Introduction to using Containers and Virtual Environments

G. Watts (University of Washington/Seattle)

Analysis Group

Established analysis group!

Just arrived New Member with some great new ideas to improve the analysis

How do they even make the first plot!?!

1. How do you log in?
2. How do you access the data?
3. How do you setup your software environment?
4. How do you access the group’s common software?
5. How do you make a plot?

Q: what if you are working on two projects with conflicting requirements?

How do they even make the first plot!?!

1. How do you log in?
2. How do you access the data?
3. How do you setup your software environment?
4. How do you access the group's common software?
5. How do you make a plot?

Containers

Virtual Environments

Containers & Virtual Environment

Taken a piece of software from one computer to another and found that it doesn’t work?

Had to install a bunch of dependencies to run a piece of software written by a colleague?

What about saying “it works on my machine” when someone else is having trouble running your code?

Containers and Virtual Environments answer these questions!

These Are Another Tool to get Science Done

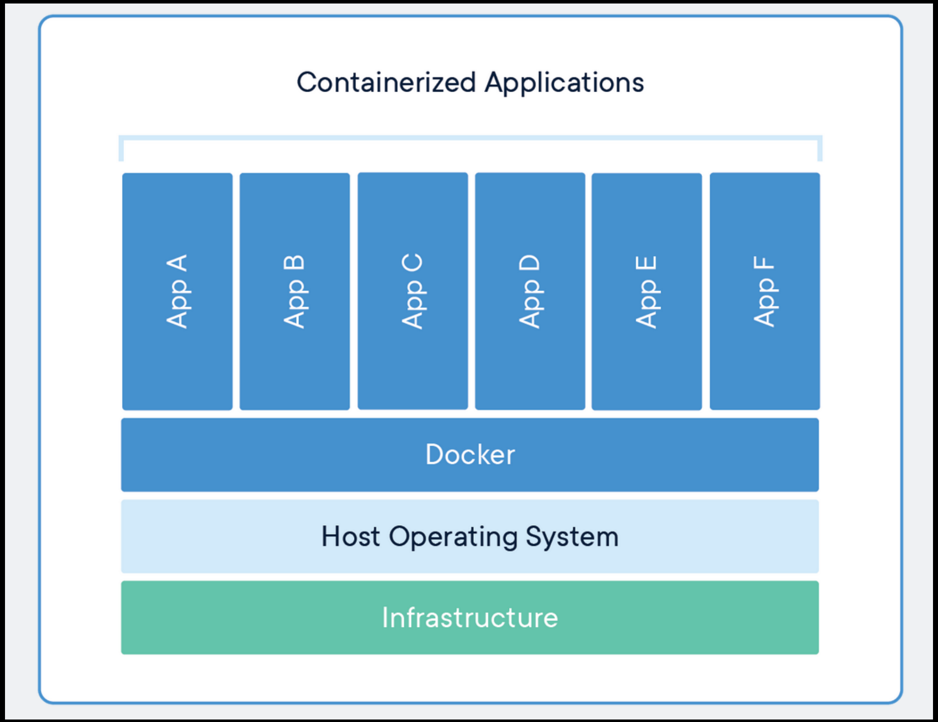
- These are tools
- They do take some time to use
- They do take resources (disk/networking)
- But they enable
  - Collaboration on software
  - Community building around software
  - Publishing of software
  - Reuse of software

Containers

Imagine if we'd tried to install all of the necessary packages for this tutorialon your laptops?

- Would have taken hours
- But…
  - Most of you would have been just fine, but one or two…
  - And some of you probably have other versions of the package installed – that you needed…

Binder is software built on containers!



Tutorial from HSF!!

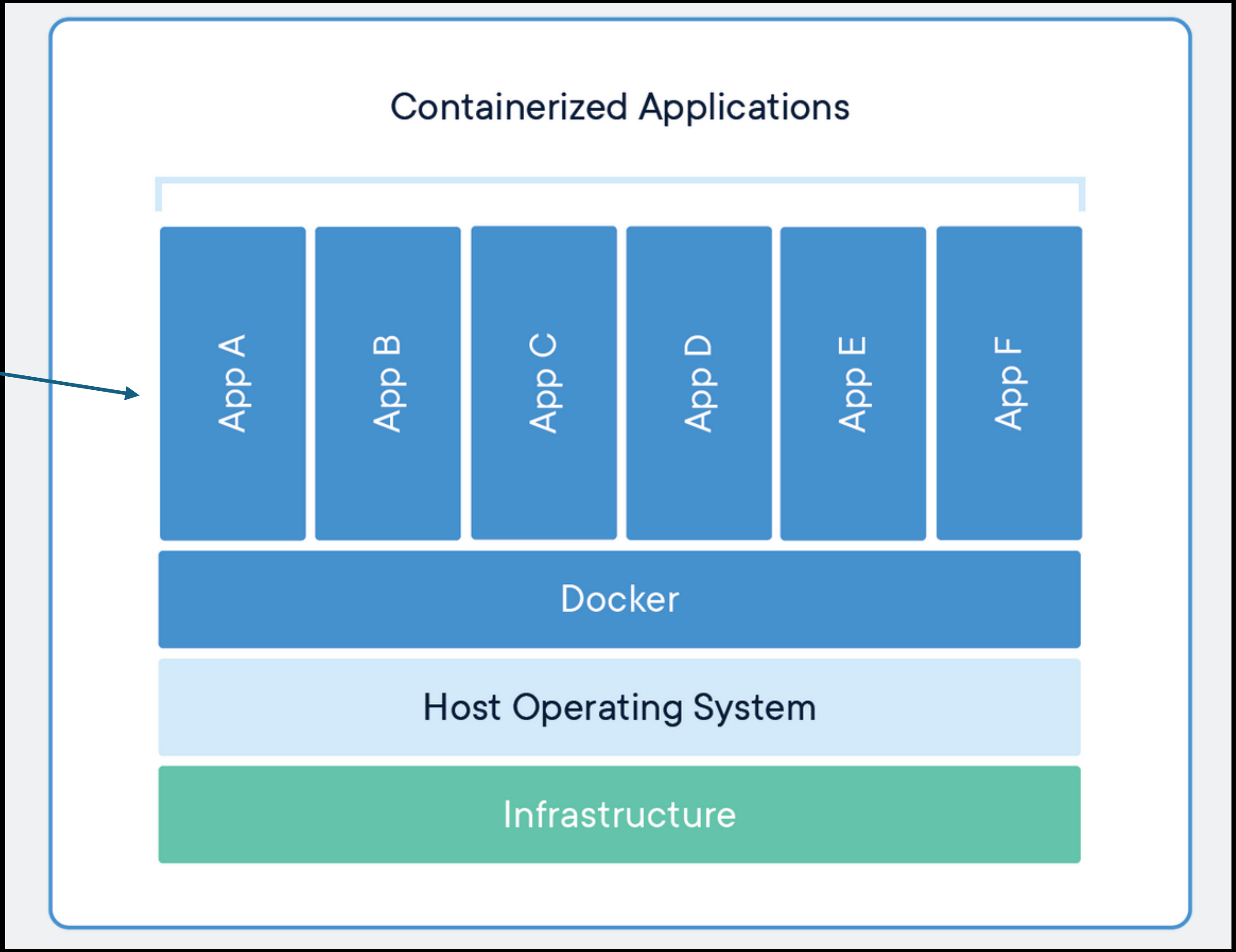
Containers

Built to emulate a computer/OS configuration!

You or your collaborators provide

Windows, Linux, Mac
- Docker
- Podman
- Sinularity
- Etc.

Provided…



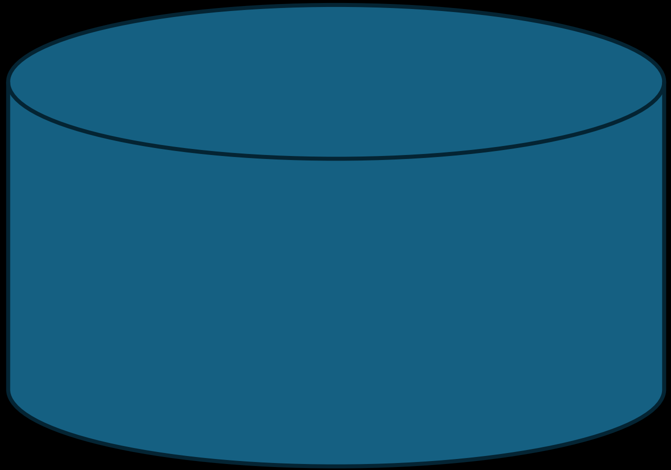
How Powerful Are they?

This is a windows laptop…

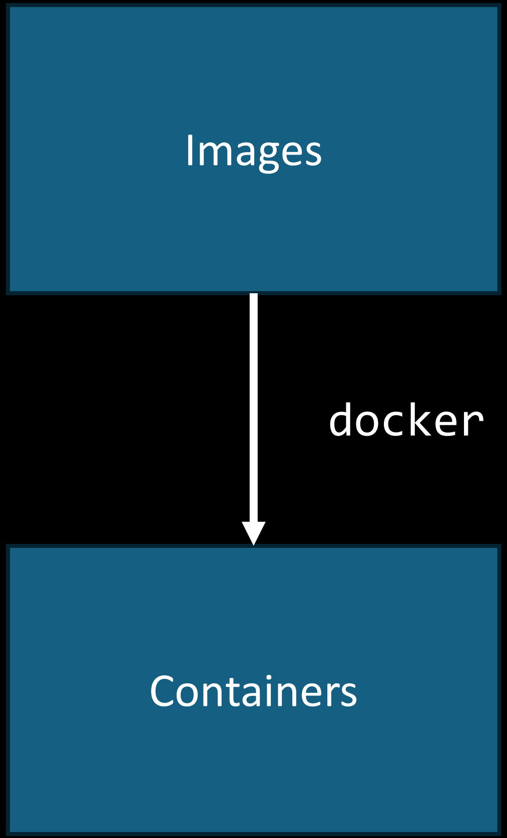
```
docker run --rm -it -v C:\\Users\\gordo\Code:/userhome rootproject/root
```

$\underbrace{\texttt{rootproject/root}}_{\text{The name of the container stored up on docker's hub}}$

What is the downside?



Working With Containers

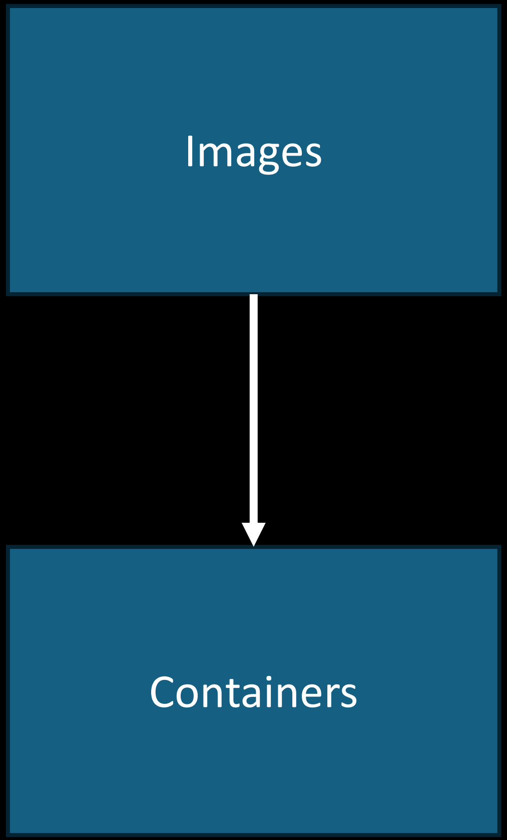


- On a repository on the net (e.g. hub.docker.com).
- Cached locally on your machine
- Built locally
- Once built or cached, can be re-used.
- No state information.

```
docker run <image-name>
```

- Two states: Running and Paused
- Running: running, you can interact with it
- Paused: cached state (files, etc.).

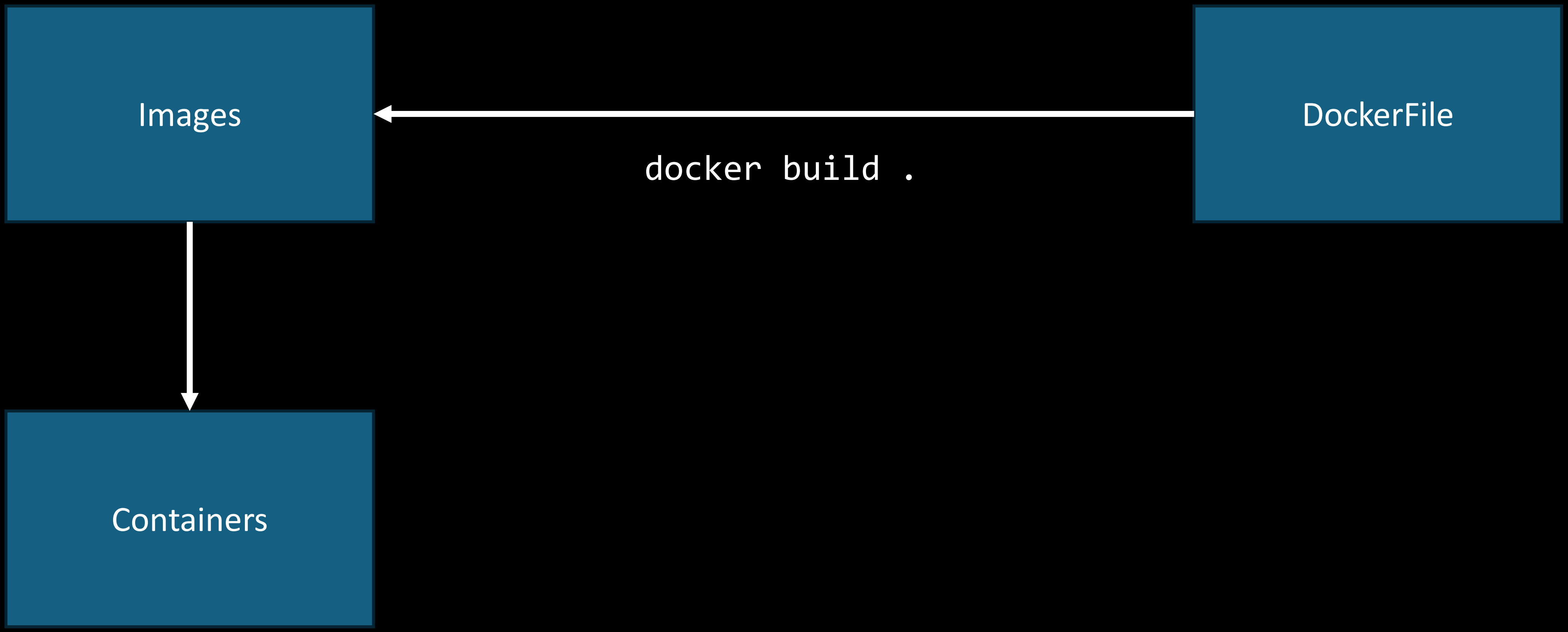
Working With Containers



```
docker images
```

```
docker ps -a
```

Working With Containers



```
docker build .
```

Let's Build A Hello World

hello.py

```
# Print hello
print(“hello world from a container!”)
```

Dockerfile

```
FROM python:3.11
```

Build & Run

```
docker build -t hello:0.1 .
docker run --rm -it hello:0.1
```

Getting the file on there…

Dockerfile

```
FROM python:3.11

WORKDIR /app
COPY hello.py hello.py
```

Build & Run

```
docker build -t hello:0.1 .
docker run --rm –it hello:0.2
docker run --rm -it hello:0.2 python /app/hello.py
```

Where is vim?
What about keeping the OS up to date?

Adding our own packages…

Dockerfile

```
FROM python:3.11

# Update the security
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y vim

# Load in the main file.
WORKDIR /app
COPY hello.py hello.py
```

```
docker build -t hello:0.3 .
docker run --rm -it hello:0.3 bash
```

Or

```
docker run --rm -it hello:0.3 python /app/hello.py
```

Starting and Stopping

This is valuable when you are using the docker container as a development environment.

The container, once up and running, will maintain state!

1. `docker run`
2. Edit the `hello.py` file
3. Exit
4. `docker start`
5. `docker attach`

Making it “stand-alone”

Note when we run this command:

docker run --rm -it hello:0.3

You were dropped into Python?

Can we make `hello.py` just run?

Dockerfile

```
FROM python:3.11

# Update the security
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y vim

# Load in the main file.
WORKDIR /app
COPY hello.py hello.py

CMD ["python", "/app/hello.py"]
```

Dockerfile you've been using…

```
FROM condaforge/mambaforge
COPY binder/environment.yml .

# Fix enter timezone issue -
ENV TZ=Asia/Kolkata
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install build-essential cmake  -y

RUN mamba env create -f environment.yml && mamba init bash

# Make RUN commands use the new environment:
RUN echo "mamba activate hsf-india" >> ~/.bashrc
SHELL ["/bin/bash", "--login", "-c"]
```

Virtual Environments

Built to solve the same problem in python

- Much more light weight than containers…
- Install only the packages that you need.

Virtual Environments

1. Install Python On Your Laptop!
2. Install Packages to work on Project 1
3. Install Packages to work on Project 2

Project 1 needs awkward < 1.0, and project 2 needs awkward >= 2.0!!!

$\}$ Virtual Environments to the rescue…

Virtual Environments are like having multiple (separate) python installations!

What do you need?

1. Python installed on your computer
2. Pip installed on your computer
3. List of packages you want installed!

The python interpreter and all libraries…

The package installer – pulls packages down from pypi!

A straight text file – “requirements.txt”

Let’s build an example

I want to run this code in a local notebook server:

```
import uproot
from hist.dask import Hist

f = uproot.dask({
    r"C:\Users\gordo\Downloads\dataWW_d1_split_removed.root": "tree_event;1;1"
})

h = Hist.new.Reg(50, 0.0, 100).Double()

filled_dask = h.fill(f.lep_pt_0)
```

Create the virtual environment

1

```
C:\Users\gordo\AppData\Local\Programs\Python\Python312\python.exe -m venv .venv
```

(customize for your OS, of course!)

Command to Create A Virtual Environment

In this directory

2

```
.\.venv\Scripts\Activate.ps1
python.exe -m pip install --upgrade pip
```

Install first packages

Create `requirements.txt`, and add `jupyterlab` to it.

```
pip install -r .\requirements.txt
jupyter-lab
```

Now – iterate until we have that code running!

Final requirements.txt

```
jupyterlab
uproot
dask[complete]
dask-awkward
hist[dask]
```

You can check this into your github repository along with the notebook file

- Now anyone can run your file!

Wait! I want to preserve this forever!!

- Run `pip freeze`.
- Use the output to create a new requirements.txt file

Keeping the mess under control…

1. Start with virtual environments
2. Containers later

Useful tools to make the code you develop more sustainable…

(next steps – create a generally useful library that you submit to pypi.org!!)