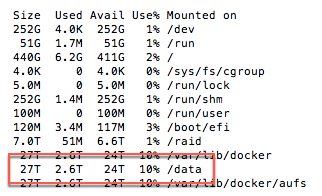
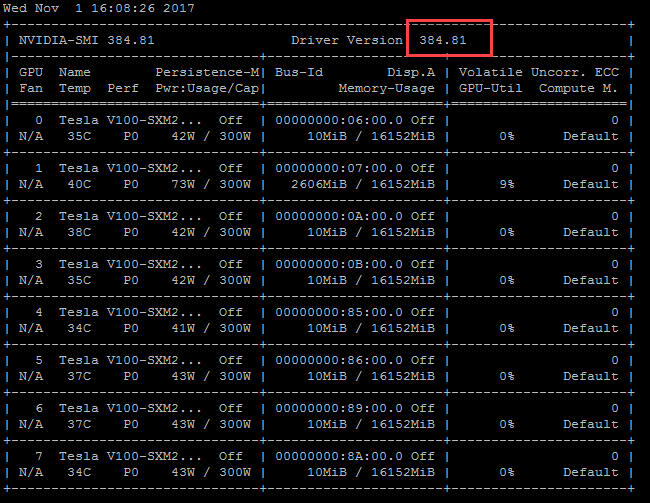
DGX-1 details

Storage for your data in the highlighted path, /data. It is an DGX build-in SSD cached NFS drive with a storage limit of 8TB.



Checking available GPU  
nvidia-smi  


Processing sensitive data in DGX-1  
  
It may come at the expense of development speed and convenience, hence, one “solution” is to do the self-monitoring by each team to ensure that their data are only accessible by the team members who really need to do so. Do not store your data in raw text format, please convert the file to binary type file, example TFRecords or compress zip with a password using Linux zip.

Example,

1. Prompt for password when you code start

2. Save the password or key in the memory

3. Unzip file to memory using password (In memory)

Sample of reading Zipfile to memory, do modify the codes and adapt it to your runtime code.

from zipfile import ZipFile

from StringIO import StringIO

import getpass

def unzip\_string(zipped\_string):

    unzipped\_string = ''

    zipfile = ZipFile(StringIO(zipped\_string), mode='r')

   pw = getpass.getpass()

    for name in zipfile.namelist():

        unzipped\_string += zipfile.open(name, pwd=pw, mode='r').read()

    return unzipped\_string

Reading a pandas dataframe directly

import getpass

import pandas as pd

from zipfile import ZipFile

def read\_df\_from\_zip(f, dtype=None):

"""

Assumes that the file we want has the path

\*/{file\_name}.zip

"""

print 'Password for the zip file'

pw = getpass.getpass()

z = ZipFile(f, mode='r')

# some logic to automatically get the csv file name in the zip archive

csv\_filename = f.split('/')[-1].split('.')[0] + '.csv'

df = pd.read\_csv(z.open(csv\_filename, pwd=pw, mode='r'), dtype=dtype)

print 'Successfully loaded %s' % f

return df

For large archive file with n number of file, you can extract the compress files sequentially or in batch. You will need to change the loop condition, do modify the codes and adapt it to your runtime code.

from zipfile import ZipFile

from PIL import Image

import getpass

pwd = getpass.getpass()

zippedImgs = ZipFile('100000-Test.zip', mode='r')

for i in range(len(zippedImgs.namelist())):

print ("iter", i, " ",)

file\_in\_zip = zippedImgs.namelist()[i]

if (".jpg" in file\_in\_zip or ".JPG" in file\_in\_zip):

print ("Found image: ", file\_in\_zip, " -- ",)

data=zippedImgs.open(file\_in\_zip, pwd=byte(pw,'utf-8'), mode='r').read()

img = Image.open(data)

else:

print ("")

Nvidia Optimized Docker Images

lssinh010.sin.sap.corp/nvidia/cuda  
lssinh010.sin.sap.corp/nvidia/tensorflow  
lssinh010.sin.sap.corp/nvidia/tensorflow-py2  
lssinh010.sin.sap.corp/nvidia/pytorch  
lssinh010.sin.sap.corp/nvidia/torch  
lssinh010.sin.sap.corp/nvidia/cntk  
lssinh010.sin.sap.corp/nvidia/caffe  
lssinh010.sin.sap.corp/nvidia/caffe2  
lssinh010.sin.sap.corp/nvidia/mxnet  
lssinh010.sin.sap.corp/nvidia/theano  
lssinh010.sin.sap.corp/nvidia/tensorrt

The tensorflow image was officially update with Python3, cuda9.

If you need to use Python 2, cuda 9. Do pull the correct version from the list

Still running on cuda8

If you want to use Tensorflow for Python 3 with cuda 8, please build your image from the following image:

lssinh010.sin.sap.corp/nvidia/tensorflow:cuda8  
lssinh010.sin.sap.corp/nvidia/cuda: cuda8

Fixed DGX-1 GPU Allocation  
Please share the GPU with your teammates.

GPU 0 – ML People SGP

GPU 2 – ML DigitCore&Fin SGP

GPU 4 – ML Sales&Service SGP

GPU 6 - ML Foundation SGP

GPU 7 – Sale Demo/ Fun projects/ PhD innovation projects

Dynamic DGX-1 GPU Allocation  
Be fair, do not lock the GPU for more than 36 hrs. Upon request GPU 1, 3, 5 should be released for other Team or ad-hoc project usage (PhD, Sales, etc)

GPU 1 – ML People SGP, ML DigitCore&Fin SGP

GPU 3 – ML Sales&Service SGP, ML Foundation SGP

GPU 5 – Sale Demo/ Fun projects/ PhD innovation projects

Note: If you are in urgent need to use more GPUs for an extended duration, please sent an email to “DL ML Singapore LSSINH031 users” and ask another team to release the dynamic GPUs for your usages. Please state the GPU and the duration.  
Example,

Hi SG DGX-1 Users,

I need to run my project/testing on:

GPUs: 1,3,5

Duration: 31st July 0000 hrs to 2nd Aug 2359 hrs

Thanks.

Regards,

Someone In Need of GPUs

Docker file Environment and Cleanup

The DC http environment is behind a proxy, thus we need to add the following ENV variable at the start of you dockerfile

ENV http\_proxy=<http://proxy.sin.sap.corp:8080>ENV HTTP\_PROXY=[http://proxy.sin.sap.corp:8080](http://proxy:8080/)ENV ftp\_proxy=<ftp://proxy.sin.sap.corp:8080>ENV FTP\_PROXY=<ftp://proxy.sin.sap.corp:8080>ENV https\_proxy=<https://proxy.sin.sap.corp:8080> #Can be exclude  
ENV HTTPS\_PROXY=<https://proxy.sin.sap.corp:8080> #Can be exclude

# Change the GPU ID below, currently set to 0  
# for multi-GPU, if you want to use GPIs 0 and 1, use "0,1"  
ENV CUDA\_VISIBLE\_DEVICES=0

# Clean up APT when done  
RUN apt-get clean && rm -rf /var/lib/apt/lists/\* /tmp/\* /var/tmp/\*

Docker command  
Create Docker images,  
<https://docs.docker.com/engine/getstarted/step_four/>

mkdir dockerfile

cd dockerfile

*vim Dockerfile 🡨 Create your own Docker file*  
docker build --force-rm=true --no-cache=true -t i310616\_<SomeDetails> .

Please use docker build with rm or force-rm=true switch to remove intermediate containers during your docker build it will help keep the server with less cluster of unused containers.  
  
Running docker container with mount host path, make sure you put a name/inumber for your docker container,

To share data between ranks, NCCL may require shared system memory for IPC and pinned (page-locked) system memory resources. The operating system’s limits on these resources may need to be increased accordingly. Refer to your system’s documentation for details. In particular, Docker containers default to limited shared and pinned memory resources. When using NCCL inside a container, following values were hardcode during docker start up, “--shm-size=1g --ulimit memlock=-1”

nvidia-docker run --name i310616\_GPU0 -e NCCL\_TOPOLOGY=CUBEMESH -v /data/somepath:/<insidedocker>/path <imageName> /bin/bash

Docker detached mode

nvidia-docker run --name i310616\_GPU0 -e NCCL\_TOPOLOGY=CUBEMESH -d -it -v /data/somepath:/<insidedocker>/path <imageName> /bin/bash

Attaching to detached container,

docker exec -i -t <containerID> /bin/bash #by ID

docker exec -i -t <containerName> /bin/bash #by Name

Committing/save change in container to image,

docker commit <containerID> imageName:lastest or docker commit <containerID> imageName:V2

Stopping docker container,

docker stop <containerID>

Removing docker container after you exit,   
docker ps -a 🡨 Find the docker container ID

docker rm -v <containerID> 🡨Remove the docker container

Removing unnecessary images,

docker images 🡨 Find the docker image ID

docker rmi <imageID> 🡨Remove the docker image

Compressing Big Image,

The Docker container with the image must be running, then you run the command to remove the history of the docker images.

docker export <CONTAINER ID> | docker import - some-image-name:latest

<https://tuhrig.de/flatten-a-docker-container-or-image/>

After you compress your image, a <none> <none> temp image was created during the process, kindly remove the image using docker rmi <imageID>

To keep some of the history, if you wanted.

 docker run --rm -v /var/run/docker.sock:/var/run/docker.sock nate/dockviz images -t

mentioned in

<http://stackoverflow.com/questions/32454679/how-to-see-tree-view-of-docker-images>

GPU Isolation

Running command inside Docker with GPU isolation,

If you are using nvidia-docker,

NV\_GPU=0 nvidia-docker run -ti --rm nvidia/cuda

If you are using notebook,

import os

os.environ["CUDA\_DEVICE\_ORDER"]="PCI\_BUS\_ID"

os.environ["CUDA\_VISIBLE\_DEVICES"]="0"

You can double check that you have the correct devices visible to TF

from tensorflow.python.client import device\_lib

print device\_lib.list\_local\_devices()

If you are running py file added the following,

username@server:/scratch/coding/src$ CUDA\_VISIBLE\_DEVICES=1 python my\_script.py

Removing your python process

You can kill your own python process that is using the GPU easily. Just go into your docker container,

Do the following,

Get the process ID

Ps -aux | grep <YourScriptName>

Kill the python process ID

Kill -9 <processID>

Cleaning of Exited Containers

Run the following command to get the list of containers,

docker ps -a

Find the container ID that belong to you and remove those exited container,

Docker rm -v <containerID>

GPU memory usage

Code snippet to better control the GPU memory usage,

Example highlighted in yellow,

import tensorflow as tf

import keras.backend as K

tfconfig = tf.ConfigProto(allow\_soft\_placement=True)

tfconfig.gpu\_options.allow\_growth=True

sess = tf.Session(config=tfconfig)

K.set\_session(sess)  
  
Reference from:

<https://github.com/tvkpz/handy_scripts/blob/master/tensorflow_allow_growth.py>

System memory usage

To allow better GC for python manual de reference of object is necessary, and call the gc.collect() function will allow python to clear away unnecessary object from the memory.

import gc

import resource

import os

import psutil

import multiprocessing as mp

process = psutil.Process(os.getpid())

def mem():

print('Memory usage, physical memory: %d' % process.memory\_info().rss)

print('Memory usage, virtual memory: %d' % process.memory\_info().vms)

print('Memory usage, physical memory percentage: %d' % process.memory\_percent(memtype="rss"))

mem()

def memoryhog():

print('...creating list of dicts...')

n = 10\*\*5

l = []

for i in range(n):

a = 1000\*'a'

b = 1000\*'b'

l.append({ 'a' : a, 'b' : b })

print('Reference to the object L: %s' % gc.get\_referrers(l))

l = None

mem()

print('GC collected objects : %d' % gc.collect())

proc = mp.Process(target=memoryhog)

proc.start()

proc.join()

proc = None

mp = None

print('GC collected objects : %d' % gc.collect())

mem()

Using screen

To start a screen,

screen –S myfirstscreen

To detach from screen,

Crtl A + D

To list all screen,

screen –ls

To attached to previous screen

screen -r <- If you only have a single screen

screen -ls

There is a screen on:

        13358.pts-1.LSSINH031   (09/20/2017 02:11:37 PM)        (Attached)

1 Socket in /var/run/screen/S-root.

screen –r 13358.pts-1.LSSINH031

To exit screen,

Type ‘exit’ when you are in the screen.

Screen Help List

root@LSSINH031:/home/sysadmin# screen -h

Use: screen [-opts] [cmd [args]]

 or: screen -r [host.tty]

Options:

-4            Resolve hostnames only to IPv4 addresses.

-6            Resolve hostnames only to IPv6 addresses.

-a            Force all capabilities into each window's termcap.

-A -[r|R]     Adapt all windows to the new display width & height.

-c file       Read configuration file instead of '.screenrc'.

-d (-r)       Detach the elsewhere running screen (and reattach here).

-dmS name     Start as daemon: Screen session in detached mode.

-D (-r)       Detach and logout remote (and reattach here).

-D -RR        Do whatever is needed to get a screen session.

-e xy         Change command characters.

-f            Flow control on, -fn = off, -fa = auto.

-h lines      Set the size of the scrollback history buffer.

-i            Interrupt output sooner when flow control is on.

-l            Login mode on (update /var/run/utmp), -ln = off.

-ls [match]   or -list. Do nothing, just list our SockDir [on possible matches].

-L            Turn on output logging.

-m            ignore $STY variable, do create a new screen session.

-O            Choose optimal output rather than exact vt100 emulation.

-p window     Preselect the named window if it exists.

-q            Quiet startup. Exits with non-zero return code if unsuccessful.

-r [session]  Reattach to a detached screen process.

-R            Reattach if possible, otherwise start a new session.

-s shell      Shell to execute rather than $SHELL.

-S sockname   Name this session <pid>.sockname instead of <pid>.<tty>.<host>.

-t title      Set title. (window's name).

-T term       Use term as $TERM for windows, rather than "screen".

-U            Tell screen to use UTF-8 encoding.

-v            Print "Screen version 4.01.00devel (GNU) 2-May-06".

-wipe [match] Do nothing, just clean up SockDir [on possible matches].

-x            Attach to a not detached screen. (Multi display mode).

-X            Execute <cmd> as a screen command in the specified session.

**Setting up a remote Jupyter server**

On the DGX:

1. In the DGX, create the container and make sure you forward the port from the container to the server using the –p flag:

$ nvidia-docker run --name <container> -d –it –p 8888:8888 -v /server/path:/container/path <image>:<tag> /bin/bash

1. Exit from the container
2. Create a screen session

$ screen –S jupyter

1. Run the container

$ docker exec –it <container> /bin/bash

1. Start the jupyter server

$ jupyter notebook –-no-browser –-alow-root –-ip=0.0.0.0

On your local machine (for Mac users):

1. In your ~/.bash\_profile or ~/.zshrc, add the following alias

alias jupyterdgx='ssh -N -f -L 7777:localhost:8888 <inumber>@lssinh031.sin.sap.corp -v'

1. In your local terminal, run

$ jupyterdgx

1. Point your browser to localhost:7777

Using private repository, lssinh010.sin.sap.corp

1. Build your required image,  
   $ docker build --force-rm=true --no-cache=true -t i311828\_lr\_py3 .
2. Tag build image to lssinh010.sin.sap.corp/mlsgdgx1

$ docker tag i311828\_lr\_py3 lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr\_py3  
$ docker rmi i311828\_lr\_py3

1. Push image to repo  
   $ docker push lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr\_py3
2. Visit this link to check:  
   <https://lssinh010.sin.sap.corp>  
   User: defaultuser  
   Pwd: Mcw8fyTyuG85F4TZuhtv
3. Check you docker images  
   $ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr\_py3 latest 28b9bb3e20f2 4 minutes ago 5.83GB

i342173 latest d3f03e97dc3f 20 minutes ago 5.78GB

i334506\_gpu0 latest ee07266cb94f 43 hours ago 5.32GB

lssinh010.sin.sap.corp/mlsgdgx1/i330688\_cashapp latest ba0343e71654 2 days ago 4.74GB

lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr latest 7d9b309570e7 2 days ago 5.32GB

lssinh010.sin.sap.corp/nvidia/tensorflow latest 9b6a599f403c 3 weeks ago 3.89GB

lssinh010.sin.sap.corp/nvidia/tensorflow cuda8 94b1afe1821c 4 months ago 4.4GB

1. Run image from tag image,  
   $ nvidia-docker run -it -d --name i311828\_lr\_py3 -p 8888:8888 -p 6006:6006 -e NCCL\_TOPOLOGY=CUBEMESH -v /home/i311828:/root/i311828/ -v /data/i311828:/data/ lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr\_py3 /bin/bash

$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

694598270d7f lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr\_py3 "/usr/local/bin/nv..." 5 seconds ago Up 4 seconds 6006/tcp i311828\_lr\_py3

d2dec1f3dc2e i342173 "/usr/local/bin/nv..." 3 minutes ago Up 3 minutes 6006/tcp, 0.0.0.0:889->8888/tcp i342173\_GPU4

aa0b6ef37b54 i334506\_gpu0 "/usr/local/bin/nv..." 41 hours ago Up 41 hours 6006/tcp, 0.0.0.0:7500->7500/tcp i334506\_jupyter

88b3173dd064 lssinh010.sin.sap.corp/mlsgdgx1/i330688\_cashapp:latest "/usr/local/bin/nv..." 42 hours ago Up 42 hours 0.0.0.0:6006->6006/tcp, 0.0.0.0:8888->8888/tcp i330688\_GPU1

45a506492b53 lssinh010.sin.sap.corp/mlsgdgx1/i311828\_lr "/usr/local/bin/nv..." 2 days ago Up 2 days 6006/tcp i311828\_lr

Removing <none> image after committing and pushing to image private repo  
After you tag the images to lssinh010, your will still need to push any committed image to the private repo so that it is update with your changes.

1. Commit container to image  
   docker commit <containerID> imageName:latest or docker commit <containerID> imageName:V2
2. Push docker image to private repo

docker push imageName:lastest

1. Stop and remove container to allow the removal of images.

docker stop <containerID>

docker rm -v <containerID>

1. Remove first layer image follow by the none image  
   docker rmi imageName:latest

docker rmi imageName:<none> or docker rmi <containerID>

1. If required, pull the images to the server if you need to use the committed images