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

First Edition (September 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.

This document was created or updated on September 17, 2020.
# Contents

Notices ................................................................................................................................. v
Trademarks .............................................................................................................................. vi

Preface ...................................................................................................................................... vii
Authors ...................................................................................................................................... vii
Now you can become a published author, too! ........................................................................... viii
Comments welcome ................................................................................................................ viii
Stay connected to IBM Redbooks ............................................................................................ ix

Chapter 1. Overview ................................................................................................................. 1
1.1 Abstract ............................................................................................................................... 2
1.2 Proof of Concept Background .............................................................................................. 2

Chapter 2. PoC Environment ..................................................................................................... 5
2.1 Prerequisites ......................................................................................................................... 6

Chapter 3. Installation ................................................................................................................ 9
3.1 Configuring the NVIDIA Mellanox EDR InfiniBand network ............................................ 10
3.2 Integrating DGX-1 Systems as Worker Nodes into the
    Red Hat OpenShift 4.4.3 Cluster ...................................................................................... 11
    3.2.1 Installing Red Hat Enterprise Linux 7.6 and DGX Software ...................................... 11
    3.2.2 Installing NVIDIA Mellanox InfiniBand Drivers (MLNX_OFED) ................................. 12
    3.2.3 Installing GPUDirect Storage Kernel Module ............................................................ 13
    3.2.4 Installing NVIDIA Mellanox SELinux Module ............................................................. 13
    3.2.5 Adding DGX-1 systems as Worker Nodes to the Red Hat OpenShift Cluster .......... 14
3.3 Adding DGX-1 Systems as Client Nodes to the IBM Spectrum Scale Cluster .................... 14
3.4 Red Hat OpenShift Stack ....................................................................................................... 15
    3.4.1 Special Resource Operator (SRO) .............................................................................. 15
    3.4.2 NVIDIA Mellanox RDMA Shared Device Plugin ......................................................... 17
    3.4.3 Enabling IPC_LOCK in User Namespace for RDMA Shared Device Plugin ................. 19
    3.4.4 MPI Operator .......................................................................................................... 22
    3.4.5 IBM Spectrum Scale CSI .......................................................................................... 22

Chapter 4. Preparation and Functional Testing ........................................................................ 25
4.1 Test RDMA via the InfiniBand Network .............................................................................. 25
4.2 Prepare Persistent Volumes with IBM Spectrum Scale CSI .............................................. 29
4.3 MPI Job Definition ............................................................................................................ 31
4.4 Connectivity Tests with NVIDIA Collective Communications Library .......................... 33
4.5 Multi-GPU, Multi-Node GPU Scaling with TensorFlow ResNet-50 Benchmark .............. 44

Chapter 5. Deep neural network (DNN) Training on A2D2 Semantic Segmentation
          Dataset ................................................................................................................................. 51
5.1 Description A2D2 Dataset .................................................................................................. 51
5.2 Multi-GPU, Multi-Node GPU Scaling Results for DNN Training Jobs .............................. 52
5.3 Application ......................................................................................................................... 55
5.4 Integrating IBM Spectrum Discover and IBM Spectrum LSF to find the right data based on
    labels .................................................................................................................................... 58

Related publications ................................................................................................................ 63
Online resources ....................................................................................................................... 63
Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

<table>
<thead>
<tr>
<th>DS6000™</th>
<th>IBM Research®</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS8000®</td>
<td>IBM Spectrum®</td>
</tr>
<tr>
<td>IBM®</td>
<td>LSF®</td>
</tr>
<tr>
<td>IBM Elastic Storage®</td>
<td>POWER®</td>
</tr>
<tr>
<td>Redbooks®</td>
<td>Redbooks (logo)</td>
</tr>
<tr>
<td>System Storage™</td>
<td></td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Preface

This IBM® Redpaper publication describes the architecture as well as the installation procedure and the results for running a typical training application working on an automotive dataset, in an orchestrated and secured environment, providing horizontal scalability of GPU resources across physical node boundaries for deep neural networks (DNN) workloads.

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

In a later stage of the Redpaper we added IBM Spectrum® LSF® as a workload manager and IBM Spectrum Discover as a metadata search engine to find the right data for our inference job and to automate the data science workflow. With the help of this solution, the data location (could be even different storage systems) and time of availability for the AI job can be fully abstracted, which introduces 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 US. Simon joined the IBM Spectrum Scale development team in 2014 and since then worked on: Openstack Swift integration, designing and building System Health and Proactive Services solutions. Since March 2019 he is the world wide 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 the 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 working at the European Storage Competence Center in Mainz, Germany, providing 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. Gero participated in the product rollout of the IBM System Storage™ DS6000/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, USA, 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/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 & verification 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  
Nischal Muchakani  
**Red Hat**

We gratefully acknowledge the contributions to the study resulting in this Redpaper. 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 assisted in the research for this paper.

**Now you can become a published author, too!**

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:  
[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

**Comments welcome**

Your comments are important to us!
We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online Contact us review Redbooks form found at:
  
  ibm.com/redbooks

- Send your comments in an email to:
  redbooks@us.ibm.com

- 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

- Find us on Facebook:
  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:
  http://twitter.com/ibmredbooks

- 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:

- Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html
Overview

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

This IBM Redpaper publication describes the architecture as well as the installation procedure and the results for running a typical training application working on an automotive dataset, in an orchestrated and secured environment, providing horizontal scalability of GPU resources across physical node boundaries for deep neural networks (DNN) workloads.

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

In a later stage of the Redpaper we added IBM Spectrum LSF as a workload manager and IBM Spectrum Discover as a metadata search engine to find the right data for our inference job and to automate the data science workflow. With the help of this solution, the data location (could be even different storage systems) and time of availability for the AI job can be fully abstracted, which introduces valuable information for data scientists.

1.2 Proof of Concept Background

For many companies running large-scale infrastructures, orchestration of containerized applications it is essential to guarantee 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. That means 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 value-added capabilities, in addition to the fundamental container orchestration capabilities provided by Kubernetes. The capabilities of Red Hat OpenShift with Kubernetes 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 hybrid cloud. Automated deployment and lifecycle management for hundreds of ISV and custom application workloads and infrastructure services using 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 from these capabilities of a supported orchestration platform fully supported by Red Hat.
Important challenges remain, when it comes to large-scale deep learning workloads such as the development of deep neural networks that are trained to be used in the perception software stack for autonomous cars. These large-scale deep learning workloads are typically associated with applications that make use of NVIDIA GPUs. NVIDIA provides a hub of GPU-accelerated, optimized containers that can easily scale to hundreds of GPUs 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 as well as the installation procedure and the results for running a typical training application working on an automotive dataset.

Deep neural network training on huge datasets is a computational expensive task and 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 using concepts like Message Passing Interface (MPI) and Horovod.

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

This Redpaper provides guidance on 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 dataset 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 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/used:

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

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

We also deployed Security Context Constraints (SCCs) to allow granular control of permissions required for pods running AI workloads, and for processing access requirements to RDMA resources for best performance. SCCs represent a concept - similar to 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, 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 in order to be accepted into the system.
Figure 2-1 shows the environment used for this proof of concept.

The solution described in this paper comprises the following major components for the installation:

- DGX-1 systems running with Red Hat 7.6 as worker nodes (40 cores, 512GB memory, four single-port NVIDIA Mellanox ConnectX-4 EDR InfiniBand cards)
- NVIDIA Mellanox Quantum HDR 7800 managed Switch to connect worker nodes and storage backend
- IB: four NVIDIA Mellanox dual-port ConnectX-5 HCAs in ESS3000, four single-port NVIDIA Mellanox ConnectX-4 HCAs in each DGX-1 system, connected with 16 NVIDIA Mellanox EDR cables
- NVIDIA Mellanox 100Gbps EDR InfiniBand Network
- Standard 1Gbps (or higher) Ethernet admin network for all components
2.1 Prerequisites

Figure X shows the software release levels used for Red Hat OpenShift and IBM Spectrum Scale and the role of each node.

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 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 details on RHCOS refer to the, “Related publications” on page 63. 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 currently supported only by RHEL, we used RHEL 7.6 for those systems.

  The compute-cluster consists of:
  - Three Master Nodes running RHCOS on Lenovo SR650
  - Two Worker Nodes running RHCOS in a VM on a Lenovo SR650 (used to have a minimal healthy OCP cluster up and 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, highly-available, clustered file system and associated management software, available on a variety of platforms. IBM Spectrum Scale
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 based on Red Hat 7.6
- IBM Spectrum Scale client running in a VM (providing GUI / REST, Quorum and Management functionality) based on Red Hat 7.6
- Remote mounted IBM Spectrum Scale file system called ess3000_4M from an IBM ESS3000 storage system, configured with a 4 MiB blocksize (good fit to the average image size of 3-4 MiB of the used A2D2 dataset)

> IBM 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, delivering 40GB/s of data throughput per 2U system.

The storage cluster consists of:

- IBM ESS3000 consists of 2 canisters, each running IBM Spectrum Scale 5.0.4.3 on Red Hat 8.1
- Autonomous Driving Dataset (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 functionality)

To review more details about IBM Spectrum Scale, IBM ESS 3000, DGX-1 systems, Red Hat OpenShift, refer to the , “Related publications” on page 63.
Chapter 3. Installation

The installation procedure comprises the following steps.

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

- Installing and adding the DGX-1 systems as worker nodes to the Red Hat OpenShift 4.4.3 cluster
  - Install Red Hat Enterprise Linux 7.6 and NVIDIA RPM packages
  - Install NVIDIA Mellanox InfiniBand drivers (MLNX_OFED)
  - Install the NVIDIA® GPUDirect® Storage Kernel Module (nv_peer_mem)
  - Install NVIDIA Mellanox SELinux Module
  - Add DGX-1 systems as RHEL7 based worker nodes to the Red Hat OpenShift 4.4.3 cluster

- Installing and configuring additional components in the Red Hat OpenShift 4.4.3 stack
  - Special Resource Operator (SRO) for GPU support
  - NVIDIA Mellanox RDMA Shared Device Plugin for InfiniBand RDMA support
  - Security Context Constraints (SCC) with IPC_LOCK capability
  - Service account, role and role binding per user namespace to grant access to RDMA resources for pods using the RDMA Shared Device Plugin
  - MPI Operator to conveniently schedule multi-GPU and multi-node training jobs
  - IBM Spectrum Scale CSI 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 the need to duplicate (copy or move) huge amounts of data for model training, model validation or inference.
3.1 Configuring the NVIDIA Mellanox EDR InfiniBand network

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 200Gb/s QM8700 InfiniBand Smart Switch as shown in the Chapter 2, “PoC Environment” on page 5, providing a total of eight 100Gbps EDR InfiniBand connections between the IBM ESS3000 storage system and the two DGX-1 systems.

The InfiniBand Subnet Manager (SM) is launched on the InfiniBand switch with:

```
[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
```

The active switch config used for this paper is:

```
[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$5IjwJuo$LYlJAJ...9qZgAVyLy1NDakzyVTXyCbiWzcO
username monitor password 7 $6$2609wpM$E1G35T.9eakk...ShKNI2r4zIMsdInBM
#### AAA remote server configuration
##
# ldap bind-password ********
```
3.2 Integrating DGX-1 Systems as Worker Nodes into the Red Hat OpenShift 4.4.3 Cluster

On x86 platforms, Red Hat OpenShift 4 can be extended with RHEL7 based worker nodes (see 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. The steps involved are as follows:

- Installing RHEL 7.6 as base OS on the DGX-1 systems
- Installing DGX Software for Red Hat Enterprise Linux
- Installing the NVIDIA Mellanox OFED (MLNX_OFED)
- Installing the GPUDirect Storage Kernel Module
- Adding the DGX-1 systems as worker nodes to the Red Hat OpenShift 4.4.3 cluster

3.2.1 Installing 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 followed by installing the DGX Software as described in Installing the DGX Software.
1. Install the NVIDIA repo for RHEL7
   
   ```
   # 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`
   
   ```
   [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:
   
   ```
   # 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 in order to gain more control of the package and kernel updates. 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 used in this paper is 3.10.0-957.27.2.e17.x86_64 on the DGX-1 systems. Red Hat OpenShift 4.4. generally supports RHEL7 minor versions RHEL 7.6 up to 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 at this time.
   
   ```
   # subscription-manager release -set=7.6
   # subscription-manager release --show
   Release: 7.6
   # yum clean all
   # yum update
   ```

   Reboot the system to load the drivers and to update system configurations.

### 3.2.2 Installing NVIDIA Mellanox InfiniBand Drivers (MLNX_OFED)

Follow the instructions at Installing NVIDIA Mellanox InfiniBand Drivers to install the **NVIDIA Mellanox OpenFabrics Enterprise Distribution (MLNX_OFED)** for Linux for Red Hat operating systems. We install NVIDIA Mellanox OFED **MLNX_OFED_LINUX-4.7-3.2.9.0** on both DGX-1 systems using the same release version as on the IBM ESS3000 system.

After installing the MLNX_OFED drivers we skip the step to install the **NVIDIA peer memory kernel module** (nv_peer_mem) from the pre-built RPM package as it will not install properly without a full CUDA installation (which we don't have installed on the host OS). Instead, we
follow the instructions in the next section Installing GPUDirect Storage Kernel Module to build and install the NVIDIA peer memory kernel module (nv_peer_mem) manually.

3.2.3 Installing GPUDirect Storage Kernel Module

The NVIDIA peer memory (nv_peer_mem) kernel module is required for GPUDirect Storage support. As the installation of the provided RPM package requires a CUDA installation which is not present in our setup, we install the kernel module manually. GPUDirect Storage enables multiple GPUs and network adapters to directly read and write CUDA host and device memory, eliminating unnecessary memory copies, dramatically lowering CPU overhead and latency which results in significant performance improvements in data transfer times for applications running on NVIDIA Tesla and Quadro products.

To install the GPUDirect Storage kernel module without CUDA follow the instructions below.

First, compile and install the nv_peer_mem source code as follows on one DGX system:

```bash
# 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/.
# rpm --rebuild /tmp/nvidia_peer_memory-
# rpm -ivh /root/rpmbuild/RPMS/x86_64/nvidia_peer_memory-1.0-9.x86_64.rpm
```

Then distribute the built RPM and install it on the other DGX systems.

Check that the module is loaded accordingly otherwise load it with modprobe nv_peer_mem:

```bash
# 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_ucm,ib_ipoib
```

Make sure that the module is loaded automatically on every system reboot!

3.2.4 Installing NVIDIA Mellanox SELinux Module

Finally download and unzip the NVIDIA Mellanox SELinux module from:

https://docs.mellanox.com/download/attachments/19804150/infiniband.zip?version=1&modificationDate=1575464686823&api=v2&download=true

and install it on the DGX worker nodes as follows:

```bash
# 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 following 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 Red Hat Subscription Manager (RHSM), attach an active Red Hat OpenShift Container Platform subscription, and enable the required repositories.

Remember to add your new worker nodes to your active Red Hat OpenShift load balancer configuration (e.g., HAproxy).

3.3 Adding DGX-1 Systems as Client Nodes to the 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. In a next step, we have to add these DGX worker nodes to the local IBM Spectrum Scale 5.0.4.3 cluster as IBM Spectrum Scale client nodes either using the IBM Spectrum Scale installation toolkit as described in Adding nodes, NSDs, or file systems to an existing installation or manually following the instructions in Adding nodes to a GPFS cluster in the IBM Spectrum Scale Knowledge Center.

In our setup the local IBM Spectrum Scale cluster remotely mounts a IBM Spectrum Scale file system called ess3000_4M from an IBM ESS3000 storage system. For more information about managing access to a remote IBM Spectrum Scale file system refer to Accessing a remote GPFS file system. The file system is configured with a 4 MiB block size, which is a good fit to the average image size of 3-4 MiB 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, i.e. four NVIDIA Mellanox ConnectX-5 ports on each of the two IBM ESS3000 I/O server node and four NVIDIA Mellanox ConnectX-4 ports on each DGX-1 system. This provides a total bandwidth of eight InfiniBand EDR 100Gbps 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 additional IP address on all mlx5_1 interfaces (using IPoIB, 10.10.11.0/24) on the IBM ESS3000 nodes and the DGX-1 nodes as additional subnet for IBM Spectrum Scale which is used for mmfsd daemon tcp/ip communication over the 100Gbps link instead of the 1Gbps admin network. With verbsRdmaSend enabled the amount of communication over this tcp/ip link is neglectable. The 1Gbps admin network is used as default admin and daemon network connecting all IBM Spectrum Scale, DGX and IBM ESS nodes.

The DGX-1 nodes in the local IBM Spectrum Scale cluster are configured as follows:

```
[dgx]
pagepool 128G
workerThreads 1024
ignorePrefetchLUNCount yes
maxFilesToCache 1M
maxStatCache 1M
```
3.4 Red Hat OpenShift Stack

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

- Special Resource Operator (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 plugin to access data in the IBM Spectrum Scale file system

3.4.1 Special Resource Operator (SRO)

The Special Resource Operator (SRO) extends Red Hat OpenShift to provide support for special resources that need extra management like the NVIDIA GPUs in the DGX-1 systems. Without the SRO there would be no GPU resource as such known to the cluster and the appropriate scheduling of pods relying on that resource as well as 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 accordingly with node specific attributes, like PCI cards, kernel or OS version, etc.

The SRO is also available as operator in the OperatorHub and can be installed and deployed directly from the Red Hat OpenShift 4 GUI using the Operator Lifecycle Management framework, as shown in Figure 3-1.
For this paper we used the latest version (as of at June 25 17:23 CEST) on GitHub at https://github.com/openshift-psap/special-resource-operator which can be installed from the command line by the system admin of the Red Hat OpenShift cluster as follows:

The necessary steps involve:

1. Installation of the NFD operator
   
   ```
   # git clone https://github.com/openshift/cluster-nfd-operator
   # cd cluster-nfd-operator
   # make deploy
   ```

2. Installation of the SRO operator
   
   ```
   # 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 of the required pods successfully - so please be patient. During that time you will notice that SRO pods are failing and restarting until all required build processes have completed successfully and the SRO pods finally enter a steady running state.

Make sure the nouveau kernel module is disabled - this should already be the case when installing the NVIDIA packages for the DGX-1 system - otherwise just disable the nouveau module manually.

The SRO adds the following allocatable resources to each DGX-1 worker node (8 GPUs in this case):

Allocatable:

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

With the SRO version that we use for this setup we have to set SElinux on the DGX-1 worker nodes to “permissive” mode so that the SRO can successfully build the required container images and finally start all the necessary pods.
3.4.2 NVIDIA Mellanox RDMA Shared Device Plugin

In addition to the NFD and SRO operators which add NVIDIA GPUs as a new resource in the Red Hat OpenShift cluster we also need to install the NVIDIA Mellanox RDMA Shared Device plugin to further provide InfiniBand resources to Red Hat OpenShift providing non-privileged pods access to GPU and InfiniBand resources and allowing Red Hat OpenShift to scheduled pods requesting these resources accordingly. InfiniBand enables high performance communication between GPUs with NVIDIA Collective Communications Library (NCCL) so that multi-node workloads can scale out seamlessly across worker nodes. The NVIDIA Mellanox RDMA device plugin runs as a daemonset on all worker nodes and allows a granular assignment of individual NVIDIA Mellanox InfiniBand resources to pods.

Cloning the GitHub repository

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

and following the instructions at

https://github.com/Mellanox/k8s-rdma-shared-dev-plugin

the RDMA device plugin can be installed in two steps:

1. Create and deploy the config map

The DGX-1 systems have four single-port NVIDIA Mellanox ConnextX-4 EDR InfiniBand cards (NVIDIA Mellanox Technologies MT27700 Family) associated with 2 NUMA nodes:

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

In our setup, we configured the config map for the NVIDIA Mellanox device driver plugin as follows:

```yaml
# cat 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"]
]
]
}

to make all four IB ports available in Red Hat OpenShift as selectable resources, named
shared_ib0, shared_ib1, shared_ib2, shared_ib3
respectively. Finally, deploy the config map using the following command:

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

2. Deploy the device plugin

Now deploy the NVIDIA Mellanox RDMA device plugin:

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

The device plugin 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, e.g.,
rdma/shared_ib0: 1, which enables access the requested RDMA resource. Higher quantities,
like, for example, rdma/shared_ib0: 2 are not possible. Instead, a container can request
access to multiple different RDMA resources at the same time, e.g. rdma/shared_ib0: 1,
dma/shared_ib1: 1, etc.

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 functionality, in particular:

# 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
You can log into the running pod with oc rsh mofed-test-pod and execute regular ibstat, ibhosts, ib_write_bw, etc. commands to verify the connectivity across your IB network interactively.

### 3.4.3 Enabling IPC_LOCK in User Namespace for RDMA Shared Device Plugin

The use of the RDMA device plugin requires the **IPC_LOCK** capability from the Red Hat OpenShift security context. This is not generally available to a regular user who is normally running under the “restricted” Security Context Constraints (SCC) in Red Hat OpenShift 4. The system admin, however, has access to the “privileged” Security Context Constraints (SCC) and can immediately run the mofed-test-pod above.

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

```yaml
defaultAddCapabilities:
  - IPC_LOCK
```

In order to make the new SCC available to a user's namespace, the system admin needs to create a service account, a role binding and role referencing this new SCC. The detailed steps are as follows.

1. Create a new SCC which includes the IPC_LOCK capability:

   We create and apply a new SCC derived from the “restricted” SCC, adding the **IPC_LOCK** Capability and naming it `scc-for-mpi` because we intend to use it when running MPI jobs requesting multiple GPUs:

   ```bash
   # 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 capabilities
   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 new service account in the user's namespace

We create a new service account with 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 new role in the user's namespace

We create a new role with name mpi in the user's namespace that is referring to the newly created security context constraints (SCC) named scc-for-mpi which 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 new role binding in the user's namespace
Finally, we create a *role binding* with name `mpi` in the user’s namespace that connects the *service account* `mpi` with the *role* `mpi`:

```bash
# 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 utilize RDMA resources by adding the service account and the `IPC_LOCK` capability to the pod’s spec section in the YAML:

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

Note that this service account is bound to a given *namespace*. The `mpi` *service account*, *role* and *role binding* needs to be applied to every namespace that requires to run jobs with the `IPC_LOCK` capability and added to the SCC.
3.4.4 MPI Operator

We use the MPI Operator Message Passing Interface (MPI) for distributed AI-workloads to scale out across GPUs and worker nodes. To achieve that we make use of 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 the section 4.3, “MPI Job Definition” on page 31.

3.4.5 IBM Spectrum Scale CSI

IBM Spectrum Scale is a proven and scalable software-defined storage solution from IBM for today's enterprise AI workloads ensuring 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 using the IBM Spectrum Scale CSI plugin. CSI stands for Container Storage Interface and was introduced as alpha in Kubernetes v1.9 and promoted to GA in the Kubernetes v1.13.

IBM Spectrum Scale CSI allows to conveniently provision persistent volumes (PV) in Kubernetes and Red Hat OpenShift with IBM Spectrum Scale as storage backend. These persistent volumes (PVs) can either be dynamically provisioned on a user's request (dynamic provisioning) 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. Users can then request these statically provisioned volumes in a similar way using persistent volume claims (PVCs) and additionally making use of labels to specify exactly which persistent volumes (i.e. which data) they are interested in.

**Note:** In Red Hat OpenShift 4.4.3 we also need to add `LimitMEMLOCK=infinity` to the default system settings of cri-o under 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 (e.g. with `ib_send_bw`) in a non-privileged pod as non-root user fail with

```bash
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 simply add it and restart cri-o with

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

This step might no longer be necessary in future Red Hat OpenShift 4 releases beyond 4.4.3.
The **IBM Spectrum Scale CSI Operator** provides a means to deploy and manage the CSI plugin for IBM Spectrum Scale. The IBM Spectrum Scale CSI plugin requires a running IBM Spectrum Scale cluster with access to the IBM Spectrum Scale GUI. In this paper we are using IBM Spectrum Scale CSI v2.0.0.

You can easily deploy the IBM Spectrum Scale CSI plugin 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](image)

**Figure 3-2** IBM Spectrum Scale CSI plugin can be installed and deployed directly from the Red Hat OperatorHub

For configuring the IBM Spectrum Scale CSI operator, follow the documentation provided in the operator. For more details and the full documentation of IBM Spectrum Scale CSI, refer to the IBM Knowledge Center or take a look at the IBM Redpaper publication **IBM Spectrum Scale CSI Driver for Container Persistent Storage, REDP-5589**.

Follow the instructions at **Performing pre-installation tasks** to prepare your environment accordingly. 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 IBM Spectrum Scale Container Storage Interface driver should run:

```bash
# 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 issuing the following command (this is especially important for Red Hat OpenShift):

```bash
# 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 comprising 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 hosting the remote file system ess3000_4M):

```yaml
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: '21505721748717715'
          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: {}
Preparation and Functional Testing

The following sections provide more details on how to prepare the environment, i.e. configuring persistent volumes and defining the YAML for running MPI jobs. They also provide baseline tests to ensure proper network performance and multi-node multi-GPU communications tests with NVIDIA Collective Communications Library (NCCL). The chapter concludes with initial GPU scaling tests running the ResNet-50 benchmark on synthetic data.

4.1 Test RDMA via the InfiniBand Network

If we haven't done so already during the installation step we should first check that 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 paragraph.

Start `ib_write_bw` in listening mode on one DGX-1 system, here we are using `dgx02`:

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

Then start the following test pod 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:

```bash
# 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,  70 Jul  6 15:52  umad0
  crw-rw-rw-. 1 root root 231,  71 Jul  6 15:52  umad1
  crw-rw-rw-. 1 root root 231,  72 Jul  6 15:52  umad2
  crw-rw-rw-. 1 root root 231,  73 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 : 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 : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"

+ for i in 0 1 2
+ 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 : 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 : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"
Ca : 0x248a0703001ef46e ports 1 "dgx02 HCA-2"
Ca : 0x98039b0300a8b706 ports 1 "Mellanox Technologies Aggregation Node"
Ca : 0xec0d9a03006efef2 ports 1 "dgx01 HCA-3"
Ca : 0xec0d9a030044bda4 ports 1 "dgx01 HCA-4"

+ 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"
You should see the proper information about all the InfiniBand interfaces and also observe a throughput of at least 11,000 MB/s for a single point-to-point InfiniBand EDR connection on a DGX-1 system.
4.2 Prepare Persistent Volumes with IBM Spectrum Scale CSI

The overall amount of data used for model building, model evaluation or inference in Enterprise AI environments is typically huge and multiple users (e.g. data scientists) will 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 at the same time 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 the need to duplicate (copy or move) huge amounts of data needed for model training, model validation, or inference. With its additional features like 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 persistent volume (PV) with IBM Spectrum Scale 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 persistent volumes (PVs) for containers in a self-service manner based on storage classes (see Dynamic Provisioning) as well as
- **static provisioning** (see Static Provisioning) for exposing existing paths and data in IBM Spectrum Scale to containerized workloads in Red Hat OpenShift.

In this Redpaper, we are using **static provisioning** to share access to a given directory named `adas` which is located in the `ess3000_4M` IBM Spectrum Scale file system on the IBM ESS3000. This directory holds all our training data, models and training scripts. You can also think of using different directories and persistent volumes for training data (e.g. 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, **persistent volumes** (Puffs) are bound to a given namespace and cannot be used across different namespaces. So in order to make this directory available to users with different namespaces the system admin needs to create one or more persistent volumes (just adjust the name of the persistent volume accordingly in the following YAML example, (e.g. pv01, pv02, pv03, etc.) depending on how many namespaces require access to the data) referencing this specific path in IBM Spectrum Scale:

```yaml
# oc apply -f nv-pv01.yaml
# cat nv-pv01.yaml
apiVersion: v1
category: PersistentVolume
metadata:
  name: "adas-data-pv01"
lables:
  type: data
dep: adas
spec:
  storageClassName: static
capacity:
  storage: 100Gi
accessModes:
- ReadWriteMany
```

Chapter 4. Preparation and Functional Testing  29
Here the following needs to be specified in the csi: volumeHandle stanza:

- the local IBM Spectrum Scale cluster ID (e.g., 16217308676014575381 as shown by `mmlscluster`)
- the file system UID (e.g., 099B6A7A:5EB99743 as shown by `mmlsfs ess3000_4M --uid`)
- as well as the directory path (e.g., /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 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 persistent volume (PV) with these specific attributes to a persistent volume claim (PVC) referencing these very labels to match.

Once the persistent volume (or a set of these volumes) is created by the system admin a user can request this volume to be bound to his or her namespace by issuing a persistent volume claim (PVC) using the `storageClassName: static` and a `selector` to match the labels `type: data` and `dept: adas`.

```yaml
# 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
```

Once a user successfully binds the persistent volume to his or her namespace it stays bound to the user's namespace until the user deletes the persistent volume claim.

`ReadWriteMany` (RWX) as access mode for the volume ensures that it can be used in multiple pods in the user’s namespace in parallel and across physical worker nodes at the same time. This 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 persistent volume claim 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:

```yaml
spec:
  containers:
  - name: your-container-name
    image: nvcr.io/nvidia/tensorflow:20.03-tf2-py3
```
4.3 MPI Job Definition

We use the *MPI Operator Message Passing Interface* (MPI) for distributed AI workloads to scale out across GPUs and worker nodes. For that we make use of the [MPI Operator project on GitHub](https://github.com/kubeflow/mpi-operator) 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, multi-node AI workload with a single MPI job on a Red Hat OpenShift cluster. In this paper, we are launching deep neural network (DNN) training jobs using the [NVIDIA GPU Cloud (NGC) TensorFlow 2 image](https://ngcr.io/nvidia/tensorflow:20.03-tf2-py3) in the user's namespace with the following MPIJob YAML:

```yaml
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
class: 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
              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
```

```yaml
volumeMounts:
  - name: a2d2-data
    mountPath: /workspace

volumes:
  - name: a2d2-data
    persistentVolumeClaim:
      claimName: adas-data-pvc
      readOnly: false
```

---

volumeMounts:
  - name: a2d2-data
    mountPath: /workspace

volumes:
  - name: a2d2-data
    persistentVolumeClaim:
      claimName: adas-data-pvc
      readOnly: false

---

4.3 MPI Job Definition

We use the *MPI Operator Message Passing Interface* (MPI) for distributed AI workloads to scale out across GPUs and worker nodes. For that we make use of the [MPI Operator project on GitHub](https://github.com/kubeflow/mpi-operator) 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, multi-node AI workload with a single MPI job on a Red Hat OpenShift cluster. In this paper, we are launching deep neural network (DNN) training jobs using the [NVIDIA GPU Cloud (NGC) TensorFlow 2 image](https://ngcr.io/nvidia/tensorflow:20.03-tf2-py3) in the user's namespace with the following MPIJob YAML:

```yaml
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
class: 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
              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
```

---

volumeMounts:
  - name: a2d2-data
    mountPath: /workspace

volumes:
  - name: a2d2-data
    persistentVolumeClaim:
      claimName: adas-data-pvc
      readOnly: false

---

4.3 MPI Job Definition

We use the *MPI Operator Message Passing Interface* (MPI) for distributed AI workloads to scale out across GPUs and worker nodes. For that we make use of the [MPI Operator project on GitHub](https://github.com/kubeflow/mpi-operator) 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, multi-node AI workload with a single MPI job on a Red Hat OpenShift cluster. In this paper, we are launching deep neural network (DNN) training jobs using the [NVIDIA GPU Cloud (NGC) TensorFlow 2 image](https://ngcr.io/nvidia/tensorflow:20.03-tf2-py3) in the user's namespace with the following MPIJob YAML:

```yaml
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
class: 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
              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
```

---

volumeMounts:
  - name: a2d2-data
    mountPath: /workspace

volumes:
  - name: a2d2-data
    persistentVolumeClaim:
      claimName: adas-data-pvc
      readOnly: false

---

4.3 MPI Job Definition

We use the *MPI Operator Message Passing Interface* (MPI) for distributed AI workloads to scale out across GPUs and worker nodes. For that we make use of the [MPI Operator project on GitHub](https://github.com/kubeflow/mpi-operator) 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, multi-node AI workload with a single MPI job on a Red Hat OpenShift cluster. In this paper, we are launching deep neural network (DNN) training jobs using the [NVIDIA GPU Cloud (NGC) TensorFlow 2 image](https://ngcr.io/nvidia/tensorflow:20.03-tf2-py3) in the user's namespace with the following MPIJob YAML:

```yaml
# cat nv-tf2-job-a2d2.yaml
apiVersion: kubeflow.org/v1alpha2
class: 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
              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
          readOnly: false
      - name: dshm
        emptyDir:
          medium: Memory
This MPI job will have one launcher pod executing 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 utilize the additional IPC_LOCK capability of the `scc-for-mpi` security context constraints (SCC). In addition we also specify a user and group ID for the worker pods matching the UID/GID of the data directory.

The NCCL environment variables

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

explicitly enable InfiniBand RDMA and GPUDirect Storage 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 GPUDirect Storage**
  
  ```
  NCCL_IB_DISABLE=0
  NCCL_NET_GDR_LEVEL=0
  ```

- **With GPUDirect Storage**
  
  ```
  NCCL_IB_DISABLE=0
  NCCL_NET_GDR_LEVEL=1
  ```

In addition to the persistent volume claim (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` backed by emptyDir: medium: memory into the container which 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 located. 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 executing 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 1 GPU. We reduced the `max_steps` to 16,000 so that we can run multiple training runs in an acceptable time. The training would typically run with many more steps and last for days and not just hours.

### 4.4 Connectivity Tests with NVIDIA Collective Communications Library

The NVIDIA Collective Communications Library (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 installed in a single node or across multiple nodes and can be used in either single- or multi-process (e.g., MPI) applications. The code with examples is provided through GitHub.

The system admin can quickly run an initial test job as shown below to check GPU communication with NCCL on each DGX-1 worker node (8 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. The output, Example 4-1, of the completed job (e.g., `oc logs job.batch/nv-nccl`) will provide 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, here running on 8 GPUs (-g 8) and scanning from 8 Bytes (-b 8) to 256MBytes (-e 256M).

---

Example 4-1  GPU information, GPU topology, and NCCL measurement results for single-node batch job using 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   | 00000000:08:00.0 Off |                    0 |
| N/A   34C    P0    43W / 300W |      0MiB / 16160MiB |      0%      Default |
+-----------------------------------------------------------------------------+
```
Chapter 4. Preparation and Functional Testing

### GPU Memory

<table>
<thead>
<tr>
<th>GPU</th>
<th>PID</th>
<th>Type</th>
<th>Process name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>SYS</td>
</tr>
<tr>
<td>GPU1</td>
<td>NV1</td>
<td>X</td>
<td>NV2</td>
<td>NV1</td>
</tr>
<tr>
<td>GPU2</td>
<td>NV1</td>
<td>NV2</td>
<td>X</td>
<td>NV2</td>
</tr>
<tr>
<td>GPU3</td>
<td>NV2</td>
<td>NV1</td>
<td>X</td>
<td>SYS</td>
</tr>
<tr>
<td>GPU4</td>
<td>NV2</td>
<td>SYS</td>
<td>SYS</td>
<td>X</td>
</tr>
<tr>
<td>GPU5</td>
<td>SYS</td>
<td>NV2</td>
<td>SYS</td>
<td>X</td>
</tr>
<tr>
<td>GPU6</td>
<td>SYS</td>
<td>SYS</td>
<td>NV1</td>
<td>X</td>
</tr>
<tr>
<td>GPU7</td>
<td>SYS</td>
<td>SYS</td>
<td>SYS</td>
<td>NV1</td>
</tr>
<tr>
<td>mlx5_0</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
</tr>
<tr>
<td>mlx5_1</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
</tr>
<tr>
<td>mlx5_2</td>
<td>SYS</td>
<td>SYS</td>
<td>SYS</td>
<td>PIX</td>
</tr>
<tr>
<td>mlx5_3</td>
<td>SYS</td>
<td>SYS</td>
<td>SYS</td>
<td>PHB</td>
</tr>
</tbody>
</table>

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

### Benchmark Results

<table>
<thead>
<tr>
<th>size (B)</th>
<th>count (elements)</th>
<th>type</th>
<th>redop</th>
<th>time (us)</th>
<th>algbw (GB/s)</th>
<th>busbw (GB/s)</th>
<th>error (GB/s)</th>
<th>time (us)</th>
<th>algbw (GB/s)</th>
<th>busbw (GB/s)</th>
<th>error (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>float</td>
<td>sum</td>
<td>40.92</td>
<td>0.00</td>
<td>0.00</td>
<td>2e-07</td>
<td>44.50</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>float</td>
<td>sum</td>
<td>38.08</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
<td>40.94</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>float</td>
<td>sum</td>
<td>41.30</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
<td>39.27</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>float</td>
<td>sum</td>
<td>42.10</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
<td>38.83</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>float</td>
<td>sum</td>
<td>42.10</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
<td>43.15</td>
<td>0.00</td>
<td>0.00</td>
<td>6e-08</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>float</td>
<td>sum</td>
<td>41.70</td>
<td>0.01</td>
<td>0.01</td>
<td>6e-08</td>
<td>43.80</td>
<td>0.01</td>
<td>0.01</td>
<td>6e-08</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>float</td>
<td>sum</td>
<td>38.83</td>
<td>0.01</td>
<td>0.02</td>
<td>6e-08</td>
<td>44.29</td>
<td>0.01</td>
<td>0.02</td>
<td>6e-08</td>
</tr>
</tbody>
</table>
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 overhead (or latency) associated with operations, while *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 (us) in a setup like ours with 100Gbps EDR InfiniBand as shown here. Should you see latencies in the range of thousands of micro-seconds then your configuration is probably not using InfiniBand but Ethernet instead. In this case, you need to 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 using 8 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
```

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 overhead (or latency) associated with operations, while *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 (us) in a setup like ours with 100Gbps EDR InfiniBand as shown here. Should you see latencies in the range of thousands of micro-seconds then your configuration is probably not using InfiniBand but Ethernet instead. In this case, you need to 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 using 8 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
    readOnly: false
- name: dshm
  emptyDir:
    medium: Memory

The results should be similar to the ones seen from the batch job above (e.g. nv-nccl-batch-job.yaml).

Next we can run a multi-node MPI job with 2 pods on two DGX-1 worker nodes each of them using 8 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
category: 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
and verify that we still see acceptable latencies for small scan sizes in the 2-digit microseconds (us) 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 distributed across two DGX-1 worker nodes (16 GPUs total)

<table>
<thead>
<tr>
<th>Using devices</th>
<th>out-of-place</th>
<th>in-place</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td># Rank 0 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 0 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 1 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 2 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 3 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 4 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 5 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 6 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 7 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 8 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 9 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 10 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 11 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 12 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 13 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 14 Pid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 15 Pid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Using devices</th>
<th>out-of-place</th>
<th>in-place</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td># size</td>
<td>count</td>
<td>type</td>
</tr>
<tr>
<td>(B) [elements]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nccl-test-mpi-worker-0:30</td>
<td>NCCL INFO Launch mode Group/CGMD</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2   float</td>
<td>sum</td>
</tr>
<tr>
<td>16</td>
<td>4   float</td>
<td>sum</td>
</tr>
<tr>
<td>32</td>
<td>8   float</td>
<td>sum</td>
</tr>
<tr>
<td>64</td>
<td>16  float</td>
<td>sum</td>
</tr>
<tr>
<td>128</td>
<td>32  float</td>
<td>sum</td>
</tr>
<tr>
<td>256</td>
<td>64  float</td>
<td>sum</td>
</tr>
<tr>
<td>512</td>
<td>128 float</td>
<td>sum</td>
</tr>
<tr>
<td>1024</td>
<td>256 float</td>
<td>sum</td>
</tr>
<tr>
<td>2048</td>
<td>512 float</td>
<td>sum</td>
</tr>
<tr>
<td>4096</td>
<td>1024 float</td>
<td>sum</td>
</tr>
<tr>
<td>8192</td>
<td>2048 float</td>
<td>sum</td>
</tr>
<tr>
<td>16384</td>
<td>4096 float</td>
<td>sum</td>
</tr>
<tr>
<td>32768</td>
<td>8192 float</td>
<td>sum</td>
</tr>
<tr>
<td>65536</td>
<td>16384 float</td>
<td>sum</td>
</tr>
<tr>
<td>131072</td>
<td>32768 float</td>
<td>sum</td>
</tr>
<tr>
<td>262144</td>
<td>65536 float</td>
<td>sum</td>
</tr>
<tr>
<td>524288</td>
<td>131072 float</td>
<td>sum</td>
</tr>
<tr>
<td>1048576</td>
<td>262144 float</td>
<td>sum</td>
</tr>
<tr>
<td>2097152</td>
<td>524288 float</td>
<td>sum</td>
</tr>
<tr>
<td>4194304</td>
<td>1048576 float</td>
<td>sum</td>
</tr>
<tr>
<td>8388608</td>
<td>2097152 float</td>
<td>sum</td>
</tr>
<tr>
<td>16777216</td>
<td>4194304 float</td>
<td>sum</td>
</tr>
<tr>
<td>33554432</td>
<td>8388608 float</td>
<td>sum</td>
</tr>
<tr>
<td>67108864</td>
<td>16777216 float</td>
<td>sum</td>
</tr>
<tr>
<td>134217728</td>
<td>33554432 float</td>
<td>sum</td>
</tr>
<tr>
<td>268435456</td>
<td>67108864 float</td>
<td>sum</td>
</tr>
</tbody>
</table>

# Out of bounds values : 0 OK
# Avg bus bandwidth : 12.3088

Again, should we observe latencies in the thousands of micro-seconds range then your configuration is probably not using InfiniBand RDMA and you need to look into the job's log to check if InfiniBand connections are properly established and used for communication queues. Watch for NCCL INFO NET/IB messages like:
to ensure that RDMA ports (e.g., [0]mlx5_3:1/IB, …,) are discovered and used by NCCL as available communication paths.

Instead of running two pods with 8 GPUs, it is often more convenient to assign only 1 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 simply by the number of worker pods assigned to the job.

**Note:** Here we assume that we will only have 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 using multiple containers in the same pod.

This 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 (giving 16 GPUs in total for the job):

```bash
# 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
```

nccl-test-mpi-worker-1:30:30 [0] NCCL INFO NET/IB : Using [0]mlx5_3:1/IB
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
        readOnly: false
    - name: dshm
      emptyDir:
medium: Memory

The log of the successfully completed job will show NCCL messages like

```
ncc1-test-mpi-worker-2:30:30 [0] NCCL INFO NET/IB : Using [0] mlx5_3:1/IB
ncc1-test-mpi-worker-11:30:39 [0] NCCL INFO Ring 00 : 10[89000] -> 11[89000]
[receive] via NET/IB/0/GDRDMA
```

indicating that InfiniBand connections (e.g., mlx5_3, mlx5_2, mlx5_1, mlx5_0) are
successfully used for NCCL communication paths and that GPU Direct Storage RDMA is
enabled (GDRDMA). The NCCL results, Example 4-3, of this configuration will 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 evenly distributed
across two DGX-1 worker nodes (16 GPUs total)

<table>
<thead>
<tr>
<th># Using devices</th>
</tr>
</thead>
<tbody>
<tr>
<td># Rank 0 Pid 30 on nccl-test-mpi-worker-0 device 0 [0x85] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 1 Pid 30 on nccl-test-mpi-worker-1 device 0 [0x08] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 2 Pid 30 on nccl-test-mpi-worker-2 device 0 [0x07] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 3 Pid 30 on nccl-test-mpi-worker-3 device 0 [0x04] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 4 Pid 30 on nccl-test-mpi-worker-4 device 0 [0x06] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 5 Pid 30 on nccl-test-mpi-worker-5 device 0 [0x86] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 6 Pid 31 on nccl-test-mpi-worker-6 device 0 [0x04] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 7 Pid 30 on nccl-test-mpi-worker-7 device 0 [0x06] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 8 Pid 30 on nccl-test-mpi-worker-8 device 0 [0x8a] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 9 Pid 30 on nccl-test-mpi-worker-9 device 0 [0x8a] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 10 Pid 30 on nccl-test-mpi-worker-10 device 0 [0x89] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 11 Pid 30 on nccl-test-mpi-worker-11 device 0 [0x89] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 12 Pid 30 on nccl-test-mpi-worker-12 device 0 [0x0b] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 13 Pid 30 on nccl-test-mpi-worker-13 device 0 [0x8a] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 14 Pid 30 on nccl-test-mpi-worker-14 device 0 [0x8a] Tesla V100-SXM2-16GB</td>
</tr>
<tr>
<td># Rank 15 Pid 30 on nccl-test-mpi-worker-15 device 0 [0x07] Tesla V100-SXM2-16GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>size</th>
<th>count</th>
<th>type</th>
<th>redop</th>
<th>time (us)</th>
<th>algbw (GB/s)</th>
<th>busbw (GB/s)</th>
<th>error</th>
<th>time (us)</th>
<th>algbw (GB/s)</th>
<th>busbw (GB/s)</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>78.92</td>
<td>0.00</td>
<td>0.00</td>
<td>0e-07</td>
<td>72.73</td>
<td>0.00</td>
<td>0.00</td>
<td>0e-07</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>72.27</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
<td>77.27</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>73.74</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
<td>72.11</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>82.74</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
<td>81.37</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>83.59</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
<td>79.80</td>
<td>0.00</td>
<td>0.00</td>
<td>1e-07</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>78.04</td>
<td>0.00</td>
<td>0.01</td>
<td>6e-08</td>
<td>78.35</td>
<td>0.00</td>
<td>0.01</td>
<td>6e-08</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>78.49</td>
<td>0.01</td>
<td>0.01</td>
<td>6e-08</td>
<td>83.42</td>
<td>0.01</td>
<td>0.01</td>
<td>6e-08</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>88.29</td>
<td>0.01</td>
<td>0.02</td>
<td>2e-07</td>
<td>91.61</td>
<td>0.01</td>
<td>0.02</td>
<td>2e-07</td>
</tr>
<tr>
<td>2048</td>
<td>512</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>107.5</td>
<td>0.02</td>
<td>0.04</td>
<td>5e-07</td>
<td>109.6</td>
<td>0.02</td>
<td>0.04</td>
<td>5e-07</td>
</tr>
<tr>
<td>4096</td>
<td>1024</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>121.4</td>
<td>0.03</td>
<td>0.06</td>
<td>5e-07</td>
<td>116.4</td>
<td>0.04</td>
<td>0.07</td>
<td>5e-07</td>
</tr>
<tr>
<td>8192</td>
<td>2048</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>122.8</td>
<td>0.07</td>
<td>0.13</td>
<td>5e-07</td>
<td>121.9</td>
<td>0.07</td>
<td>0.13</td>
<td>5e-07</td>
</tr>
<tr>
<td>16384</td>
<td>4096</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>143.3</td>
<td>0.11</td>
<td>0.21</td>
<td>5e-07</td>
<td>135.9</td>
<td>0.12</td>
<td>0.23</td>
<td>5e-07</td>
</tr>
<tr>
<td>32768</td>
<td>8192</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>152.4</td>
<td>0.22</td>
<td>0.40</td>
<td>5e-07</td>
<td>158.5</td>
<td>0.21</td>
<td>0.39</td>
<td>5e-07</td>
</tr>
<tr>
<td>65536</td>
<td>16384</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>193.0</td>
<td>0.34</td>
<td>0.64</td>
<td>5e-07</td>
<td>189.7</td>
<td>0.35</td>
<td>0.65</td>
<td>5e-07</td>
</tr>
<tr>
<td>131072</td>
<td>32768</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>233.7</td>
<td>0.56</td>
<td>1.05</td>
<td>5e-07</td>
<td>220.0</td>
<td>0.60</td>
<td>1.12</td>
<td>5e-07</td>
</tr>
<tr>
<td>262144</td>
<td>65536</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>313.7</td>
<td>0.84</td>
<td>1.57</td>
<td>5e-07</td>
<td>326.3</td>
<td>0.80</td>
<td>1.51</td>
<td>5e-07</td>
</tr>
<tr>
<td>524288</td>
<td>131072</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>594.7</td>
<td>0.88</td>
<td>1.65</td>
<td>5e-07</td>
<td>598.3</td>
<td>0.88</td>
<td>1.64</td>
<td>5e-07</td>
</tr>
<tr>
<td>1048576</td>
<td>262144</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>754.3</td>
<td>1.39</td>
<td>2.61</td>
<td>5e-07</td>
<td>769.7</td>
<td>1.36</td>
<td>2.55</td>
<td>5e-07</td>
</tr>
<tr>
<td>2097152</td>
<td>524288</td>
<td>float</td>
<td>sum</td>
<td></td>
<td>1068.4</td>
<td>1.96</td>
<td>3.68</td>
<td>5e-07</td>
<td>1072.8</td>
<td>1.95</td>
<td>3.67</td>
<td>5e-07</td>
</tr>
</tbody>
</table>
4.5 Multi-GPU, Multi-Node GPU Scaling with TensorFlow ResNet-50 Benchmark

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

AI workloads may or may not scale equally well with the number of GPUs used in parallel. This depends heavily on the infrastructure, the model and the implementation. The TensorFlow ResNet-50 benchmark with synthetic data (in the following 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 have made available in Red Hat OpenShift through a statically provisioned persistent volume (PV) that can be mounted into each MPI worker pod as described in the section 4.2, “Prepare Persistent Volumes with IBM Spectrum Scale CSI” on page 29.

First, we schedule MPI jobs with multi-GPU pods using 1, 2, 4, 8 GPUs in a single worker pod on a single DGX-1 system and finally 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):

```yaml
# 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
In Figure 4-1 we can see that the number of images/second increases accordingly (with some scaling costs) when we scale multi-GPU pods from 1 to 8 GPUs with 7096 images/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/second.

Each column in Figure 4-1 (and in Figure 4-2) represents the median value of 5 measurements for the given 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 sub-optimal resource utilization across all the resources in large clusters as such a pod will only be scheduled and started when 4 or 8 GPUs as requested are actually 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 will lead 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 using 1, 2, 4, 8 GPUs and the same number of single-GPU worker pods, respectively, 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
              readOnly: false
          - name: dshm
            emptyDir:
              medium: Memory

In Figure 4-2 we can see that the number of images/second increases steadily to 5918 images/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/second. Especially with 8 single-GPU pods on a single DGX-1 system (i.e. the 8-GPU and 16-GPU cases) when going for the limit of a single DGX-1 system with 8 GPUs, we experience slightly higher costs in overall performance using single-GPU pods compared to multi-GPU pods. The 1, 2, and 4GPU cases, however, 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 a given AI workload using MPI jobs. Single-GPU pods provide a finer granularity when scheduling jobs and allocating resources in large clusters and thus will lead to a better overall resource utilization across all compute nodes which may well justify the slightly higher performance costs that come with single-GPU worker pods at full utilization compared to multi-GPU worker pods.
Deep neural network (DNN) Training on A2D2 Semantic Segmentation Dataset

The following section provides details about the multi-GPU multi-node training with the A2D2 dataset we performed.

5.1 Description A2D2 Dataset

Audi recently published the Autonomous Driving Dataset (A2D2) which can be used to support academic institutions and commercial start-ups working on autonomous driving research (A2D2 - Jacob Geyer et al, 2020 for more A2D2 Dataset license details see “Related publications” on page 63). The dataset consists of recorded images and labels like bounding boxes, semantic segmentation, instance segmentation, and data extracted from the automotive bus. The sensor suite consists of six cameras and five LIDAR units, providing 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 there are 12,497 frames which have 3D bounding box annotations for objects within the field of view of the front camera.

The semantic segmentation dataset features 38 categories. Each pixel in an image is given a label describing the type of object it represents, e.g. pedestrian, car, vegetation, etc.

Figure 5-1 shows an example of a real picture compared to the segmentation picture.
5.2 Multi-GPU, Multi-Node GPU Scaling Results for DNN Training Jobs

We perform a multi-GPU multi-node training with the A2D2 dataset and train a segmentation network. Not all classes are considered and the training dataset 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 a size of ~27,300 image and label pairs. This is roughly one 10th of a representative dataset, which usually contains more than 300,000 well selected frames per network and task. The subset used for the training in this paper has a total size of 92 GB.

The training MPI job is utilizing 16 GPUs on both DGX-1 systems and is executed 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
    - s1ot
    - -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=checkpoint
    - --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" ]
The job is based on the MPI job YAML described in 4.3, “MPI Job Definition” on page 31. The job schedules 16 single-GPU pods for the training and is using the 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 CSI with a statically provisioned volume as created in 4.2, “Prepare Persistent Volumes with IBM Spectrum Scale CSI” on page 29, 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 distributed parallel file system all worker pods have shared read/write access to the same data in IBM Spectrum Scale across physical node boundaries.

We compare a multi-GPU training run using a single MPI job with 8 single-GPU pods on a single DGX-1 node (setting Launcher: np=8 and Worker: replicas=8 in the above YAML) with a training job using 16 single-GPU pods on two DGX-1 nodes utilizing Red Hat OpenShift 4.4.3 as 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 (8 GPUs) and on two DGX-1 nodes (16 GPUs) in Red Hat OpenShift

By scaling the training job from a single DGX-1 node (8 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 job runtime from 1328 seconds down to 740 seconds (55.7%). This results in a speed up of 1.8x (i.e. 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 conveniently and efficiently scale out these workloads across GPU resources 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 DNN training jobs works seamlessly and efficiently in Red Hat OpenShift 4.4.3 as container orchestration platform with DGX worker nodes, NVIDIA Mellanox InfiniBand RDMA network infrastructure, and IBM Spectrum Scale as backend storage for a scalable high-performance “data lake”.

5.3 Application

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

Especially the validation of a trained neural network in a variety of relevant situations provides confidence about its performance but also valuable insights on its weaknesses. Multiple validation datasets 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 dataset is crucial to build the necessary datasets and understand neural network strength and weaknesses.

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

In following we name a few samples where a metadata store helps to derive steps for developing the dataset or network:

- E.g., the DNN under test performs well on large objects and worse on smaller ones given the object occurrence is balanced. This also happened with the randomly chosen subset of our training dataset. As shown in Figure 5-3 we have evaluated a training after just 20,000 steps using a confusion matrix. For a perfect predictor, the diagram would only have a diagonal line from top left to bottom right. That would read as the network would have classified all pixels of a certain class right.

Figure 5-3 shows a confusion matrix presenting the training evaluation. Note that there is a logarithmic scale chosen for the color coding.

![Figure 5-3 Confusion matrix presenting the training evaluation](image)

There are two obvious observations:

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

  ```
  https://localhost/db2whrest/v1/sql_query -X POST -d
  select sum(int(t_rd_normal_street.value)) /
  ```
[((count(t_rd_normal_street.value)*1920*1208)/100) from t_rd_normal_street with ur;  
Result:  
22.22743915175 
In contrast, other classes such as “Road Blocks” appear with only 0.85 percent and frequency of object occurrence is not sufficient. 
– Furthermore, it is obvious that the data used for the evaluation has no representatives for bicycle class (19). Thus the validation dataset needs to be extended. A query like the following can help to select frames which are know to contain the required class:  
https://localhost/db2whrest/v1/sql_query -X POST -d select 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  
Result:  
0,"IBM COS","a2d2","camera_lidar_semantic/20181107_133445/camera/cam_front_center/20181107133445_camera_frontcenter_000021093.png"  
1,"IBM COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/20181107132300_camera_frontcenter_000001645.png"  
2,"IBM COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/20181107132300_camera_frontcenter_000004159.png"  
3,"IBM COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/20181107132300_camera_frontcenter_000002707.png"  
4,"IBM COS","a2d2","camera_lidar_semantic/20181107_132300/camera/cam_front_center/20181107132300_camera_frontcenter_000004093.png"  
... 
> The DNN performs poorly in a certain case, e.g. close to bridges and the lighting could be a source of such a weakness. A join by location of a map database together with the used training data would allow such a query. 
> The DNN performs not as expected on a specific class though it has plenty of training data. Here the dataset might be too simple and not as complicated as the validation dataset. A selection of those frames with an active learning uncertainty estimate would allow to reduce the dataset 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 dataset without the help of other data sources. On uploading the dataset, the available information was extracted, such as 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 e.g. 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 a large amount of attributes. Likely the time will come that one of those attributes is needed. Data minimalism on the metadata is not encouraged. Thus it is more practical to keep all of them rather to repeatedly suffer from missing attributes and add them with high effort.
Here, we leverage IBM Spectrum Discover as a metadata database. It works together with IBM storage solutions and used as a metadata store in the context of this work. Metadata information for each frame of the A2D2 dataset is created in IBM Spectrum Discover to show how it can be used in that context.

Building a representative validation dataset is a challenge as well. Validation datasets are usually much larger for autonomous vehicles than their training dataset.

Figure 5-4 shows the process of building a representative validation dataset.

![Figure 5-4 Building a representative validation dataset](image)

Training and validation of the neural networks are done on a very frequent basis. Validating the trained DNN against a large validation dataset is critical to understand its weaknesses. For representative results the validation dataset is significantly larger than the training dataset and reflects the distribution of the target operational domain. Large scale deep neural networks (DNN) validation is an inference job which is rolled out into the cluster in a fan-out manner to multiple workers. It follows a map-reduce 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 right 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 right 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 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, as we added IBM Spectrum Discover after the data was stored in the storage system. Spectrum Discover comes with built-in functionality that can react on new incoming data and automatically detects metadata for the new data and catalogs it.

To find the data needed for model training or inference based on label details a user could run a simple REST query in SQL format against IBM Spectrum Discover.

```sql
https://localhost/db2whrest/v1/sql_query -X POST -d select 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 could also work with a customized job user interface panel created in IBM Spectrum LSF.

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

![IBM Spectrum LSF AV job submission template example](image)

As a user, start a job by selecting the needed labels as shown in Figure 5-5. 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 shown in the following example.
IBM Spectrum Discover is able to catalog multiple different storage systems and returns the platform and data source with the data details.

With the help of IBM Spectrum Scale's Active File Management (AFM), the data can be pre-cached close to the AI workloads. Pre-caching helps to keep the Accelerators busy as the to-be-analyzed data is present at the right time, even if rather “slow” storage systems / data lakes host the data. And it helps to ensure that high performing storage is not over utilized 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.
Related publications

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

Online resources

These websites are also relevant as further information sources:

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

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

- NVIDIA Mellanox
  
  https://docs.mellanox.com/pages/releaseview.action?pageId=19804150

- IBM Automotive 2030
  
  https://www.ibm.com/downloads/cas/NWDQPK5B

- IBM Spectrum Scale
  

- IBM Spectrum Scale 5.0.4 Knowledge Center
  
  https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.4/ibmspectrumscale504_welcome.html

- IBM ESS3000
  

- IBM Elastic Storage System (ESS) 3000 Version 6.0.0 Knowledge Center
  

- IBM Spectrum Storage for AI with DGX Systems
  
  https://www.ibm.com/downloads/cas/MNEQGQVP

- 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

- Red Hat Enterprise Linux CoreOS (RHCOS)
  

- A2D2: Audi Autonomous Driving Dataset 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 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 dataset is released under the CC BY-ND 4.0 license. See:  
 https://creativecommons.org/licenses/by-nd/4.0/

– Liability and Copyright (Licensed Material was not modified for this study)  

– Driving Dataset Downloads and Citation  
 https://www.a2d2.audi/a2d2/en/download.html

– A2D2: Audi Autonomous Driving Dataset paper  

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 with fully integrated hardware and software that can be deployed quickly and easily. DGX-1 features 8 NVIDIA V100 GPU accelerators with Tensor Core architecture connected through NVIDIA NVLink, 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 deep learning training.

The successor system called NVIDIA DGX A100 was announced on June 8th 2020. It is planned 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. NVIDIA DGX A100 system features the world's most advanced accelerator, the NVIDIA A100 Tensor Core GPU, enabling 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 details visit the NVIDIA DGX A100 system whitepaper.

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

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services