Redbooks

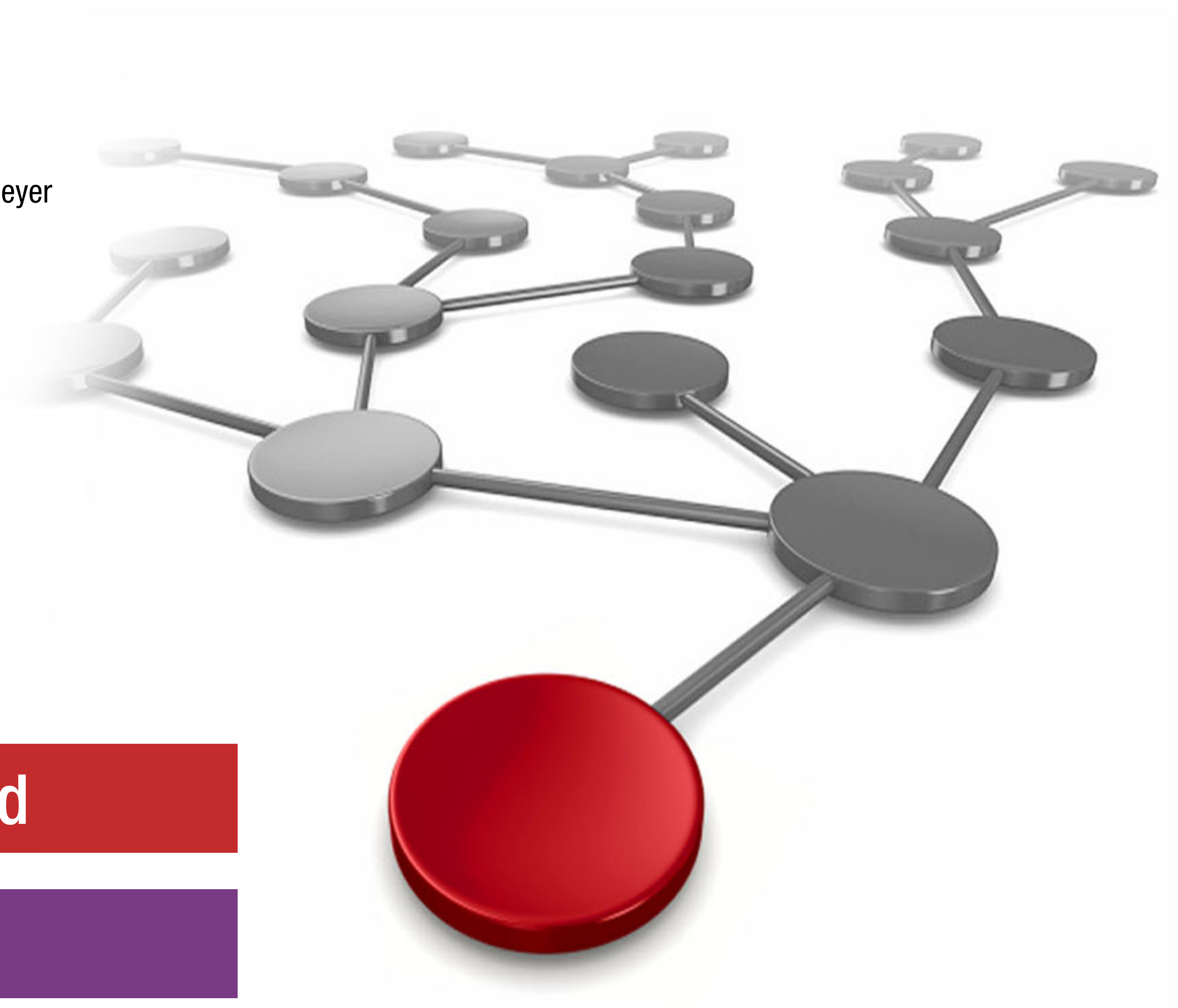
ibm.com/redbooks

Deployment and Usage Guide for Running AI Workloads on Red Hat OpenShift and NVIDIA DGX Systems with IBM Spectrum Scale

Simon Lorenz

Gero Schmidt

Thomas Schoenemeyer



Cloud

Storage

IBM

Redpaper

IBM Redbooks

Deployment and Usage Guide for Running AI Workloads on Red Hat OpenShift and NVIDIA DGX Systems with IBM Spectrum Scale

November 2020

REDP-5610-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (November 2020)

This edition applies to Version 4, Release 4, Modification 3 of Red Hat OpenShift and Version 5 and Release 0, Modification 4.3 of IBM Spectrum Scale.

Contents

Notices

Trademarks

Preface

This IBM® Redpaper publication describes the architecture, installation procedure, and results for running a typical training application that works on an automotive data set in an orchestrated and secured environment that provides horizontal scalability of GPU resources across physical node boundaries for deep neural network (DNN) workloads.

This paper is mostly relevant for systems engineers, system administrators, or system architects that are responsible for data center infrastructure management and typical day-to-day operations such as system monitoring, operational control, asset management, and security audits.

This paper also describes IBM Spectrum® LSF® as a workload manager and IBM Spectrum Discover as a metadata search engine to find the right data for an inference job and automate the data science workflow. With the help of this solution, the data location, which may be on different storage systems, and time of availability for the AI job can be fully abstracted, which provides valuable information for data scientists.

Authors

This paper was produced by a team of specialists from around the world working with IBM Redbooks, Tucson Center.

Simon Lorenz is an IT Architect at IBM Research® and Development in Frankfurt, Germany. He joined IBM Germany in 1993 and has held various positions within IBM and IBM Research and Development. During international assignments, he helped to improve fully automated chip factories in Asia and the US. Simon joined the IBM Spectrum Scale development team in 2014 and since then worked on OpenStack Swift integration by designing and building System Health and Proactive Services solutions. Since March 2019, he is the worldwide leader of the IBM Spectrum Scale Big Data and Analytics team. His role includes designing and building solutions in the area of AI, such as Data Accelerator for AI and Analytics.

Gero Schmidt is a Software Engineer at IBM Germany Research and Development GmbH in the IBM Spectrum Scale development group. He joined IBM in 2001 by working at the European Storage Competence Center in Mainz, Germany to provide technical presales support for a broad range of IBM storage products, with a primary focus on enterprise storage solutions, system performance, and IBM Power Systems servers. Gero participated in the product rollout of the IBM System Storage™ DS6000 and DS8000 series, co-authored several IBM Redbooks®, and was a frequent speaker at IBM international conferences. In 2015, he joined the storage research group at the IBM Almaden Research Center in California, US, where he worked on IBM Spectrum Scale, compression of genomic data in next generation sequencing pipelines, and the development of a cloud-native backup solution for containerized applications in Kubernetes and Red Hat OpenShift. In 2018, he joined the Big Data & Analytics team in the IBM Spectrum Scale development group. He holds a degree in Physics (Dipl.-Phys.) from the Braunschweig University of Technology in Germany.

Thomas Schoenemeyer is a Senior Solution Architect and works for the NVIDIA® EMEA Automotive Enterprise Team since March 2019. His area of expertise covers defining the computing architecture and infrastructure for the development of autonomous vehicles including validation and verification by using NVIDIA enterprise products, such as the DGX family, the Drive Constellation simulation platform, and T4 GPUs.

Thanks to the following people for their contributions to this project:

Larry Coyne
IBM Redbooks, Tucson Center

Juergen Reis, Olaf Weiser, John Lewars, Joseph Dain, Gregory Kishi, Piyush Chaudhary, Przemyslaw Podfigurny, David Wohlford, Pallavi Galgali
IBM Systems

Zvonko Kaiser and Nischal Muchakani
Red Hat

We gratefully acknowledge the contributions to the study that resulted in this paper. We could not have completed this project without the support and guidance of our partners from NVIDIA.

Hilmar Beck, Sebastian Kalcher, Adolf Hohl, Benedikt Keyzers, Vitaliy Razinkov, Dai Yang
NVIDIA

Our sincere appreciation and thanks to all these individuals, who provided insight and expertise that greatly helped the research for this paper.

Now you can become a published author, too!

Comments welcome

- Mail your comments to:

  IBM Corporation, IBM Redbooks
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Overview

This paper focuses on helping companies address the challenges of running large-scale workloads by using orchestration platforms for containerized applications, which are essential to ensure performance, high availability, and efficient horizontal scaling across compute resources. The proof of concept (PoC) that is described in this chapter and explained in detail in this paper provides guidance about how to configure a Red Hat OpenShift 4.4.3 cluster for multi-GPU and multi-node deep learning (DL) workloads. It describes as an example about how to run a real-world automotive industry training workload on a public data set that is provided by Audi.

The scope of the PoC architecture uses Red Hat OpenShift V4.4 on NVIDIA DGX™ systems with IBM Spectrum Scale storage.

1.1 Proof of concept background

For many companies running large-scale infrastructures, orchestration of containerized applications is essential to ensure performance and high availability. Among the top orchestration tools are Kubernetes and Red Hat OpenShift. At the heart of Red Hat OpenShift is Kubernetes, and it is 100% certified Kubernetes, fully open source, and non-proprietary. The API to the Red Hat OpenShift cluster is the same as native Kubernetes. Nothing changes between a container running on any other Kubernetes environment and running on Red Hat OpenShift, and the applications require no changes. Red Hat OpenShift brings several added value capabilities in addition to the fundamental container orchestration capabilities that are provided by Kubernetes.

The capabilities of Red Hat OpenShift with Kubernetes, which are shown in Figure 1-1, makes it a complete, enterprise-ready, and hybrid cloud platform to build, deploy, and manage cloud-native and traditional applications across a hybrid cloud. Automated deployment and lifecycle management for hundreds of independent software vendor (ISV) and custom application workloads and infrastructure services that use Kubernetes Operators and Helm charts is one of many other appealing features. Red Hat OpenShift has over 1,700 customer deployments worldwide across many industry verticals. Also, companies in the automotive industry take advantage of these capabilities to support an orchestration platform that is fully supported by Red Hat.

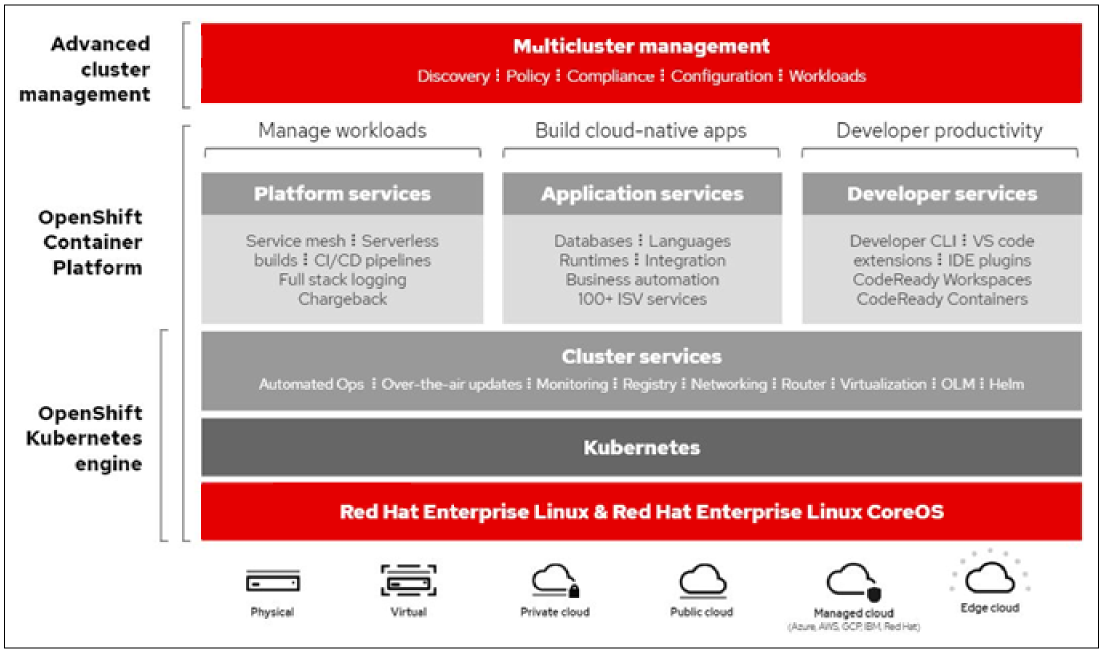

Advanced cluster management
Multicluster management
Discovery ⁝ Policy ⁝ Compliance ⁝ Configuration ⁝ Workloads
OpenShift Container Platform
Manage workloads
Build cloud-native apps
Developer productivity
Platform services
Service mesh ⁝ Serverless builds ⁝ CI/CD pipelines Full stack logging Chargeback
Application services
Databases ⁝ Languages Runtimes ⁝ Integration Business automation 100+ ISV services
Developer services
Developer CLI ⁝ VS code extensions ⁝ IDE plugins CodeReady Workspaces CodeReady Containers
OpenShift Kubernetes engine
Cluster services
Automated Ops ⁝ Over-the-air updates ⁝ Monitoring ⁝ Registry ⁝ Networking ⁝ Router ⁝ Virtualization ⁝ OLM ⁝ Helm
Kubernetes
Red Hat Enterprise Linux & Red Hat Enterprise Linux CoreOS
Physical
Virtual
Private cloud
Public cloud
Managed cloud (Azure, AWS, GCP, IBM, Red Hat)
Edge cloud

Figure 1-1 Red Hat OpenShift Container Platform

For more information, see Red Hat OpenShift Container Platform: Kubernetes for rapid innovation.

Important challenges remain when it comes to large-scale DL workloads, such as the development of DNNs that are trained to be used in the perception software stack for autonomous cars. These large-scale DL workloads are typically associated with applications that use NVIDIA GPUs. NVIDIA provides a hub of GPU-accelerated, optimized containers that can easily scale to hundreds of GPUs that are spread over multiple GPU-accelerated servers.

Scalability requires a balanced system where all components (compute, network, and storage) work hand-in-hand and avoid performance bottlenecks. This PoC describes the architecture and the installation procedure, and it shows the results for running a typical training application working on an automotive data set.

DNN training on huge data sets is a computational expensive task that can take several days on a single server, even with multiple GPUs. The only solution to reduce the training time from days to hours or even minutes is by running DNN training on multiple accelerated servers by using concepts like Message Passing Interface (MPI) and Horovod.

However, studies on multi-node training workloads with NVIDIA GPUs that use Red Hat OpenShift 4.4 are barely found in the literature. The best match is a public study that was created by NVIDIA that is based on Red Hat OpenShift 4.1.

This paper provides guidance about how to configure a Red Hat OpenShift 4.4.3 cluster for DL workloads and describes how to run a training workload on a public data set that is provided by Audi. It also takes a closer look at the *horizontal scalability of GPU resources* across physical node boundaries for DNN workloads on *Red Hat OpenShift 4* as a container orchestration platform and *IBM Spectrum Scale* as highly scalable “data lake” conveniently providing access to the data in a global namespace for containerized AI workloads, without the need to duplicate or copy any data.

For this PoC, the following key components were deployed or used:

- IBM Elastic Storage® System (ESS) 3000 and IBM Spectrum Scale
- IBM Spectrum Scale Container Storage Interface (CSI) drive
- A NVIDIA DGX-1™ system
- A NVIDIA® Mellanox® InfiniBand EDR/HDR interconnect
- A NVIDIA GPU Cloud (NGC) container catalog

The main goal of this PoC is to demonstrate the successful integration of these components and provide performance benchmarks for multi-GPU and multi-node training workloads with a real data set, such as the Audi autonomous electronic vehicle (AEV) Audi Autonomous Driving Dataset (A2D2) that is used for the development of autonomous vehicles.

We also deployed Security Context Constraints (SCCs) to enable granular control of the permissions that are required for pods running AI workloads and for processing access requirements to remote direct memory access (RDMA) resources for best performance. SCCs represent a concept like the way that role-based access control (RBAC) resources control user access that administrators can apply to manage security in Red Hat OpenShift. These permissions include actions that a pod, which is a collection of containers, can perform and what resources it can access. SCCs are used to define a set of conditions that a pod must run with to be accepted into the system.

2 Proof of concept environment

This chapter describes the proof of concept (PoC) environment that is used in this paper.

2.1 Overview

Figure 2-1 shows the environment that is used for this proof of concept (PoC).

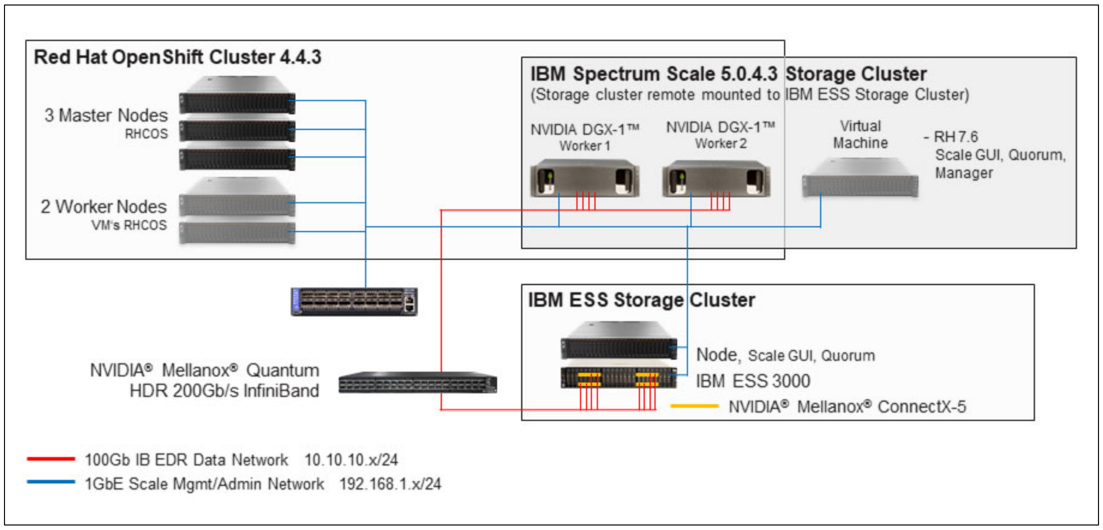

Red Hat OpenShift Cluster 4.4.3
3 Master Nodes
RHCOS
2 Worker Nodes
VM's RHCOS
IBM Spectrum Scale 5.0.4.3 Storage Cluster
(Storage cluster remote mounted to IBM ESS Storage Cluster)
NVIDIA DGX-1™
Worker 1
NVIDIA DGX-1™
Worker 2
Virtual Machine
- RH 7.6
Scale GUI, Quorum, Manager
IBM ESS Storage Cluster
Node, Scale GUI, Quorum
IBM ESS 3000
NVIDIA® Mellanox® ConnectX-5
NVIDIA® Mellanox® Quantum
HDR 200Gb/s InfiniBand
100Gb IB EDR Data Network 10.10.10.x/24
1GbE Scale Mgmt/Admin Network 192.168.1.x/24

Figure 2-1 The environment that is used for this proof of concept

The solution that is described in this paper uses the following major components in the installation:

- DGX-1 systems running with Red Hat 7.6 as worker nodes (40 cores, 512 GB memory, and four single-port NVIDIA Mellanox ConnectX-4 EDR InfiniBand cards).
- A NVIDIA Mellanox Quantum HDR 7800 managed switch to connect worker nodes and a storage back end.
- InfiniBand: Four NVIDIA Mellanox dual-port ConnectX-5 Host Channel Adapters (HCAs) in ESS3000 and four single-port NVIDIA Mellanox ConnectX-4 HCAs in each DGX-1 system that are connected by 16 NVIDIA Mellanox EDR cables.
- NVIDIA Mellanox 100 Gbps EDR InfiniBand Network.
- Standard 1 Gbps (or higher) Ethernet admin network for all components.
- Orchestrator: Red Hat OpenShift 4.4.3 with access to Red Hat subscriptions and other external resources on the internet (for example, GitHub and container images registries).
- IBM Spectrum Scale Cluster 5.0.4.3 with IBM Spectrum Scale GUI/REST.
- IBM ESS3000 with four dual-port NVIDIA Mellanox ConnectX-5 InfiniBand cards running IBM Spectrum Scale 5.0.4.3.

2.2 Prerequisites

Figure 2-2 shows the software release levels that are used for Red Hat OpenShift and IBM Spectrum Scale, and the role of each node.

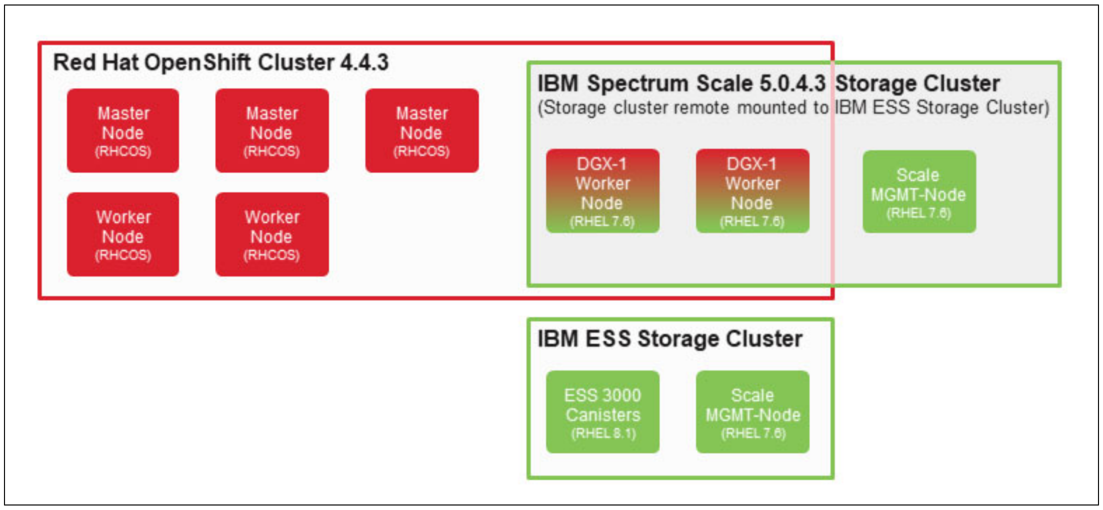

Red Hat OpenShift Cluster 4.4.3
Master Node (RHCOS)
Master Node (RHCOS)
Master Node (RHCOS)
Worker Node (RHCOS)
Worker Node (RHCOS)
IBM Spectrum Scale 5.0.4.3 Storage Cluster
(Storage cluster remote mounted to IBM ESS Storage Cluster)
DGX-1 Worker Node (RHEL 7.6)
DGX-1 Worker Node (RHEL 7.6)
Scale MGMT-Node (RHEL 7.6)
IBM ESS Storage Cluster
ESS 3000 Canisters (RHEL 8.1)
Scale MGMT-Node (RHEL 7.6)

Figure 2-2 Red Hat OpenShift and IBM Spectrum Scale node roles and software releases

The following clusters were created for this PoC:

- Red Hat OpenShift Cluster 4.4.3

  Red Hat OpenShift is an open source container orchestration platform that is based on the Kubernetes container orchestrator. It is designed for enterprise app development and deployment. As an operating system, we deployed Red Hat Enterprise Linux CoreOS (RHCOS), and Red Hat Enterprise Linux (RHEL). For more information about RHCOS, see “Related publications” on page 65. RHCOS is the only supported operating system for Red Hat OpenShift Container Platform master node hosts. RHCOS and RHEL are both supported operating systems for Red Hat OpenShift Container Platform x86-based worker nodes. As IBM Spectrum Scale and DGX-1 systems are supported only by RHEL, we used RHEL 7.6 for those systems.

  The compute-cluster consists of the following components:

  – Three master nodes running RHCOS on a Lenovo SR650.
  – Two worker nodes running RHCOS in a VM on a Lenovo SR650 (used to have a minimal healthy OCP cluster running as a base before adding the DGX-1 systems and testing with different configurations).
  – Two DGX-1 worker nodes.

- IBM Spectrum Scale 5.0.4.3 Storage Cluster

  IBM Spectrum Scale is a high-performance and highly available clustered file system and associated management software that is available on various platforms. IBM Spectrum Scale can scale in several dimensions, including performance (bandwidth and IOPS), capacity, and number of nodes or instances that can mount the file system.

  The storage client cluster consists of:

  – IBM Spectrum Scale clients running on every DGX-1 that is based on Red Hat 7.6.
  – IBM Spectrum Scale client running in a VM (providing GUI / REST, Quorum, and Management functions) that is based on Red Hat 7.6.

  - A remote-mounted IBM Spectrum Scale file system that is called ess3000_4M from an IBM ESS3000 storage system, which is configured with a 4 MiB blocksize (good fit to the average image size of 3 - 4 MiB of the used Audi Autonomous Driving Dataset (A2D2)).

- IBM Elastic Storage System (ESS) Storage Cluster

  IBM ESS 3000 combines IBM Spectrum Scale file management software with NVMe flash storage for the ultimate scale-out performance and unmatched simplicity by delivering 40 GBps of data throughput per 2U system.

  The storage cluster consists of the following components:

  - The IBM ESS3000 consists of two canisters, each running IBM Spectrum Scale 5.0.4.3 on Red Hat 8.1.
  - A2D2 downloaded and extracted into the IBM ESS3000 storage system.
  - Lenovo SR650 server running IBM Spectrum Scale 5.0.4.3 on Red Hat 7.6 (providing GUI / REST, Quorum, and Management functions).

For more information about IBM Spectrum Scale, IBM ESS 3000, DGX-1 systems, and Red Hat OpenShift, see "Related publications" on page 65.

3

Installation

The installation procedure that is described in this chapter has the following steps:

- Configuring the NVIDIA Mellanox EDR InfiniBand network
- Integrating DGX-1 systems as worker nodes into a Red Hat OpenShift 4.4.3 cluster
- Adding DGX-1 systems as client nodes to the IBM Spectrum Scale cluster
- Installing and configuring more components in the Red Hat OpenShift 4.4.3 stack

3.1 Configuring the NVIDIA Mellanox EDR InfiniBand network

In this section, we complete the following tasks:

- Enable Subnet Manager (SM) on the NVIDIA Mellanox HDR switch.
- Configure and connect IBM ESS3000 with eight EDR ports total (8x 100 Gbps).
- Configure and connect DGX-1 nodes with four EDR ports each (4x 100 Gbps / DGX-1).

In this deployment scenario, each of the two DGX-1 systems and each of the two IBM ESS3000 I/O server nodes is connected with four EDR ports to a NVIDIA Mellanox Quantum HDR 200 Gbps QM8700 InfiniBand Smart Switch, as shown in Chapter 2, “Proof of concept environment” on page 5. This configuration provides a total of eight 100 Gbps EDR InfiniBand connections between the IBM ESS3000 storage system and the two DGX-1 systems.

The InfiniBand SM is started on the InfiniBand switch by using the following configuration:

```
[standalone: master] > enable
[standalone: master] # configure terminal
[standalone: master] (config) # ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com enable
[standalone: master] (config) # ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com sm-priority 0
[standalone: master] (config) # ib sm virt enable
[standalone: master] (config) # write memory
[standalone: master] (config) # reload
```

Here is the active switch configuration that is used for the PoC:

```
[standalone: master] (config) # show running-config
##
## Running database "initial"
## Generated at 2020/05/13 15:45:50 +0000
## Hostname: DGX-IB-switch1.kelsterbach.de.ibm.com
## Product release: 3.9.0300
##
##
## Running-config temporary prefix mode setting
##
no cli default prefix-modes enable
##
## Subnet Manager configuration
##
ib sm virt enable
##
## Network interface configuration
##
no interface mgmt0 dhcp
interface mgmt0 ip address 9.155.106.250 /24
##
## Other IP configuration
##
hostname DGX-IB-switch1.kelsterbach.de.ibm.com
ip domain-list kelsterbach.de.ibm.com
ip name-server 9.155.106.9
ip route vrf default 0.0.0.0/0 9.155.106.1
##
## Other IPv6 configuration
##
no ipv6 enable
##
## Local user account configuration
```

```
##
username admin password 7 $6$dHjW/Juo$LYlJA...9qZgAVyLylNDAkzyVTXyCbWzcO
username monitor password 7 $6$26O9wpNF$E1G35T.9eakkl...ShK6NIn2r4zMIsdIn8M1
##
## AAA remote server configuration
##
# ldap bind-password ********
# radius-server key ********
# tacacs-server key ********
##
## Network management configuration
##
# web proxy auth basic password ********
##
## X.509 certificates configuration
##
#
# Certificate name system-self-signed, ID 8a57f97b8877d86c46cc3bf2c3b9a1c15a6259d9
# (public-cert config omitted since private-key config is hidden)
##
## IB nodename to GUID mapping
##
ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com create
ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com enable
ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com sm-priority 0
##
## Persistent prefix mode setting
##
cli default prefix-modes enable

[standalone: master] (config) # show ib smnode DGX-IB-switch1.kelsterbach.de.ibm.com
sm-state enabled
```

3.2 Integrating DGX-1 systems as worker nodes into a Red Hat OpenShift 4.4.3 cluster

In this section, we complete the following tasks:

- Install the Red Hat Enterprise Linux (RHEL) 7.6 and NVIDIA RPM packages.
- Install the NVIDIA Mellanox InfiniBand drivers (`MLNX_OFED`).
- Install the NVIDIA® GPUDirect® Remote Direct Memory Access (RDMA) Kernel Module (`nv_peer_mem`).
- Install the NVIDIA Mellanox SELinux Module.
- Add DGX-1 systems as RHEL7 based worker nodes to the Red Hat OpenShift 4.4.3 cluster.

On x86 platforms, Red Hat OpenShift 4 can be extended with RHEL7 based worker nodes (as described in Adding RHEL compute machines to a Red Hat OpenShift Container Platform cluster). In this paper, we use this concept to integrate the DGX-1 systems as worker nodes into the existing OpenShift 4.4.3 cluster. Here are the steps that are involved:

- Installing RHEL 7.6 as the base OS on the DGX-1 systems
- Installing the DGX software for RHEL

- Installing the NVIDIA Mellanox OpenFabrics Enterprise Distribution (OFED) (MLNX_OFED)
- Installing the GPUDirect RDMA (GDRDMA) Kernel Module
- Adding the DGX-1 systems as worker nodes to the Red Hat OpenShift 4.4.3 cluster

3.2.1 Installing the Red Hat Enterprise Linux 7.6 and DGX software

The RHEL 7.6 base OS can be installed according to the DGX-1 instructions in Installing Red Hat Enterprise Linux. Then, you install the DGX Software as described in Installing the DGX Software.

Important: When following the steps in these instructions, you *must stop* just before installing the NVIDIA® CUDA® driver (Installing and Loading the NVIDIA CUDA Drivers). The Compute Unified Device Architecture (CUDA) driver should not be installed on the host OS when the node is integrated as a worker node into a Red Hat OpenShift cluster because the CUDA functions are provided by the SRO running in Red Hat OpenShift.

DGX worker nodes for Red Hat OpenShift must not be pre-configured with NVIDIA components (CUDA driver, container runtime, and device plug-in). For more information, see the documentation for the NVIDIA GPU-Operator.

Complete the following steps:

1. Install the NVIDIA repo for RHEL7 by running the following command:

```
# yum install -y
https://international.download.nvidia.com/dgx/repos/rhel-files/dgx-repo-setup-1
9.07-2.el7.x86_64.rpm
```

2. Enable the NVIDIA updates repo in `/etc/yum.repos.d/nvidia-dgx-7.repo` with the following code:

```
[nvidia-dgx-7-updates]
name=NVIDIA DGX EL7 Updates
baseurl=https://international.download.nvidia.com/dgx/repos/rhel7-updates/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-dgx-cosmos-support
```

3. Install the NVIDIA software packages for DGX-1 systems by running the following command:

```
# yum groupinstall -y 'DGX-1 Configurations'
```

4. Update the RHEL 7.6 packages and kernel.

   Before running the **`yum update`** command, we set the minor release to RHEL 7.6 in the Red Hat Subscription Manager (RHSM) to gain more control over the package and kernel updates, as shown in the following code. By setting it to RHEL 7.6, we ensure that we stay within this minor release for the kernel and package updates. The Linux kernel that is used in this paper is 3.10.0-957.27.2.el7.x86_64 on the DGX-1 systems. Red Hat OpenShift 4.4. generally supports RHEL7 minor versions RHEL 7.6 - RHEL 7.8 according to System requirements for RHEL compute nodes and Red Hat OpenShift Container Platform 4.x Tested Integrations (for x86_x64), but not RHEL 8 currently.

```
# subscription-manager release -set=7.6
# subscription-manager release --show
Release: 7.6
```

```
# yum clean all
# yum update
```

5. Restart the system to load the drivers and to update the system configurations.

3.2.2 Installing NVIDIA Mellanox InfiniBand drivers (MLNX_OFED)

To install the *NVIDIA Mellanox OFED* (`MLNX_OFED`) for Linux for Red Hat operating systems, follow the instructions at Installing NVIDIA Mellanox InfiniBand Drivers. In our PoC, we install NVIDIA Mellanox OFED `MLNX_OFED_LINUX-4.7-3.2.9.0` on both DGX-1 systems by using the same release version that is on the IBM ESS3000 system.

After installing the `MLNX_OFED` drivers, skip the step to install the NVIDIA peer memory kernel module (`nv_peer_mem`) from the pre-built RPM package because it does not install properly without a full CUDA installation (which you do not have installed on the host OS). Instead, build and install the NVIDIA peer memory kernel module (`nv_peer_mem`) manually by following the instructions in 3.2.3, "Installing the GDRDMA kernel module" on page 13.

3.2.3 Installing the GDRDMA kernel module

The NVIDIA peer memory (`nv_peer_mem`) kernel module is required for GDRDMA support. Because the installation of the provided RPM package requires a CUDA installation that is not present in our setup, we install the kernel module manually. GDRDMA enables multiple GPUs and network adapters to directly read and write CUDA host and device memory, which eliminates unnecessary memory copies and lowers CPU processing impact and latency, which results in performance improvements in data transfer times for applications running on NVIDIA Tesla and Quadro products.

To install the GDRDMA Kernel Module without CUDA, complete the following steps:

1. Compile and install the `nv_peer_mem` source code as follows on one DGX system:

```
# export NVIDIA=/run/nvidia/driver
# export KERNEL_VERSION=$(uname -r)
# ln -sf ${NVIDIA}/usr/src/nvidia-* /usr/src/.
# yum -y group install "Development Tools"
# yum -y install kernel-devel-${KERNEL_VERSION}
kernel-headers-${KERNEL_VERSION} kmod binutils perl elfutils-libelf-devel
# git clone https://github.com/Mellanox/nv_peer_memory.git
# cd /root/nv_peer_memory
# sed -i 's/updates\/dkms/kernel\/drivers\/video/g' create_nv.symvers.sh
# ./build_module.sh
# ln -sf  \
/run/nvidia/driver/lib/modules/3.10.0-957.27.2.el7.x86_64/kernel/drivers/video/
nvidia* \
/lib/modules/3.10.0-957.27.2.el7.x86_64/kernel/drivers/video/.
# rpmbuild --rebuild /tmp/nvidia_peer_memory-*
# rpm -ivh /root/rpmbuild/RPMS/x86_64/nvidia_peer_memory-1.0-9.x86_64.rpm
```

2. Distribute the built RPM and install it on the other DGX systems.
3. Check that the module is loaded. If it is not loaded, load it by running **`modprobe nv_peer_mem`**:

```
# lsmod | grep peer
nv_peer_mem 13163 0
nvidia 19893032 3261 nv_peer_mem,nvidia_modeset,nvidia_uvm
```

```
   ib_core 368807 11
   rdma_cm,ib_cm,iw_cm,nv_peer_mem,mlx4_ib,mlx5_ib,ib_ucm,ib_umad,ib_uverbs,rdma_u
   cm,ib_ipoib
```

Make sure that the module is loaded automatically on *every* system restart.

3.2.4 Installing the NVIDIA Mellanox SELinux module

Download and decompress the NVIDIA Mellanox SELinux module from the Mellanox docs website and install it on the DGX worker nodes by running the following commands:

```
# semodule -i infiniband.pp
# semodule -l | grep -i infi
infiniband 1.0
```

3.2.5 Adding DGX-1 systems as worker nodes to the Red Hat OpenShift cluster

Finally, the DGX-1 systems are ready to be added to the Red Hat OpenShift cluster as RHEL7 based worker nodes. To do this task, follow the instructions at Adding RHEL compute machines to a Red Hat OpenShift Container Platform cluster.

Ensure that you have an active Red Hat OpenShift Container Platform subscription on your Red Hat account. You must register each host with RHSM, attach an active Red Hat OpenShift Container Platform subscription, and enable the required repositories.

Add your new worker nodes to your active Red Hat OpenShift load balancer configuration (for example, HAproxy).

After this step you should see the DGX-1 systems appearing as worker nodes in the Red Hat OpenShift cluster:

```
# oc get nodes -o wide
NAME                        STATUS   ROLES    AGE   VERSION   INTERNAL-IP    OS-IMAGE
dgx01.ocp4.scale.ibm.com    Ready    worker   6d3h  v1.17.1   192.168.1.19   OpenShift Enterprise
dgx02.ocp4.scale.ibm.com    Ready    worker   6d2h  v1.17.1   192.168.1.20   OpenShift Enterprise
master01.ocp4.scale.ibm.com Ready    master   13d   v1.17.1   192.168.1.11   Red Hat Enterprise Linux CoreOS
master02.ocp4.scale.ibm.com Ready    master   13d   v1.17.1   192.168.1.12   Red Hat Enterprise Linux CoreOS
master03.ocp4.scale.ibm.com Ready    master   13d   v1.17.1   192.168.1.13   Red Hat Enterprise Linux CoreOS
worker03.ocp4.scale.ibm.com Ready    worker   12d   v1.17.1   192.168.1.17   Red Hat Enterprise Linux CoreOS
worker04.ocp4.scale.ibm.com Ready    worker   12d   v1.17.1   192.168.1.18   Red Hat Enterprise Linux CoreOS
```

3.3 Adding DGX-1 systems as client nodes to the IBM Spectrum Scale cluster

In this section, we complete the following tasks:

- Add DGX worker nodes to the local IBM Spectrum Scale V 5.0.4.3 cluster as IBM Spectrum Scale client nodes.
- Configure the DGX-1 nodes in the local IBM Spectrum Scale cluster.

Both DGX-1 systems are now part of the Red Hat OpenShift 4.4.3 cluster as RHEL7 based worker nodes. You must add these DGX worker nodes to the local IBM Spectrum Scale 5.0.4.3 cluster as IBM Spectrum Scale *client nodes* either by using the *IBM Spectrum Scale installation toolkit* that is described in Adding nodes, NSDs, or file systems to an existing installation or manually following the instructions in Adding nodes to a GPFS cluster at the IBM Spectrum Scale IBM Knowledge Center.

By using the IBM Spectrum Scale installation toolkit and meeting all the prerequisites, you can add the DGX-1 worker nodes as client nodes to the IBM Spectrum Scale cluster by running the following commands:

```
# ./spectrumscale node add dgx01.ocp4.scale.ibm.com
# ./spectrumscale node add dgx02.ocp4.scale.ibm.com
# ./spectrumscale install [--precheck]
# ./spectrumscale deploy [--precheck]
```

Afterward, the two DGX-1 worker nodes show up as new clients in the IBM Spectrum Scale cluster:

```
# mmlscluster

GPFS cluster information
========================
   GPFS cluster name:         SpectrumScale.ocp4.scale.ibm.com
   GPFS cluster id:           16217308676014575381
   GPFS UID domain:           SpectrumScale.ocp4.scale.ibm.com
   Remote shell command:      /usr/bin/ssh
   Remote file copy command:  /usr/bin/scp
   Repository type:           CCR

Node  Daemon node name        IP address    Admin node name         Designation
----------------------------------------------------------------------------------
   1   scale00.ocp4.scale.com  192.168.1.30  scale00.ocp4.scale.ibm.com quorum- manager-perfmon
   2   dgx01.ocp4.scale.com    192.168.1.19  dgx01.ocp4.scale.ibm.com    perfmon
   3   dgx02.ocp4.scale.com    192.168.1.20  dgx02.ocp4.scale.ibm.com    perfmon
[ ...  plus additional nodes in the IBM Spectrum Scale client cluster ...]
```

In our setup, the local IBM Spectrum Scale cluster remotely mounts an IBM Spectrum Scale file system that is called *ess3000_4M* from an IBM ESS3000 storage system. For more information about managing access to a remote IBM Spectrum Scale file system, see Accessing a remote GPFS file system and Mounting a remote GPFS file system.

```
# mmremotecluster show
Cluster name:    ess3000.bda.scale.ibm.com
Contact nodes:   fscc-fab3-3-a-priv.bda.scale.com,fscc-fab3-3-b-priv.bda.scale.com
SHA digest: 9c58d9df69804393571044a9f08b005db12345e4cb87637cad83870a9f74c24d
File systems:    ess3000_4M (ess3000_4M)
```

The file system is configured with a 4 MiB block size, which is a good fit for the average image size of 3 - 4 MiB that is used for the autonomous vehicle training data in this paper.

The IBM ESS3000 and DGX-1 systems are configured to allow all ports on the EDR InfiniBand network to be used for storage I/O, that is, the four NVIDIA Mellanox ConnectX-5 ports on each of the two IBM ESS3000 I/O server nodes, and the four NVIDIA Mellanox ConnectX-4 ports on each DGX-1 system. This configuration provides a total bandwidth of eight InfiniBand EDR 100 Gbps links to the IBM ESS3000 and four to each DGX-1 system.

In addition to the InfiniBand daemon network for data transfers, we configure an extra IP address on all `mlx5_1` interfaces (by using Internet Protocol over InfiniBand and 10.10.11.0/24) on the IBM ESS3000 nodes and the DGX-1 nodes as an extra *subnet* for IBM Spectrum Scale (by using the subnets configuration parameter), which is used for the **`mmfsd`** daemon TCP/IP communication over the 100 Gbps link instead of the 1 Gbps admin network. With **`verbsRdmaSend`** enabled, the amount of communication over this TCP/IP link is negligible. The 1 Gbps admin network is used as the default admin and daemon network connecting all IBM Spectrum Scale, DGX, and IBM Elastic Storage System (ESS) nodes.

Note: For more information about using the “subnets” configuration parameter in IBM Spectrum Scale, see the following resources:

- Using public and private IP addresses for GPFS nodes
- The mmchconfig command

The DGX-1 nodes in the local IBM Spectrum Scale cluster are configured as follows, with `verbsRdma`, `verbsRdmaSend`, and `verbsPorts` enabling InfiniBand RDMA on the DGX-1 worker nodes and `enforceFilesetQuotaOnRoot yes` and `controlSetxattrImmutableSELinux yes` as requirements for the IBM Spectrum Scale CSI driver:

```
[dgx]
pagepool 128G
workerThreads 1024
ignorePrefetchLUNCount yes
maxFilesToCache 1M
maxStatCache 1M
subnets 10.10.11.0/ess3000.bda.scale.ibm.com;SpectrumScale.ocp4.scale.ibm.com
verbsRdma enable
verbsRdmaSend yes
verbsPorts mlx5_0/1 mlx5_1/1 mlx5_2/1 mlx5_3/1
[common]
maxMBpS 20000
enforceFilesetQuotaOnRoot yes
controlSetxattrImmutableSELinux yes
```

3.4 Installing and configuring more components in the Red Hat OpenShift 4.4.3 stack

In this section, we complete the following tasks:

- Describe the Special Resource Operator (SRO) for GPU support.
- Describe the NVIDIA Mellanox RDMA Shared Device Plugin for InfiniBand RDMA support.
- Describe Security Context Constraints (SCCs) with the `IPC_LOCK` capability.
- Describe service account, role, and role binding per user namespace to grant access to RDMA resources for pods that use the RDMA Shared Device Plugin.

- Describe the Message Passing Interface (MPI) Operator, which you use to conveniently schedule multi-GPU and multi-node training jobs.
- Describe the IBM Spectrum Scale Container Storage Interface (CSI), which you use to provide access to data in IBM Spectrum Scale, which offers parallel access to data in a global namespace across worker nodes in the Red Hat OpenShift cluster without needing to duplicate (copy or move) huge amounts of data for model training, model validation, or inference.

To fully use the GPU and high-performance InfiniBand RDMA capabilities of the DGX-1 systems and shared access to the data in the global namespace of the parallel IBM Spectrum Scale file system, we install more components into the Red Hat OpenShift stack:

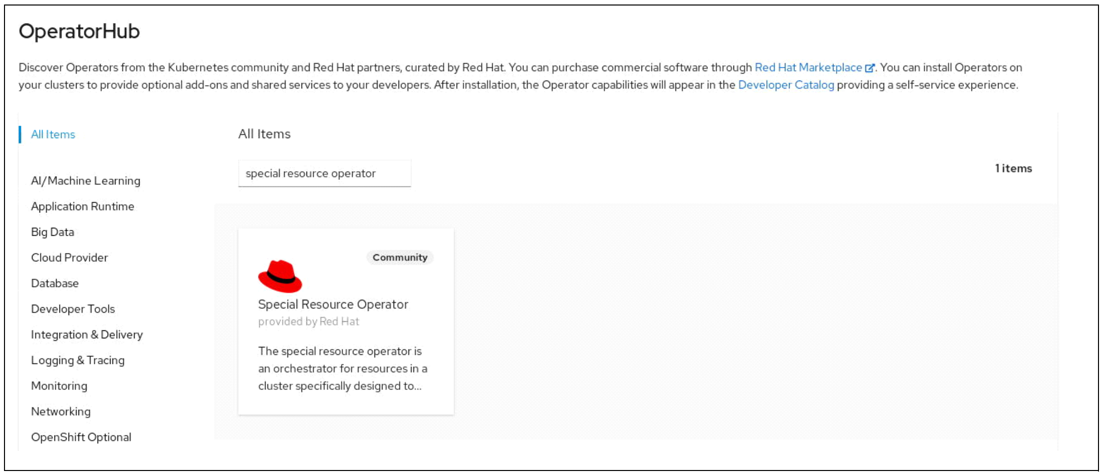
- SRO for GPU-resources
- NVIDIA Mellanox RDMA Shared Device Plugin for InfiniBand RDMA resources
- MPI Operator to run orchestrated MPI jobs in Red Hat OpenShift
- IBM Spectrum Scale CSI plug-in to access data in the IBM Spectrum Scale file system

3.4.1 Special Resource Operator

The SRO extends Red Hat OpenShift to support special resources that need extra management like the NVIDIA GPUs in the DGX-1 systems. Without the SRO, there would be no GPU resource that is known to the cluster, and the appropriate scheduling of pods relying on that resource and providing the correct drivers (like CUDA) would not be available. The SRO (or a similar extension to Red Hat OpenShift) is required to effectively schedule workloads that rely on GPUs in a Red Hat OpenShift cluster.

The SRO relies on the NFD *operator* and its node feature discovery capabilities to label the worker nodes in the Red Hat OpenShift cluster with node-specific attributes, like PCI cards, kernel or OS version, and other components.

The SRO is also available as an operator in the OperatorHub and can be installed and deployed directly from the Red Hat OpenShift 4 GUI by using the Operator Lifecycle Management framework, as shown in Figure 3-1.


OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through Red Hat Marketplace. You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the Developer Catalog providing a self-service experience.

Special Resource Operator
provided by Red Hat

The special resource operator is an orchestrator for resources in a cluster specifically designed to...

Figure 3-1 Special Resource Operator: Installed and deployed directly from the Red Hat OperatorHub

For this paper, we used the latest version (as of 25 June 17:23 CEST) of SRO on GitHub, which can be installed from the command line by the system admin of the Red Hat OpenShift cluster as follows:

1. Install the NFD operator by running the following commands:

   ```
   # git clone https://github.com/openshift/cluster-nfd-operator
   # cd cluster-nfd-operator
   # make deploy
   ```

2. Install the SRO operator by running the following commands:

   ```
   # git clone https://github.com/openshift-psap/special-resource-operator
   # cd special-resource-operator
   # PULLPOLICY=Always make deploy
   ```

It might take a while for the SRO operator to build the required container images and start all the required pods successfully. During that time, you notice that SRO pods are failing and restarting until all required build processes complete successfully and the SRO pods finally enter a steady running state.

Make sure that the nouveau kernel module is disabled, which should already be the case when installing the NVIDIA packages for the DGX-1 system. Otherwise, disable the nouveau module manually.

The SRO adds the following allocatable resources to each DGX-1 worker node (eight GPUs in this case):

```
Allocatable:
  nvidia.com/gpu: 8
```

With the SRO version that we use for this setup, we must set SElinux on the DGX-1 worker nodes to "permissive" mode so that the SRO can successfully build the required container images and start all the necessary pods.

3.4.2 NVIDIA Mellanox RDMA Shared Device plug-in

In addition to the NFD and SRO operators that add NVIDIA GPUs as a new resource in the Red Hat OpenShift cluster, we also must install the NVIDIA Mellanox RDMA Shared Device plug-in to further provide InfiniBand resources to Red Hat OpenShift, which provides non-privileged pods access to GPU and InfiniBand resources and allows Red Hat OpenShift access to scheduled pods requesting these resources.

InfiniBand enables high-performance communication between GPUs with the NVIDIA Collective Communications Library (NCCL) so that multi-node workloads can scale out seamlessly across worker nodes. The NVIDIA Mellanox RDMA device plug-in runs as a daemonset on all worker nodes and allows a granular assignment of individual NVIDIA Mellanox InfiniBand resources to pods.

To install the RDMA device plug-in, complete the following steps:

1. Clone the GitHub repository by running the following command:

   ```
   # git clone https://github.com/Mellanox/k8s-rdma-shared-dev-plugin.git
   ```

2. Follow the instructions at GitHub.
3. Create and deploy the configuration map.

The DGX-1 systems have four single-port NVIDIA Mellanox ConnextX-4 EDR InfiniBand cards (NVIDIA Mellanox Technologies MT27700 Family) that are associated with two NUMA nodes:

```
-      mlx5_0, mlx5_1     (numa0)
-      mlx5_2, mlx5_3     (numa1)
```

In our setup, we configured the configuration map for the NVIDIA Mellanox device driver plug-in to make all four InfiniBand ports that are available in Red Hat OpenShift as selectable resources that are named `shared_ib0`, `shared_ib1`, `shared_ib2`, and `shared_ib3`:

```
# cat images/k8s-rdma-shared-dev-plugin-config-map.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: rdma-devices
  namespace: kube-system
data:
  config.json: |
    {
        "configList": [{
             "resourceName": "shared_ib0",
             "rdmaHcaMax": 100,
             "devices": ["ib0"]
           },
           {
             "resourceName": "shared_ib1",
             "rdmaHcaMax": 100,
             "devices": ["ib1"]
           },
           {
             "resourceName": "shared_ib2",
             "rdmaHcaMax": 100,
             "devices": ["ib2"]
           },
           {
             "resourceName": "shared_ib3",
             "rdmaHcaMax": 100,
             "devices": ["ib3"]
           }
        ]
    }
```

Then, we deployed the configmap by running the following command:

```
# oc apply -f images/k8s-rdma-shared-dev-plugin-config-map.yaml
```

4. Deploy the NVIDIA Mellanox RDMA device plug-in by running the following command:

   ```
   # oc apply -f images/k8s-rdma-shared-dev-plugin-ds.yaml
   ```

   The device plug-in adds the following allocatable resources to the DGX-1 worker nodes in Red Hat OpenShift:

   ```
   Allocatable:
     rdma/shared_ib0:    100
     rdma/shared_ib1:    100
     rdma/shared_ib2:    100
     rdma/shared_ib3:    100
   ```

Unlike GPUs, a container can request only one quantity of a specific RDMA resource, for example, `rdma/shared_ib0: 1`, which enables access to the requested RDMA resource. Higher quantities, for example, `rdma/shared_ib0: 2` are not possible. Instead, a container can request access to multiple different RDMA resources concurrently, for example, `rdma/shared_ib0: 1`, `rdma/shared_ib1: 1`, and so on.

The GitHub repository provides a `mofed-test-pod` that can be adapted to the local configuration and used by the system admin to check proper RDMA functions, for example:

```
# cat ../k8s-rdma-shared-dev-plugin/example/test-hca-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mofed-test-pod
spec:
  restartPolicy: OnFailure
  containers:
  - image: mellanox/centos_7_4_mofed_4_2_1_2_0_0_60
    name: mofed-test-ctr
    securityContext:
      capabilities:
        add: [ "IPC_LOCK" ]
    resources:
      limits:
        rdma/shared_ib0: 1
        rdma/shared_ib1: 1
        rdma/shared_ib2: 1
        rdma/shared_ib3: 1
    command:
    - sh
    - -c
    - |
      ls -l /dev/infiniband /sys/class/net
      sleep 1000000
```

You can log in to the running pod by running **`oc rsh mofed-test-pod`** and run regular **`ibstat`**, **`ibhosts`**, **`ib_write_bw`**, and similar commands to verify the connectivity across your InfiniBand network interactively.

3.4.3 Enabling the IPC_LOCK capability in the user namespace for the RDMA Shared Device plug-in

Using the RDMA device plug-in requires the `IPC_LOCK` capability from the Red Hat OpenShift security context. This capability is not generally available to a regular user who is normally running under the “restricted” SCCs in Red Hat OpenShift 4. However, the system admin has access to the “privileged” SCC and can immediately run the `mofed-test-pod` that is mention in 3.4.2, “NVIDIA Mellanox RDMA Shared Device plug-in” on page 18.

To allow a regular user to run jobs requesting the `IPC_LOCK` capability (for example, for jobs requesting RDMA resources for optimal multi-GPU usage across nodes), the system admin can add, for example, a new SCC that is derived from the “restricted” SCC, and extend it by using the IPC_LOCK capability:

```
defaultAddCapabilities:
- IPC_LOCK
```

To make the new SCC available to a user's namespace, the system admin must create a *service account*, a *role* binding, and a role referencing this new SCC by completing the following steps:

1. Create an SCC that includes the `IPC_LOCK` capability.

   We create and apply an SCC that is derived from the “restricted” SCC, add the `IPC_LOCK` capability, and name it `scc-for-mpi` because we intend to use it when running MPI jobs requesting multiple GPUs.

```
# apply -f mpi-scc.yaml
# cat mpi-scc.yaml
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities:
- IPC_LOCK
fsGroup:
  type: MustRunAs
groups:
- system:authenticated
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: cloned from restricted SCC adds IPC_LOCK
  name: scc-for-mpi
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:name-of-user-namespace:mpi
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

2. Create a *service account* in the user's namespace.

   We create a service account with the name `mpi` in the user's namespace:

   ```
   # apply -f mpi-sa.yaml
   # cat mpi-sa.yaml
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: mpi
     namespace: name-of-user-namespace
   ```

3. Create a role in the user's namespace.

   We create a role with the name `mpi` in the user's namespace that is referring to the newly created SCC that is named `scc-for-mpi` that includes the `IPC_LOCK` capability:

   ```
   # apply -f mpi-role.yaml
   # cat mpi-role.yaml
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     name: mpi
     namespace: name-of-user-namespace
   rules:
   - apiGroups:
     - security.openshift.io
     resources:
     - securitycontextconstraints
     verbs:
     - use
     resourceNames:
     - scc-for-mpi
   ```

4. Create a role binding in the user's namespace.

   We create a role binding with the name `mpi` in the user's namespace that connects the service account MPI with the role MPI:

   ```
   # apply -f mpi-rolebinding.yaml
   # cat mpi-rolebinding.yaml
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: mpi
     namespace: name-of-user-namespace
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: Role
     name: mpi
     namespace: name-of-user-namespace
   subjects:
   - kind: ServiceAccount
     name: mpi
     namespace: name-of-user-namespace
   userNames:
   - system:serviceaccount:name-of-user-namespace:mpi
   ```

A regular user in this namespace (also called *project* in Red Hat OpenShift) can now run a pod under the created service account `mpi` and fully use RDMA resources by adding the service account and the `IPC_LOCK` capability to the pod's spec section in the YAML:

```
spec:
  serviceAccount: mpi
  serviceAccountName: mpi
  containers:
  - name: your-container-name:tag
  image: your-container-image:tag
  securityContext:
    capabilities:
    add: [ "IPC_LOCK" ]
```

This service account is bound to a *namespace*. The `mpi` service account, role, and role binding must be applied to every namespace that must run jobs with the `IPC_LOCK` capability and then added to the SCC.

Note: In Red Hat OpenShift 4.4.3, we also must add `LimitMEMLOCK=infinity` to the default system settings of `cri-o` under `[Service]` in `/usr/lib/systemd/system/cri-o.service` on all DGX-1 worker nodes to propagate an unlimited `ulimit` setting of `max locked memory` to the pods. Otherwise, RDMA transfers (for example, with `ib_send_bw`) in a non-privileged pod as non-root user fail with the following error:

```
sh-4.2$ ib_write_bw dgx02 -d mlx5_1
Couldn't allocate MR
failed to create mrd
Failed to create MR
Couldn't create IB resources
```

If the attribute is not present, then add it and restart `cri-o`:

```
# systemctl daemon-reload
# systemctl restart cri-o
```

This step *might* no longer be necessary in future Red Hat OpenShift 4 releases beyond Version 4.4.3.

3.4.4 MPI Operator

We use MPI Operator for distributed AI workloads to scale out across GPUs and worker nodes. To achieve this configuration, we use the MPI Operator project on GitHub, which makes it easy to run distributed AI workloads as MPI jobs on Kubernetes and Red Hat OpenShift.

The MPI Operator can be installed as follows:

```
# git clone https://github.com/kubeflow/mpi-operator.git
# cd mpi-operator/
# oc apply -f deploy/v1alpha2/mpi-operator.yaml
```

The MPI Operator allows users to define and run MPIJob resources, as shown in 4.3, “MPIJob definition” on page 33.

3.4.5 IBM Spectrum Scale CSI

IBM Spectrum Scale is a scalable software-defined storage (SDS) solution from IBM for today's enterprise AI workloads that ensures security, reliability, and high performance at scale. As a distributed parallel file system, it provides a global namespace for your data. The data can easily be made accessible to workloads running in Red Hat OpenShift or Kubernetes by using the IBM Spectrum Scale CSI plug-in. The Container Storage Interface was introduced as an alpha function in Kubernetes V1.9 and promoted to general availability in Kubernetes V1.13.

With IBM Spectrum Scale CSI, you can provision *persistent volumes* (PVs) in Kubernetes and Red Hat OpenShift with IBM Spectrum Scale as the storage back end. These PVs can be either *dynamically* provisioned at a user's request (dynamic provisioning) by using a *persistent volume claim* (PVC) and a *storage class*, or *statically* provisioned by a system admin when a specific path with existing data in IBM Spectrum Scale should directly be made available to users for their AI training or inference workloads to share direct access to huge amounts of training or inference data and models. Then, users can request these statically provisioned volumes like when using PVCs and *labels* to specify exactly which PVs (that is, which data) in which they are interested.

The *IBM Spectrum Scale CSI Operator* can deploy and manage the CSI plug-in for IBM Spectrum Scale. The IBM Spectrum Scale CSI plug-in requires a running IBM Spectrum Scale cluster with access to the IBM Spectrum Scale GUI. In this paper, we use IBM Spectrum Scale CSI V2.0.0.

You can deploy the IBM Spectrum Scale CSI plug-in by installing the IBM Spectrum Scale CSI Operator from the OperatorHub in the Red Hat OpenShift GUI, as shown in Figure 3-2.

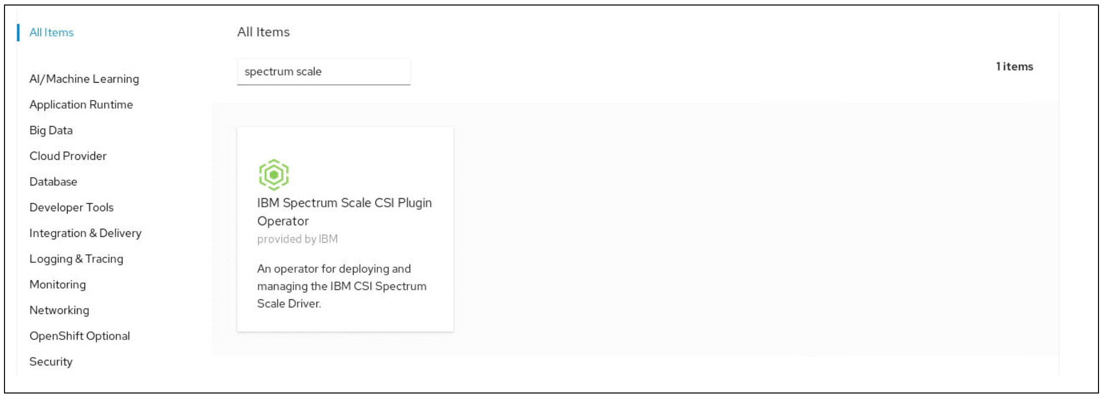


Figure 3-2 IBM Spectrum Scale CSI plug-in installed and deployed directly from Red Hat OperatorHub

To configure the IBM Spectrum Scale CSI Operator, follow the documentation that is provided in the operator. For more information and the full documentation of IBM Spectrum Scale CSI, see IBM Knowledge Center and *IBM Spectrum Scaale CSI Driver for Container Persistent Storage*, REDP-5589.

To prepare your environment, follow the instructions at Performing pre-installation tasks. In our setup, we label the Red Hat OpenShift DGX-1 worker nodes to specify where the IBM Spectrum Scale client is installed and where the IBM Spectrum Scale CSI driver should run:

```
# oc label node dgx01.ocp4.scale.ibm.com scale=true --overwrite=true
# oc label node dgx02.ocp4.scale.ibm.com scale=true --overwrite=true
```

Ensure that the **controlSetxattrImmutableSELinux** parameter is set to `yes` by running the following command (this parameter is especially important for Red Hat OpenShift):

```
# mmchconfig controlSetxattrImmutableSELinux=yes -i
```

The following IBM Spectrum Scale CSI driver configuration in the IBM Spectrum Scale CSI Operator custom resource YAML is used in this setup, which is composed of a local IBM Spectrum Scale cluster (ID 16217308676014575381 with GUI on 192.168.1.30 with the primary file system fs0 that is being used for hosting IBM Spectrum Scale configuration data) and a remote IBM ESS3000 cluster (ID 215057217487177715 with GUI on 192.168.1.52 hosting the remote file system ess3000_4M):

```
apiVersion: csi.ibm.com/v1
kind: CSIScaleOperator
metadata:
  labels:
    app.kubernetes.io/instance: ibm-spectrum-scale-csi-operator
    app.kubernetes.io/managed-by: ibm-spectrum-scale-csi-operator
    app.kubernetes.io/name: ibm-spectrum-scale-csi-operator
  name: ibm-spectrum-scale-csi
  release: ibm-spectrum-scale-csi-operator
  namespace: ibm-spectrum-scale-csi-driver
spec:
  # cluster definitions
  clusters:
    - id: '16217308676014575381'
      primary:
        primaryFs: fs0
        primaryFset: csifset
      restApi:
        - guiHost: 192.168.1.30
      secrets: csisecret-local
      secureSslMode: false
    - id: '215057217487177715'
      restApi:
        - guiHost: 192.168.1.52
      secrets: csisecret-remote
      secureSslMode: false
  scaleHostpath: /gpfs/fs0
  # node selector
  attacherNodeSelector:
    - key: scale
      value: "true"
  provisionerNodeSelector:
    - key: scale
      value: "true"
  pluginNodeSelector:
    - key: scale
      value: "true"
status: {}
```

For more information about the IBM Spectrum Scale CSI Operator custom resource YAML, see IBM Spectrum Scale Container Storage Interface driver configurations.

4

Preparation and functional testing

This chapter provides more information about how to prepare the environment, that is, configuring persistent volumes (PVs) and defining YAML for running Message Passing Interface (MPI) jobs. It also provides baseline tests to ensure proper network performance and multi-node and multi-GPU communications tests with the NVIDIA Collective Communications Library (NCCL). The chapter concludes with initial GPU scaling tests running the ResNet-50 benchmark on synthetic data.

4.1 Testing remote direct memory access through an InfiniBand network

If you have not done so during the installation (see Chapter 3, “Installation” on page 9), you should check that remote direct memory access (RDMA) over InfiniBand is working properly. A quick test to access resources over an InfiniBand connection and to verify link throughput can be done as shown in the following steps:

1. Start `ib_write_bw` in listening mode on one DGX-1 system. In our example, we use *dgx02*.

```
[root@dgx02 ~]# ib_write_bw
************************************
* Waiting for client to connect... *
************************************
```

2. Start the following test pod by using `nodeName: dgx01.ocp4.scale.ibm.com` to schedule the pod on the other DGX-1 system:

```
# oc apply -f nv-rdma-batch-job-run.yaml
# cat nv-rdma-batch-job-run.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: mofed-test-job
spec:
  template:
    spec:
      restartPolicy: OnFailure
      serviceAccount: mpi
      serviceAccountName: mpi
      nodeName: dgx01.ocp4.scale.ibm.com
      containers:
      - image: mellanox/centos_7_4_mofed_4_2_1_2_0_0_60
        name: mofed-test-ctr
        resources:
          limits:
            rdma/shared_ib0: 1
            rdma/shared_ib1: 1
            rdma/shared_ib2: 1
            rdma/shared_ib3: 1
        securityContext:
          capabilities:
            add: [ "IPC_LOCK" ]
        command: ["/bin/sh","-c"]
        args: ["whoami; ls -l /dev/infiniband /sys/class/net; ibstatus; set -x; for i in
0 1 2 3; do ibhosts -C mlx5_$i; done; ib_write_bw dgx02 -d mlx5_1"]
```

Wait for the job to complete and look at the job's log:

```
# oc logs job.batch/mofed-test-job
1075480000
/dev/infiniband:
total 0
crw-rw-rw-. 1 root root 231,  64 Jul  6 15:52 issm0
crw-rw-rw-. 1 root root 231,  65 Jul  6 15:52 issm1
crw-rw-rw-. 1 root root 231,  66 Jul  6 15:52 issm2
crw-rw-rw-. 1 root root 231,  67 Jul  6 15:52 issm3
crw-rw-rw-. 1 root root  10,  57 Jul  6 15:52 rdma_cm
crw-rw-rw-. 1 root root 231, 224 Jul  6 15:52 ucm0
crw-rw-rw-. 1 root root 231, 225 Jul  6 15:52 ucm1
```

```
crw-rw-rw-. 1 root root 231, 226 Jul  6 15:52 ucm2
crw-rw-rw-. 1 root root 231, 227 Jul  6 15:52 ucm3
crw-rw-rw-. 1 root root 231,   0 Jul  6 15:52 umad0
crw-rw-rw-. 1 root root 231,   1 Jul  6 15:52 umad1
crw-rw-rw-. 1 root root 231,   2 Jul  6 15:52 umad2
crw-rw-rw-. 1 root root 231,   3 Jul  6 15:52 umad3
crw-rw-rw-. 1 root root 231, 192 Jul  6 15:52 uverbs0
crw-rw-rw-. 1 root root 231, 193 Jul  6 15:52 uverbs1
crw-rw-rw-. 1 root root 231, 194 Jul  6 15:52 uverbs2
crw-rw-rw-. 1 root root 231, 195 Jul  6 15:52 uverbs3

/sys/class/net:
total 0
lrwxrwxrwx. 1 root root 0 Jul  6 15:52 eth0 -> ../../devices/virtual/net/eth0
lrwxrwxrwx. 1 root root 0 Jul  6 15:52 lo -> ../../devices/virtual/net/lo
Infiniband device 'mlx5_0' port 1 status:
        default gid:     fe80:0000:0000:0000:ec0d:9a03:0044:d2d0
        base lid:        0xb
        sm lid:          0x9
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            100 Gb/sec (4X EDR)
        link_layer:      InfiniBand

Infiniband device 'mlx5_1' port 1 status:
        default gid:     fe80:0000:0000:0000:ec0d:9a03:0044:d608
        base lid:        0xd
        sm lid:          0x9
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            100 Gb/sec (4X EDR)
        link_layer:      InfiniBand

Infiniband device 'mlx5_2' port 1 status:
        default gid:     fe80:0000:0000:0000:ec0d:9a03:006e:fef2
        base lid:        0xc
        sm lid:          0x9
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            100 Gb/sec (4X EDR)
        link_layer:      InfiniBand

Infiniband device 'mlx5_3' port 1 status:
        default gid:     fe80:0000:0000:0000:ec0d:9a03:0044:bda4
        base lid:        0xe
        sm lid:          0x9
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            100 Gb/sec (4X EDR)
        link_layer:      InfiniBand

+ for i in 0 1 2 3
+ ibhosts -C mlx5_0
Ca      : 0x248a0703001f0426 ports 1 "dgx02 HCA-3"
Ca      : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca      : 0x248a0703001f0786 ports 1 "dgx02 HCA-4"
Ca      : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca      : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca      : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca      : 0x248a0703001ef3a2 ports 1 "dgx02 HCA-1"
Ca      : 0xec0d9a030044d608 ports 1 "dgx01 HCA-2"
Ca      : 0x98039b03005cd071 ports 1 "fscc-fab3-3-a HCA-3"
Ca      : 0x98039b03005cbe79 ports 1 "fscc-fab3-3-b HCA-5"
Ca      : 0x98039b03005cbe78 ports 1 "fscc-fab3-3-b HCA-4"
Ca      : 0x98039b03005cbdd0 ports 1 "fscc-fab3-3-b HCA-2"
Ca      : 0x98039b03005cd081 ports 1 "fscc-fab3-3-a HCA-5"
Ca      : 0x98039b03005cd070 ports 1 "fscc-fab3-3-a HCA-2"
Ca      : 0x98039b03005cd080 ports 1 "fscc-fab3-3-a HCA-4"
Ca      : 0x98039b03005cbdd1 ports 1 "fscc-fab3-3-b HCA-3"
Ca      : 0xec0d9a030044d2d0 ports 1 "dgx01 HCA-1"
+ for i in 0 1 2 3
+ ibhosts -C mlx5_1
Ca      : 0x248a0703001f0426 ports 1 "dgx02 HCA-3"
Ca      : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca      : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca      : 0x248a0703001f0786 ports 1 "dgx02 HCA-4"
Ca      : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca      : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca      : 0x248a0703001ef3a2 ports 1 "dgx02 HCA-1"
Ca      : 0xec0d9a030044d2d0 ports 1 "dgx01 HCA-1"
Ca      : 0x98039b03005cd071 ports 1 "fscc-fab3-3-a HCA-3"
Ca      : 0x98039b03005cbe79 ports 1 "fscc-fab3-3-b HCA-5"
Ca      : 0x98039b03005cbe78 ports 1 "fscc-fab3-3-b HCA-4"
Ca      : 0x98039b03005cbdd0 ports 1 "fscc-fab3-3-b HCA-2"
Ca      : 0x98039b03005cd081 ports 1 "fscc-fab3-3-a HCA-5"
Ca      : 0x98039b03005cd070 ports 1 "fscc-fab3-3-a HCA-2"
Ca      : 0x98039b03005cd080 ports 1 "fscc-fab3-3-a HCA-4"
Ca      : 0x98039b03005cbdd1 ports 1 "fscc-fab3-3-b HCA-3"
Ca      : 0xec0d9a030044d608 ports 1 "dgx01 HCA-2"
+ for i in 0 1 2 3
+ ibhosts -C mlx5_2
Ca      : 0x248a0703001f0426 ports 1 "dgx02 HCA-3"
Ca      : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca      : 0x248a0703001f0786 ports 1 "dgx02 HCA-4"
Ca      : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca      : 0x248a0703001ef3a2 ports 1 "dgx02 HCA-1"
Ca      : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca      : 0xec0d9a030044d608 ports 1 "dgx01 HCA-2"
Ca      : 0xec0d9a030044d2d0 ports 1 "dgx01 HCA-1"
Ca      : 0x98039b03005cd071 ports 1 "fscc-fab3-3-a HCA-3"
Ca      : 0x98039b03005cbe79 ports 1 "fscc-fab3-3-b HCA-5"
Ca      : 0x98039b03005cbe78 ports 1 "fscc-fab3-3-b HCA-4"
Ca      : 0x98039b03005cbdd0 ports 1 "fscc-fab3-3-b HCA-2"
Ca      : 0x98039b03005cd081 ports 1 "fscc-fab3-3-a HCA-5"
Ca      : 0x98039b03005cd070 ports 1 "fscc-fab3-3-a HCA-2"
Ca      : 0x98039b03005cd080 ports 1 "fscc-fab3-3-a HCA-4"
Ca      : 0x98039b03005cbdd1 ports 1 "fscc-fab3-3-b HCA-3"
Ca      : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
+ for i in 0 1 2 3
+ ibhosts -C mlx5_3
Ca      : 0x248a0703001f0426 ports 1 "dgx02 HCA-3"
Ca      : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca      : 0x248a0703001f0786 ports 1 "dgx02 HCA-4"
Ca      : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca      : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca      : 0x248a0703001ef3a2 ports 1 "dgx02 HCA-1"
Ca      : 0xec0d9a030044d608 ports 1 "dgx01 HCA-2"
Ca      : 0xec0d9a030044d2d0 ports 1 "dgx01 HCA-1"
Ca      : 0x98039b03005cd071 ports 1 "fscc-fab3-3-a HCA-3"
Ca      : 0x98039b03005cbe79 ports 1 "fscc-fab3-3-b HCA-5"
Ca      : 0x98039b03005cbe78 ports 1 "fscc-fab3-3-b HCA-4"
Ca      : 0x98039b03005cbdd0 ports 1 "fscc-fab3-3-b HCA-2"
Ca      : 0x98039b03005cd081 ports 1 "fscc-fab3-3-a HCA-5"
Ca      : 0x98039b03005cd070 ports 1 "fscc-fab3-3-a HCA-2"
Ca      : 0x98039b03005cd080 ports 1 "fscc-fab3-3-a HCA-4"
Ca      : 0x98039b03005cbdd1 ports 1 "fscc-fab3-3-b HCA-3"
Ca      : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
+ ib_write_bw dgx02 -d mlx5_1
Conflicting CPU frequency values detected: 3403.930000 != 3029.296000. CPU Frequency is
not max.
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx5_1
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 TX depth        : 128
 CQ Moderation   : 100
 Mtu             : 4096[B]
 Link type       : IB
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0x0d QPN 0x16cb PSN 0xa2ff90 RKey 0x038f5c VAddr 0x007fa65e670000
 remote address: LID 0x12 QPN 0x2251 PSN 0x914a1d RKey 0x0a82f8 VAddr 0x007f451bb10000
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      5000             11300.40            11264.11                 0.180226
---------------------------------------------------------------------------------------
```

You should see the correct information about all the InfiniBand interfaces and observe a throughput of at least 11,000 MBps for a single point-to-point InfiniBand EDR connection on a DGX-1 system.

4.2 Preparing persistent volumes with IBM Spectrum Scale Container Storage Interface

The overall amount of data that is used for model building, model evaluation, or inference in Enterprise AI environments is typically huge, and multiple users (for example, data scientists) require access to the shared data in parallel with their containerized applications running on multiple worker nodes in the Red Hat OpenShift cluster. Only a parallel file system like IBM Spectrum Scale can meet all these requirements while providing extreme scalability and ensuring security, reliability, and high performance.

IBM Spectrum Scale offers parallel access to data in a global namespace from every worker node in the cluster without needing to duplicate (copy or move) huge amounts of data that is needed for model training, model validation, or inference. With its additional features like Active File Management (AFM) to provide access to the global namespace across globally dispersed IBM Spectrum Scale clusters and also allow data access and data ingestion through SMB, NFS, and S3 Object protocols, it truly meets the expectations of a universal “data lake”.

In this section, we introduce a basic example of how to provision a PV with IBM Spectrum Scale Container Storage Interface (CSI) to provide access to existing training data in IBM Spectrum Scale for containerized applications running AI workloads.

The IBM Spectrum Scale CSI driver provides:

- *Dynamic provisioning* of new PVs for containers in a self-service manner based on storage classes (see Dynamic provisioning).
- *Static provisioning* (see Static provisioning) for exposing existing paths and data in IBM Spectrum Scale to containerized workloads in Red Hat OpenShift.

In this paper, we use *static provisioning* to share access to a directory that is named `adas`, which is in the `ess3000_4M` IBM Spectrum Scale file system on the IBM ESS3000. This directory holds all the training data, models, and training scripts. You can also use different directories and PVs for training data (for example, shared input data with read only permissions), an individual user's workspace for training scripts (individual read/write access), and models (shared output data with read/write access).

Once claimed, PVs are bound to a namespace and cannot be used across different namespaces. So, to make this directory available to users with different namespaces, the system admin must create one or more PVs (adjust the name of the PV in the following YAML example, for example, pv01, pv02, pv03, and so on, depending on how many namespaces require access to the data) referencing this specific path in IBM Spectrum Scale:

```
# oc apply -f nv-pv01.yaml
# cat nv-pv01.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "adas-data-pv01"
  labels:
    type: data
    dept: adas
spec:
  storageClassName: static
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: spectrumscale.csi.ibm.com
    volumeHandle: "16217308676014575381;099B6A7A:5EB99743;path=/gpfs/ess3000_4M/adas"
```

The following items must be specified in the `csi: volumeHandle stanza`:

- The local IBM Spectrum Scale *cluster ID* (for example, `16217308676014575381`, as shown by **`mmlscluster`**)
- The *file system UID* (for example, `099B6A7A:5EB99743`, as shown by **`mmlsfs ess3000_4M --uid`**)
- The *directory path* (for example, `/gpfs/ess3000_4M/adas`) to the directory that we want to make accessible for the containerized AI workloads

In addition, we annotate the volume with a `storageClassName` “`static`” (make sure that there is no real storage class with this name) and use *labels* like `type: data` and `dept: adas` to allow a user to bind a PV with these specific attributes to a *persistent volume claim* (PVC) referencing these labels to match.

After the PV (or a set of these volumes) is created by the system admin, a user can request this volume to be bound to their namespace by issuing a PVC by using `storageClassName: static` and a `selector` to match the labels `type: data` and `dept: adas`:

```
# oc apply -f nv-data-pvc.yaml
# cat nv-data-pvc.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: "adas-data-pvc"
spec:
  storageClassName: static
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  selector:
    matchLabels:
      type: data
      dept: adas
```

After a user successfully binds the PV to their namespace, it stays bound to the user's namespace until the user deletes the PVC.

ReadWriteMany (RWX) is an access mode for the volume that ensures that the volume can be used in multiple pods in the user's namespace in parallel and across physical worker nodes concurrently. RWX is convenient for running MPI jobs with multiple worker pods across multiple nodes that all need access to the same data.

In the pod's YAML, the PVC can be applied as follows to mount the data in IBM Spectrum Scale under `/gpfs/ess3000_4M/adas/` to the directory `/workspace` within the container:

```
spec:
 containers:
 - name: your-container-name
   image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
   volumeMounts:
   - name: a2d2-data
     mountPath: /workspace
 volumes:
 - name: a2d2-data
   persistentVolumeClaim:
     claimName: adas-data-pvc
```

4.3 MPIJob definition

We use the MPI Operator for distributed AI workloads to scale out across GPUs and worker nodes by using the MPI Operator project on GitHub, which makes it easy to run distributed AI workloads as MPI jobs on Kubernetes and Red Hat OpenShift.

With the MPI Operator installed, we can use the MPIJob resource to run a multi-GPU and multi-node AI workload with a single MPI job on a Red Hat OpenShift cluster. In this paper, we start deep neural network (DNN) training jobs by using the NVIDIA GPU Cloud (NGC) TensorFlow 2 image (`nvcr.io/nvidia/tensorflow:20.03-tf2-py3`) in the user's namespace with the following MPIJob YAML:

```
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: tf2-a2d2-16x01x02-gpu
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - name: tf2-a2d2-16x01x02-gpu
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            command:
            - mpirun
            - -np
            - "16"
            - -wdir
            - "/workspace/scripts/tf2_comparison/hvd"
            - -bind-to
            - none
            - -map-by
            - slot
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_IB_DISABLE=0
            - -x
            - NCCL_NET_GDR_LEVEL=1
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - python
            - main.py
            - --model_dir=checkpt
            - --batch_size=16
            - --exec_mode=train
            - --max_steps=16000
    Worker:
      replicas: 16
      template:
        spec:
          serviceAccount: mpi
          serviceAccountName: mpi
          containers:
          - name: tf2-a2d2-16x01x02-gpu
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            env:
            - name: IBV_DRIVERS
              value: "/usr/lib/libibverbs/libmlx5"
            securityContext:
              runAsUser: 1075481000
              runAsGroup: 1075481000
              capabilities:
                add: [ "IPC_LOCK" ]
            resources:
              limits:
                nvidia.com/gpu: 1
                rdma/shared_ib0: 1
                rdma/shared_ib1: 1
                rdma/shared_ib2: 1
                rdma/shared_ib3: 1
            volumeMounts:
            - name: a2d2-data
              mountPath: /workspace
            - name: dshm
              mountPath: /dev/shm
          volumes:
          - name: a2d2-data
            persistentVolumeClaim:
              claimName: adas-data-pvc
          - name: dshm
            emptyDir:
              medium: Memory
```

This MPI job has *one* launcher pod running the **mpirun** command on *16* worker pods with each of them running with a single GPU (`nvidia.com/gpu: 1`) and having access to all four InfiniBand ports (`rdma/shared_ibX: 1; X=0,1,2,3`) on the DGX-1 systems. The worker pods run under the previously created *service account* `mpi` in the user's namespace and use the additional `IPC_LOCK` capability of the `scc-for-mpi` Security Context Constraints (SCCs). In addition, we specify a user and group ID for the worker pods that matches the UID/GID of the data directory.

The NCCL environment variables `NCCL_IB_DISABLE=0` and `NCCL_NET_GDR_LEVEL=1` explicitly enable InfiniBand RDMA and GPUDirect RDMA (GDRDMA) support on the worker pods, which provides the best performance in our setup. These environment variables can be used to switch between the following NCCL modes:

- TCP mode:

  ```
  NCCL_IB_DISABLE=1
  NCCL_NET_GDR_LEVEL=0
  ```

- Without GDRDMA:

  ```
  NCCL_IB_DISABLE=0
  NCCL_NET_GDR_LEVEL=0
  ```

- With GDRDMA:

  ```
  NCCL_IB_DISABLE=0
  NCCL_NET_GDR_LEVEL=1
  ```

In addition to the PVC `adas-data-pvc` that mounts the `/gpfs/ess3000_4M/adas` directory of the IBM Spectrum Scale file system to the `/workspace` directory in the worker pod's NGC TensorFlow v2 container, we also mount a POSIX shared memory volume as `/dev/shm` that is backed by `emptyDir: medium: memory` into the container that provides optimized communication path options for NCCL under certain conditions.

The `/workspace` directory in each worker pod provides access to both the training data under `/workspace/dataset` and to the Python scripts at `/workspace/scripts`, where the code of the DNN training is. The checkpoints of each training run are written to the same directory that provides the Python scripts.

The **`mpirun`** command changes the active working directory in the worker pods to `/workspace/scripts/tf2_comparison/hvd` and starts 16 tasks that run the Python script **`main.py`** with options **`--model_dir=checkpt --batch_size=16 --exec_mode=train --max_steps=16000`** in each of the 16 worker pods with one GPU. We reduced the `max_steps` to 16,000 so that we can run multiple training runs in an acceptable time. The training typically runs with many more steps and last for days and not just hours.

4.4 Connectivity tests with the NVIDIA Collective Communications Library

The NCCL implements multi-GPU and multi-node collective communication primitives that are performance-optimized for NVIDIA GPUs. NCCL provides routines such as `all-gather`, `all-reduce`, `broadcast`, `reduce`, and `reduce-scatter` that are optimized to achieve high bandwidth and low latency over PCIe, NVLink, and other high-speed interconnects.

NCCL supports an arbitrary number of GPUs that is installed in a single node or across multiple nodes and can be used in either single- or multi-process (for example, MPI) applications. The code with examples is provided at GitHub.

The system admin can quickly run an initial test job to check GPU communication with the NCCL on each DGX-1 worker node (eight GPUs per DGX-1) by running a Kubernetes job as follows:

```
# oc apply -f nv-nccl-batch-job.yaml
# cat nv-nccl-batch-job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: nv-nccl
spec:
  template:
    spec:
      nodeName: dgx01.ocp4.scale.ibm.com
      containers:
      - name: nv-nccl
        image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh","-c"]
        args: ["nvidia-smi && nvidia-smi topo -m && git clone 
https://github.com/NVIDIA/nccl-tests.git && cd nccl-tests && make && 
./build/all_reduce_perf -b 8 -e 256M -f 2 -g 8"]
        resources:
          limits:
            nvidia.com/gpu: 8
        restartPolicy: Never
```

Under `nodeName`, you can specify the DGX-1 worker node on which you want to run this job. Example 4-1 shows the output of the completed job (for example, `oc logs job.batch/nv-nccl`), which provides information about the GPUs (`nvidia-smi`), the GPU topology (`nvidia-smi topo -m`), and a table with the measurement results of the NCCL `all_reduce_perf` test running on eight GPUs (`-g 8`) and scanning 8 bytes (`-b 8`) - 256 MB (`-e 256M`).

Example 4-1 GPU information, GPU topology, and NCCL measurement results for single-node batch job with 8 GPUs

```
[root@fscc-sr650-6 nccl]# oc logs job.batch/nv-nccl
Mon Jul  6 12:00:05 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   32C    P0    58W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   34C    P0    44W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:0A:00.0 Off |                    0 |
| N/A   34C    P0    42W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:0B:00.0 Off |                    0 |
| N/A   34C    P0    43W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
| N/A   33C    P0    42W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2...  On   | 00000000:86:00.0 Off |                    0 |
| N/A   36C    P0    41W / 300W |      0MiB / 16160MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Tesla V100-SXM2...  On   | 00000000:89:00.0 Off |                    0 |
| N/A   36C    P0    42W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Tesla V100-SXM2...  On   | 00000000:8A:00.0 Off |                    0 |
| N/A   34C    P0    42W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+


+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity
GPU0     X      NV1     NV1     NV2     NV2     SYS     SYS     SYS     PIX     PHB     SYS     SYS     0-19,40-59
GPU1    NV1      X      NV2     NV1     SYS     NV2     SYS     SYS     PIX     PHB     SYS     SYS     0-19,40-59
GPU2    NV1     NV2      X      NV2     SYS     SYS     NV1     SYS     PHB     PIX     SYS     SYS     0-19,40-59
GPU3    NV2     NV1     NV2      X      SYS     SYS     SYS     NV1     PHB     PIX     SYS     SYS     0-19,40-59
GPU4    NV2     SYS     SYS     SYS      X      NV1     NV1     NV2     SYS     SYS     PIX     PHB     20-39,60-79
GPU5    SYS     NV2     SYS     SYS     NV1      X      NV2     NV1     SYS     SYS     PIX     PHB     20-39,60-79
GPU6    SYS     SYS     NV1     SYS     NV1     NV2      X      NV2     SYS     SYS     PHB     PIX     20-39,60-79
GPU7    SYS     SYS     SYS     NV1     NV2     NV1     NV2      X      SYS     SYS     PHB     PIX     20-39,60-79
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS      X      PHB     SYS     SYS
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS     PHB      X      SYS     SYS
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS     SYS      X      PHB
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     SYS     PHB      X
```

```
Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
[...]
# nThread 1 nGpus 8 minBytes 8 maxBytes 268435456 step: 2(factor) warmup iters: 5 iters: 20 validation: 1
#
# Using devices
#   Rank  0 Pid      1 on nv-nccl-7f9ns device  0 [0x06] Tesla V100-SXM2-16GB
#   Rank  1 Pid      1 on nv-nccl-7f9ns device  1 [0x07] Tesla V100-SXM2-16GB
#   Rank  2 Pid      1 on nv-nccl-7f9ns device  2 [0x0a] Tesla V100-SXM2-16GB
#   Rank  3 Pid      1 on nv-nccl-7f9ns device  3 [0x0b] Tesla V100-SXM2-16GB
#   Rank  4 Pid      1 on nv-nccl-7f9ns device  4 [0x85] Tesla V100-SXM2-16GB
#   Rank  5 Pid      1 on nv-nccl-7f9ns device  5 [0x86] Tesla V100-SXM2-16GB
#   Rank  6 Pid      1 on nv-nccl-7f9ns device  6 [0x89] Tesla V100-SXM2-16GB
#   Rank  7 Pid      1 on nv-nccl-7f9ns device  7 [0x8a] Tesla V100-SXM2-16GB
#
#                                                     out-of-place                       in-place
#       size         count    type   redop     time   algbw   busbw  error     time   algbw   busbw  error
#        (B)    (elements)                     (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
           8             2   float     sum    40.92    0.00    0.00  2e-07    44.50    0.00    0.00  1e-07
          16             4   float     sum    38.08    0.00    0.00  6e-08    40.94    0.00    0.00  6e-08
          32             8   float     sum    41.30    0.00    0.00  6e-08    39.27    0.00    0.00  6e-08
          64            16   float     sum    43.86    0.00    0.00  6e-08    38.48    0.00    0.00  6e-08
         128            32   float     sum    42.10    0.00    0.01  6e-08    43.15    0.00    0.01  6e-08
         256            64   float     sum    41.70    0.01    0.01  6e-08    43.80    0.01    0.01  6e-08
         512           128   float     sum    38.83    0.01    0.02  6e-08    44.29    0.01    0.02  6e-08
        1024           256   float     sum    44.58    0.02    0.04  2e-07    45.88    0.02    0.04  2e-07
        2048           512   float     sum    39.82    0.05    0.09  2e-07    46.95    0.04    0.08  2e-07
        4096          1024   float     sum    39.32    0.10    0.18  2e-07    42.10    0.10    0.17  2e-07
        8192          2048   float     sum    41.99    0.20    0.34  2e-07    39.79    0.21    0.36  2e-07
       16384          4096   float     sum    45.90    0.36    0.62  2e-07    43.01    0.38    0.67  2e-07
       32768          8192   float     sum    40.47    0.81    1.42  2e-07    44.78    0.73    1.28  2e-07
       65536         16384   float     sum    43.01    1.52    2.67  2e-07    41.39    1.58    2.77  2e-07
      131072         32768   float     sum    51.51    2.54    4.45  2e-07    47.67    2.75    4.81  2e-07
      262144         65536   float     sum    48.17    5.44    9.52  5e-07    47.02    5.58    9.76  5e-07
      524288        131072   float     sum    59.04    8.88   15.54  5e-07    60.19    8.71   15.24  5e-07
     1048576        262144   float     sum    100.4   10.45   18.28  5e-07    99.56   10.53   18.43  5e-07
     2097152        524288   float     sum    118.9   17.64   30.88  5e-07    119.6   17.54   30.69  5e-07
     4194304       1048576   float     sum    187.8   22.33   39.08  5e-07    183.7   22.83   39.95  5e-07
     8388608       2097152   float     sum    291.0   28.83   50.45  5e-07    287.4   29.18   51.07  5e-07
    16777216       4194304   float     sum    355.2   47.23   82.66  5e-07    352.5   47.60   83.30  5e-07
    33554432       8388608   float     sum    557.2   60.22  105.38  5e-07    553.3   60.65  106.13  5e-07
    67108864      16777216   float     sum    994.8   67.46  118.05  5e-07    987.4   67.96  118.94  5e-07
   134217728      33554432   float     sum   1862.1   72.08  126.14  5e-07   1863.6   72.02  126.03  5e-07
   268435456      67108864   float     sum   3671.2   73.12  127.96  5e-07   3637.3   73.80  129.15  5e-07
```

The NCCL table provides information about NCCL communication latencies and bandwidths per scan size. The *time* is useful with small sizes to measure the constant latency that is associated with operations, and *bandwidth* is typically of interest for large scan sizes. It is important that the latency for small scan sizes is in the range of 2-digit microseconds (μs) in a setup like ours with 100 Gbps EDR InfiniBand, as shown here. Should you see latencies in the range of thousands of microseconds, then your configuration is probably not using InfiniBand but Ethernet. In this case, you must investigate your RDMA setup and InfiniBand network configurations.

We can also run the same task as a single-node MPI job with just one worker pod by using eight GPUs on a single DGX-1 worker node as follows:

```
# oc apply -f nv-nccl-mpi-job-01x08.yaml
# cat nv-nccl-mpi-job-01x08.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: nccl-test-mpi
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            name: nccl-test-mpi
            env:
            - name: IBV_DRIVERS
              value: "/usr/lib/libibverbs/libmlx5"
            command:
            - mpirun
            - -np
            - "1"
            - -bind-to
            - none
            - -map-by
            - slot
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_IB_DISABLE=0
            - -x
            - NCCL_NET_GDR_LEVEL=1
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - /workspace/other/nccl/all_reduce_perf
            - -b
            - "8"
            - -e
            - 256M
            - -f
            - "2"
            - -g
            - "8"
    Worker:
      replicas: 1
      template:
        spec:
          serviceAccount: mpi
          serviceAccountName: mpi
          containers:
          - name: nccl-test-mpi
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            securityContext:
              runAsUser: 1075481000
              runAsGroup: 1075481000
              capabilities:
                add: [ "IPC_LOCK" ]
            env:
            - name: IBV_DRIVERS
              value: "/usr/lib/libibverbs/libmlx5"
            resources:
              limits:
                nvidia.com/gpu: 8
                rdma/shared_ib0: 1
                rdma/shared_ib1: 1
                rdma/shared_ib2: 1
                rdma/shared_ib3: 1
            volumeMounts:
            - name: a2d2-data
              mountPath: /workspace
            - mountPath: /dev/shm
              name: dshm
          volumes:
          - name: a2d2-data
            persistentVolumeClaim:
              claimName: adas-data-pvc
          - name: dshm
            emptyDir:
              medium: Memory
```

The results should be like the ones that are seen from the `nv-nccl-batch-job.yaml` batch job.

We can run a *multi-node* MPI job with two pods on two DGX-1 worker nodes where each of them uses eight GPUs (so we have 16 GPUs in total) as follows:

```
# oc apply -f nv-nccl-mpi-job.yaml
# cat nv-nccl-mpi-job.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: nccl-test-mpi
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
```

```
      spec:
        containers:
        - image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
          name: nccl-test-mpi
          env:
          - name: IBV_DRIVERS
            value: "/usr/lib/libibverbs/libmlx5"
          command:
          - mpirun
          - -np
          - "2"
          - -bind-to
          - none
          - -map-by
          - slot
          - -x
          - NCCL_DEBUG=INFO
          - -x
          - NCCL_IB_DISABLE=0
          - -x
          - NCCL_NET_GDR_LEVEL=1
          - -x
          - LD_LIBRARY_PATH
          - -x
          - PATH
          - -mca
          - pml
          - ob1
          - -mca
          - btl
          - ^openib
          - /workspace/other/nccl/all_reduce_perf
          - -b
          - "8"
          - -e
          - 256M
          - -f
          - "2"
          - -g
          - "8"
  Worker:
    replicas: 2
    template:
      spec:
        serviceAccount: mpi
        serviceAccountName: mpi
        containers:
        - name: nccl-test-mpi
          image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
          imagePullPolicy: IfNotPresent
          securityContext:
            runAsUser: 1075481000
            runAsGroup: 1075481000
            capabilities:
              add: [ "IPC_LOCK" ]
          env:
          - name: IBV_DRIVERS
            value: "/usr/lib/libibverbs/libmlx5"
          resources:
            limits:
              nvidia.com/gpu: 8
              rdma/shared_ib0: 1
              rdma/shared_ib1: 1
              rdma/shared_ib2: 1
              rdma/shared_ib3: 1
          volumeMounts:
          - name: a2d2-data
            mountPath: /workspace
          - mountPath: /dev/shm
            name: dshm
        volumes:
        - name: a2d2-data
          persistentVolumeClaim:
            claimName: adas-data-pvc
        - name: dshm
          emptyDir:
            medium: Memory
```

We verify that we still see acceptable latencies for small scan sizes in the 2-digit microseconds (µs) range, as shown In Example 4-2.

Example 4-2 NCCL measurement results of a multi-node MPI job with two 8-GPU worker pods that are distributed across two DGX-1 worker nodes (16 GPUs total)

```
# Using devices
#   Rank  0 Pid     30 on nccl-test-mpi-worker-0 device  0 [0x06] Tesla V100-SXM2-16GB
#   Rank  1 Pid     30 on nccl-test-mpi-worker-0 device  1 [0x07] Tesla V100-SXM2-16GB
#   Rank  2 Pid     30 on nccl-test-mpi-worker-0 device  2 [0x0a] Tesla V100-SXM2-16GB
#   Rank  3 Pid     30 on nccl-test-mpi-worker-0 device  3 [0x0b] Tesla V100-SXM2-16GB
#   Rank  4 Pid     30 on nccl-test-mpi-worker-0 device  4 [0x85] Tesla V100-SXM2-16GB
#   Rank  5 Pid     30 on nccl-test-mpi-worker-0 device  5 [0x86] Tesla V100-SXM2-16GB
#   Rank  6 Pid     30 on nccl-test-mpi-worker-0 device  6 [0x89] Tesla V100-SXM2-16GB
#   Rank  7 Pid     30 on nccl-test-mpi-worker-0 device  7 [0x8a] Tesla V100-SXM2-16GB
#   Rank  8 Pid     30 on nccl-test-mpi-worker-1 device  0 [0x06] Tesla V100-SXM2-16GB
#   Rank  9 Pid     30 on nccl-test-mpi-worker-1 device  1 [0x07] Tesla V100-SXM2-16GB
#   Rank 10 Pid     30 on nccl-test-mpi-worker-1 device  2 [0x0a] Tesla V100-SXM2-16GB
#   Rank 11 Pid     30 on nccl-test-mpi-worker-1 device  3 [0x0b] Tesla V100-SXM2-16GB
#   Rank 12 Pid     30 on nccl-test-mpi-worker-1 device  4 [0x85] Tesla V100-SXM2-16GB
#   Rank 13 Pid     30 on nccl-test-mpi-worker-1 device  5 [0x86] Tesla V100-SXM2-16GB
#   Rank 14 Pid     30 on nccl-test-mpi-worker-1 device  6 [0x89] Tesla V100-SXM2-16GB
#   Rank 15 Pid     30 on nccl-test-mpi-worker-1 device  7 [0x8a] Tesla V100-SXM2-16GB
#
#                                                     out-of-place                       in-place
#       size     count    type   redop     time   algbw   busbw  error     time   algbw   busbw  error
#        (B)(elements)                     (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
nccl-test-mpi-worker-0:30:30 [0] NCCL INFO Launch mode Group/CGMD
           8         2   float     sum    52.72    0.00    0.00  4e-07    50.09    0.00    0.00  4e-07
          16         4   float     sum    58.60    0.00    0.00  2e-07    49.80    0.00    0.00  2e-07
          32         8   float     sum    58.46    0.00    0.00  2e-07    61.82    0.00    0.00  1e-07
          64        16   float     sum    52.95    0.00    0.00  1e-07    50.00    0.00    0.00  1e-07
         128        32   float     sum    54.79    0.00    0.00  1e-07    60.03    0.00    0.00  1e-07
         256        64   float     sum    60.63    0.00    0.01  1e-07    51.02    0.01    0.01  1e-07
         512       128   float     sum    51.87    0.01    0.02  1e-07    51.36    0.01    0.02  1e-07
        1024       256   float     sum    52.68    0.02    0.04  4e-07    51.93    0.02    0.04  4e-07
        2048         512   float     sum    62.35    0.03    0.06  4e-07    49.45    0.04    0.08  4e-07
        4096        1024   float     sum    55.25    0.07    0.14  4e-07    58.66    0.07    0.13  4e-07
        8192        2048   float     sum    63.85    0.13    0.24  5e-07    58.16    0.14    0.26  5e-07
       16384        4096   float     sum    87.96    0.19    0.35  5e-07    78.23    0.21    0.39  5e-07
       32768        8192   float     sum    104.6    0.31    0.59  5e-07    100.1    0.33    0.61  5e-07
       65536       16384   float     sum    103.7    0.63    1.19  5e-07    107.4    0.61    1.14  5e-07
      131072       32768   float     sum    110.2    1.19    2.23  5e-07    110.4    1.19    2.23  5e-07
      262144       65536   float     sum    118.5    2.21    4.15  5e-07    119.4    2.20    4.12  5e-07
      524288      131072   float     sum    123.7    4.24    7.95  5e-07    144.1    3.64    6.82  5e-07
     1048576      262144   float     sum    167.5    6.26   11.74  5e-07    159.4    6.58   12.34  5e-07
     2097152      524288   float     sum    247.7    8.47   15.88  5e-07    241.3    8.69   16.29  5e-07
     4194304     1048576   float     sum    367.9   11.40   21.37  5e-07    376.4   11.14   20.90  5e-07
     8388608     2097152   float     sum    556.2   15.08   28.28  5e-07    512.2   16.38   30.71  5e-07
    16777216     4194304   float     sum    918.7   18.26   34.24  5e-07    907.6   18.48   34.66  5e-07
    33554432     8388608   float     sum   1594.9   21.04   39.45  5e-07   1557.2   21.55   40.40  5e-07
    67108864    16777216   float     sum   2950.1   22.75   42.65  5e-07   2954.5   22.71   42.59  5e-07
   134217728    33554432   float     sum   4777.7   28.09   52.67  5e-07   4781.1   28.07   52.64  5e-07
   268435456    67108864   float     sum   9115.8   29.45   55.21  5e-07   9115.3   29.45   55.22  5e-07
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 12.3088
```

Again, should we observe latencies in the thousands of microseconds range, then your configuration is probably not using InfiniBand RDMA, and you must look into the job's log to check whether the InfiniBand connections are correctly established and used for communication queues. Watch for NCCL INFO NET/InfiniBand messages like the following one to ensure that RDMA ports (for example, `[0]mlx5_3:1/IB, …,`) are discovered and used by NCCL as available communication paths:

```
nccl-test-mpi-worker-1:30:30 [0] NCCL INFO NET/IB : Using [0]mlx5_3:1/IB
[1]mlx5_2:1/IB [2]mlx5_1:1/IB [3]mlx5_0:1/IB ; OOB eth0:10.128.7.148<0>
```

Instead of running two pods with eight GPUs, it is often more convenient to assign only one GPU to a *single pod* (the smallest compute unit that can be scheduled on Red Hat OpenShift or Kubernetes) and scale the total number of GPUs for your DNN training job by the number of worker pods that is assigned to the job.

Note: Here we assume that we have only *one* container per pod requesting GPUs to run the AI workload. In general, you can have more than one container per pod, although you would not scale your application by using multiple containers in the same pod.

This configuration offers the highest granularity for Red Hat OpenShift resource allocation when scheduling pods for DNN workload. This way, a much better resource utilization and scheduling can be achieved.

The following example schedules 16 pods across both DGX-1 systems with each pod using a single GPU (16 GPUs in total for the job):

```
# oc apply -f nv-nccl-mpi-job-16x01.yaml
# cat nv-nccl-mpi-job-16x01.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: nccl-test-mpi
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
```

```
Launcher:
  replicas: 1
  template:
    spec:
      containers:
      - image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
        name: nccl-test-mpi
        env:
        - name: IBV_DRIVERS
          value: "/usr/lib/libibverbs/libmlx5"
        command:
        - mpirun
        - -np
        - "16"
        - -bind-to
        - none
        - -map-by
        - slot
        - -x
        - NCCL_DEBUG=INFO
        - -x
        - NCCL_IB_DISABLE=0
        - -x
        - NCCL_NET_GDR_LEVEL=1
        - -x
        - LD_LIBRARY_PATH
        - -x
        - PATH
        - -mca
        - pml
        - ob1
        - -mca
        - btl
        - ^openib
        - /workspace/other/nccl/all_reduce_perf
        - -b
        - "8"
        - -e
        - 256M
        - -f
        - "2"
        - -g
        - "1"
Worker:
  replicas: 16
  template:
    spec:
      serviceAccount: mpi
      serviceAccountName: mpi
      containers:
      - name: nccl-test-mpi
        image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
        imagePullPolicy: IfNotPresent
        securityContext:
          runAsUser: 1075481000
          runAsGroup: 1075481000
          capabilities:
            add: [ "IPC_LOCK" ]
        env:
        - name: IBV_DRIVERS
          value: "/usr/lib/libibverbs/libmlx5"
        resources:
          limits:
            nvidia.com/gpu: 1
            rdma/shared_ib0: 1
            rdma/shared_ib1: 1
            rdma/shared_ib2: 1
            rdma/shared_ib3: 1
        volumeMounts:
        - name: a2d2-data
          mountPath: /workspace
        - mountPath: /dev/shm
          name: dshm
      volumes:
      - name: a2d2-data
        persistentVolumeClaim:
          claimName: adas-data-pvc
      - name: dshm
        emptyDir:
          medium: Memory
```

The log of the successfully completed job shows NCCL messages like the following ones:

```
nccl-test-mpi-worker-2:30:30 [0] NCCL INFO NET/IB : Using [0]mlx5_3:1/IB
[1]mlx5_2:1/IB [2]mlx5_1:1/IB [3]mlx5_0:1/IB ; OOB eth0:10.131.4.63<0>

nccl-test-mpi-worker-11:30:39 [0] NCCL INFO Ring 00 : 10[89000] -> 11[89000]
[receive] via NET/IB/0/GDRDMA
```

These messages indicate that InfiniBand connections (for example, `mlx5_3`, `mlx5_2`, `mlx5_1`, and `mlx5_0`) are successfully used for NCCL communication paths and that GDRDMA is enabled.

Example 4-3 shows the NCCL results of this configuration, which show slightly higher latencies due to the increased parallelization costs but allow for a more granular and robust way to schedule pods and DNN workloads across all the available GPU resources in a Red Hat OpenShift cluster.

Example 4-3 NCCL measurement results of a multi-node MPI job with sixteen 1-GPU worker pods that are evenly distributed across two DGX-1 worker nodes (16 GPUs total)

```
# Using devices
#   Rank  0 Pid     30 on nccl-test-mpi-worker-0 device  0 [0x85] Tesla V100-SXM2-16GB
#   Rank  1 Pid     30 on nccl-test-mpi-worker-1 device  0 [0x0b] Tesla V100-SXM2-16GB
#   Rank  2 Pid     30 on nccl-test-mpi-worker-2 device  0 [0x07] Tesla V100-SXM2-16GB
#   Rank  3 Pid     30 on nccl-test-mpi-worker-3 device  0 [0x0a] Tesla V100-SXM2-16GB
#   Rank  4 Pid     30 on nccl-test-mpi-worker-4 device  0 [0x06] Tesla V100-SXM2-16GB
#   Rank  5 Pid     30 on nccl-test-mpi-worker-5 device  0 [0x86] Tesla V100-SXM2-16GB
#   Rank  6 Pid     31 on nccl-test-mpi-worker-6 device  0 [0x0a] Tesla V100-SXM2-16GB
#   Rank  7 Pid     30 on nccl-test-mpi-worker-7 device  0 [0x06] Tesla V100-SXM2-16GB
#   Rank  8 Pid     30 on nccl-test-mpi-worker-8 device  0 [0x8a] Tesla V100-SXM2-16GB
#   Rank  9 Pid     30 on nccl-test-mpi-worker-9 device  0 [0x8a] Tesla V100-SXM2-16GB
#   Rank 10 Pid     30 on nccl-test-mpi-worker-10 device  0 [0x89] Tesla V100-SXM2-16GB
#   Rank 11 Pid     30 on nccl-test-mpi-worker-11 device  0 [0x89] Tesla V100-SXM2-16GB
#   Rank 12 Pid     30 on nccl-test-mpi-worker-12 device  0 [0x0b] Tesla V100-SXM2-16GB
#   Rank 13 Pid     30 on nccl-test-mpi-worker-13 device  0 [0x85] Tesla V100-SXM2-16GB
#   Rank 14 Pid     30 on nccl-test-mpi-worker-14 device  0 [0x86] Tesla V100-SXM2-16GB
#   Rank 15 Pid     30 on nccl-test-mpi-worker-15 device  0 [0x07] Tesla V100-SXM2-16GB
#
#                                                    out-of-place                       in-place
#       size     count    type   redop     time   algbw   busbw  error     time   algbw   busbw  error
#        (B)(elements)                     (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
           8         2   float     sum    78.92    0.00    0.00  2e-07    72.73    0.00    0.00  1e-07
          16         4   float     sum    72.27    0.00    0.00  1e-07    77.27    0.00    0.00  1e-07
          32         8   float     sum    73.74    0.00    0.00  1e-07    72.11    0.00    0.00  1e-07
          64        16   float     sum    82.74    0.00    0.00  1e-07    81.37    0.00    0.00  6e-08
         128        32   float     sum    83.59    0.00    0.00  6e-08    79.80    0.00    0.00  6e-08
         256        64   float     sum    78.04    0.00    0.01  6e-08    78.35    0.00    0.01  6e-08
         512       128   float     sum    78.49    0.01    0.01  6e-08    83.42    0.01    0.01  6e-08
        1024       256   float     sum    88.29    0.01    0.02  2e-07    91.61    0.01    0.02  2e-07
        2048       512   float     sum    107.5    0.02    0.04  5e-07    109.6    0.02    0.04  5e-07
        4096      1024   float     sum    121.4    0.03    0.06  5e-07    116.4    0.04    0.07  5e-07
        8192      2048   float     sum    122.8    0.07    0.13  5e-07    121.9    0.07    0.13  5e-07
       16384      4096   float     sum    143.3    0.11    0.21  5e-07    135.9    0.12    0.23  5e-07
       32768      8192   float     sum    152.4    0.22    0.40  5e-07    158.5    0.21    0.39  5e-07
       65536     16384   float     sum    193.0    0.34    0.64  5e-07    189.7    0.35    0.65  5e-07
      131072     32768   float     sum    233.7    0.56    1.05  5e-07    220.0    0.60    1.12  5e-07
      262144     65536   float     sum    313.7    0.84    1.57  5e-07    326.3    0.80    1.51  5e-07
      524288    131072   float     sum    594.7    0.88    1.65  5e-07    598.3    0.88    1.64  5e-07
     1048576    262144   float     sum    754.3    1.39    2.61  5e-07    769.7    1.36    2.55  5e-07
     2097152    524288   float     sum   1068.4    1.96    3.68  5e-07   1072.8    1.95    3.67  5e-07
     4194304   1048576   float     sum   1748.3    2.40    4.50  5e-07   1742.8    2.41    4.51  5e-07
     8388608   2097152   float     sum   2966.9    2.83    5.30  5e-07   2955.6    2.84    5.32  5e-07
    16777216   4194304   float     sum   5637.3    2.98    5.58  5e-07   5617.9    2.99    5.60  5e-07
    33554432   8388608   float     sum    11270    2.98    5.58  5e-07    11231    2.99    5.60  5e-07
    67108864  16777216   float     sum    22060    3.04    5.70  5e-07    22009    3.05    5.72  5e-07
   134217728  33554432   float     sum    44146    3.04    5.70  5e-07    43998    3.05    5.72  5e-07
   268435456  67108864   float     sum    88376    3.04    5.70  5e-07    88034    3.05    5.72  5e-07
```

4.5 Multi-GPU and multi-Node GPU scaling with TensorFlow ResNet-50 benchmark

We run a test to verify how a DNN training workload scales out with more GPUs in an orchestrated fashion by using a single MPI job in Red Hat OpenShift. Each DGX-1 system has eight NVIDIA V100 SXM2 GPUs with 16-GB GPU memory.

AI workloads might scale equally well with the number of GPUs that is used in parallel. This configuration depends heavily on the infrastructure, the model, and the implementation. The TensorFlow ResNet-50 benchmark with synthetic data (referred to as ResNet-50 benchmark) typically scales well with the number of GPUs. Therefore, we use the ResNet-50 benchmark with synthetic data from the TensorFlow benchmarks GitHub repository to explore the GPU scaling behavior of MPI jobs in Red Hat OpenShift 4. We clone the repository into our `adas` directory in IBM Spectrum Scale that we made available in Red Hat OpenShift through a statically provisioned PV that can be mounted into each MPI worker pod, as described in 4.2, “Preparing persistent volumes with IBM Spectrum Scale Container Storage Interface” on page 31.

We schedule MPI jobs with multi-GPU pods that use 1, 2, 4, and 8 GPUs in a single worker pod on a single DGX-1 system, and then 16 GPUs in two 8-GPU worker pods on two DGX-1 systems to scale beyond physical node boundaries (horizontal scaling).

The MPI Job YAML looks as follows for the 02x08 GPU case (16 GPUs total):

```
# cat nv-tf2-job-resnet50-02x08x02.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: tf2-resnet50-02x08x02
spec:
  slotsPerWorker: 8
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - name: tf2-resnet50-02x08x02
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            command:
            - mpirun
            - -np
            - "16"
            - -bind-to
            - none
            - -map-by
            - slot
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_IB_DISABLE=0
            - -x
            - NCCL_NET_GDR_LEVEL=1
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - python
            -
/workspace/other/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
            - --model=resnet50
            - --batch_size=64
            - --use_fp16=true
            - --variable_update=horovod
    Worker:
      replicas: 2
```

```
template:
  spec:
    serviceAccount: mpi
    serviceAccountName: mpi
    containers:
    - name: tf2-resnet50-02x08x02
      image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
      imagePullPolicy: IfNotPresent
      securityContext:
        runAsUser: 1075481000
        runAsGroup: 1075481000
        capabilities:
          add: [ "IPC_LOCK" ]
      env:
      - name: IBV_DRIVERS
        value: "/usr/lib/libibverbs/libmlx5"
      resources:
        limits:
          nvidia.com/gpu: 8
          rdma/shared_ib0: 1
          rdma/shared_ib1: 1
          rdma/shared_ib2: 1
          rdma/shared_ib3: 1
      volumeMounts:
      - name: a2d2-data
        mountPath: /workspace
      - name: dshm
        mountPath: /dev/shm
    volumes:
    - name: a2d2-data
      persistentVolumeClaim:
        claimName: adas-data-pvc
    - name: dshm
      emptyDir:
        medium: Memory
```

Figure 4-1 on page 49 shows that the number of images per second increases (with some scaling costs) when we scale multi-GPU pods from 1 to 8 GPUs with 7096 images per second on a single DGX-1 node and almost doubles when seamlessly scaling out to 16 GPUs on two DGX-1 systems with 13,745 images per second.

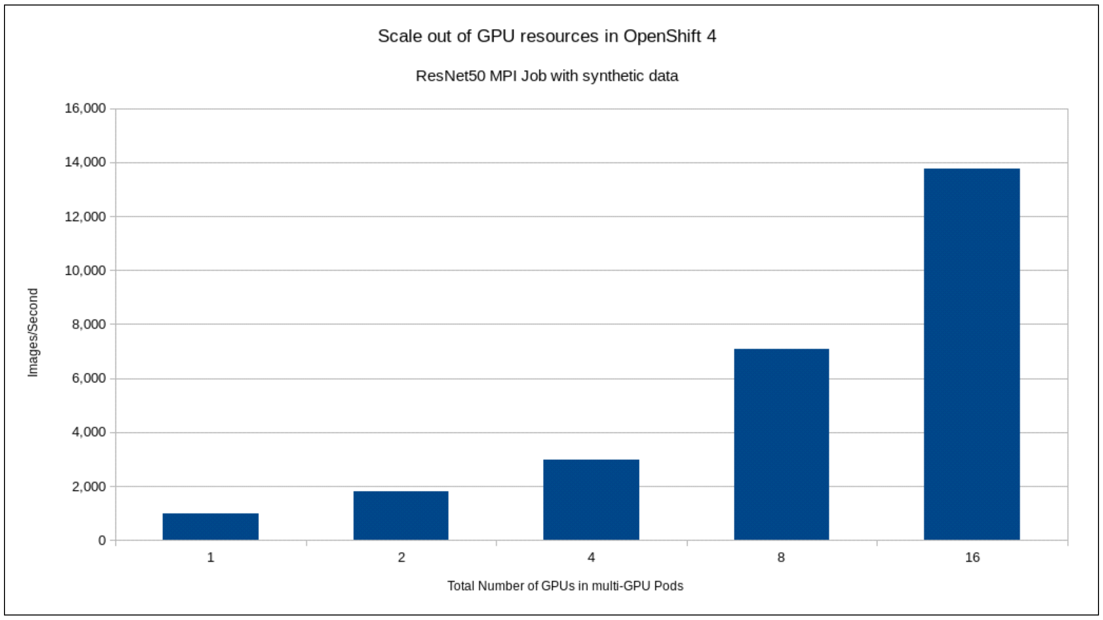


Figure 4-1   ResNet-50 results when scaling multi-GPU worker pods that use 1 - 8 GPUs per pod on a single system and 16 GPUs on two DGX-1 systems

Each column in Figure 4-1 (and in Figure 4-2 on page 51) represents the median value of five measurements for the configuration.

Not every AI workload can make efficient use of multiple GPUs in a single pod. Furthermore, orchestrating jobs with multi-GPU pods requesting 4 or 8 GPUs in a single worker pod might also lead to less optimal scheduling and a suboptimal resource utilization across all the resources in large clusters because such a pod is scheduled and started only when 4 or 8 GPUs as requested are available on a single worker node.

Scheduling jobs with only single-GPU pods allows a more granular scheduling policy. A single-GPU worker pod can be scheduled on any worker node in the cluster that has a GPU resource available and can contribute to the overall AI workload of a larger multi-GPU and multi-node MPI job. This configuration leads to a more balanced and optimal overall resource utilization in a cluster. In addition, single-GPU worker pods might also tend to provide a more robust scheduling and runtime behavior.

In the second test case, we schedule MPI jobs with single-GPU pods that use 1, 2, 4, an d8 GPUs and the same number of single-GPU worker pods on a single DGX-1 system (enforced by using `nodeName` in the worker pod YAML template). Finally, 16 GPUs in 16 worker pods on two DGX-1 systems scale beyond physical node boundaries (horizontal scaling).

The MPI Job YAML looks as follows for the 16x01 GPU case (16 GPUs total):

```
# cat nv-tf2-job-resnet50-16x01x02.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: tf2-resnet50-16x01x02
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - name: tf2-resnet50-16x01x02
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            command:
            - mpirun
            - -np
            - "16"
            - -bind-to
            - none
            - -map-by
            - slot
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_IB_DISABLE=0
            - -x
            - NCCL_NET_GDR_LEVEL=1
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - python
            - /workspace/other/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
            - --model=resnet50
            - --batch_size=64
            - --use_fp16=true
            - --variable_update=horovod
    Worker:
      replicas: 16
      template:
        spec:
          serviceAccount: mpi
          serviceAccountName: mpi
          containers:
          - name: tf2-resnet50-16x01x02
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            securityContext:
              runAsUser: 1075481000
              runAsGroup: 1075481000
              capabilities:
                add: [ "IPC_LOCK" ]
            env:
            - name: IBV_DRIVERS
              value: "/usr/lib/libibverbs/libmlx5"
            resources:
              limits:
                nvidia.com/gpu: 1
                rdma/shared_ib0: 1
                rdma/shared_ib1: 1
                rdma/shared_ib2: 1
                rdma/shared_ib3: 1
            volumeMounts:
            - name: a2d2-data
              mountPath: /workspace
            - name: dshm
              mountPath: /dev/shm
          volumes:
          - name: a2d2-data
            persistentVolumeClaim:
              claimName: adas-data-pvc
          - name: dshm
            emptyDir:
              medium: Memory
```

Figure 4-2 shows that the number of images per second increases steadily to 5918 images per second when we scale with single-GPU worker pods from 1 to 8 GPUs on a single DGX-1 node, and it almost doubles when seamlessly scaling out to 16 GPUs on two DGX-1 systems with 11,632 images per second.

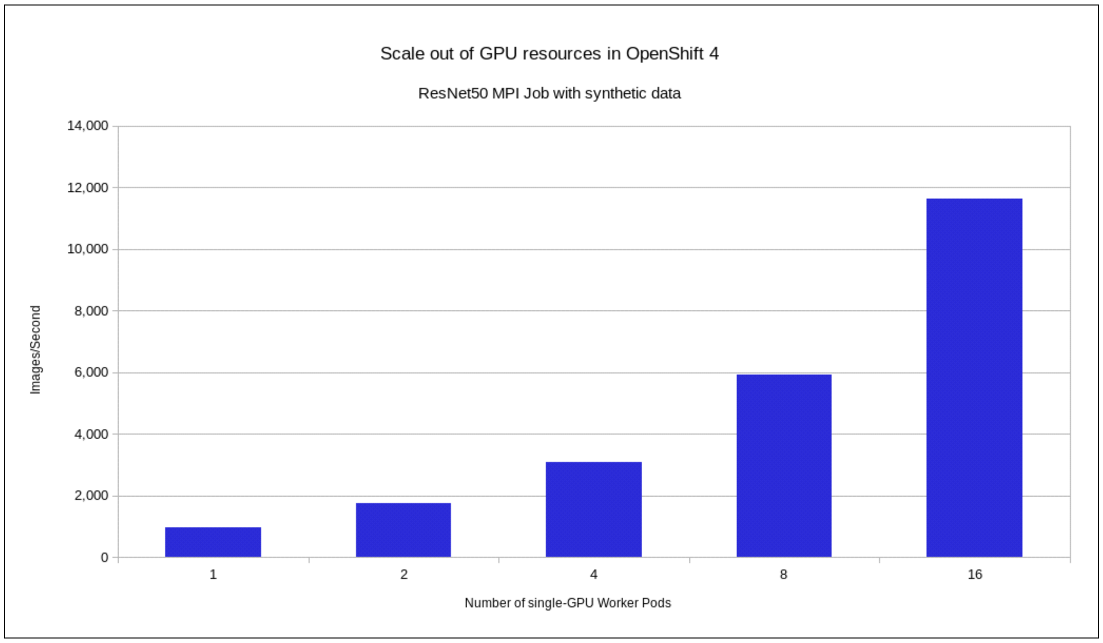


Figure 4-2 ResNet-50 results when scaling single-GPU worker pods that use 1 - 8 GPUs on a single system and 16 GPUs on two DGX-1 systems

Especially with eight single-GPU pods on a one DGX-1 system (the 8-GPU and 16-GPU cases) when going for the limit of a single DGX-1 system with eight GPUs, we experience slightly higher costs in overall performance by using single-GPU pods compared to multi-GPU pods. However, the 1, 2, and 4GPU cases show identical performance results for single-GPU and multi-GPU pods.

These results illustrate that we can efficiently and seamlessly scale out the GPU resources beyond physical node boundaries in Red Hat OpenShift for an AI workload by using MPI jobs. Single-GPU pods provide a finer granularity when scheduling jobs and allocating resources in large clusters, which leads to a better overall resource utilization across all compute nodes and can justify the slightly higher performance costs that come with single-GPU worker pods at full utilization compared to multi-GPU worker pods.

5

Deep neural network training on the Audi Autonomous Driving Dataset semantic segmentation data set

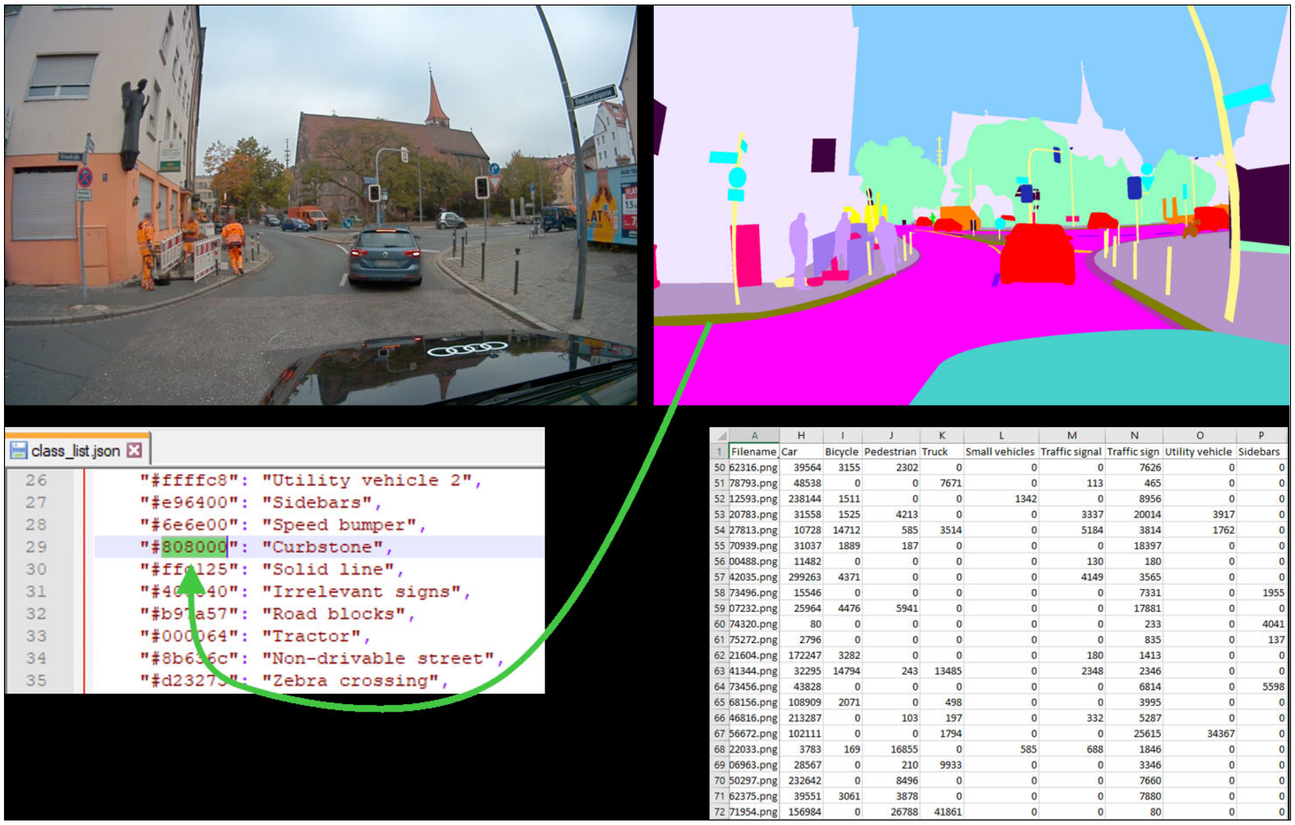
This chapter provides details about multi-GPU and multi-node training that uses the Audi Autonomous Driving Dataset (A2D2).

5.1 Description of the A2D2

Audi recently published the A2D2, which can be used to support academic institutions and commercial startups working on autonomous driving research (for more A2D2 license details, see “Related publications” on page 65). The data set consists of recorded images and labels like bounding boxes, semantic segmentation, instance segmentation, and data that is extracted from the automotive bus. The sensor suite consists of six cameras and five LIDAR units, which provide full 360 coverage. The recorded data is time-synchronized and mutually registered. There are 41,277 frames with semantic segmentation and point cloud labels. Out of that are 12,497 frames that have 3D bounding box annotations for objects within the field of view of the front camera.

The semantic segmentation data set features 38 categories. Each pixel in an image has a label that describes the type of object it represents, such pedestrian, car, or vegetation.

Figure 5-1 shows an example of a real picture that is compared to the segmentation picture.



```
    "#ffffc8": "Utility vehicle 2",
    "#e96400": "Sidebars",
    "#6e6e00": "Speed bumper",
    "#808000": "Curbstone",
    "#ffc125": "Solid line",
    "#400040": "Irrelevant signs",
    "#b97a57": "Road blocks",
    "#000064": "Tractor",
    "#8b636c": "Non-drivable street",
    "#d23273": "Zebra crossing",
```

	A	H	I	J	K	L	M	N	O	P
1	Filename	Car	Bicycle	Pedestrian	Truck	Small vehicles	Traffic signal	Traffic sign	Utility vehicle	Sidebars
50	62316.png	39564	3155	2302	0	0	0	7626	0	0
51	78793.png	48538	0	0	7671	0	113	465	0	0
52	12593.png	238144	1511	0	0	1342	0	8956	0	0
53	20783.png	31558	1525	4213	0	0	3337	20014	3917	0
54	27813.png	10728	14712	585	3514	0	5184	3814	1762	0
55	70939.png	31037	1889	187	0	0	0	18397	0	0
56	00488.png	11482	0	0	0	0	130	180	0	0
57	42035.png	299263	4371	0	0	0	4149	3565	0	0
58	73496.png	15546	0	0	0	0	0	7331	0	1955
59	07232.png	25964	4476	5941	0	0	0	17881	0	0
60	74320.png	80	0	0	0	0	0	233	0	4041
61	75272.png	2796	0	0	0	0	0	835	0	137
62	21604.png	172247	3282	0	0	0	180	1413	0	0
63	41344.png	32295	14794	243	13485	0	2348	2346	0	0
64	73456.png	43828	0	0	0	0	0	6814	0	5598
65	68156.png	108909	2071	0	498	0	0	3995	0	0
66	46816.png	213287	0	103	197	0	332	5287	0	0
67	56672.png	102111	0	0	1794	0	0	25615	34367	0
68	22033.png	3783	169	16855	0	585	688	1846	0	0
69	06963.png	28567	0	210	9933	0	0	3346	0	0
70	50297.png	232642	0	8496	0	0	0	7660	0	0
71	62375.png	39551	3061	3878	0	0	0	7880	0	0
72	71954.png	156984	0	26788	41861	0	0	80	0	0

Figure 5-1 Real picture that is compared to the segmentation picture (showing the color to label match)

5.2 Multi-node GPU scaling results for deep neural network training jobs

In this section, we perform multi-GPU and multi-node training with the A2D2 and train a segmentation network. Not all classes are considered, and the training data set runs on a reduced set of classes. The frames and labels are from the front camera. A random subset of the original set is used with ~27,300 image and label pairs, which is roughly 1/10th of a representative data set that usually contains more than 300,000 selected frames per network and task. The subset that is used for the training in this paper has a total size of 92 GB.

The training Message Passing Interface (MPI) job uses 16 GPUs on both DGX-1 systems and runs as follows:

```
# oc apply -f nv-tf2-job-a2d2.yaml
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: tf2-a2d2-16x01x02-gpu
spec:
  slotsPerWorker: 1
  cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - name: tf2-a2d2-16x01x02-gpu
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            command:
            - mpirun
            - -np
            - "16"
            - -wdir
            - "/workspace/scripts/tf2_comparison/hvd"
            - -bind-to
            - none
            - -map-by
            - slot
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_IB_DISABLE=0
            - -x
            - NCCL_NET_GDR_LEVEL=1
            - -x
            - LD_LIBRARY_PATH
            - -x
            - PATH
            - -mca
            - pml
            - ob1
            - -mca
            - btl
            - ^openib
            - python
            - main.py
            - --model_dir=checkpt
            - --batch_size=16
            - --exec_mode=train
            - --max_steps=16000
    Worker:
      replicas: 16
      template:
        spec:
          serviceAccount: mpi
          serviceAccountName: mpi
          containers:
          - name: tf2-a2d2-16x01x02-gpu
            image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
            imagePullPolicy: IfNotPresent
            env:
            - name: IBV_DRIVERS
              value: "/usr/lib/libibverbs/libmlx5"
            securityContext:
              runAsUser: 1075481000
              runAsGroup: 1075481000
              capabilities:
                add: [ "IPC_LOCK" ]
            resources:
              limits:
                nvidia.com/gpu: 1
                rdma/shared_ib0: 1
                rdma/shared_ib1: 1
                rdma/shared_ib2: 1
                rdma/shared_ib3: 1
            volumeMounts:
            - name: a2d2-data
              mountPath: /workspace
            - name: dshm
              mountPath: /dev/shm
          volumes:
          - name: a2d2-data
            persistentVolumeClaim:
              claimName: adas-data-pvc
          - name: dshm
            emptyDir:
              medium: Memory
```

The job is based on the MPI job YAML that is described in 4.3, “MPIJob definition” on page 33. The job schedules 16 single-GPU pods for the training and uses the NVIDIA GPU Cloud (NGC) TensorFlow v2 image (`nvcr.io/nvidia/tensorflow:20.03-tf2-py3`).

The training data and TensorFlow Python scripts are stored in the IBM Spectrum Scale file system at the following location:

- Training data: `/gpfs/ess3000_4M/adas/dataset/a2d2_8channel`
- Python scripts: `/gpfs/ess3000_4M/adas/scripts/tf2_comparison/hvd/`

Using IBM Spectrum Scale Container Storage Interface (CSI) with a statically provisioned volume that was created in 4.2, “Preparing persistent volumes with IBM Spectrum Scale Container Storage Interface” on page 31, the directory `/gpfs/ess3000_4M/adas` is mounted locally under `/workspace` in each TensorFlow container in all the worker pods of the MPI job. With IBM Spectrum Scale as the distributed parallel file system, all worker pods share read/write access to the same data in IBM Spectrum Scale across physical node boundaries.

We compare a multi-GPU training run that uses a single MPI job with eight single-GPU pods on a single DGX-1 node (setting `Launcher: np=8` and `Worker: replicas=8` in the YAML) with a training job that uses 16 single-GPU pods on two DGX-1 nodes that use Red Hat OpenShift 4.4.3 as a container orchestration and scheduling platform.

Figure 5-2 shows the median value of the elapsed time of five training runs for each configuration.

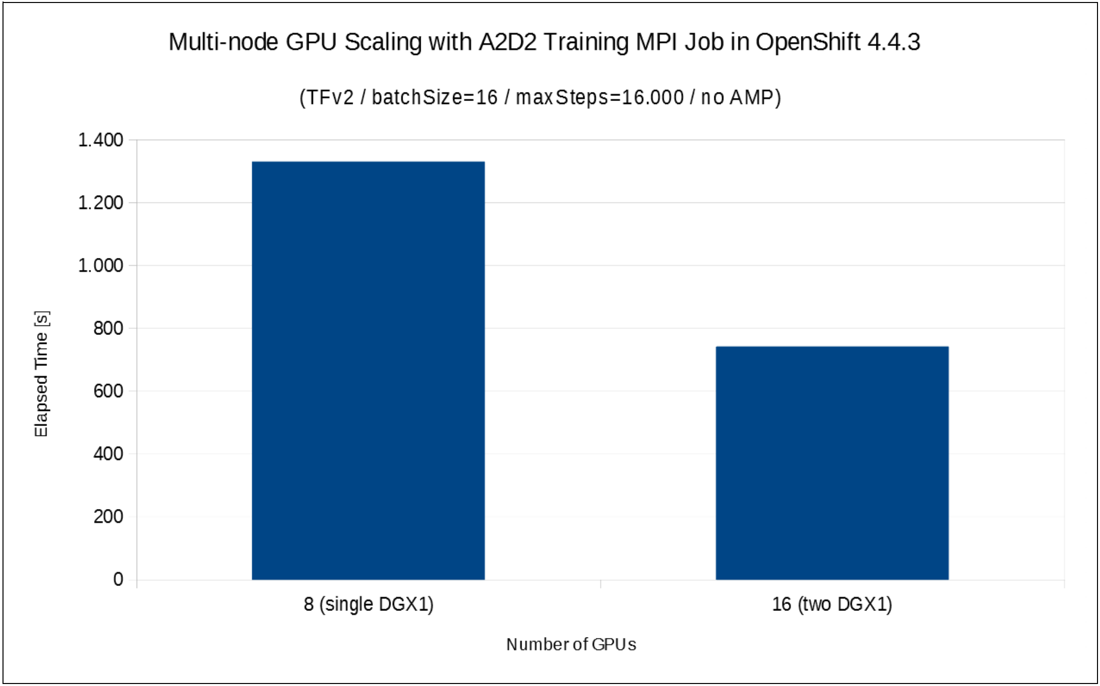


Figure 5-2 Median value of the elapsed time of five training runs on a single DGX-1 node (eight GPUs) and on two DGX-1 nodes (16 GPUs) in Red Hat OpenShift

By scaling the training job from a single DGX-1 node (eight GPUs) to two DGX-1 nodes (16 GPUs) in Red Hat OpenShift, we observe that we can seamlessly scale out across physical node boundaries and reduce the job run time from 1328 seconds to 740 seconds (55.7%). This reduction results in an increase in speed of 1.8x (at only 10% costs in performance compared to an ideal value of 2.0X).

Repeated and frequent training and validation cycles of the neural networks are common in the autonomous vehicle industry. Being able to scale out conveniently and efficiently these workloads across GPU resources is essential to reduce run times and improve time to new insights and better models.

The results in this paper prove that multi-GPU and multi-node (horizontal) scaling of GPU resources for deep neural network (DNN) training jobs works seamlessly and efficiently in Red Hat OpenShift 4.4.3 as a container orchestration platform with DGX worker nodes, with NVIDIA Mellanox InfiniBand remote direct memory access (RDMA) as the network infrastructure and IBM Spectrum Scale as back-end storage for a scalable high-performance “data lake”.

5.3 Application

The rate of innovation in developing deep learning (DL) based solutions is crucial. One of the most challenging engineering tasks is building the right training and validation data set. Many DL practitioners agree with the statement that they are essentially building data sets rather than networks (see Revisiting the Unreasonable Effectiveness of Data).

The validation of a trained neural network in various relevant situations provides confidence about its performance and valuable insights into its weaknesses. Multiple validation data sets might be built to validate certain aspects or the model performance in a subset of the operational domain.

A rich set of metadata for the data set is crucial to building the necessary data sets and understanding neural network strength and weaknesses.

In the context of this paper, we use IBM Spectrum Discover as a metadata store. The metadata that is available in the A2D2 was maintained in IBM Spectrum Discover. IBM Spectrum Discover allows us to ask queries with SQL. We are aware that many more metadata tags are necessary in a production environment.

In this section, we name a few samples where a metadata store helps to derive steps for developing the data set or network:

- The DNN under test performs well on large objects and worse on smaller ones when object occurrence is balanced. This situation also happens with the randomly chosen subset of our training data set. As shown in Figure 5-3 on page 59, we evaluated a training after just 20,000 steps by using a confusion matrix. For a perfect predictor, the diagram would have only a diagonal line from upper left to lower right that would read that the network classified all pixels of a certain class right.

  Figure 5-3 on page 59 shows a confusion matrix presenting the training evaluation. There is a logarithmic scale for the color coding.

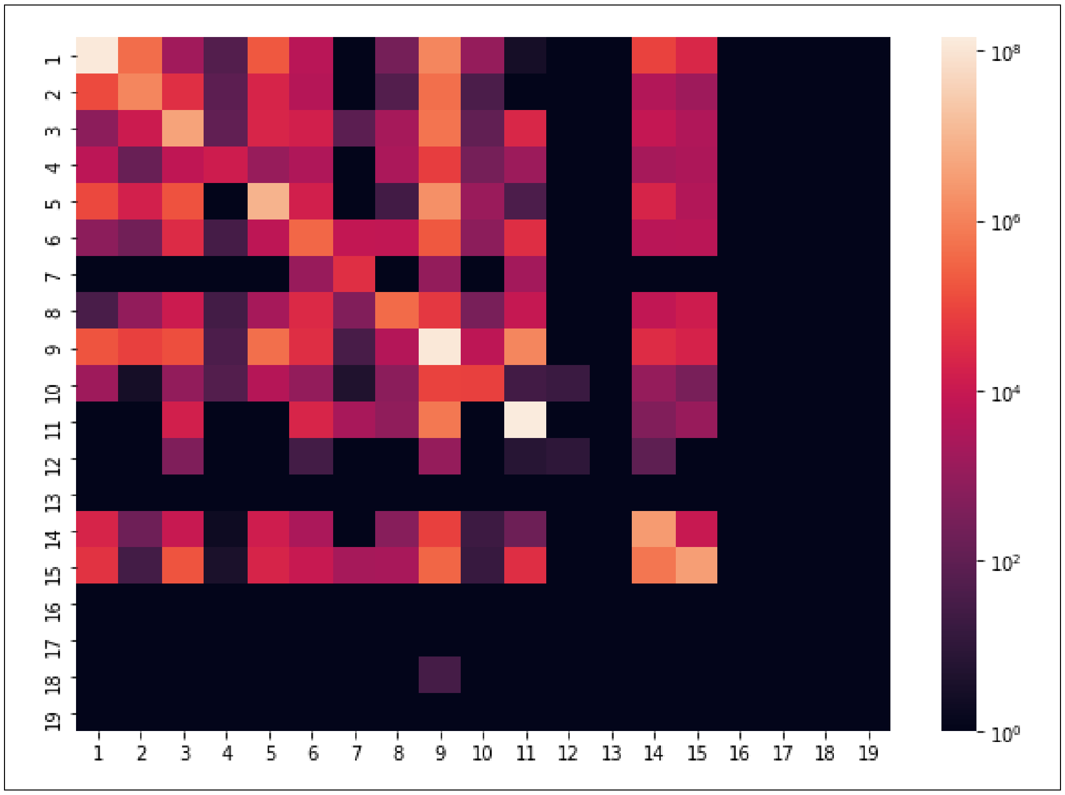


Figure 5-3 Confusion matrix presenting the training evaluation

There are two obvious observations:

– There are classes (the bright ones on the diagonal, such as class 1, RD normal street; class 9, Sky; and class 11, Nature Object) that can be detected well. With the metadata in IBM Spectrum Discover, we see that the frequency of those pixels occurring correlate. A query like the following one can help us to understand potential reasons:

```
https://localhost/db2whrest/v1/sql_query -X POST -dselect
sum(int(t_rd_normal_street.value)) /
((count(t_rd_normal_street.value)*1920*1208)/100) from t_rd_normal_street
with ur;
```

Here is the result:

```
22.22743915175
```

In contrast, other classes such as “Road Blocks” appear at only 0.85 percent, and the frequency of object occurrence is not sufficient.

– Furthermore, it is obvious that the data that used for the evaluation has no representatives for bicycle class (19). Thus, the validation data set must be extended. A query like the following one can help to select frames that are known to contain the required class:

```
https://localhost/db2whrest/v1/sql_query -X POST -dselect
platform,datasource,filename from metaocean,t_front,t_center,t_bicycle where
t_front.fkey=metaocean.fkey and int(t_front.value)=1 and
t_center.fkey=metaocean.fkey and int(t_center.value)=1 and
t_bicycle.fkey=metaocean.fkey and int(t_bicycle.value)>0
```

Here is the result:

```
0,"IBM
COS","a2d2","camera_lidar_semantic/20181107_133445/camera/cam_front_center/2
0181107133445_camera_frontcenter_000021093.png"
1,"IBM
COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/2
0181107132300_camera_frontcenter_000001645.png"
2,"IBM
COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/2
0181107132300_camera_frontcenter_000004159.png"
3,"IBM
COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/2
0181107132300_camera_frontcenter_000002707.png"
4,"IBM
COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/2
0181107132300_camera_frontcenter_000004093.png"
...
```

- The DNN performs poorly in a certain case, for example, close to bridges and poor lighting. A join by location of a map database together with the used training data allows such a query.
- The DNN performs not as expected on a specific class though it has plenty of training data. Here, the data set might be too simple and not as complicated as the validation data set. A selection of those frames with an active learning uncertainty estimate allows you to reduce the data set to the most informative training samples.

It is common in the automotive industry that each training frame carries several hundred metadata tags. Data from the recording ego vehicle and the subdomains of the operational design domain (ODD) are typical sources of metadata. In our experiments, we solely rely on the A2D2 without the help of other data sources. On uploading the data set, the available information was extracted, such as the presence of certain classes and their pixel count.

Most of that metadata information is derived by joining information from multiple sources together with the recorded data, for example, from high-definition maps.

Usually, this data is pre-joined for practicality reasons and to avoid joins at query time. Pre-joining easily leads to many attributes. Likely, the time comes that one of those attributes is needed. Data minimalism on the metadata is not encouraged. Thus, it is more practical to keep all of the attributes rather to repeatedly suffer from missing attributes and then add them with much effort.

Here, we leverage IBM Spectrum Discover as a metadata database. It works together with IBM storage solutions, and it is used as a metadata store in the context of this work. Metadata information for each frame of the A2D2 is created in IBM Spectrum Discover to show how it can be used in that context.

Building a representative validation data set is a challenge. Validation data sets are much larger for autonomous vehicles than their training data set.

Figure 5-4 on page 61 shows the process of building a representative validation data set.

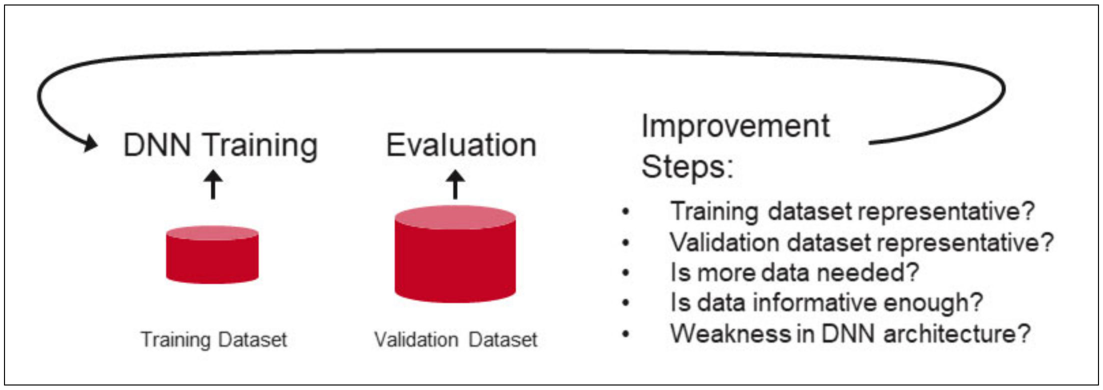

DNN Training
Evaluation
Improvement Steps:
Training dataset representative?
Validation dataset representative?
Is more data needed?
Is data informative enough?
Weakness in DNN architecture?
Training Dataset
Validation Dataset

Figure 5-4 Building a representative validation data set

Training and validation of the neural networks are done on a frequent basis. Validating the trained DNN against a large validation data set is critical to understanding its weaknesses. For representative results, the validation data set is larger than the training data set and reflects the distribution of the target operational domain. Large-scale DNN validation is an inference job that is rolled out into the cluster in a fan-out manner to multiple workers. It follows a mapreduce pattern where each worker takes care of a certain partition of the validation data. Results are collected and aggregated.

5.4 Integrating IBM Spectrum Discover and IBM Spectrum LSF to find the correct data based on labels

For demonstration purposes, we add IBM Spectrum LSF as a workload manager and IBM Spectrum Discover as a metadata search engine to find the correct data for our inference job and to automate the workflow.

IBM Spectrum Discover is a modern metadata management software that provides data insights for exabyte-scale heterogeneous file, object, backup, and archive storage on-premises and in the cloud. The software easily connects to these data sources to rapidly ingest, consolidate, and index metadata for billions of files and objects.

IBM Spectrum Discover provides a rich metadata layer that enables storage administrators, data stewards, and data scientists to efficiently manage, classify, and gain insights from massive amounts of data. It improves storage economics, helps mitigate risk, and accelerates large-scale analytics to create competitive advantage and speed critical research.

IBM Spectrum LSF is a complete workload management solution for demanding high-performance computing (HPC) environments. Featuring intelligent, policy-driven scheduling and easy to use interfaces for job and workflow management, it helps organizations to improve competitiveness by accelerating research and design while controlling costs through superior resource utilization.

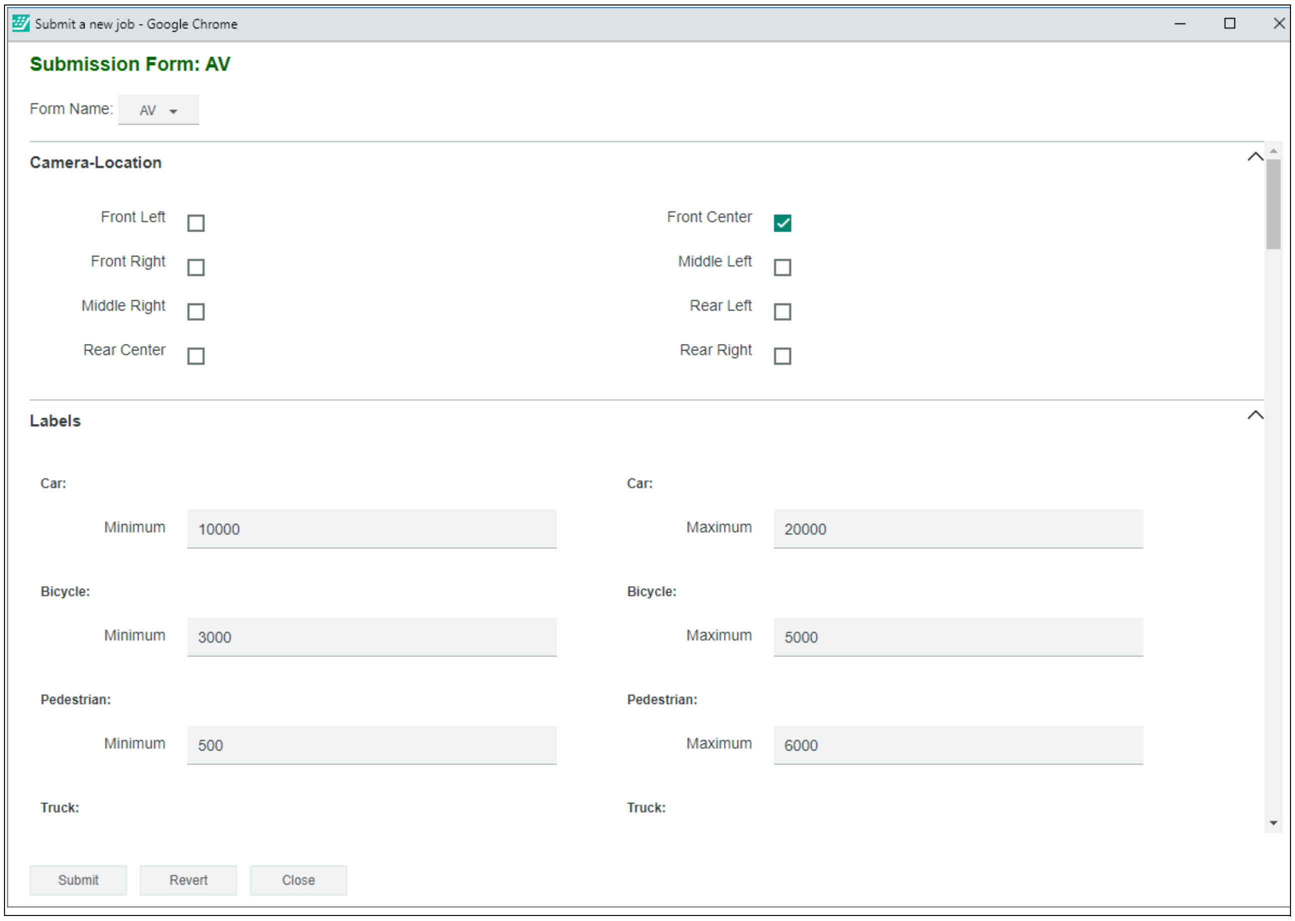
For this use case, we connected IBM Spectrum Discover to the data source and let IBM Spectrum Discover scan the content. The scan was needed because we added IBM Spectrum Discover after the data was stored in the storage system. IBM Spectrum Discover comes with built-in functions that can react to new incoming data and automatically detect metadata for the new data and catalogs it.

To find the data that is needed for model training or inference based on label details, a user can run a simple REST query in SQL format against IBM Spectrum Discover as follows:

```
https://localhost/db2whrest/v1/sql_query -X POST -dselect
platform,datasource,filename from
metaocean,t_front,t_center,t_car,t_bicycle,t_pedestrian where
t_front.fkey=metaocean.fkey and int(t_front.value)=1 and
t_center.fkey=metaocean.fkey and int(t_center.value)=1 and
t_car.fkey=metaocean.fkey and int(t_car.value)>=10000 and int(t_car.value)<=20000
and t_bicycle.fkey=metaocean.fkey and int(t_bicycle.value)>=3000 and
int(t_bicycle.value)<=5000 and t_pedestrian.fkey=metaocean.fkey and
int(t_pedestrian.value)>=500 and int(t_pedestrian.value)<=6000
```

A user can also work with a customized job user interface window that is created in IBM Spectrum LSF.

Figure 5-5 shows an example of an IBM Spectrum LSF AV job submission template.


Submit a new job - Google Chrome

Submission Form: AV

Form Name: AV

Camera-Location

Front Left	☐	Front Center	✓
Front Right	☐	Middle Left	☐
Middle Right	☐	Rear Left	☐
Rear Center	☐	Rear Right	☐

Labels

Car: Minimum	10000	Car: Maximum	20000
Bicycle: Minimum	3000	Bicycle: Maximum	5000
Pedestrian: Minimum	500	Pedestrian: Maximum	6000
Truck:		Truck:	

Figure 5-5 IBM Spectrum LSF AV job submission template example

As a user, start a job by selecting the needed labels, as shown in Figure 5-5 on page 62. IBM Spectrum LSF forwards the request to IBM Spectrum Discover. IBM Spectrum Discover answers with a list of files that match the requested label details that are shown in the following example:

```
0,"IBM COS","a2d2",
"camera_lidar_semantic/20181008_095521/camera/cam_front_center/20181008095521_came
ra_frontcenter_000038997.png"
1,"IBM COS","a2d2",
"camera_lidar_semantic/20180925_135056/camera/cam_front_center/20180925135056_came
ra_frontcenter_000062196.png"
2,"IBM COS","a2d2",
"camera_lidar_semantic/20181016_125231/camera/cam_front_center/20181016125231_came
ra_frontcenter_000005145.png"
3,"IBM COS","a2d2",
"camera_lidar_semantic/20180807_145028/camera/cam_front_center/20180807145028_came
ra_frontcenter_000010529.png"
4,"IBM COS","a2d2",
"camera_lidar_semantic/20181008_095521/camera/cam_front_center/20181008095521_came
ra_frontcenter_000031069.png"
5,"IBM COS","a2d2",
"camera_lidar_semantic/20181016_095036/camera/cam_front_center/20181016095036_came
ra_frontcenter_000041815.png"
6,"IBM COS","a2d2",
"camera_lidar_semantic/20181016_095036/camera/cam_front_center/20181016095036_came
ra_frontcenter_000020381.png"
7,"IBM COS","a2d2",
"camera_lidar_semantic/20180925_124435/camera/cam_front_center/20180925124435_came
ra_frontcenter_000045908.png"
8,"IBM COS","a2d2",
"camera_lidar_semantic/20180925_124435/camera/cam_front_center/20180925124435_came
ra_frontcenter_000043968.png"
...
```

IBM Spectrum Discover can catalog multiple different storage systems and returns the platform and data source with the data details.

With the help of IBM Spectrum Scale Active File Management (AFM), the data can be pre-cached close to the AI workloads. Pre-caching helps keep the accelerators busy as the to-be-analyzed data is present at the correct time, even if rather “slow” storage systems / data lakes host the data. Pre-caching helps to ensure that high-performing storage is not overutilized and runs out of space.

With the help of this solution, the data location and time of availability for the AI job can be fully abstracted.

NVIDIA DGX family:

When this PoC was started, the new NVIDIA DGX A100 system was not yet announced. Instead, two DGX-1 systems were deployed. The DGX-1 is a purpose-built system for deep learning (DL) with fully integrated hardware and software that can be deployed quickly and easily. DGX-1 features eight NVIDIA V100 GPU accelerators with a Tensor Core architecture that is connected through NVIDIA NVLink, which is the NVIDIA high-performance GPU interconnect, in a hybrid cube-mesh network. Together with dual socket Intel Xeon CPUs and four 100 Gb InfiniBand network interface cards, DGX-1 provides unprecedented performance for DL training.

The successor system, which is called NVIDIA DGX A100, was announced on 8 June 2020. There is a plan to replicate this PoC with the next-generation NVIDIA DGX platform. The NVIDIA DGX A100 is the universal system for all AI workloads, offering unprecedented compute density, performance, and flexibility in the world's first 5 petaFLOPS AI system. The NVIDIA DGX A100 system features the world's most advanced accelerator, the NVIDIA A100 Tensor Core GPU, which enables enterprises to consolidate training, inference, and analytics into a unified, easy-to-deploy AI infrastructure that includes direct access to NVIDIA AI experts. For more information, see *NIVIDIA DGX A100: The Universal System for AI Infrastructure*.

Architecture details for Ampere can be found in this blog. Details about the new MIG mode are described in this blog.

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this paper.

Online resources

These websites are also relevant as further information sources:

- Audi Autonomous Driving Dataset (A2D2) citation

  ```
  @article{geyer2020a2d2,
  title={{A2D2: Audi Autonomous Driving Dataset}},
  author={Jakob Geyer and Yohannes Kassahun and Mentar Mahmudi and Xavier Ricou
  and Rupesh Durgesh and Andrew S. Chung and Lorenz Hauswald and Viet Hoang Pham
  and Maximilian M{\"u}hlegg and Sebastian Dorn and Tiffany Fernandez and Martin
  J{\"a}nicke and Sudesh Mirashi and Chiragkumar Savani and Martin Sturm and
  Oleksandr Vorobiov and Martin Oelker and Sebastian Garreis and Peter
  Schuberth},
  year={2020},
  eprint={2004.06320},
  archivePrefix={arXiv},
  primaryClass={cs.CV},
  url = {https://www.a2d2.audi}
  }
  ```

  – Public License

    https://aev-autonomous-driving-dataset.s3.eu-central-1.amazonaws.com/LICENSE.txt

  – This data set is released under the CC BY-ND 4.0 license

    https://creativecommons.org/licenses/by-nd/4.0/

  – Liability and Copyright (licensed material was not modified for this study)

    https://www.a2d2.audi/a2d2/en/legal.html

  – Driving Dataset Downloads and Citation

    https://www.a2d2.audi/a2d2/en/download.html

  – *A2D2: Audi Autonomous Driving Dataset*

    https://arxiv.org/pdf/2004.06320.pdf

- Designing and Building End-to-End Data Pipeline Using IBM Spectrum Storage for AI with NVIDIA DGX-2™ Systems

  https://www.ibm.com/downloads/cas/GGWQ40KE

- How IBM ESS3000 makes your GPUs fly: A field report based on the Deep Thought project

  https://www.spectrumscaleug.org/wp-content/uploads/2020/03/SSSD20DE-How-ESS3000-makes-your-GPUs-fly-A-field-report-based-on-the-Deep-Thought-project-SVA.pdf

- How Volkswagen Tests Autonomous Cars with GPUs and Red Hat OpenShift

  https://www.openshift.com/blog/how-volkswagen-tests-autonomous-cars-with-gpus-and-openshift
- IBM Automotive 2030

  https://www.ibm.com/downloads/cas/NWDQPK5B
- IBM Elastic Storage System (ESS) 3000 Version 6.0.0 IBM Knowledge Center

  https://www.ibm.com/support/knowledgecenter/SSZL24_6.0.0/ess3000_600_welcome.html
- IBM ESS3000

  https://www.ibm.com/us-en/marketplace/elastic-storage-system-3000
- IBM Spectrum Scale

  https://www.ibm.com/products/scale-out-file-and-object-storage
- IBM Spectrum Storage for AI with DGX Systems

  https://www.ibm.com/downloads/cas/MNEQGQVP
- IBM Spectrum Scale 5.0.4 IBM Knowledge Center

  https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.4/ibmspectrumscale504_welcome.html
- NVIDIA Mellanox

  https://docs.mellanox.com/pages/releaseview.action?pageId=19804150
- Red Hat Enterprise Linux CoreOS (RHCOS)

  https://access.redhat.com/documentation/en-us/openshift_container_platform/4.1/html/architecture/architecture-rhcos

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5610-00

ISBN 0738459097