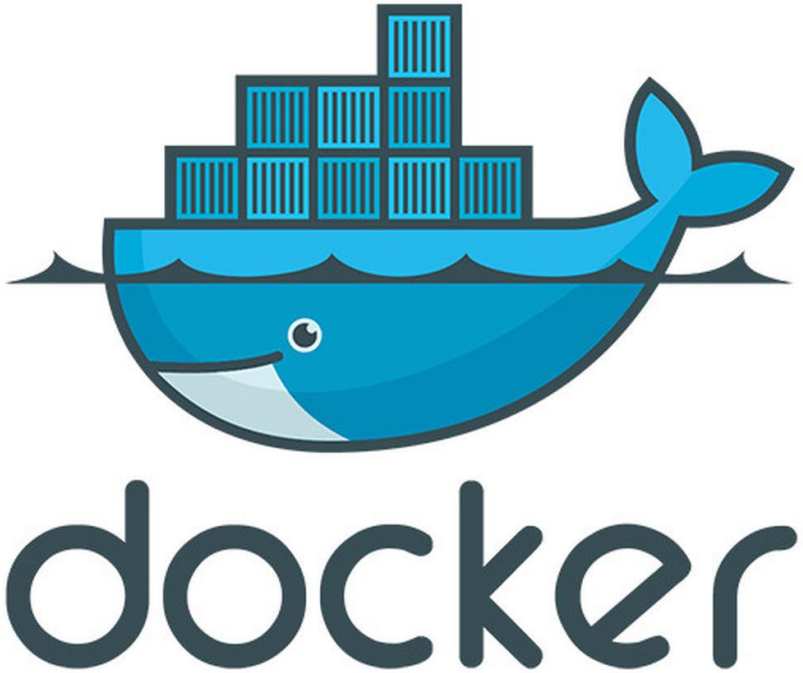
UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

SESAR LAB

What is  ?

Reference: Dr. Corrado Mio

What is Docker?

There are several definitions of **Docker**, but one of them can be based on the following problem.

We consider to install a software inside some *physical machine*. We have to

1. download the software, for example **Neo4J**
2. install **Neo4J** using the *installer*
3. configure **Neo4J** specifying
    1. the port to use (the default is 7474)
    2. the directory where databases are saved
    3. the user root and password
    4. the first time the default values can be enough
4. start the **Neo4J** daemon

At the *end*, we have a running instance of **Neo4J**

What is Docker?

Now, we suppose that we need *another* instance of **Neo4J**, for example because we need to compare the behavior of different versions.

There are several problems to resolve:

1. it is not possible to have, in the same computer, two **Neo4J** listen on the same port (7474). We need to change the port.

2. there is **not only** one listen port to configure, but **two**: one for the HTTP protocol (7474) and the other for the **bolt** protocol (7687)

3. because **Neo4J** is a Java application, it is possible that each **Neo4J** version needs a specific Java version. For example **Neo4J v3.5** uses Java 8 and **Neo4j v4.0** uses Java 11.

Another important problem is the:

- each application has specific configurations and specific mechanisms to configure them. **Neo4J** uses a file inside the directory *NEO4J_HOME/conf/neo4j.conf*. **Apache Web Server** uses a file inside the directory APACHE_HOME/conf/httpd.conf, **but** the file syntax is totally different, …

What is Docker?

Next problem, we suppose that there is a *computer farm*, and we request to the *system administrator*, responsible on the software installed, to install some little clusters of **Neo4J** v3.0, v3.5 and v4.0.

The *system administrator* must know that each version of **Neo4J** need a specific Java version, how to install and to configure it. And how to install and configure Java. It have to resolve port conflicts, configuration conflicts, …

But these are just two of hundreds of software that he must know how to install and configure.

What is Docker?

Instead, we consider the concepts that a *system administrator* handles every day:

- cpu, memory
- local/remote/mounted filesystem
- networking: IP/port, NAT, bridge, network segments, …
- (bash) scripts, environment variables
- cluster node, rack, …

What is Docker?

If the *installation problems* can be converted in terms of *standard hardware* and *system concepts* it is possible to simplify the software installation and configuration.

Docker offers exactly this.

Each **Docker** instance is composed by:

1. (configurable) standard hardware: cpu, memory, storage, network card. Eventually, graphic card, cdrom, or other pci/usb devices …

2. a minimal *normal* operating system with complete support for the hardware, but, in general, without support for GUI, or other client services (e-mail, video/audio players, …).
   But it is possible to install them.

In this sense, a **Docker** instance can be considered as (*light*) version of a **VMware** / **Virtualbox** *virtual machine.*

What is Docker?

However, the **Docker** infrastructure permits a **Docker** instance to communicate with the *physical hardware*. The two most important mechanisms are

1. *mounted filesystem*: the instance *mounts* as *local* directory a *physical* directory (a classic Unix/Linux method to mount CDROM, FTP/Webdav servers, NTFS external disks, …)

2. *network mapping*: the *local* IP/ports are mapped to a *physical* IP/ports

What is Docker?

Inside the **Docker** instance, a **Neo4J** *expert* can install **Neo4J** and all necessary dependencies. Then, to configure it to serve some specific task (standalone, in cluster, …)

Because each instance can be considered as a *standalone* computer, there are no port conflicts, directory conflicts, …

If it is necessary to change some configuration parameter, this **must** be done

- using *bash scripts* and/or *environment variables*.

If it is necessary to export the logs of **Neo4J** outside the **Docker** instances, this can be done

- using *mounted filesystems*.

If some **Neo4J** instance needs more memory or computation power, it can be done

- changing the *hardware configuration* of the **Docker** instance.

If it is necessary to install multiple instance of the same **Neo4J** installation, it can be done

- copying multiple times the **Docker** instance and mapping *internal* IP/ports to *external* IP/ports.

What is Docker?

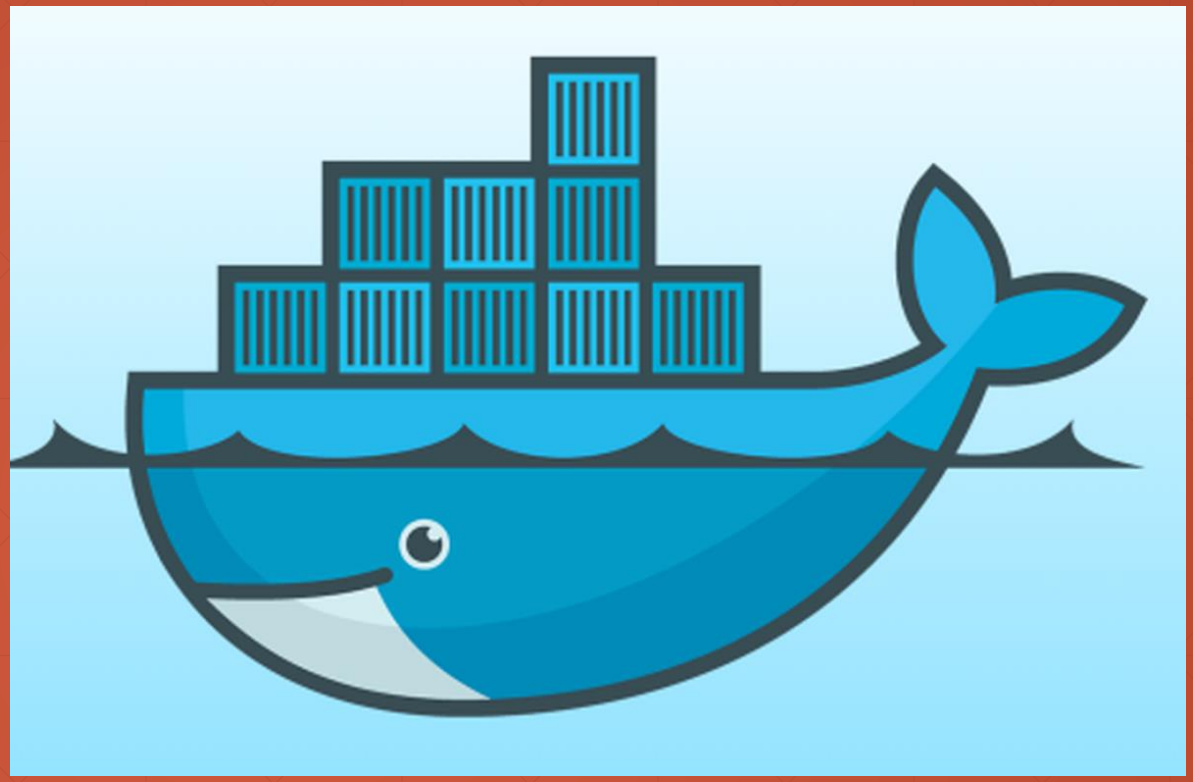
In this way,

- the **Neo4J** *expert* is responsible to install and configure a *single* **Neo4J** *instance*, and
- the *system administrator* can install and configure **Neo4J** *instances* using *only* system concepts.

But this is not all

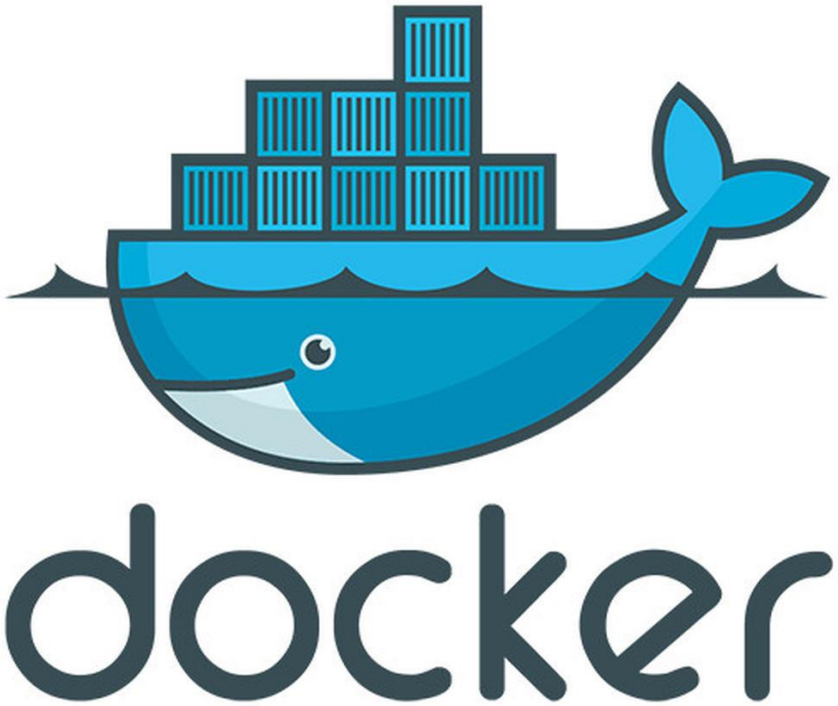
Docker is a *light* version of a *virtual machine*: **why?**

Thanks



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

SESAR LAB



Reference: Dr. Corrado Mio

Prerequisites

In these exercitations we will use a Virtual Machine

- Ubuntu LTS (18.04, 20.04,…):
  http://releases.ubuntu.com/
  http://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso

installed using

- VMware Player/Workstation:
  https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

or

- Virtual Box:
  https://www.virtualbox.org/

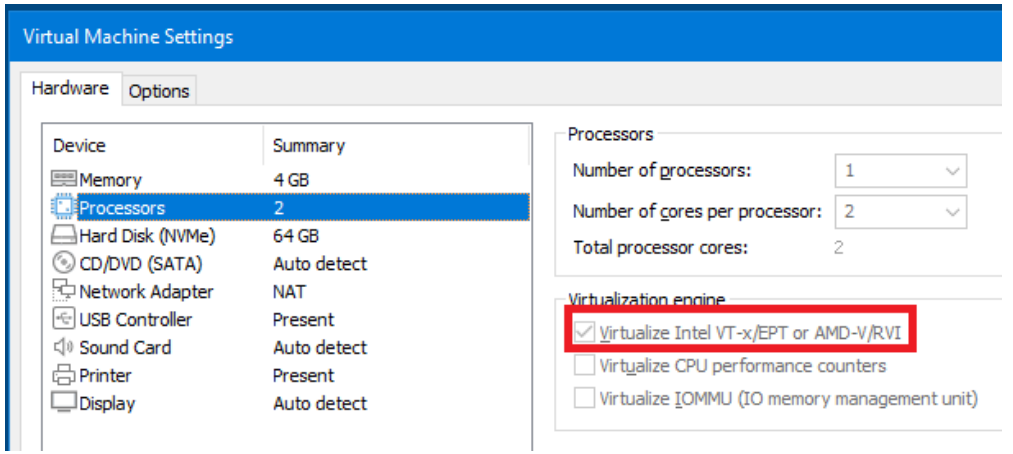
Installation

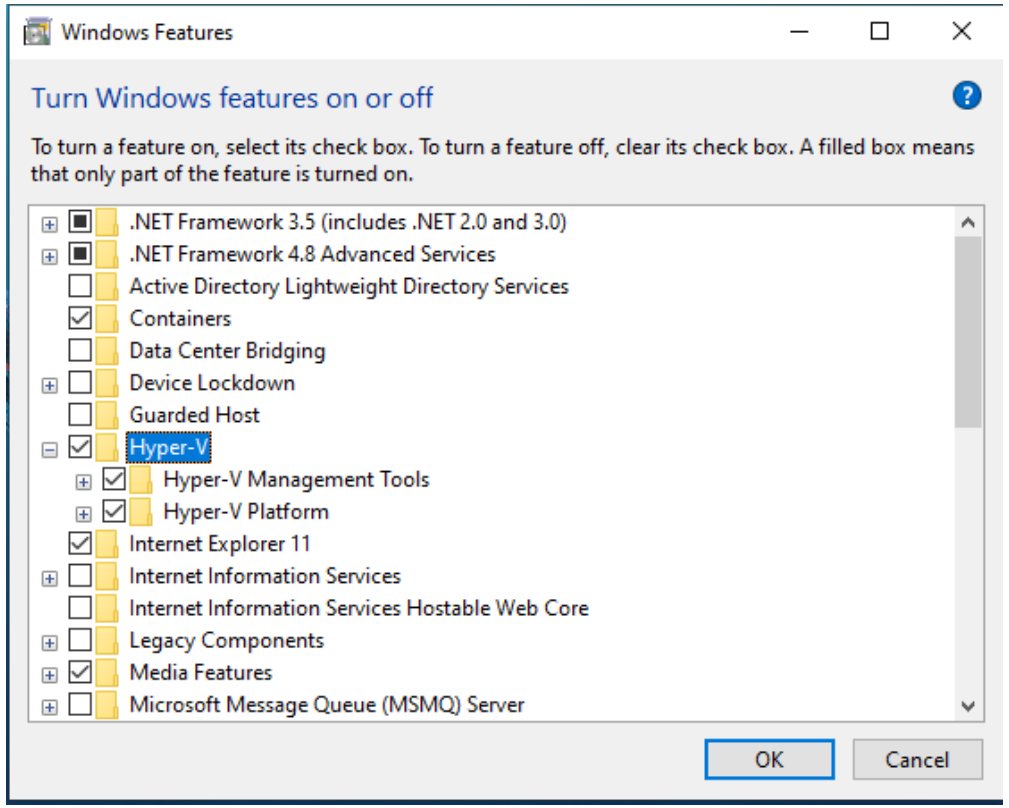
Docker can be installed on **Linux**, **Windows** and **Mac**

1. **Linux**: the installation is very simple and direct
2. **Windows**: it is necessary to enable *Hyper-V*
3. **Mac**: …

In Windows, the **problem** is that *Hyper-V* is **incompatible** with other virtualization software (**Vmware**, **VirtualBox**, ..).

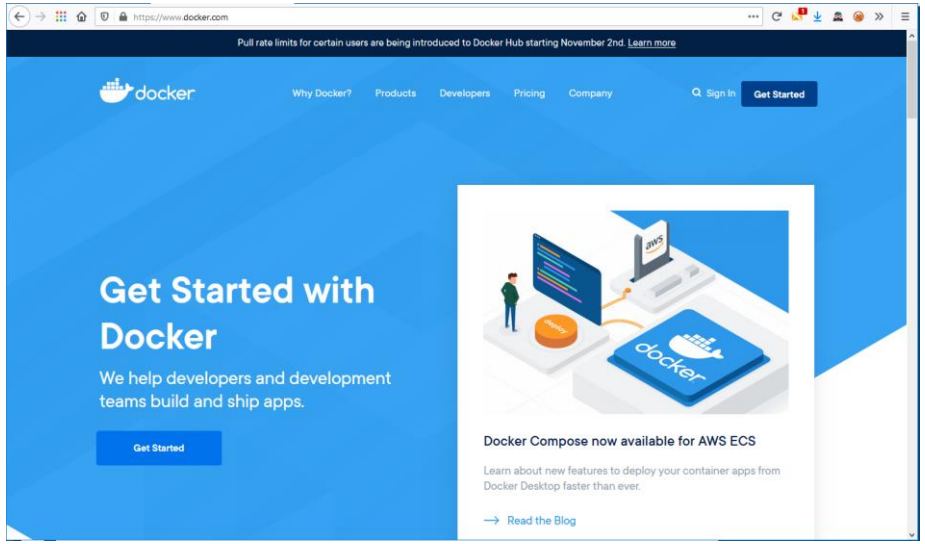
But *Hyper-V* can be **enabled inside** the virtual machine!

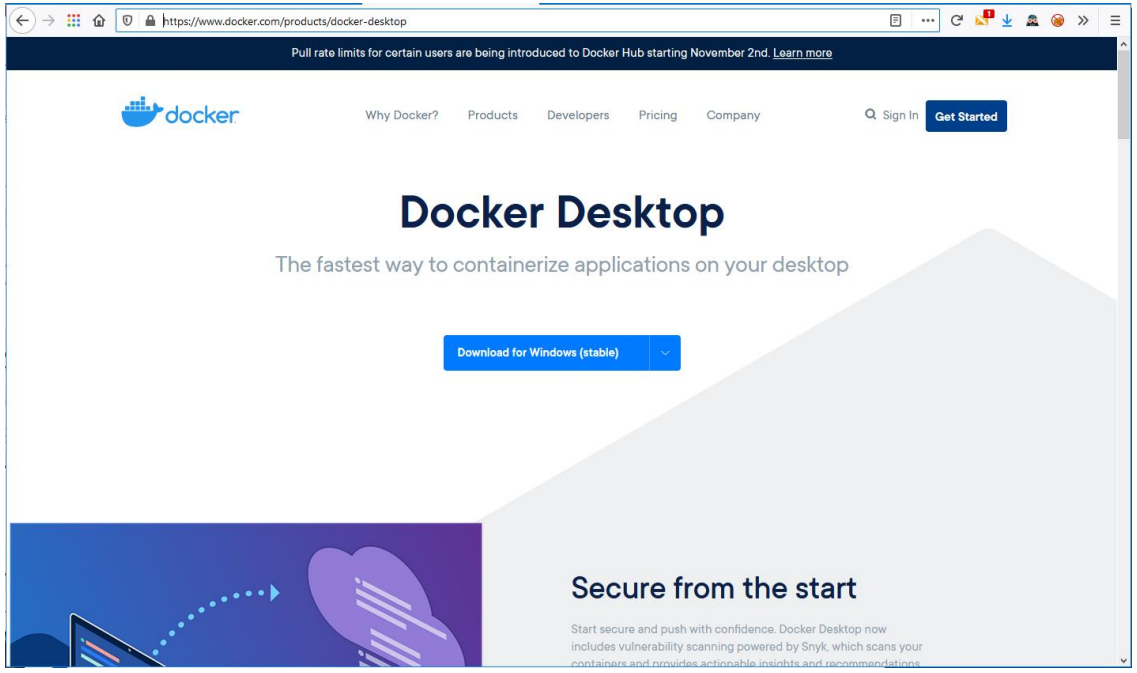


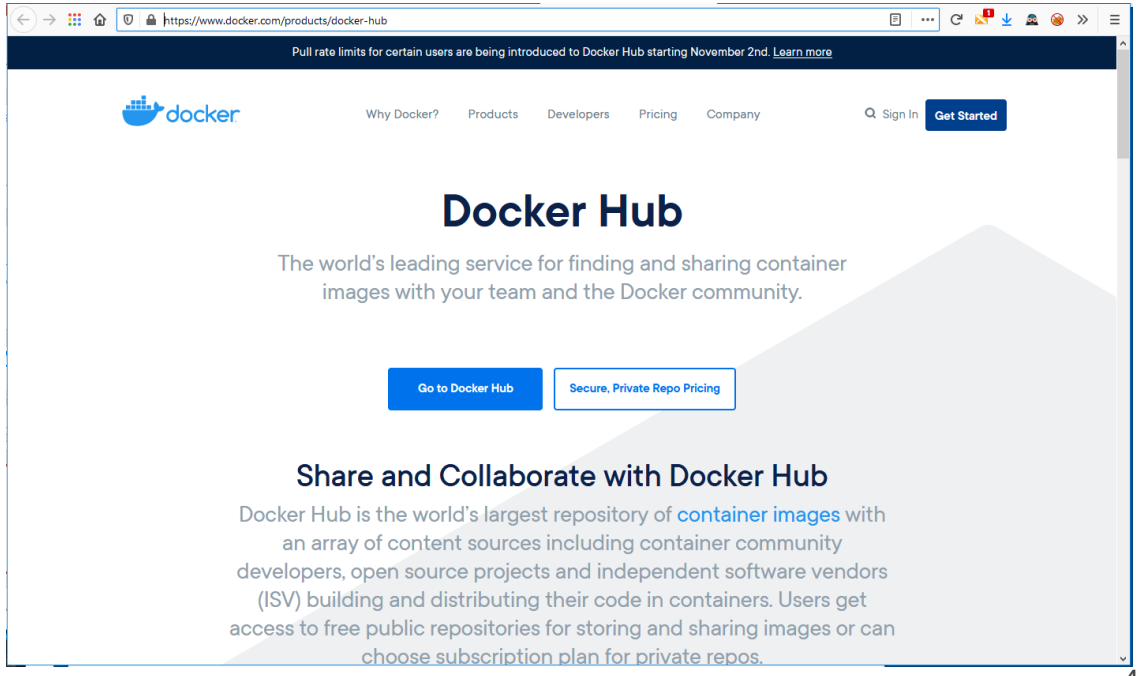


Docker Web sites

- https://www.docker.com
- https://www.docker.com/products/docker-desktop
- https://www.docker.com/products/docker-hub



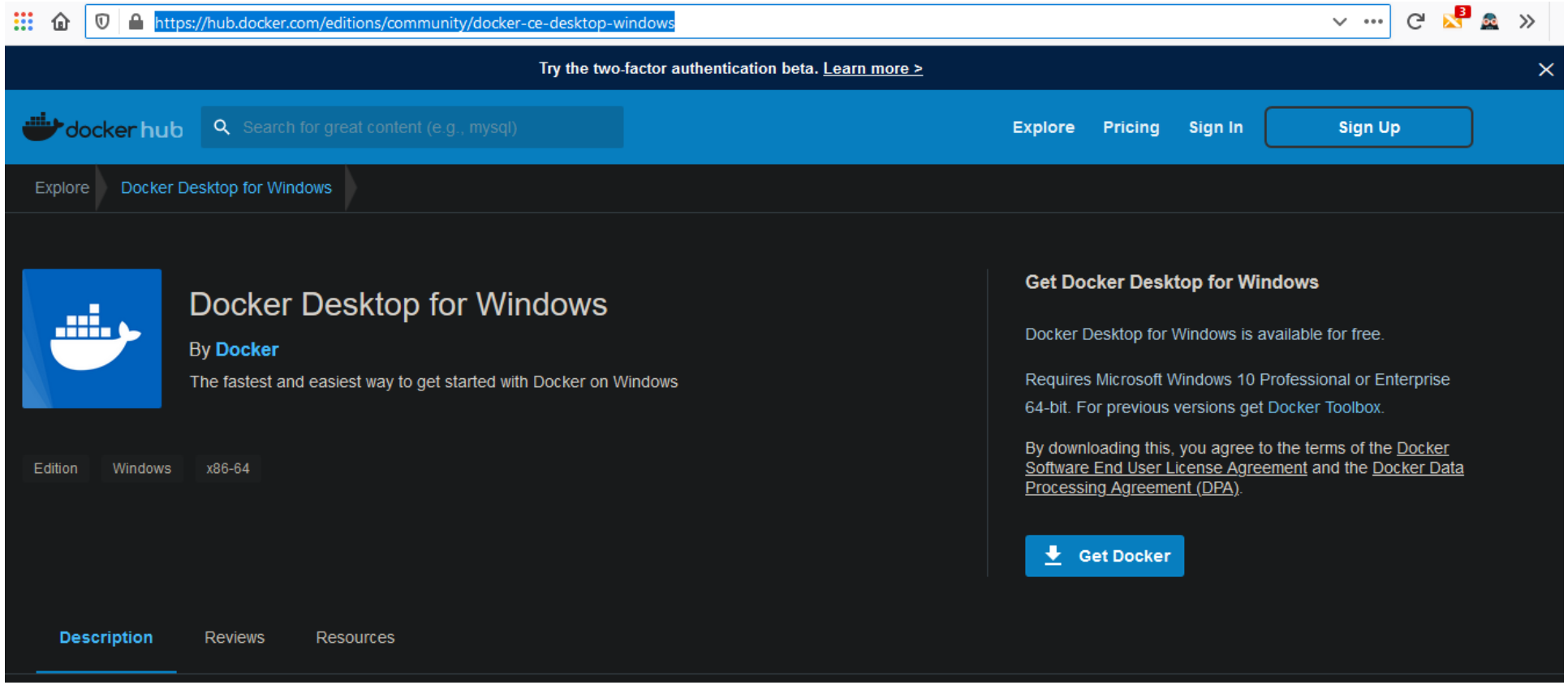




Installation: Windows

Docker Desktop for Windows is available for Windows 10 Pro/Enterprise, 64bit

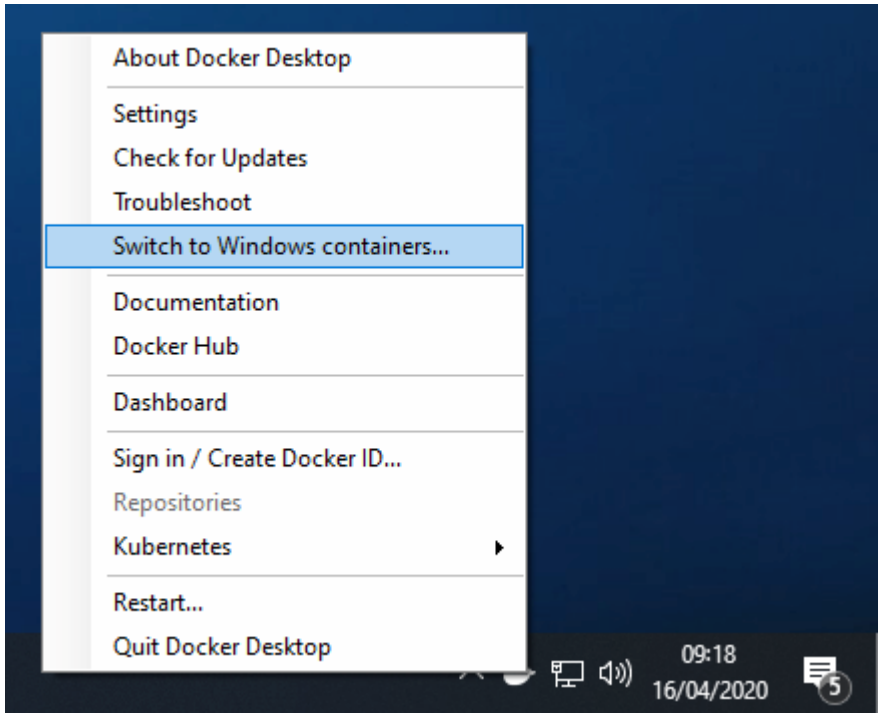
https://desktop.docker.com/win/stable/Docker%20Desktop%20Installer.exe



Docker for Windows

It supports **two** *container's* **types**:

1. **Linux** *containers* (default)
2. **Windows** *containers*

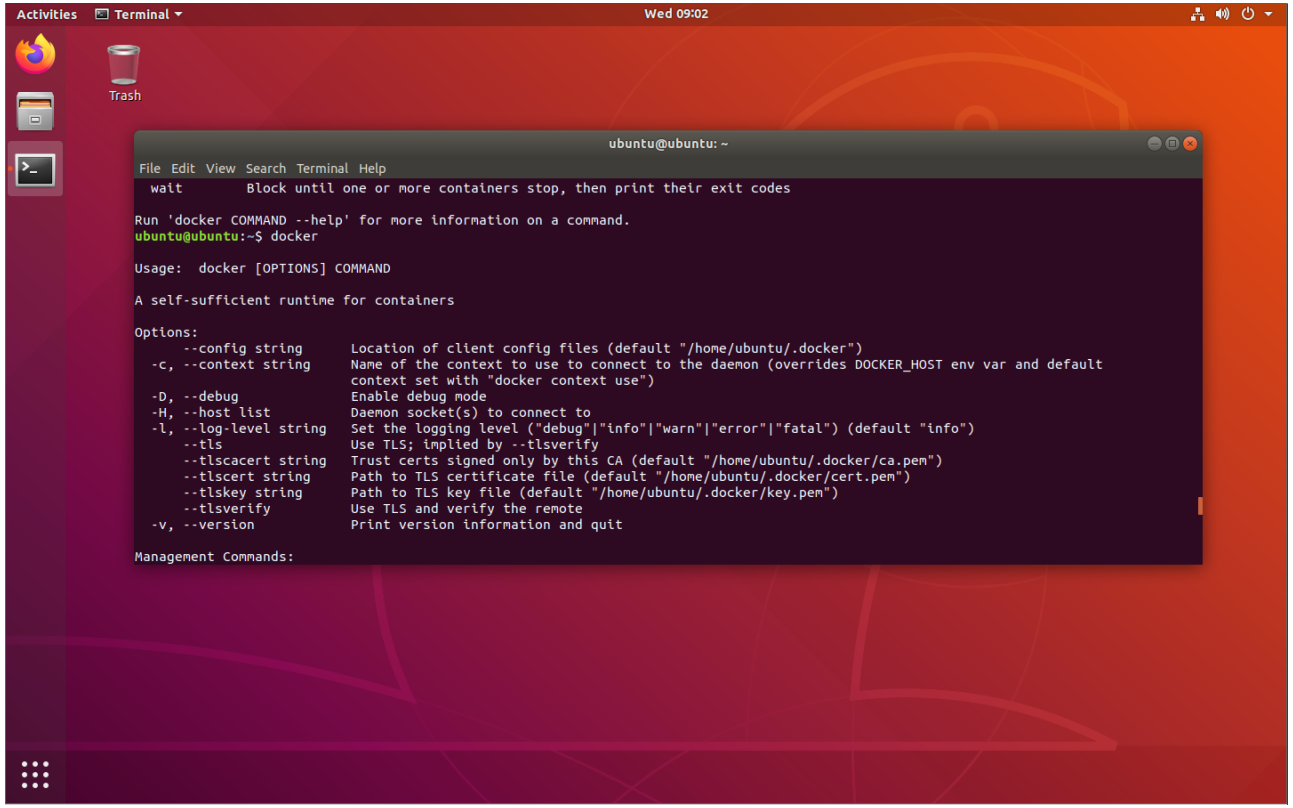


It is possible to change from one container type to the other using item specified in the image.

Note: what is a *container* will be described in the following slides.

Installation: Linux

We start with a clean installation of Ubuntu LTS (18.04, 20.04, …).



Some simple tutorials

https://docs.docker.com/install/linux/docker-ce/ubuntu/

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04

Installation: Linux - commands

Based on: https://docs.docker.com/install/linux/docker-ce/ubuntu/

Commands
> sudo apt-get remove docker docker-engine docker.io containerd runc
> sudo apt-get update
> sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
> curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
> sudo apt-key fingerprint 0EBFCD88
> sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
> sudo apt-get update
> sudo apt-get install docker-ce docker-ce-cli containerd.io
> sudo docker run hello-world

Check the installation

.

Example (Linux)
`> docker run hello-world`

https://hub.docker.com/_/microsoft-windows

Example (Windows)
`> docker run mcr.microsoft.com/windows:1903`

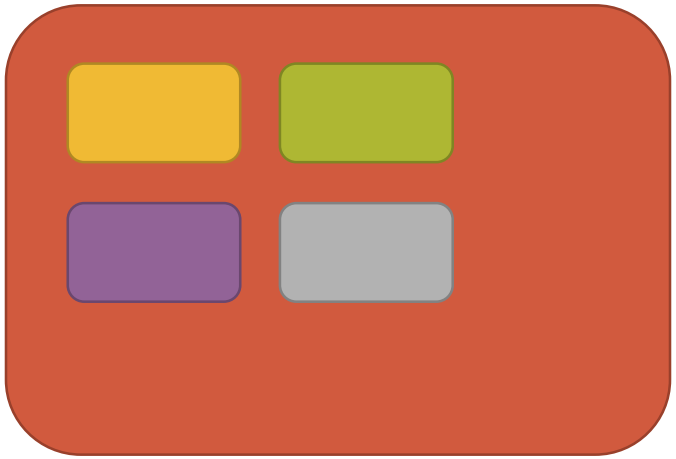
Virtual Machine vs Docker

Virtual Machines

- Host: SO + Vmware + …
  - Guest: SO + applications + …

Docker images

- **Host**: Linux + *cgroups* + *namespaces* + …
  - Guest1: Host + applications + …
    - Guest2: Guest1 +…
      - Guest3: guest2 + …



Virtual Machines vs Docker/2

Each **virtual machine** needs:

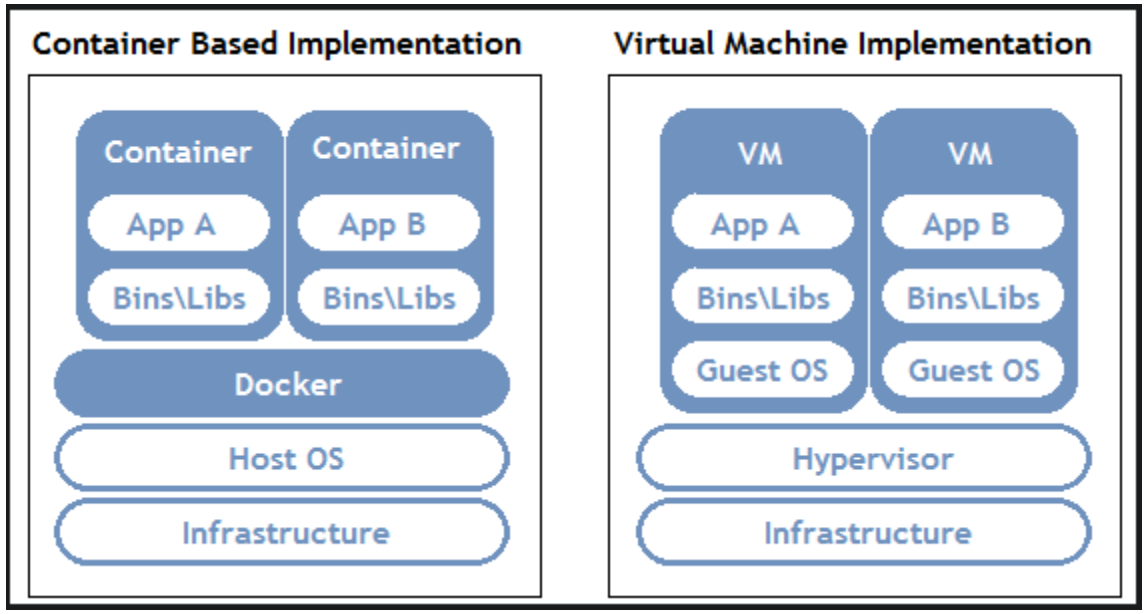
- Virtualized hardware (CPU, ram, disk, video/network cards, USB, …)
- An operating system (it can be different from the host OS)
- The applications

Each **Docker image** can use the **real** hardware under Kernel Linux control:

- **Cgroups**: it controls CPU, ram, block devices, network cards, …
- **Namespaces**: it controls processes, users, filesystem, networking, …
- **Union Filesystem**: it permit to create a new **Docker** image using *previous* created ones.
- It is a *controlled* version of the real hardware, where the user can **install** his software.

Virtual Machines vs Docker/3

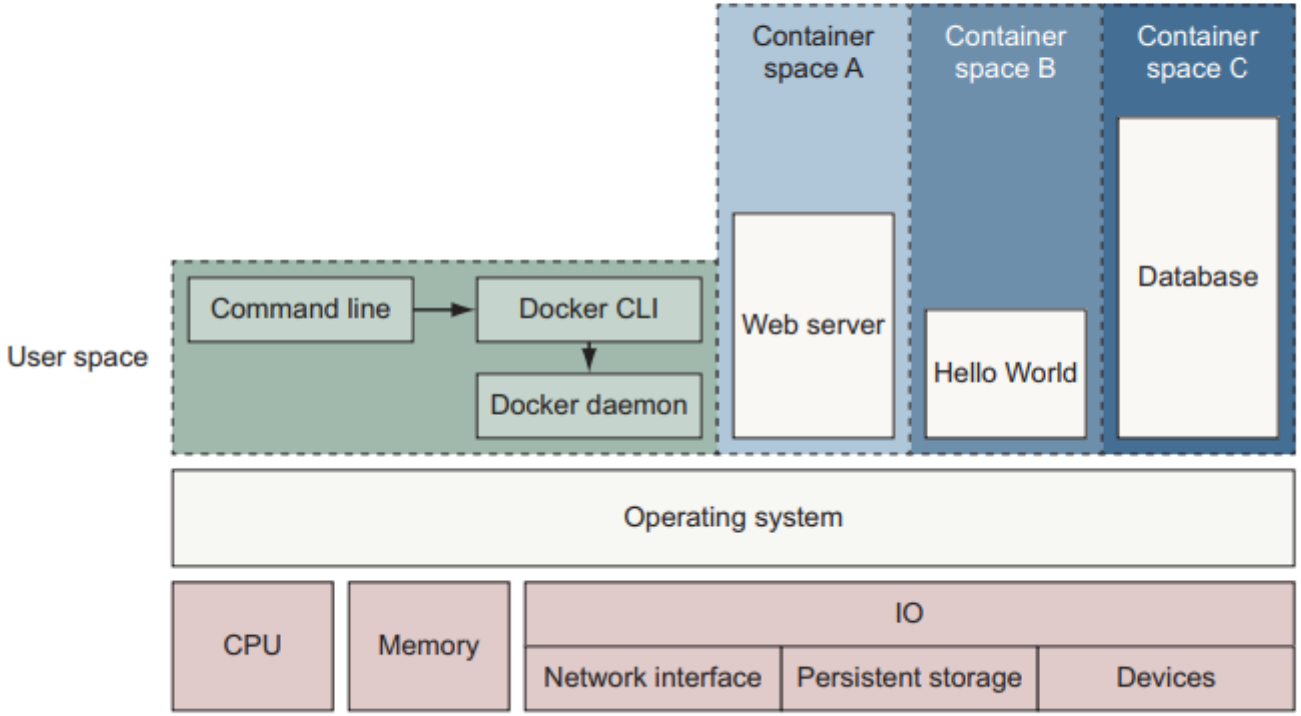
.



Virtual Machines vs Docker/4

.

Docker running three containers on a Linux system



What can be a Docker object

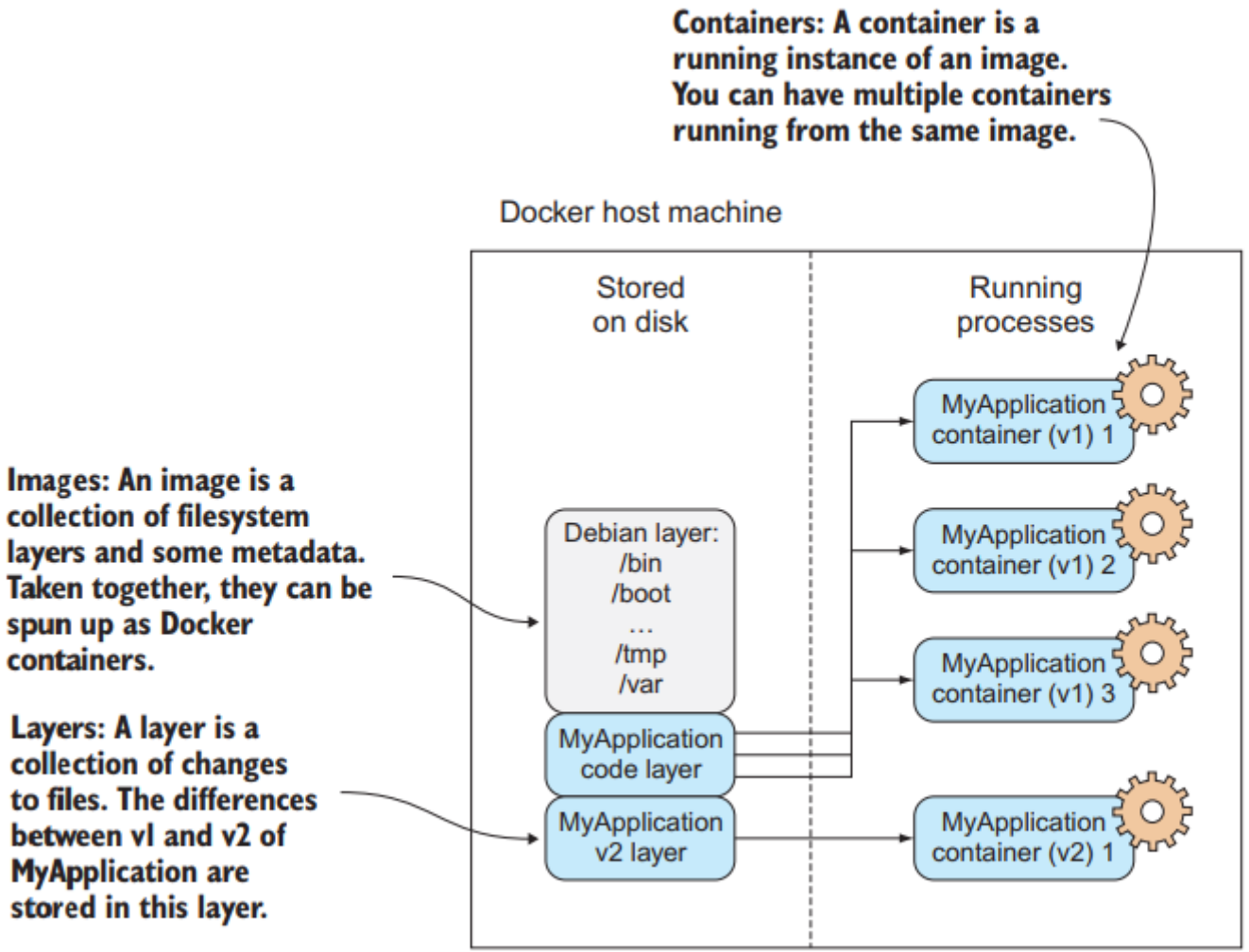
Docker is lite but not so lite!

- Application server (Tomcat, …)
- A DBMS (Oracle, Mysql, …)
- A complex application (Mathematica, Matlab, …)
- …

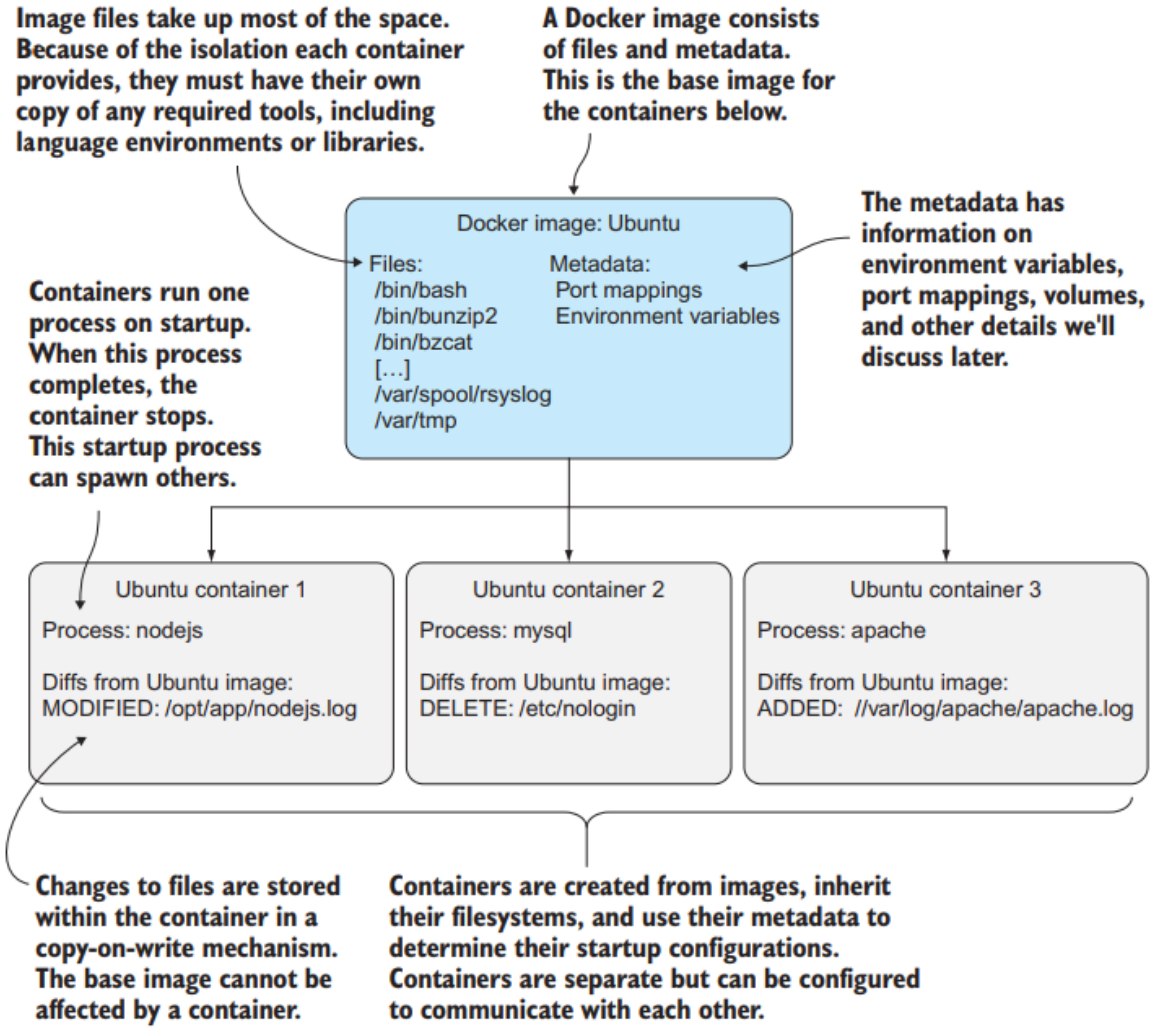
It is not necessary **to install** the application, it is enough **to copy/download** the Docker files (that contain the application) in the Docker infrastructure, to **configure** the **Docker** object (if necessary) and **to execute** it.

In this case, the configuration is more simple than a application configuration.

Key concepts: layers, images, containers



Key concepts: layers



Key concepts: images

A **Docker image** is the main component of the Docker infrastructure.

It is a stack of layers, created using a **Dockerfile**.

Example:

- `FROM openjdk:8-alpine`: previous layer
- Rest of file: the new layer is defined as **changes** respect the previous layer

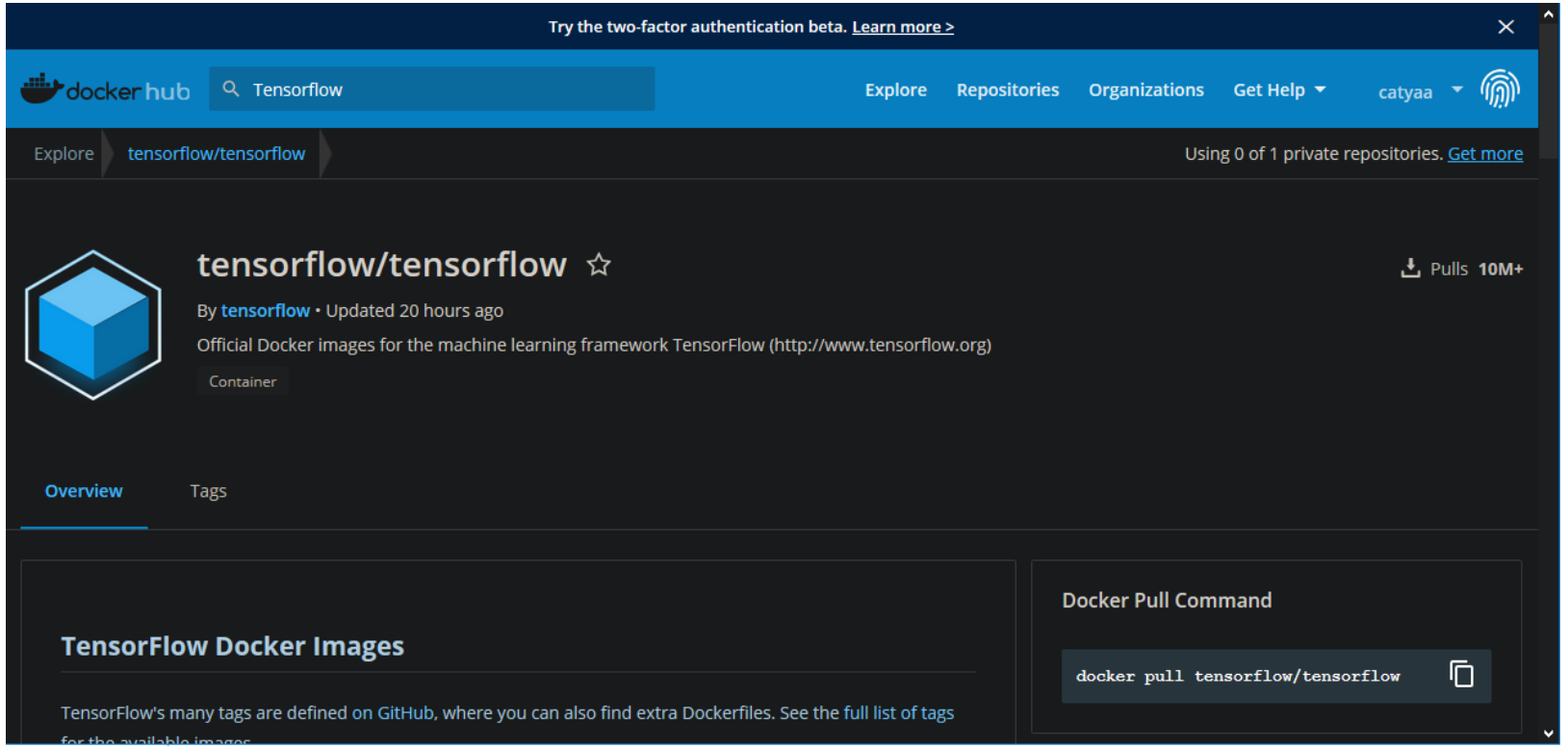
The **previous** layers are **read-only**, the changes are available only on the **current** layer.

```
FROM openjdk:8-alpine

ARG spark_jars=jars
ARG img_path=kubernetes/dockerfiles
ARG k8s_tests=kubernetes/tests

RUN set -ex && \
    apk upgrade --no-cache && \
    ln -s /lib /lib64 && \
    apk add --no-cache bash tini libc6-compat linux-pam nss && \
    mkdir -p /opt/spark && \
    mkdir -p /opt/spark/work-dir && \
    touch /opt/spark/RELEASE && \
    rm /bin/sh && \
    ln -sv /bin/bash /bin/sh && \
    echo "auth required pam_wheel.so use_uid" >> /etc/pam.d/su && \
    chgrp root /etc/passwd && chmod ug+rw /etc/passwd

COPY ${spark_jars} /opt/spark/jars
COPY bin /opt/spark/bin
COPY sbin /opt/spark/sbin
COPY ${img_path}/spark/entrypoint.sh /opt/
COPY examples /opt/spark/examples
COPY ${k8s_tests} /opt/spark/tests
COPY data /opt/spark/data

ENV SPARK_HOME /opt/spark

WORKDIR /opt/spark/work-dir

ENTRYPOINT [ "/opt/entrypoint.sh" ]
```

Key concepts: containers

A **Docker container** is a *running* instance of a **Docker image**.

It is possible to create a Docker image from scratch, but an alternative approach is to search the required image in **Docker Hub**:



Key concepts: containers/2

Steps to launch an image:

1. Search the image in **Docker Hub**. For example:
   “`tensorflow`”
2. Download (**pull**) the image in local
   “`docker pull tensorflow/tensorflow:2.3.1-jupyter`”
3. Execute (**run**) the downloaded image
   “`docker run -it --rm -p 8888:8888 tensorflow/tensorflow:2.3.1-jupyter`”

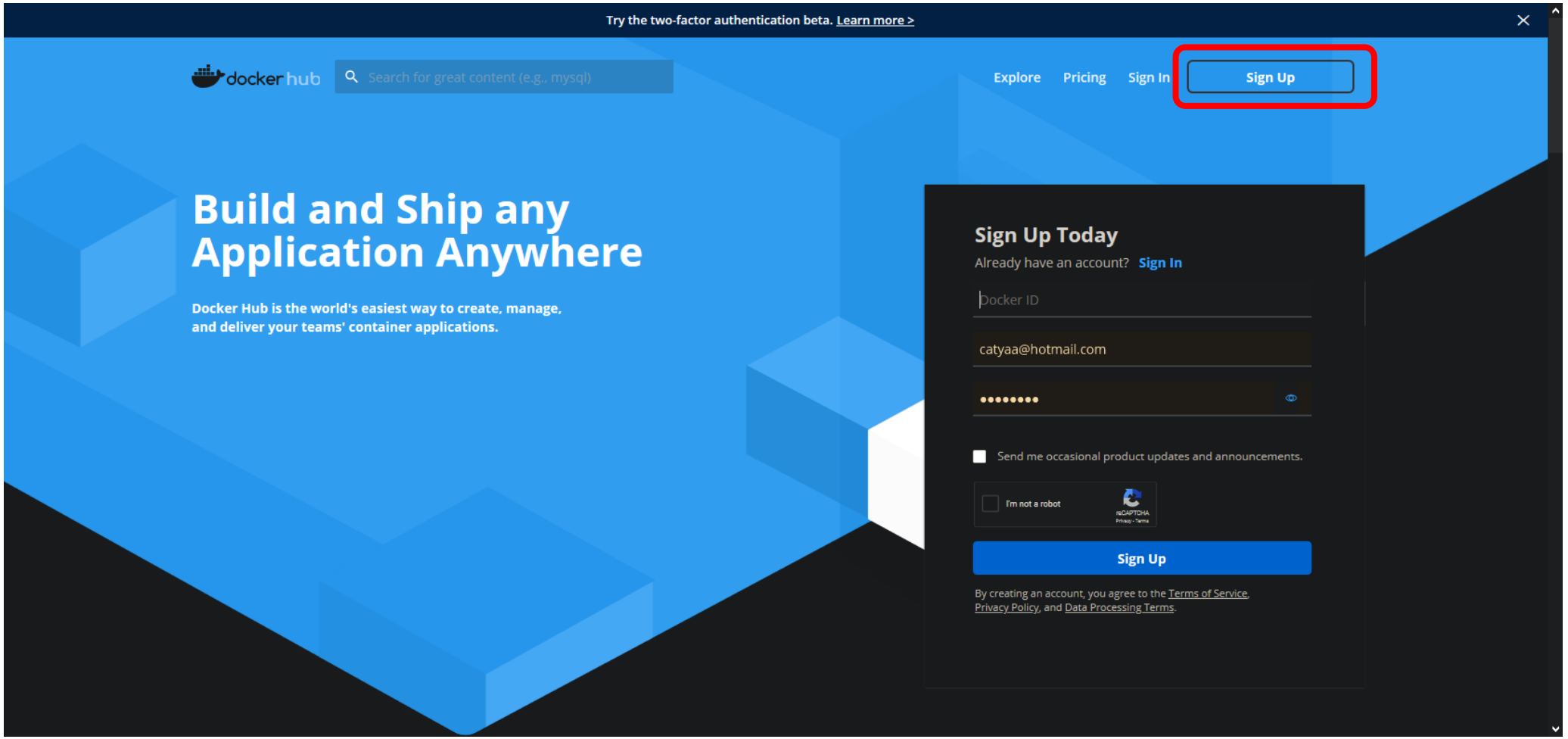
The structure of an image’s name is:

`[http://<website>:dockerhub/][<company:docker>/]<image-name>:<tag:latest>`

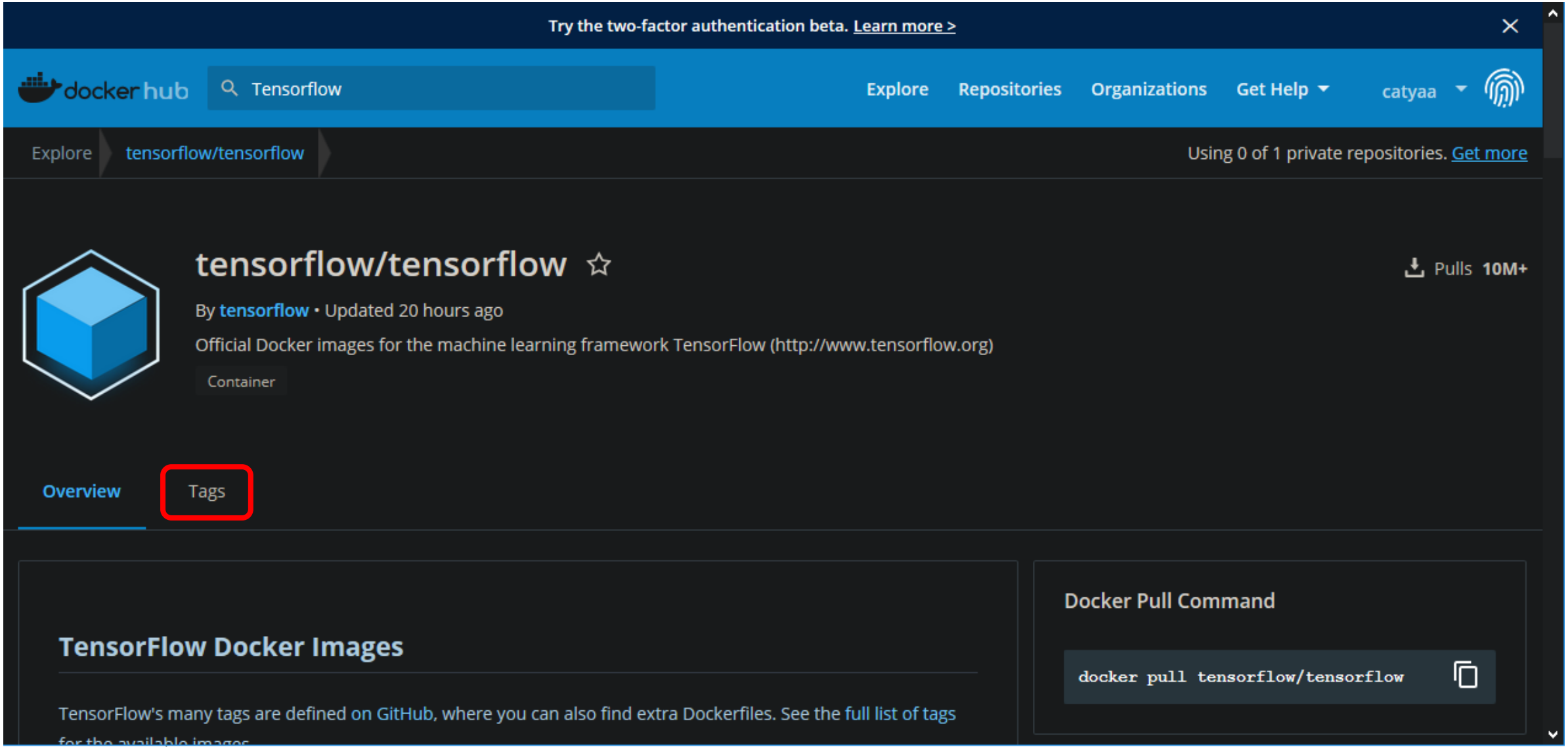
.

Access to Docker Hub

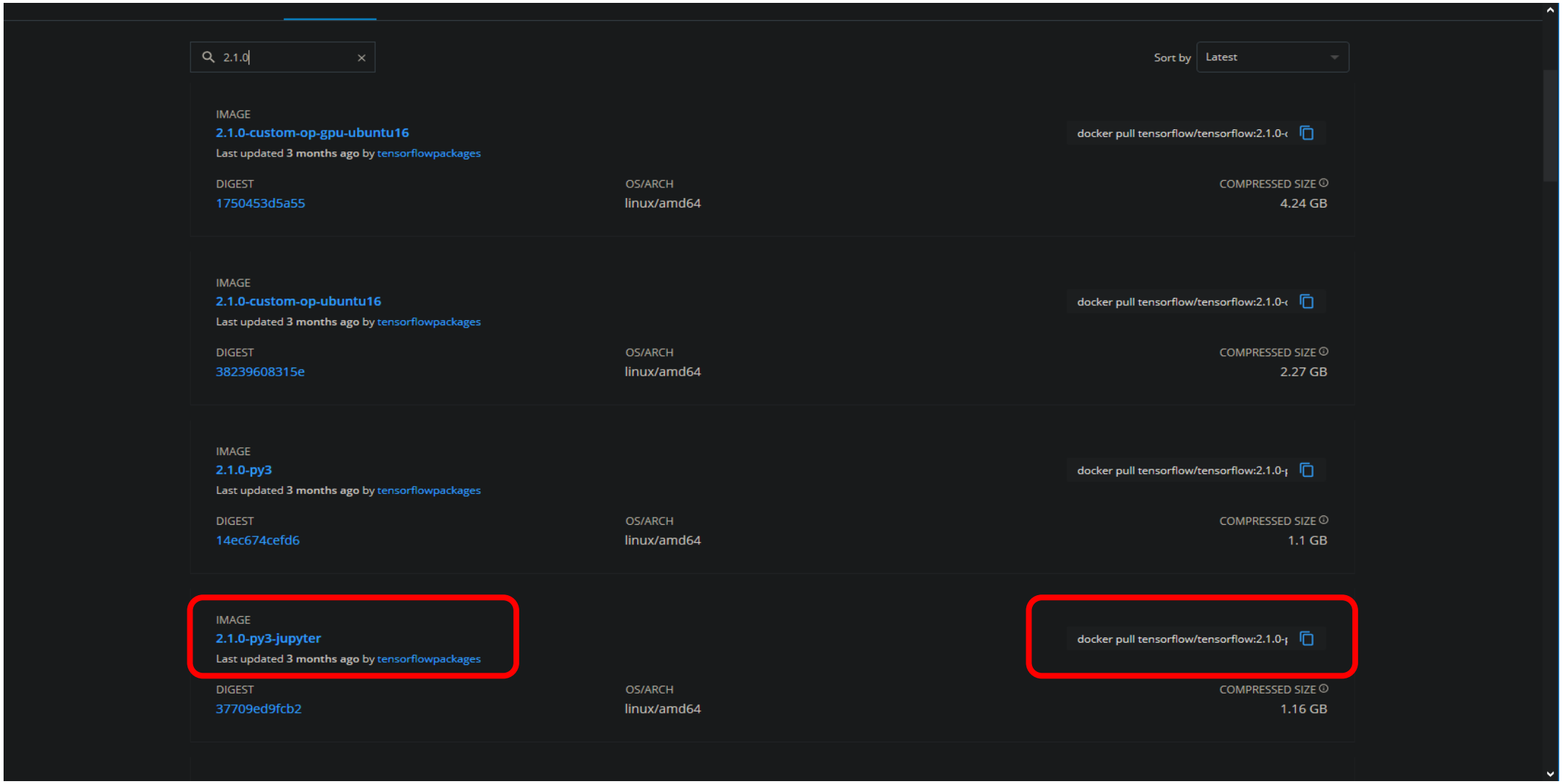
To access to **Docker Hub** you have to register.



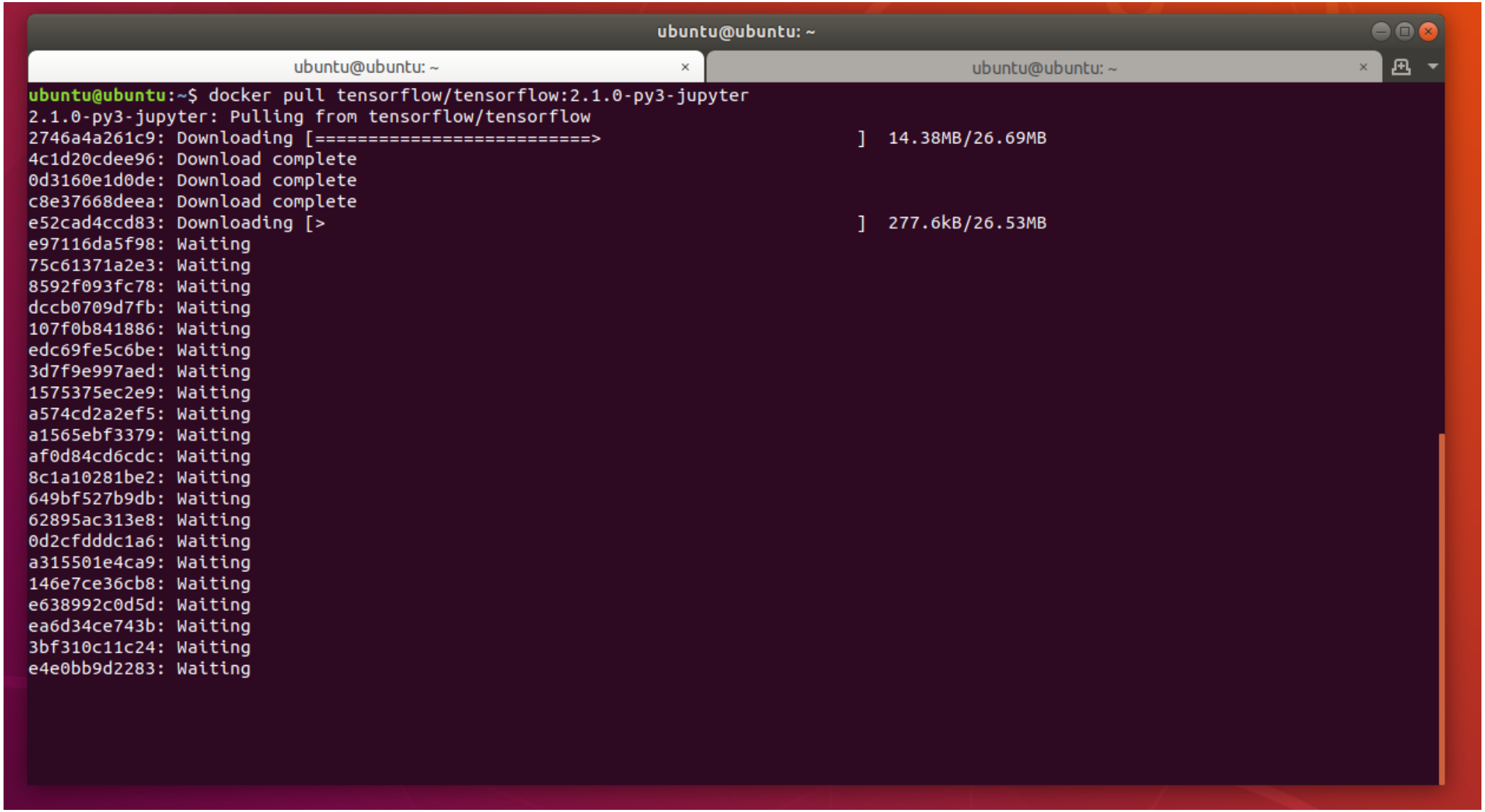
Search an image



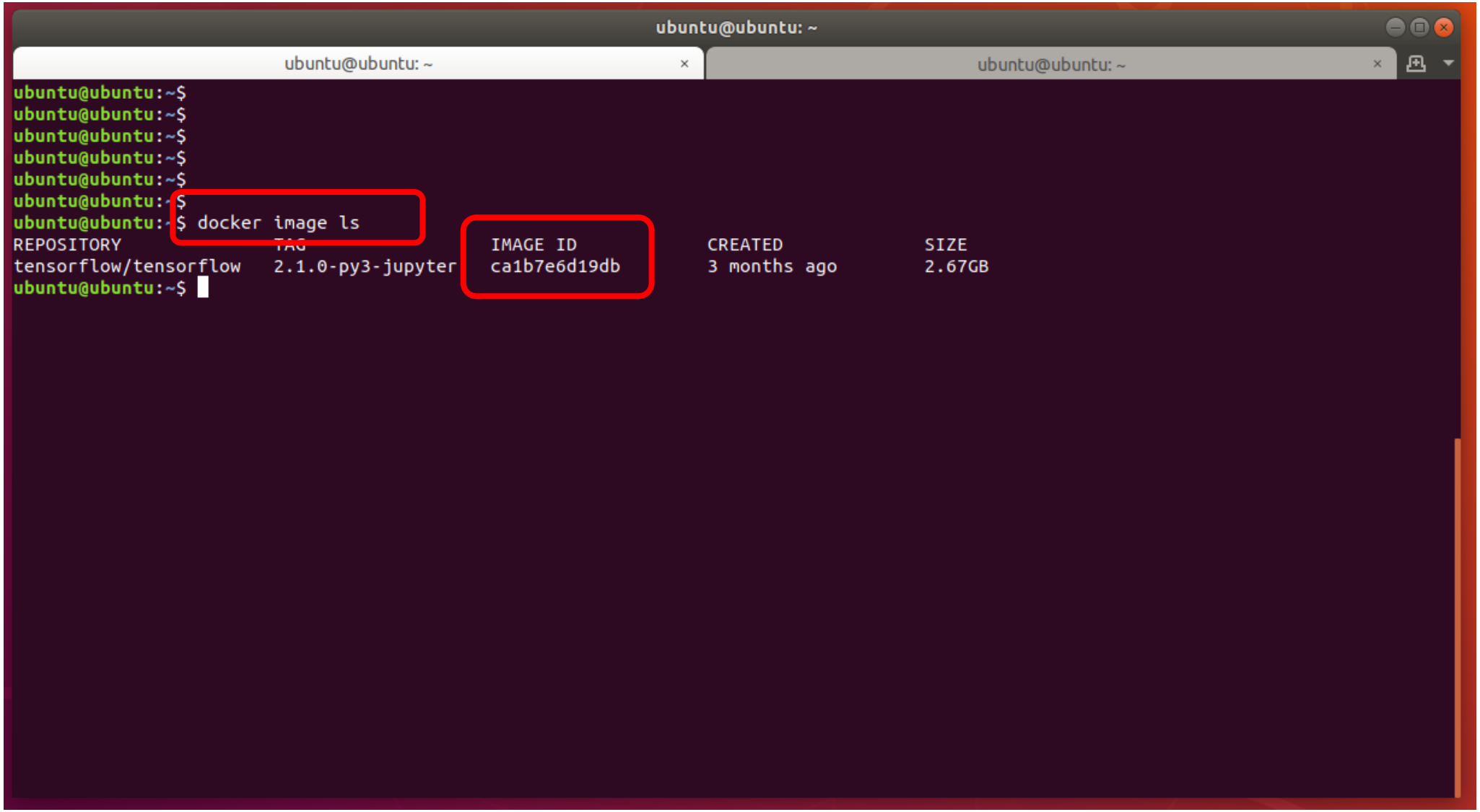
Select a Tag


2.1.0
Sort by
Latest
IMAGE
2.1.0-custom-op-gpu-ubuntu16
Last updated 3 months ago by tensorflowpackages
docker pull tensorflow/tensorflow:2.1.0-
DIGEST
1750453d5a55
OS/ARCH
linux/amd64
COMPRESSED SIZE
4.24 GB
IMAGE
2.1.0-custom-op-ubuntu16
Last updated 3 months ago by tensorflowpackages
docker pull tensorflow/tensorflow:2.1.0-
DIGEST
38239608315e
OS/ARCH
linux/amd64
COMPRESSED SIZE
2.27 GB
IMAGE
2.1.0-py3
Last updated 3 months ago by tensorflowpackages
docker pull tensorflow/tensorflow:2.1.0-
DIGEST
14ec674cefd6
OS/ARCH
linux/amd64
COMPRESSED SIZE
1.1 GB
IMAGE
2.1.0-py3-jupyter
Last updated 3 months ago by tensorflowpackages
docker pull tensorflow/tensorflow:2.1.0-
DIGEST
37709ed9fcb2
OS/ARCH
linux/amd64
COMPRESSED SIZE
1.16 GB

Download an image



Find the image ID



Executing an image

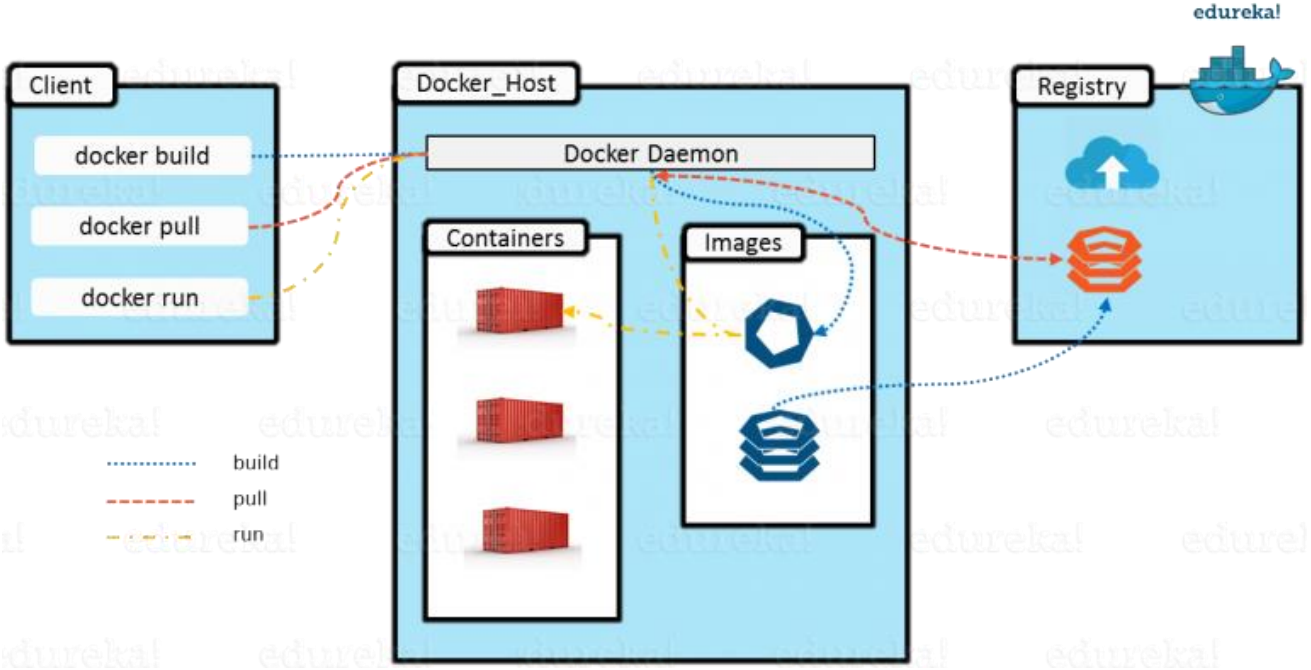
```
ubuntu@ubuntu:~$ docker run -it --rm -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.1.0-py3-jupyter

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...

[I 13:23:19.130 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
jupyter_http_over_ws extension initialized. Listening on /http_over_websocket
[I 13:23:19.708 NotebookApp] Serving notebooks from local directory: /tf
[I 13:23:19.709 NotebookApp] The Jupyter Notebook is running at:
[I 13:23:19.710 NotebookApp] http://8d9bee90baa5:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
[I 13:23:19.711 NotebookApp]  or http://127.0.0.1:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
[I 13:23:19.713 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:23:19.726 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
    Or copy and paste one of these URLs:
        http://8d9bee90baa5:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
     or http://127.0.0.1:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
```

Key concepts vs OOP vs OS

Docker	OOP	Operating System
Image	Class	Program/Executable
Container	Object instance	Process
Layered filesystem	Inheritance: • Class L1 **extends** class O • Class L2 **extends** class L1	Storage

Docker	
Image	A installed program
Container	An image/program in execution. Multiple containers can use the same image, same as multiple processes can be started from the same program
Layered filesystem	**read-write** container's fs *above multiple* **read-only** image's filesystems

Docker Engine



Docker is also the term used describe the software infrastructure used to handle **images** and **containers**:

1. **Docker** client: **docker** command line. Other clients: plugins for IDEs
2. **Docker** RESTful service: **docker daemon**
3. filesystem where the images are saved (used by **docker daemon**)
4. registry from where the images are downloaded (**Docker Hub**)

Docker Engine/2

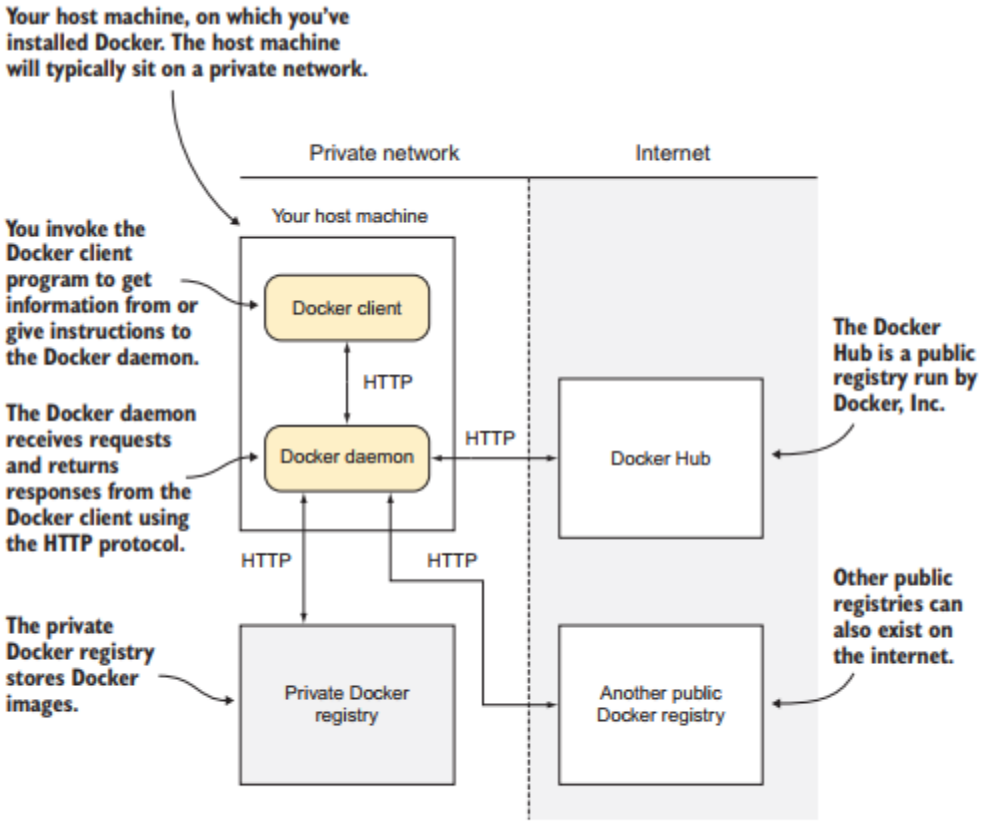


Figure 2.1 Overview of Docker's architecture

(Some) Docker *general* commands

<command>	**docker <command> <args…>**
`ps`	List containers (running, stopped, …)
`attach`	Attach local std input, output, error streams to a running container
`cp`	Copy files/folders between a container and the local filesystem
`exec`	Run a command in a running container
`images`	List images
`kill`	Kill one or more running containers
`rm`	Remove one or more containers
`rmi`	Remove one or more images
`run`	Run a command in a new container
`start`	Start one or more stopped containers
`stop`	Stop one or more running containers
`restart`	Restart one or more containers
`top`	Display the running processes of a container

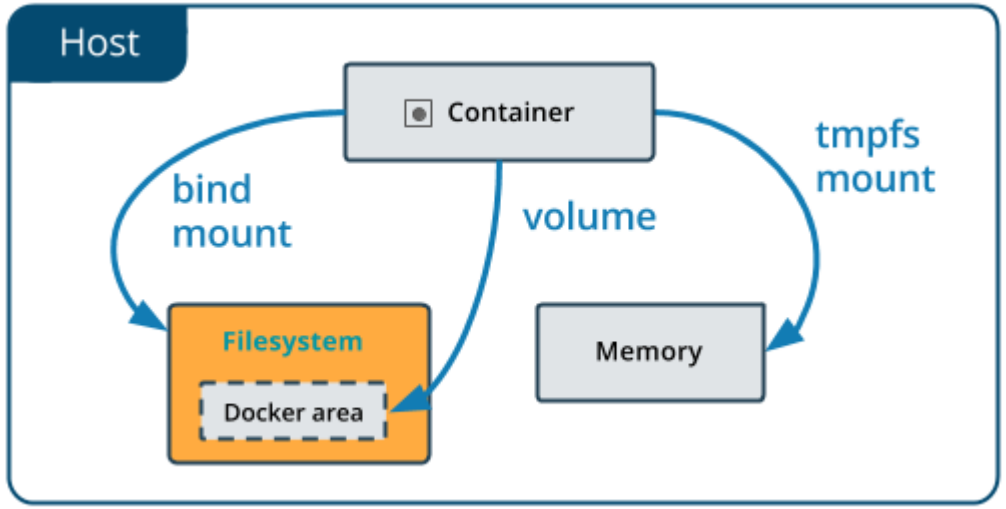
(Some) Docker management commands

<command>	**docker image <command> <args>**
`ls`	List images
`pull`	Pull an image or a repository from a registry
`rm`	Remove one or more images

<command>	**docker container <command> <args>**
`ls`	List containers
`run`	Run a command in a new container
`start`	Start one or more stopped containers
`stop`	Stop one or more running containers
`kill`	Kill one or more running containers
`rm`	Remove one or more containers
`cp`	Copy files/folders between a container and the local filesystem
`exec`	Run a command in a running container

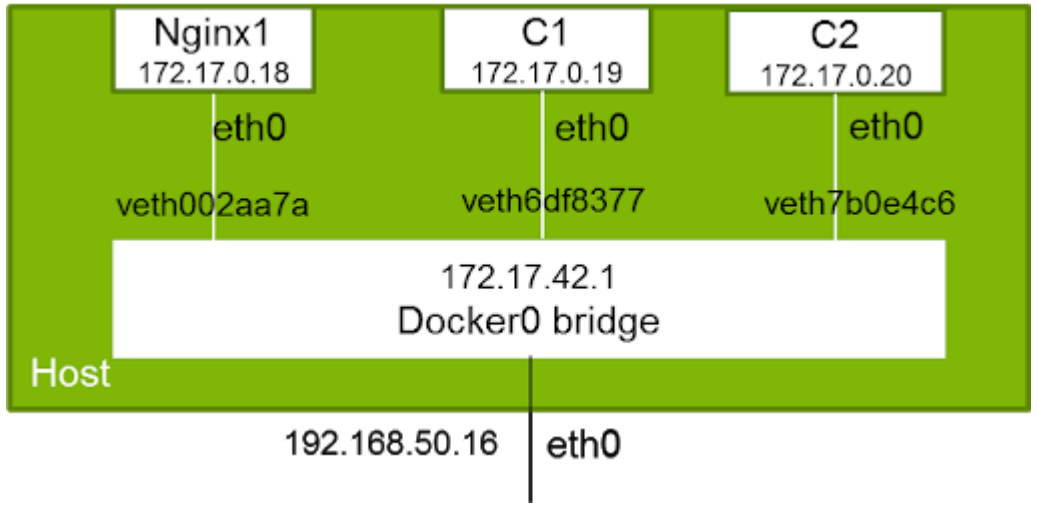
Docker & filesystem

A **container** Docker can **mount** *external filesystems* (*external* from the **container** point of view) as, for example a *local directory* (*local* from the **user** point of view)




```
$ docker run -d \
  -it \
  --name devtest \
  --mount type=bind,source="$(pwd)"/target,target=/app \
  nginx:latest
```

Docker & networking



Each **container** is isolated from the other ones. Multiple containers can use the **same** port to publish they services.

The **Docker bridge** *map* the container ports into **different** host ports. The syntax is:

```
-p <external [ip:]?port>:<internal port>
```

```
$ docker run -d -p 5000:5000 -v $HOME/registry:/var/lib/registry
registry:2
```

(Some) Docker management commands/2

<command>	**docker network <command>**
`ls`	List networks
`create`	Create a network
`connect`	Connect a container to a network
`disconnect`	Disconnect a container from a network
`rm`	Remove one or more networks

<command>	**docker volume <command>**
`ls`	List volumes
`create`	Create a volume
`rm`	Remove one or more volumes

Docker GUI

The **docker** command line is for Linux fan. For Windows fan, there exists several GUI. Some of them are:

1. **Portainer** (https://www.portainer.io/)
2. **Kitematic** (https://kitematic.com/)
3. **Shipyard** (https://github.com/shipyard/shipyard)
4. **DockStation** (https://dockstation.io/)

Kitematic is part of Docker distribution for Windows & Mac **but** in GitHub there exists also the distribution for Ubuntu. The cons is that it has limited features.

Portainer is a better alternative.

Portainer

Portainer is a good alternative to the command line. It is a web application installed (and distributed) as a **Docker** image. It is available in **Docker Hub**.

Commands (Linux)

```
> docker volume create portainer_data
```

```
> docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

Commands (Windows)

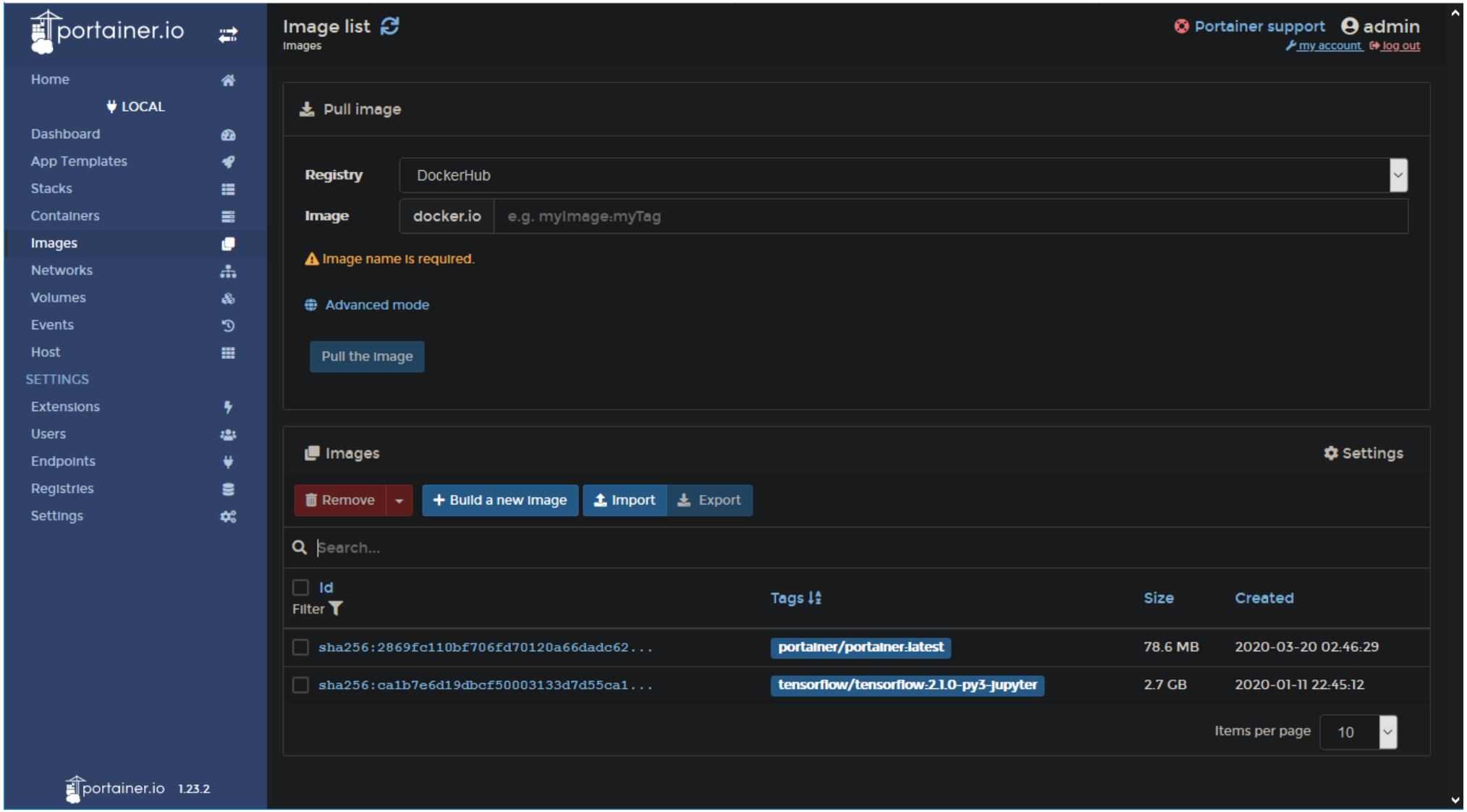
```
> mkdir \var\run
```

```
> docker pull portainer/portainer-ce
```

```
> docker volume create portainer_data
```

```
> docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

Portainer/2


portainer.io
Home
LOCAL
Dashboard
App Templates
Stacks
Containers
Images
Networks
Volumes
Events
Host
SETTINGS
Extensions
Users
Endpoints
Registries
Settings
portainer.io 1.23.2
Image list
Images
Portainer support
admin
my account
log out
Pull image
Registry
DockerHub
Image
docker.io
e.g. myImage:myTag
Image name is required.
Advanced mode
Pull the image
Images
Settings
Remove
+ Build a new image
Import
Export
Search...
Id
Filter
Tags
Size
Created
sha256:2869fc110bf706fd70120a66dadc62...
portainer/portainer:latest
78.6 MB
2020-03-20 02:46:29
sha256:ca1b7e6d19dbcf50003133d7d55ca1...
tensorflow/tensorflow:2.1.0-py3-jupyter
2.7 GB
2020-01-11 22:45:12
Items per page
10

Examples

Tensorflow

Possible versions

1. `tensorflow/tensorflow:1.15.2-py3-jupyter`
2. `tensorflow/tensorflow:2.0.1-py3-jupyter`
3. **`tensorflow/tensorflow:2.1.0-py3-jupyter`**

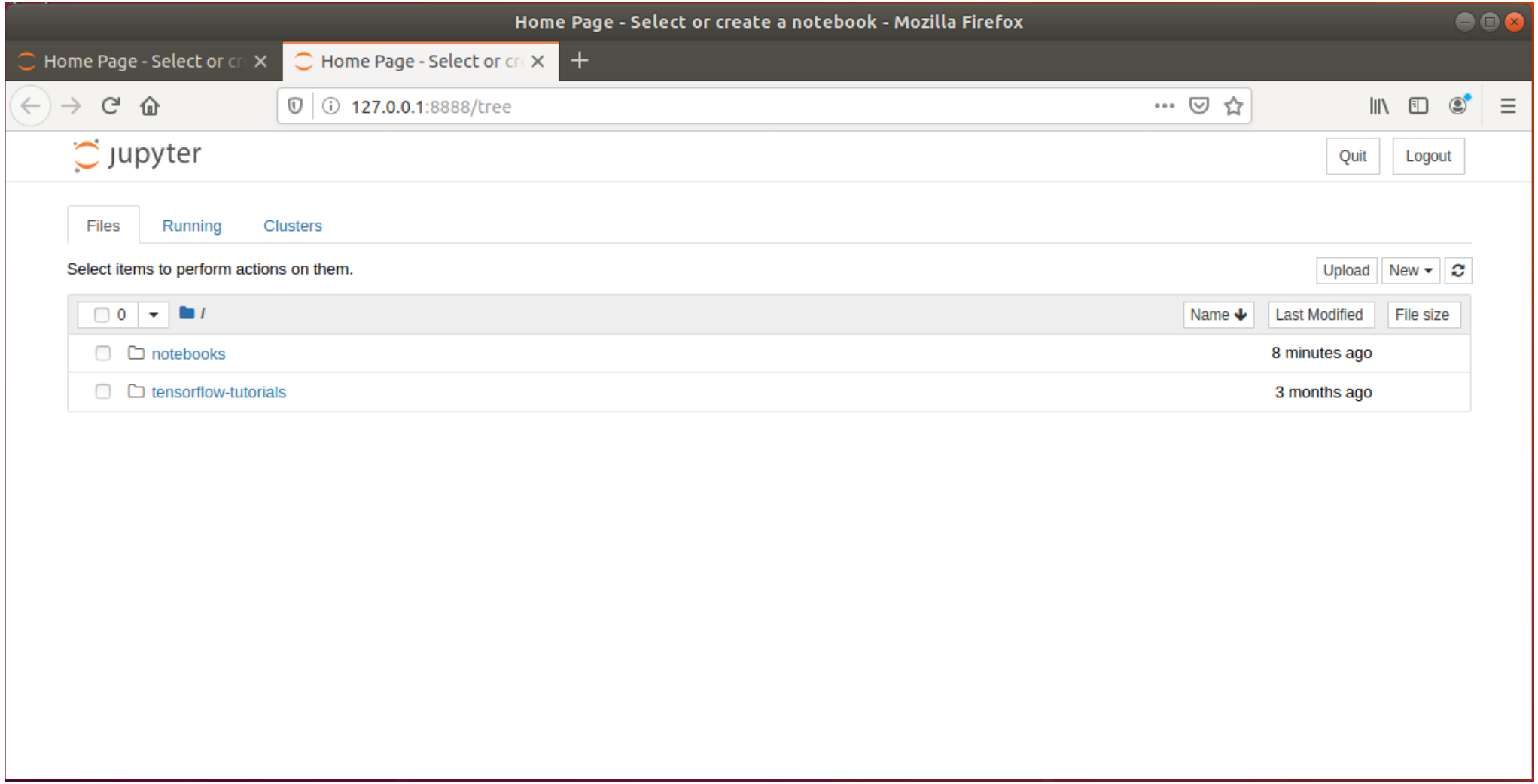
Commands
`> docker pull tensorflow/tensorflow:2.1.0-py3-jupyter`
`> mkdir ~/notebooks`
`> docker run -it --rm -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.1.0-py3-jupyter`
`> firefox http://127.0.0.1:8888/?token=<tokenid> &`

Tensorflow/2

<tokenid>

```
ubuntu@ubuntu:~$ docker run -it --rm -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.1.0-py3-jupyter

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...

[I 13:23:19.130 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
jupyter_http_over_ws extension initialized. Listening on /http_over_websocket
[I 13:23:19.708 NotebookApp] Serving notebooks from local directory: /tf
[I 13:23:19.709 NotebookApp] The Jupyter Notebook is running at:
[I 13:23:19.710 NotebookApp] http://8d9bee90baa5:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
[I 13:23:19.711 NotebookApp]  or http://127.0.0.1:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
[I 13:23:19.713 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:23:19.726 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
    Or copy and paste one of these URLs:
        http://8d9bee90baa5:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
     or http://127.0.0.1:8888/?token=04a7aba4099173a44372c0ace94487edc808484b0ea73e69
```

Tensorflow: Jupyter



Anaconda Python

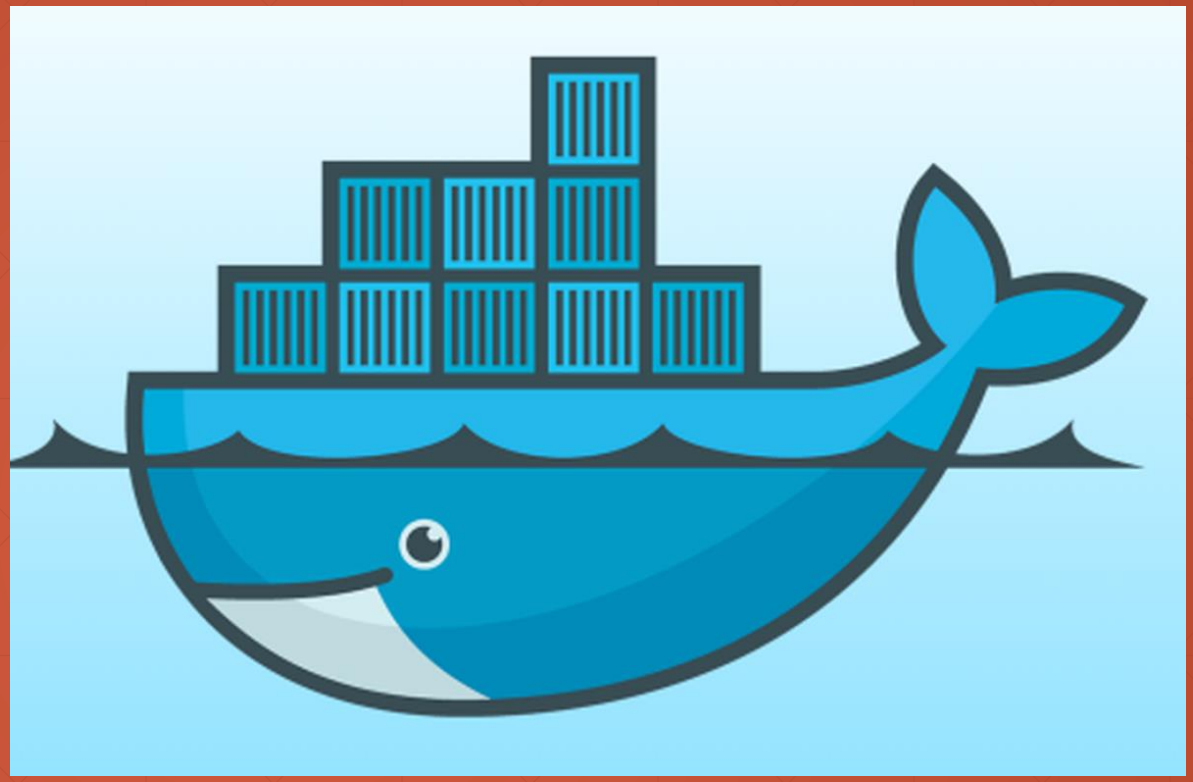
Installation of Anaconda Python/R distribution via Docker

Commands
`> docker pull continuumio/anaconda3:2020.02`
`> docker run -it continuumio/anaconda3:2020.02 /bin/bash`

References:

- https://docs.docker.com/
- Docker Up and Running (O'Reilly, 2015)
- Using Docker (O'Reilly, 2016)
- Docker Deep Dive (O'Reilly, 2018)
- Docker in Action (O'Reilly, 2019)
- Docker in Practice (O'Reilly, 2019)

Thanks



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

SESAR LAB

TensorFlow

Reference: Dr. Corrado Mio

Tensorflow: Docker installation

Version to install:

1. **tensorflow/tensorflow:2.3.1-jupyter**

Commands (Linux)
> docker pull tensorflow/tensorflow:2.3.1-jupyter
> mkdir ~/notebooks
> docker run -it -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.3.1-jupyter
> firefox http://127.0.0.1:8888/?token=<**tokenid**> &

Commands (Windows)
> docker pull tensorflow/tensorflow:2.3.1-jupyter
> mkdir \var\notebooks
> docker run -it -v "/var/notebooks:/tf/notebooks" -p 8888:8888 tensorflow/tensorflow:2.3.1-jupyter
> firefox http://127.0.0.1:8888/?token=<**tokenid**> &

Command executed

Commands

```
> docker run -it -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888
tensorflow/tensorflow:2.1.0-py3-jupyter
```

1. `run`: creates and run a *new* container
2. `-it`: interactive mode (`-i`), allocate a *pseudo* tty (`-t`)
3. `-v $(realpath ~/notebooks):/tf/notebooks`: *mounts* the *real path* `$(realpath ~/notebooks)` (this is an expression evaluated by the **bash**) as the path, inside the container, `/tf/notebooks` ,
4. `-p 8888:8888`: publish the *internal* local port `8888` on the external local port `8888` (in this case, the ports are the same)
5. `tensorflow/tensorflow:2.1.0-py3-jupyter`: the name of the image to execute

Start & Stop the docker image

Commands
`docker run -it -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.1.0-py3-jupyter`
`docker container stop <container_id>`
`docker container start -ai <container_id>`
`docker container ls`
`docker ps`

Useful Jupyter commands

Commands	Description
`!pip install --upgrade pip`	Update pip
`!pip freeze`	List of installed packages
`!pip list –outdated`	List of outdated packages
`!pip install <package>`	Install a package
`!pip install sklearn`	Install SciKit-Learn
`!pip install pandas`	Install Pandas
`!pip install seaborn`	Install Seaborn
`!pip install sklearn_pandas`	Install Scikit-Pandas
`!pip install scikit-image`	Install SciKit-Image
`!pip freeze > requirements.txt && {replace == with >= } && pip install -r requirements.txt --upgrade`	Update a list of packages: replace “==“ with “>=“ in “requirements.txt”

Useful Jupyter commands

Commands	
`!apt update`	Update the applications database
`!apt install -y git`	Install “git”

Tensorflow Serving

Tensorflow Serving: Docker installation (Linux)

Version to install:

1. **tensorflow/serving**

Commands (Linux)
> cd $HOME
> mkdir Projects
> cd Projects
> git clone https://github.com/tensorflow/serving
> TESTDATA="$HOME/Projects/serving/tensorflow_serving/servables/tensorflow/testdata"
>
> docker pull tensorflow/serving
> docker run -it --rm -p 8500:8500 -p 8501:8501 -v "$TESTDATA/saved_model_half_plus_two_cpu:/models/half_plus_two" -e MODEL_NAME=half_plus_two tensorflow/serving

Tensorflow Serving: -- (Windows)

Version to install:

1. **tensorflow/serving**

Commands (Windows)
> cd %USERPROFILE%
> mkdir Projects
> cd Projects
> git clone https://github.com/tensorflow/serving
> set TESTDATA=%USERPROFILE%\Projects\serving\tensorflow_serving\servables\tensorflow\testdata
>
> docker pull tensorflow/serving
> docker run -it --rm -p 8500:8500 -p 8501:8501 -v "%TESTDATA%\saved_model_half_plus_two_cpu":"/models/half_plus_two" -e MODEL_NAME=half_plus_two tensorflow/serving

Tensorflow Serving: Client

Ports:

- REST: 8501
- Grpc: 8500

Commands

```
> curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST
http://localhost:8501/v1/models/half_plus_two:predict
```

Cloud & Notebook Services

Azure	https://azure.microsoft.com/en-us/
Amazon Web Services	https://aws.amazon.com/
Digital Ocean	https://www.digitalocean.com/
Heroku	https://www.heroku.com/

https://clockwise.software/blog/amazon-web-services-introduction-largest-cloud-services-provider/

PythonAnywhere	https://www.pythonanywhere.com/
Heroku	https://www.heroku.com/
AWS Cloud9	https://aws.amazon.com/cloud9/
Google App Engine	https://cloud.google.com/appengine/docs
Codeanywhere	https://alternativeto.net/software/codeanywhere/

Google Colab	
Azure Notebooks	https://notebooks.azure.com/
Kaggle	https://www.kaggle.com/orgs-under-maintenance
Amazon Sagemaker	https://aws.amazon.com/sagemaker/
IBM DataPlatform Notebooks	https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/notebooks-parent.html
Jupyter Notebook	https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

Apache Zeppelin	https://zeppelin.apache.org/
nteract	https://nteract.io/
Beaker	http://beakerx.com/
	conda install -c conda-forge ipywidgets beakerx
	docker pull beakerx/beakerx
	docker run -p 8888:8888 beakerx/beakerx
Polynote	https://polynote.org/
paperspace	https://www.paperspace.com/

Python Notebooks	
Blinder	https://mybinder.org/
Kaggle	https://www.kaggle.com/
Google Colab	https://colab.research.google.com/
Azure Notebook	https://notebooks.azure.com/
CoCalc	https://cocalc.com/
Datalore	https://datalore.io/
Jupyter	https://jupyter.org/try
Paiza	https://paiza.cloud/en/

https://www.dataschool.io/cloud-services-for-jupyter-notebook/

Thanks

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

SESAR LAB

Reference: Dr. Corrado Mio

Spark

Apache Spark is a *cluster computing framework*.

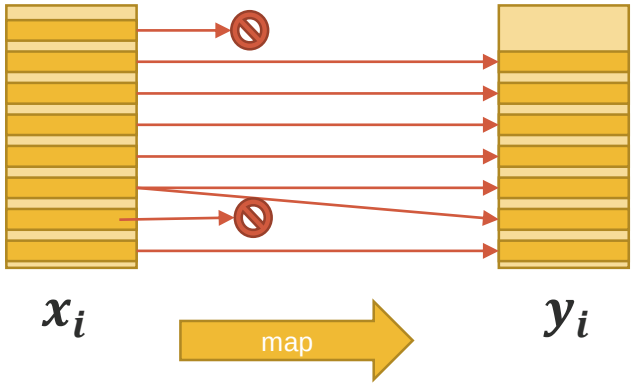
It extends the original *map/reduce* **Hadoop**'s computing model in a more coherent infrastructure based on:

- **RDD**: Resilient Distributed Dataset (v1.0)
- **DF**: Data Frame (v2.0)

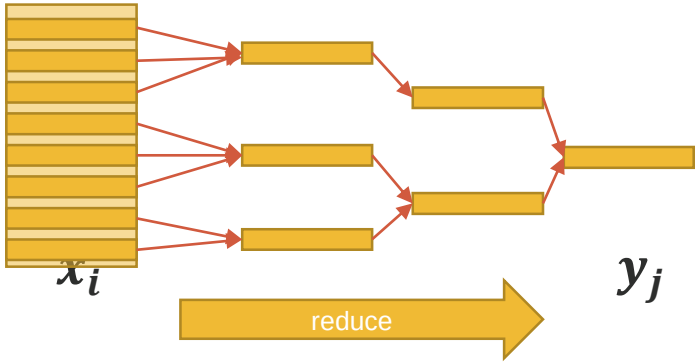
DF (and **RDD**) is similar to a relational table. The table can be partitioned horizontally and each partition can be handled by a different node of a cluster.

Spark: map/reduce computation model

Spark implements the *map/reduce* computation model*:*



$$y_i = \text{map}(x_i)$$

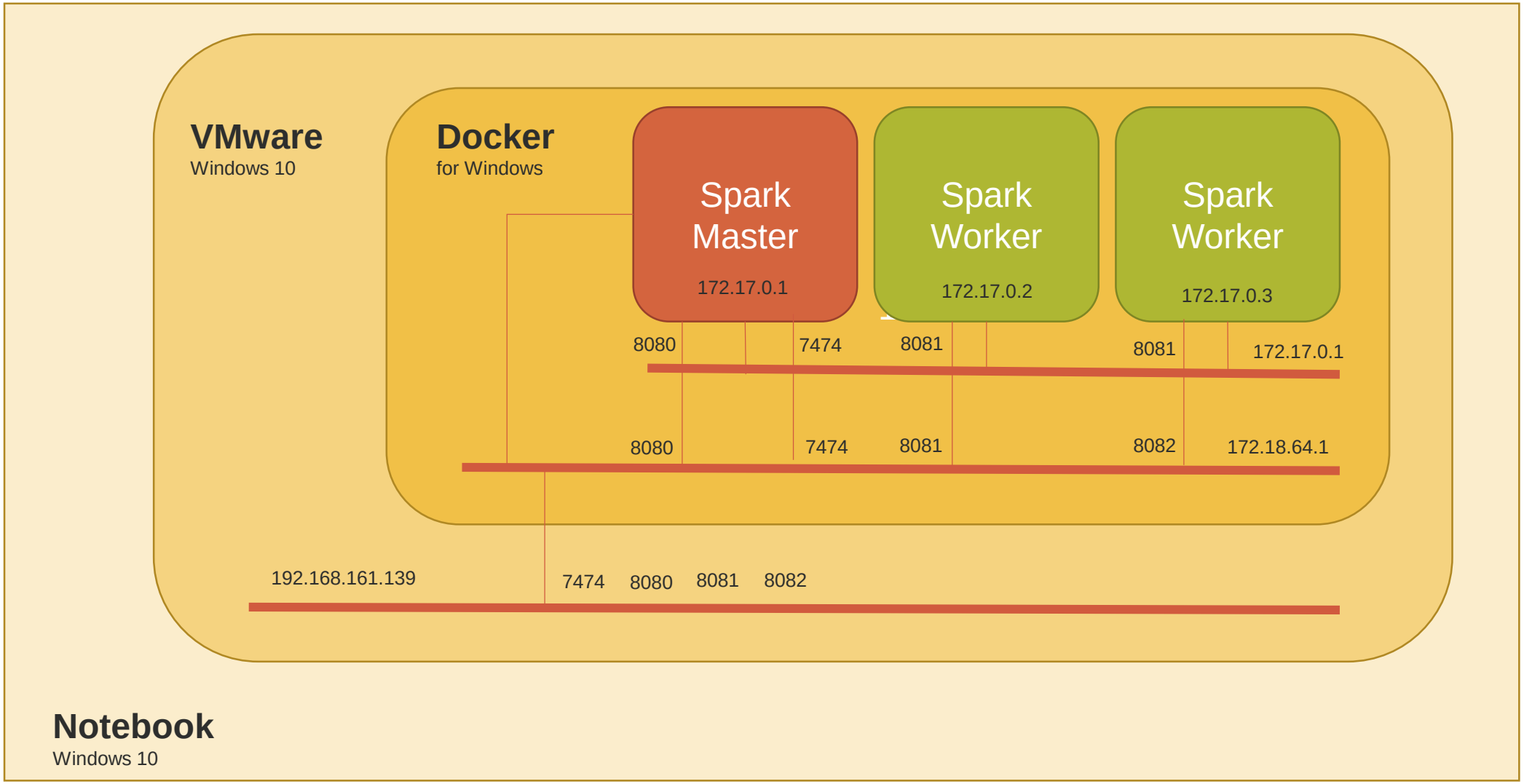


$$y_{t1} = \text{reduce}(x_{i1}, x_{i2}, \ldots)$$

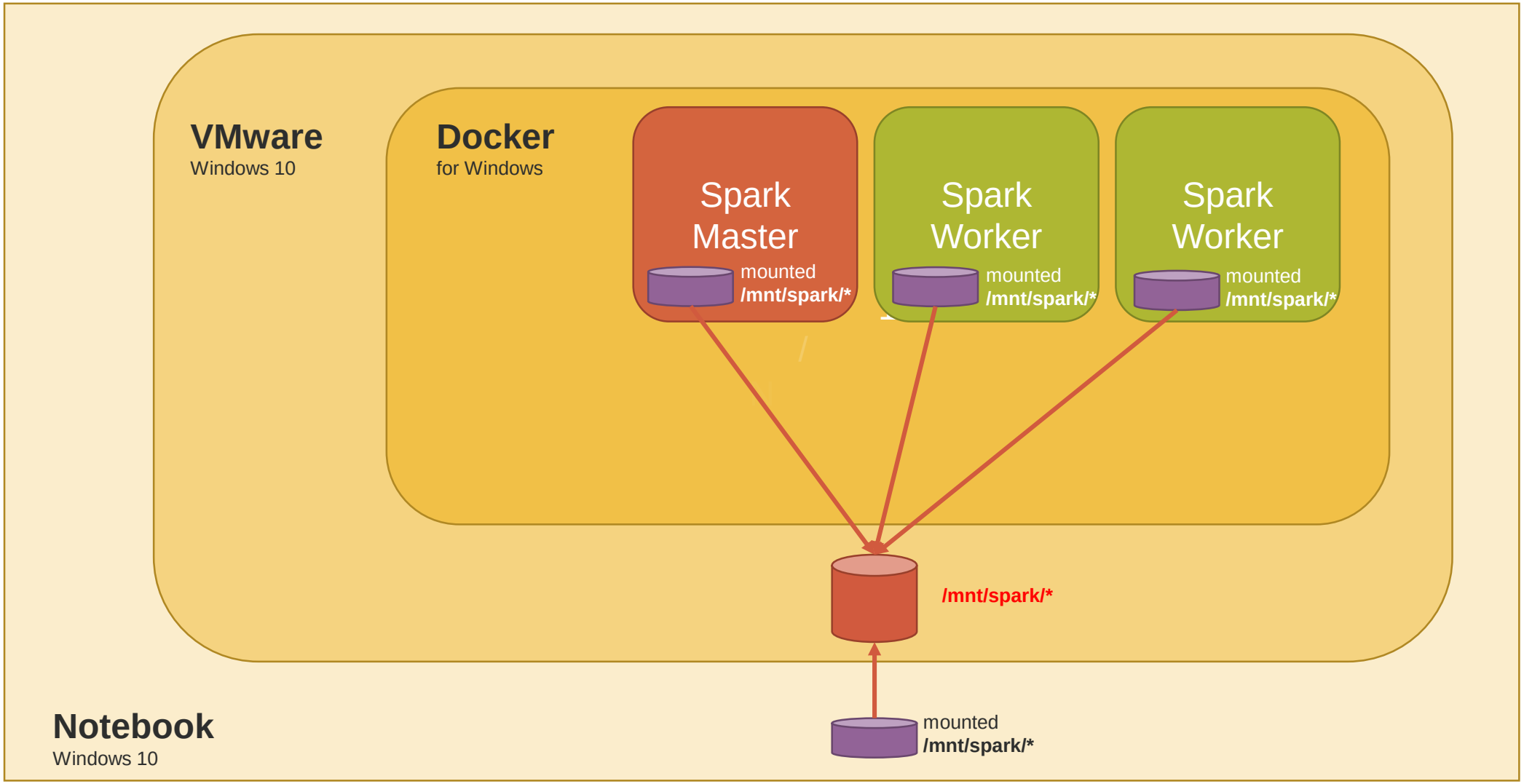
$$y_j = \text{reduce}(y_{t1}, y_{t2}, \ldots)$$

$$\text{reduce}: \mathrm{T}^{\mathrm{n}} \rightarrow \mathrm{T}, \qquad \mathrm{n} \in N \quad \text{(commutative \& associative)}$$

Spark: network architecture



Spark: filesystem architecture



Spark: infrastructure

The general **Spark Cluster** infrastructure must be composed by 4 elements:

1. the **Spark Master**
2. one or more **Spark Workers** (2)
3. a *network* that connects master and workers: each node must see all the others
4. a *distributed filesystem* accessible from master and workers **with the same paths** where to read the data/to write the results.
   This filesystem can be:
   - a **FTP/WEBDAV** server
   - a **HTTP** server
   - a mounted filesystem

Spark: Docker Installation

Commands (Windows)
> docker pull bde2020/spark-master
> docker pull bde2020/spark-worker
> mkdir c:\mnt\spark\spark-apps
> mkdir c:\mnt\spark\spark-data
>
> docker run --name spark-master -h spark-master -p 7077:7077 -p 8080:8080 -v "c:\mnt\spark\spark-data:/mnt/spark/spark-data" -v "c:\mnt\spark\spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-master
> docker run --name spark-worker-1 --link spark-master:spark-master -p 8081:8081 -v "c:\mnt\spark\spark-data:/mnt/spark/spark-data" -v "c:\mnt\spark\spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-worker
> docker run --name spark-worker-2 --link spark-master:spark-master -p 8082:8081 -v "c:\mnt\spark\spark-data:/mnt/spark/spark-data" -v "c:\mnt\spark\spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-worker

Spark: Docker Installation

Commands (Linux)
> docker pull bde2020/spark-master
> docker pull bde2020/spark-worker
> sudo mkdir /mnt/spark/spark-apps
> sudo mkdir /mnt/spark/spark-data
> sudo chown -R ubuntu:ubuntu /mnt/spark
> docker run --name spark-master -h spark-master -p 7077:7077 -p 8080:8080 -v "/mnt/spark/spark-data:/mnt/spark/spark-data" -v "/mnt/spark/spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-master
> docker run --name spark-worker-1 --link spark-master:spark-master -p 8081:8081 -v "/mnt/spark/spark-data:/mnt/spark/spark-data" -v "/mnt/spark/spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-worker
> docker run --name spark-worker-2 --link spark-master:spark-master -p 8082:8081 -v "/mnt/spark/spark-data:/mnt/spark/spark-data" -v "/mnt/spark/spark-apps:/mnt/spark/spark-apps" -e ENABLE_INIT_DAEMON=false -d bde2020/spark-worker

Thanks

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Google
Natural Language
API

SESAR LAB

Google
Cloud Platform

Google BERT & Docker

Reference: Dr. Corrado Mio

Micro NLP Introduction

Natural Language Processing is the area of the Computer Science that studies the analysis and interpretation of speech and text.

Some services offered are (https://en.wikipedia.org/wiki/Natural_language_processing):

- speech ←→ text
- part of speech tagging: objects and what it the object referred by a personal pronoun (it, you, …)
- parsing/tagging: the hierarchical structure of the text
- …

Another service is: *word/sentence embeddings*

Word/Sentence embeddings

The **word embedding** (https://en.wikipedia.org/wiki/Word_embedding) is a technique used to map a word in a **numerical vector**.

The main idea is that a word can be characterized by several *features* (singular/plural/…, male/female/neutral/…, etc) and these features can be described using a numerical vector with enough elements.

There are several algorithms able to obtain this result:

- word2vec (https://en.wikipedia.org/wiki/Word2vec)
- GloVe (https://en.wikipedia.org/wiki/GloVe_(machine_learning))
- fastText (https://en.wikipedia.org/wiki/FastText)
- **BERT** (https://en.wikipedia.org/wiki/BERT_(language_model))
- …

Little problem: there is no relation between the *human* concept of *feature* and the features generated by the algorithms! This is similar to the *filters* in a Convolutional NN.

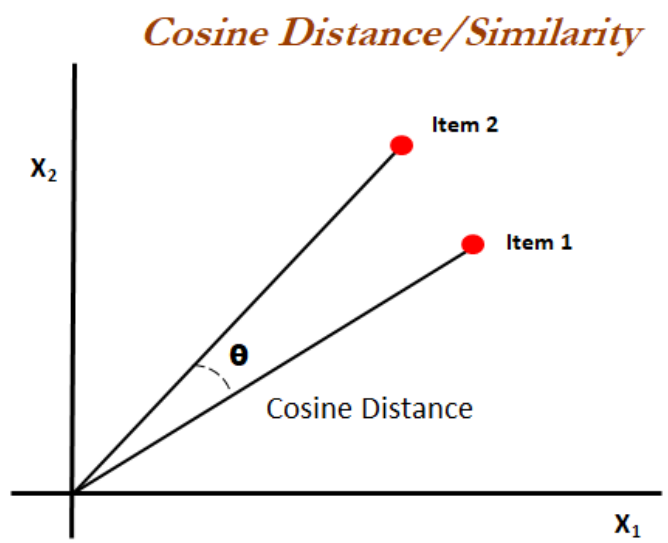
Cosine distance

If we have two words w_1 and w_2 and their *embeddings* (numerical vectors!) $v_1 = [a_1, \dots a_n]$, $v_2 = [b_1, \dots b_n]$, we can compare the *distance* between words using the **cosine distance**. The **dot product** of two vectors is defined as:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i$$

and this is equal to

$$\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| \cos \alpha$$



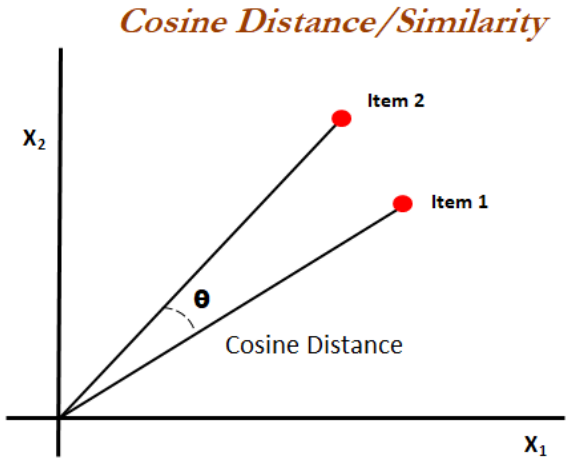
if $|\boldsymbol{a}| = |\boldsymbol{b}| = 1$, the result is

$$\boldsymbol{a} \cdot \boldsymbol{b} = \cos \alpha$$

Cosine distance/2

Cosine distance is defined as:

$$d(\boldsymbol{a}, \boldsymbol{b}) = \frac{|\boldsymbol{a} \cdot \boldsymbol{b}|}{|\boldsymbol{a}| \cdot |\boldsymbol{b}|} = |\cos \alpha|$$



we can have

- $d(\boldsymbol{a}, \boldsymbol{b}) \sim 1$ the words are used in the similar contexts (*cat*, *dog*)
- $d(\boldsymbol{a}, \boldsymbol{b}) \sim 0$ the words are rarely used in the same contexts (*bear*, *space*)

Word embeddings → sentence embeddings

Using the *word embedding* it can be possible to do *operations* with words.

The classic example using $king, queen, male, female$ is:

$$d(king, queen) \cong d(male, female)$$

$$qeen \cong king - male + female$$

(remember that the word is represented by a *numerical vector*)

The generalization of **word embedding** is **sentence embedding** where it is computed the embedding vector for a complete sentence (in theory, an entire book!)

Google BERT

Bidirectional Encoder Representations from Transformers (BERT).

The *word embedding* evaluation can be *context free* or *contextual*. The *contextual* version can be *unidirectional* or *bidirectional*.

Algorithms as **word2vec** and **GloVe** are *context free* and are able *only* to evaluate the embedding of a **single** word: they are based on the concepts of *bag of words,* where the order of the words are totally lost.

The *contextual* algorithms consider the sentence (the words in their order). In a phrase like "*I accessed the **bank** account*", in the analysis of *"**bank**",* the unidirectional algorithms consider only the sentence "*I accessed the*", where the bidirectional algorithms consider also the following word *"account".*

BERT uses the bidirectional approaches.

Google BERT in numbers

	max n of inputs	n of layers	n of parameters
BERT base	512	12	110.000.000
BERT large	512	12	340.000.000

BERT as service

Download the source from

- https://github.com/hanxiao/bert-as-service
- copy the compressed file in a machine where is installed Docker
- unzip it
- enter into **bert-as-service-master**
- edit the file **docker/Dockerfile** and replace **tensorflow/tensorflow:1.12.0-gpu-py3** with **tensorflow/tensorflow:1.12.0-py3** (remove the **gpu** support, if this is not available)
- download a **BERT** models and unzip them into the same machine where is installed Docker (next slide)

BERT models

List of available BERT models.

Model	URL
BERT-Base, Uncased	https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip
BERT-Large, Uncased	https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-24_H-1024_A-16.zip
BERT-Base, Cased	https://storage.googleapis.com/bert_models/2018_10_18/cased_L-12_H-768_A-12.zip
BERT-Large, Cased	https://storage.googleapis.com/bert_models/2018_10_18/cased_L-24_H-1024_A-16.zip
BERT-Base, Multilingual Cased (New)	https://storage.googleapis.com/bert_models/2018_11_23/multi_cased_L-12_H-768_A-12.zip

BERT as service

Create the image and run it

Commands
> export PATH_MODEL=~/models/<model_name>
> export NUM_WORKER=2
> docker build -t bert-as-service -f ./docker/Dockerfile .
> docker run --runtime nvidia -dit -p 5555:5555 -p 5556:5556 -v **$PATH_MODEL**:/model -t bert-as-service **$NUM_WORKER**
> docker run -dit -p 5555:5555 -p 5556:5556 -v **$PATH_MODEL**:/model -t bert-as-service **$NUM_WORKER**
win:> docker run -dit -p 5555:5555 -p 5556:5556 -v **%PATH_MODEL%**:/model -t bert-as-service **%NUM_WORKER%**

References

- Neural Network Methods in Natural Language Processing – 2017
- https://bert-as-service.readthedocs.io/en/latest/section/get-start.html
- https://www.blog.google/products/search/search-language-understanding-bert/
- https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
- https://arxiv.org/abs/1810.04805
- https://github.com/google-research/bert
- https://en.wikipedia.org/wiki/BERT_(language_model)

Thanks