Dell EMC Ready Solutions for AI – Deep Learning with NVIDIA

Deep Learning with NVIDIA Architecture Guide

Authors: Rengan Xu, Frank Han, Quy Ta

Abstract

There has been an explosion of interest in Deep Learning and the plethora of choices makes designing a solution complex and time consuming. Dell EMC's Ready Solutions for AI – Deep Learning with NVIDIA is a complete solution, designed to support all phases of Deep Learning, incorporates the latest CPU, GPU, memory, network, storage, and software technologies with impressive performance for both training and inference phases. The architecture of this Deep Learning solution is presented in this document.

February 2019

Revisions

Date	Description
February 2019	Release 1.1
August 2018	Initial release

Table of Contents

Revisions....2

Table of Contents....3

Executive summary....4

1 Solution Overview....5

2 Solution Architecture....7

2.1 Head Node Configuration....7

2.1.1 Shared Storage via NFS over InfiniBand....8

2.2 Compute Node Configuration....8

2.2.1 GPU....9

2.3 Processor recommendation for Head Node and Compute Nodes....10

2.4 Memory recommendation for Head Node and Compute Nodes....10

2.5 Isilon Storage....11

2.6 Network....12

2.7 Software....13

2.8 NVIDIA DIGITS Tool and the Deep Learning Solution....13

3 Containers for Deep Learning....15

3.1 Singularity Containers....15

3.2 Running NVIDIA GPU Cloud with the Ready Solutions for AI - Deep Learning....16

4 The Data Science Provisioning Portal....20

4.1 Creating and Running a Notebook....20

4.2 TensorBoard Integration....24

4.3 Slurm Scheduler....25

5 Conclusions....28

Executive summary

Deep Learning techniques have enabled great success in many fields such as computer vision, natural language processing (NLP), gaming and autonomous driving by enabling a model to learn from existing data and then to make corresponding predictions. The success is due to a combination of improved algorithms, access to large datasets and increased computational power. To be effective at enterprise scale, the computational intensity of Deep Learning neural network training requires highly powerful and efficient parallel architectures. The choice and design of the system components are carefully selected and tuned for Deep Learning use-cases which can impact business outcomes of applying Deep Learning techniques. In addition to several options for processors, accelerators and storage technologies, there are multiple Deep Learning software frameworks and libraries that must be considered. These software components are under active development, updated frequently and cumbersome to manage. It is complicated to simply build and run Deep Learning applications successfully, leaving little time for focus on the actual business problem.

To resolve this complexity challenge, Dell EMC has developed an architecture for Deep Learning that provides a complete, validated and supported solution. This solution includes carefully selected technologies across all aspects of Deep Learning, processing capabilities, memory, storage and network technologies as well as the software ecosystem. This document presents the architecture of this Deep Learning solution including details on the design choice for each component.

AUDIENCE

This document is intended for organizations interested in accelerating Deep Learning with advanced computing and data management solutions. Solution architects, system administrators and others interested readers within those organizations constitute the target audience.

1 Solution Overview

Dell EMC has developed an architecture for Deep Learning that provides a complete, supported solution. This solution includes carefully selected technologies across all aspects of Deep Learning, processing capabilities, memory, storage and network technologies as well as the software ecosystem. This complete solution is provided as Dell EMC's Ready Solutions for AI – Deep Learning with NVIDIA. The solution includes fully integrated and optimized hardware, software, and services including deployment, integration and support making it easier for organizations to start and grow their Deep Learning practice.

The high level overview of Dell EMC Ready Solutions for AI - Deep Learning is shown in Figure 1.

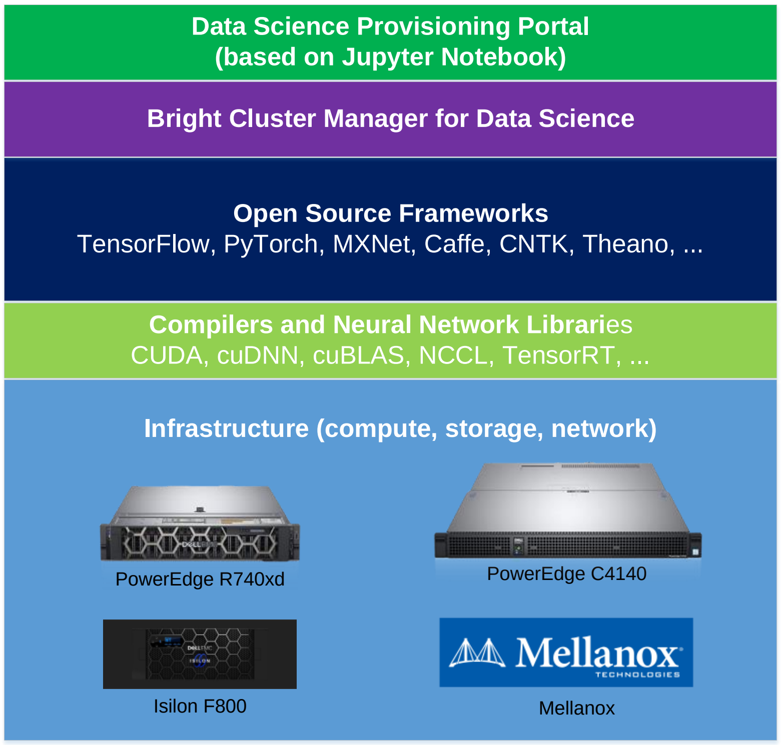

Data Science Provisioning Portal
(based on Jupyter Notebook)
Bright Cluster Manager for Data Science
Open Source Frameworks
TensorFlow, PyTorch, MXNet, Caffe, CNTK, Theano, ...
Compilers and Neural Network Libraries
CUDA, cuDNN, cuBLAS, NCCL, TensorRT, ...
Infrastructure (compute, storage, network)
PowerEdge R740xd
PowerEdge C4140
Isilon F800
Mellanox

Figure 1: Overview of Dell EMC Ready Solutions for AI - Deep Learning

Data Science Provisioning Portal: This is a new portal for data scientists created in this solution. It enables data scientists, who do not have expertise in cluster technologies use a simple web portal to take advantage of the underlying technology. The scientists can write, train and do inference for different Deep Learning models within Jupyter Notebook which includes Python 2, Python 3, R and other kernels**.**

Bright Cluster Manager for Data Science: Bright Cluster Manager is used for the monitoring, deployment, management, and maintenance of the cluster. This software provides the deep learning frameworks, libraries, and compilers and dependency rpms for supported configuration.

Deep Learning Frameworks and Libraries: This category includes TensorFlow, PyTorch, MXNet, Caffe, CUDA, cuDNN, and NCCL. The stable version of these frameworks and libraries are integrated into the solution.

Infrastructure: The infrastructure comprises of a cluster with a master node, compute nodes, shared storage and network. In this instance of the solution, the master node is a Dell EMC PowerEdge R740xd, each compute node is PowerEdge C4140 with NVIDIA Tesla GPUs, the storage includes Network File System (NFS) and Isilon, and the networks include Ethernet and Mellanox InfiniBand.

Section 2 describes each of these solution components in more detail, covering the compute, network, storage and software configurations. That is followed by Section 3 that describes containerization techniques for Deep Learning. Section 4 has details on the Data Science Provisioning Portal developed by Dell EMC. Section 5 gives the conclusions.

2 Solution Architecture

The hardware for this solution comprises of a cluster with a master node, compute nodes, shared storage and networks. The master node often called as head node is used for deploying the cluster of compute nodes, managing the compute nodes images, user logins and access, providing a compilation environment as modules, and tools to support job submissions to compute nodes. The compute nodes are the work horse and execute the submitted jobs. Software from Bright Computing called Bright Cluster Manager is used to deploy and manage the whole cluster.

Figure 2 shows the high-level overview of the cluster which includes one head node, *n* compute nodes, the local disks on the cluster head node exported over NFS, Isilon storage, and two networks. All compute nodes are interconnected through an InfiniBand switch. The head node is also connected to the InfiniBand switch as it needs to access the Isilon storage when included and uses IPoIB to export the scratch space of NFS share to the compute nodes. All compute nodes and the head node are also connected to a 1 Gigabit Ethernet management switch which is used for in-band and out of band management via iDRAC9 (Integrated Dell Remote Access Controller) as well as provisioning and deployment network by Bright Cluster Manager to administer the cluster. An Isilon storage solution is connected to the FDR-40GigE Gateway switch so that it can be accessed by the head node and all compute nodes.

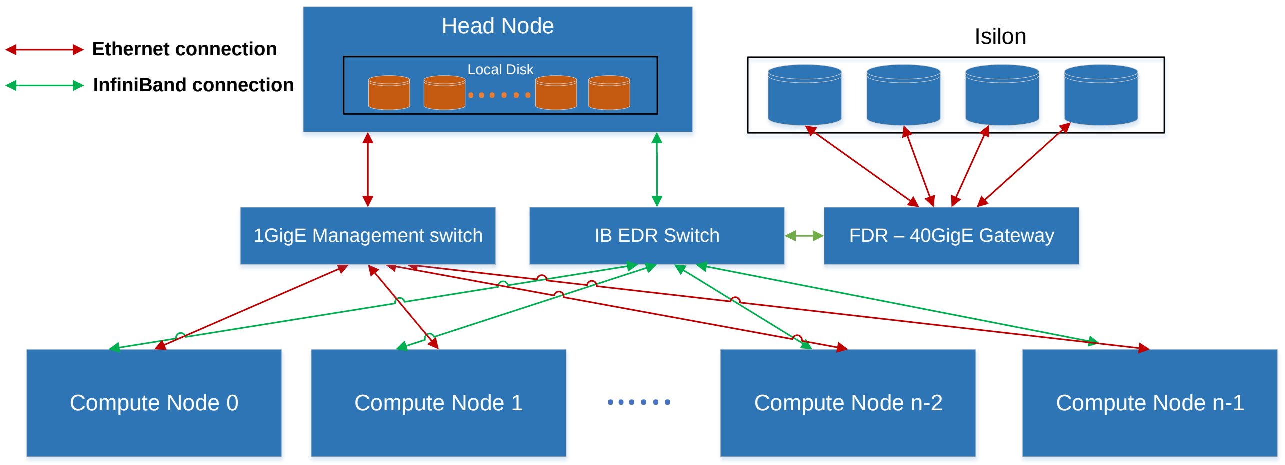


Figure 2: The overview of the cluster

2.1 Head Node Configuration

The Dell EMC PowerEdge R740xd is recommended for the role of the head node. This is Dell EMC's latest two socket, 2U rack server that can support the memory capacities, I/O needs and network options required of the head node. The head node will perform the cluster administration, cluster management, NFS server, user login node and compilation node roles.

The suggested configuration of the PowerEdge R740xd is listed in Table 1. It includes 12 x 12TB NL SAS local disks that are formatted as an XFS file system and exported via NFS to the compute nodes over Ethernet, and the scratch space of the local disks are exported via NFS to the compute nodes over IPoIB. RAID 50 is used to build this solution and is considered due to the faster rebuild time and capacity advantages. Details of each configuration choice are described in the following sections. For more information on this server model please refer to PowerEdge R740/740xd Technical Guide.

Table 1: PowerEdge R740xd configurations

Component	Details
Server Model	PowerEdge R740xd
Processor	2 x Intel Xeon Gold 6148 CPU @ 2.40GHz
Memory	24 x 16GB DDR4 2666MT/s DIMMs - 384GB
Disks	12 x 12TB NL SAS RAID 50
I/O & Ports	Network daughter card with 2 x 10GE + 2 x 1GE
Network Adapter	1x InfiniBand EDR adapter
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller
Power Supplies	Titanium 1100W, Platinum
Storage Controllers	PowerEdge RAID Controller (PERC) H730p

2.1.1 Shared Storage via NFS over InfiniBand

The default shared storage system for the cluster is provided over NFS. It is built using 12x 12TB NL SAS disks that are local to the head node configured in RAID 50 with two parity check disks. This provides usable capacity of 120TB (109TiB). RAID 50 was chosen because it has balanced performance and shorter rebuild time compared to RAID 6 or RAID 60 (and also since RAID 50 has fewer parity disks than RAID 6 or RAID 60). This 120TB volume is formatted as an XFS file system and exported to the compute nodes via NFS.

In the default configuration, both home directories and shared application and library install locations are hosted on this NFS share and they are exported to the compute nodes over Ethernet. The scratch space of the local disks is exported via NFS through IPoIB. The scratch space stores the users' data files. In addition to this, for solutions which require a larger capacity shared storage solution, the Isilon F800 is as an alternative option and is described in Section 2.5.

2.2 Compute Node Configuration

Deep Learning methods would not have gained success without the computational power to drive the iterative training process. Therefore, a key component of Deep Learning solutions is highly capable nodes that can support compute intensive workloads. The state-of-the-art neural network models in Deep Learning have more than 100 layers which require the computation to be able to scale across many compute nodes in order for any timely results. The Dell EMC PowerEdge C4140, an accelerator-optimized, high density 1U rack server, is used as the compute node unit in this solution. The PowerEdge C4140 server used is Configuration M, which supports four NVIDIA Volta SMX2 GPUs. Figure 3 shows the CPU-GPU and GPU-GPU connection topology of a compute node.

The detailed configuration of each PowerEdge C4140 compute node is listed in Table 2.

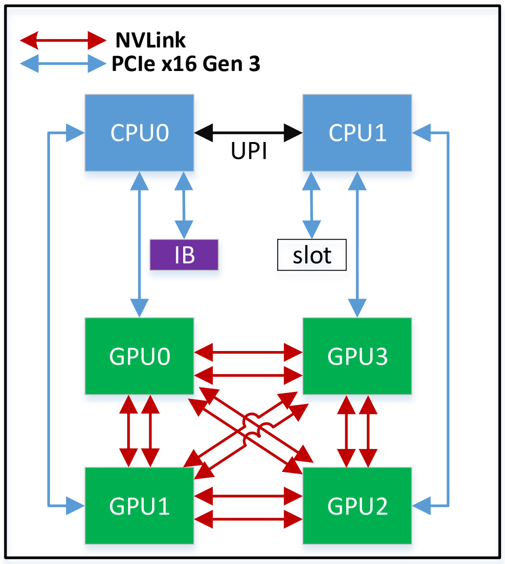


Figure 3: The topology of a compute node

Table 2: PowerEdge C4140 Configuration M

Component	Details
Server Model	PowerEdge C4140 Configuration M
Processor	2 x Intel Xeon Gold 6148 CPU @ 2.40GHz
Memory	24 x 16GB DDR4 2666MT/s DIMMs - 384GB
Local Disks	2 x 240GB SSD
I/O & Ports	Network daughter card with 2 x 10GE + 2 x 1GE
Network Adapter	1 x InfiniBand EDR adapter
GPU	4 x V100-SXM2-16GB or 4 x V100-SXM2-32GB
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller
Power Supplies	2400W hot-plug Redundant Power Supply Unit

2.2.1 GPU

The NVIDIA Tesla V100 is the latest data center GPU available to accelerate Deep Learning. Powered by NVIDIA Volta™, the latest GPU architecture, Tesla V100 GPUs enable data scientists, researchers, and engineers to tackle challenges that were once difficult. With 640 Tensor Cores, Tesla V100 is the first GPU to break the 100 teraflops (TFLOPS) barrier of Deep Learning performance.

Tesla V100 product line includes two variations, V100-PCIe and V100-SXM2. The comparison of two variants is shown in Table 3.

Table 3: V100-SXM2 vs V100-PCIe

Description	V100-PCIe	V100-SXM2
CUDA Cores	5120	5120
GPU Max Clock Rate (MHz)	1380	1530
Tensor Cores	640	640
GPU Memory	16GB or 32GB	16GB or 32GB
Memory Bandwidth (GB/s)	900	900
NVLink Bandwidth (GB/s) (uni-direction)	N/A	300
Deep Learning (Tensor OPS)	112	120
TDP (Watts)	250	300

In the V100-PCIe, all GPUs communicate with each other over PCIe buses. With the V100-SXM2 model, all GPUs are connected by NVIDIA NVLink. In use-cases where multiple GPUs are required, the NVLink interconnect used by V100-SXM2 cards provide the advantage of faster GPU-to-GPU communication when compared to PCIe. V100-SXM2 GPUs provide six NVLinks per GPU for bi-directional communication. The bandwidth of each NVLink is 25GB/s in uni-direction and all four GPUs within a node can communicate at the same time, therefore the theoretical peak bandwidth is 6*25*4=600GB/s in bi-direction. However, the theoretical peak bandwidth using PCIe is only 16*2=32GB/s as the GPUs can only communicate serially, which means the communication cannot be done in parallel. So in theory the data communication with NVLink could be up to 600/32=18x faster than PCIe. Because of this advantage, the PowerEdge C4140 compute node in the Deep Learning solution uses V100-SXM2 instead of V100-PCIe GPUs. In this solution, both 16GB and 32GB V100-SXM2 are supported.

2.3 Processor recommendation for Head Node and Compute Nodes

The processor chosen for the head node and compute nodes is Intel® Xeon® Gold 6148 CPU. This is the latest Intel® Xeon® Scalable processor with 20 physical cores. Additionally, this CPU model is recommended for the compute nodes as well, making this a consistent choice across the cluster.

2.4 Memory recommendation for Head Node and Compute Nodes

The recommended memory for the head node is 24x 16GB 2666MT/s DIMMs. Therefore, the total size of memory is 384GB. This is chosen based on the following facts:

- Capacity: An ideal configuration must support system memory capacity that is larger than the total size of GPU memory. Each compute node has 4 GPUs and each GPU has 16GB or 32GB memory, so the system memory must be at least 16GBx4=64GB. The head node memory also affects I/O performance. For NFS service, larger memory will reduce disk read operations since NFS service needs to send out data from memory. 16GB DIMMs demonstrate the best performance/dollar value.
- DIMM configuration: Choices like 24x 16GB or 12x 32GB will provide the same capacity of 384GB system memory, but according to our studies as shown in Figure 4, the combination of 24x 16GB DIMMs provides 11% better performance than using 12x 32GB. The results shown here was on the Intel Xeon Platinum 8180 processor, but the same trends will apply across other models in the Intel

Scalable Processor Family including the Gold 6148, although the actual percentage differences across configurations may vary. More details can be found in our Skylake memory study.

- Serviceability: The head node and compute nodes memory configurations are designed to be similar to reduce parts complexity while satisfying performance and capacity needs. Fewer parts need to be stocked for replacement, and in extreme urgent cases if a memory module in the head node needs to be replaced immediately, a DIMM module from a compute node can be temporarily considered to restore the head node until replacement modules arrive.

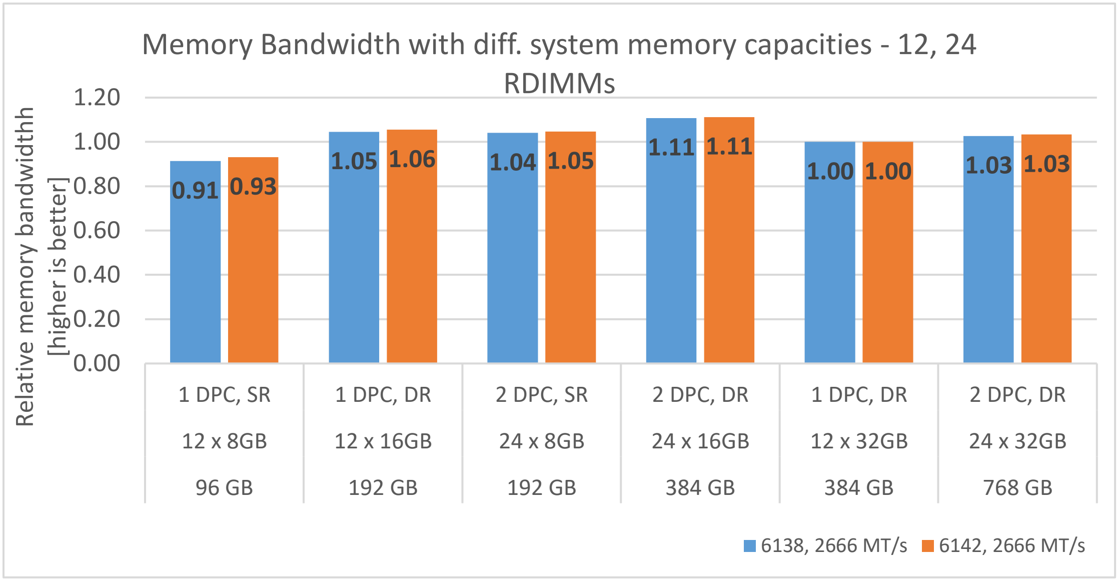


Figure 4: Relative memory bandwidth for different system capacities

2.5 Isilon Storage

Dell EMC Isilon is a proven scale-out network attached storage (NAS) solution that can handle the unstructured data prevalent in many different workflows. The Isilon storage architecture automatically aligns application needs with performance, capacity, and economics. As performance and capacity demands increase, both can be scaled simply and non-disruptively, allowing applications and users to continue working.

Dell EMC Isilon OneFS operating system powers all Dell EMC Isilon scale-out NAS storage solutions and has the following features.

- A high degree of scalability, with grow-as-you-go flexibility
- High efficiency to reduce costs
- Multi-protocol support such as SMB, NFS, HTTP and HDFS to maximize operational flexibility
- Enterprise data protection and resiliency
- Robust security options

The recommended Isilon storage is Isilon F800 all-flash scale-out NAS storage. Dell EMC Isilon F800 all-flash Scale-out NAS storage is uniquely suited for modern Deep Learning applications delivering the flexibility to deal with any data type, scalability for data sets ranging in the PBs, and concurrency to support the massive concurrent I/O request from the GPUs. Isilon's scale-out architecture eliminates the I/O bottleneck between storage and compute, allowing you to start with 10's of TB's of data with up to 15 GB/s bandwidth and then you can scale-out up to 68 PB with up to 540 GB/s of performance in a single cluster. This allows Isilon to accelerate AI innovation with faster model training, provide more accurate insights with deeper data sets, and deliver a higher ROI by fully saturating the data requests of up to 1000's of GPU's per cluster.

The Isilon storage can be used if the local NFS storage capacity is insufficient for the environment. If the Isilon is used in conjunction with the local NFS storage, datasets and project results can be stored on the Isilon with applications installed on the local NFS. The specifications of the Isilon F800 are listed in Table 4.

Table 4: Specification of Isilon F800

External storage	Isilon F800 All-flash Scale-out NAS		
Bandwidth	15 GB/s per chassis, Scales to 540 GB/s per cluster		
IOPS	250,000 IOPS per chassis, Scales to 9 Million IOPS per cluster		
Chassis Capacity (4U)	1.6 TB SSD x 60	3.2 TB SSD x 60	15.4 TB SSD x 60
	96 TB	192 TB	924 TB
Cluster Capacity	All-Flash: 96TB up to 33 PB; Hybrid: 96TB up to 68 PB		
Network	8x 40GbE (QSFP+) per chassis, (2x per node)		

This solution is also capable of mounting existing external storage on either the head node or each compute node.

To monitor and analyze the performance and file system of Isilon storage, the tool InsightIQ can be used. InsightIQ allows a user to monitor and analyze Isilon storage cluster activity using standard reports in the InsightIQ web-based application. The user can customize these reports to provide information about storage cluster hardware, software, and protocol operations. InsightIQ transforms data into visual information that highlights performance outliers, and helps users diagnose bottlenecks and optimize workflows. For more details about InsightIQ, refer to Isilon InsightIQ User Guide.

2.6 Network

The solution comprises of three network fabrics. The head node and all compute nodes are connected with a 1 Gigabit Ethernet fabric. The Ethernet switch recommended is the Dell Networking S3048-ON which has 48 ports. This connection is primarily used by Bright Cluster Manager for deployment, maintenance and monitoring the solution.

The second fabric connects the head node and all compute nodes are through 100 Gb/s EDR InfiniBand. The EDR InfiniBand switch is Mellanox SB7800 which has 36 ports. This fabric is used for Inter-Process Communication (IPC) by the applications as well as to serve NFS from the head node (IPoIB) and Isilon. GPU-to-GPU communication across servers can use a technique called GPUDirect Remote Direct Memory Access (RDMA) which is supported by InfiniBand. This enables GPUs to communicate directly without the involvement of CPUs. Without GPUDirect, when GPUs across servers need to communicate, the GPU in one node has to copy data from its GPU memory to system memory, then that data is sent to the system memory of another node over the network, and finally the data is copied from the system memory of the second node to the receiving GPU's memory. With GPUDirect however, the GPU on one node can send the data directly from its GPU memory to the GPU memory in another node, without going through the system memory in both nodes. Therefore GPUDirect decreases the GPU-GPU communication latency significantly.

The third switch in the solution is called a gateway switch in Figure 2 and connects the Isilon F800 to EDR switch. Isilon's external interfaces are 40 Gigabit Ethernet. Hence, a switch which can serve as the gateway between the 40GbE Ethernet and InfiniBand networks is needed for connectivity to the head and compute nodes. The Mellanox SX6710G is used for this purpose. The gateway is connected to the InfiniBand EDR switch and the Isilon as shown in Figure 2.

2.7 Software

The software portion of the solution is provided by Dell EMC and Bright Computing. The software includes several pieces.

The first piece is Bright Cluster Manager which is used to easily deploy and manage the clustered infrastructure and provides all cluster software including the operating system, GPU drivers and libraries, InfiniBand drivers and libraries, MPI middleware, the Slurm schedule, etc.

The second piece is the Bright machine learning (ML) which includes any deep learning library dependencies to the base operating system, deep learning frameworks including Pytorch, Theano, Tensorflow, Horovod, Keras, DIGITS, CNTK and MXNet, and deep learning libraries including cuDNN, NCCL, and the CUDA toolkit.

The third piece is the Data Science Provisioning Portal which was developed by Dell EMC. The portal was created to abstract the complexity of the deep learning ecosystems by providing a single pane of glass which provides users with an interface to get started with their models. The portal includes spawner for Jupyterhub and integrates with

- Resource managers and schedulers (Slurm)
- LDAP for user management
- Deep Learning framework environments (Tensor Flow, Keras, MXNet, Pytorch etc.) from Bright's module environment, Python2, Python3 and R kernel support
- TensorBoard
- Terminal CLI environments.

It also provides templates to get started with for various DL environments and adds support for singularity containers. For more details about how to use the Data Science Provisioning Portal, refer to Section 4.

2.8 NVIDIA DIGITS Tool and the Deep Learning Solution

DIGITS is a web frontend to Caffe, Torch and TensorFlow, developed by NVIDIA. The user can use the DIGITS graphical user interface for Deep Learning training and inference. The Deep Learning Solution presented in this paper integrates the DIGITS software with the cluster software making it trivial for the system administrator to deploy DIGITS for the users, and easy for the user to use DIGITS.

In order to use DIGITS, the user first needs to allocate the resource, load the DIGITS module and start DIGITS service by running "`digits-devserver`". These steps are listed in Figure 5.

```
[user1@headnode ~]$ srun -N 1 -n 8 -p shareq --gres=gpu:2 --mem=96G -t 8:00:00 --pty bash
[user1@node001 ~]$ module load shared digits/6.1.1
[user1@node001 ~]$ mkdir -p /home/user1/digits-logs
[user1@node001 ~]$ export DIGITS_JOBS_DIR=/home/user1/digits-logs
[user1@node001 ~]$ /cm/shared/apps/digits/6.1.1/digits-devserver
```

Figure 5: How to initialize DIGITS on the Deep Learning Solution

Once the DIGITS server is up and running, a web browser can be used to navigate to the home page of DIGITS. As shown in Figure 6, the DIGITS home page can be accessed from "http://node001:5000". To know how to use DIGITS in more details, refer to Bright Computing's Machine Learning Manual Chapter 2 which includes an example on handwritten digits recognition using a Caffe backend.

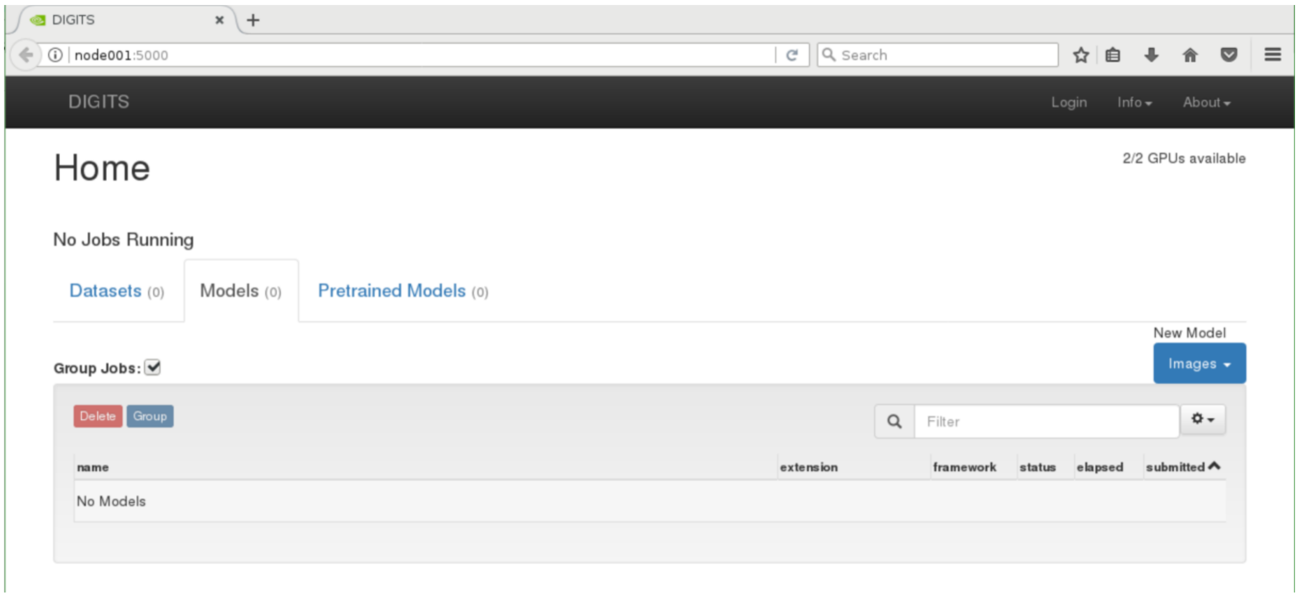


Figure 6: DIGITS home page

3 Containers for Deep Learning

Deep Learning frameworks tend to be complex to install and configure with a myriad of library dependencies. These frameworks and their requisite libraries are under constant development, which makes test environment and test result reproducibility a challenge for researchers. Another layer of complexity is that most enterprise data centers use Red Hat Enterprise Linux (or its derivatives) whereas Ubuntu is the default target for most Deep Learning frameworks.

Containerization technology has surged in popularity as it is a powerful tool for handling the three issues just described – portable test environment, reduced dependency on the underlying operating system and better test result reproducibility. A container packs all the environment and libraries an application needs into an image file and this container can be deployed without any additional changes. It also allows users to easily create, distribute and destroy a container image. Compared to Virtual Machines, containers are lightweight with less overhead. In this section, Singularity, a container designed specifically for use in HPC environments, is used to containerize different Deep Learning applications.

3.1 Singularity Containers

Singularity is developed by Lawrence Berkeley National Laboratory to provide container technology specifically for HPC. It enables applications to be encapsulated in an isolated virtual environment to simplify application deployment. Unlike virtual machines, the container does not have a virtual hardware layer and its own Linux kernel inside the host operating system (OS), therefore the overhead and the performance loss are minimal. The main goal of the container is flexibility and reproducibility. The container has all environment and libraries an application needs to run, and it is portable so that other users can reproduce the results the container creator generated for that application. To use Singularity container, the user only needs to load its module using the command "`module load singularity`" inside a Slurm script which will be described in Section 4.3.

Many HPC applications, especially Deep Learning applications, have extensive library dependencies and it is time consuming to solve these dependencies and debug build issues. Most Deep Learning frameworks are developed in Ubuntu, but they need to be deployed to Red Hat Enterprise Linux (RHEL). It is therefore beneficial to build those applications once in a container and then deploy them anywhere. The most important goal of Singularity is portability which means once a Singularity container is created, the container should be able to run on any system. However, there may be kernel dependencies to consider if a user needs to leverage any kernel specific functionality (e.g. OFED). Usually a user would build a container on a laptop or a server, a cluster or a cloud, and then deploy that container on a server, a cluster or a cloud.

When building a container, one challenge is when using GPU-based systems. If GPU drivers are installed inside the container, and the driver version does not match the host GPU driver, then an error will occur. Hence the container should always use the host GPU driver. The next option is to bind the paths of the GPU driver binary file and libraries to the container so that these paths are visible to the container. However, if the container OS is different than the host OS, such binding may have problems. For instance, assume the container OS is Ubuntu while the host OS is RHEL, and on the host the GPU driver binaries are installed in `/usr/bin` and the driver libraries are installed in `/usr/lib64`. Note that the container OS also has `/usr/bin` and `/usr/lib64`; therefore, if we bind those paths from the host to the container, the other binaries and libraries inside the container may not work anymore because they may not be compatible across different Linux distributions. One workaround is to move all those driver related files to a new central directory location that does not exist in the container and then bind that central location.

The second solution is to implement the above workaround inside the container so that the container can use the driver related files automatically. This feature has already been implemented in the development branch of Singularity repository. A user simply needs to use the option "--nv" when launching the container. However, a cluster environment typically installs GPU driver in a shared file system instead of the default local path on all nodes, and in this case, Singularity is unable to find the GPU driver path since the driver is not installed in the default or common paths (e.g. `/usr/bin`, `/usr/sbin`, `/bin`, etc.). Even if the container is able to find the GPU driver and the corresponding driver libraries and the container is built successfully, the host driver version on the target system must be updated enough to support the GPU libraries which were linked to the application when building the container, else an error will occur due to outdated and incompatible versions between the host system and the container. Given the backward compatibility of GPU drivers, the burden is on the cluster system administrators to keep GPU drivers up to date to ensure the cluster GPU libraries are equal to or newer than the versions of the GPU libraries used when building the container.

Another challenge is when using InfiniBand with the containers because the InfiniBand driver is kernel dependent. There should be no issues if the container OS and host OS are similar or compatible. For instance, RHEL and Centos are compatible, and Debian and Ubuntu are compatible. But if these two OSs are not compatible, then there will be library compatibility issues if the container attempts to use the host InfiniBand driver and libraries. If the InfiniBand driver is installed inside the container, then the drivers in the container and the host might not be compatible since the InfiniBand driver is kernel dependent and the container and the host share the same kernel. If the container and host have different InfiniBand drivers, then a conflict will occur. The Singularity community is working to solve this InfiniBand issue. The current solution is to ensure the container OS and host OS are compatible and allow the container to reuse the InfiniBand driver and libraries on the host. These are only workarounds. The container community is still pushing hard to make containers portable with ease across platforms.

3.2 Running NVIDIA GPU Cloud with the Ready Solutions for AI - Deep Learning

NVIDIA GPU Cloud (NGC) is a cloud that hosts the Docker container images of many Deep Learning frameworks and HPC applications. The Deep Learning frameworks include TensorFlow, MXNet, PyTorch, and so on. These frameworks are optimized for NVIDIA GPUs.

Docker is a very prominent containerization technology used avidly in the context of deep learning frameworks. There are several pros and cons of Docker and a comparison between Docker and Singularity containers is shown “Singularity Containers for HPC & Deep Learning”. NVIDIA added the GPU support in Docker.

Docker is not installed by default in this solution. However, the administrator is expected to simply run `cm-docker-setup` utility provided by Bright Cluster Manager. After installing Docker, the user is able to use the Docker images provided by NVIDIA GPU Cloud which will be discussed next.

To use NGC, a user needs to first register in the cloud and get an API key that is specific to that user. Figure 7 shows the registration page. The user can create an account based on the instructions on the screen.

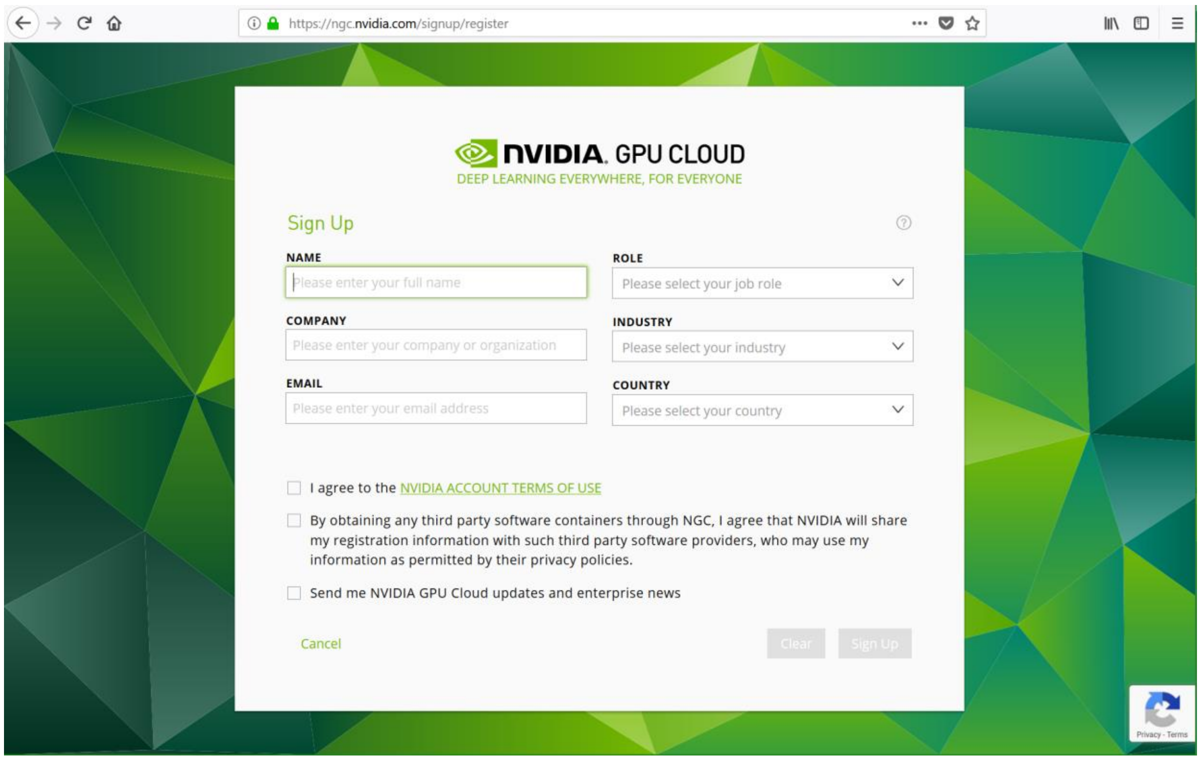
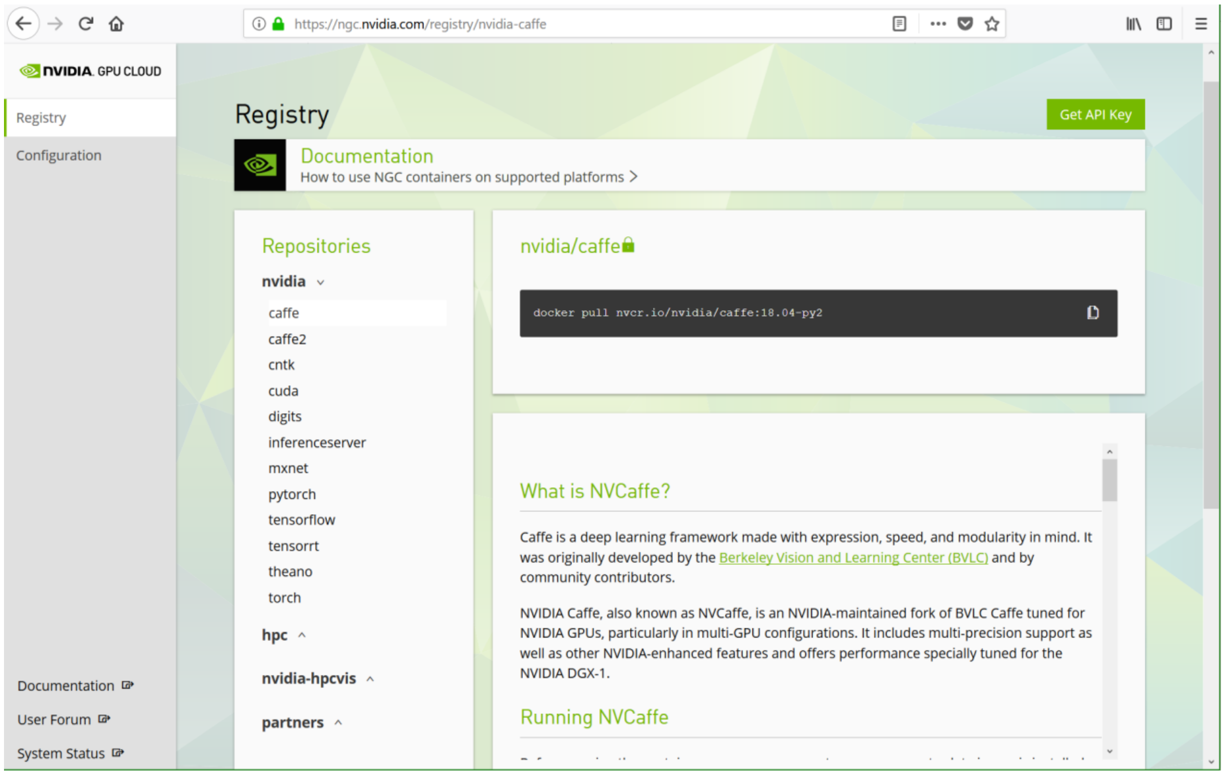

https://ngc.nvidia.com/signup/register
NVIDIA. GPU CLOUD
DEEP LEARNING EVERYWHERE, FOR EVERYONE
Sign Up
NAME
Please enter your full name
ROLE
Please select your job role
COMPANY
Please enter your company or organization
INDUSTRY
Please select your industry
EMAIL
Please enter your email address
COUNTRY
Please select your country
I agree to the NVIDIA ACCOUNT TERMS OF USE
By obtaining any third party software containers through NGC, I agree that NVIDIA will share my registration information with such third party software providers, who may use my information as permitted by their privacy policies.
Send me NVIDIA GPU Cloud updates and enterprise news
Cancel
Clear
Sign Up

Figure 7: NVIDIA GPU Cloud register page

After creating the account, the user will be forwarded to the registry page which lists the available repositories (Figure 8).



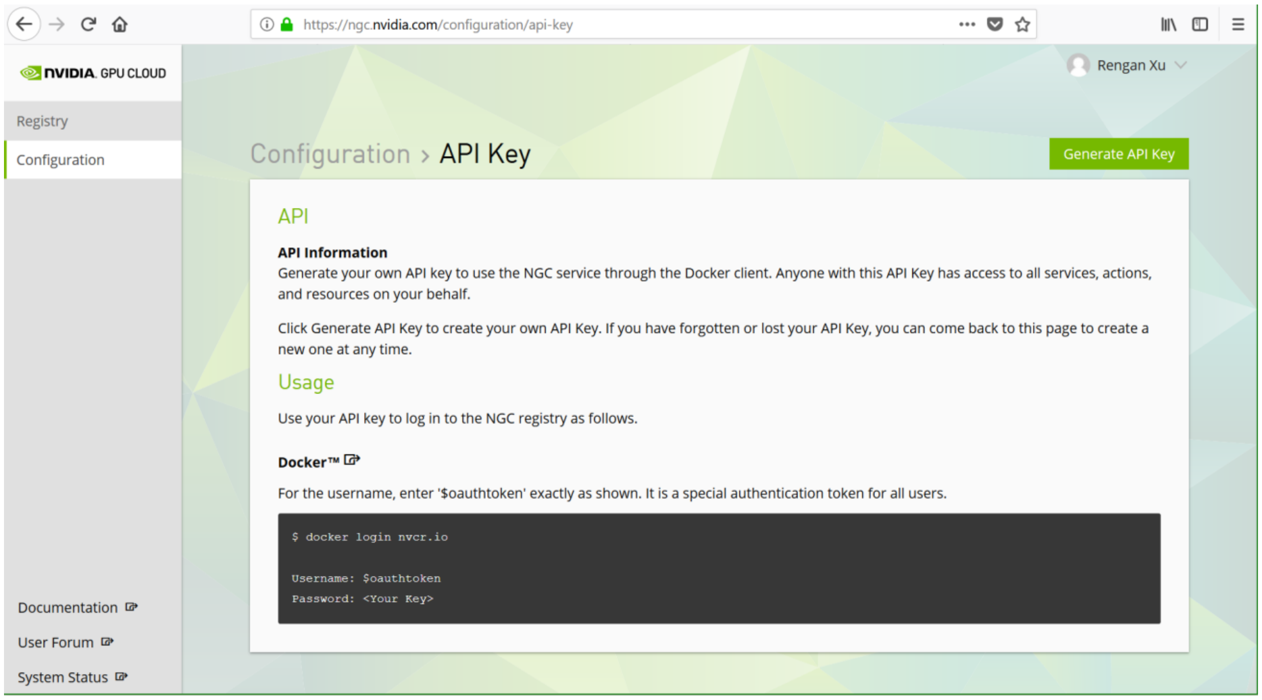
```
docker pull nvcr.io/nvidia/caffe:18.04-py2
```

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It was originally developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors.

NVIDIA Caffe, also known as NVCaffe, is an NVIDIA-maintained fork of BVLC Caffe tuned for NVIDIA GPUs, particularly in multi-GPU configurations. It includes multi-precision support as well as other NVIDIA-enhanced features and offers performance specially tuned for the NVIDIA DGX-1.

Figure 8: NVIDIA GPU Cloud registry page

In Figure 8, clicking the button "`Get API Key`" on the top right will take the user to the configuration page as shown in Figure 9. "Generate API Key" will generate an API key that is specific to the registered user.



API

API Information

Generate your own API key to use the NGC service through the Docker client. Anyone with this API Key has access to all services, actions, and resources on your behalf.

Click Generate API Key to create your own API Key. If you have forgotten or lost your API Key, you can come back to this page to create a new one at any time.

Usage

Use your API key to log in to the NGC registry as follows.

Docker™

For the username, enter '$oauthtoken' exactly as shown. It is a special authentication token for all users.

```
$ docker login nvcr.io

Username: $oauthtoken
Password: <Your Key>
```

Figure 9: NVIDIA GPU Cloud configuration page

A user can download the Docker image of a framework directly with the command in Figure 10.

```
 $ docker login nvcr.io

  Username: $oauthtoken

  Password:

  Login Succeed

 $ docker pull nvcr.io/nvidia/mxnet:17.10

 $ docker images

  REPOSITORY                  TAG                 IMAGE ID            CREATED
SIZE

  nvcr.io/nvidia/mxnet        17.10               9a4558c1fa1a        7 months ago
2.6GB

 $ nvidia-docker run --interactive --detach 9a4558c1fa1a

 $ docker exec 5d2405dbb31c python /opt/mxnet/example/image-
classification/train_mnist.py --gpu 0 --num-epochs 10
```

Figure 10: Steps to download a Docker image from NGC

If the user wants to use Singularity container instead of Docker containers, Figure 11 shows how to create a Singularity image from the Docker image downloaded from NGC. The user must provide a set of environment variables which includes the user's NGC API key, and then use `sregistry` to convert a Docker image to Singularity image. After the Singularity image is generated, the user can use "`exec`" or other commands to execute the script `run_script.sh` within the created Singularity container. For more Singularity commands, please refer to Singularity User Guide.

```
  $ pip install sregistry

  $ export SREGISTRY_NVIDIA_BASE=``ngcr.io”

  $ export SREGISTRY_CLIENT=nvidia

  $ export SREGISTRY_NVIDIA_USERNAME='$oauthtoken’

  $ export SREGISTRY_NVIDIA_TOKEN='[NGC_API_KEY]’

  $ sregistry backend activate nvidia

  $ sregistry pull mxnet:18.11-py3

   Success! /home/user1/.singularity/shub/nvidia-mxnet-18.11-py3-latest.simg

  $ singularity exec /home/user1/.singularity/shub/nvidia-mxnet-18.11-py3-latest.simg
run_script.sh
```

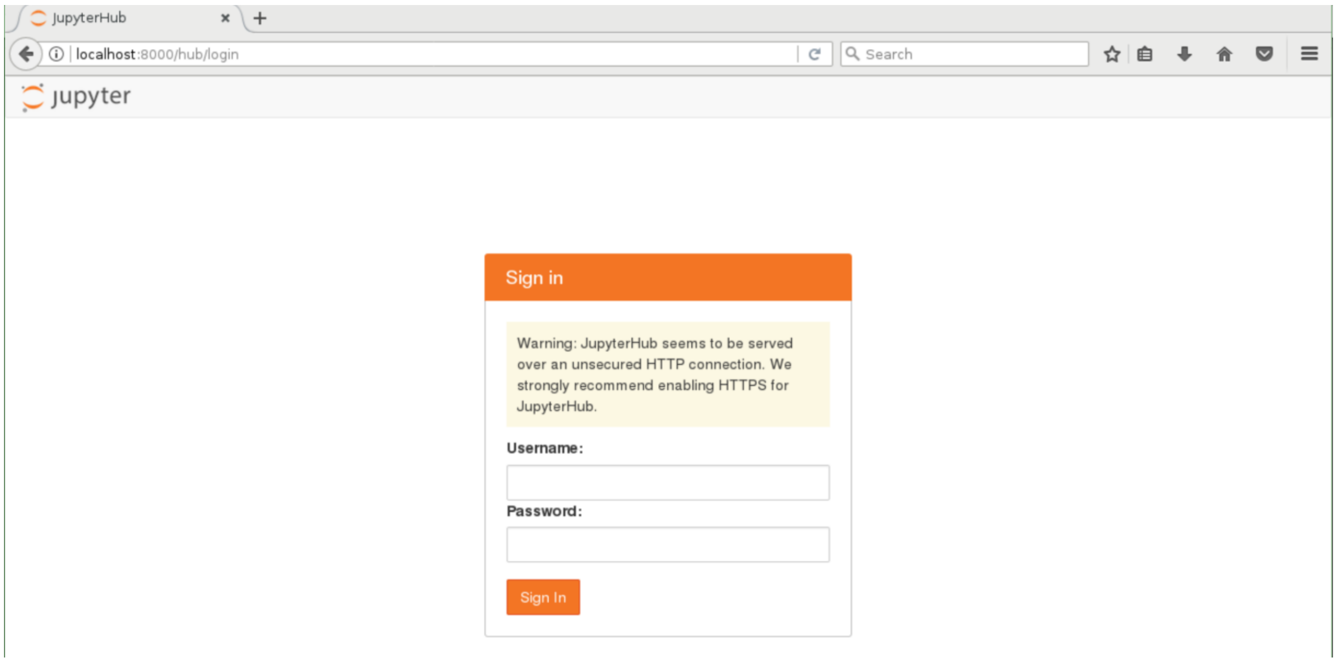
Figure 11: The commands of creating Singularity image from NGC

4 The Data Science Provisioning Portal

The data science provisioning portal was developed by Dell EMC and it simplifies the users' work allowing model implementation, training and inference tests to be run in Jupyter Notebook or directly from the Linux terminal. The Jupyter Notebook allows a user to write explanatory text and intersperse it with raw codes and the tables and figures that those codes generate. It allows a user to directly execute the codes that are embedded in the document that is being created or has been created. The portal also allows multiple users to request resources from the cluster without causing conflicts. In the current version, the resources allocated by the portal is limited to utilizing up to 4 GPUs from a single node. If the user needs to use resources that span across two or more nodes, the request should be submitted directly to the Slurm job scheduler as described in Section 4.3. A deeper dive into the portal and its components is listed in Section 2.7.

4.1 Creating and Running a Notebook

A user can access the portal via a web browser on port 8000 (Figure 12). The user can specify the address either through "`localhost`" if logging to the cluster head node, or through the cluster head node IP address without logging directly into the cluster first. The portal uses the same username and password that is provided for logging into the cluster.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Figure 12: Portal login screen

After logging in, the page is forwarded to the server page (Figure 13). It lists the available server.

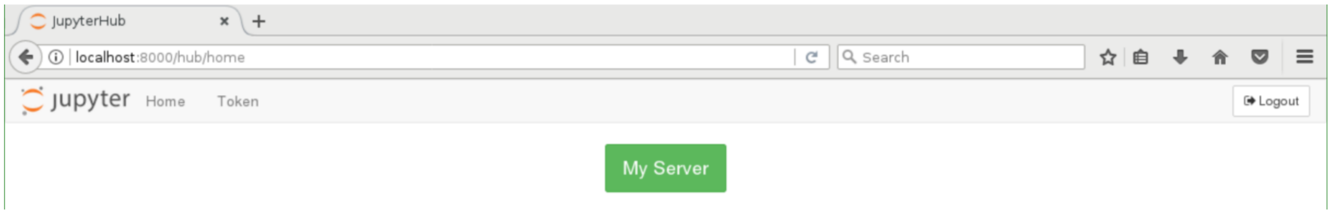
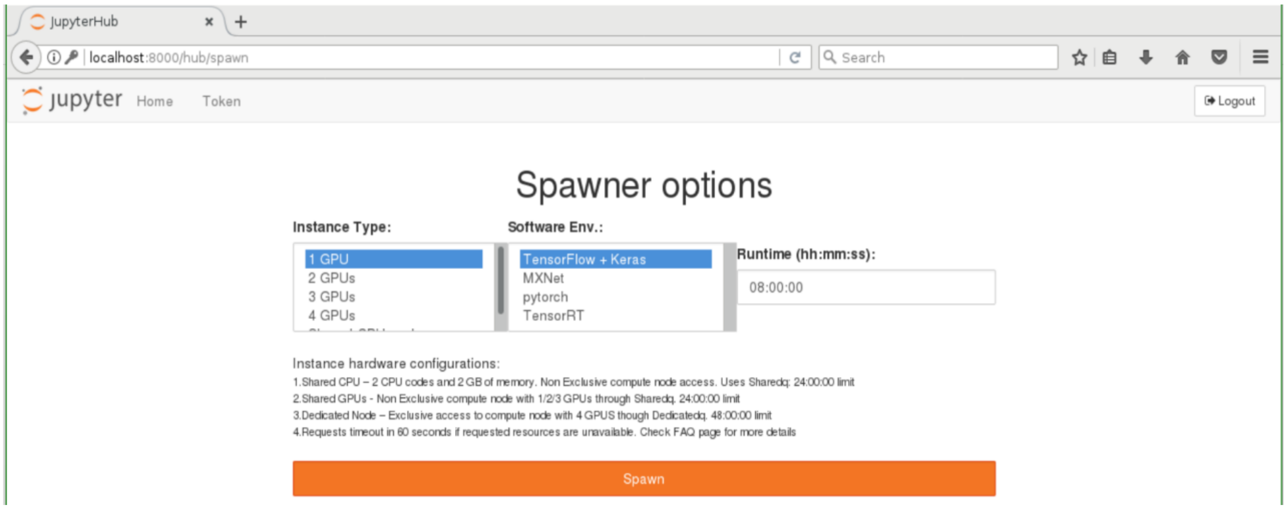


Figure 13: Portal server page

After clicking the “`My Server`” button, the user is forwarded to a list of instance options (Figure 14). The user is given options to choose how many GPUs are being requested. For each allocated GPU, 8 CPU cores and 48GB system memory are allocated. By default, 8 hours of session time is allocated to the user.



Spawner options

Instance Type: 1 GPU, 2 GPUs, 3 GPUs, 4 GPUs

Software Env.: TensorFlow + Keras, MXNet, pytorch, TensorRT

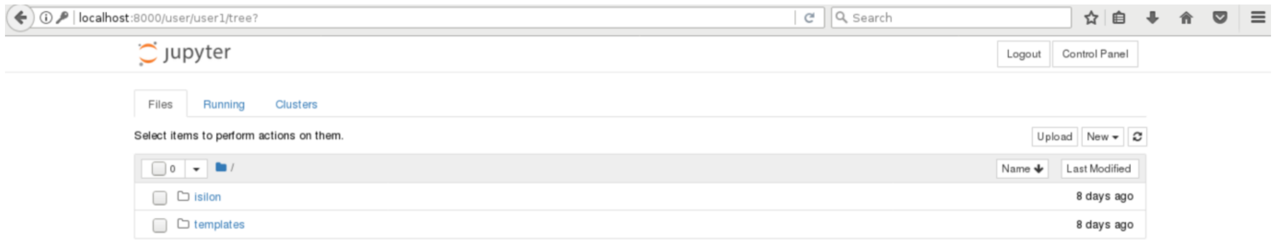
Runtime (hh:mm:ss): 08:00:00

Instance hardware configurations:
1.Shared CPU – 2 CPU codes and 2 GB of memory. Non Exclusive compute node access. Uses Sharedq: 24:00:00 limit
2.Shared GPUs - Non Exclusive compute node with 1/2/3 GPUs through Sharedq. 24:00:00 limit
3.Dedicated Node – Exclusive access to compute node with 4 GPUS though Dedicatedq. 48:00:00 limit
4.Requests timeout in 60 seconds if requested resources are unavailable. Check FAQ page for more details

Spawn

Figure 14: Portal instance screen

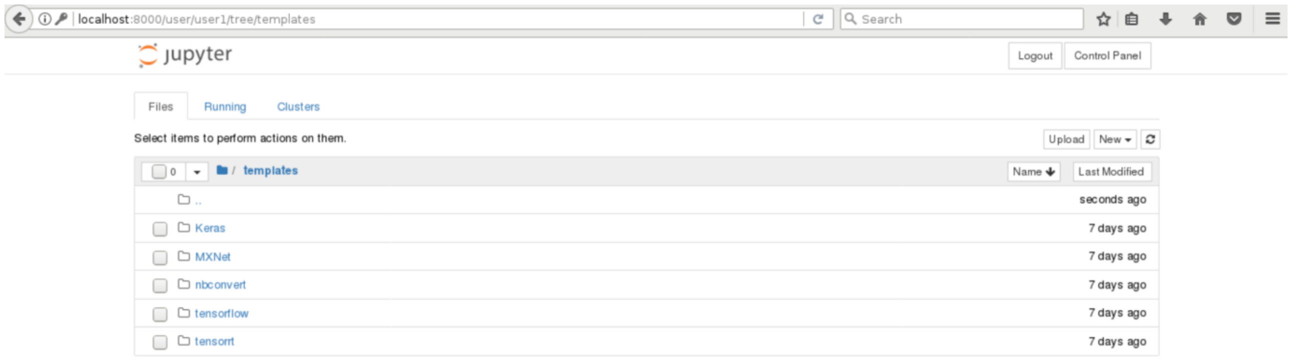
After the resources are allocated, the portal goes to the landing screen (Figure 15). It includes the folder “`templates`” which contains a set of template files of running different Deep Learning frameworks and libraries, and “`isilon`” which is the mount point where Isilon storage is mounted.



Select items to perform actions on them.

Name	Last Modified
isilon	8 days ago
templates	8 days ago

Figure 15: Portal landing screen



Select items to perform actions on them.

/ templates

Name	Last Modified
..	seconds ago
Keras	7 days ago
MXNet	7 days ago
nbconvert	7 days ago
tensorflow	7 days ago
tensorrt	7 days ago

Figure 16: The list of available framework Jupyter Notebook templates

Clicking the “`New`” button in Figure 15 displays a list of kernels (Figure 17). A user can choose Python 2 or Python 3 kernel depending on the user’s preferred Python version. The portal also provides R kernel so that the user can use R package. In “`Other`” kernels, the user can create a text file, a folder, launch a terminal or launch a TensorBoard. Figure 18 is a screenshot after launching a terminal. How to use TensorBoard is shown in Section 4.2.

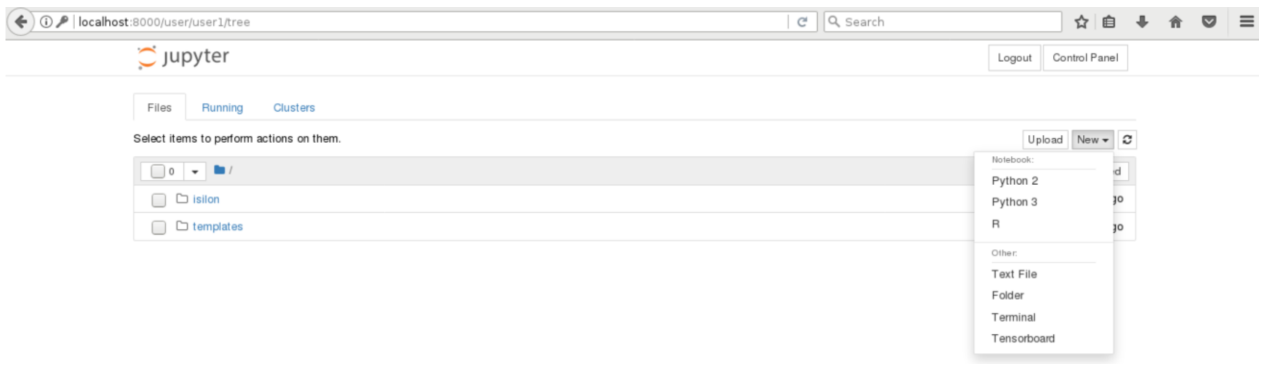
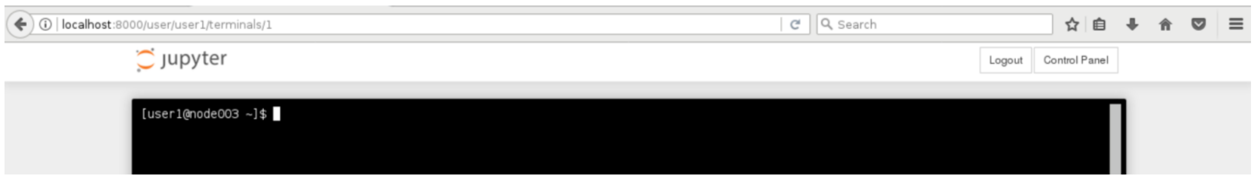


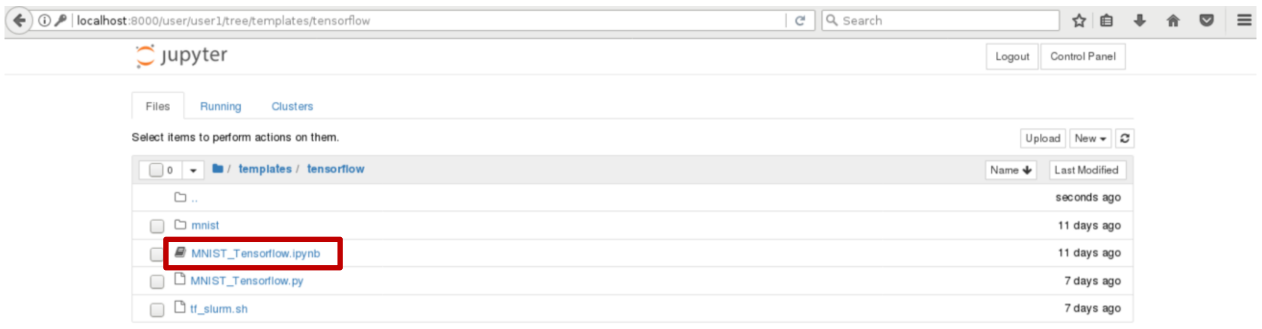
Figure 17: Portal kernel list



```
[user1@node003 ~]$
```

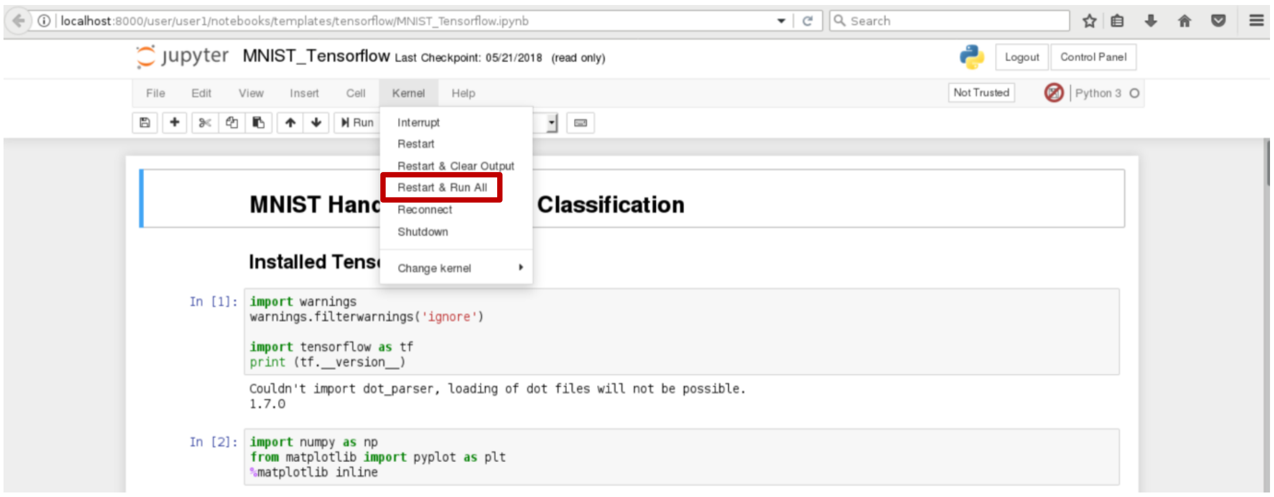
Figure 18: Portal terminal kernel

In the list of framework templates, choosing “`tensorflow`” will bring up the MNIST handwritten digits classification example shown in Figure 19. Now click on “`Kernel`” tab and select “`Restart and Run all`” as shown in Figure 20, the training of the handwritten digits classification will start. An example output is shown in Figure 21.


localhost:8000/user/user1/tree/templates/tensorflow
Search
jupyter
Logout
Control Panel
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
/ templates / tensorflow

Name	Last Modified
..	seconds ago
mnist	11 days ago
MNIST_Tensorflow.ipynb	11 days ago
MNIST_Tensorflow.py	7 days ago
tf_slurm.sh	7 days ago

Figure 19: TensorFlow notebook



```
In [1]: import warnings
        warnings.filterwarnings('ignore')

        import tensorflow as tf
        print (tf.__version__)

        Couldn't import dot_parser, loading of dot files will not be possible.
        1.7.0

In [2]: import numpy as np
        from matplotlib import pyplot as plt
        %matplotlib inline
```

Figure 20: TensorFlow notebook handwritten digits classification

```
Epoch 1 Global Step 1, Training Accuracy 0.1953
Test Accuracy 0.2130
Epoch 1 Global Step 101, Training Accuracy 0.9688
Test Accuracy 0.9571
Epoch 1 Global Step 201, Training Accuracy 0.9297
Test Accuracy 0.9725
Epoch 1 Global Step 301, Training Accuracy 0.9766
Test Accuracy 0.9741
Epoch 1 Global Step 401, Training Accuracy 1.0000
Test Accuracy 0.9782
Epoch 2 Global Step 430, Training Accuracy 1.0000
Test Accuracy 0.9813
Epoch 2 Global Step 530, Training Accuracy 0.9844
Test Accuracy 0.9824
Epoch 2 Global Step 630, Training Accuracy 0.9922
Test Accuracy 0.9832
Epoch 2 Global Step 730, Training Accuracy 0.9922
Test Accuracy 0.9853
Epoch 2 Global Step 830, Training Accuracy 0.9766
Test Accuracy 0.9869
Epoch 3 Global Step 859, Training Accuracy 0.9766
Test Accuracy 0.9860
Epoch 3 Global Step 959, Training Accuracy 1.0000
Test Accuracy 0.9874
Epoch 3 Global Step 1059, Training Accuracy 0.9766
Test Accuracy 0.9874
Epoch 3 Global Step 1159, Training Accuracy 0.9922
Test Accuracy 0.9889
Epoch 3 Global Step 1259, Training Accuracy 1.0000
Test Accuracy 0.9885
```

Figure 21: TensorFlow notebook handwritten digits classification output

After the user starts a kernel, the kernel will keep running until being stopped or the session reaches the end of the allocated time. To stop a kernel, the user needs to click the “`Control Panel`” in Figure 17, then the page will go to the page in Figure 22. After clicking “`Stop My Server`”, the kernel will be stopped.

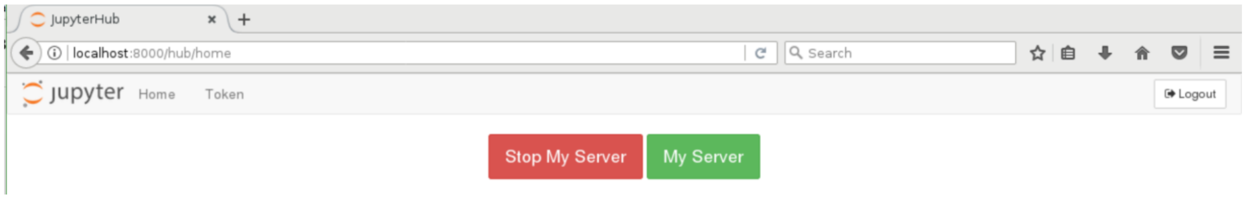


Figure 22: Stop the server

4.2 TensorBoard Integration

Besides the TensorFlow framework, the portal also provides the TensorBoard visualization tool. The TensorBoard is used to visualize the TensorFlow computation graph, plot quantitative metrics about the execution of a graph, and show some other additional data. To use TensorBoard, the user needs to use TensorFlow FileWriter API to serialize the wanted data into a directory. Figure 23 shows an example in which the TensorFlow outputs the serialized data into the folder "`logs`".

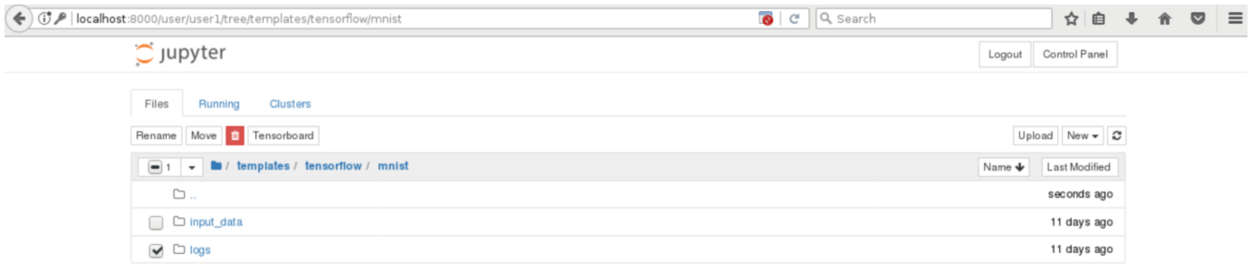


Figure 23: An example of using TensorBoard

After all serialized data are written into the folder "`logs`", the user can choose that folder and then TensorBoard button will be shown up. Then the user clicks "`Tensorboard`", TensorBoard will output all information in a separate tab as shown in Figure 24. The output result in Figure 24 is only an example. A user may get a different result depending on what information is written to the log files.

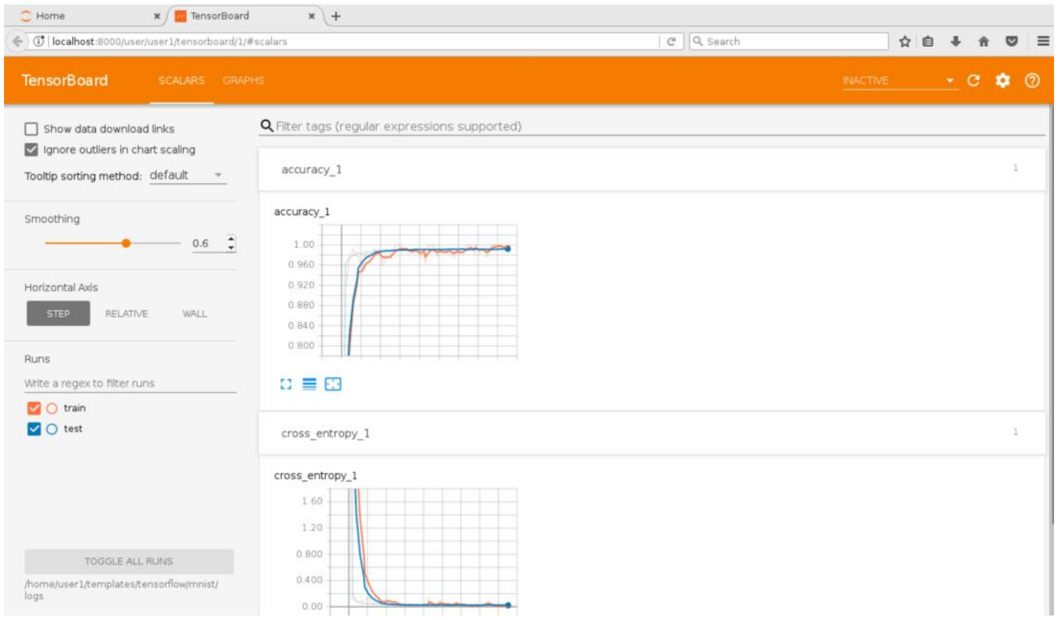

Home
TensorBoard
localhost:8000/user/user1/tensorboard/1/#scalars
TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train
test
TOGGLE ALL RUNS
/home/user1/templates/tensorflow/mnist/logs
Filter tags (regular expressions supported)
accuracy_1
cross_entropy_1

Figure 24: An example TensorBoard output

4.3 Slurm Scheduler

Section 4.1 describes the data science provisioning portal, but the current portal version can allocate resources only within single node. To use resources on multiple nodes, the user can use Slurm job scheduler. The Slurm scheduler is used to manage the resource allocation and job submissions for all users in a cluster. To use Slurm, the user needs to submit a script on the cluster head node that specifies what resources are required and what job should be executed on those resources once allocated.

Figure 25 shows an example Slurm script. In this example, the user asks for 2 nodes (`-N`) and 4 tasks per node (`--ntasks-per-node`), resulting in 8 tasks in total (`-n`). One task implies one CPU process. Nodes that include four GPUs are requested using the `-gres=gpu:4` descriptor. This job will be submitted to the "`sxm2`" queue (`-p`).

The job itself will run two commands: "`hostname`" and "`p2pBandwidthLatencyTest`" from the CUDA SDK. Since running this job requires CUDA toolkit, the user also needs to load the module files for "`shared`" and "`cuda91/toolkit`" that set the path and environment variables needed to execute the `p2pBandwidthLatencyTest` test.

The command "`sinfo`" can be used to check on the types of resources available on the system. Figure 26 shows an example output before running sample.job. After checking enough resources are available, the script can be run with the command "`sbatch sample.job`". The command "`squeue`" can be used to query the status of running jobs. Figure 27 shows an example output after running "`squeue`" to check the status of the current jobs. Note the output script name includes the Slurm job number that was visible in the squeue command output. An example output file name for Figure 25 is slurm-27325.out where 27325 is an example job id. Figure 28 shows an example content of this file. The output file displays the `hostname` and the output of `p2pBandwidthLatencyTest`. Note that although the script in Figure 25 allocated 8 processes, it only used one process in the execution command. To use all processes, the user can use MPI or other multi-process programming model.

NOTE: The queue names, slurm features depend on how Slurm is configured.

```
#!/bin/bash

#SBATCH -N 2

#SBATCH -n 8

#SBATCH --ntasks-per-node 4

#SBATCH -p sxm2

#SBATCH --gres=gpu:4

module load shared

module load cuda92/toolkit

hostname

/cm/shared/apps/cuda92/sdk/9.2.88/bin/x86_64/linux/release/p2pBandwidthLatencyTest
```

Figure 25: An example Slurm job script named sample.job

```
PARTITION  AVAIL  TIMELIMIT NODES  STATE  NODELIST
hipri*      up   12:00:00   17     down*  node[001-007,009-010,012,015-016,018-019]
sxm2        up   12:00:00   2      idle   node[008,014]
iq          up   6:00:00    2      down*  node[011,013]
requestq    up   infinite   2      idle   node[008,014]
```

Figure 26: An example output of the command sinfo before running sample.job

```
JOBID     PARTITION     NAME     USER    ST      TIME  NODES NODELIST(REASON)
 27325     sxm2        sample.j  root     R      0:01   2    node[008,014]
```

Figure 27: An example output of the command squeue showing the job status

```
node008
[P2P (Peer-to-Peer) GPU Bandwidth Latency Test]
Device: 0, Tesla V100-SXM2-32GB, pciBusID: 18, pciDeviceID: 0, pciDomainID:0
Device: 1, Tesla V100-SXM2-32GB, pciBusID: 3b, pciDeviceID: 0, pciDomainID:0
Device: 2, Tesla V100-SXM2-32GB, pciBusID: 86, pciDeviceID: 0, pciDomainID:0
Device: 3, Tesla V100-SXM2-32GB, pciBusID: af, pciDeviceID: 0, pciDomainID:0
Device=0 CAN Access Peer Device=1
Device=0 CAN Access Peer Device=2
Device=0 CAN Access Peer Device=3
Device=1 CAN Access Peer Device=0
Device=1 CAN Access Peer Device=2
Device=1 CAN Access Peer Device=3
Device=2 CAN Access Peer Device=0
Device=2 CAN Access Peer Device=1
Device=2 CAN Access Peer Device=3
Device=3 CAN Access Peer Device=0
Device=3 CAN Access Peer Device=1
Device=3 CAN Access Peer Device=2

***NOTE: In case a device doesn't have P2P access to other one, it falls back to normal
memcopy procedure.
So you can see lesser Bandwidth (GB/s) and unstable Latency (us) in those cases.

P2P Connectivity Matrix
     D\D     0     1     2     3
     0       1     1     1     1
     1       1     1     1     1
     2       1     1     1     1
     3       1     1     1     1
Unidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 730.14  10.93  11.00  10.97
     1  10.93 739.82  10.99  10.96
     2  11.03  10.93 741.22  10.97
     3  11.03  10.93  10.99 741.22
Unidirectional P2P=Enabled Bandwidth (P2P Writes) Matrix (GB/s)
   D\D     0      1      2      3
     0 732.88  48.33  48.35  48.32
     1  48.33 742.63  48.33  48.35
     2  48.33  48.35 741.22  48.36
     3  48.34  48.36  48.32 741.22
Bidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 746.89  19.30  19.92  19.31
     1  19.39 751.92  20.08  19.37
     2  19.53  19.22 749.04  19.29
     3  19.45  19.57  19.37 757.76
Bidirectional P2P=Enabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 754.79  96.26  96.47  96.48
     1  96.28 755.56  96.51  96.50
     2  96.47  96.28 755.56  96.28
     3  96.30  96.30  96.53 757.03
P2P=Disabled Latency Matrix (us)
   GPU     0      1      2      3
     0   1.88  14.54  15.57  15.67
     1  14.71   1.96  15.77  15.47
     2  15.49  15.41   1.84  14.43
     3  15.52  15.51  14.42   1.84

   CPU     0      1      2      3
     0   3.82  10.69  10.13   9.11
     1  10.25   4.19  10.69   9.74
     2   9.95  10.85   4.04   9.33
     3   9.31  10.19   9.76   3.46
P2P=Enabled Latency (P2P Writes) Matrix (us)
   GPU     0      1      2      3
     0   1.87   1.72   1.72   1.71
     1   1.70   2.09   1.70   1.70
     2   1.65   1.64   1.88   1.64
     3   1.64   1.64   1.64   1.84

   CPU     0      1      2      3
     0   3.63   2.67   2.64   2.66
     1   2.94   4.19   2.96   3.01
     2   2.74   2.74   4.03   2.73
     3   2.46   2.44   2.51   3.45

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary
when GPU Boost is enabled.
```

Figure 28: An example output for the after running sample.job. The file name is slurm-27325.out.

5 Conclusions

This document describes the integrated Dell EMC Ready Solutions for AI - Deep Learning with NVIDIA. The goal of this solution is to provide a complete, tuned and supported solution for Deep Learning training and inference use cases. The solution takes into account the ideal compute, storage, network, and software configuration for this workload. It includes tools that improve the usability of the system for data scientists, like the Data Science Provisioning Portal that makes it easy for users to develop, train and run inferencing for Deep Learning models. The solution pre-installs a set of frameworks and libraries that are necessary for Deep Learning in addition to all other the software (operating system, drivers, libraries, management utilities, resource manager) required on a cluster.