DGX_wiki Documentation

Release 1

shzhang

May 19, 2020

Contents

CHAPTER 1

About the wiki

The aim of this wiki is to provide members in Prof. Lei Xie's group or other users/researchers the basic knowledge to use the NVIDIA® DGX Station™ for research. The wiki contains materials from other tutorials/organizations for a better explanation.

Following sessions are listed on the navigation bar at the left of the page:

DGX station introduces the basic information and using of NVIDIA® DGX Station™.

Linux tutorial includes some basic infomation about how to use Linux system.

Docker tutorial consists of the knowledge of Docker and the specific tutorial of using Docker with NVIDIA GPU Cloud service.

HTCondor tutorial focuses on the general approach of running jobs via HTCondor™ workload management system.

CHAPTER 2

Administrators

If you have any problems of using workstation or reading wiki. Please contact the administrators of DGX-Lei workstation:

Shuo Zhang(email: szhang4@gradcenter.cuny.edu)

Hansaim Lim(email: hlim1@gradcenter.cuny.edu)

Di He(email: dhe@gradcenter.cuny.edu)

CHAPTER 3

Table of Contents

3.1 DGX station

3.1.1 About DGX station

What is DGX Station

NVIDIA® DGX Station™ is the world's first personal supercomputer for leading-edge AI development.

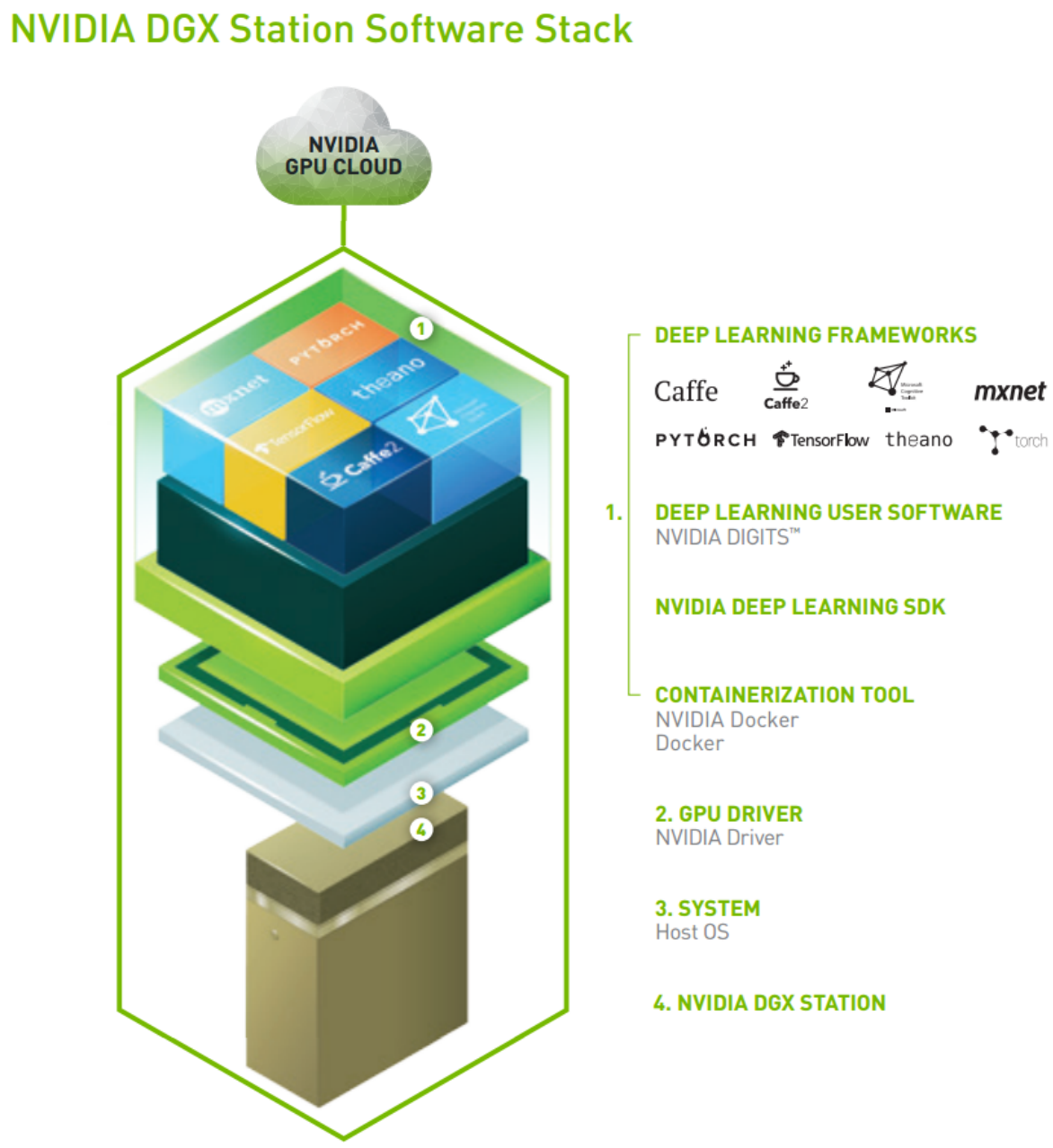
Features of DGX Station

Deep learning platforms require software engineering expertise to keep today's frameworks optimized for maximum performance, with time spent waiting on stable versions of open source software. This means hundreds of thousands of dollars in lost productivity, dwarfing the initial hardware cost.

NVIDIA DGX Station includes the same software stack found in all DGX solutions. This innovative, integrated system includes access to popular deep learning frameworks, updated monthly, each optimized by NVIDIA engineers for maximized performance. It also includes access to NVIDIA DIGITS™ deep learning training application, third-

party accelerated solutions, the the NVIDIA Deep Learning SDK (e.g. cuDNN, cuBLAS, NCCL), CUDA® Toolkit, and NVIDIA drivers.

Built on container technology powered by NVIDIA Docker, this unified deep learning software stack simplifies workflow, saving you days in re-compilation time when you need to scale your work and deploy your models in the data center or cloud.



Hardware Summary

Processors

Component	Qty	Description
CPU	1	Intel Xeon E5-2698 v4 2.2 GHz (20-Core)
GPU	4	NVIDIA Tesla® V100 with 16 GB per GPU (64 GB total) of GPU memory

System Memory and Storage

Component	Qty	Unit Capacity	Total Capacity	Description
System memory	8	32 GB	256 GB	ECC Registered LRDIMM DDR4 SDRAM
Data storage	3	1.92 TB	5.76 TB	2.5” 6 Gb/s SATA III SSD in RAID 0 configuration
OS storage	1	1.92 TB	1.92 TB	2.5” 6 Gb/s SATA III SSD

Read more at: http://docs.nvidia.com/dgx/dgx-station-user-guide/index.html

3.1.2 Connect to DGX station

SSH Login

Currently, DGX station(IP: 146.95.214.135) can only be accessed via CS network inside Hunter College. You can use ssh via lab computer to access it:

```
ssh yourname@146.95.214.135
```

If you want to access DGX station outside, the following steps are needed:

1. Use ssh to connect to eniac from anywhere using your eniac account:

```
ssh yourname@eniac.cs.hunter.cuny.edu
```

(Note: Eniac account should be claimed by logging in from one of the Computer Science Linux Labs (1001B or 1001C), or it will be deleted)

Then use ssh to connect to lab computers from eniac.

2. Use ssh to connect to DGX station from lab computers:

```
ssh yourname@146.95.214.135
```

3.1.3 Using the DGX station

User Directories

Each user will have two directories according to the user's account:

```
/home/user_name
```

and:

```
/raid/home/user_name
```

Note that files under "/home/user_name" consume the 1.92 TB OS storage while files under "/raid/home/user_name" consume the 5.76 TB Data storage(see here for detailed Hardware Summary of DGX station).

To avoid the unnecessary consumption of OS storage, all users should only use:

```
/raid/home/user_name
```

as their working directory.

Data Backup

Only storing data under "/raid" without a backup is unsafe since we use the RAID 0 configuration.

All users are required to back up their data by themselves using their preferred ways under the directory:

```
/data/dgx/backup/user_name
```

The data under "/data/dgx/backup" is stored in an external 8TB disk. It is also recommended that all users still make another copy of their important data outside the DGX station.

3.2 Linux tutorial

3.2.1 Linux Commands

Linux commands should be prerequisite knowledge of using DGX station, this part just lists some links you can use to find commands you need.

Linux commands: https://ss64.com/bash/

Bash Shell Reference: https://courses.cs.washington.edu/courses/cse390a/14au/bash.html

Learning the Shell: http://linuxcommand.org/lc3_learning_the_shell.php

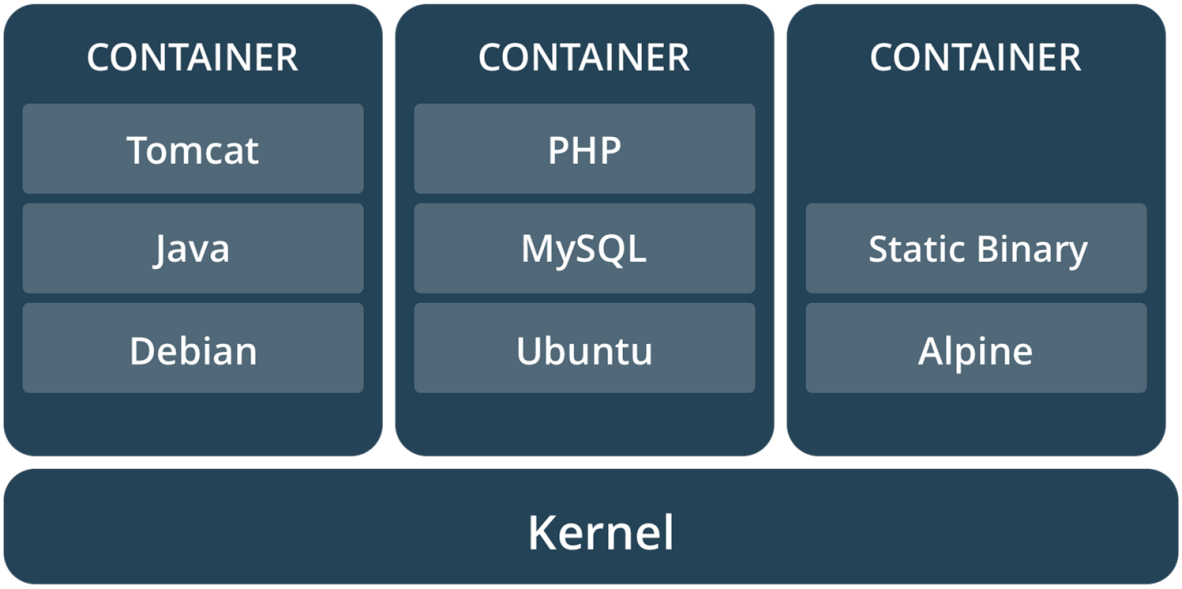
3.3 Docker tutorial

3.3.1 Introduction to Docker

Docker is an open-source project for automating the deployment of applications as portable, self-sufficient *containers* that can run on the cloud or on-premises. Docker is also a company that promotes and evolves this technology. Docker works in collaboration with cloud, Linux, and Windows vendors.

Official website of Docker: https://www.docker.com/

About Containers

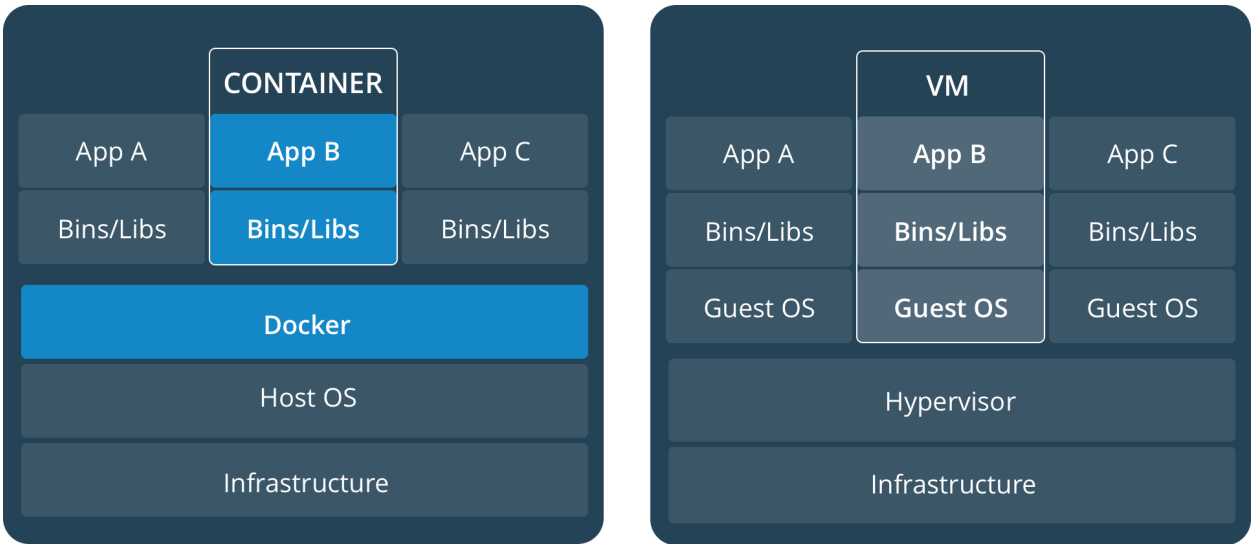


A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment.

Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.



CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

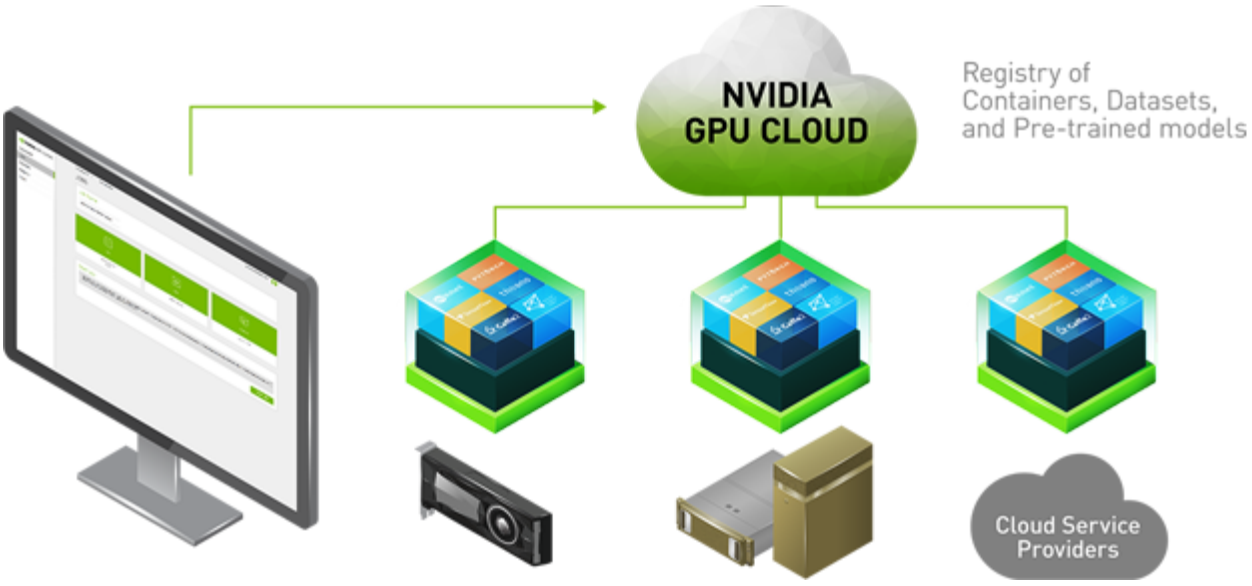
VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

3.3.2 Docker on NVIDIA GPU Cloud

After knowing about the basic knowledge of Docker platform and containers, we will use these in our computing. Fortunately, NVIDIA offers NVIDIA GPU Cloud (NGC), which empowers AI researchers with performance-engineered deep learning framework containers, allowing them to spend less time on IT, and more time experimenting, gaining insights, and driving results.

About NVIDIA GPU Cloud(NGC)



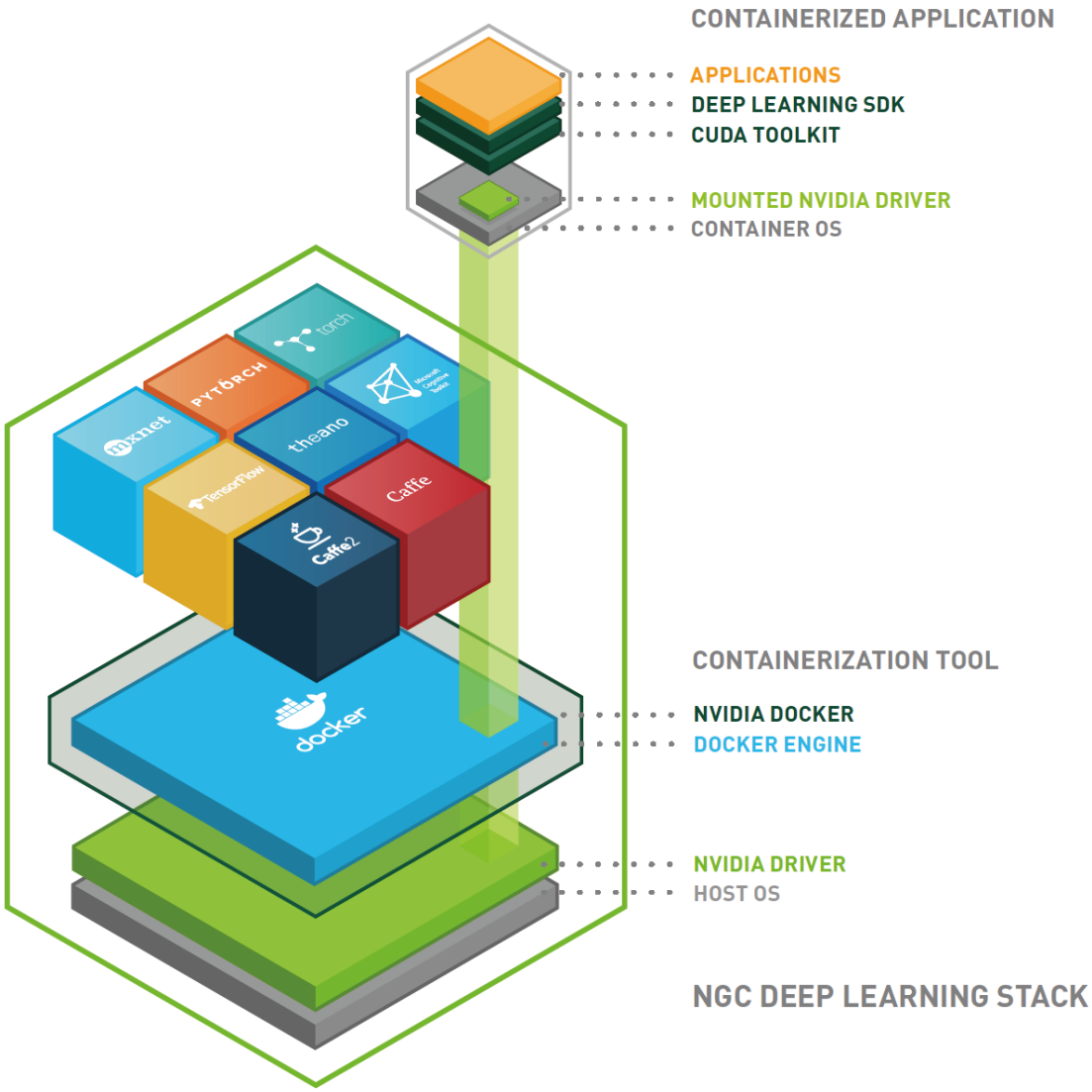
NVIDIA GPU Cloud is a GPU-accelerated cloud platform optimized for deep learning. NGC manages a catalog of fully integrated and optimized deep learning framework containers that take full advantage of NVIDIA GPUs. These framework containers are delivered ready-to-run, including all necessary dependencies such as CUDA runtime, NVIDIA libraries, and an operating system. They are tuned, tested, and certified by NVIDIA to run on NVIDIA DGX Systems. NVIDIA updates these containers monthly to ensure they continue to provide peak performance.

NGC Container Registry

The NGC container registry is a catalog of GPU-accelerated deep learning software. It includes CUDA Toolkit, DIGITS workflow, and the following deep learning frameworks: NVCaffe, Caffe2, Microsoft Cognitive Toolkit (CNTK), MXNet, PyTorch, TensorFlow, Theano, and Torch.

The NGC container registry provides containerized versions of these frameworks. These frameworks, including all necessary dependencies, form the NGC Deep Learning Stack. For users who need more flexibility to build custom deep learning solutions, each framework container image also includes the framework source code to enable custom modifications and enhancements, along with the complete software development stack.

The design of the platform software is centered around a minimal OS and driver install on the server and provisioning of all application and SDK software in NVIDIA Docker containers through NVIDIA Docker Registry.



NVIDIA Docker

To enable portability in Docker images that leverage GPUs, NVIDIA developed NVIDIA Docker, an open source project that provides a command line tool to mount the user mode components of the NVIDIA driver and the GPUs into the Docker container at launch. nv-docker is essentially a wrapper around Docker that transparently provisions a container with the necessary components to execute code on the GPU.

Use NGC Service on DGX Station

To use NGC Service and framework container images on DGX station, administrators have the ability to pull new or the latest images from NGC, and other people can directly run it for there research purpose.

Pull Docker Images from NGC(For Administrators)

Before trying to pull any image, you can first check if the container images are already on DGX station by:

```
docker images
```

It will return the Docker images on your system(below is an example):

```
REPOSITORY                    TAG                                 IMAGE ID
↪CREATED              SIZE
nvcr.io/nvidia/theano         18.01                               dc2fda091dd4
↪weeks ago          4GB
nvcr.io/nvidia/mxnet          18.01-py3                           1a70c143e041
↪weeks ago          2.67GB
nvcr.io/nvidia/mxnet          18.01-py2                           ffe7de2f34c3
↪weeks ago          2.64GB
nvcr.io/nvidia/pytorch        17.12                               5ac6ff8f9a81
↪months ago         4.52GB
nvcr.io/nvidia/tensorflow     17.12                               19afd620fc8e
↪months ago         2.88GB
nvcr.io/nvidia/caffe2         17.12                               ac17621b6d70
↪months ago         2.8GB
nvcr.io/nvidia/caffe          17.12                               cd94fcaaec3f
↪months ago         3.25GB
nvcr.io/nvidia/cuda           9.0-cudnn7-devel-ubuntu16.04        634be617d3ed
↪months ago         1.75GB
```

If there are no images you need or the images are out of date, you can continue the following steps:

1. Activate your NVIDIA DGX Cloud Services account.
2. Log on to the NVIDIA GPU CLOUD website. Obtain your API Key so you can access the NVIDIA DGX Container Registry.
3. Log in to the NVIDIA DGX Container Registry from a command line:

   ```
   sudo docker login nvcr.io
   ```

   When prompted for your username and password, enter the following text and the API key you already got:

   ```
   Username: $oauthtoken
   Password: **************************
   ```

4. Pull Docker images.

   Download the container that you want from the registry:

   ```
   docker pull registry/registry-space/repository:tag
   ```

   You can click on the image in Registry you need and scroll down the web page to find the specific pull command of it. For example:

   ```
   docker pull nvcr.io/nvidia/tensorflow:18.01-py2
   ```

Use Docker Images(For Users)

Run a container:

```
nvidia-docker run --name containerName -it --rm -v local_dir:container_dir nvcr.io/
↪nvidia/repository:<xx.xx>
```

For example: If you want to run a Docker image of tensorflow with 18.01-py2 version. Your working directory contains files is /raid/home/user_name/workspace. user_name is your account name on DGX station. And you want to see those files in the directory of /workspace in the container image. Then the command to run that container image with a name of 'test' will be like this:

```
nvidia-docker run --name test -it --rm -v /raid/home/user_name/workspace:/workspace␣
↪nvcr.io/nvidia/tensorflow:18.01-py2
```

Then you will be able to synchronize files between these two specified directories. A more detailed explanation of this command can be found here.

More docker commands for NVIDIA DGX Cloud Service using can be found at 1 and 2.

To be mentioned that, this part only talks about the using of Docker specifically related with DGX Cloud Service. The following part will introduce the commonly used docker commands.

3.3.3 Docker Commands

Besides the Docker commands mentioned in previous part for administrators and users. This part contains a collection of basic and useful Docker commands in daily using. A full list of Docker commands can be found at here.

Some common use commands:

docker images	List images on the system
docker ps	List containers that are running now
docker pull	Pull an image or a repository from a registry
docker push	Push an image or a repository to a registry
docker run	Run a command in a new container
docker rename	Rename a container
docker kill	Kill one or more running containers

There are two usual ways to exit a container. In most cases, people use "ctrl+P+Q" to exit a container and jobs running in it will not be killed:

```
exit or ctrl+D          Exit a container and close it, then container cannot be found␣
↪via docker ps
ctrl+P+Q                Exit a container without closing it, container can be found via␣
↪docker ps
```

To resume a container after exiting with "ctrl+P+Q":

```
docker exec -it containerNAME /bin/bash
```

or

```
docker attach containerNAME
```

Note: Using the second one will sometimes get stuck and have no response for a long time. The first one is preferred.

3.4 HTCondor tutorial

HTCondor™ is chosen as the workload management system on DGX-Lei workstation. One reason is that it can directly submit Docker job as we mainly use Docker images to run Machine Learning tasks. This part talks about the background of HTCondor and the using of it.

3.4.1 Introduction of HTCondor™

What is HTCondor

HTCondor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

HTCondor is the product of years of research by the Center for High Throughput Computing in the Department of Computer Sciences at the University of Wisconsin-Madison (UW-Madison), and it was first installed as a production system in the UW-Madison Department of Computer Sciences over 15 years ago. This HTCondor installation has since served as a major source of computing cycles to UW-Madison faculty and students. Additional HTCondor installations have been established over the years across our campus and the world. Hundreds of organizations in industry, government, and academia have used HTCondor to establish compute installations ranging in size from a handful to many thousands of workstations.

The HTCondor software, source code, and complete documentation are freely available under an open source license. Linux, MacOS, and Windows platforms are supported.

Features of HTCondor

While providing functionality similar to that of a more traditional batch queueing system, HTCondor's novel architecture allows it to succeed in areas where traditional scheduling systems fail. HTCondor can be used to manage a cluster of dedicated compute nodes (such as a "Beowulf" cluster). In addition, unique mechanisms enable HTCondor to effectively harness wasted CPU power from otherwise idle desktop workstations. For instance, HTCondor can be configured to only use desktop machines where the keyboard and mouse are idle. Should HTCondor detect that a machine is no longer available (such as a key press detected), in many circumstances HTCondor is able to transparently produce a checkpoint and migrate a job to a different machine which would otherwise be idle. HTCondor does not require a shared file system across machines - if no shared file system is available, HTCondor can transfer the job's data files on behalf of the user, or HTCondor may be able to transparently redirect all the job's I/O requests back to the submit machine. As a result, HTCondor can be used to seamlessly combine all of an organization's computational power into one resource.

The ClassAd mechanism in HTCondor provides an extremely flexible and expressive framework for matching resource requests (jobs) with resource offers (machines). Jobs can easily state both job requirements and job preferences. Likewise, machines can specify requirements and preferences about the jobs they are willing to run. These requirements and preferences can be described in powerful expressions, resulting in HTCondor's adaptation to nearly any desired policy.

3.4.2 Manage Job with HTCondor™

Quick Start Guide

To users, HTCondor is a job scheduler. You give HTCondor a file containing commands that tell it how to run jobs. HTCondor locates a machine that can run each job within the pool of machines, packages up the job and ships it off to this execute machine. The jobs run, and output is returned to the machine that submitted the jobs.

At the beginning, we are going to run the traditional 'hello world' program. In order to demonstrate the distributed resource nature, we will produce a 'Hello DGX' message 3 times, where each time is its own job. Since you are not directly invoking the execution of each job, you need to tell HTCondor how to run the jobs for you. The information needed is placed into a submit file, which defines variables that describe the set of jobs.

1. Copy the text below, and paste it into file called hello-DGX.sub, the submit file, in your home directory on the submit machine:

```
# hello-DGX.sub
# My very first HTCondor submit file
#
# Specify the HTCondor Universe (vanilla is the default and is used
#  for almost all jobs), the desired name of the HTCondor log file,
#  and the desired name of the standard error file.
#  Wherever you see $(Cluster), HTCondor will insert the queue number
#  assigned to this set of jobs at the time of submission.
universe = vanilla
log = hello-DGX_$(Cluster).log
error = hello-DGX_$(Cluster)_$(Process).err
#
# Specify your executable (single binary or a script that runs several
#  commands), arguments, and a files for HTCondor to store standard
#  output (or "screen output").
#  $(Process) will be a integer number for each job, starting with "0"
#  and increasing for the relevant number of jobs.
executable = hello-DGX.sh
arguments = $(Process)
output = hello-DGX_$(Cluster)_$(Process).out
#
# Specify that HTCondor should transfer files to and from the
#  computer where each job runs. The last of these lines *would* be
#  used if there were any other files needed for the executable to run.
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
# transfer_input_files = file1,/absolute/pathto/file2,etc
#
# Tell HTCondor what amount of compute resources
#  each job will need on the computer where it runs.
request_cpus = 1
request_gpus = 1
request_memory = 1GB
request_disk = 1MB
#
# Tell HTCondor to run 3 instances of our job:
queue 3
```

2. Now, create the executable that we specified above: copy the text below and paste it into a file called hello-DGX.sh:

```
#!/bin/bash
#
# hello-chtc.sh
```

(continues on next page)

(continued from previous page)

```
# My very first DGX job
#
echo "Hello DGX from Job $1 running on `whoami`@`hostname`"
```

When HTCondor runs this executable, it will pass the $(Process) value for each job and hello-DGX.sh will insert that value for "$1", above.

3. Now, submit your job to the queue using condor_submit:

```
condor_submit hello-DGX.sub
```

The condor_submit command actually submits your jobs to HTCondor. If all goes well, you will see output from the condor_submit command that appears as:

```
Submitting job(s)...
3 job(s) submitted to cluster 9.
```

4. To check on the status of your jobs, run the following command:

```
condor_q
```

The output of condor_q should look like this:

```
-- Schedd: DGX-Lei : <146.95.214.135:9618?... @ 01/29/18 02:15:29
OWNER     BATCH_NAME          SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
shuozhang CMD: hello-DGX.s   1/29 02:14      _      _      3      3 9.0-2

3 jobs; 0 completed, 0 removed, 3 idle, 0 running, 0 held, 0 suspended
```

You can run the condor_q command periodically to see the progress of your jobs. By default, condor_q shows jobs grouped into batches by batch name (if provided), or executable name. To show all of your jobs on individual lines, add the -nobatch option.

5. When your jobs complete after a few minutes, they'll leave the queue. If you do a listing of your home directory with the command ls -l, you should see something like:

```
-rw-r--r-- 1 shuozhang shuozhang    0 Jan 29 02:14 hello-DGX_9_0.err
-rw-r--r-- 1 shuozhang shuozhang   47 Jan 29 02:47 hello-DGX_9_0.out
-rw-r--r-- 1 shuozhang shuozhang    0 Jan 29 02:14 hello-DGX_9_1.err
-rw-r--r-- 1 shuozhang shuozhang   47 Jan 29 02:47 hello-DGX_9_1.out
-rw-r--r-- 1 shuozhang shuozhang    0 Jan 29 02:14 hello-DGX_9_2.err
-rw-r--r-- 1 shuozhang shuozhang   47 Jan 29 02:47 hello-DGX_9_2.out
-rw-rw-r-- 1 shuozhang shuozhang 3425 Jan 29 02:47 hello-DGX_9.log
-rw-rw-r-- 1 shuozhang shuozhang  115 Jan 29 02:13 hello-DGX.sh
-rw-rw-r-- 1 shuozhang shuozhang 1416 Jan 29 02:14 hello-DGX.sub
```

Useful information is provided in the user log and the output files.

HTCondor creates a transaction log of everything that happens to your jobs. Looking at the log file is very useful for debugging problems that may arise.

Congratulations. You've run your first jobs in the CHTC! The parts below are detailed information about how to use HTCondor to deal with your jobs.

HTCondor Commands

HTCondor commands cheat-sheet: https://raggleton.github.io/condor-cheatsheet/

Submitting a Job: http://research.cs.wisc.edu/htcondor/manual/current/2_5Submitting_Job.html

Managing a Job: http://research.cs.wisc.edu/htcondor/manual/current/2_6Managing_Job.html

Run a docker Job: http://research.cs.wisc.edu/htcondor/manual/current/2_12Docker_Universe.html#sec:dockeruniverse http://chtc.cs.wisc.edu/docker-jobs.shtml#image

Other important commands/procedures maybe useful:

1. To ensure that HTCondor is running, you can run:

```
ps -ef | egrep condor_
```

   On a central manager machine that can submit jobs as well as execute them, there will be processes for:

```
condor_master
condor_collector
condor_negotiator
condor_startd
condor_schedd
```

2. Change HTCondor configurations(do it only if you know the effects):

```
vi /etc/condor_config
```

3. Looking at the SchedLog if you met some problems in using of HPCondor:

```
less /var/log/condor/SchedLog
```

4. Another way to debug:

```
condor_q -analyze
```

3.5 Better Deep Learning

An internal place to share deep learning modeling experience, feel free to make frequent addition/deletion of the content

3.5.1 Better Deep Learning

Configure capacity with nodes and layers

1. make sure model has sufficient capacity (can be over-trained)

Configure what to optimize with loss functions

1. distill all aspects of the model down into a single number in such a way that improvements in the number are a sign of a better model
2. understand the relationship of specified loss with objective
3. understand implication of loss
4. expect the value range of loss term is possible (possible after certain data scaling)
5. customized loss

6. weighted loss

Configure gradient precision with batch size

1. smaller batches, as noisy updates to the model weights, brings (generally) a more robust model, regularizing effects and lower generation error
2. small batch results in generally rapid learning but a volatile learning process with higher variance in the performance metrics.
3. small batches (more noisy updates) to the model generally require a smaller learning rate
4. large batch sizes may enable the use of larger learning rate

Configure the speed of learning with learning rate

1. too large learning rate could make gradient descent inadvertently increase rather than decrease the training error.
2. too large learning could lead to gradient explosion (numerical overflow), or oscillations in loss.
3. too small learning rate could result in slower training and stuck training in a region with high training error
4. too small learning rate could demonstrate over-fitting-like pattern
5. observe training it learns to quickly (sharp rise/decline and plateau) or learn to slowly (little or no change)
6. learning rate schedule
7. the method of momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients

Stabilize learning with data scaling

1. unscaled input variables can result in slow or unstable learning process
2. unscaled target variables on regression problems can result in exploding gradients causing the learning process to fail

Fixing vanishing gradients with ReLU (sigmoid and hyperbolic tangent not suitable as activations)

1. when using ReLU in the network, to avoid “dead” nodes in the begining of training,consider setting the bias to small value, such as 0.1 or 1.0
2. use ‘he-normal’ to initialize the weights
3. extensions of ReLU

Fixing exploding gradients with gradient clipping

1. a chosen vector norm and clipping gradient values that exceed a preferred range
2. only solves numerical stability issues, no implication of overall model performance
3. possible cause: learning rate that that results in large wight updates, data prep that allows large differences in the target variable, loss function that allows calculation of large error values

Accelerate learning with batch normalization

train deep neural networks with tens of layers is challenging as they can be sensitive to the initial random weights and configuration of the learning algorithm. one possible reason for this difficulty is the distribution of the inputs to layers deep in the network may change after each mini batch when the weights are updates. this can cause the learning algorithm to forever change a moving target. this change in the distribution of inputs to layers in the network is referred to by the internal covariate shift. batch normalization is a technique for training very deep neural networks that standardize the inputs to a layer for each mini batch. the weights of a layer are updated given an expectation that the prior layer outputs values with a given distribution.

1. to standardize the inputs to a network, applied to either activations of a prior layer or inputs directly accelerates training, providers some regularization effect, reducing generalizing error 2. could make network training less sensitive to weight initialization 3. probably use before the activations 4. enable the use use large learning rate (also increase decay rate if learning rate schedule is in place)

Greedy layer-wise pre-training

1.the choice of initial parameters for a deep neural network can have a significant regularizing effect (and, to a lesser extent, that it can improve optimization)

Jump start with transfer learning

Issues Log

3.5.2 Better Deep Learning

Penalize the large weights with weight regularization

Sparse representation with activity regularization

1. large activations may indicate an over-fit model
2. there is a tension between the expressiveness and the generalization of the learned features
3. encourage small activations with additional penalty
4. track activation mean value

Force small weights with weight constraints

Decouple layers with dropout

Promote robustness with Noise

Halt training at the right time with early stopping

Issues Log

3.5.3 Better Deep Learning

Reduce model variance with ensemble learning