MNIST in Docker

1 Introduction

1.1 Background

The MNIST database is a critical benchmark for image recognition models in AI research. Docker, a containerization technology, ensures that these models are deployed consistently and scalable across different environments, which is essential for reproducible and efficient machine learning workflows.

1.2 Purpose

This project, Homework #4 - Docker MNIST, showcases the execution of the MNIST model in a Docker container, using frameworks like PyTorch or TensorFlow. It highlights how Docker facilitates the easy deployment of machine learning models and explores necessary optimizations for enhancing model performance.

1.3 Approach

The methodology includes setting up Docker, modifying example code to improve the model's functionality and performance, and running the MNIST model in Docker. The project also compares Docker with Singularity, analysing their effectiveness in scientific computing through practical implementation and output documentation.

2 Methodology

2.1 Experimental Setup

The Docker and Singularity containers were set up on both a local laptop and an HPC environment to run the MNIST model. The local laptop served to demonstrate the simplicity and efficiency of using Docker, while the HPC environment focused on the adaptability and functionality of Singularity containers for scientific computing.

2.2 Code Configuration

The MNIST Python script (**mnist.py**) was configured to execute within both Docker and Singularity containers. The script included command-line options to adjust model parameters such as batch size and epoch count, aiming to optimize runtime and performance.

2.3 Container Configuration

- **Docker**: A Dockerfile was created specifying the PyTorch base image, the environment setup, and the execution command for the MNIST script.
- **Singularity**: The Docker container setup was adapted to Singularity by converting the Docker image into a Singularity Image File (SIF), which was then used to run the same MNIST model on the HPC system.

2.4 Execution and Monitoring

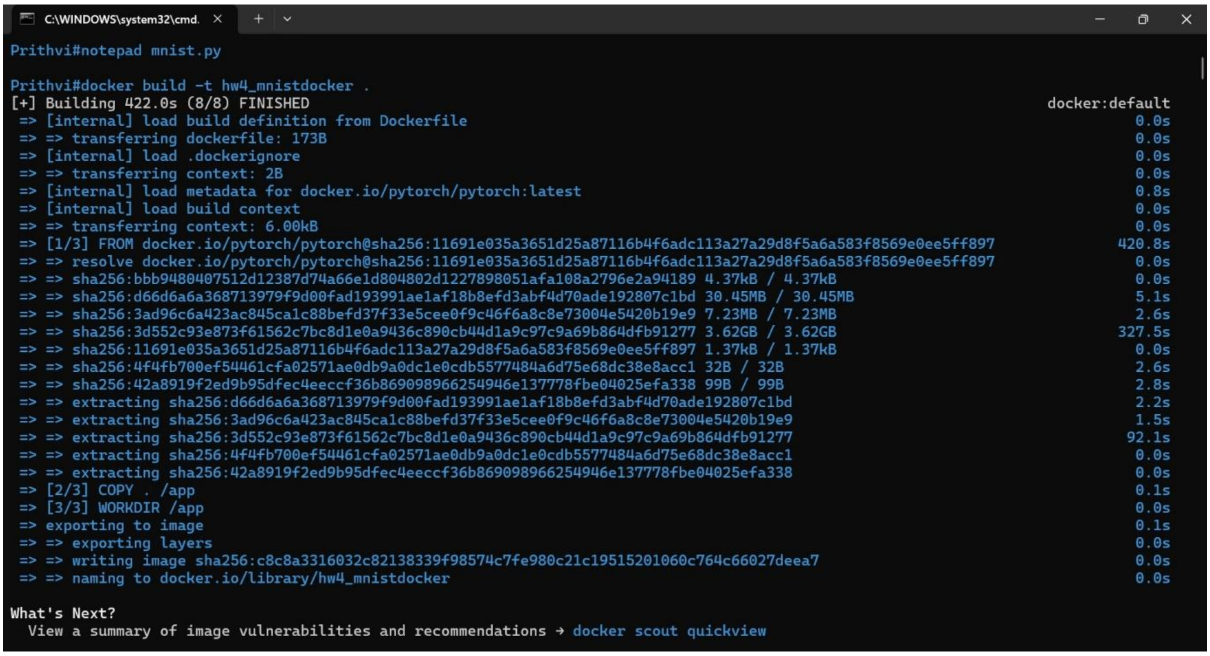
The execution involved building and running the Docker container on the local laptop and converting and running the Singularity container on the HPC. The process was closely monitored, with outputs and system logs captured to analyze the performance and behavior of the MNIST model under each container technology.

3 Experiment Process

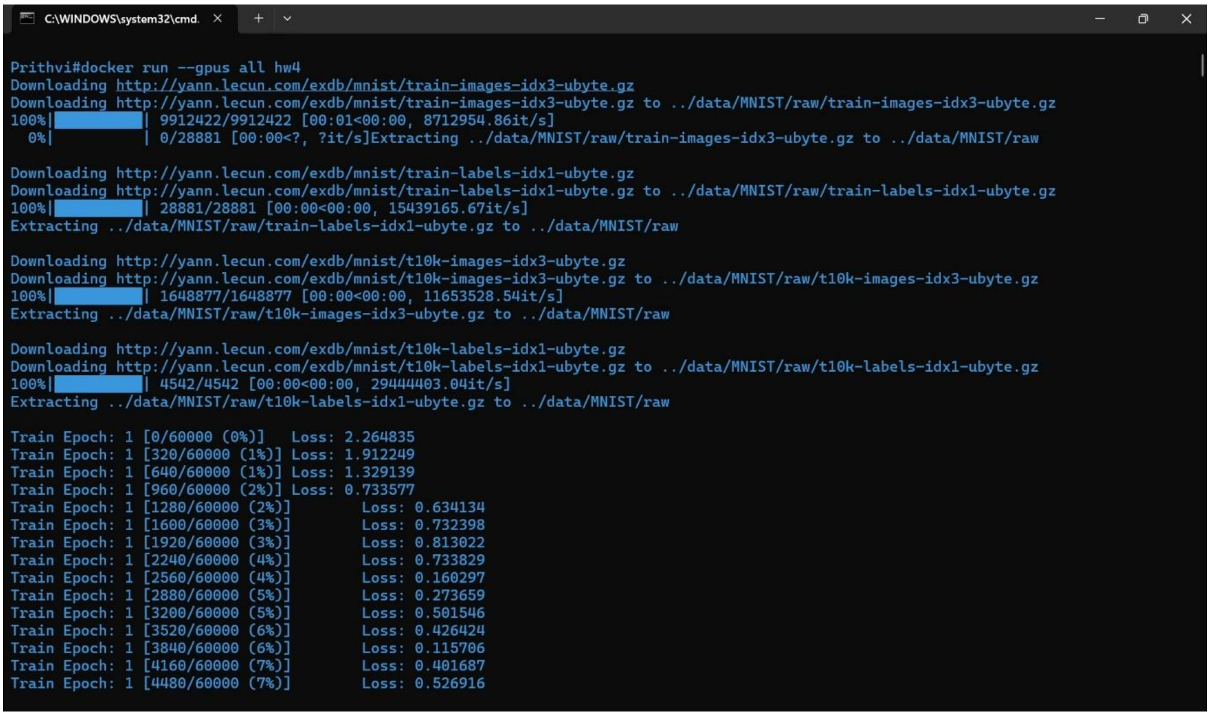
3.1 Steps on Local Laptop

1. **Docker Setup and Execution:**

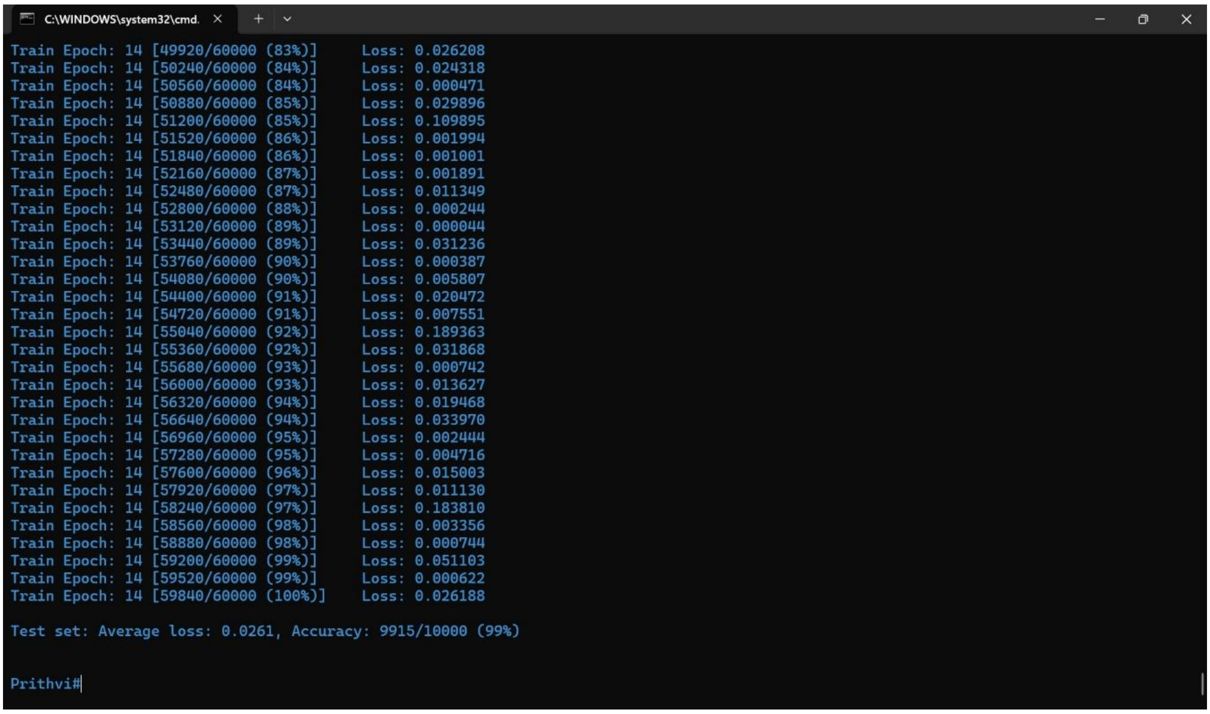
    - Created a **Dockerfile** and placed it in a project directory along with the **mnist.py**.
    - Built the Docker image using the command **docker build -t hw4_mnist ..**
    - Ran the MNIST model inside the Docker container with **docker run hw4_mnist** and captured relevant outputs and logs.

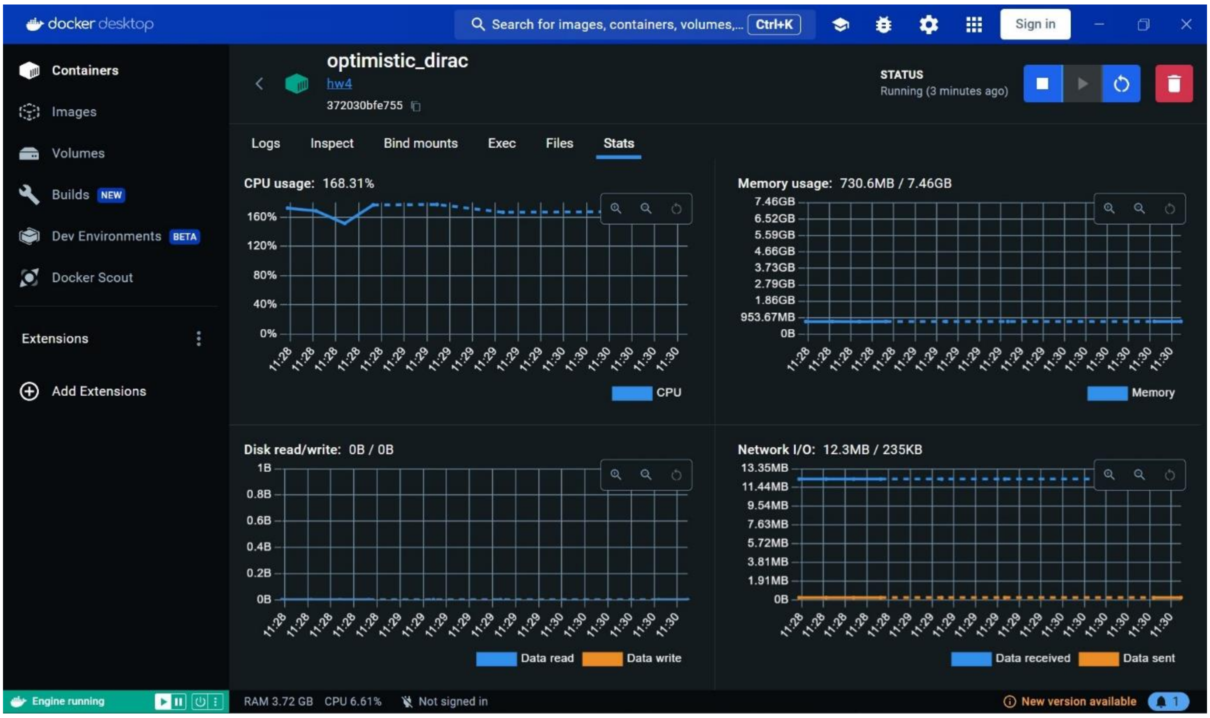


```
C:\WINDOWS\system32\cmd.
Prithvi#notepad mnist.py

Prithvi#docker build -t hw4_mnistdocker .
[+] Building 422.0s (8/8) FINISHED                                                               docker:default
 => [internal] load build definition from Dockerfile                                                       0.0s
 => => transferring dockerfile: 173B                                                                       0.0s
 => [internal] load .dockerignore                                                                          0.0s
 => => transferring context: 2B                                                                            0.0s
 => [internal] load metadata for docker.io/pytorch/pytorch:latest                                          0.8s
 => [internal] load build context                                                                          0.0s
 => => transferring context: 6.00kB                                                                        0.0s
 => [1/3] FROM docker.io/pytorch/pytorch@sha256:11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897   420.8s
 => => resolve docker.io/pytorch/pytorch@sha256:11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897   0.0s
 => => sha256:bbb9480407512d12387d74a66e1d804802d1227898051afa108a2796e2a94189 4.37kB / 4.37kB             0.0s
 => => sha256:d66d6a6a368713979f9d00fad193991ae1af18b8efd3abf4d70ade192807c1bd 30.45MB / 30.45MB           5.1s
 => => sha256:3ad96c6a423ac845ca1c88befd37f33e5cee0f9c46f6a8c8e73004e5420b19e9 7.23MB / 7.23MB             2.6s
 => => sha256:3d552c93e873f61562c7bc8d1e0a9436c890cb44d1a9c97c9a69b864dfb91277 3.62GB / 3.62GB           327.5s
 => => sha256:11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897 1.37kB / 1.37kB             0.0s
 => => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B                   2.6s
 => => sha256:42a8919f2ed9b95dfec4eeccf36b869098966254946e137778fbe04025efa338 99B / 99B                   2.8s
 => => extracting sha256:d66d6a6a368713979f9d00fad193991ae1af18b8efd3abf4d70ade192807c1bd                  2.2s
 => => extracting sha256:3ad96c6a423ac845ca1c88befd37f33e5cee0f9c46f6a8c8e73004e5420b19e9                  1.5s
 => => extracting sha256:3d552c93e873f61562c7bc8d1e0a9436c890cb44d1a9c97c9a69b864dfb91277                 92.1s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                  0.0s
 => => extracting sha256:42a8919f2ed9b95dfec4eeccf36b869098966254946e137778fbe04025efa338                  0.0s
 => [2/3] COPY . /app                                                                                      0.1s
 => [3/3] WORKDIR /app                                                                                     0.0s
 => exporting to image                                                                                     0.1s
 => => exporting layers                                                                                    0.0s
 => => writing image sha256:c8c8a3316032c82138339f98574c7fe980c21c19515201060c764c66027deea7               0.0s
 => => naming to docker.io/library/hw4_mnistdocker                                                         0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```



```
C:\WINDOWS\system32\cmd.

Prithvi#docker run --gpus all hw4
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:01<00:00, 8712954.86it/s]
  0%|          | 0/28881 [00:00<?, ?it/s]Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 15439165.67it/s]
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 11653528.54it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 29444403.04it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw

Train Epoch: 1 [0/60000 (0%)]   Loss: 2.264835
Train Epoch: 1 [320/60000 (1%)] Loss: 1.912249
Train Epoch: 1 [640/60000 (1%)] Loss: 1.329139
Train Epoch: 1 [960/60000 (2%)] Loss: 0.733577
Train Epoch: 1 [1280/60000 (2%)]        Loss: 0.634134
Train Epoch: 1 [1600/60000 (3%)]        Loss: 0.732398
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.813022
Train Epoch: 1 [2240/60000 (4%)]        Loss: 0.733829
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.160297
Train Epoch: 1 [2880/60000 (5%)]        Loss: 0.273659
Train Epoch: 1 [3200/60000 (5%)]        Loss: 0.501546
Train Epoch: 1 [3520/60000 (6%)]        Loss: 0.426424
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.115706
Train Epoch: 1 [4160/60000 (7%)]        Loss: 0.401687
Train Epoch: 1 [4480/60000 (7%)]        Loss: 0.526916
```

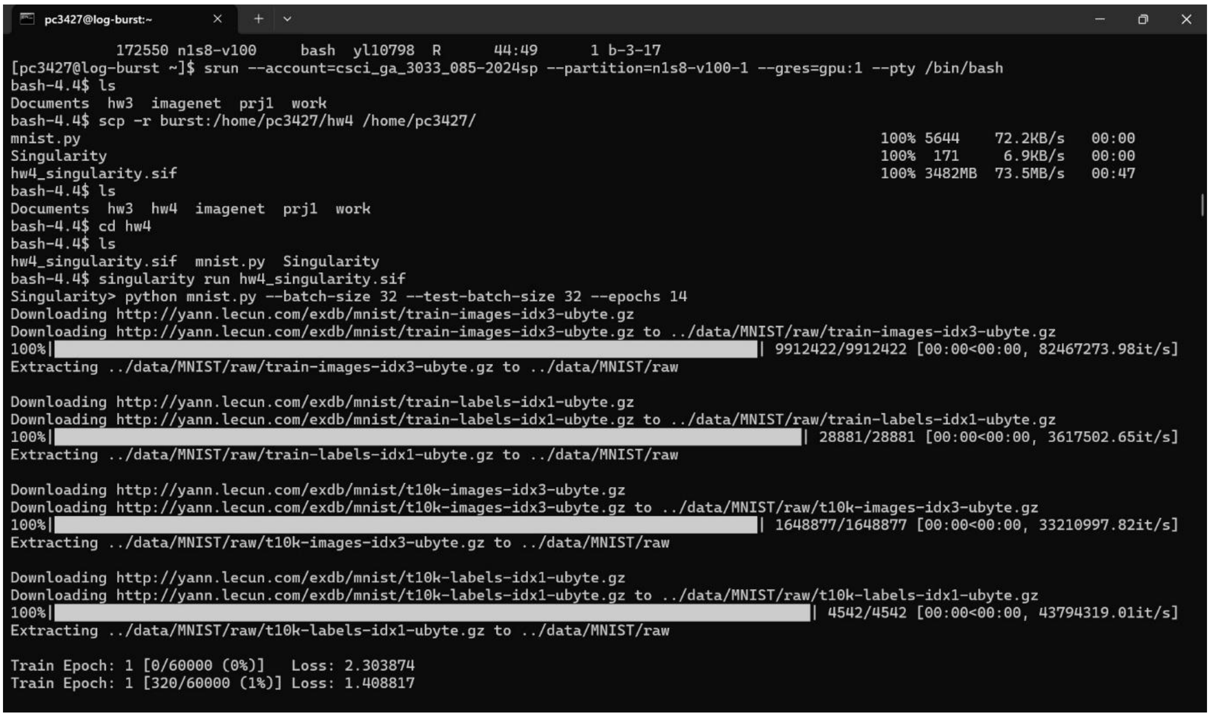




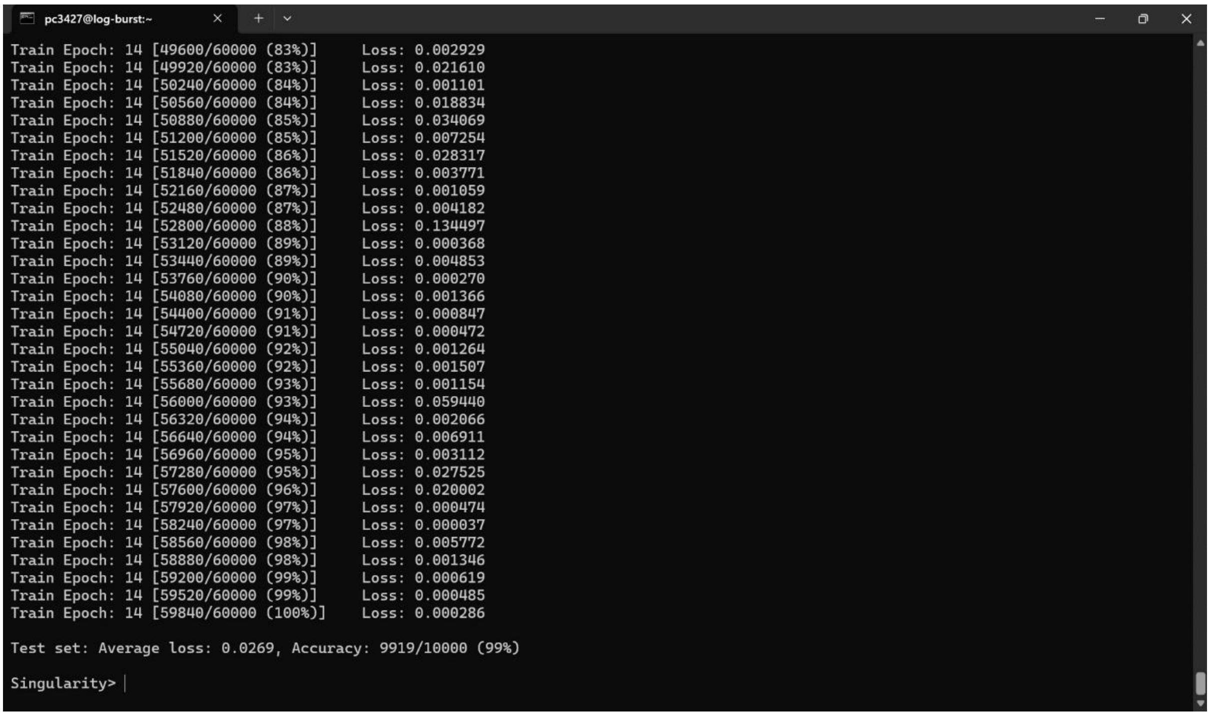
3.2 Steps in HPC Environment

1. **Singularity Setup and Execution:**

   - Used the same Dockerfile to build a Docker image suitable for HPC.
   - Converted the Docker image to a Singularity container using singularity **pull hw4_singularity.sif docker://pytorch/pytorch.**
   - Ran the MNIST model inside the Singularity container using the command **singularity run hw4_singularity.sif python mnist.py --batch-size 32 --test-batch-size 32 --epochs 14** and captured the outputs.



```
pc3427@log-burst:~
          172550 n1s8-v100     bash  yl10798  R      44:49      1 b-3-17
[pc3427@log-burst ~]$ srun --account=csci_ga_3033_085-2024sp --partition=n1s8-v100-1 --gres=gpu:1 --pty /bin/bash
bash-4.4$ ls
Documents  hw3  imagenet  prj1  work
bash-4.4$ scp -r burst:/home/pc3427/hw4 /home/pc3427/
mnist.py                                                                100% 5644     72.2KB/s   00:00
Singularity                                                             100%  171      6.9KB/s   00:00
hw4_singularity.sif                                                     100% 3482MB  73.5MB/s   00:47
bash-4.4$ ls
Documents  hw3  hw4  imagenet  prj1  work
bash-4.4$ cd hw4
bash-4.4$ ls
hw4_singularity.sif  mnist.py  Singularity
bash-4.4$ singularity run hw4_singularity.sif
Singularity> python mnist.py --batch-size 32 --test-batch-size 32 --epochs 14
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
100%|████████████████████████████████████████| 9912422/9912422 [00:00<00:00, 82467273.98it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|████████████████████████████████████████| 28881/28881 [00:00<00:00, 3617502.65it/s]
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|████████████████████████████████████████| 1648877/1648877 [00:00<00:00, 33210997.82it/s]
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|████████████████████████████████████████| 4542/4542 [00:00<00:00, 43794319.01it/s]
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw

Train Epoch: 1 [0/60000 (0%)]   Loss: 2.303874
Train Epoch: 1 [320/60000 (1%)] Loss: 1.408817
```



```
pc3427@log-burst:~
Train Epoch: 14 [49600/60000 (83%)]     Loss: 0.002929
Train Epoch: 14 [49920/60000 (83%)]     Loss: 0.021610
Train Epoch: 14 [50240/60000 (84%)]     Loss: 0.001101
Train Epoch: 14 [50560/60000 (84%)]     Loss: 0.018834
Train Epoch: 14 [50880/60000 (85%)]     Loss: 0.034069
Train Epoch: 14 [51200/60000 (85%)]     Loss: 0.007254
Train Epoch: 14 [51520/60000 (86%)]     Loss: 0.028317
Train Epoch: 14 [51840/60000 (86%)]     Loss: 0.003771
Train Epoch: 14 [52160/60000 (87%)]     Loss: 0.001059
Train Epoch: 14 [52480/60000 (87%)]     Loss: 0.004182
Train Epoch: 14 [52800/60000 (88%)]     Loss: 0.134497
Train Epoch: 14 [53120/60000 (89%)]     Loss: 0.000368
Train Epoch: 14 [53440/60000 (89%)]     Loss: 0.004853
Train Epoch: 14 [53760/60000 (90%)]     Loss: 0.000270
Train Epoch: 14 [54080/60000 (90%)]     Loss: 0.001366
Train Epoch: 14 [54400/60000 (91%)]     Loss: 0.000847
Train Epoch: 14 [54720/60000 (91%)]     Loss: 0.000472
Train Epoch: 14 [55040/60000 (92%)]     Loss: 0.001264
Train Epoch: 14 [55360/60000 (92%)]     Loss: 0.001507
Train Epoch: 14 [55680/60000 (93%)]     Loss: 0.001154
Train Epoch: 14 [56000/60000 (93%)]     Loss: 0.059440
Train Epoch: 14 [56320/60000 (94%)]     Loss: 0.002066
Train Epoch: 14 [56640/60000 (94%)]     Loss: 0.006911
Train Epoch: 14 [56960/60000 (95%)]     Loss: 0.003112
Train Epoch: 14 [57280/60000 (95%)]     Loss: 0.027525
Train Epoch: 14 [57600/60000 (96%)]     Loss: 0.020002
Train Epoch: 14 [57920/60000 (97%)]     Loss: 0.000474
Train Epoch: 14 [58240/60000 (97%)]     Loss: 0.000037
Train Epoch: 14 [58560/60000 (98%)]     Loss: 0.005772
Train Epoch: 14 [58880/60000 (98%)]     Loss: 0.001346
Train Epoch: 14 [59200/60000 (99%)]     Loss: 0.000619
Train Epoch: 14 [59520/60000 (99%)]     Loss: 0.000485
Train Epoch: 14 [59840/60000 (100%)]    Loss: 0.000286

Test set: Average loss: 0.0269, Accuracy: 9919/10000 (99%)

Singularity> |
```

4 Discussion

The MNIST model was executed within Docker and Singularity containers to analyze and compare their functionality and performance in both a local and high-performance computing (HPC) environment.

4.1 General Performance:

Both Docker and Singularity efficiently managed the execution of the MNIST model, but their architectural differences influenced the setup and operational dynamics. Docker provided a straightforward setup process on the local laptop, involving minimal steps to get the container up and running. Conversely, setting up Singularity, on a system, involved more complex steps such as installing additional software like a hypervisor and configuring Vagrant or Multipass.

4.2 Security and User Access:

One significant difference observed was in security protocols and user access management. Docker often requires root privileges to run its containers, which can pose a security risk, especially in multi-user systems like those found in HPC environments. Singularity addresses this by allowing containers to run without root privileges, enhancing security for sensitive computational tasks in shared environments.

4.3 Integration with HPC Resources:

On the HPC, Singularity's integration was more seamless, leveraging existing HPC resources and GPUs more effectively compared to Docker. This integration facilitated faster training and inference times for the MNIST model, showcasing Singularity's optimization for HPC usage.

4.4 Community Support and Resources:

Docker benefits from a broader community support network and extensive resources, including Docker Hub, which provides a vast repository of pre-built images. This contrasts with Singularity's community, which, while smaller, is highly specialized towards scientific and high-performance computing applications.

4.5 Container Isolation and System Integration:

Docker's containerization approach focuses on strong isolation, creating a distinct separation between the container and the host system. This contrasts with Singularity's approach, which by default shares more components with the host, such as namespaces, aiming to simplify integration and resource sharing in scientific computing environments.

4.6 Operational Overhead:

Docker containers are managed via a daemon that runs in the background, adding a layer of complexity in cluster management. Singularity, in contrast, operates directly on the host system without the need for a daemon, simplifying the operational overhead and potentially improving performance.

5 Concepts Learnt

Through this assignment, several fundamental and advanced concepts related to container technologies were explored, providing both technical skills and strategic insights

5.1 Containerization and Management

The core principles of containerization were covered, focusing on how containers leverage the host system's kernel, making them more resource-efficient than traditional virtual machines. Practical skills in creating, managing, and deploying Docker containers, as well as setting up and running Singularity containers, were developed. This included understanding Docker's layered filesystem and Singularity's unique security features like running without root privileges.

5.2 Differences and Use Cases of Container Technologies

Hands-on implementation highlighted the operational and security differences between Docker and Singularity. Learning about Docker's broad application across various environments and Singularity's optimization for scientific computing provided insights into selecting the appropriate technology based on specific project requirements.

5.3 Performance Optimization and Reproducibility

The assignment enhanced understanding of performance benchmarking in containerized environments and the importance of reproducibility in scientific computing. Containers ensure consistent software execution across different environments, which is crucial for collaborative projects and maintaining consistency in deployment.

6 Conclusion

These observations highlight the tailored use cases for each container technology. Docker's generalist, broad-application approach makes it ideal for diverse development environments, while Singularity's specialized, security-focused design aligns well with the needs of scientific research and HPC applications. The choice between Docker and Singularity should be guided by the specific requirements of the deployment environment and the nature of the tasks to be performed.

References

1. **Docker, Inc.** "Docker Documentation." Accessed April 19, 2024. https://docs.docker.com/ .
2. **Sylabs.** "Singularity User Guide." Accessed April 19, 2024. https://sylabs.io/guides/3.0/user-guide/ .
3. **LeCun, Yann, et al.** "The MNIST Database of Handwritten Digits." Accessed April 19, 2024. http://yann.lecun.com/exdb/mnist/ .
4. **PyTorch.** "PyTorch Official Website." Accessed April 19, 2024. https://pytorch.org/.
5. **TensorFlow.** "TensorFlow Official Website." Accessed April 19, 2024. https://www.tensorflow.org/.