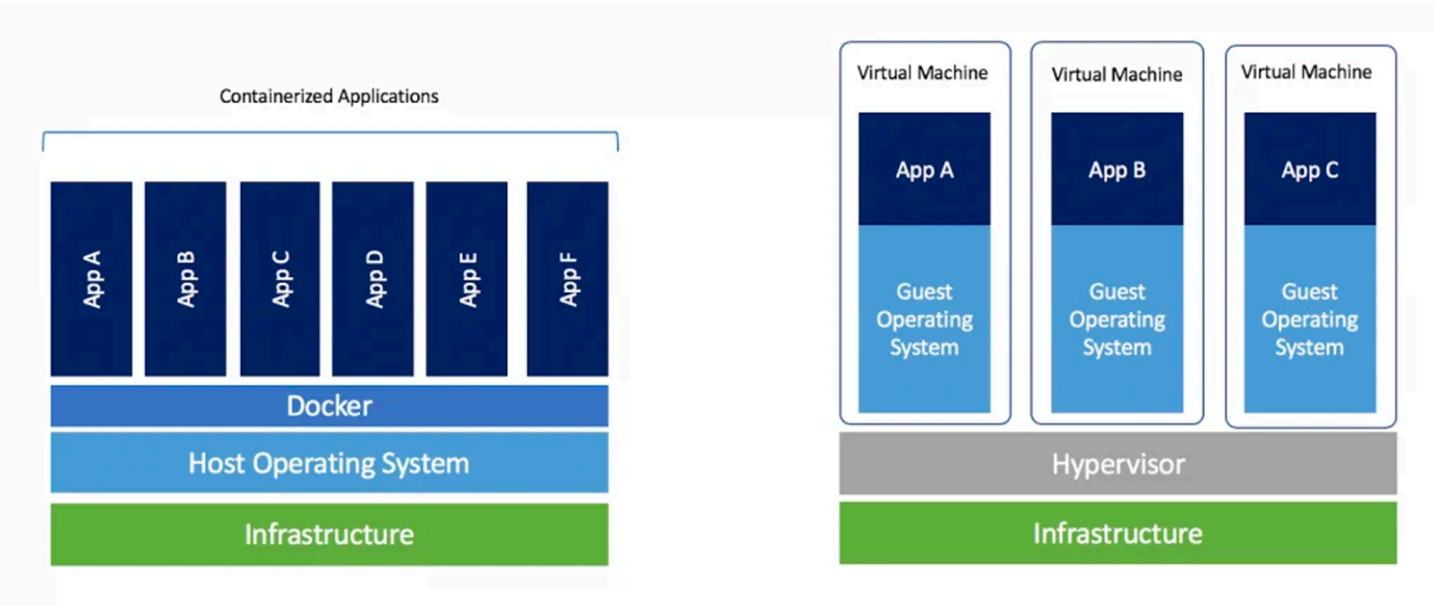
Session 02 - Docker - I

New Attempt

- Due Saturday by 23:59
- Points 500
- Submitting a text entry box

Introduction to Docker and "Containerization"

Containers are lightweight, resource-efficient, and portable. They share a single host operating system (OS) with other containers — sometimes hundreds or even thousands of them. By isolating the software code from the operating environment, developers can build applications on one host — for example, Linux — and deploy it in Windows without worrying about configuration issues during deployment.



- Docker packages, provisions and runs containers. Container technology is available through the operating system: A container packages the application service or function with all of the libraries, configuration files, dependencies and other necessary parts and parameters to operate.

- Each container shares the services of one underlying operating system. Docker images contain all the dependencies needed to execute code inside a container, so containers that move between Docker environments with the same OS work with no changes.
- Docker uses resource isolation in the OS kernel to run multiple containers on the same OS. This is different than virtual machines (VMs), which encapsulate an entire OS with executable code on top of an abstracted layer of physical hardware resources.

> Docker is your best friend for reproducibility

Moby

Moby has three distinct functional offerings:

- **A library of backend components** that implement common container features such as image building, storage management, and log collection.
- **A framework with supporting tooling** that helps you combine, build, and test assemblies of components within your own system. The toolchain produces executable artifacts for all modern architectures, operating systems, and cloud environments.
- **Examples of the framework's uses, including a reference assembly.** This reference asser is the open-source core which the Docker product is built on. You can use it to better understand how Moby components are pulled together into a cohesive system.

you can find moby here: https://github.com/moby/moby (https://github.com/moby/moby)

Docker Installation

For macOS and Windows: https://docs.docker.com/engine/install/ (https://docs.docker.com/engine/install/)

For Linux: https://docs.docker.com/engine/install/ubuntu/ (https://docs.docker.com/engine/install/ubuntu/)

You can use WSL (Windows Subsystem for Linux) (Recommended if you are on Windows) !

https://docs.docker.com/desktop/windows/wsl/ (https://docs.docker.com/desktop/windows/wsl/)

Fix Permissions on Linux

`sudo usermod -aG docker $USER`

Hello World

`docker run hello-world`

You can follow along with me right now using Play With Docker: https://labs.play-with-docker.com/ (https://labs.play-with-docker.com/)

Ubuntu Container

`docker run -it ubuntu bash`

Here `-it` means interactive

Docker from scratch

What exactly are containers?

https://github.com/satyajitghana/containers-from-scratch (https://github.com/satyajitghana/containers-from-scratch)

Docker Architecture

- Docker uses a client-server architecture. The Docker client **talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon can **run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

Container

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

Sample WebApp

- You can follow along by using a GitPod instance
- Or use Play With Docker https://labs.play-with-docker.com/ (https://labs.play-with-docker.com/)

```
git clone https://github.com/satyajitghana/catgif-docker
cd catgif-docker
```

Build the Image

```
docker build --tag catgif:latest .
```

```
[+] Building 16.5s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
0.8s
 => => transferring dockerfile: 457B
0.0s
 => [internal] load .dockerignore
0.6s
 => => transferring context: 2B
0.2s
 => [internal] load metadata for docker.io/library/python:3.10.5-alpine
4.3s
 => [auth] library/python:pull token for registry-1.docker.io
0.0s
 => [1/5] FROM docker.io/library/python:3.10.5-alpine@sha256:a746f64081fca7d6368935750ffcbf04d447
cb0131408c60cbf1a4392981890a          4.9s
 => => resolve docker.io/library/python:3.10.5-alpine@sha256:a746f64081fca7d6368935750ffcbf04d447
cb0131408c60cbf1a4392981890a          0.2s
 => => sha256:70ee0541a51dfb65c8b014de3e891b5ed3ffeebde23db79169e7ba4490af6ed9 1.37kB / 1.37kB
0.0s
 => => sha256:1acd11d41336795110922c6e543fa5dfb8027f21ed65e8b07079c444fb5a03ba 7.04kB / 7.04kB
0.0s
 => => sha256:a746f64081fca7d6368935750ffcbf04d447cb0131408c60cbf1a4392981890a 1.65kB / 1.65kB
0.0s
 => => sha256:530afca65e2ea04227630ae746e0c85b2bd1a179379cbf2b6501b49c4cab2ccc 2.80MB / 2.80MB
0.4s
 => => sha256:cc8c14b1a767335de44f2bc926cb52487979a0fe602ac5429643dbb238f297fb 666.77kB / 666.77k
B                                     0.9s
 => => sha256:bd99fa58365b603cbeb71c93c19182410b640606f36158c10d7ee09d63c1a4f6 12.18MB / 12.18MB
0.7s
 => => extracting sha256:530afca65e2ea04227630ae746e0c85b2bd1a179379cbf2b6501b49c4cab2ccc
1.3s
 => => sha256:777a82aef5431a7852c25deef1cc9e479a3be2c2a9f49e41306b9da78184f1ef 231B / 231B
1.2s
 => => sha256:0c721bc97b97d093d6cf6bb93fc30b60966109ed45ca0d7b5254dffd433c3d89 2.88MB / 2.88MB
1.4s
 => => extracting sha256:cc8c14b1a767335de44f2bc926cb52487979a0fe602ac5429643dbb238f297fb
0.1s
 => => extracting sha256:bd99fa58365b603cbeb71c93c19182410b640606f36158c10d7ee09d63c1a4f6
0.3s
 => => extracting sha256:777a82aef5431a7852c25deef1cc9e479a3be2c2a9f49e41306b9da78184f1ef
0.0s
 => => extracting sha256:0c721bc97b97d093d6cf6bb93fc30b60966109ed45ca0d7b5254dffd433c3d89
0.2s
 => [internal] load build context
0.3s
 => => transferring context: 29.60kB
0.1s
 => [2/5] WORKDIR /app
0.2s
 => [3/5] COPY requirements.txt requirements.txt
0.3s
 => [4/5] RUN pip3 install -r requirements.txt
4.4s
 => [5/5] COPY . .
0.3s
 => exporting to image
0.7s
```

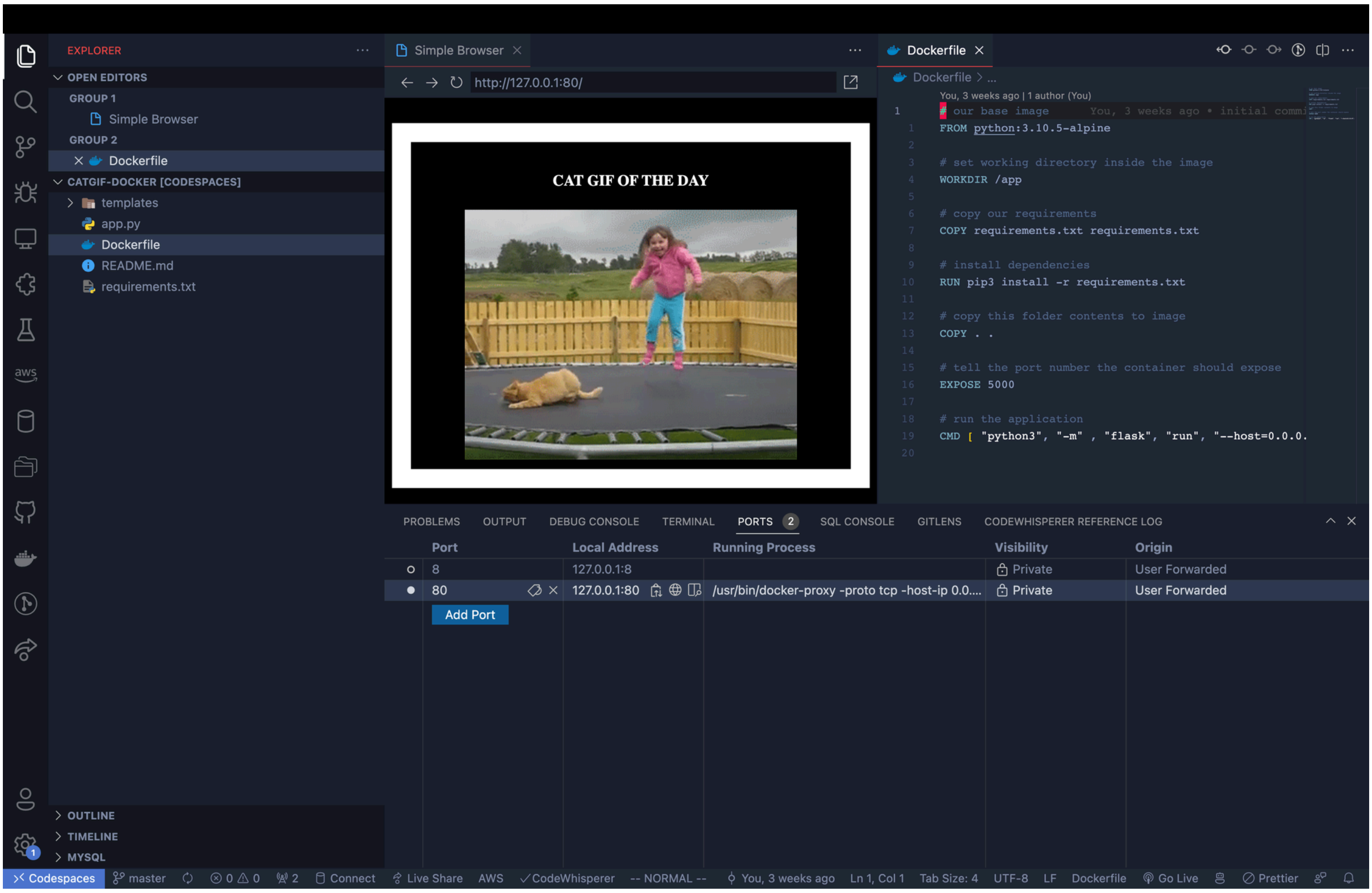
```
 => => exporting layers
0.6s
 => => writing image sha256:dff9a599bb538306ff6155784e22201111fc97d555374aa004b2983cdb3147ea
0.0s
 => => naming to docker.io/library/catgif:latest
```

Run an instance of the image (container)

```
docker run -p 80:5000 catgif:latest
```

```
* Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production
WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on <http://127.0.0.1:5000>
 * Running on <http://172.17.0.2:5000>
```

Seems like something is running on port `5000`, lets open it !



List all containers (including exited)

```
docker ps -a
```

```
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
PORTS     NAMES
92696c261ef9   catgif:latest   "python3 -m flask ru…"   3 minutes ago   Exited (0) 9 seconds ago
compassionate_jepsen
```

List images

```
docker images
```

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
catgif       latest    dff9a599bb53   6 minutes ago   59.6MB
```

Start a container

```
docker start compassionate_jepsen
```

View running container

```
docker ps
```

```
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS          PORTS
NAMES
92696c261ef9   catgif:latest   "python3 -m flask ru…"   5 minutes ago   Up 19 seconds   0.0.0.0:8
0->5000/tcp, :::80->5000/tcp   compassionate_jepsen
```

Get details about a container

```
docker inspect compassionate_jepsen
```

`docker exec -it <container> bash` to go inside running container

`netstat -tulnp` inside the running container to view services and its ports

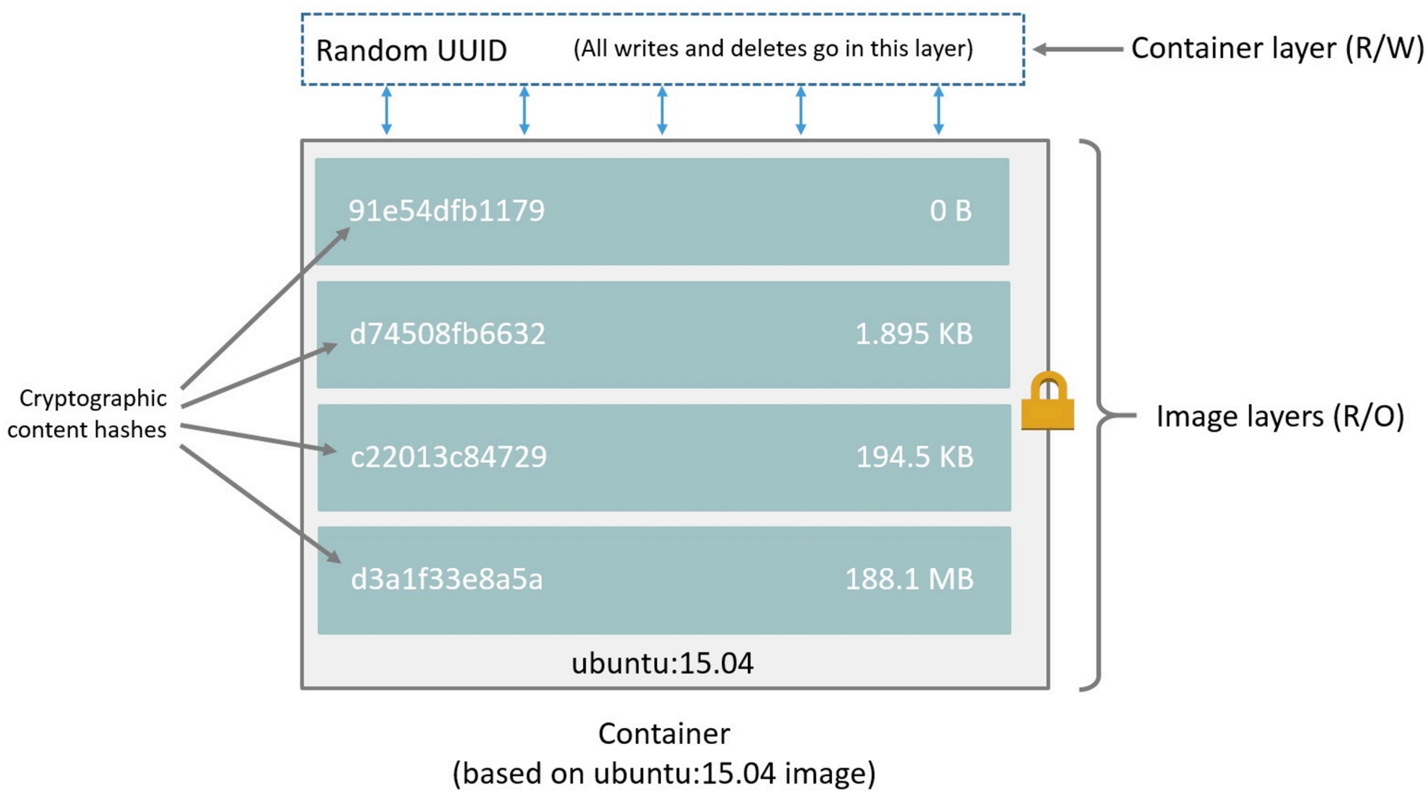
The complete list of commands can be found here:
https://docs.docker.com/engine/reference/commandline/docker/
(https://docs.docker.com/engine/reference/commandline/docker/)

Let’s break down on whats happened

here’s its `Dockerfile`

```
# our base image
FROM python:3.10.5-alpine

# set working directory inside the image
WORKDIR /app

# copy our requirements
COPY requirements.txt requirements.txt

# install dependencies
RUN pip3 install -r requirements.txt

# copy this folder contents to image
COPY . .

# tell the port number the container should expose
EXPOSE 5000
```

Every line in the Dockerfile is a “Layer”



Each layer is an image itself, just one without a human-assigned tag. They have auto-generated IDs though. Each layer stores the changes compared to the image it's based on.

Basically, a layer, or *image layer* is a change on an image, or an intermediate image . Every command you specify (`FROM`, `RUN`, `COPY`, etc.) in your Dockerfile causes the previous image to change, thus creating a new layer. You can think of it as staging changes when you're using git: You add a file's change, then another one, then another one...

There are a few things you need to keep in mind, and they are often misunderstood or used interchangeably without really knowing whats happening

1. ADD vs COPY

COPY only supports the basic copying of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious.

So lets say you want to extract and add a .tar file to the image, you can simply do

```
ADD myfiles.tar.xz /workspace
```

2. CMD vs ENTRYPOINT

When you run docker like this: `docker run -i -t ubuntu bash` the entrypoint is the default `/bin/sh -c`, the image is `ubuntu` and the command is `bash.`

The ENTRYPOINT specifies a command that will always be executed when the container starts.

The CMD specifies arguments that will be fed to the ENTRYPOINT.

3. command in shell form vs exec form

Most shells do not forward process signals to child processes, which means the `SIGINT` generated by pressing `CTRL-C` may not stop a child process

```
FROM ubuntu:latest

# Shell: `bash` doesn't forward CTRL-C SIGINT to `top`
ENTRYPOINT top -b

# Exec: `top` traps CTRL-C SIGINT and stops
ENTRYPOINT ["top", "-b"]
```

it is better to use the exec form, so the interrupts go directly to your process. also generally python programs expect a SIGINT (equivalent to a ctrl-c or KeyboardInterrupt) to gracefully exit.

4. kill vs stop

Dockerfile

```
FROM python:3.7.13-alpine
COPY main.py main.py
CMD ./main.py
```

[main.py (http://main.py)](http://main.py)

```
#!/usr/local/bin/python3 -u
import sys
import signal

import time

def signal_handler(signum, frame):
    print(f"Gracefully shutting down after receiving signal {signum}")
    sys.exit(0)

if __name__ == "__main__":
    signal.signal(signal.SIGTERM, signal_handler)
    signal.signal(signal.SIGINT, signal_handler)
    while True:
        time.sleep(0.5)  # simulate work
        print("Interrupt me")
```

Let's build and run this.

If you do docker kill to this container, it will kill it instantly with an exit status of 137 (128 + 9 and 9 is SIGINT, 128+n are fatal error signals). this Is not considered graceful and is only needed when forcefully a container needs to be terminated. another way is to stop which sends SIGTERM to the program.

But for python we should generally prefer to do a SIGTERM in case of stop, so the below line can be added to the Dockerfile. Now whenever we do a docker stop to this container, it will send a SIGINT instead of a SIGTERM.

```
STOPSIGNAL SIGINT
```

also SIGINT can be sent using kill as well,

```
docker kill --signal=SIGTERM <container_id>
```

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D)
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default)

`Dockerfile` complete syntax guide: [https://docs.docker.com/engine/reference/builder/ (https://docs.docker.com/engine/reference/builder/)](https://docs.docker.com/engine/reference/builder/)

The versions you specify in the Dockerfile are very important, its the whole point of reproducibility and docker !

Let's do this for a pytorch model training process

`Dockerfile`

```
FROM python:3.9-slim

WORKDIR /opt/src

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

COPY . .

ENTRYPOINT ["python3", "main.py"]
```

`requirements.txt`

```

torch==1.12.1
torchvision==0.13.1
```

`main.py`

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.multiprocessing as mp
from torch.utils.data.sampler import Sampler
from torchvision import datasets, transforms

from train import train, test

# Training settings
parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                    help='input batch size for training (default: 64)')
parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                    help='input batch size for testing (default: 1000)')
parser.add_argument('--epochs', type=int, default=10, metavar='N',
                    help='number of epochs to train (default: 10)')
parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
                    help='learning rate (default: 0.01)')
parser.add_argument('--momentum', type=float, default=0.5, metavar='M',
                    help='SGD momentum (default: 0.5)')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
```

```
parser.add_argument('--num-processes', type=int, default=2, metavar='N',
                    help='how many training processes to use (default: 2)')
parser.add_argument('--cuda', action='store_true', default=False,
                    help='enables CUDA training')
parser.add_argument('--dry-run', action='store_true', default=False,
                    help='quickly check a single pass')

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

if __name__ == '__main__':
    args = parser.parse_args()

    use_cuda = args.cuda and torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])
    dataset1 = datasets.MNIST('../data', train=True, download=True,
                       transform=transform)
    dataset2 = datasets.MNIST('../data', train=False,
                       transform=transform)
    kwargs = {'batch_size': args.batch_size,
              'shuffle': True}
    if use_cuda:
        kwargs.update({'num_workers': 1,
                       'pin_memory': True,
                      })

    torch.manual_seed(args.seed)
    mp.set_start_method('spawn')

    model = Net().to(device)
    model.share_memory() # gradients are allocated lazily, so they are not shared here

    processes = []
    for rank in range(args.num_processes):
        p = mp.Process(target=train, args=(rank, args, model, device,
                                           dataset1, kwargs))
        # We first train the model across `num_processes` processes
        p.start()
        processes.append(p)
    for p in processes:
        p.join()

    # Once training is complete, we can test the model
    test(args, model, device, dataset2, kwargs)
```

`train.py`

```
import os
import torch
```

```
import torch.optim as optim
import torch.nn.functional as F

def train(rank, args, model, device, dataset, dataloader_kwargs):
    torch.manual_seed(args.seed + rank)

    train_loader = torch.utils.data.DataLoader(dataset, **dataloader_kwargs)

    optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)
    for epoch in range(1, args.epochs + 1):
        train_epoch(epoch, args, model, device, train_loader, optimizer)

def test(args, model, device, dataset, dataloader_kwargs):
    torch.manual_seed(args.seed)

    test_loader = torch.utils.data.DataLoader(dataset, **dataloader_kwargs)

    test_epoch(model, device, test_loader)

def train_epoch(epoch, args, model, device, data_loader, optimizer):
    model.train()
    pid = os.getpid()
    for batch_idx, (data, target) in enumerate(data_loader):
        optimizer.zero_grad()
        output = model(data.to(device))
        loss = F.nll_loss(output, target.to(device))
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('{}\\tTrain Epoch: {} [{}/{} ({:.0f}%)]\\tLoss: {:.6f}'.format(
                pid, epoch, batch_idx * len(data), len(data_loader.dataset),
                100. * batch_idx / len(data_loader), loss.item()))
            if args.dry_run:
                break

def test_epoch(model, device, data_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in data_loader:
            output = model(data.to(device))
            test_loss += F.nll_loss(output, target.to(device), reduction='sum').item() # sum up b
atch loss
            pred = output.max(1)[1] # get the index of the max log-probability
            correct += pred.eq(target.to(device)).sum().item()

    test_loss /= len(data_loader.dataset)
    print('\\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\\n'.format(
        test_loss, correct, len(data_loader.dataset),
        100. * correct / len(data_loader.dataset)))
```

```
docker build --tag mnist-hogwild .
docker run --rm -it mnist-hogwild
```

Let's look into some PyTorch Docker Images: **https://hub.docker.com/r/pytorch/pytorch/tags (https://hub.docker.com/r/pytorch/pytorch/tags)**, you'll find all sorts of tags with cuda.

Here's a maintainer for PyTorch CPU as well: **https://hub.docker.com/r/zironycho/pytorch/tags (https://hub.docker.com/r/zironycho/pytorch/tags)**

Alpine Linux Images **https://hub.docker.com/_/alpine (https://hub.docker.com/_/alpine)**

Using GPUs with Docker

Follow this guide: [https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html (https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html)](https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html)

`nvidia-docker2`

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)\\&& curl -fsSL <https://nvidia.github.io/libnvidia-container/gpgkey> | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg\\&& curl -s -L <https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list> |\\
            sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' |\\
            sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

```
sudo apt update
sudo apt install -y nvidia-docker2
sudo systemctl restart docker
```

Testing if it installed

```
sudo docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04 nvidia-smi
```

Multi Stage Builds

- [https://pythonspeed.com/articles/multi-stage-docker-python/ (https://pythonspeed.com/articles/multi-stage-docker-python/)](https://pythonspeed.com/articles/multi-stage-docker-python/)
- [https://www.cynnovative.com/simple-multi-stage-docker-builds/ (https://www.cynnovative.com/simple-multi-stage-docker-builds/)](https://www.cynnovative.com/simple-multi-stage-docker-builds/)

Assignment

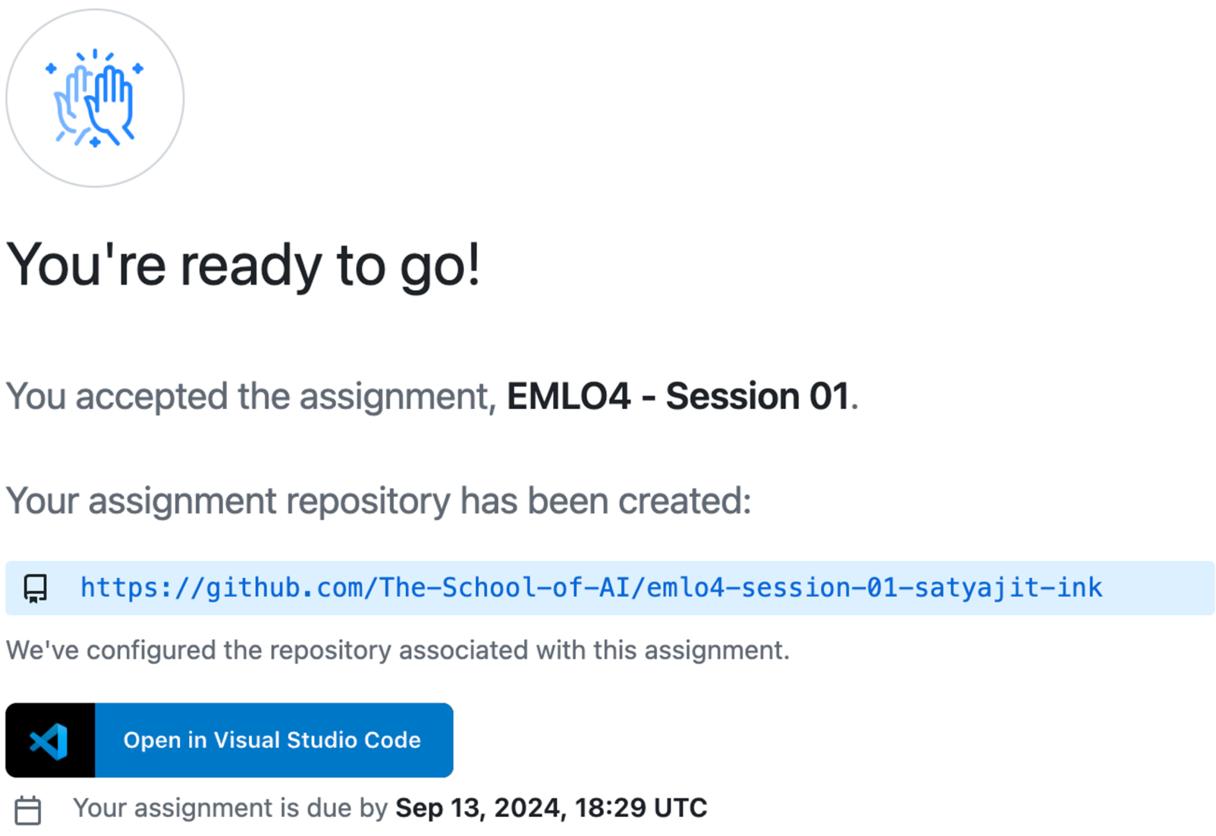
1. Create a `Dockerfile` for pytorch cpu
2. Size of your docker image MUST be < 1.1GB (uncompressed)
3. Train any model on MNIST dataset inside the docker container, call this file `train.py`
4. Save the trained model checkpoint to host os
5. Add option to resume model training from checkpoint in your `train.py`
6. Use black to format your code
7. You can test your code with `bash ./tests/grading.sh`
8. Push the Image to Docker Hub
9. [README.md (http://README.md)](http://README.md) explaining how to run your code
10. Solution to your assignment is in this session

What to Submit on Canvas?

1. Link to your Github Classroom Repository
2. Public Access Link to Docker Hub Repo of the Docker Image

Link to Github Classroom Assignment

https://classroom.github.com/a/A2tcAnZG (https://classroom.github.com/a/A2tcAnZG)



Take a look at tests scripts to see how the assignment is going to be evaluated

Autograding

Summary

Jobs

run-autograding-tests

Run details

Usage

Workflow file

run-autograding-tests

Started 29s ago

Search logs

Test Docker and Training 27s

```
#6 [1/4] FROM docker.io/library/python:3.12-slim@sha256:c24c34b502635f1f7c4e99dc09a2cbd85d480b7dcfd077198c6b5af138906390
#6 resolve docker.io/library/python:3.12-slim@sha256:c24c34b502635f1f7c4e99dc09a2cbd85d480b7dcfd077198c6b5af138906390 done
#6 sha256:eac7a234d33269f362593c31d2ff1db7b116fbd794929f1f6015f5ea812ff254 1.94kB / 1.94kB done
#6 sha256:d78016d8a6986d091bc02622818bccf0d92b07d8f3e153066cb325d19cc79a1f 6.71kB / 6.71kB done
#6 sha256:a2318d6c47ec9cac5acc500c47c79602bcf953cec711a18bc898911a0984365b 0B / 29.13MB 0.1s
#6 sha256:467daa3d4c8b6f3916948d9277922e51161a5d42c977631819a635d2fa9fb32e 0B / 3.51MB 0.1s
#6 sha256:96d51d0de90c323e22cf7900a0ee9185e1e3c185b3ed4fecd9202ae1534e60c7 0B / 11.72MB 0.1s
#6 sha256:c24c34b502635f1f7c4e99dc09a2cbd85d480b7dcfd077198c6b5af138906390 9.12kB / 9.12kB done
#6 sha256:a2318d6c47ec9cac5acc500c47c79602bcf953cec711a18bc898911a0984365b 22.02MB / 29.13MB 0.2s
#6 sha256:a2318d6c47ec9cac5acc500c47c79602bcf953cec711a18bc898911a0984365b 29.13MB / 29.13MB 0.3s done
#6 sha256:467daa3d4c8b6f3916948d9277922e51161a5d42c977631819a635d2fa9fb32e 3.51MB / 3.51MB 0.3s done
#6 sha256:96d51d0de90c323e22cf7900a0ee9185e1e3c185b3ed4fecd9202ae1534e60c7 3.96MB / 11.72MB 0.3s
#6 sha256:e5e6c34f2b20e314d433bf1d444a0c72dc1b27dc93964a4fc102a4dccd25e799 0B / 232B 0.3s
#6 sha256:7ab50b4ae9438decebef449b4afb727284fe0c8f9198b603c588fea80266eb13 0B / 1.75MB 0.3s
#6 extracting sha256:a2318d6c47ec9cac5acc500c47c79602bcf953cec711a18bc898911a0984365b
#6 sha256:96d51d0de90c323e22cf7900a0ee9185e1e3c185b3ed4fecd9202ae1534e60c7 11.72MB / 11.72MB 0.4s done
#6 sha256:e5e6c34f2b20e314d433bf1d444a0c72dc1b27dc93964a4fc102a4dccd25e799 232B / 232B 0.3s done
#6 sha256:7ab50b4ae9438decebef449b4afb727284fe0c8f9198b603c588fea80266eb13 1.75MB / 1.75MB 0.4s done
#6 extracting sha256:a2318d6c47ec9cac5acc500c47c79602bcf953cec711a18bc898911a0984365b 1.2s done
#6 extracting sha256:467daa3d4c8b6f3916948d9277922e51161a5d42c977631819a635d2fa9fb32e 0.1s
#6 extracting sha256:467daa3d4c8b6f3916948d9277922e51161a5d42c977631819a635d2fa9fb32e 0.1s done
#6 extracting sha256:96d51d0de90c323e22cf7900a0ee9185e1e3c185b3ed4fecd9202ae1534e60c7
#6 extracting sha256:96d51d0de90c323e22cf7900a0ee9185e1e3c185b3ed4fecd9202ae1534e60c7 0.6s done
#6 extracting sha256:e5e6c34f2b20e314d433bf1d444a0c72dc1b27dc93964a4fc102a4dccd25e799
#6 extracting sha256:e5e6c34f2b20e314d433bf1d444a0c72dc1b27dc93964a4fc102a4dccd25e799 done
#6 extracting sha256:7ab50b4ae9438decebef449b4afb727284fe0c8f9198b603c588fea80266eb13 0.1s
#6 extracting sha256:7ab50b4ae9438decebef449b4afb727284fe0c8f9198b603c588fea80266eb13 0.1s done
#6 DONE 2.6s
#7 [2/4] RUN pip install     torch==2.3.1+cpu     torchvision==0.18.1+cpu     -f https://download.pytorch.org/whl/torch_stable.html     && rm -rf /root/.cache/pip
#7 1.406 Looking in links: https://download.pytorch.org/whl/torch_stable.html
#7 2.962 Collecting torch==2.3.1+cpu
#7 2.977   Downloading https://download.pytorch.org/whl/cpu/torch-2.3.1%2Bcpu-cp312-cp312-linux_x86_64.whl (190.4 MB)
#7 4.641      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.4/190.4 MB 115.7 MB/s eta 0:00:00
#7 5.131 Collecting torchvision==0.18.1+cpu
#7 5.171   Downloading https://download.pytorch.org/whl/cpu/torchvision-0.18.1%2Bcpu-cp312-cp312-linux_x86_64.whl (1.6 MB)
#7 5.186      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 123.8 MB/s eta 0:00:00
#7 5.403 Collecting filelock (from torch==2.3.1+cpu)
#7 5.465   Downloading filelock-3.15.4-py3-none-any.whl.metadata (2.9 kB)
#7 5.956 Collecting typing-extensions>=4.8.0 (from torch==2.3.1+cpu)
#7 5.967   Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
#7 6.223 Collecting sympy (from torch==2.3.1+cpu)
#7 6.233   Downloading sympy-1.13.2-py3-none-any.whl.metadata (12 kB)
#7 6.459 Collecting networkx (from torch==2.3.1+cpu)
```

Summary

Jobs

run-autograding-tests

Run details

Usage

Workflow file

run-autograding-tests

succeeded now in 4m 6s

Search logs

Test Docker and Training 4m 2s

```
1       Train Epoch: 1 [46080/60000 (76%)]      Loss: 0.077351
1       Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.090368
1       Train Epoch: 1 [56320/60000 (93%)]      Loss: 0.048185
Test set: Average loss: 0.0586, Accuracy: 9813/10000 (98%)
🎉 All checks passed!
```

Autograding Reporter 1s

```
▶ Run classroom-resources/autograding-grading-reporter@v1
🔄 Processing: test-docker-and-training
✅ Test Docker and Training
Test code:
bash ./tests/grading.sh
Total points for test-docker-and-training: 500.00/500
Test runner summary
┌────────────────────┬────────────┬───────────┐
│ Test Runner Name   │ Test Score │ Max Score │
├────────────────────┼────────────┼───────────┤
│ test-docker-and-t… │ 500        │ 500       │
├────────────────────┼────────────┼───────────┤
│ Total:             │ 500        │ 500       │
└────────────────────┴────────────┴───────────┘
🏆 Grand total tests passed: 1/1
Workflow Run Response: https://api.github.com/repos/The-School-of-AI/emlo4-session-01-satyajit-ink/check-suites/28060918278
```

Post Checkout code 0s

```
Post job cleanup.
/usr/bin/git version
git version 2.46.0
Temporarily overriding HOME='/home/runner/work/_temp/e00553c7-0eb4-4518-a6c0-4d25cbe140a9' before making global git config changes
Adding repository directory to the temporary git global config as a safe directory
/usr/bin/git config --global --add safe.directory /home/runner/work/emlo4-session-01-satyajit-ink/emlo4-session-01-satyajit-ink
/usr/bin/git config --local --name-only --get-regexp core\.sshCommand
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all 'core.sshCommand' || :"
/usr/bin/git config --local --name-only --get-regexp http\.https\:\/\/github\.com\/\.extraheader
http.https://github.com/.extraheader
/usr/bin/git config --local --unset-all http.https://github.com/.extraheader
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https\:\/\/github\.com\/\.extraheader' && git config --local --unset-all 'http.https://github.com/.extraheader' || :"
```

Complete job 0s

Recordings

Studio

EMLO4 - Session 02 - Studio