Docker: A Quick Start Guide

What we will cover

1. What is a dockerfile and a compose file
2. How is Gym set up using docker
3. How to access the GPU with Docker
4. How is CoastSeg set up Docker

by Sharon Fitzpatrick

Docker Fundamentals & Benefits

Containerization Concept

Docker packages applications with all dependencies, creating isolated, portable environments that run identically regardless of the host system.

Development Simplification

Eliminates "works on my machine" problems by ensuring consistent environments from development through production.

GPU Access

Using Docker gives us a standardized way of accessing the GPU

Installing Docker on Windows

Download Docker Desktop

Visit docker.com and download the Docker Desktop installer for Windows. The installer includes both Docker Engine and Docker Desktop GUI.

Install WSL2

Windows Subsystem for Linux 2 is required for Docker on Windows. Enable WSL2 through PowerShell with administrator privileges using: **wsl --install**.

Run Installer

Execute the Docker Desktop installer and follow the prompts. Ensure "Use WSL2 instead of Hyper-V" option is selected during installation.

System Reboot

Restart your computer when prompted to complete the installation process and initialize all components properly.

Verifying Your Docker Installation

Launch Docker Desktop

Open Docker Desktop from your Start Menu. This automatically starts the Docker engine in the background.

Run Verification Command

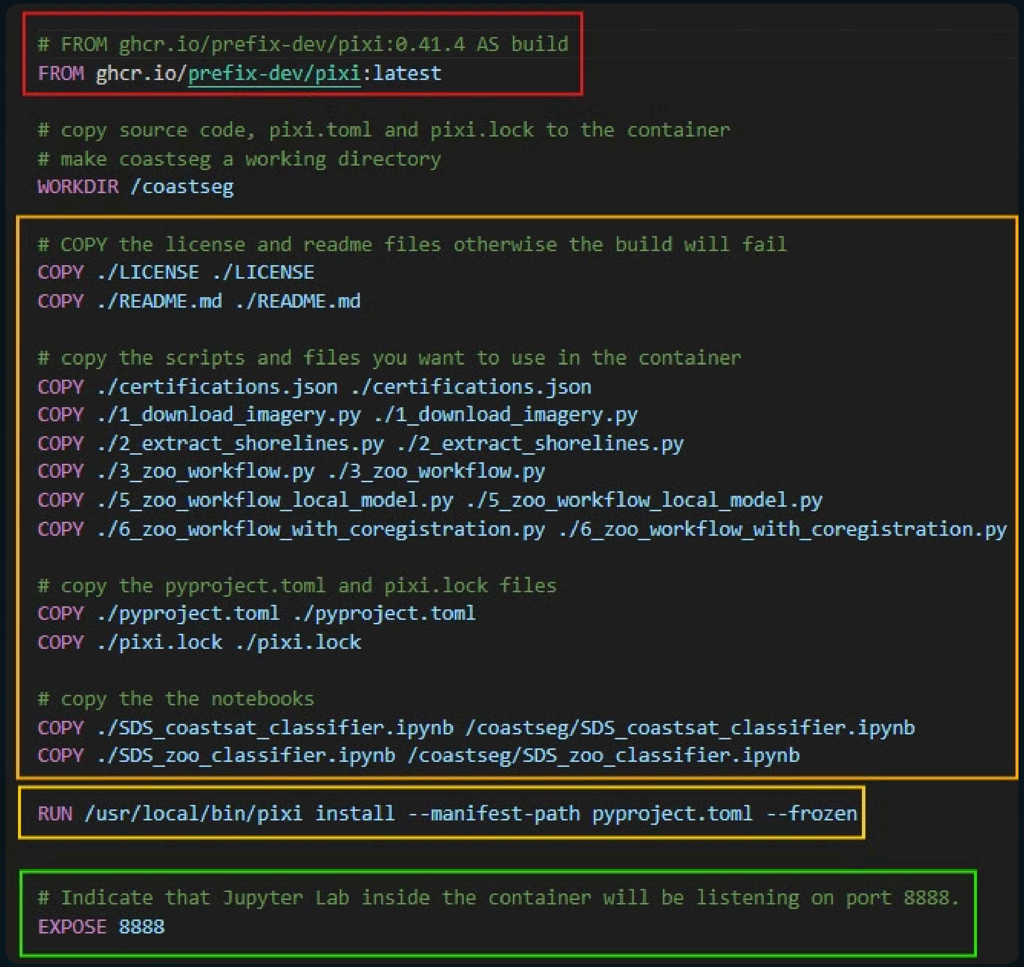
Launch PowerShell

```
docker info
```

What is a Dockerfile?

Instructions to build a container image, defining its base, dependencies, and runtime behavior

- **Base Image & Build Context:** Begins with a base image (using the FROM instruction) that sets the foundation for your container.
- **Layered Instructions:** Uses commands (like RUN, COPY, ADD) to build image layers, which enable caching and incremental builds.
- **Runtime Configuration:** Defines how the container behaves at startup through CMD or ENTRYPOINT, and configures environment variables, working directories, and metadata.
- **Networking & Storage:** Optionally exposes ports and specifies volumes for networking and persistent data.

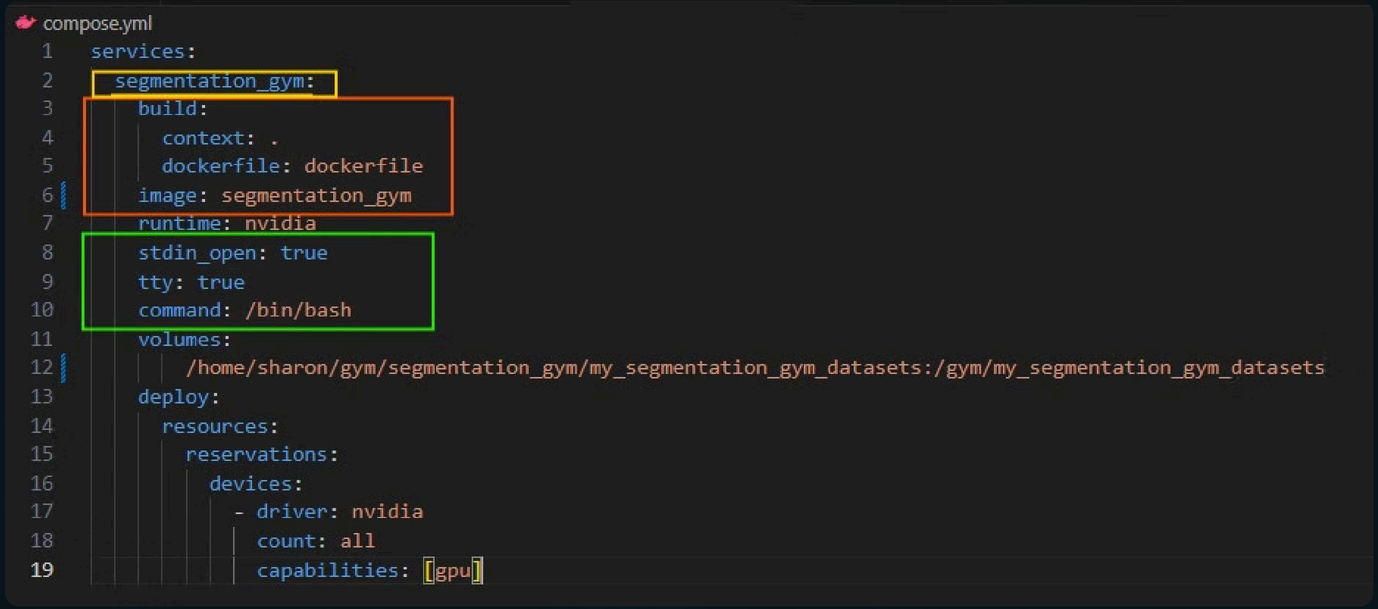


```dockerfile
# FROM ghcr.io/prefix-dev/pixi:0.41.4 AS build
FROM ghcr.io/prefix-dev/pixi:latest

# copy source code, pixi.toml and pixi.lock to the container
# make coastseg a working directory
WORKDIR /coastseg

# COPY the license and readme files otherwise the build will fail
COPY ./LICENSE ./LICENSE
COPY ./README.md ./README.md

# copy the scripts and files you want to use in the container
COPY ./certifications.json ./certifications.json
COPY ./1_download_imagery.py ./1_download_imagery.py
COPY ./2_extract_shorelines.py ./2_extract_shorelines.py
COPY ./3_zoo_workflow.py ./3_zoo_workflow.py
COPY ./5_zoo_workflow_local_model.py ./5_zoo_workflow_local_model.py
COPY ./6_zoo_workflow_with_coregistration.py ./6_zoo_workflow_with_coregistration.py

# copy the pyproject.toml and pixi.lock files
COPY ./pyproject.toml ./pyproject.toml
COPY ./pixi.lock ./pixi.lock

# copy the the notebooks
COPY ./SDS_coastsat_classifier.ipynb /coastseg/SDS_coastsat_classifier.ipynb
COPY ./SDS_zoo_classifier.ipynb /coastseg/SDS_zoo_classifier.ipynb

RUN /usr/local/bin/pixi install --manifest-path pyproject.toml --frozen

# Indicate that Jupyter Lab inside the container will be listening on port 8888.
EXPOSE 8888
```

What is a Docker Compose File?

YAML configuration that sets up the environment, dependencies, and runtime settings for a Dockerized application

- **YAML Configuration:** Defines services, networks, and volumes in a declarative format.
  - **Service:** A defined container configuration that runs a specific application.
  - **Network:** A virtual bridge that connects containers for communication.
- **Simplified Commands:** Build, start, and stop all services with a single command.



compose.yml

```yaml
services:
  segmentation_gym:
    build:
      context: .
      dockerfile: dockerfile
    image: segmentation_gym
    runtime: nvidia
    stdin_open: true
    tty: true
    command: /bin/bash
    volumes:
      - /home/sharon/gym/segmentation_gym/my_segmentation_gym_datasets:/gym/my_segmentation_gym_datasets
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

Name of the service is segmentation gym

How to build service from the image segmentation gym

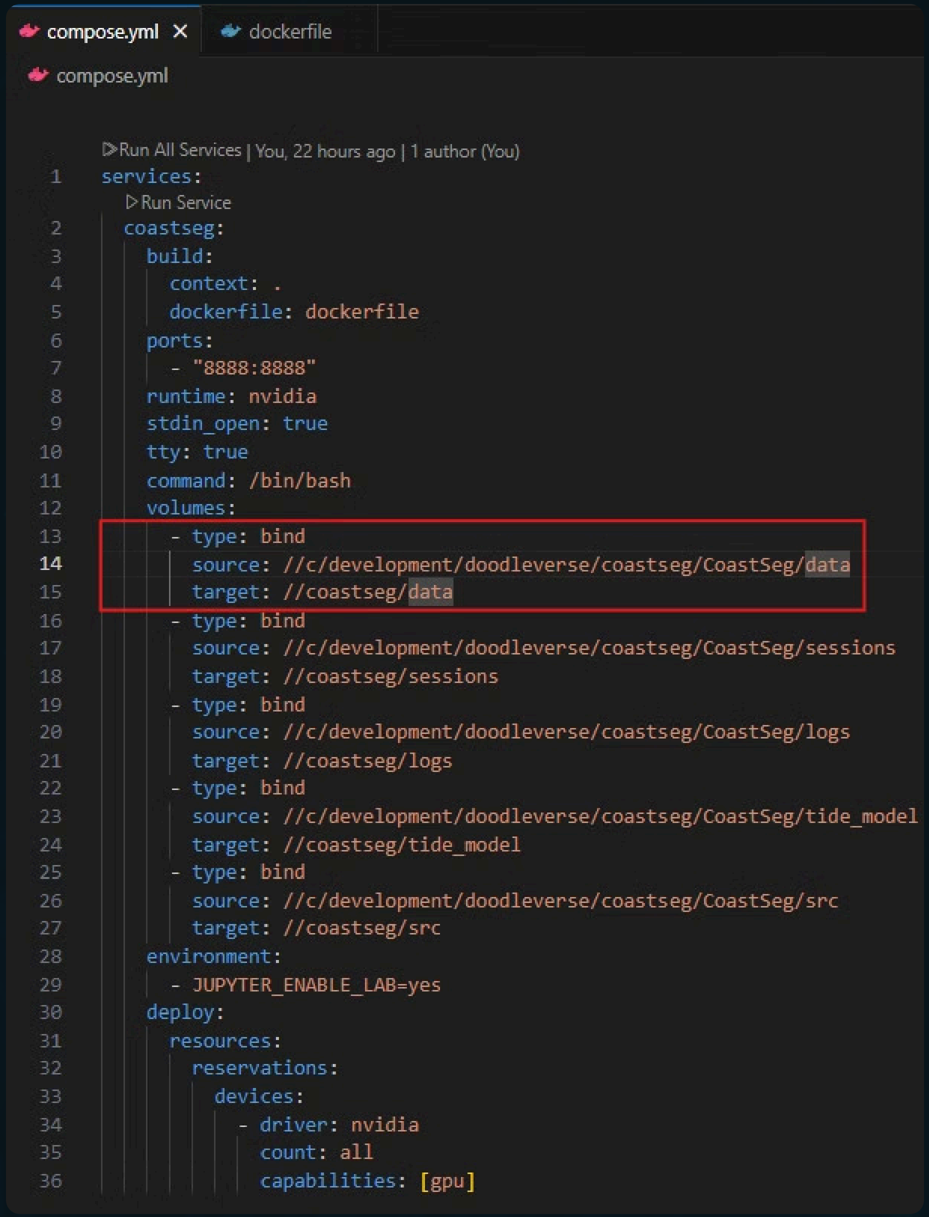
Allow the container to run in an interactive shell

Bind Mounts

⭐ Bind Mounts essentially link your local folder to your container's folder

⭐ Any changes made to a bind mount folder are reflected in the folder locally

- **Direct Host-Container Link:** A bind mount directly maps a file or directory from your host system to a specific location inside the container.
- **Real-Time Synchronization:** Changes made in the bind-mounted directory on the host are immediately reflected in the container, and vice versa.
- **Runtime Configuration:** Bind mounts are set up when the container is started (via Docker Compose or the Docker CLI), not during the image build process.



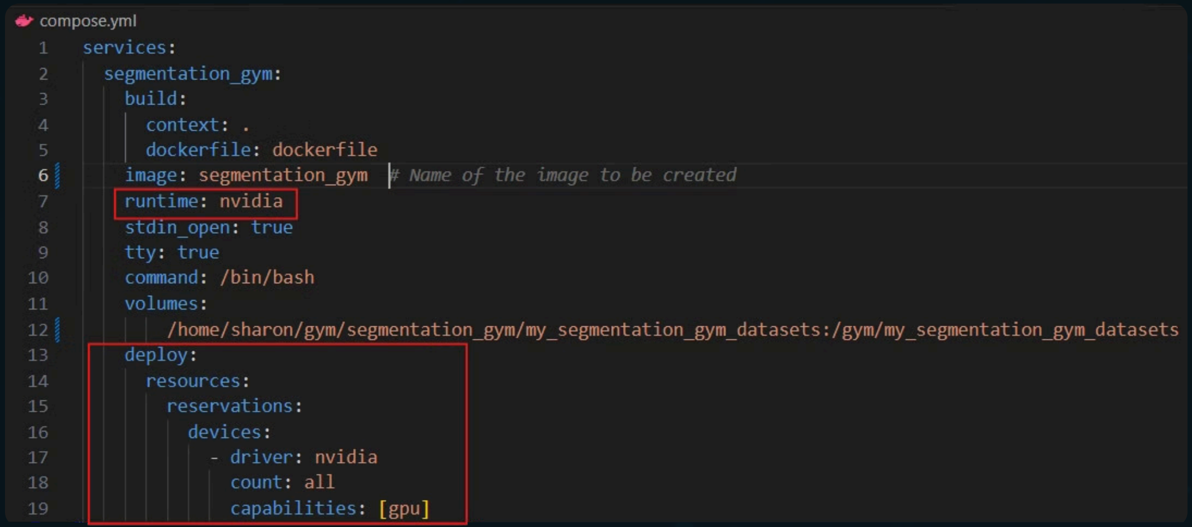
```yaml
services:
  coastseg:
    build:
      context: .
      dockerfile: dockerfile
    ports:
      - "8888:8888"
    runtime: nvidia
    stdin_open: true
    tty: true
    command: /bin/bash
    volumes:
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/data
        target: //coastseg/data
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/sessions
        target: //coastseg/sessions
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/logs
        target: //coastseg/logs
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/tide_model
        target: //coastseg/tide_model
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/src
        target: //coastseg/src
    environment:
      - JUPYTER_ENABLE_LAB=yes
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

- Here my folder "//c/development/doodleverse/coastseg/CoastSeg/data" is linked to my containers "CoastSeg/data"

GPU Access with Compose File

- **Compose Configuration:** Specify GPU requirements in the Compose file using deploy resources (e.g., setting device capabilities).
- **Runtime Option:** Use the appropriate runtime flag (such as `runtime: nvidia` or `--gpus`) to enable GPU support.

⚠️ This will NOT work if your computer does not have NVIDIA drivers and the NVIDIA Container Toolkit installed (QR code for guide)



compose.yml

```yaml
services:
  segmentation_gym:
    build:
      context: .
      dockerfile: dockerfile
    image: segmentation_gym  # Name of the image to be created
    runtime: nvidia
    stdin_open: true
    tty: true
    command: /bin/bash
    volumes:
      /home/sharon/gym/segmentation_gym/my_segmentation_gym_datasets:/gym/my_segmentation_gym_datasets
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

- This allows your computer's nvidia GPU during the containers runtime

Gym Dockerfile

Base Image: Pixi

Use the base docker image provided by Pixi

🚫 DO NOT COPY the .pixi folder

- its massive
- it may cause your environment to solve incorrectly

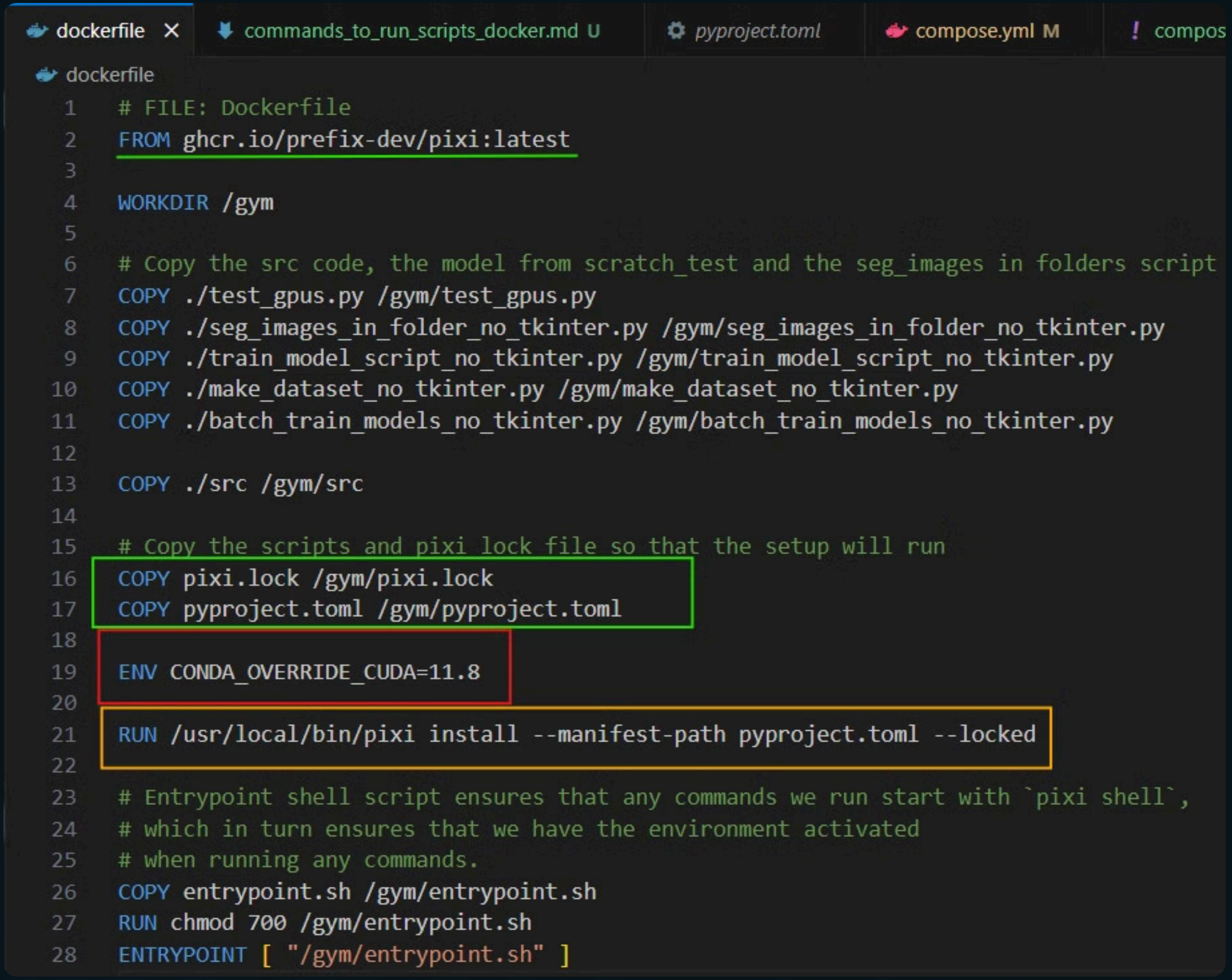
Set the CUDA version

This tells Pixi what version of CUDA will be available at RUNTIME

⚠️ You will get build errors about missing cuda libraries if you don't put this here

Install the Pixi Environment

Install the Pixi environment defined in the pyproject.toml + pixi.lock file we copied over



```
# FILE: Dockerfile
FROM ghcr.io/prefix-dev/pixi:latest

WORKDIR /gym

# Copy the src code, the model from scratch_test and the seg_images in folders script
COPY ./test_gpus.py /gym/test_gpus.py
COPY ./seg_images_in_folder_no_tkinter.py /gym/seg_images_in_folder_no_tkinter.py
COPY ./train_model_script_no_tkinter.py /gym/train_model_script_no_tkinter.py
COPY ./make_dataset_no_tkinter.py /gym/make_dataset_no_tkinter.py
COPY ./batch_train_models_no_tkinter.py /gym/batch_train_models_no_tkinter.py

COPY ./src /gym/src

# Copy the scripts and pixi lock file so that the setup will run
COPY pixi.lock /gym/pixi.lock
COPY pyproject.toml /gym/pyproject.toml

ENV CONDA_OVERRIDE_CUDA=11.8

RUN /usr/local/bin/pixi install --manifest-path pyproject.toml --locked

# Entrypoint shell script ensures that any commands we run start with `pixi shell`,
# which in turn ensures that we have the environment activated
# when running any commands.
COPY entrypoint.sh /gym/entrypoint.sh
RUN chmod 700 /gym/entrypoint.sh
ENTRYPOINT [ "/gym/entrypoint.sh" ]
```

How to Get Your Cuda Version

NVCC --version

```
(base) sharon@Sharonator:~$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Thu_Nov_18_09:45:30_PST_2021
Cuda compilation tools, release 11.5, V11.5.119
Build cuda_11.5.r11.5/compiler.30672275_0
```

Gym Pyproject.toml

[tool.pixi.system-requirements]

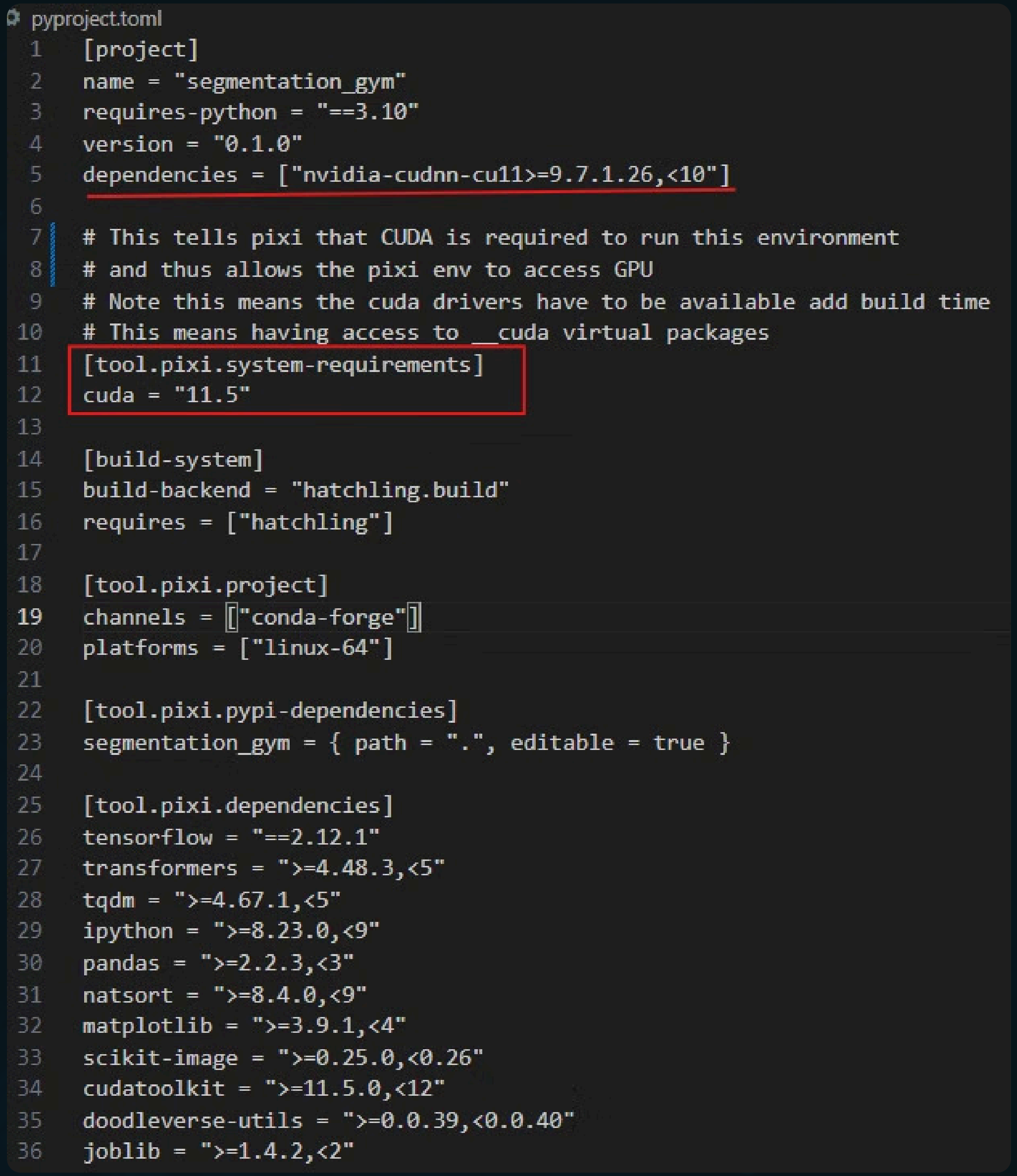
```
[tool.pixi.system-requirements]
cuda = "11.5"
```

- This tells pixi that the environment needs to be installed so that it can access CUDA

[project]

```
dependencies = ["nvidia-cudnn-cu11>=9.7.1.26,<10"]
```

- Notice that I am using `nvidia-cudnn-cu11` instead of `nvidia-cudnn-cu12`
- Your `nvidia-cudnn` MUST match the CUDA version defined in system requirements



pyproject.toml

```toml
[project]
name = "segmentation_gym"
requires-python = "==3.10"
version = "0.1.0"
dependencies = ["nvidia-cudnn-cu11>=9.7.1.26,<10"]

# This tells pixi that CUDA is required to run this environment
# and thus allows the pixi env to access GPU
# Note this means the cuda drivers have to be available add build time
# This means having access to __cuda virtual packages
[tool.pixi.system-requirements]
cuda = "11.5"

[build-system]
build-backend = "hatchling.build"
requires = ["hatchling"]

[tool.pixi.project]
channels = ["conda-forge"]
platforms = ["linux-64"]

[tool.pixi.pypi-dependencies]
segmentation_gym = { path = ".", editable = true }

[tool.pixi.dependencies]
tensorflow = "==2.12.1"
transformers = ">=4.48.3,<5"
tqdm = ">=4.67.1,<5"
ipython = ">=8.23.0,<9"
pandas = ">=2.2.3,<3"
natsort = ">=8.4.0,<9"
matplotlib = ">=3.9.1,<4"
scikit-image = ">=0.25.0,<0.26"
cudatoolkit = ">=11.5.0,<12"
doodleverse-utils = ">=0.0.39,<0.0.40"
joblib = ">=1.4.2,<2"
```

Gym Compose File

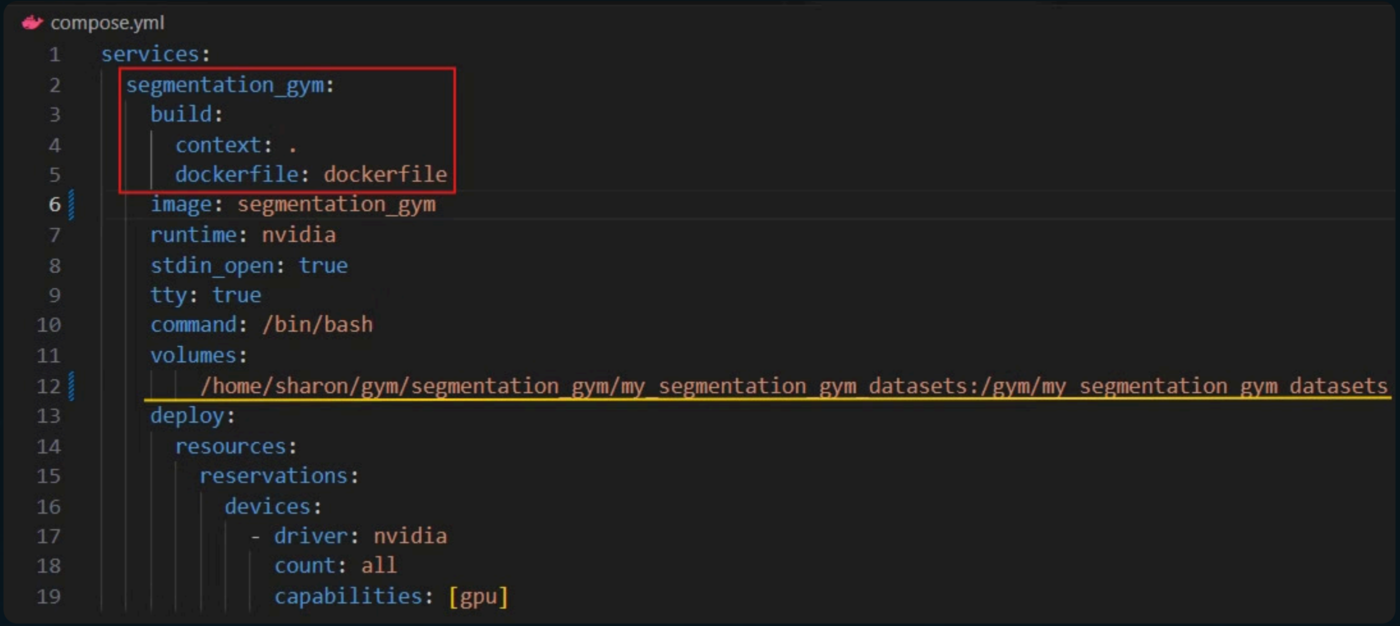
Use the gym dockerfile

- Tells compose to use the dockerfile in the current folder (context .) and call this service segmentation_gym

Bind Mount

Here we mount my local segmentation gym dataset to my container's segmentation gym dataset folder

- We use the <SOURCE>:<DEST> format



compose.yml

```yaml
services:
  segmentation_gym:
    build:
      context: .
      dockerfile: dockerfile
    image: segmentation_gym
    runtime: nvidia
    stdin_open: true
    tty: true
    command: /bin/bash
    volumes:
        /home/sharon/gym/segmentation_gym/my_segmentation_gym_datasets:/gym/my_segmentation_gym_datasets
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

Made with Gamma

Gym Docker Container GPU Access Issues

Docker does NOT have access to all the libdevice libraries since the compose file only gives the container access to the GPU at runtime NOT build time

- This means that you will need to run your model training and inference in eager mode

```
tf.config.run_functions_eagerly(True)
```

- If you want to access libdevice libraries you would need use a Nvidia image with a GPU as the base image

If you don't run the models in eager mode you will get the error below because these files are NOT in the container

```
packages/tensorflow/python/eager/execute.py", line 52, in quick_execute
    tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
tensorflow.python.framework.errors_impl.InternalError: libdevice not found at ./libdevice.10.bc
[Op:__inference__update_step_xla_5925]
```

CoastSeg Files

Essential Docker Files

Navigate to the CoastSeg repository or website to download these critical configuration files:

- Dockerfile - Contains instructions for building the CoastSeg image
- compose.yml - Defines services, networks, and volumes

These files provide the blueprint for creating your containerized CoastSeg environment and handling dependencies automatically.

Essential Pixi Files

Download these supporting files to ensure proper environment configuration:

- pixi.lock - Locks dependency versions for reproducibility
- pyproject.toml - Defines Python package specifications

These files ensure consistent package versions and proper Python environment configuration within your container.

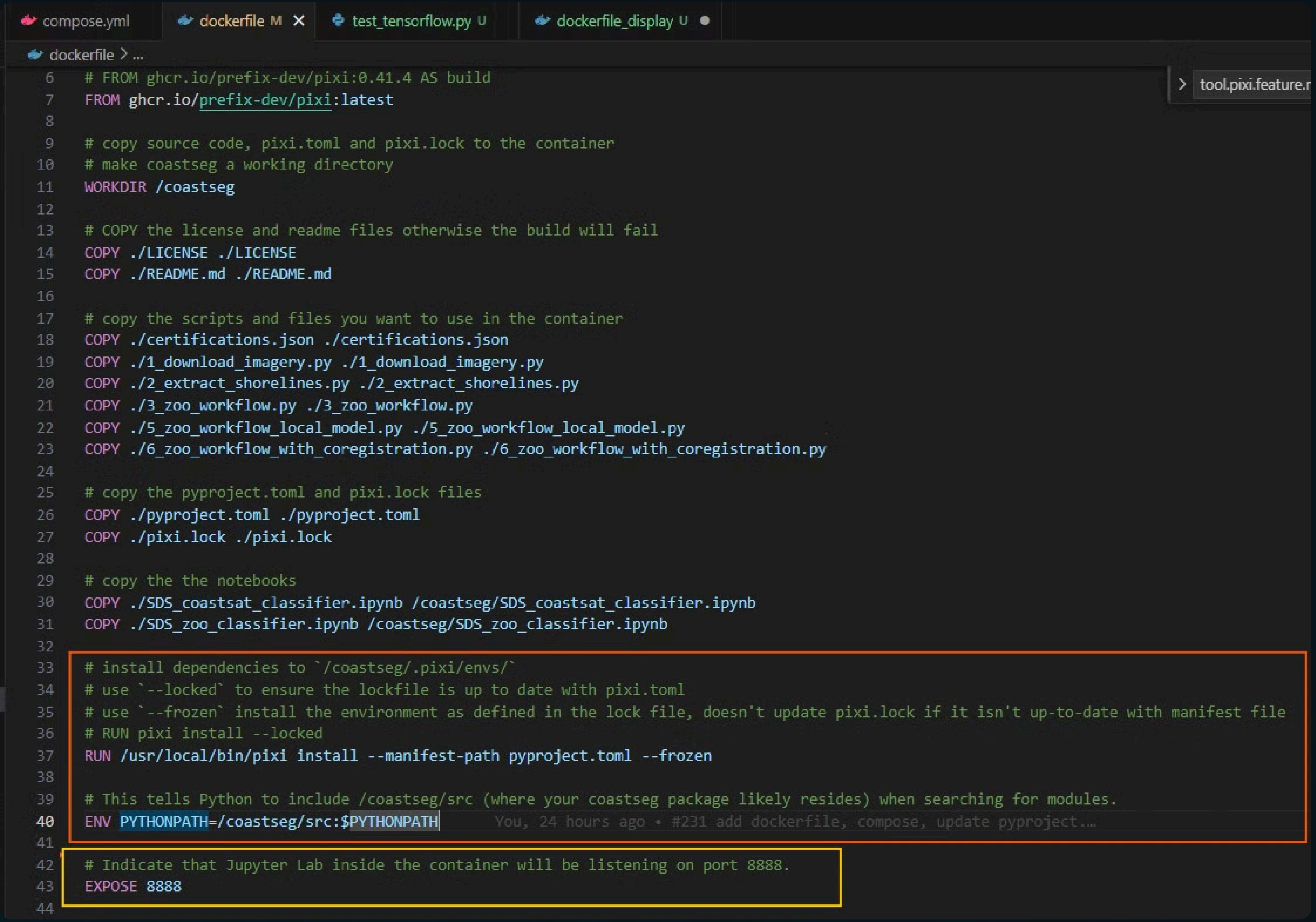
CoastSeg DockerFile

Pixi Setup

Install the exact versions of the dependencies listed in the pixi.lock file

Jupyter Lab Setup

Expose the container's port 8888 so we can connect to the jupyter lab instance will run on that port on our host machine



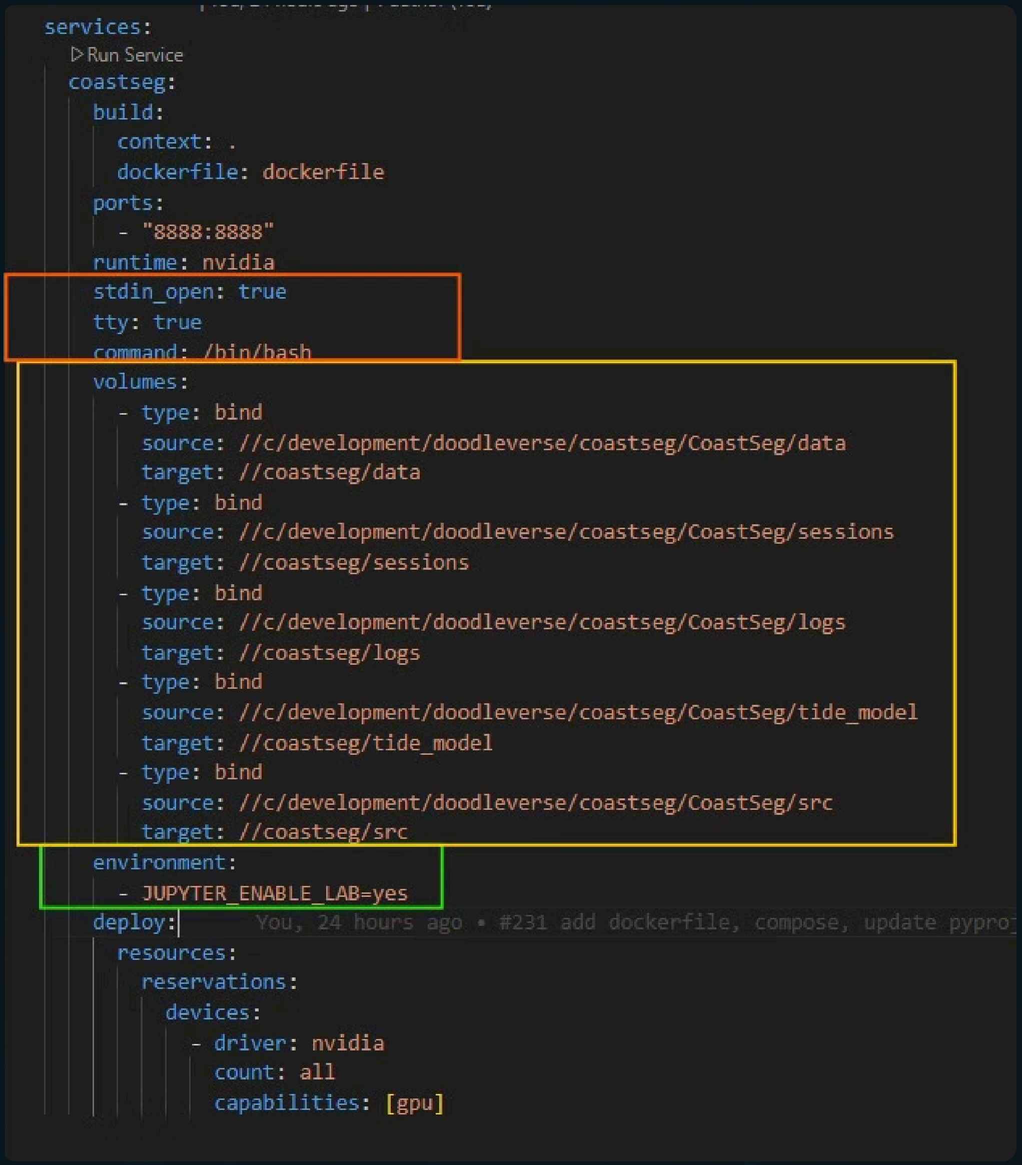
```dockerfile
# FROM ghcr.io/prefix-dev/pixi:0.41.4 AS build
FROM ghcr.io/prefix-dev/pixi:latest

# copy source code, pixi.toml and pixi.lock to the container
# make coastseg a working directory
WORKDIR /coastseg

# COPY the license and readme files otherwise the build will fail
COPY ./LICENSE ./LICENSE
COPY ./README.md ./README.md

# copy the scripts and files you want to use in the container
COPY ./certifications.json ./certifications.json
COPY ./1_download_imagery.py ./1_download_imagery.py
COPY ./2_extract_shorelines.py ./2_extract_shorelines.py
COPY ./3_zoo_workflow.py ./3_zoo_workflow.py
COPY ./5_zoo_workflow_local_model.py ./5_zoo_workflow_local_model.py
COPY ./6_zoo_workflow_with_coregistration.py ./6_zoo_workflow_with_coregistration.py

# copy the pyproject.toml and pixi.lock files
COPY ./pyproject.toml ./pyproject.toml
COPY ./pixi.lock ./pixi.lock

# copy the the notebooks
COPY ./SDS_coastsat_classifier.ipynb /coastseg/SDS_coastsat_classifier.ipynb
COPY ./SDS_zoo_classifier.ipynb /coastseg/SDS_zoo_classifier.ipynb

# install dependencies to `/coastseg/.pixi/envs/`
# use `--locked` to ensure the lockfile is up to date with pixi.toml
# use `--frozen` install the environment as defined in the lock file, doesn't update pixi.lock if it isn't up-to-date with manifest file
# RUN pixi install --locked
RUN /usr/local/bin/pixi install --manifest-path pyproject.toml --frozen

# This tells Python to include /coastseg/src (where your coastseg package likely resides) when searching for modules.
ENV PYTHONPATH=/coastseg/src:$PYTHONPATH

# Indicate that Jupyter Lab inside the container will be listening on port 8888.
EXPOSE 8888

```

CoastSeg Compose File

Allow the container to be run in an interactive shell

Mount the folders data, sessions, logs and the coastseg source code from my computer into the container

Enable JUPYTER LAB to run



```yaml
services:
  ▷ Run Service
  coastseg:
    build:
      context: .
      dockerfile: dockerfile
    ports:
      - "8888:8888"
    runtime: nvidia
    stdin_open: true
    tty: true
    command: /bin/bash
    volumes:
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/data
        target: //coastseg/data
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/sessions
        target: //coastseg/sessions
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/logs
        target: //coastseg/logs
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/tide_model
        target: //coastseg/tide_model
      - type: bind
        source: //c/development/doodleverse/coastseg/CoastSeg/src
        target: //coastseg/src
    environment:
      - JUPYTER_ENABLE_LAB=yes
    deploy:      You, 24 hours ago • #231 add dockerfile, compose, update pypro
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

Build & Run the Container

Step 1

```
docker compose up -d --build
```

- -d : mean build the container in deatached mode
- --build: means build the docker container if it does not exist

Step 2

```
docker ps
```

- Lists all the running docker containers

Step 3

```
docker exec -it <CONTAINER ID>
```

-it creates an interactive terminal session, allowing you to run commands directly inside the container as if you were working in a normal shell environment.

Step 4

```
pixi shell --frozen
```

- --frozen: install the environment as defined in the lockfile. Without checking the status of the lockfile.
- --locked: only install if the pixi.lock is up-to-date with the pixi.toml1. Conflicts with --frozen.

Made with Gamma

Building the CoastSeg Container

Build the Container

```
PS C:\development\doodleverse\coastseg\CoastSeg> docker compose up -d --build
```

Docker PS

```
PS C:\development\doodleverse\coastseg\CoastSeg> docker ps
CONTAINER ID   IMAGE              COMMAND       CREATED              STATUS              PORTS                    NAMES
d0ce32be11f6   coastseg-coastseg   "/bin/bash"   About a minute ago   Up About a minute
0.0.0.0:8888->8888/tcp   coastseg-coastseg-1
```

Launch the Docker Container in Interactive Mode

```
PS C:\development\doodleverse\coastseg\CoastSeg> docker exec -it d0ce32be11f6 /bin/bash
```

Running Jupyter Notebook in Docker

CoastSat workflow

```
> pixi shell --frozen
(coastseg) > jupyter lab SDS_coastsat_classifier.ipynb --ip=0.0.0.0 --allow-root --no-browser
```

Zoo Workflow

```
> pixi shell -e ml --frozen
(coastseg:ml) > pixi run run_notebook
```

Core Docker Commands

Command	Description
`docker ps`	List only running containers.
`docker ps -a`	List all containers (both running and stopped).
`docker stop`	Stop one or more running containers (use container ID or name).
`docker images`	List all Docker images stored locally.
`docker rm`	Remove one or more containers (use container ID or name).
`docker info`	Display system-wide information about Docker, including configuration and usage.

Wrap Up

- Dockerfiles let you configure how to build your image
- Docker compose files let you define how to build your image
- I have more guides available at https://2320sharon.github.io/reproducible_environments_guide/