MPCS 52040 Distributed Systems
Homework 2
Due: 1/24 @ 10:00 am

Containers and Virtualization

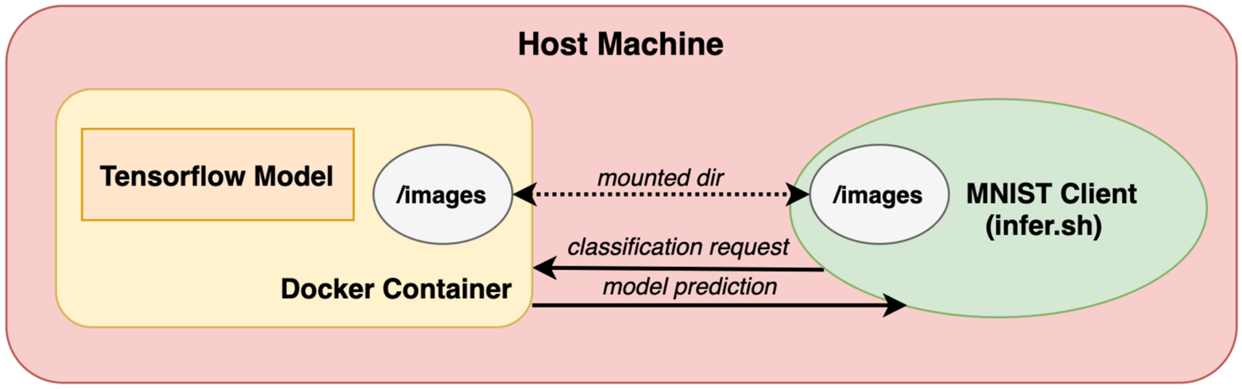
In this assignment, you will learn to work with Docker, which is a popular virtualization technology. You will develop an application which will be run in Docker containers.

The first application is a deep-learning application written in PyTorch, and the second application is a word-counter that applies the map-reduce paradigm. You may find the Docker tutorial helpful: `https://docs.docker.com`

Important: please include your GitHub username in your container names to simplify grading.

Part 1: Machine learning training

In this part, you will deploy a classifier model for the MNIST dataset in a Docker container. The code to train and use the model (using PyTorch) is already provided to you as two Python files (`pt_train.py` and `pt_classify.py`). You are also provided a set of test images (`images.tar.gz`) to test your classifier.



Create a Docker image which contains both Python files. You will need PyTorch installed, for example, by using the latest Python docker image (a py3 version) as a base image and installing PyTorch (alternatively you may use the PyTorch image and install other dependencies). Using Docker, you should run the `pt_train.py` Python file to train an MNIST classifier model and store it in the Docker image. The images directory in the repository should not be used for training. The training script will download it's own training data.

Create a Bash script called `build.sh` which builds the Docker image (`docker build ...`).

Note: training should take less than 20 minutes (and less than 10 minutes on a newish laptop). If you are using an M1, the standard pytorch image will not work. In this case, you should use the base image "FROM

–platform=linux/arm64 python:3.9-slim-bullseye", install python3 via apt and then pip install torch and torchvision. You may see some warnings on inference, but it should still produce the prediction.

Create a second Bash script called `run.sh` that runs the Docker container (`docker run ...`) in a **persistent** way. That is, your container should not die unless you manually kill or stop it from the command line. Note that the `images` directory (containing the test images) should be shared between the host file-system and the Docker container. Once this script is run, there should be a running Docker container containing the trained MNIST model.

Create a third Bash script called `infer.sh` which randomly chooses a test image from the `images` directory (this can be done with the Linux shuf command) and makes the Docker container classify it (`classify.py` and `docker exec ...`). Output the expected and the predicted classification.

Finally, create a README file with the following:

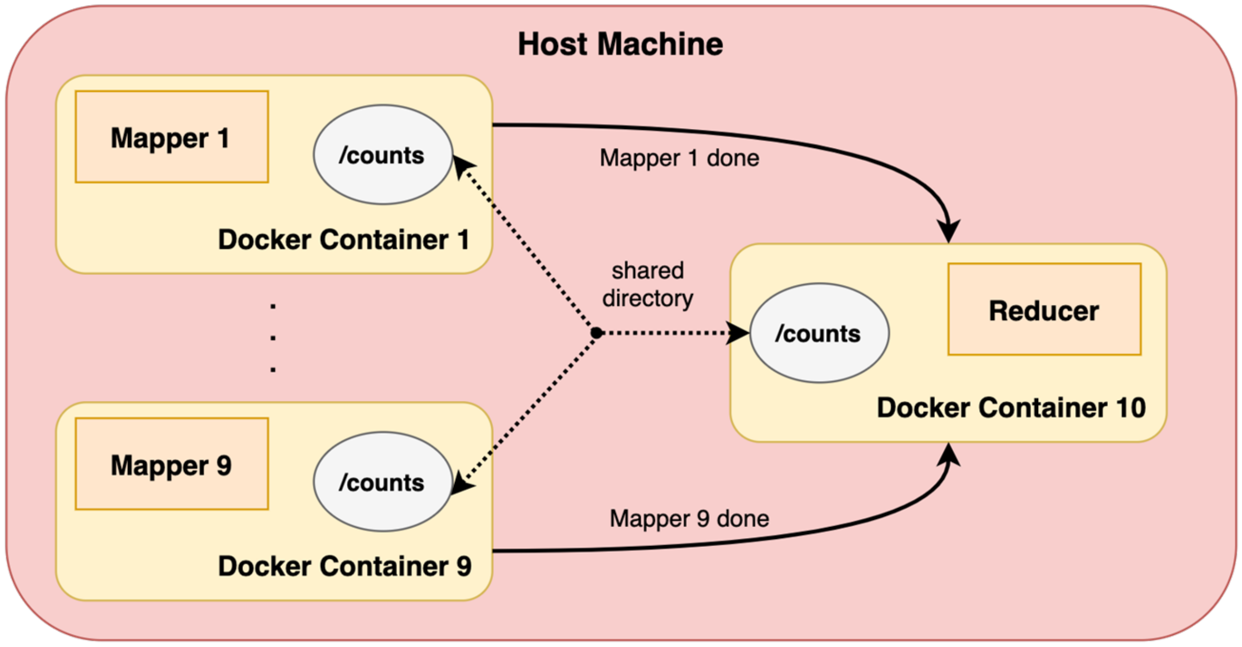
- Write clear instructions with steps and commands to run your application.
- Include any known issues/bugs.

Note: For ease of grading, please DO NOT put absolute paths (e.g., `/home/kyle/images`) in your scripts. Instead, use `.` to refer to the current directory in your `Dockerfile` and `docker build` command. Please also include your GitHub name in your container name.

Submission: Create a directory called part1, and in it, put your `Dockerfile`, `build.sh`, `run.sh`, `infer.sh`, and `README`.

Part 2: Map Reduce

In this part, you will write a map-reduce computation to count the different words in a set of documents. This computation will be run in 10 concurrent Docker containers (9 "mappers" and 1 "reducer"). We have provided a small dataset of Stack Overflow question titles, conveniently divided across 9 text files in `titles.tar.gz`.



Write 2 Python files `map.py` and `reduce.py`. These are described next. The `map.py` should take in 1 command-line argument, `i`, and do the following:

- Read the file `titles/<i>.txt`.
- Find counts for all different words in this file (ignore casing,i.e., convert all to lowercase).
- Store these counts in an output file in `counts/<i>.json` (the JSON format is convenient, but you are welcome to use any other format as well).

The reduce.py should do the following:

- Wait until all 9 mappers are done with their computation.
- Read `counts/1.json`, . . . , `counts/9.json` and combine the wordcounts.
- Store these combined counts in `counts/total_counts.json` ordered by total count.

Note that each mapper will read from a different file and write to a different file. What is the simplest way for the mappers to tell the reducer that they are done?

Now, create a Docker image with `map.py`, `reduce.py`, and the provided `titles` directory. We will use the same Docker image for 10 different containers (9 mappers and 1 reducer). Thus, you will need a way of sharing the counts directory across different containers.

Create a Bash script called `build.sh` that builds this Docker image. Next, create another Bash script called `count.sh` that does the following:

- Create 9 Docker containers (using the above image) and run `map.py` in each, with a different argument each time (from `python map.py 1` to `python map.py 9`).
- Create 1 Docker container (using the above image) and run `reduce.py` in it
- Once the reducer is done, copy the `total_counts.json` file produced to the host operating system

As a correctness check, you should find that "linux" is the 16th most common word in these question titles.

Finally, create a README file that describes

- Write clear instructions with steps and commands to run your application.
- Include any known issues/bugs.

Submission: Create a directory called `part2`, and in it, put your `Dockerfile`, `map.py`, `reduce.py`, `build.sh`, `count.sh`, `total_counts.json`, and `README`.

Again, please include your GitHub username in your container name.

Submission Instructions

Please submit via GitHub classroom: `https://classroom.github.com/a/hX4bWd3-`

We have already seeded your repositories with part1 and part2 directories and starter code. Let us know if you run into any issues.

You can extract the images folder from `images.tar.gz` by running `tar xvf images.tar.gz` (similarly with `titles.tar.gz`).

Grading Guide

This HW will be graded out of 80 points, divided as follows. All points are for functionality/correctness.

Part 1 (40 points total):

- Dockerfile correctly defines image with files and dependencies as specified (10 points)
- `build.sh` builds Docker image correctly (5 points)

- `run.sh` runs Docker container correctly and persistently with a shared mount (15 points)
- `infer.sh` runs inference correctly by communicating with Docker container (10 points)

Part 2 (40 points total)

- Dockerfile correctly defines image with files and dependencies as specified (5 points)
- `map.py` runs as specified, including counting words, I/O, etc. (5 points)
- `reduce.py` runs as specified, including combining word counts, I/O, etc. (5 points)
- Reducer correctly waits for mappers to finish (5 points)
- `build.sh` builds Docker image correctly (5 points)
- `count.sh` runs all 10 containers correctly (and concurrently) producing correct `total_counts.json` (15 points)