HW-3: Running MNIST on Docker & Singularity

1. INTRODUCTION

In modern computing, the emergence of virtualization and containers have changed how software is developed,deployed and maintained. As computing needs of quickly growing softwares increases, virtualization and containers are heavily used for efficient use of resources and providing flexibility in deployment. This report created as part of *Homework-3* talks about running MNIST digit recognition in a Docker container on my local machine and also running this in a Singularity container.

2. BACKGROUND STUDY

In the past, servers served as a single machine which had their own operating system, over which applications were installed. Such servers were called bare-metal machines and were used to run some specific application. However, as businesses grew, the cost to buy these serves and also store them became increasingly high. There were often dependency conflicts when more than one application was run on the same server, and it was not always possible to run more than one application on the same machine. This paved the way for virtualization.

Virtualization is the abstraction of compute resources like CPU,RAM,memory etc. among various virtual machines. It allows for partitioning of a single physical computer or server into several virtual machines (VM). Each VM can then interact independently and run different operating systems or applications while sharing the resources of a single computer. The hypervisor acts as an intermediary between virtual machines and the underlying hardware, allocating host resources such as memory, CPU, and storage. One of the major advantages of virtualization is the reduced infrastructure cost of maintaining and hosting multiple servers.We can also easily create, destroy and migrate VMs between different hosts, making it easier to scale and manage your computing resources.

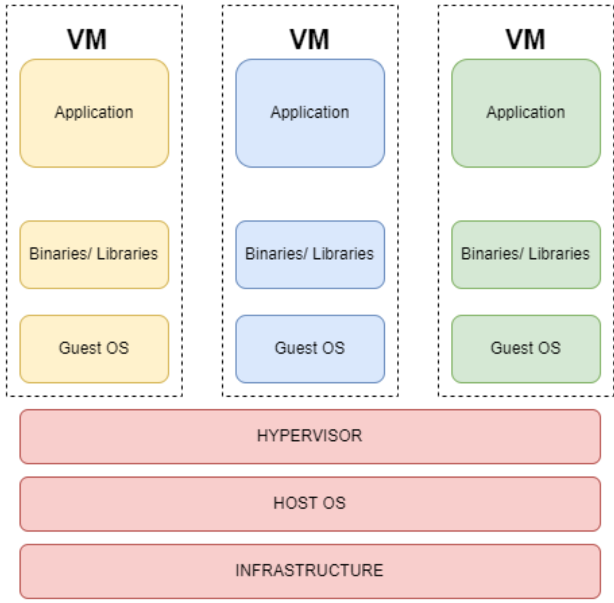


Figure 1: Virtual Machine Architecture

Containers are built on the concept of virtualization and are used to run many instances on a single physical host machine. However, instead of introducing a hypervisor as an intermediary, containers make use of the host machine kernel to isolate the multiple independent instances or containers.Containers are great due to their lightweight architecture and efficiency. They also allow quick application deployment.

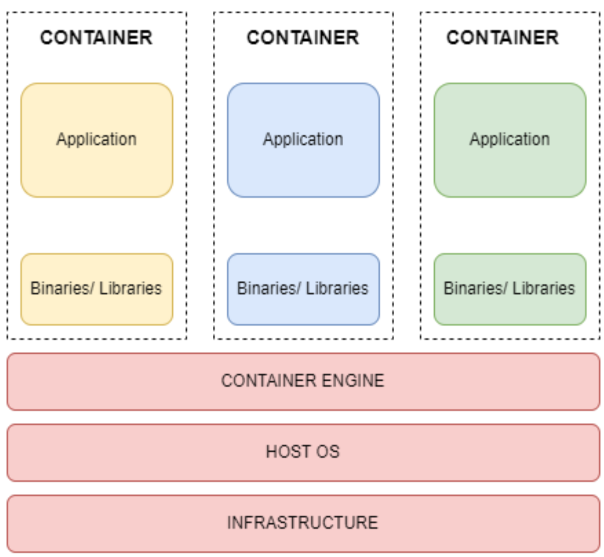


Figure 2: Container Architecture

Virtualization increases the overhead because it adds an additional layer between the host and the OS of the VM. Depending on the workload there might also be a decrease in the performance.
On the other hand, containers share the host kernel and operating system,avoiding the overhead of virtualization, as there's no need to provide a separate virtual kernel and OS for each instance. Containers are thus a more lightweight solution of the virtualization problem.

3. RUNNING MNIST IN DOCKER ON LOCAL MACHINE (LAPTOP)

1. Download & Install Docker for Windows. I used the WSL backend instead of the Hyper-V backend while installing Docker.

2. My Windows laptop doesn't have a GPU. Since training would take a lot of time here,I did a representative model run with just 1 training epoch. I also specified the cuda and mps command line arguments to false to disable CUDA and macOS GPU training.
   Command used-

```
python mnist.py --batch-size 32 --test-batch-size 32
--epochs 1 --no-cuda --no-mps
```

3. Create a Docker File
   Docker can build by reading the instructions from a Dockerfile which is a text document that contains all the commands a user could call on the command line to assemble an image.

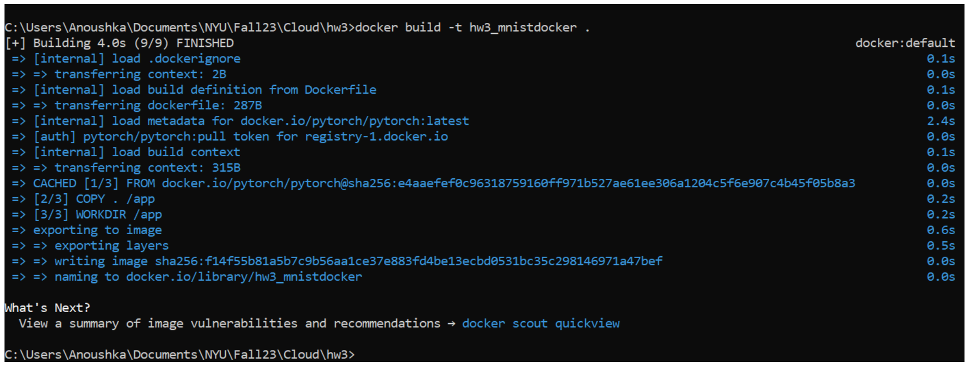
Dockerfile

```
#Use the Pytorch image
FROM pytorch/pytorch

# Copy the all source code into the container.
COPY . /app
WORKDIR /app

USER root

# Run the application.
CMD python mnist.py --batch-size 32 --test-batch-size 32
--epochs 1 --no-cuda --no-mps
```

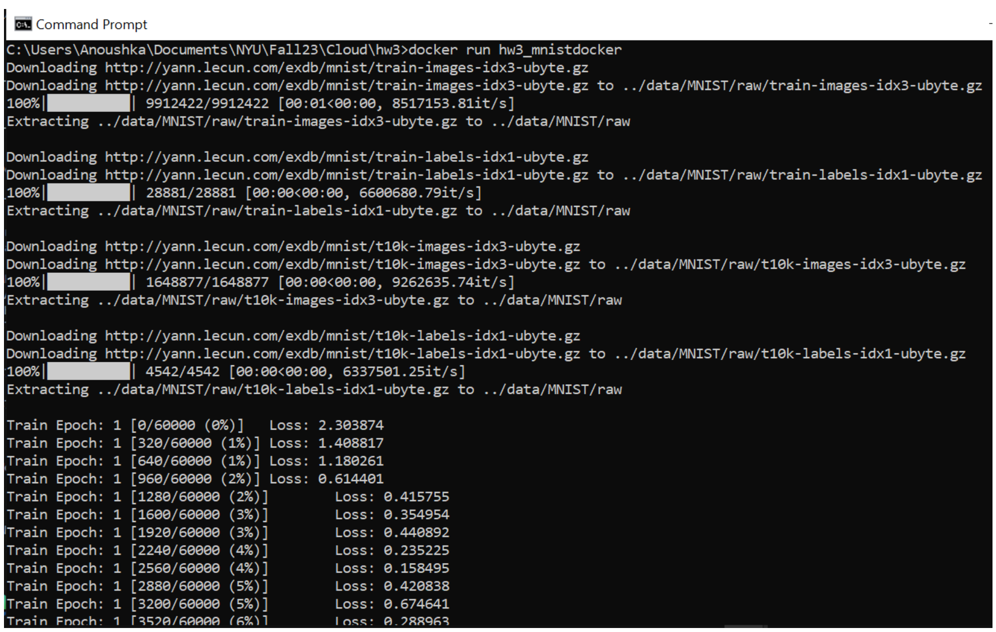
Line 11, Column 87   Tab Size: 4   Plain Text

- The *FROM* instruction is used to initialize a build stage and set the base image. Here I have pulled the latest pytorch image which has all the required dependencies to run the mnist.py script.
- Copy all the source code into the container using the *COPY* instruction
- Use *USER* instruction to set root as the user.
- To run the application, use the *CMD* instruction followed by the python command.

4. Build an image from the dockerfile
   To build an image I used the *docker build* command with the tag option to give a name to my docker image.
   Command used: *docker build -t hw3_mnistdocker .*



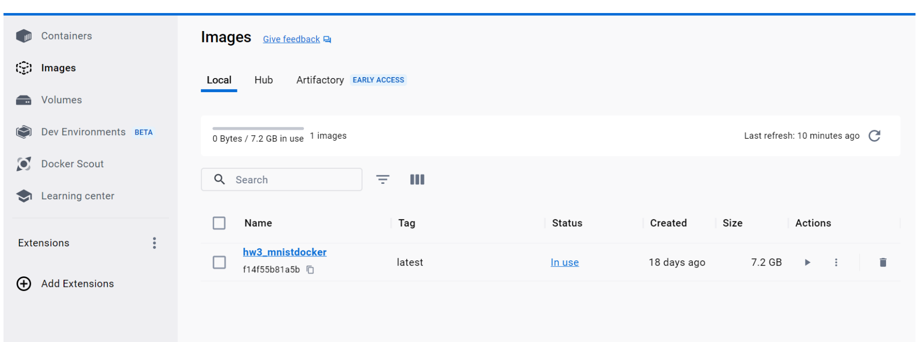
5. Create and run a container
   Use the *docker run* command to create and run a container using the image we just built.
   Command used: *docker run hw3_mnistdocker*



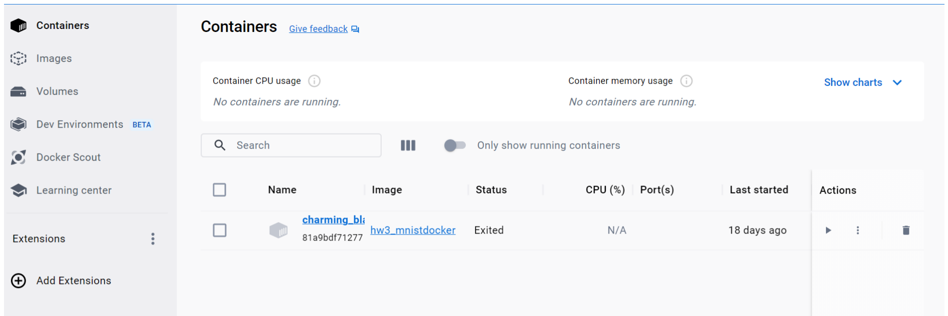
```
Command Prompt
Train Epoch: 1 [3520/60000 (6%)]	Loss: 0.288963
Train Epoch: 1 [3840/60000 (6%)]	Loss: 0.098887
Train Epoch: 1 [4160/60000 (7%)]	Loss: 0.280345
Train Epoch: 1 [4480/60000 (7%)]	Loss: 0.648713
Train Epoch: 1 [4800/60000 (8%)]	Loss: 0.141194
Train Epoch: 1 [5120/60000 (9%)]	Loss: 0.267053
Train Epoch: 1 [5440/60000 (9%)]	Loss: 0.209678
Train Epoch: 1 [5760/60000 (10%)]	Loss: 0.259347
Train Epoch: 1 [6080/60000 (10%)]	Loss: 0.516709
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.174707
Train Epoch: 1 [6720/60000 (11%)]	Loss: 0.128012
Train Epoch: 1 [7040/60000 (12%)]	Loss: 0.222974
Train Epoch: 1 [7360/60000 (12%)]	Loss: 0.146631
Train Epoch: 1 [7680/60000 (13%)]	Loss: 0.161312
Train Epoch: 1 [8000/60000 (13%)]	Loss: 0.240708
Train Epoch: 1 [8320/60000 (14%)]	Loss: 0.164778
Train Epoch: 1 [8640/60000 (14%)]	Loss: 0.294474
Train Epoch: 1 [8960/60000 (15%)]	Loss: 0.257723
Train Epoch: 1 [9280/60000 (15%)]	Loss: 0.215883
Train Epoch: 1 [9600/60000 (16%)]	Loss: 0.139093
Train Epoch: 1 [9920/60000 (17%)]	Loss: 0.091032
Train Epoch: 1 [10240/60000 (17%)]	Loss: 0.290568
Train Epoch: 1 [10560/60000 (18%)]	Loss: 0.082904
Train Epoch: 1 [10880/60000 (18%)]	Loss: 0.309875
Train Epoch: 1 [11200/60000 (19%)]	Loss: 0.293838
Train Epoch: 1 [11520/60000 (19%)]	Loss: 0.282038
Train Epoch: 1 [11840/60000 (20%)]	Loss: 0.084676
Train Epoch: 1 [12160/60000 (20%)]	Loss: 0.257291
Train Epoch: 1 [12480/60000 (21%)]	Loss: 0.518006
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.122288
Train Epoch: 1 [13120/60000 (22%)]	Loss: 0.437862
Train Epoch: 1 [13440/60000 (22%)]	Loss: 0.170447
Train Epoch: 1 [13760/60000 (23%)]	Loss: 0.095377
Train Epoch: 1 [14080/60000 (23%)]	Loss: 0.060325
Train Epoch: 1 [14400/60000 (24%)]	Loss: 0.108839
Train Epoch: 1 [14720/60000 (25%)]	Loss: 0.190348
Train Epoch: 1 [15040/60000 (25%)]	Loss: 0.111847
Train Epoch: 1 [15360/60000 (26%)]	Loss: 0.124302
Train Epoch: 1 [15680/60000 (26%)]	Loss: 0.071424
Train Epoch: 1 [16000/60000 (27%)]	Loss: 0.234130
Train Epoch: 1 [16320/60000 (27%)]	Loss: 0.015247
```

```
Command Prompt
Train Epoch: 1 [53120/60000 (89%)]	Loss: 0.040489
Train Epoch: 1 [53440/60000 (89%)]	Loss: 0.300219
Train Epoch: 1 [53760/60000 (90%)]	Loss: 0.026834
Train Epoch: 1 [54080/60000 (90%)]	Loss: 0.109451
Train Epoch: 1 [54400/60000 (91%)]	Loss: 0.005509
Train Epoch: 1 [54720/60000 (91%)]	Loss: 0.005017
Train Epoch: 1 [55040/60000 (92%)]	Loss: 0.069018
Train Epoch: 1 [55360/60000 (92%)]	Loss: 0.030489
Train Epoch: 1 [55680/60000 (93%)]	Loss: 0.013314
Train Epoch: 1 [56000/60000 (93%)]	Loss: 0.185237
Train Epoch: 1 [56320/60000 (94%)]	Loss: 0.060923
Train Epoch: 1 [56640/60000 (94%)]	Loss: 0.018763
Train Epoch: 1 [56960/60000 (95%)]	Loss: 0.204844
Train Epoch: 1 [57280/60000 (95%)]	Loss: 0.168295
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.090267
Train Epoch: 1 [57920/60000 (97%)]	Loss: 0.014858
Train Epoch: 1 [58240/60000 (97%)]	Loss: 0.000333
Train Epoch: 1 [58560/60000 (98%)]	Loss: 0.031292
Train Epoch: 1 [58880/60000 (98%)]	Loss: 0.000365
Train Epoch: 1 [59200/60000 (99%)]	Loss: 0.011694
Train Epoch: 1 [59520/60000 (99%)]	Loss: 0.000375
Train Epoch: 1 [59840/60000 (100%)]	Loss: 0.004210

Test set: Average loss: 0.0494, Accuracy: 9834/10000 (98%)

C:\Users\Anoushka\Documents\NYU\Fall23\Cloud\hw3>
```

6. In the Docker GUI, we can see all the images and containers of a particular image.

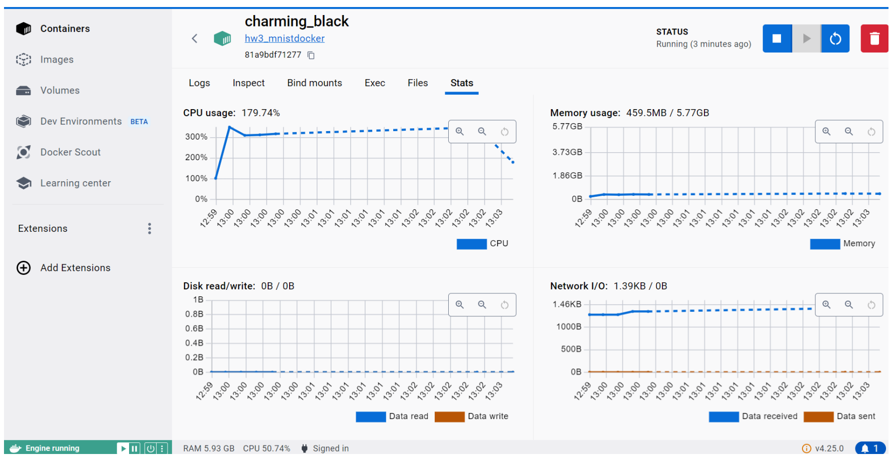


All the Local Images

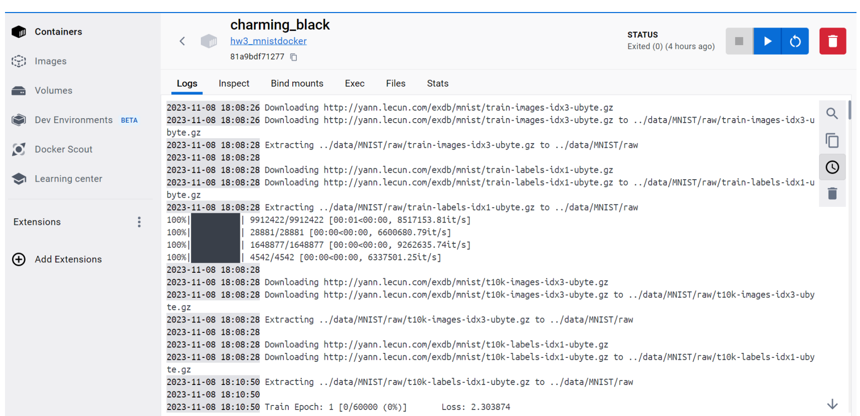


__All the container of the hw3_mnistdocker image__

7. In the Docker GUI, we can see the logs generated and also the stats of the container.



__Stats of the container__

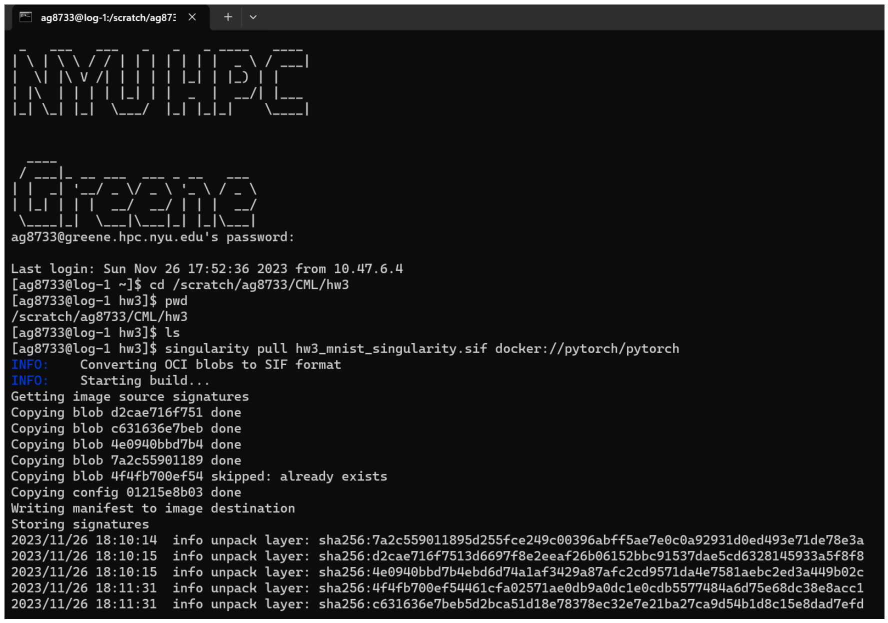


__Logs of the container__

4. **SINGULARITY**

For running the MNIST script on singularity I made use of the Singularity on NYU HPC.

1. Initially log in to the Greene cluster and pull the pytorch docker image.
   (This same thing was done via the FROM command in the dockerfile when i used Docker on my laptop)
   Commands:
   ssh ag8733@greene.hpc.nyu.edu
   cd scratch/ag8733/CML/hw3
   Singularity pull hw3_mnist_singularity.sif docker://pytorch/pytorch



2. Run the singularity image and launch your command. To make use of HPC GPU's we can also use SLURM to first allocate resources and then launch singularity container
   Commands Used:
   singularity run hw3_mnist_singularity.sif
   (This launches the singularity container and now inside the container we can execute our mnist.py script)
   Singularity> *python mnist.py --batch-size 32 --test-batch-size 32 --epochs 1 --no-cuda --no-mps*

```
[ag8733@log-1 hw3]$ ls
hw3_mnist_singularity.sif  mnist.py
[ag8733@log-1 hw3]$ srun --pty --gres=gpu:1 --mem=8GB /bin/bash
srun: job 40383820 queued and waiting for resources
srun: job 40383820 has been allocated resources
[ag8733@gv007 hw3]$ singularity run hw3_mnist_singularity.sif
Singularity> python mnist.py --batch-size 32 --test-batch-size 32 --epochs 1  --no-mps
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.303874
Train Epoch: 1 [320/60000 (1%)] Loss: 1.408849
Train Epoch: 1 [640/60000 (1%)] Loss: 1.191048
Train Epoch: 1 [960/60000 (2%)] Loss: 0.592945
Train Epoch: 1 [1280/60000 (2%)]        Loss: 0.399473
Train Epoch: 1 [1600/60000 (3%)]        Loss: 0.301588
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.367664
Train Epoch: 1 [2240/60000 (4%)]        Loss: 0.239496
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.137093
```

```
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.149301
Train Epoch: 1 [2880/60000 (5%)]        Loss: 0.454813
Train Epoch: 1 [3200/60000 (5%)]        Loss: 0.664438
Train Epoch: 1 [3520/60000 (6%)]        Loss: 0.340718
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.101305
Train Epoch: 1 [4160/60000 (7%)]        Loss: 0.330100
Train Epoch: 1 [4480/60000 (7%)]        Loss: 0.699412
Train Epoch: 1 [4800/60000 (8%)]        Loss: 0.164207
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.222386
Train Epoch: 1 [5440/60000 (9%)]        Loss: 0.229746
Train Epoch: 1 [5760/60000 (10%)]       Loss: 0.332391
Train Epoch: 1 [6080/60000 (10%)]       Loss: 0.545888
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.170668
Train Epoch: 1 [6720/60000 (11%)]       Loss: 0.113199
Train Epoch: 1 [7040/60000 (12%)]       Loss: 0.274721
Train Epoch: 1 [7360/60000 (12%)]       Loss: 0.146807
Train Epoch: 1 [7680/60000 (13%)]       Loss: 0.125959
Train Epoch: 1 [8000/60000 (13%)]       Loss: 0.249238
Train Epoch: 1 [8320/60000 (14%)]       Loss: 0.192714
Train Epoch: 1 [8640/60000 (14%)]       Loss: 0.352342
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.263809
Train Epoch: 1 [9280/60000 (15%)]       Loss: 0.217242
Train Epoch: 1 [9600/60000 (16%)]       Loss: 0.137952
Train Epoch: 1 [9920/60000 (17%)]       Loss: 0.087238
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.266473
Train Epoch: 1 [10560/60000 (18%)]      Loss: 0.077783
Train Epoch: 1 [10880/60000 (18%)]      Loss: 0.322080
Train Epoch: 1 [11200/60000 (19%)]      Loss: 0.309636
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.287141
Train Epoch: 1 [11840/60000 (20%)]      Loss: 0.087346
Train Epoch: 1 [12160/60000 (20%)]      Loss: 0.263650
Train Epoch: 1 [12480/60000 (21%)]      Loss: 0.446106
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.139259
Train Epoch: 1 [13120/60000 (22%)]      Loss: 0.485825
Train Epoch: 1 [13440/60000 (22%)]      Loss: 0.165730
Train Epoch: 1 [13760/60000 (23%)]      Loss: 0.135129
```

```
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.031968
Train Epoch: 1 [50240/60000 (84%)]      Loss: 0.200331
Train Epoch: 1 [50560/60000 (84%)]      Loss: 0.142326
Train Epoch: 1 [50880/60000 (85%)]      Loss: 0.148587
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.042457
Train Epoch: 1 [51520/60000 (86%)]      Loss: 0.260046
Train Epoch: 1 [51840/60000 (86%)]      Loss: 0.009614
Train Epoch: 1 [52160/60000 (87%)]      Loss: 0.033343
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.004479
Train Epoch: 1 [52800/60000 (88%)]      Loss: 0.188543
Train Epoch: 1 [53120/60000 (89%)]      Loss: 0.035366
Train Epoch: 1 [53440/60000 (89%)]      Loss: 0.125526
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.027893
Train Epoch: 1 [54080/60000 (90%)]      Loss: 0.101083
Train Epoch: 1 [54400/60000 (91%)]      Loss: 0.004799
Train Epoch: 1 [54720/60000 (91%)]      Loss: 0.025040
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.095440
Train Epoch: 1 [55360/60000 (92%)]      Loss: 0.034624
Train Epoch: 1 [55680/60000 (93%)]      Loss: 0.016409
Train Epoch: 1 [56000/60000 (93%)]      Loss: 0.116951
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.059052
Train Epoch: 1 [56640/60000 (94%)]      Loss: 0.043005
Train Epoch: 1 [56960/60000 (95%)]      Loss: 0.138000
Train Epoch: 1 [57280/60000 (95%)]      Loss: 0.131516
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.122737
Train Epoch: 1 [57920/60000 (97%)]      Loss: 0.057099
Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.001341
Train Epoch: 1 [58560/60000 (98%)]      Loss: 0.003073
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.000158
Train Epoch: 1 [59200/60000 (99%)]      Loss: 0.006803
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.000172
Train Epoch: 1 [59840/60000 (100%)]     Loss: 0.006759

Test set: Average loss: 0.0495, Accuracy: 9836/10000 (98%)
```

5. OBSERVATIONS & DISCUSSIONS

Singularity was built to run complex applications on HPC clusters in a simple, portable, and reproducible way focusing on security , while Docker is a platform which was designed to package and run an application in a containerized form for more general use. Singularity emerged due to problems faced by researchers and scientists.

The installation of Docker for Windows was very straightforward where I just had to install one application. To run singularity on my local machine I first had to install a hypervisor like virtualBox followed by Vagrant or multipass. While trying to install Singularity I faced a lot of errors and the community which could help out was much smaller compared to more resources for troubleshooting with Docker. Because of these reasons, I decided to use Singularity directly on the NYU HPC clusters. Using Singularity on HPC was faster since I could leverage the HPC GPU. The training and inference time was faster.

Some key differences between these two are:

1. Security
   Docker typically requires root (administrative) privileges to run containers.Singularity allows users to run containers without requiring root access, and thus they can be used on HPC which are multi-user systems. Singularity doesn't provide this superuser elevated access and is thus more secure.

2. Image Formats
   Docker uses its own image format.Singularity is compatible with docker images as well.

3. Community
   Docker has a more widespread community due to the presence of Docker Hub where users can post their images. Singularity is a smaller community focused more in scientific and research fields.
4. Isolation
   Docker aims for strong process isolation within containers, whereas singularity, while still providing isolation, is designed to share the host system's user namespace. In docker very little is shared(namespaces and user space) but this can be changed. For singularity by default the namespaces etc are shared with the host system.

5. Daemon
   Docker launches a daemon to manage the clusters whereas singularity doesn't. Singularity can run directly on the host system.

6. CONCLUSION

Docker and Singularity are both great containerization tools and we should choose based on our use case. For running on HPC or multi-user systems or when security is a big concern using Singularity is advisable. For more general purpose application packaging and containerization, ease of use especially for windows users, community support etc. Docker is a better option. Docker is mainly used for microservices where isolation is the key requirement whereas Singularity focuses on security.

7. REFERENCES

1. Ubuntu Containerization-vs-virtualization (https://ubuntu.com/blog/containerization-vs-virtualization)
2. Virtualization-vs-containerization (https://www.baeldung.com/cs/virtualization-vs-containerization)
3.Docker windows installation (https://docs.docker.com/desktop/install/windows-install)
4.NYU HPC running Singularity Containers (https://sites.google.com/nyu.edu/nyu-hpc/training-support/general-hpc-topics/singularity-run-custom-applications-with-containers)
5.Docker Documentation (https://docs.docker.com/)