   : 1       Transport type : IB
 Connection type : RC      Using SRQ    : OFF
 PCIe relax order: ON
 ibv_wr* API  : ON
 CQ Moderation   : 1
 Mtu          : 4096[B]
 Link type    : IB
 Max inline data : 0[B]
 rdma_cm QPs  : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
--------
 local address: LID 0x04 QPN 0x0ce2 PSN 0x81fa7f RKey 0x1fff00 VAddr
0x007f7e8c890000
 remote address: LID 0x05 QPN 0x0ce2 PSN 0x5389f7 RKey 0x1fff00 VAddr
0x007f7368df3000
---------------------------------------------------------------------------------
--------
 #bytes      #iterations   BW peak[MB/sec]     BW average[MB/sec]
MsgRate[Mpps]
 65536       5000         44490.32            44467.35            0.711478
```

```
--------------------------------------------------------------------------------
--------
```

6. When you have finished, you can clean up the pods since testing is complete. You can also remove the project if you are done with any networking testing.