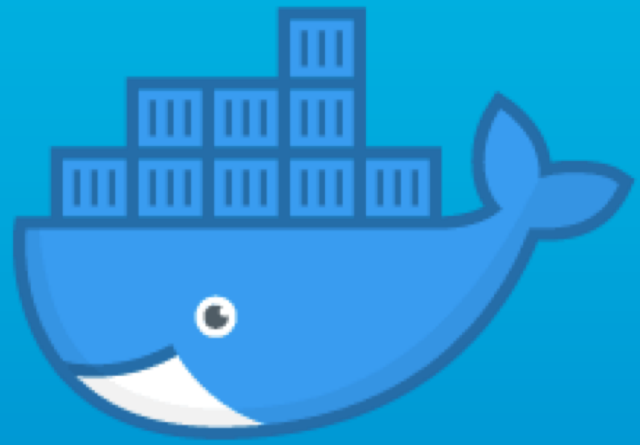
Docker basics

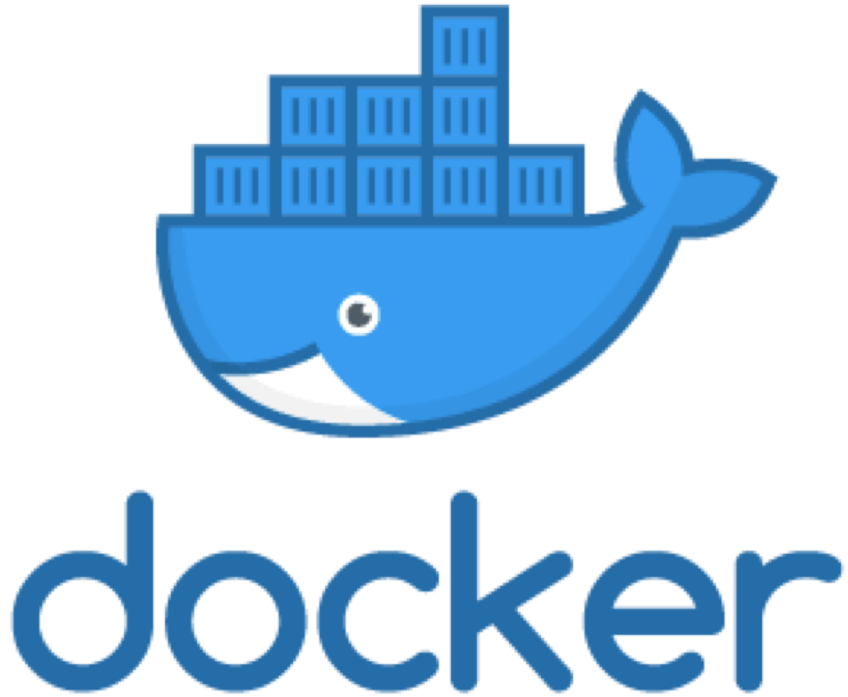
Andrea Panizza



October 3, 2019

Contents

```
$ docker why
$ docker what
$ docker how
$ depend --on-docker
$ docker --deep-learning
$ docker resources
```



docker why

- Choose an OS
- Install CUDA and CuDNN
- Install python (2.7? 3.5? 3.8?) & libraries
- Install tensorflow
- TF & CUDA versions not compatible
- Expose ports
- Possible conflicts
- Breaking updates
- Different OS/libraries in Dev/Prod
- Ok, but now it doesn't scale
- HEEEEELLLLPP!!!!!!

- Install Docker
- Write a Dockerfile
- docker build
- docker run
-

docker why

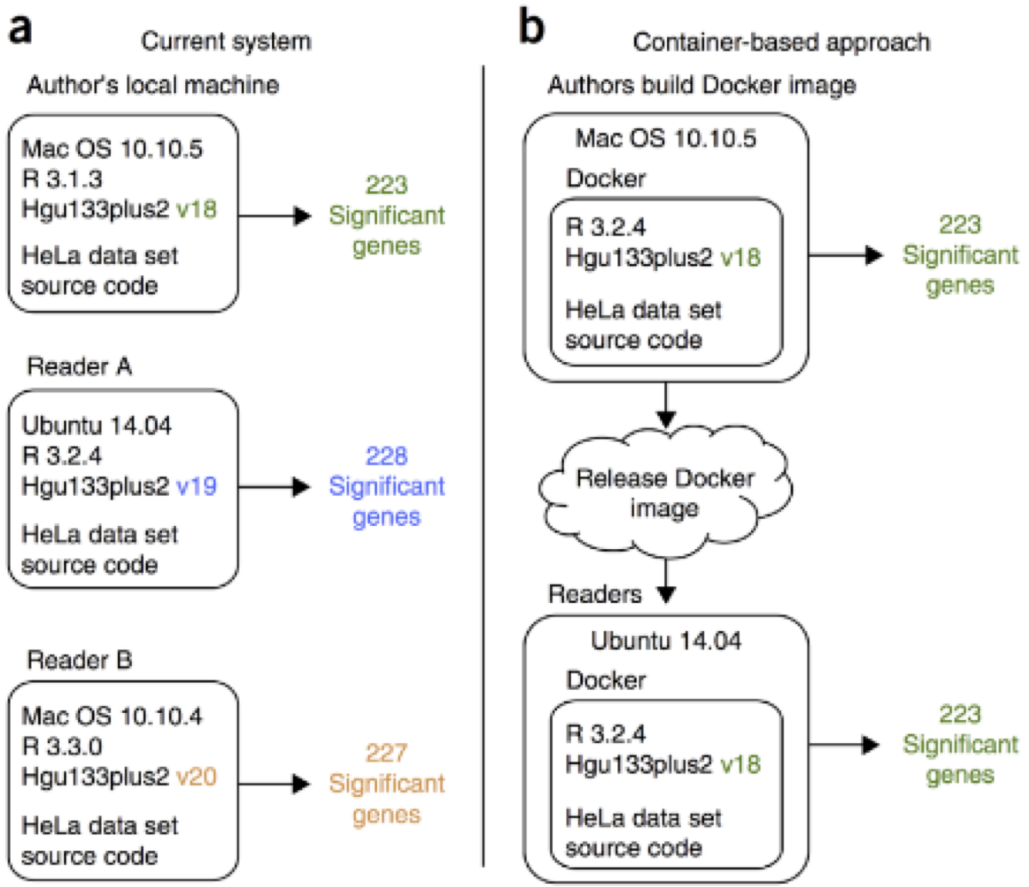
A true story

nature biotechnology

Reproducibility of computational workflows is automated using continuous analysis

Brett K Beaulieu-Jones[1] & Casey S Greene[2]

Beaulieu-Jones & Greene (2017) doi:10.1038/nbt.3780



`docker why`

Q: isn't this solved by `venv/pipenv/conda/` etc.?

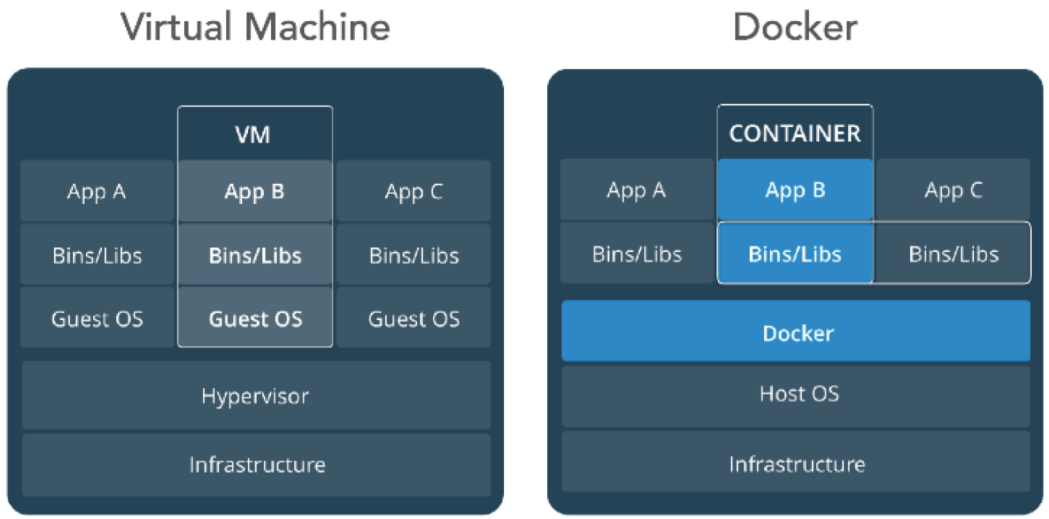
A: Nope!

- `venv & pipenv` only deal with Python packages dependencies. For anything else (e.g., Python version, CUDA, libglib2.0-0, etc.) you're pretty much f*ck*d. Also, a virtual environment is not really self-contained: some Python packages install stuff outside the virtual env dir.
- `conda` can manage environments for other languages other than Python, but it still doesn't manage the other dependencies
- None of these approaches work seamlessly across completely different environments, different compilers available, etc.

docker what

virtual machines solve the **dependency** issue…

…but they're very resource-intensive (**scalability**). We need a lighter framework



`docker what`

Docker containers are:

- Cheaper
- Less computational overhead
- Trendy
- Made to crash and be restarted
- Scaling and coordinating is easy
- **Develop code in your preferred environment, run anywhere**

docker what

Docker containers are **not** virtual machines:

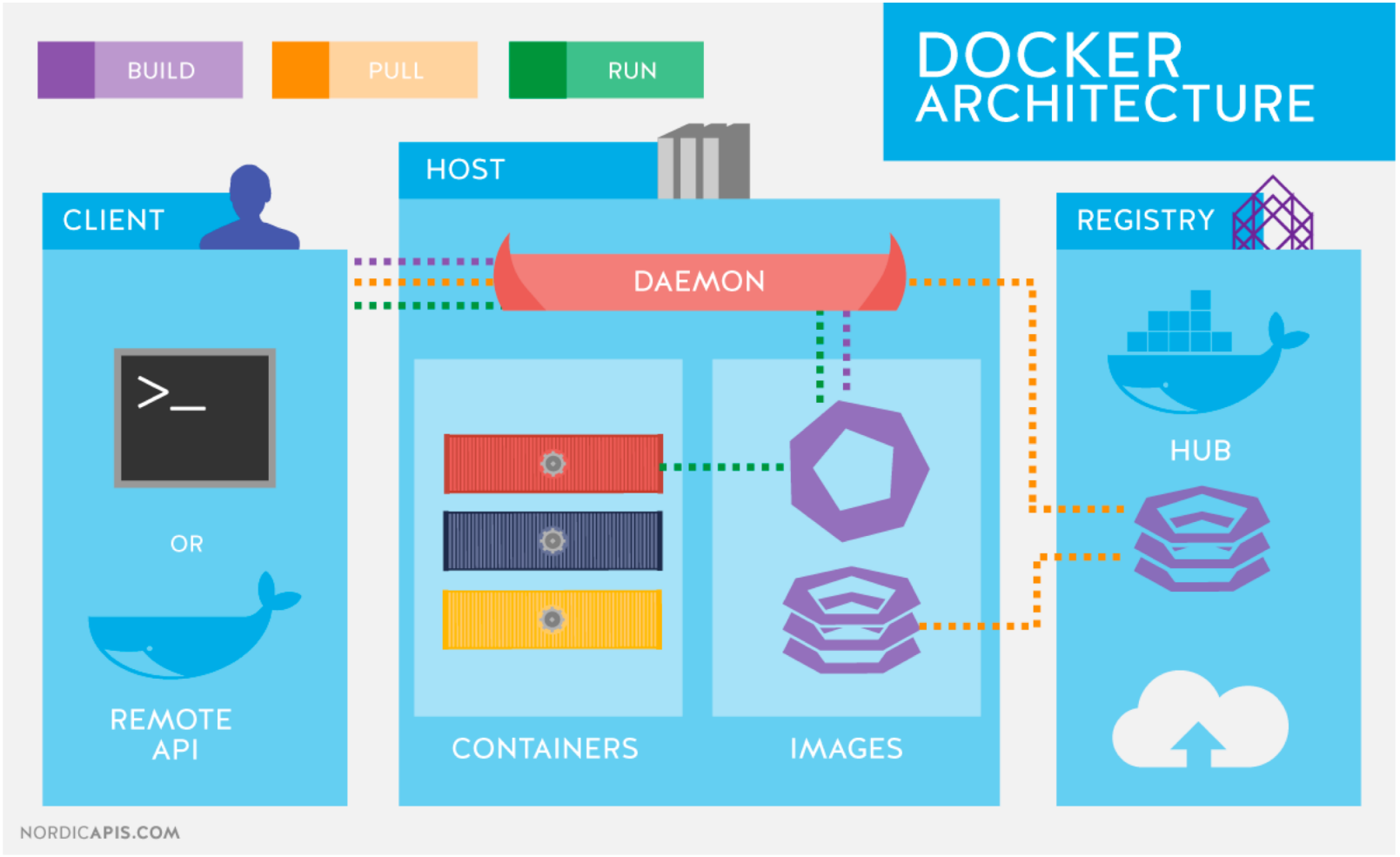
- “VMs are like houses, each is a single unit w/ standalone plumbing and wiring. Containers are like apartments; each one is a unit (process) in a host (building), sharing the plumbing and wiring w/ one another as well as the rest of the building” (Mike Coleman - Technology Evangelist, Docker)

docker vocabulary

image	package defining a working environment (binary file)
container	a running instance of an image (process)
Dockerfile	text file containing commands to build an image
Docker Hub	A website/repository for sharing Docker images
Docker client	sends CLI commands to Docker daemon through Docker REST API (process)
Docker daemon	executes calls from Docker client

Docker native client/server architecture makes it easy to design web services

docker architecture



docker how

- Define an image by writing a Dockerfile
- FROM: initialize a new build stage & set base image. A valid Dockerfile **must** start with FROM.
- LABEL: add metadata to an image (key/value pair). A typical key is maintainer
- COPY: copy files/folders from <src> and add to container filesystem at path <dest>
- RUN: execute commands in a new layer on top of the current image and commit results
- WORKDIR: set working directory for any command which follows
- CMD: defaults for the container execution

```
FROM tensorflow/tensorflow:1.13.1-py3-jupyter

LABEL maintainer="youremail@here.com"

RUN apt-get -y update -o Acquire::https::Verify-Peer=false \
    && apt-get -y install --no-install-recommends \
    libglib2.0-0 \
    libsm6 \
    libxext6 \
    libxrender1 \
    git \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /

ENV PATH="${PATH}:/root/.local/bin"

COPY requirements.txt /

RUN pip3 install --user --upgrade -r /requirements.txt

RUN git clone https://github.com/fizyr/keras-retinanet.git

RUN cd keras-retinanet && \
    pip3 install --user .

RUN pip3 install --user git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI

COPY Container-Root /

CMD ./startup.sh
```

docker how

To build the image, from the Dockerfile directory run

`docker build [BUILD_OPTS] -t IMAGE:[TAG]` (tip: **always** name & tag your images)

Docker downloads the base image from the Web (if it didn't do it before) and starts building. After building, to run the container, in any directory run

`docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]`
(can be complicated…but see later!)

Each container is an instance of an image: you can run multiple (containers) instances based on the same image, as long as each has a unique name.

Containers are stateless and immutable: once a container completes execution, all its content is lost, and any other container started from the same image doesn't inherit any state from the first one (**reproducibility**).

Mapping local folders

All content from a Docker container is lost, once it terminates execution. In order to export content from it, we **map** a folder on the host filesystem, to a folder in the Docker container filesystem. Example:

```
docker run -v /home/andrea/docker_test:/test -it ubuntu /bin/bash
```

maps the folder `docker_test` on the host system to the folder `/test` in the Docker container, starts a container based on the `ubuntu` image in interactive mode and drops you in the `bash` shell.

depend --on-docker

Depend On Docker is a framework developed at BHGE Digital which helps building, shipping & running applications with Docker.

`docker --deep-learning`

- Let’s build a simple image and make some practice with Depend on Docker
- We will create an image to run two models in Jupyter:
  - a pre-trained ResNet-50 model (uses default Keras implementation)
  - a pre-trained RetinaNet model (uses https://github.com/fizyr/keras-retinanet/)

```
git clone https://github.com/AndreaPi/docker-training-2019-public.git
```

Manage images/containers

docker images -aq	List all images
docker ps -a	**List all containers:** you need to remove containers before you can remove images
docker rm <container-ID>	remove container
docker rm $(docker ps -aq)	remove all containers
docker rmi <image-ID>	remove image
docker rmi $(docker images -aq)	Remove all images
docker run --rm --it --name cntr myimage	run container with name **cntr**, from image *myimage*, in interactive (*it*) mode and remove (*rm*) container after you exit: **great** to avoid having many hanging containers

A few extra tips on writing Dockerfiles

Order matters for caching

```
FROM debian
COPY . /app
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
COPY . /app
CMD ["java", "-jar", "/app/target/app.jar"]
```

Order from least to most frequently changing content.

More specific COPY to limit cache busts

```
FROM debian
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
COPY . /app
COPY target/app.jar /app
CMD ["java", "-jar", "/app/target/app.jar"]
```

Only copy what's needed. Avoid "COPY ." if possible

Line buddies: apt-get update & install

```
FROM debian
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
RUN apt-get update \
 && apt-get -y install \
    openjdk-8-jdk ssh vim
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Prevents using an outdated package cache

Remove unnecessary dependencies

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk ssh vim
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

from https://blog.docker.com/2019/07/intro-guide-to-dockerfile-best-practices/

A few extra tips on writing Dockerfiles

Remove package manager cache

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk \
 && rm -rf /var/lib/apt/lists/*
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Use official images when possible

```
~~FROM debian~~
~~RUN apt-get update \~~
~~ && apt-get -y install --no-install-recommends \~~
~~    openjdk-8-jdk\~~
~~ && rm -rf /var/lib/apt/lists/*~~
FROM openjdk
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Use more specific tags

```
~~FROM openjdk:latest~~
FROM openjdk:8
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

The "latest" tag is a rolling tag. Be specific, to prevent unexpected changes in your base image.

- The Docker blog post lists even more best practices which you may want to learn and use!

from https://blog.docker.com/2019/07/intro-guide-to-dockerfile-best-practices/

docker resources

- the official Docker cheatsheet
- Another cheatsheet
- Yet another one
- Some Docker notes
- https://docs.docker.com/engine/reference/builder/
- https://docs.docker.com/develop/develop-images/dockerfile_best-practices
- https://blog.docker.com/2019/07/intro-guide-to-dockerfile-best-practices/