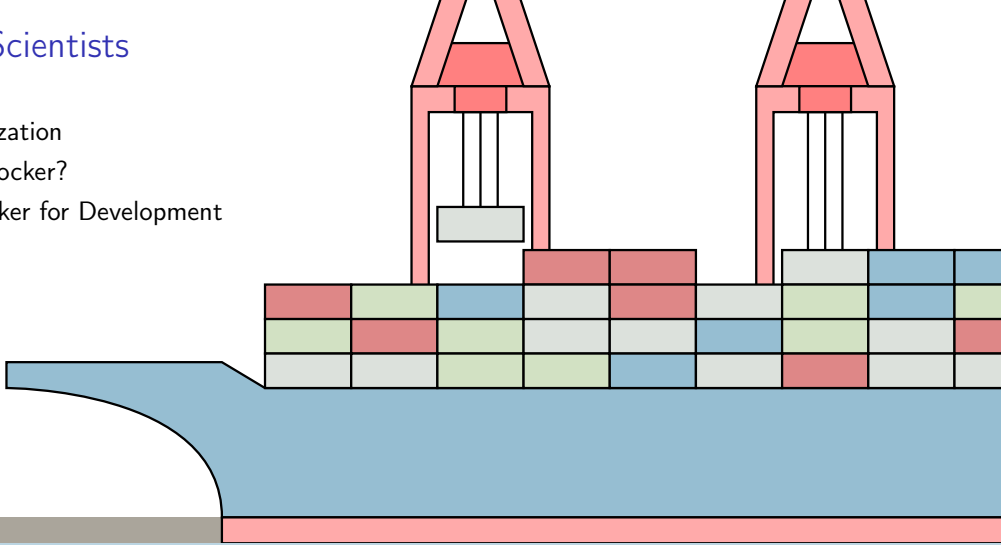
Docker for Scientists

- Containerization
- What is Docker?
- Using Docker for Development



Containerization

- *"Well, it runs on my computer."* - software doesn't always work correctly in all environments.
- Too many variables (OS, version, clutter, directory layout, etc.).

- Shipping containers revolutionized bulk shipping & transit.
- Interchangable, scalable, & standardized.
- On its own, each container is a single pre-packaged unit of cargo.

- Can we do the same with software?
- Package an application/code into its own environment.
- Known as containerization.

What is Docker?

- Docker is an application that helps ~~developers~~ scientists build and deploy software by packaging code in lightweight, portable containers.

- Software "repeatability".
- Share code *& the environment.*
- *Run anywhere.*

- *"Kind of like a mini virtual machine."* - Package code with the environment it was designed in.

Using Docker for Development

- First, install Docker.
  `https://docs.docker.com/get-docker/`

```
user@host:~$ docker --version

Docker version 20.10.17, build 100c701
```

Images

- Pre-built environments with certain applications installed.
- E.g. `python`, `julia`, `mathworks/matlab`, `r-base`

Containers

- Instance of the image.
- Images are the “template”, containers are the running environments.

Docker Images

- Images are the "template" that we build upon.
- Specifies:
  - Operating system (typically linux).
  - Installed applications (such as Python).
  - Installed packages (e.g. from `pip`).
  - Other configuration.
- Several pre-built images, e.g. `python`, `python:3.9`, `python:3.7`
- To install, use the `docker pull` command.

```
user@host:~$ docker pull python:3.9
user@host:~$ docker images

REPOSITORY   TAG   IMAGE ID       CREATED       SIZE
python       3.9   b673851840e8   2 weeks ago   915MB
```

Docker Containers

- Containers are the instances of the images.
- Each one is an isolated environment.
- **Example:** start a container, exit a container, list all containers, delete a container.

```
user@host:~$ docker run -it python bash
root@ebc5edf68c13:~$ exit
```

```
user@host:~$ docker ps -a

CONTAINER ID   IMAGE    COMMAND   CREATED          STATUS                      PORTS     NAMES
ebc5edf68c13   python   "bash"    5 seconds ago    Exited (0) 2 seconds ago              xenodochial_y
```

```
user@host:~$ docker rm ebc5edf68c13

ebc5edf68c13
```

A Deeper Look

```
user@host:~$ docker run -it python bash
```

- The `-it` flags specify that we want the container to be interactive (meaning we can type commands) and create a TTY (connected to our terminal).
- We then tell it which image to use, e.g. `python`.
- We then optionally specify which command to run. E.g. `bash`, a command prompt.
- We are now connected to the container, meaning we can type commands.

```
root@ebc5edf68c13:~$ python

Python 3.9.13 (main, Jul 12 2022, 12:04:02)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>> exit()
```

Reconnecting to Containers

```
user@host:~$ docker run -it python bash
root@ebc5edf68c13:~$ exit
```

- Once we start and exit a container, the container is "stopped".
- We can see all containers with `docker ps -a`.

```
user@host:~$ docker ps -a

CONTAINER ID   IMAGE    COMMAND   CREATED          STATUS                      PORTS     NAMES
ebc5edf68c13   python   "bash"    5 seconds ago    Exited (0) 2 seconds ago              xenodochial_y
```

- We can reconnect to a stopped container using `docker start -i <Container ID>`.

```
user@host:~$ docker start -i ebc5edf68c13
root@ebc5edf68c13:~$
```

Why is this useful?

- At this point, you may be asking, "why is this useful?"

- Pull images with programming languages already installed.
- The containers are completely independent.
  - Means we can destroy them without fear of breaking things.
- The container has its own file system & applications.
  - Means we can download tools & code in the container without cluttering up the actual machine.
  - We can install the packages/versions we need.

```
user@host:~$ docker run -it python bash
root@ebc5edf68c13:~$ git pull https://github.com/<user>/<your-repository>.git code
root@ebc5edf68c13:~$ cd code
root@ebc5edf68c13:~/code$ pip install numpy==0.21
root@ebc5edf68c13:~/code$ python my_script.py
```

Connecting Directories to Containers

- Code is on your machine in some folder, say `~/my_project/` .
- We want to run a script, `~/my_project/my_script.py` inside the docker container.
- We use `-v "$PWD:~/code"` to map the current folder to the `~/code` folder in the container[1].
- All changes are shared between the container and your machine.

```
user@host:~$ cd my_project
user@host:~/my_project$ ls

my_script.py  README.md  data


user@host:~/my_project$ docker run -it -v "$PWD:~/code" python bash
root@ebc5edf68c13:~$ cd code
root@ebc5edf68c13:~/code$ ls

my_script.py  README.md  data


root@ebc5edf68c13:~/code$ python my_script.py
```

[1] Use `%cd%` instead of `$PWD` on windows machines.

Dockerfiles

- Can provide a `Dockerfile`, which specifies the commands to set up an image.
- For example, the following `Dockerfile` is based on `python:3.9` and installs some packages.

```
FROM python:3.9
RUN apt-get -y update && apt-get install -y libblas-dev liblapack-dev
RUN pip install -U numpy==0.21 scipy scikit-learn matplotlib
```

```
user@host:~$ cd my_project
user@host:~/my_project$ ls

my_script.py  README.md  data  Dockerfile


user@host:~/my_project$ docker build -t my_image .
```

- Here, `-t my_image` specifies the "tag" or name of your image.

```
user@host:~/my_project$ docker run -it my_image bash
```

(Some) Tools Using Docker

- Docker Hub - `https://hub.docker.com`
  - Collection of pre-made images that you can download using `docker pull`.
  - Post your own images that other people can download.
- CodeOcean - `https://codeocean.com`
  - Cloud-based service for scientific code sharing & repeatability.
- Visual Studio Code
  - Great compatibility with docker & docker development workflows.
  - Develop in VS Code, run code in container.
  - Interactive code debugging.
- Github Actions
  - Tools for testing & building code automatically.
  - Works seamlessly with existing Github repos.
- Github Codespaces
  - Develop & run code online, using cloud computing resources (more ram, more cores, bigger GPUs).

Advanced Topics

- Github Actions for testing & building code automatically.
- Connecting to ports to serve websites & run web applications.
- Data volumes for sharing datasets.
- Composing containers (i.e. `docker compose`).
- Connecting to devices, peripherals, GPUs, etc.
- Running containers in the cloud (AWS, Google Cloud, Azure, etc.)