

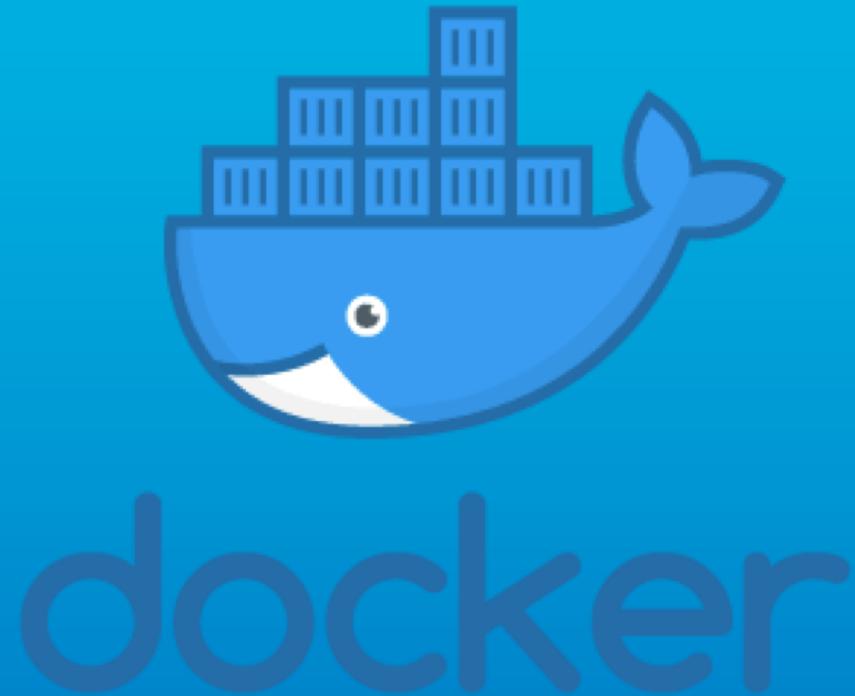
<https://github.com/AndreaPi/docker-for-deep-learning>

# Docker for DL

Andrea Panizza

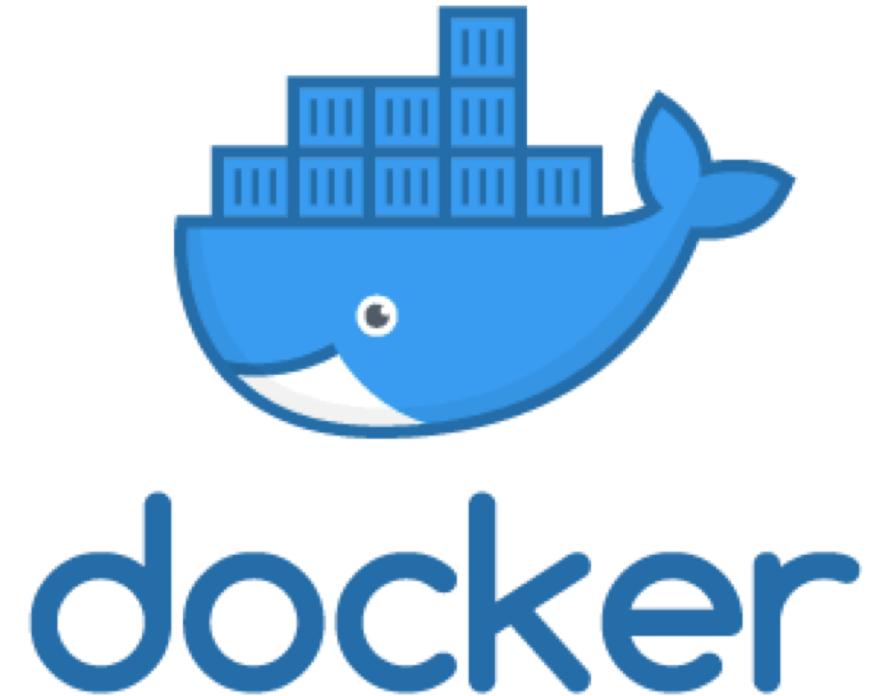


September 23, 2020



# Contents

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



# docker why

- Choose an OS
- Install CUDA and CuDNN
- Install python (3.2? 3.5? 3.8?) & libraries
- Install tensorflow & keras (or PyTorch)
- TF & CUDA versions not compatible
- TF & Keras versions not compatible
- Expose ports: possible conflicts
- Breaking updates
- Different OS/libraries in Dev/Prod
- Ok, but now it doesn't scale
- HEEEEELLLLPP!!!!!!



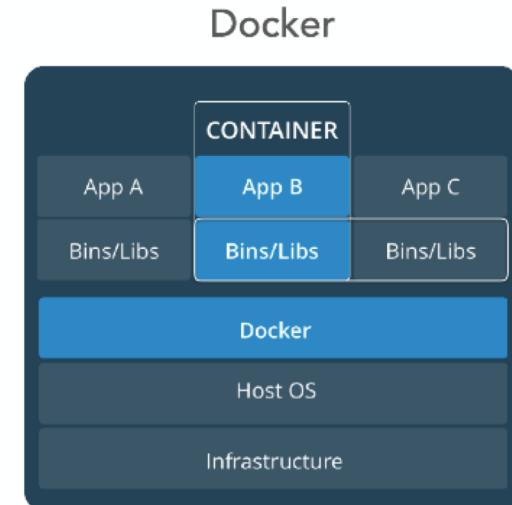
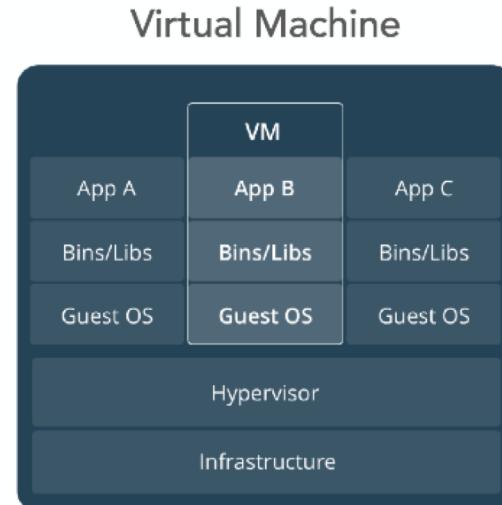
- Install Docker
- Write a Dockerfile
- docker build
- docker run
- 



# docker 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 memory & CPU overhead)
- Trendy
- **Reproducible & portable**
- Made to crash and be restarted
- Scaling and orchestrating 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)

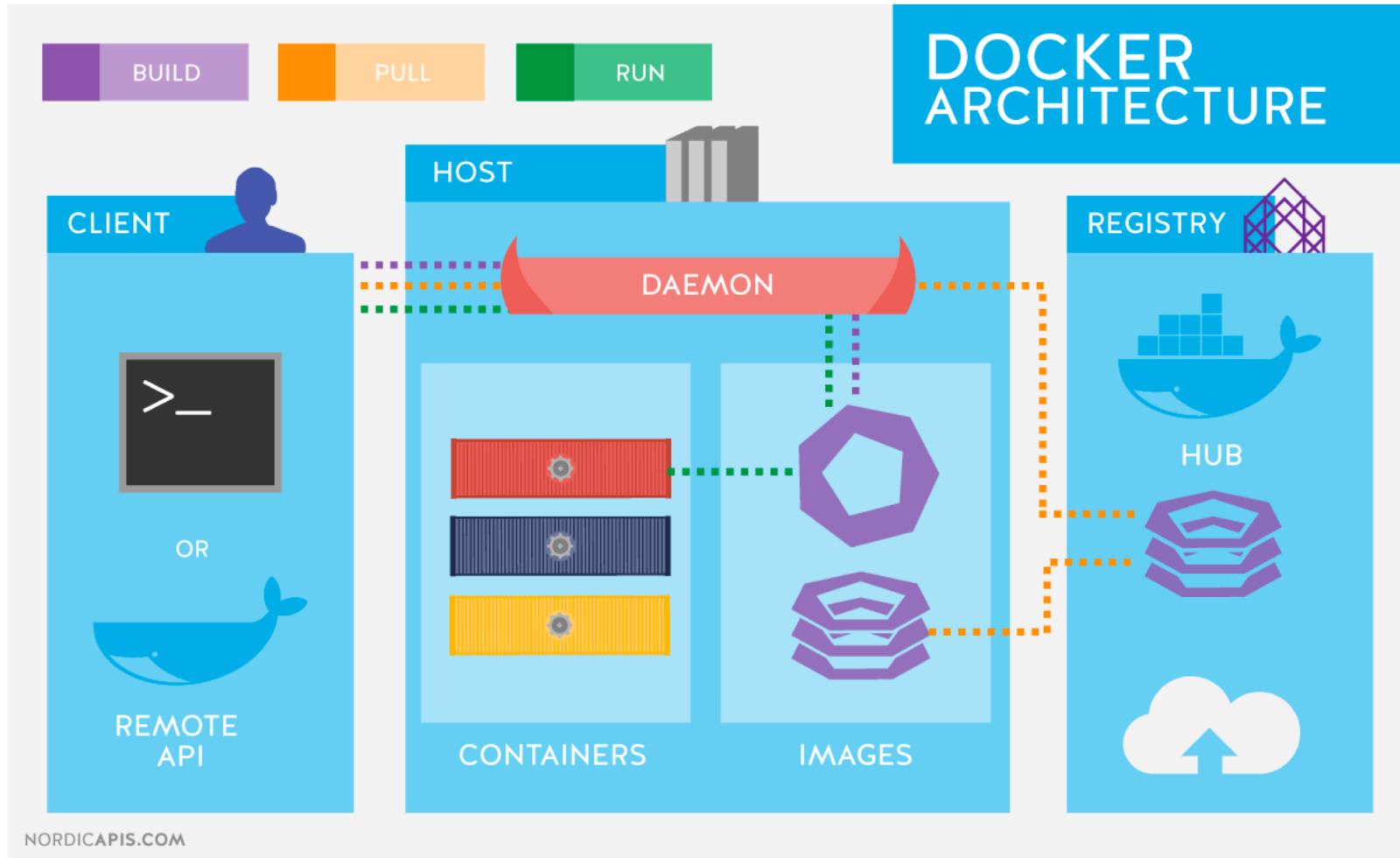


# docker vocabulary

<b>image</b>	package defining a working environment (binary file)
<b>container</b>	a running instance of an image (process)
<b>Dockerfile</b>	text file containing commands to build an image
<b>Docker Hub</b>	A web server/repository for sharing Docker images
<b>Docker client</b>	sends CLI commands to Docker daemon through Docker REST API
<b>Docker daemon</b>	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:2.3.0-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 \
libgl1-mesa-glx \
libsmb6 \
libxext6 \
libxrender1 \
git \
&& apt-get clean \
&& rm -rf \
/tmp/hyperfdata* \
/var/*/apt/*/partial \
/var/lib/apt/lists/*

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

WORKDIR /

CMD ["/bin/bash", "-c", "jupyter notebook --allow-root --ip='0.0.0.0' --no-browser --NotebookApp.token=''"']
```

# 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 pulls 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<sup>1</sup>, starts a container based on the `ubuntu` image in interactive mode and drops you in the `bash` shell.

---

<sup>1</sup>Warning: this kind of mapping is called a **bind mount** and it's discouraged in modern Docker versions, in favor of **volume mount**. As a Docker curmudgeon & a developer, I find bind mounting to be more useful for people like me who develop *inside* of a container, but you've been warned.

# portainer

Portainer is a web app (running inside a...you guessed it, a container) which allows you to manage your containers (especially local ones), remove unused containers, volumes, images, start and stop containers, etc.

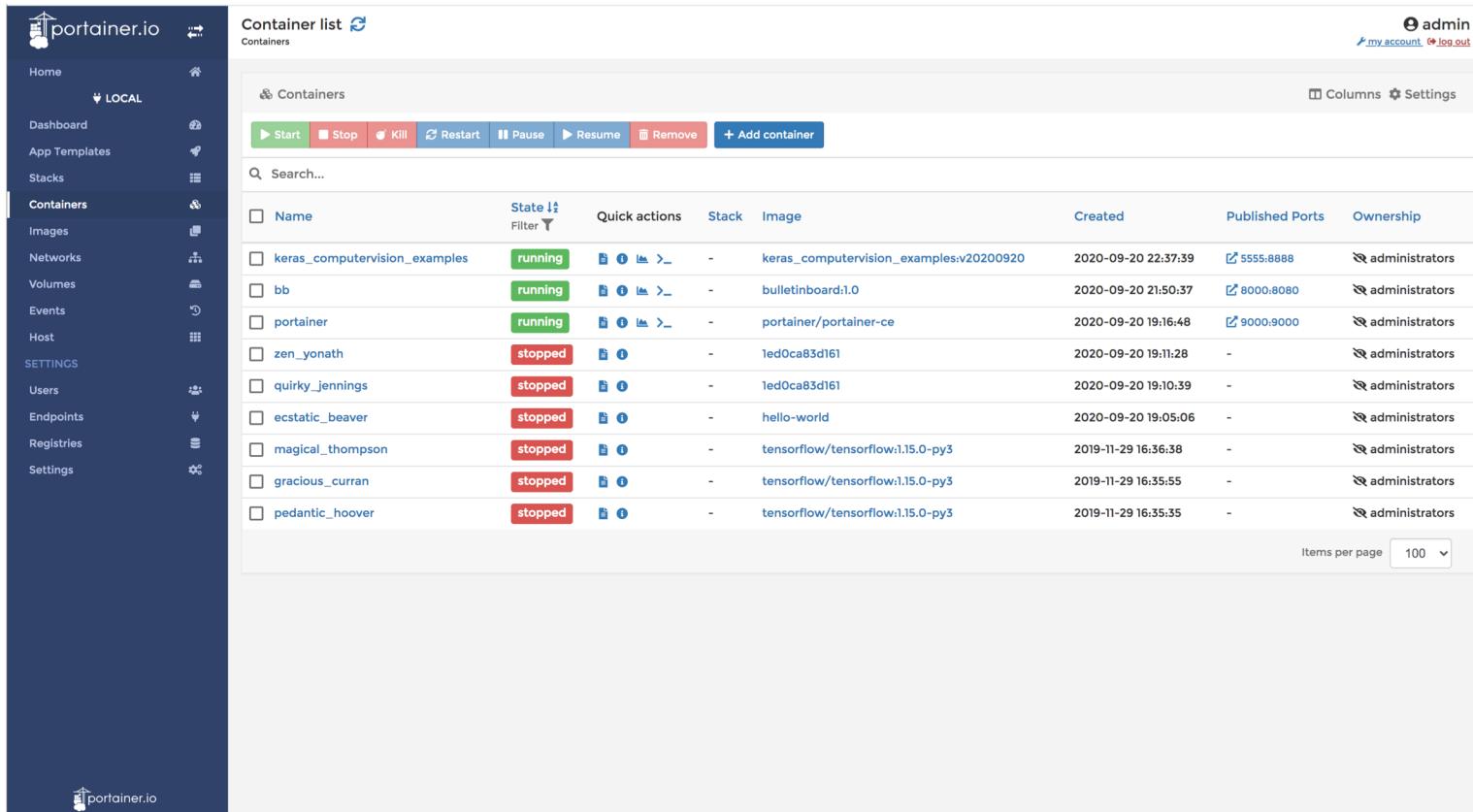


# portainer

Being based on Docker, installing Portainer requires just 2 shell commands:

```
docker volume create  
portainer_data
```

```
docker run -d -p 9000:9000 --  
name=portainer --  
restart=always -v  
/var/run/docker.sock:/var/run/  
docker.sock -v  
portainer_data:/data  
portainer/portainer-ce
```



The screenshot shows the Portainer web interface. On the left is a sidebar with a dark blue header containing the Portainer logo and the text "portainer.io". Below the header, the sidebar has a "LOCAL" section with several items: Home, Dashboard, App Templates, Stacks, Containers (which is currently selected and highlighted in blue), Images, Networks, Volumes, Events, Host, SETTINGS, Users, Endpoints, Registries, and Settings. To the right of the sidebar is the main content area. At the top of the content area is a header with the text "Container list" and a "Containers" dropdown. In the top right corner of the header, there is a user account icon with the text "admin" and links for "my account" and "log out". Below the header is a toolbar with buttons for "Start", "Stop", "Kill", "Restart", "Pause", "Resume", "Remove", and "+ Add container". Underneath the toolbar is a search bar with the placeholder "Search...". The main table area has a header row with columns: Name, State, Quick actions, Stack, Image, Created, Published Ports, and Ownership. There are 12 rows of data in the table, each representing a different Docker container. The containers listed are: keras\_computervision\_examples (running), bb (running), portainer (running), zen\_yonath (stopped), quirky\_jennings (stopped), ecstatic\_beaver (stopped), magical\_thompson (stopped), gracious\_curran (stopped), and pedantic\_hoover (stopped). The "Created" column shows dates ranging from 2020-09-20 to 2019-11-29. The "Published Ports" column shows various ports like 5555:8888, 8000:8080, 9000:9000, etc. The "Ownership" column shows "administrators" for most containers. At the bottom right of the table area, there is a "Items per page" dropdown set to 100.

Name	State	Quick actions	Stack	Image	Created	Published Ports	Ownership
keras_computervision_examples	running	[actions]	-	keras_computervision_examples:v20200920	2020-09-20 22:37:39	5555:8888	administrators
bb	running	[actions]	-	bulletinboard:1.0	2020-09-20 21:50:37	8000:8080	administrators
portainer	running	[actions]	-	portainer/portainer-ce	2020-09-20 19:16:48	9000:9000	administrators
zen_yonath	stopped	[actions]	-	TedOca83d161	2020-09-20 19:11:28	-	administrators
quirky_jennings	stopped	[actions]	-	TedOca83d161	2020-09-20 19:10:39	-	administrators
ecstatic_beaver	stopped	[actions]	-	hello-world	2020-09-20 19:05:06	-	administrators
magical_thompson	stopped	[actions]	-	tensorflow/tensorflow:1.15.0-py3	2019-11-29 16:36:38	-	administrators
gracious_curran	stopped	[actions]	-	tensorflow/tensorflow:1.15.0-py3	2019-11-29 16:35:55	-	administrators
pedantic_hoover	stopped	[actions]	-	tensorflow/tensorflow:1.15.0-py3	2019-11-29 16:35:35	-	administrators

Then, point your browser to **localhost:9000** et voilà! You're done.

# portainer

The screenshot shows the Portainer interface for connecting to a container. At the top, it says "Connect to containers". Below that, it says "Exec into container as default user using command bash" and has a "Disconnect" button. The main area is a terminal window displaying a black background with red text. The text includes a warning about running as root, instructions to run as a specific user, and a docker command. The terminal prompt shows "root@127.8a00d7b80:/#".

WARNING: You are running this container as root, which can cause new files in mounted volumes to be created as the root user on your host machine.  
To avoid this, run the container by specifying your user's userid:  
\$ docker run -u \$(id -u):\$(id -g) args...  
root@127.8a00d7b80:/#

The screenshot shows two Portainer pages. The top page is titled "Container list" and "Containers". It features a toolbar with buttons for Start, Stop, Kill, Restart, Pause, Resume, Remove, and Add container. A table lists three stopped containers: "pedantic\_hoover", "gracious\_curran", and "magical\_thompson", all based on the "tensorflow/tensorflow:1.15.0-py3" image. The bottom page is titled "Delete unused images" and shows a table of unused Docker images. The table includes columns for Id, Tags, Size, and Created. Several images are listed, including "node:current-slim", "tensorflow/tensorflow:2.3.0-jupyter", "keras\_retinanet:latest", "tensorflow/tensorflow:1.15.0-py3-jupyter", "mask-rcnn:v20190404", "python:latest", and "todoapp:latest".

Id	Tags	Size	Created
sha256:397ef203e8c4592663b16c9b396fb5...	Unused node:current-slim	167.3 MB	2020-09-
sha256:2d87e2e84687dcdfb878dab711fa48...	Unused tensorflow/tensorflow:2.3.0-jupyter	1.7 CB	2020-07-
sha256:5e4d82222818c5ae388fbace9ce947...	Unused keras_retinanet:latest	2.8 GB	2019-12-0
sha256:00c7a2dd3c3dd685a1e5d057fc20c...	Unused tensorflow/tensorflow:1.15.0-py3-jupyter	2.5 GB	2019-10-2
sha256:d87d69e3f7f792906ee1921882cc55...	Unused mask-rcnn:v20190404	2.5 GB	2019-04-0
sha256:32260605cf7a4819c1490d0d1d1a2e...	Unused python:latest	928.9 MB	2019-03-0
sha256:20a0a412e0f344d8909c7bb98208d7...	Unused todoapp:latest	943.9 MB	2018-11-18

...and more!

# portainer



- Portainer 2.0 CE (08/31/2020) supports Kubernetes, allowing users to manage app deployments on Kubernetes clusters using the Portainer UX (“no more CLI, no more YAML, no more Kubernetes manifests”)
- Faces the competition of Rancher and OpenShift!

## **docker --deep-learning**

- Let's build a simple image and make some practice with 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-for-deep-learning.git
```

# docker resources

- [How to Get Started with Docker \(\*official intro atm\*\)](#)
  - [What is Docker in 5 minutes](#)
  - [Learn Docker in 12 minutes](#)
  - [Portainer - Lightweight Management UI for Docker](#)
  - [the official Docker cheatsheet](#)
  - [Some Docker notes](#)
  - [https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices](https://docs.docker.com/develop/develop-images/dockerfile_best-practices)
  - <https://blog.docker.com/2019/07/intro-guide-to-dockerfile-best-practices/>
-

# BACKUP

# docker why

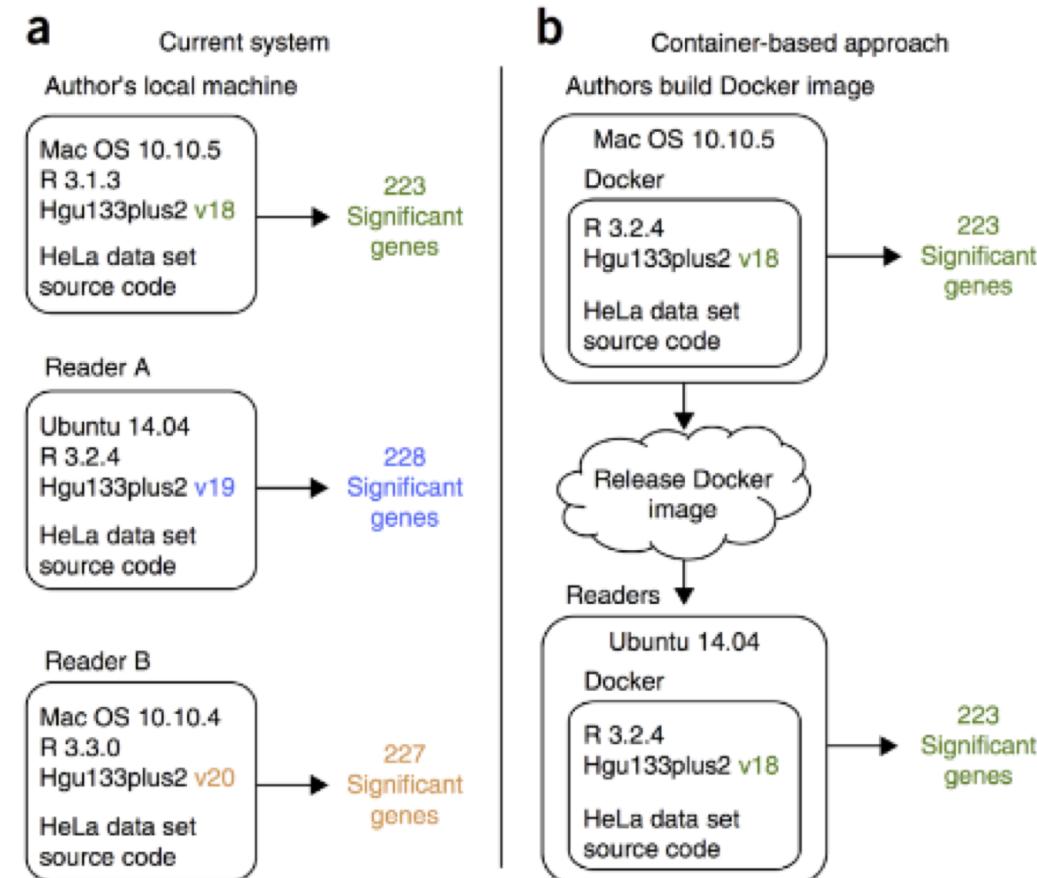
## A true story

nature  
biotechnology

Reproducibility of computational workflows is automated using continuous analysis

Brett K Beaulieu-Jones<sup>1</sup> & Casey S Greene<sup>2</sup>

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 guarantees seamless transferability across different environments. **You will all have experiences of conda/venv crashing after some admin change on your machines!**

## `depend --on-docker`

[Depend On Docker](#) is a framework developed at former BHGE Digital which helps building, shipping & running applications with Docker.



# Manage images/containers

**docker images -aq**

**docker ps -a**

**docker rm <container-ID>**

**docker rm \$(docker ps -aq)**

**docker rmi <image-ID>**

**docker rmi \$(docker images -aq)**

**docker run --rm --it --name cntr myimage**

List all images

**List all containers:** you need to remove containers before you can remove images  
remove container

remove all containers

remove image

Remove all images

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"]
```

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

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