AI System

NSCC Training

17 December 2019

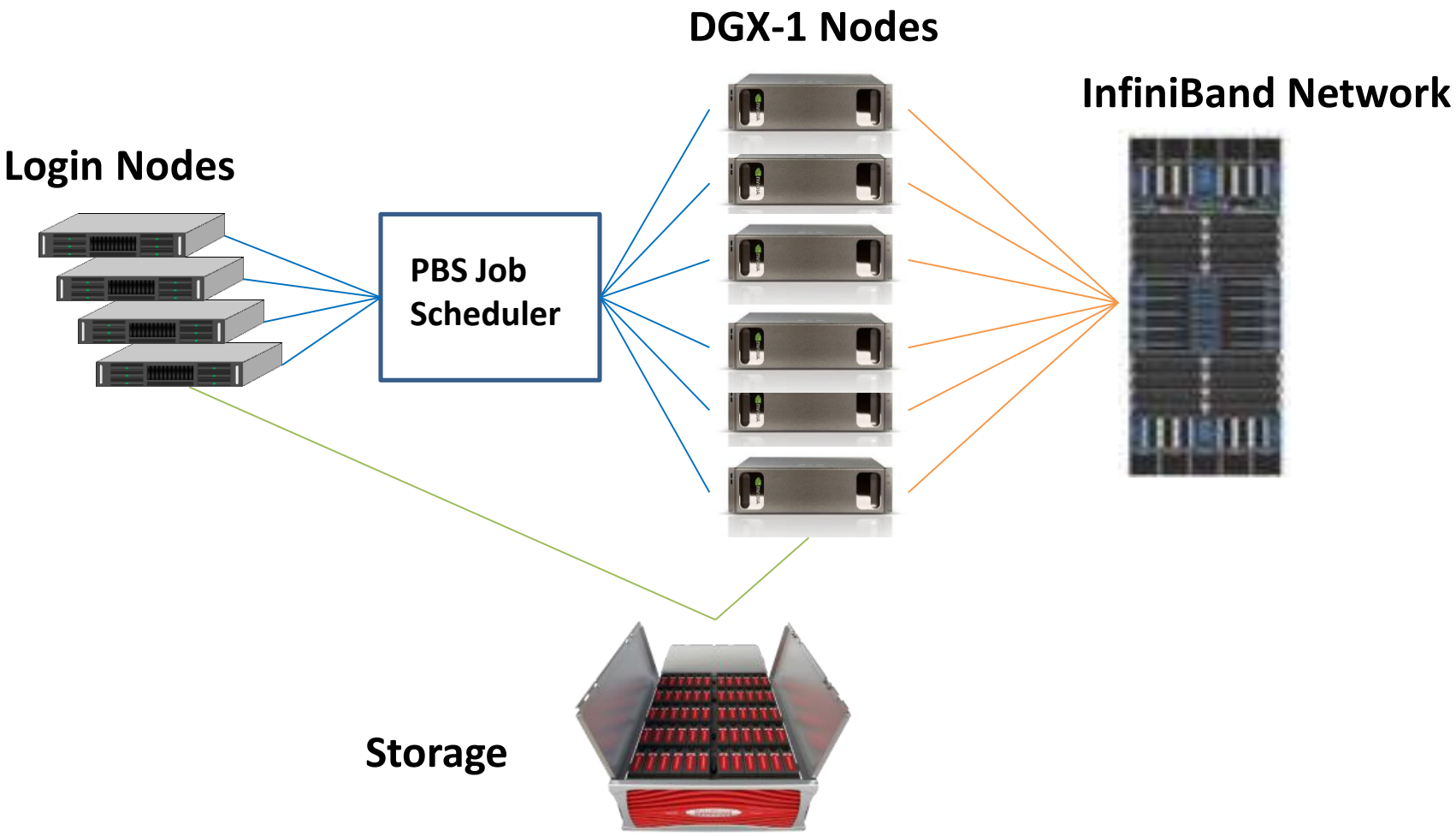
Expectations

- The DGX-1 nodes are most suited to large, batch workloads
  - e.g. training complex models with large datasets
- We encourage users to do development and preliminary testing on local resources
- Users are encouraged to use the optimized NVIDIA GPU Cloud Docker images

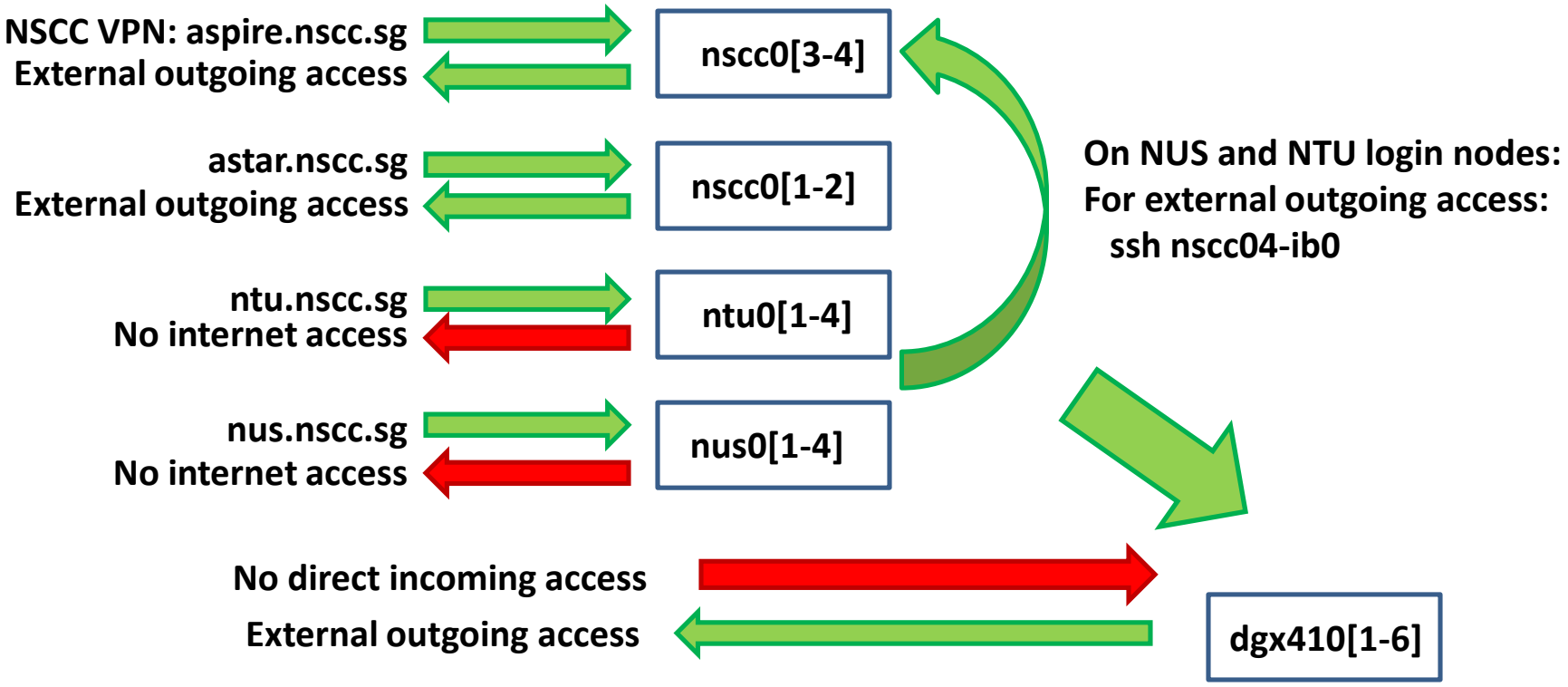
Utilisation

- Access is through **PBS job scheduler**
- We encourage workloads which can scale up to utilise all 8 GPUs on a node or run across multiple nodes
- Users can request fewer than 8 GPUs
  - Multiple jobs will then run on a node with GPU resource isolation (using cgroups)
  - You will only see the number of GPUs you request

System Overview



NSCC Networks



Project ID

- Project IDs provide access to computational resources and project storage.
- AI projects are in GPU hours
- Only AI project codes can will run on the dgx queues
- In the following material where you see *$PROJECT* replace with the code for your project, for example the stakeholder pilot project code was `41000001`

Filesystems

There are multiple filesystems available on the NSCC systems

- `/home` GPFS filesystem exported to the DGX nodes as an NFS filesystem
- `/scratch` high-performance Lustre filesystem
- `/raid` Local SSD filesystem on each on the DGX nodes

I/O intensive workloads should use either the Lustre `/scratch` filesytem or the local SSD `/raid` filesystem

	Visible on Login nodes	Visible on DGX host O/S	Visible in DGX in containers	Description
`/home/users/`*`ORG`*`/`*`USER`*	YES	YES	YES	Home directory: $HOME 50GB limit
`/home/projects/`*`$PROJECT`*	YES	YES	YES	Project directory Larger storage limits
`/scratch/users/`*`ORG`*`/`*`USER`*	YES	YES	YES	High performance Lustre filesystem. Soft linked to $HOME/scratch No quota, will be purged when filesystem is full.
`/raid/users/`*`ORG`*`/`*`USER`*	NO	YES	YES	Local SSD filesytem on each DGX node. 7TB filesystem on visible on that specific node. No quota, will be purged when filesystem is full.

Filesystems

The /home filesystem (home and project directories) is mounted and visible on all login and DGX nodes and inside Docker containers. This filesystem should be used for storing job scripts, logs and archival of inactive datasets. Active datasets which are being used in calculations should be placed on either the Lustre /scratch filesystem or the local SSD /raid filesystems.

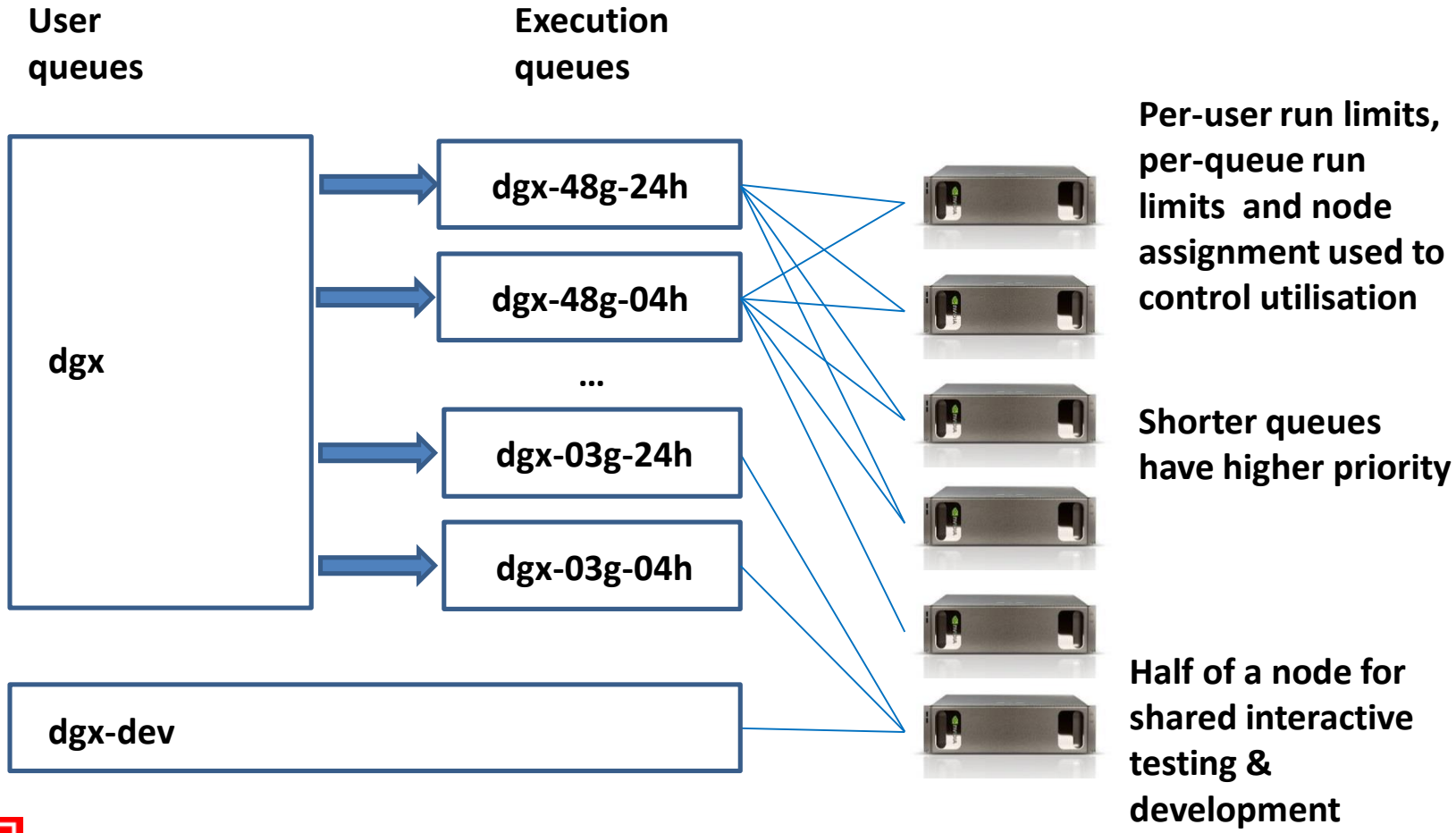
Intensive I/O workloads on large datasets should use the Lustre filesytem. The Lustre /scratch directory is now mounted directly on the DGX nodes and automatically mounted inside Docker containers (previously it was only visible on login nodes and mounted in Docker containers)

The local SSD /raid filesystem is fast but only visible on a specific DGX node. This can be used for temporary files during a run or for static long-term datasets.

Datasets with very large numbers of small files (e.g. 100,000 files which are approx. 1kB in size) **MUST** use the local SSD (/raid) filesystem or Lustre (/scratch) filesystem.

Network filesystems (/home & /scratch) are not suited to datasets which have very large number of small files because metadata operations on network filesystems are slow.

PBS Queue Configuration



Typical PBS Node Configuration

	dgx-48g-*	dgx-03g-*	dgx-dev
dgx4101	✓	×	
dgx4102	✓	×	
dgx4103	✓	×	
dgx4104	✓	×	
dgx4105	✓	×	
dgx4106 (4GPUS)	×	✓	
dgx4106 (4GPUS)			✓

- Different queues can access different sets of nodes
- Shorter queues have been given higher priority
- Configuration may be changed to match requirements based on the load in the queues

Interactive Use – Access

- Shared access to half of a DGX node (4 GPUs) is available for testing of workflows before submission to the batch queues
- To open an interactive session use the following qsub command from a login node:

```
user@nscc:~$ qsub -I -q dgx-dev -l walltime=8:00:00 -P $PROJECT
                                    # $PROJECT=41000001 or 22270170
```

- Resources are shared between all users, check activity before use
- Usage of the dgx-dev queue is **not charged** against your project quota

Interactive Use – Docker

- To run an interactive session in a Docker container then add the “-t” flag to the “nscc-docker run” command:

```
user@dgx:~$ nscc-docker run -t nvcr.io/nvidia/tensorflow:latest
$  ls
README.md  docker-examples  nvidia-examples
$ tty
/dev/pts/0
```

- The –t flag will cause job to fail if used in a batch script, only use for interactive use:

```
user@dgx:~$ echo tty | nscc-docker run -t nvcr.io/nvidia/tensorflow:latest
the input device is not a TTY
```

Batch scheduler

- Accessing the batch scheduler generally involves 3 commands:
  - Submitting a job: `qsub`
  - Querying the status of a job: `qstat`
  - Killing a job: `qdel`

```
qsub job.pbs      # submit a PBS job script to scheduler
qstat             # query the status of your jobs
qdel 11111.wlm01 # terminate job with id 11111.wlm01
```

See https://help.nscc.sg/user-guide/ for more information on how to use the PBS scheduler

Introductory workshops are held regularly, more information at https://www.nscc.sg/hpc-calendar/

Example PBS Job Script (Headers)

```
#!/bin/sh
## Lines which start with #PBS are directives for the scheduler
## Directives in job scripts are superceded by command line options passed to qsub

## The following line requests the resources for 1 DGX node
#PBS -l select=1:ncpus=40:ngpus=8

## Run for 1 hour, modify as required
#PBS -l walltime=1:00:00

## Submit to correct queue for DGX access
#PBS –q dgx

## Specify project ID
# Replace $PROJECT with Project ID such as 41000001 or 22270170
#PBS -P $PROJECT

## Job name
#PBS -N mxnet

## Merge standard output and error from PBS script
#PBS -j oe
```

Example PBS Script (Commmands)

```
# Change to directory where job was submitted
cd "$PBS_O_WORKDIR" || exit $?

# Specify which Docker image to use for container
image="nvcr.io/nvidia/tensorflow:latest"

# Pass the commands that you wish to run inside the container to the
standard input of “nscc-docker run”
nscc-docker run $image < stdin > stdout.$PBS_JOBID 2> stderr.$PBS_JOBID
```

Hands-on

`/home/projects/ai/examples`

- Example PBS job scripts to demonstrate how to:
  - submit a job to run on a DGX-1 node
  - start a container
  - run a standard MXNet training job
  - install a python package inside in a container

See https://help.nscc.sg/user-guide/ for more information on how to use the NSCC systems

Hands-on

Step 1: Log on to NSCC machine

Step 2: Run the following commands and confirm that they work:

```
cp -a /home/projects/ai/examples .
# submit first basic example
cd examples/1-basic-job && \
 qsub submit.pbs
# run a training job
cd ../../examples/2-mxnet-training && \
 qsub train.pbs
# install a python package inside container
cd ../../examples/3-pip-install && \
 qsub pip.pbs
```

Use `qstat` to check job status and when jobs have finished examine output files to confirm everything is working correctly

Partial Node Job Submission

Specify required ngpus resource in job script:

`#PBS -l select=1:ngpus=`*N*`:ncpus=`*5N*

where *N* is the number of GPUs required

e.g. "`-l select=1:ngpus=4:ncpus=20`"

```
$ echo nvidia-smi | qsub -l select=1:ncpus=5:ngpus=1 -l walltime=0:05:00 -q fj5 -P41000001
7590401.wlm01
$ grep Tesla STDIN.o7590401
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |

$ echo nvidia-smi | qsub -l select=1:ncpus=10:ngpus=2 -l walltime=0:05:00 -q fj5 -P41000001
7590404.wlm01
$ grep Tesla STDIN.o7590404
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
|   1  Tesla V100-SXM2...  On   | 00000000:0A:00.0 Off |                    0 |

$ echo nvidia-smi | qsub -l select=1:ncpus=20:ngpus=4 -l walltime=0:05:00 -q fj5 -P41000001
7590408.wlm01
$ grep Tesla STDIN.o7590408
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
|   1  Tesla V100-SXM2...  On   | 00000000:0A:00.0 Off |                    0 |
|   2  Tesla V100-SXM2...  On   | 00000000:0B:00.0 Off |                    0 |
|   3  Tesla V100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
```

NOTE THAT THE INTERACTIVE QUEUE (dgx-dev) WILL STILL GIVE SHARED ACCESS TO A SET OF GPUS ON THE TEST&DEV NODE

Checking where a job is running

4 available options to see which host a job is running on:

$ qstat -f *JOBID*

```
Job Id: 7008432.wlm01
<snip>
 comment = Job run at Wed May 30 at 13:25 on (dgx4106:ncpus=40:ngpus=8)
<snip>
```

$ qstat -wan *JOBID*

```
wlm01:
                                                                                  Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
7008432.wlm01                   fsg3            fj5             STDIN              67452    1    40    --  01:00 R 00:05:09
   dgx4106/0*40
```

$ pbsnodes -sj dgx410{1..6}

```
                                                  mem       ncpus    nmics    ngpus
vnode           state           njobs   run   susp      f/t        f/t      f/t      f/t   jobs
--------------- --------------- ------ ----- ------ ------------ -------- -------- -------- --------
dgx4101         free                 0     0      0  504gb/504gb   40/40      0/0      8/8 --
dgx4102         free                 0     0      0  504gb/504gb   40/40      0/0      8/8 --
dgx4103         free                 0     0      0  504gb/504gb   40/40      0/0      8/8 --
dgx4104         free                 0     0      0  504gb/504gb   40/40      0/0      8/8 --
dgx4105         free                 0     0      0  504gb/504gb   40/40      0/0      8/8 --
dgx4106         job-busy             1     1      0  504gb/504gb    0/40      0/0      0/8 7008432
```

$ gstat -dgx

```
# similar information to above commands but shows information on jobs from all users and is cached so has
a quicker response (but data may be up to 5 minutes old)
```

Attaching ssh Session to PBS Job

If you `ssh` to a node where you are running a job then the `ssh` session will be attached to the cgroup for your job.

If you have multiple jobs running on a node you can select which job to be attached to with the command "`pbs-attach`"

```
$ pbs-attach -l                    # list available jobs
7590741.wlm01 7590751.wlm01
$ pbs-attach 7590751.wlm01
executing: cgclassify -g devices:/7590751.wlm01 43840
```

Available workflows

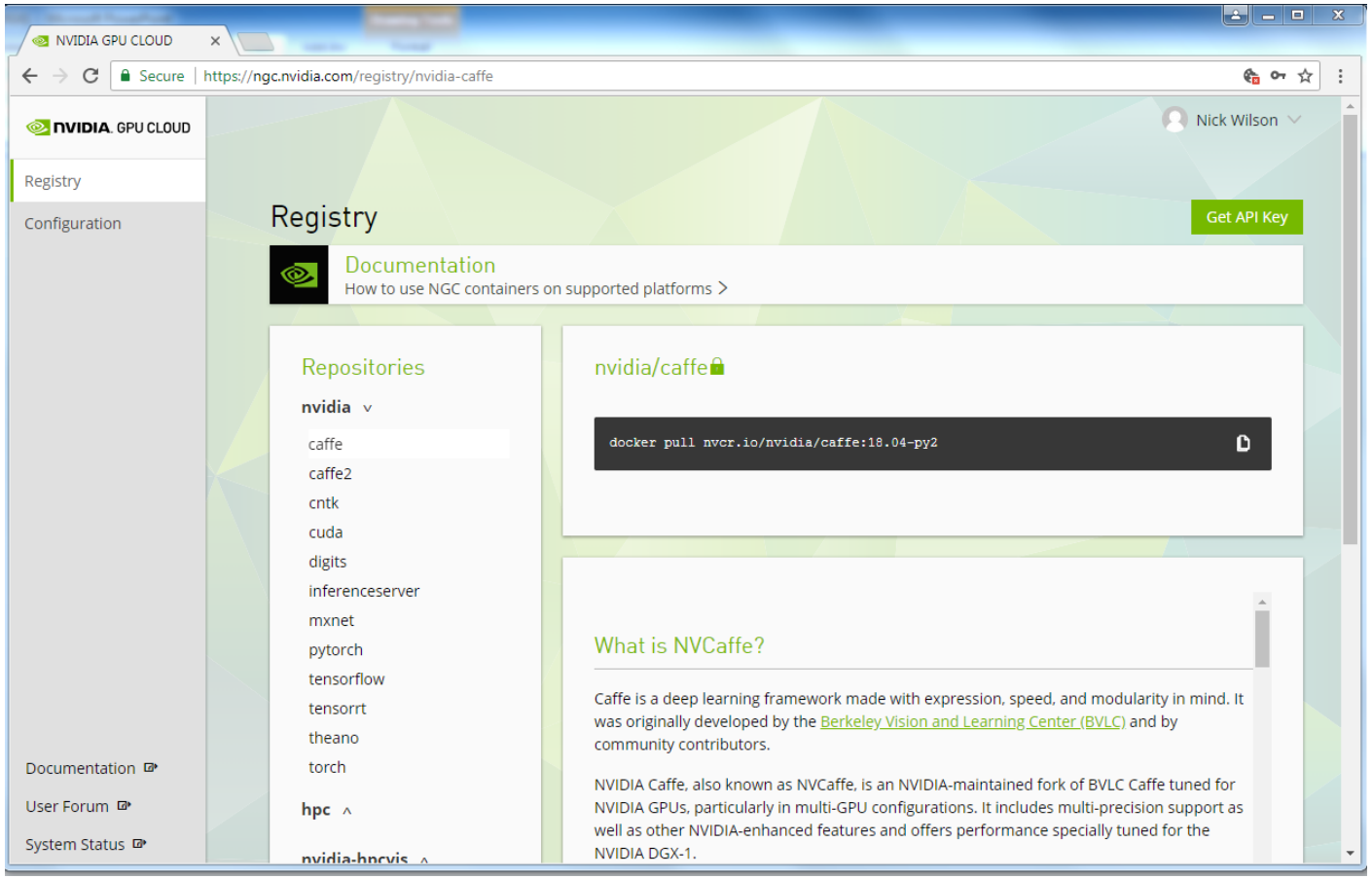
- Docker containers (recommended)
  - Optimized DL frameworks from NVIDIA GPU Cloud (fully supported)
- Singularity containers (best effort support)
  - http://singularity.lbl.gov/
- Applications installed by user in home directory (e.g. Anaconda) (best effort support)

Docker Images

- The “nscc-docker images" command shows all images currently in repository
- Currently installed includes:
  - `nvcr.io/nvidia/{mxnet,pytorch,tensorflow,tensorrt}:*`
  - `nvcr.io/nvidia/cuda:*`
- Older images will be removed if they have not been used recently, if you need a specific version then it can be pulled on request
- Contact help@nscc.sg or https://servicedesk.nscc.sg

NVIDIA GPU Cloud

To see which optimised DL frameworks are available from NVIDIA create account on https://ngc.nvidia.com/



Using Docker on the DGX-1

- Direct access to the docker command or docker group is not possible for technical reasons
- Utilities provide pre-defined templated Docker commands:

nscc-docker run *image*	nvidia-docker -u *UID*:*GID* --group-add *PROJECT* \ -v /home:/home -v /scratch:/scratch -v /raid:/raid \ --rm -i --shm-size=1g --ulimit memlock=-1 \ --ulimit stack=67108864 run *image* /bin/sh
nscc-docker images	nvidia-docker images

Docker wrapper

```
$ nscc-docker run -h

Usage: nscc-docker run [--net=host] [--ipc=host] [--pid=host] [-t] [-h] IMAGE
  --net=host   adds docker option  --net=host
  --ipc=host   adds docker option  --ipc=host
  --pid=host   adds docker option  --pid=host
  -t           adds docker option  -t
  -h           display this help and exit
  --help       display this help and exit
  --usage      display this help and exit
  --lustre     deprecated, turns on --net=host for backwards compatibility
  --ports      adds docker options -p8888:8888 -p5901:5901 ... -p5909:5909 -p6006:6006
```

The following options are added to the docker command by default:

```
-u UID:GID --group-add GROUP –v /home:/home –v /raid:/raid -v /scratch:/scratch \
--rm –i --ulimit memlock=-1 --ulimit stack=67108864
```

If --ipc=host is not specified then the following option is also added:

```
--shm-size=1g
```

Singularity

- **Singularity** is an alternative container technology
  - Can be used as a normal user
  - Commonly used at HPC sites
  - Images are flat files (or directories) rather than layers
- Latest NGC Docker images converted to Singularity images and available in:

  `/home/projects/ai/singularity`
- Example job script in:

  `/home/projects/ai/examples/singularity`

https://www.sylabs.io/docs/

https://devblogs.nvidia.com/docker-compatibility-singularity-hpc/

Multinode Training with Horovod

Horovod is a **distributed** training framework for **TensorFlow**, **Keras**, and **PyTorch**.

Can be used for:

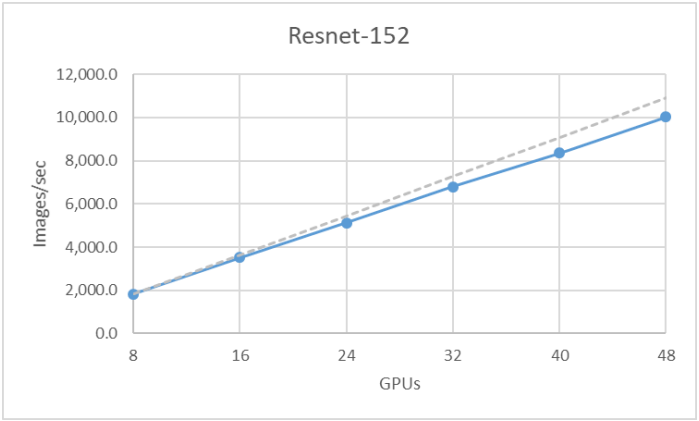
- multi-GPU parallelization in a single node
- multi-node parallelization across multiple nodes

Uses NCCL and MPI

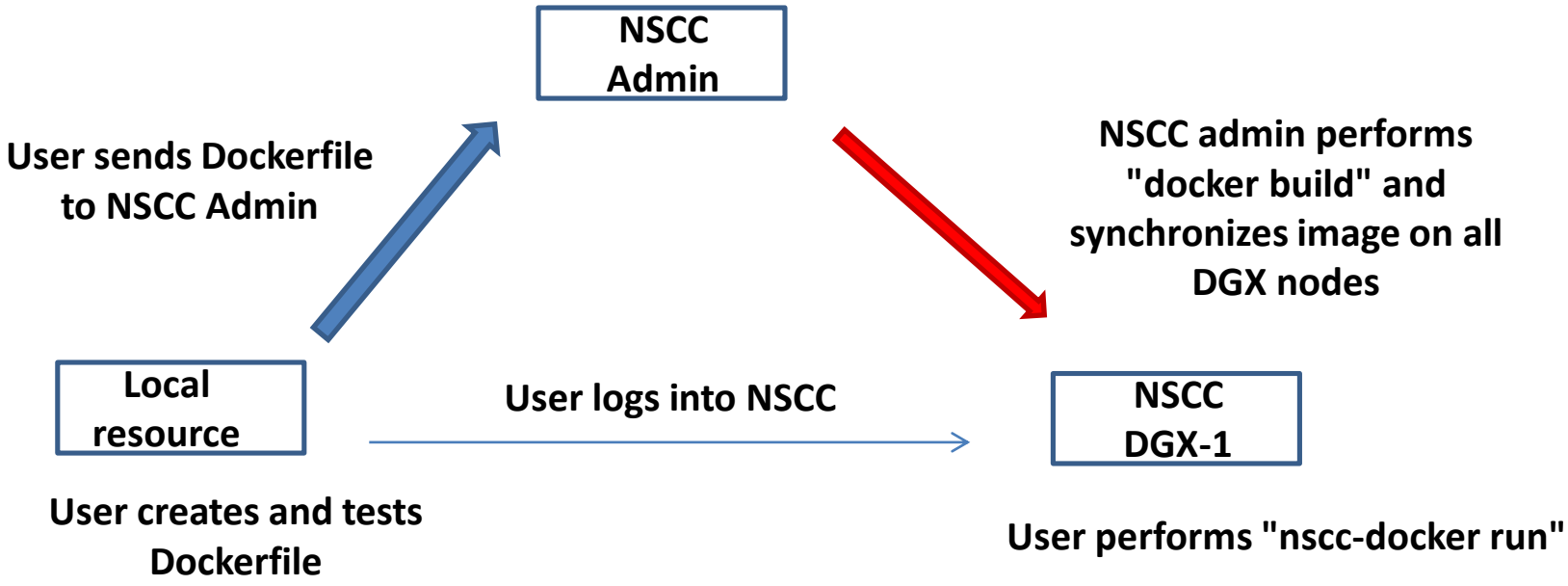
https://github.com/uber/horovod

Example job script for multi-node Horovod using Singularity to run across multiple nodes:

`/home/projects/ai/examples/horovod`

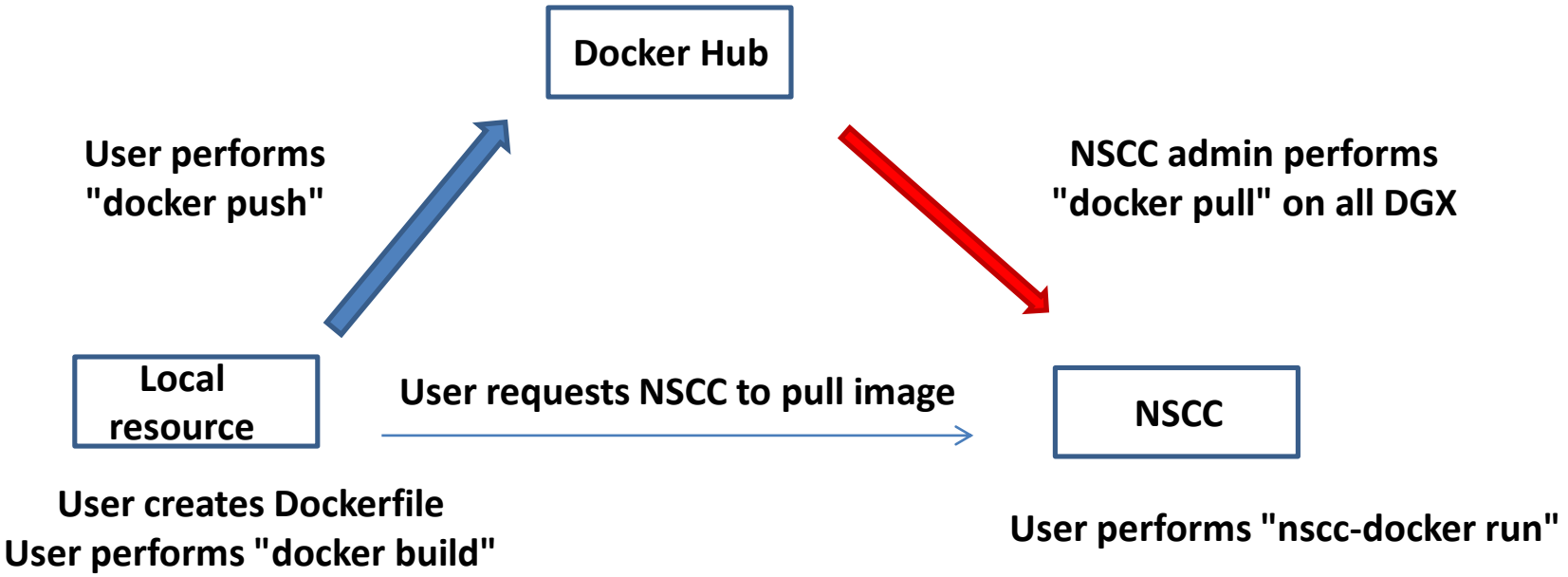


Custom Images (Method 1)



- User creates Docker image locally and sends Dockerfile to NSCC admin

Custom Images (Method 2)



- User creates Docker image locally and pushes image to Docker Hub

Custom Images (Singularity)

- Singularity can download Docker images
- Can be run as a normal user (no admin rights required)

```
$ singularity exec docker://ubuntu:18.04 echo hello world
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob sha256:7ddbc47eeb70dc7f08e410a6667948b87ff3883024eb41478b44ef9a81bf400c
 25.45 MiB / 25.45 MiB [============================================================] 25s
Copying blob sha256:c1bbdc448b7263673926b8fe2e88491e5083a8b4b06ddfabf311f2fc5f27e2ff
 34.53 KiB / 34.53 KiB [============================================================] 0s
Copying blob sha256:8c3b70e3904492c753652606df4726430426f42ea56e06ea924d6fea7ae162a1
 845 B / 845 B [============================================================] 0s
Copying blob sha256:45d437916d5781043432f2d72608049dcf74ddbd27daa01a25fa63c8f1b9adc4
 162 B / 162 B [============================================================] 0s
Copying config sha256:910b24073dd40c7834406a9bee144ae815cf81d0d648769430e58561ed99497c
 2.42 KiB / 2.42 KiB [============================================================] 0s
Writing manifest to image destination
Storing signatures
2019/12/17 16:09:13  info unpack layer: sha256:7ddbc47eeb70dc7f08e410a6667948b87ff3883024eb41478b44ef9a81bf400c
2019/12/17 16:09:14  info unpack layer: sha256:c1bbdc448b7263673926b8fe2e88491e5083a8b4b06ddfabf311f2fc5f27e2ff
2019/12/17 16:09:14  info unpack layer: sha256:8c3b70e3904492c753652606df4726430426f42ea56e06ea924d6fea7ae162a1
2019/12/17 16:09:14  info unpack layer: sha256:45d437916d5781043432f2d72608049dcf74ddbd27daa01a25fa63c8f1b9adc4
INFO:    Creating SIF file...
hello world
$ singularity build image.sif docker://ubuntu:18.04 ; \
  singularity exec image.sif echo hello world
```

Custom python packages

```
# “pip install” fails due to permissions error
# “pip install --user” installs into ~/.local
#   This is not best practice as it is external to container
#   It can also cause unexpected conflicts
# Use PYTHONUSERBASE to install packages inside container

nscc-docker run nvcr.io/nvidia/tensorflow:latest << EOF
mkdir /workspace/.local
export PYTHONUSERBASE=/workspace/.local
pip install --user scikit-learn
EOF

# Packages installed will be wiped out when container stops
# For permanent solution build a custom image
```

Custom python packages (virtualenv)

```
# Install into a virtualenv (not installed in default image)
nscc-docker run nscc/local/tensorflow:latest << EOF
virtualenv $HOME/mypthon
. $HOME/mypython/bin/activate
pip install scikit-learn
EOF
# virtualenv is in home directory so persists after container
stops
# Therefore virtualenv can be reused
# Not best practice as it affects portability and replicability
nscc-docker run nscc/local/tensorflow:latest << EOF
. $HOME/mypython/bin/activate
python script.py
EOF
```

ssh miscellany

```
# ProxyCommand  can make a 2 hop ssh connection appear direct
# On local machine do:
cat << EOF >> .ssh/config
host dgx410?
  ProxyCommand ssh aspire.nscc.sg nc %h %p
  user myusername
host aspire.nscc.sg
  user myusername
EOF
# and then you can ssh to remote machine with one command:
ssh dgx410X

# Port forwarding (forwards a port on local host to a port on remote host)
# Replace X with 1 to 6 depending on target
ssh -L8888:dgx410X:8888 aspire.nscc.sg
```

Jupyter notebook

Instructions in /home/projects/ai/guides/jupyter

On DGX start container running jupyter

Note different wrapper command which opens ports 8888 and 5901 in container

```
nscc-docker run --ports nvcr.io/nvidia/tensorflow:latest << EOF
export PYTHONUSERBASE=/workspace/.local
mkdir \$PYTHONUSERBASE
pip install --user --no-cache-dir jupyter
PATH=\$PYTHONUSERBASE/bin:\$PATH
export PATH
jupyter notebook --ip=0.0.0.0 --port=8888
EOF
```

On local machine use ssh port forwarding to tunnel to DGX (replace *X* with 1 to 6)

```
ssh -L8888:dgx410X:8888 aspire.nscc.sg
```

On local machine go to http://localhost:8888 and use token from container

Alternatively use a reverse proxy through a mutually accessible machine

VNC server in container

Instructions in /home/projects/ai/guides/vncserver

Install VNC server inside container . For example add following to Dockerfile:

```
FROM nvcr.io/nvidia/tensorflow:latest
RUN apt-get update
RUN apt-get install -y tightvncserver
```

Start vncserver on display :1 in container with port 5901 exposed:

```
nscc-docker run --ports nscc/local/tensorflow:latest << EOF
vncserver :1
sleep 900 # Replace this with actual tasks in real case
EOF
```

On local machine use ssh port forwarding to tunnel to DGX (replace *X* with 1 to 6)

```
ssh -L5901:dgx410X:5901 aspire.nscc.sg
```

and then connect to display :1 (port 5901) on localhost in VNC client