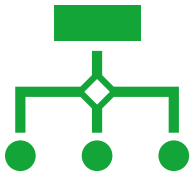
# Docker Ecosystem and Tools

Speaker: Lorenzo Patera

Imola, 25/11/2020

imola informatica

# Agenda



**Containers and**

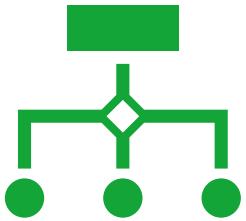
**Docker ecosystem**

**Docker hands-on**

**Portrainer**

**Web interface**

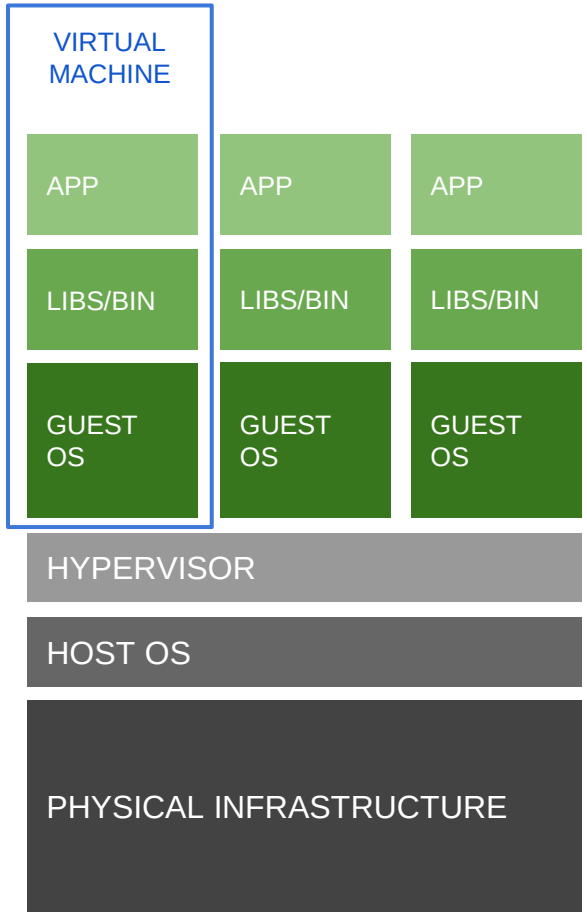
# Agenda



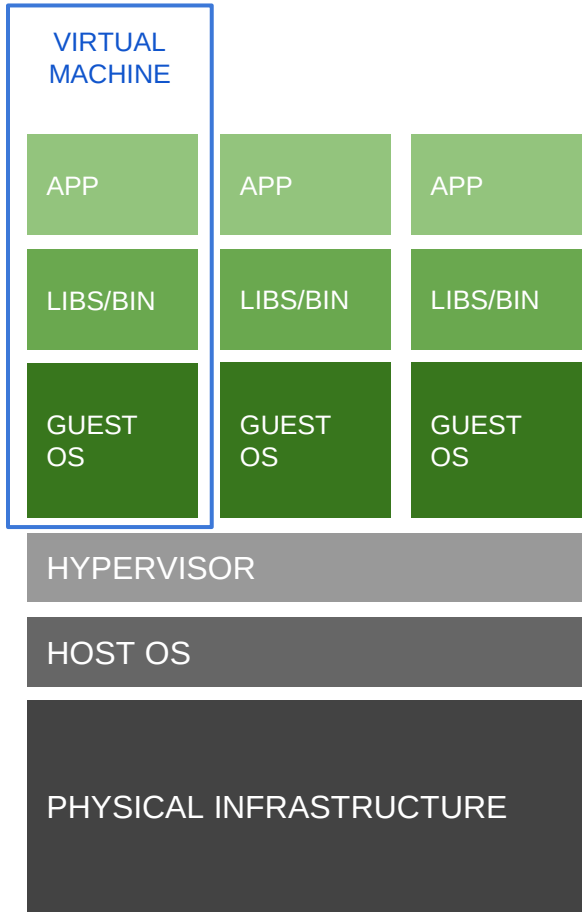
**Containers and**

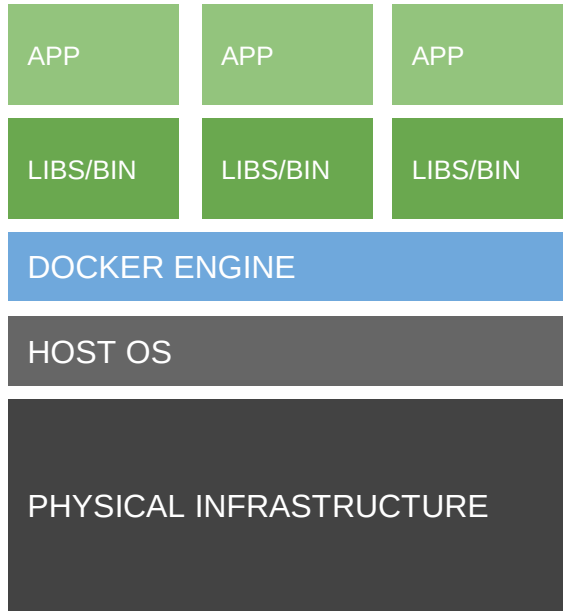
**Docker ecosystem**

# Virtualization vs Containerization

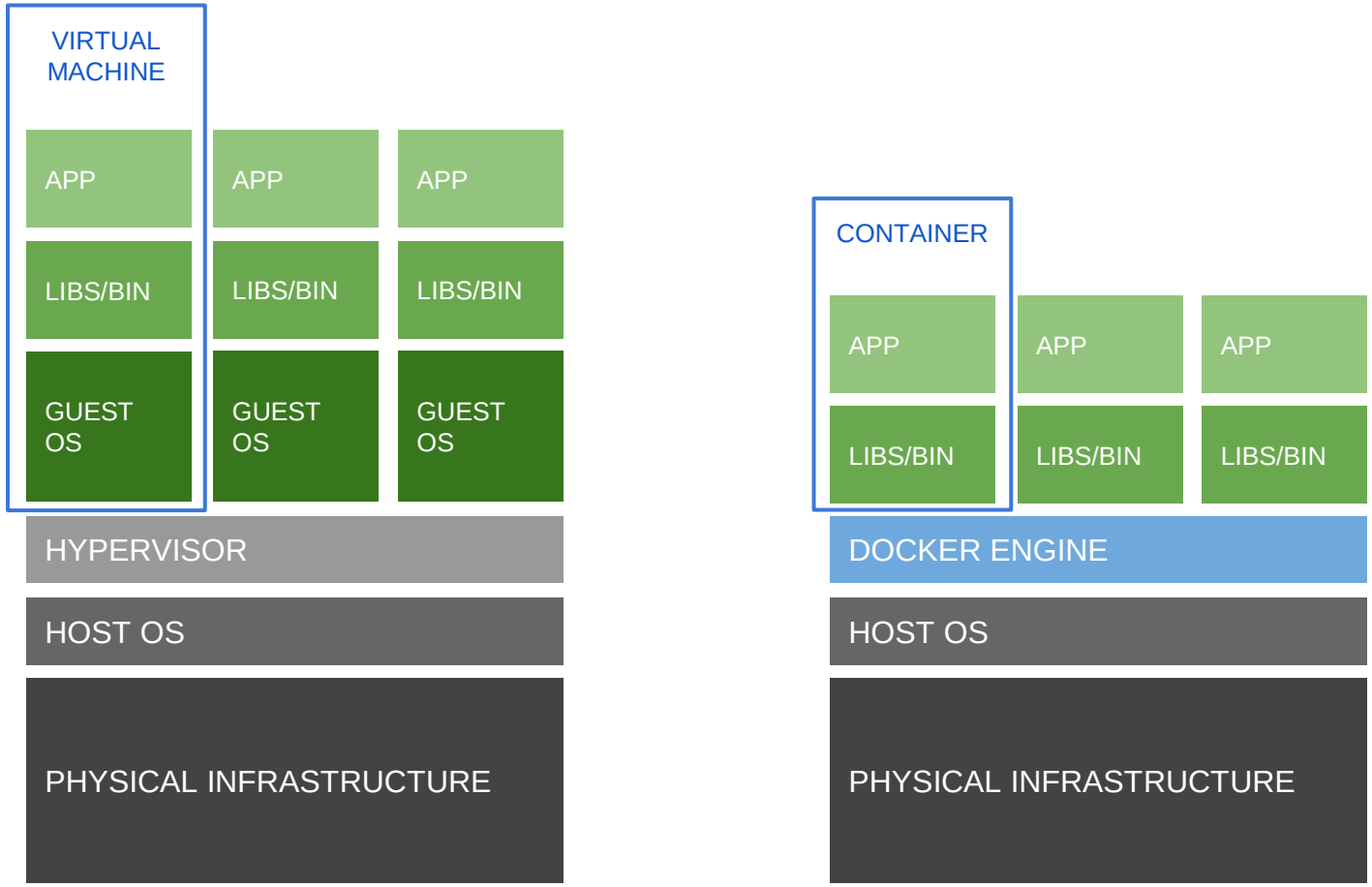


# Virtualization vs Containerization

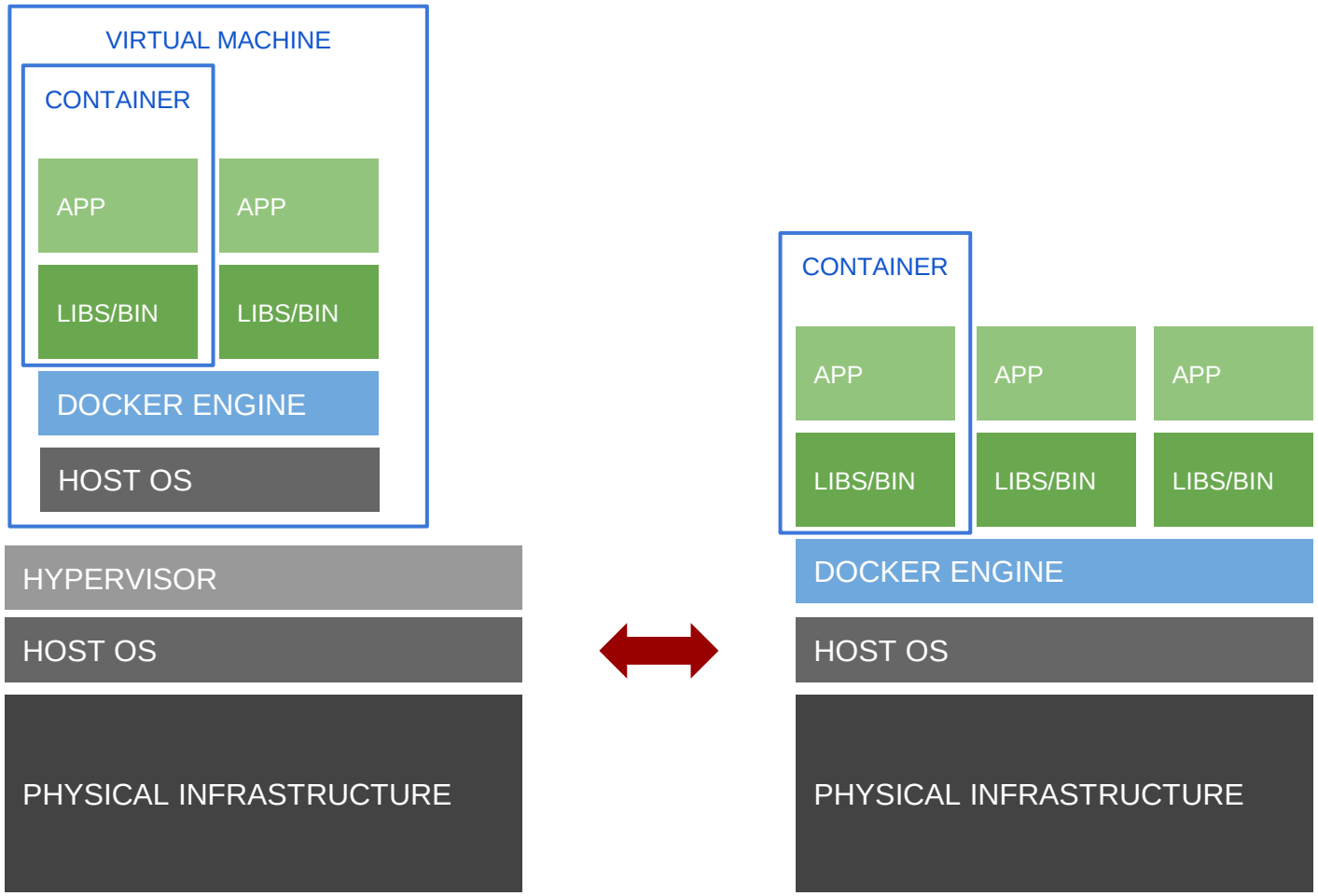


# Virtualization vs Containerization



# Docker flavors



# Containerization vs Virtualization

containers include an **application/service** together with its dependencies

containers **share kernel** with other containers

containers run as **isolated processes**

higher efficiency w/r to virtualization

**images** are the cornerstone in crafting declarative/automated, easily repeatable, and scalable services and applications

imola informatica

# What is Docker?

Docker is a platform for developers and sysadmins to develop, ship, and run applications.

Docker lets you quickly assemble applications from components and eliminates the friction that can come when shipping code.

Docker lets you get your code tested and deployed into production as fast as possible.

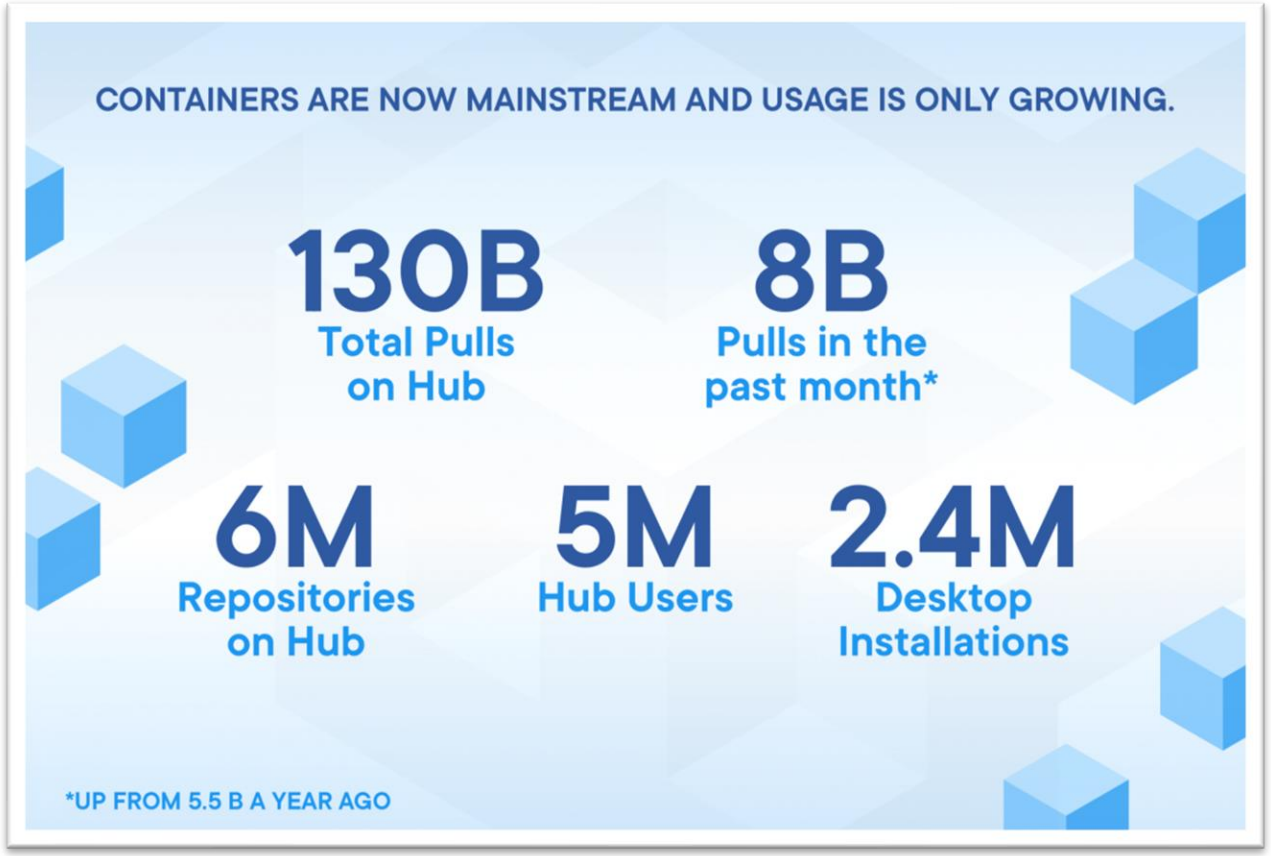
https://docs.docker.com/engine/

# What is Docker?

Docker consists of:

- The Docker Engine - our lightweight and powerful open source containerization technology combined with a work flow for building and containerizing your applications.
- Docker Hub - our SaaS service for sharing and managing your application stacks.

# Docker inception




**2013**: Docker comes to life as an open-source project at *dotCloud Inc.*

**2014:** company changed name to “Docker Inc.” and joined the Linux Foundation

**2015**: tremendous increase in popularity

**Today:** 130 Billion pulls on Hub, 8 Billion pulls in a month, 6M repositories, 5M users

*https://www.docker.com/blog/introducing-the-docker-index/*

# Docker - Under the hood

Standard Bodies: **Open Container Initiative** (OCI), **Cloud Native Computing Foundation** (CNCF)

- OCI Image specification
- OCI Runtime Specification

**runc** runtime (formerly libcontainer)

The Docker **Images:**

- **copy-on-write** filesystems (e.g. AUFS)

The **Go programming language**

- a statically typed programming language developed by Google with syntax loosely based on C

# Running Docker on Windows/MacOS

On **Windows** (Windows 10 Pro 17.09 «Falls Creator Update»):

- «**Docker for Windows**» official tool
  - Linux containers → run on a Hyper-V Linux VM
  - Windows containers → run on a Hyper-V «Windows server kernel» VM

  Limitation: other hypervisors (eg. VirtualBox) cannot run if Hyper-V if enabled

On **Windows Server** (Windows Server 2016)

- **native** Windows Server Containers (no need for VM)

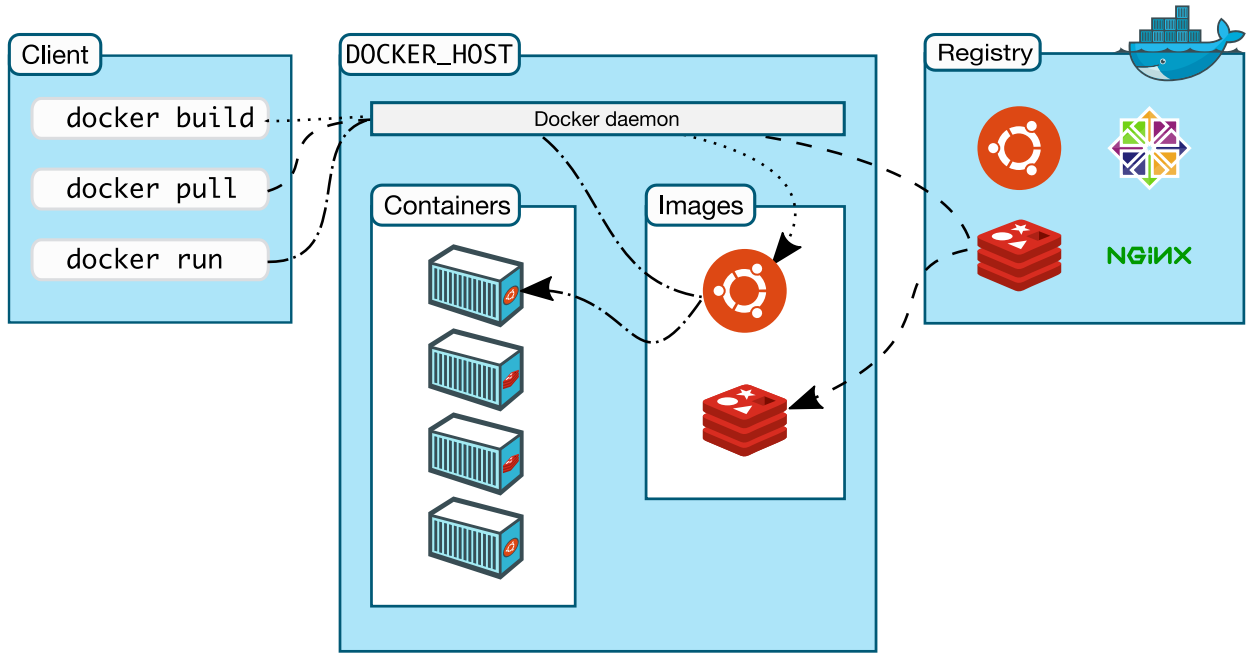
On **MacOS**: «Docker for Mac» official tool

- run Linux containers on a HyperKit VM

... or you can always manually create a Linux or Windows Server VM with VirtualBox/VMWare with shared folders and install Docker on it

# Docker Architecture

- **Docker daemon** – The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- **Docker client** – The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out.
- **Docker registries** – A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. Docker registries are the distribution component of Docker.



# Docker objects

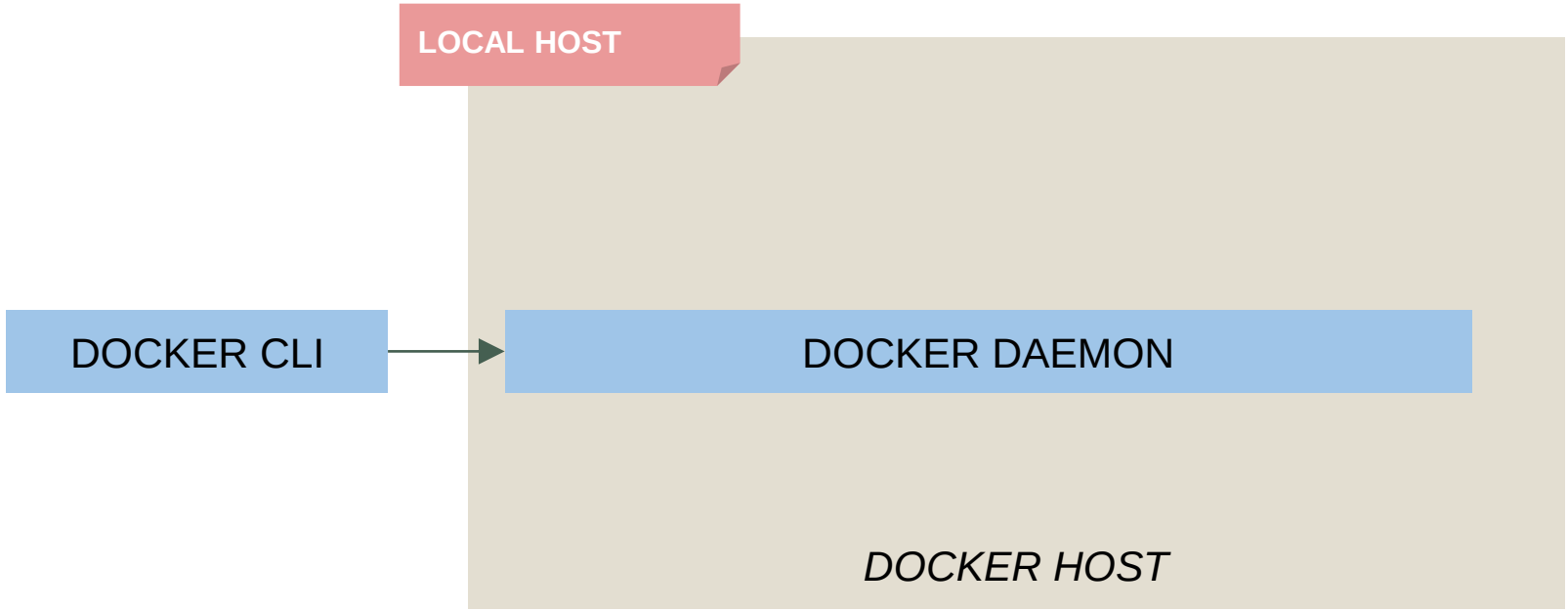
## Docker images

A Docker image is a read-only template. For example, an image could contain an Ubuntu operating system with Apache and your web application installed. Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created. Docker images are the **build component of Docker**.

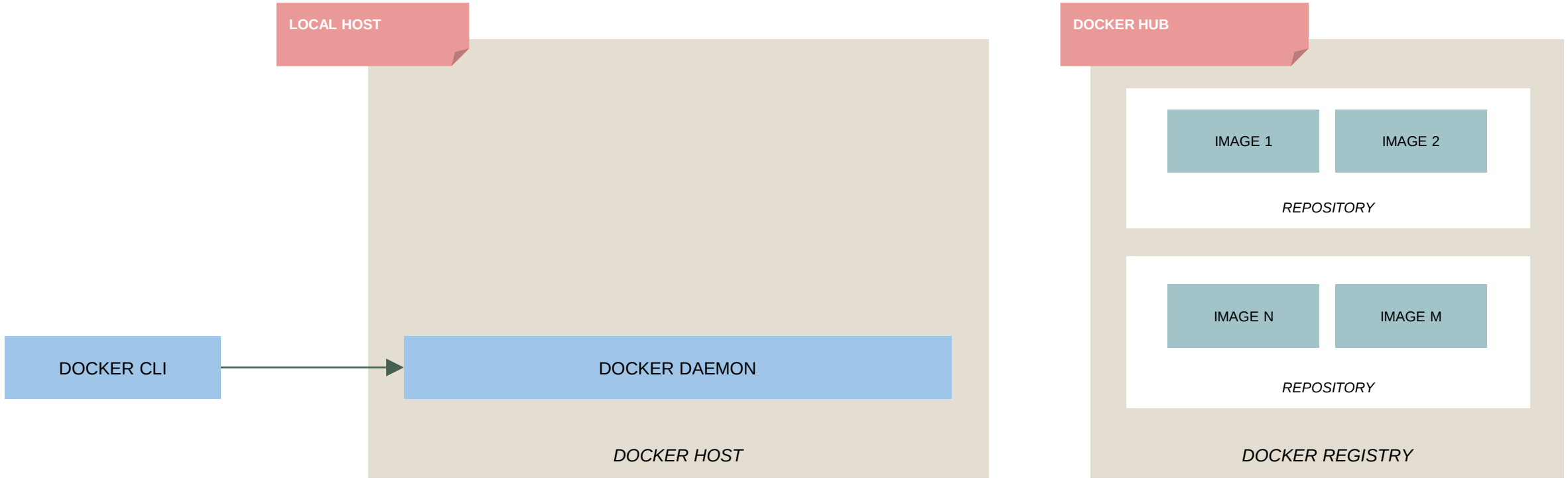
## Docker containers

Docker containers are similar to a directory. A Docker container holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform. Docker containers are the **run component of Docker**.

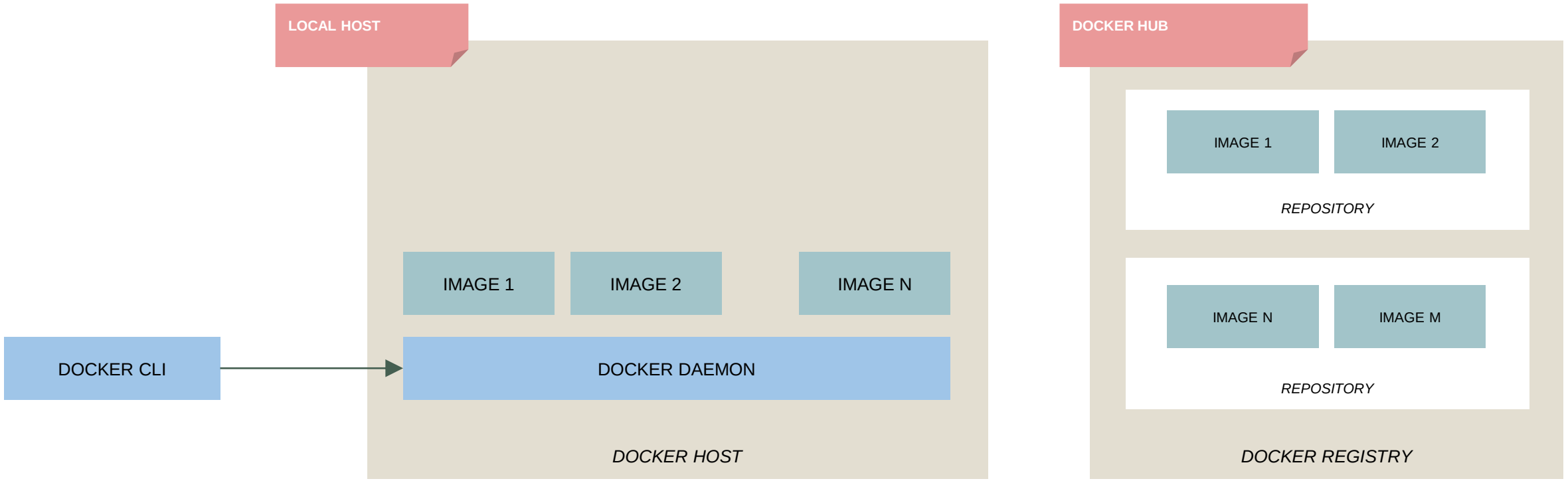
# Docker components



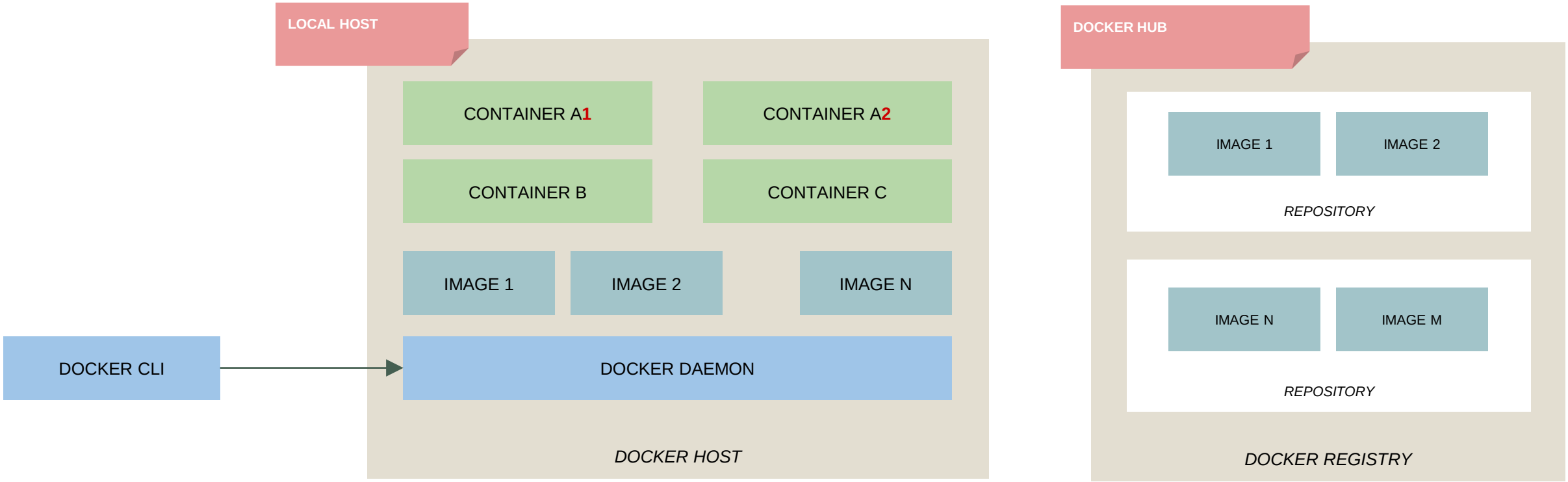
# Docker components



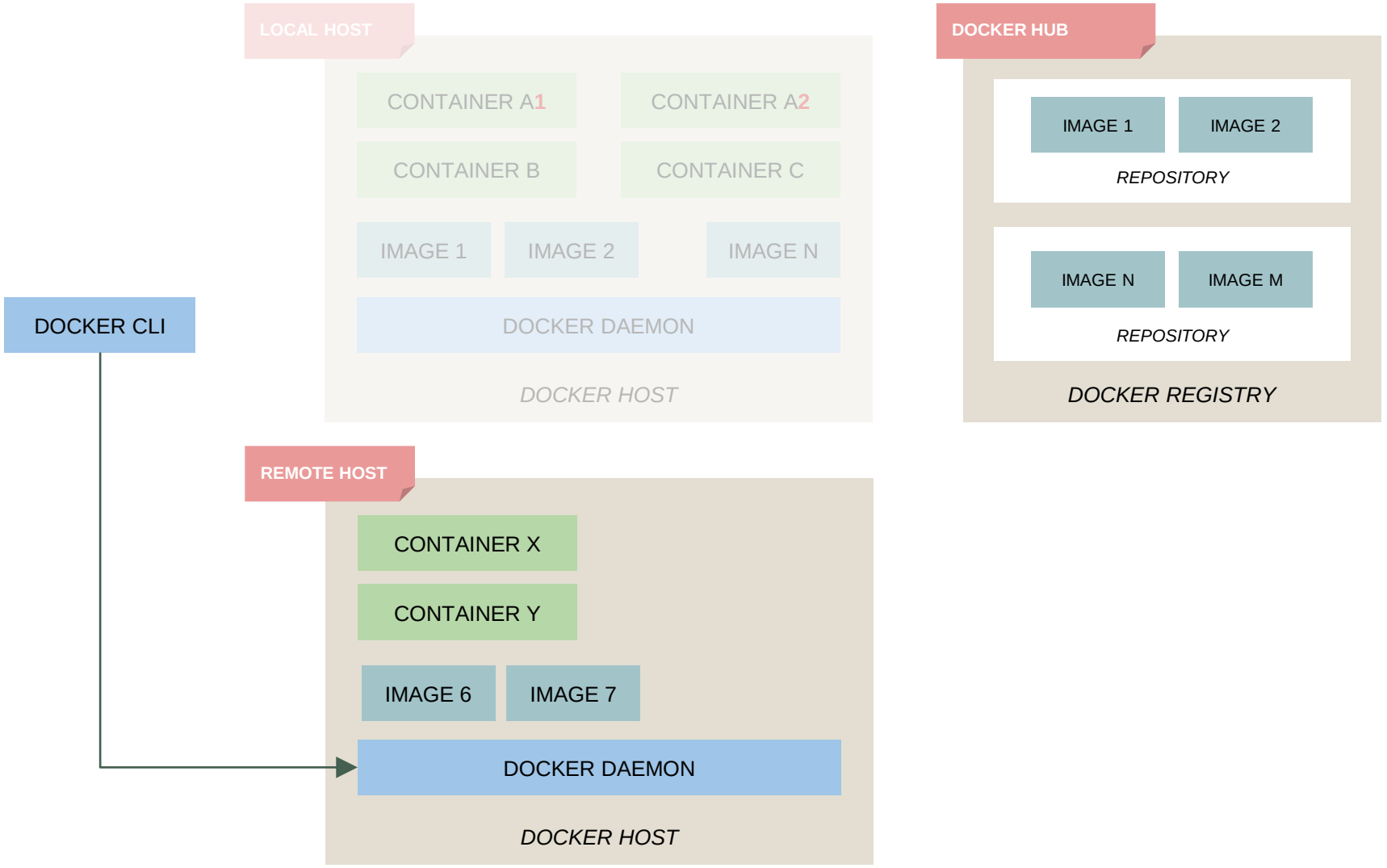
# Docker components



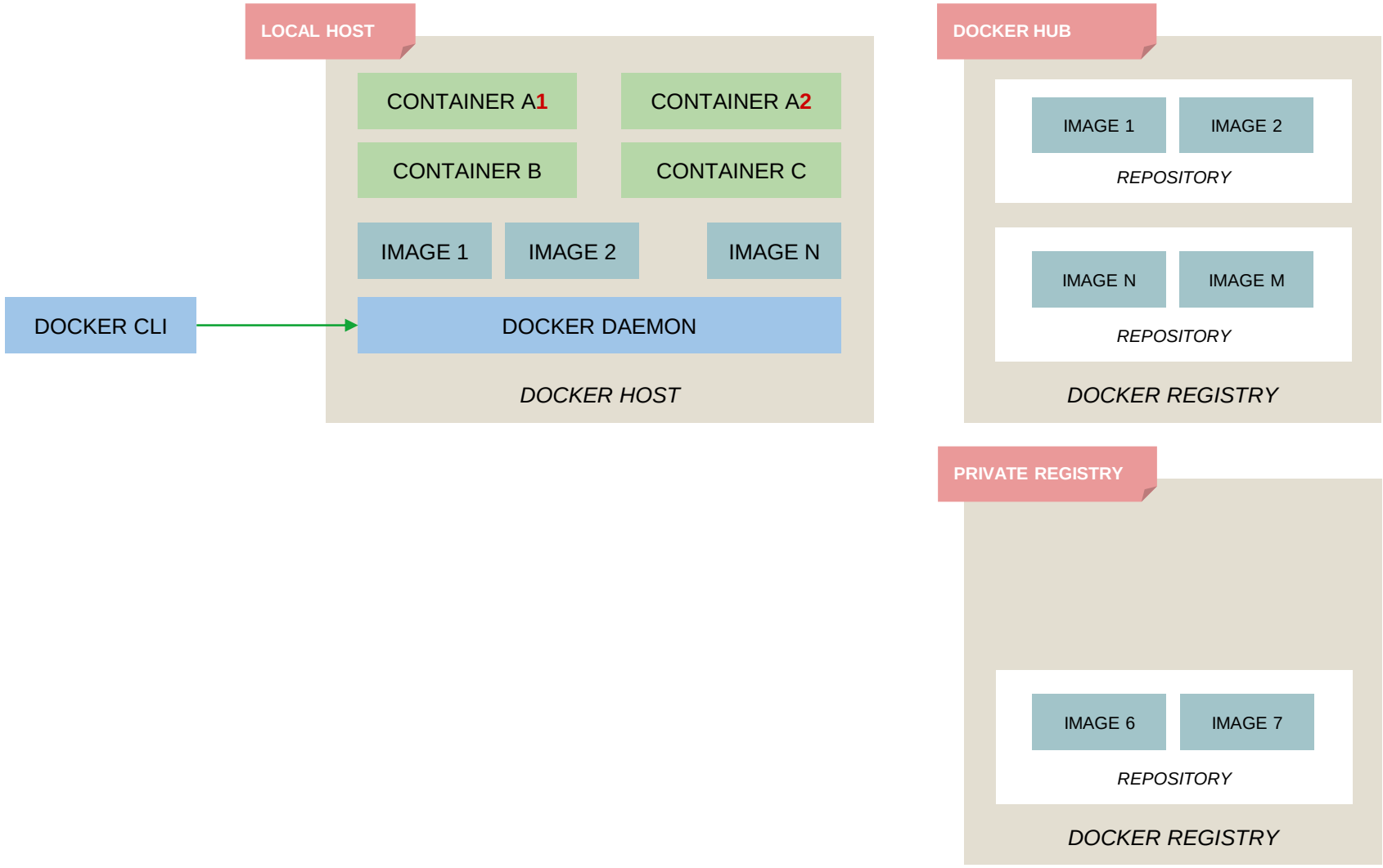
# Docker components



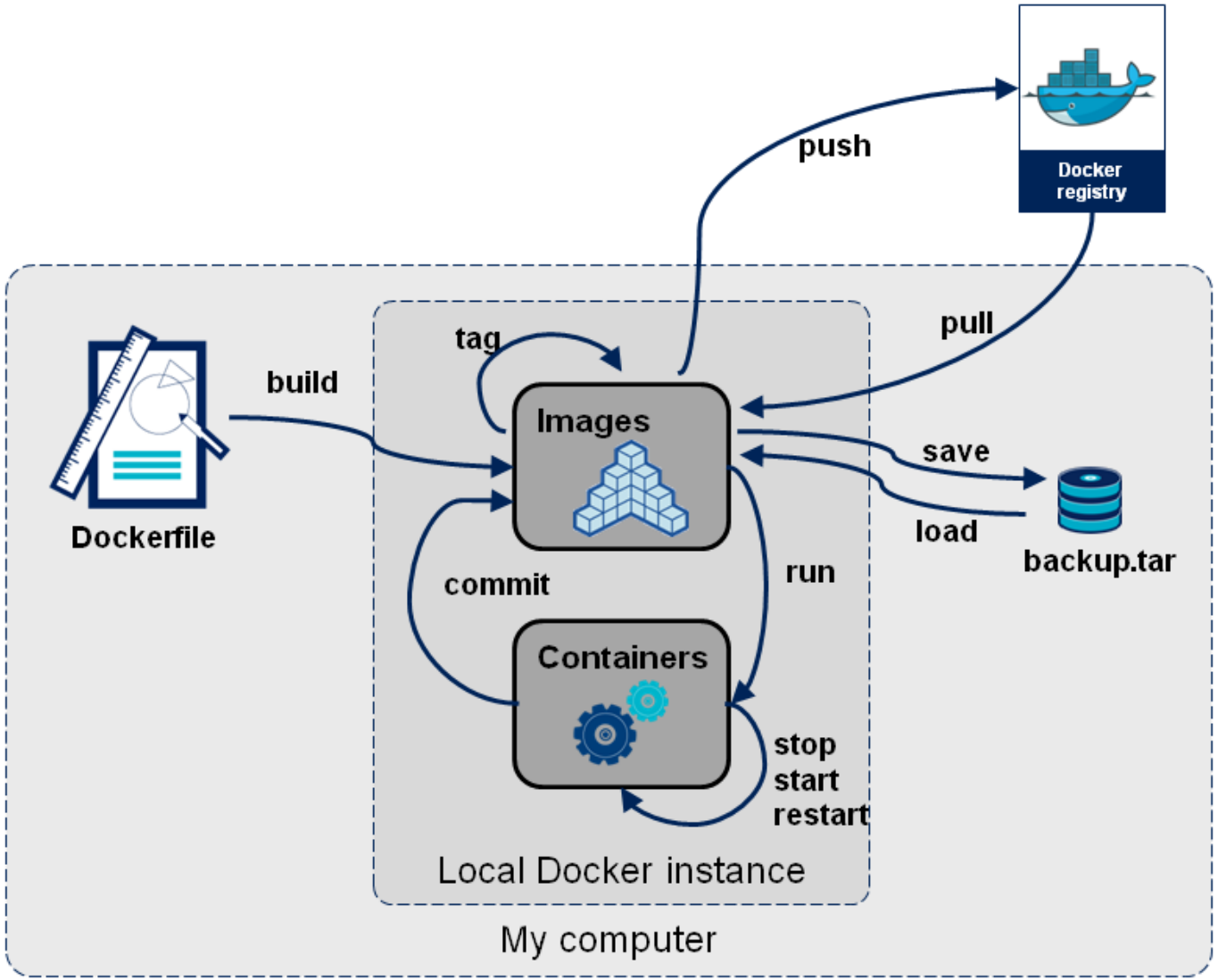
# Docker components



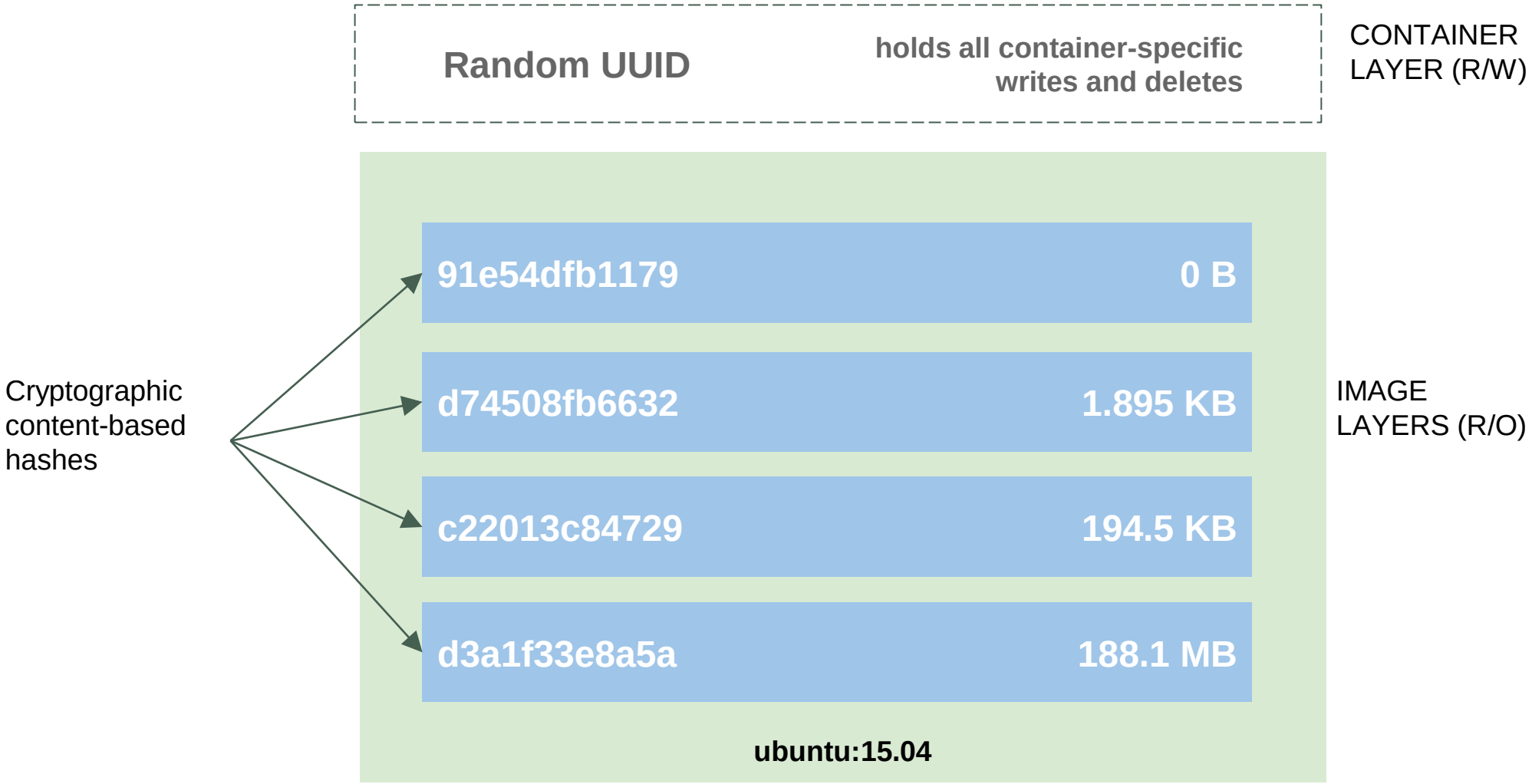
# Docker components



# Docker Container Lifecycle



# Docker images



# Docker Images

Docker images are **read-only** stacks of **layers** → copy-on-write approach

each layer is uniquely identified by a **cryptographic content-based hash** (>=v.1.10)

- **collision detection** mitigation
- strong and efficient **content comparison mechanism**

This approach is hugely beneficial

- **efficient disk usage**
  - each new layer keeps only differences from preceding layers
  - layers can be shared among images, e.g. “base” layers such as OS layers (fedora:latest, ubuntu:latest)
- **ease of modification**
  - new images may be built by simply stacking new layers on top of preceding ones, leaving the below layers unmodified

# Docker Images - Naming convention

[hostname[:port]]/[username]/reponame[:tag]

**Hostname/port** of **registry** holding the image. If missing, defaults to Docker Hub public registry.

**Username**. If missing, defaults to **library** username on Docker Hub, which hosts official, curated images.

**Reponame**. Actual image repository.

**Tag**. Optional image specification (e.g., version number). If missing, defaults to **latest.**

# Docker CLI

**docker pull** – get image from registry

```
$ docker pull imagename
```

copy image from **registry** to **localrepo**

**docker build** - builds an image from a Dockerfile

```
$ docker build .
```

builds a **new image** based on a **Dockerfile located** on the current directory (**.**)

```
$ docker build -t imagename .
```

builds a **new image** based on a **Dockerfile located** on the current directory (.) and **names that image as *imagename***

# Docker CLI

**docker run** - runs a command in a new container, based on a specific image

```
$ docker run hello-world
```

runs the **default command** on a newly created container, based on the **public hello-world image**

```
$ docker run -it ubuntu /bin/bash
```

runs the **bash command interactively** on a newly created container, based on the **public ubuntu image**

```
$ docker run -d tomcat:8.0
```

runs the **default command (*catalina.sh*)** on a newly created container, based on the **public tomcat V.8.0 image**, and **detaches (-d) it to background**

**docker stop** - stops a running container

```
$ docker stop containerId
```

stops container identified by *containerId*

**docker start** – starts a stopped container

```
$ docker start containerId
```

start container identified by *containerId*

imola informatica

# Docker CLI

## docker exec

runs a command in an already **running container**

**$ docker exec -it *containerId* /bin/bash**

runs the **bash command interactively** on container ***containerId***

## docker container

manage container

**$ docker container «command»**

List containers

**$ docker container ls** — **docker ps**

lists **running** containers

**$ docker container ls –a** — **docker ps –a**

lists all containers (including stopped ones)

Remove container

**$ docker container rm «containerName»**

remove container **containerName** — **docker rm containerName**

# Docker CLI

**docker image**

manage image

**$ docker image comando**

List containers

**$ docker image ls** **$ docker images**

lists images

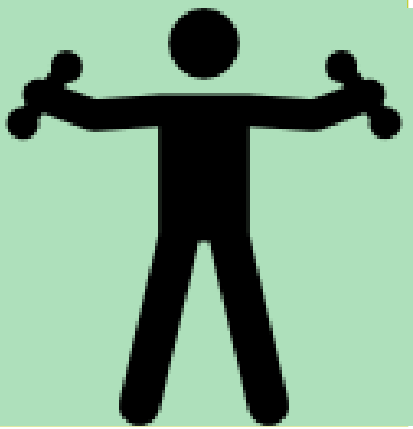
Remove image

**$ docker image rm imageName** **$ docker rmi imageName**

removes image imageName

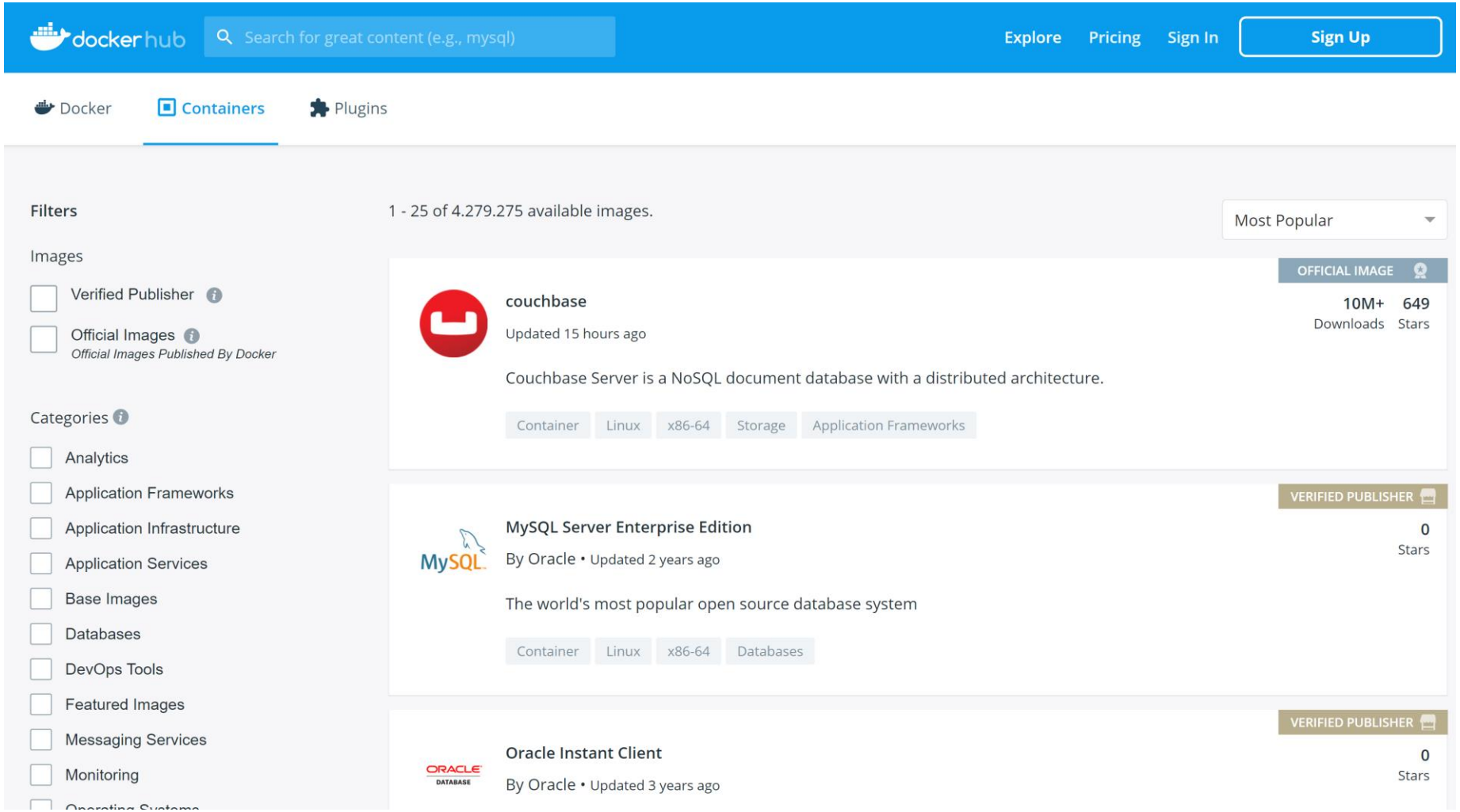
# Docker images

```
# Vedi https://hub.docker.com/_/httpd/

docker pull httpd
docker run httpd
docker stop ???
docker start ???
```



# Docker images

Browse to: https://hub.docker.com/explore/




imola informatica

# Dockerfile example - PostgreSQL

```
FROM ubuntu
MAINTAINER SvenDowideit@docker.com

RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8

RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" > /etc/apt/sources.list.d/pgdg.list
RUN apt-get update && apt-get install -y python-software-properties software-properties-common postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3

USER postgres

RUN    /etc/init.d/postgresql start &&\
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
    createdb -O docker docker

RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/9.3/main/pg_hba.conf
RUN echo "listen_addresses='*'" >> /etc/postgresql/9.3/main/postgresql.conf

EXPOSE 5432

VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c", "config_file=/etc/postgresql/9.3/main/postgresql.conf"]
```

# Dockerfile Reference

**FROM**: sets the **base image** for subsequent instructions

**MAINTAINER**: reference and credit to image author

**RUN**: runs a command and commits changes to a layer on top of previous image layers; the committed image will be visible to the next steps in the Dockerfile

**ADD**: copies files from the source on the host (or remote URL) into the container's filesystem destination

**COPY**: copies files from the source on the host into the container's filesystem destination (no URL, no automatic archive expansion support)

**CMD:** sets the default command for an executing container

**ENTRYPOINT**: sets/overrides the default entrypoint that will (optionally) execute the provided CMD

**ENV**: sets environment variables

**EXPOSE**: instructs Docker daemon that containers based on the current image will listen on the specified **network port**

**USER**: sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile

**VOLUME**: creates a mount point for external data (from native host or other containers)

**WORKDIR**: sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile

**LABEL**: adds metadata to an image

# Dockerfile Reference

**FROM**: sets the **base image** for subsequent instructions

**MAINTAINER**: reference and credit to image author

**RUN**: runs a command and commits changes to a layer on top of previous image layers; the committed image will be visible to the next steps in the Dockerfile

**ADD**: copies files from the source on the host (or remote URL) into the container's filesystem destination

**COPY**: copies files from the source on the host into the container's filesystem destination (no URL, no automatic archive expansion support)

**CMD:** sets the default command for an executing container

**ENTRYPOINT**: sets/overrides the default entrypoint that will (optionally) execute the provided CMD

**ENV**: sets environment variables

**EXPOSE**: instructs Docker daemon that containers based on the current image will listen on the specified **network port**

**USER**: sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile

**VOLUME**: creates a mount point for external data (from native host or other containers)

**WORKDIR**: sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile

**LABEL**: adds metadata to an image

# Dockerfile reference – Shell vs Exec form

For CMD, RUN, ENTRYPOINT you can use:

## Shell form

- `RUN echo ciao` → `/bin/sh -c "echo ciao"`
- Command is executed in a shell
- Command don’t receive container signals

## **Exec form** – prefered form for CMD e ENTRYPOINT

- `RUN ["apt-get", "install", "python3"]` → `apt-get install python3`
  `CMD ["/bin/echo", "Hello world"]` → `/bin/echo Hello world`
  `ENTRYPOINT ["/bin/echo", "Hello world"]` → `/bin/echo Hello world`
- Direct esecution of command, no shell
- Command receive container signals

imola informatica

# Dockerfile reference - CMD vs ENTRYPOINT

Both CMD and ENTRYPOINT instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

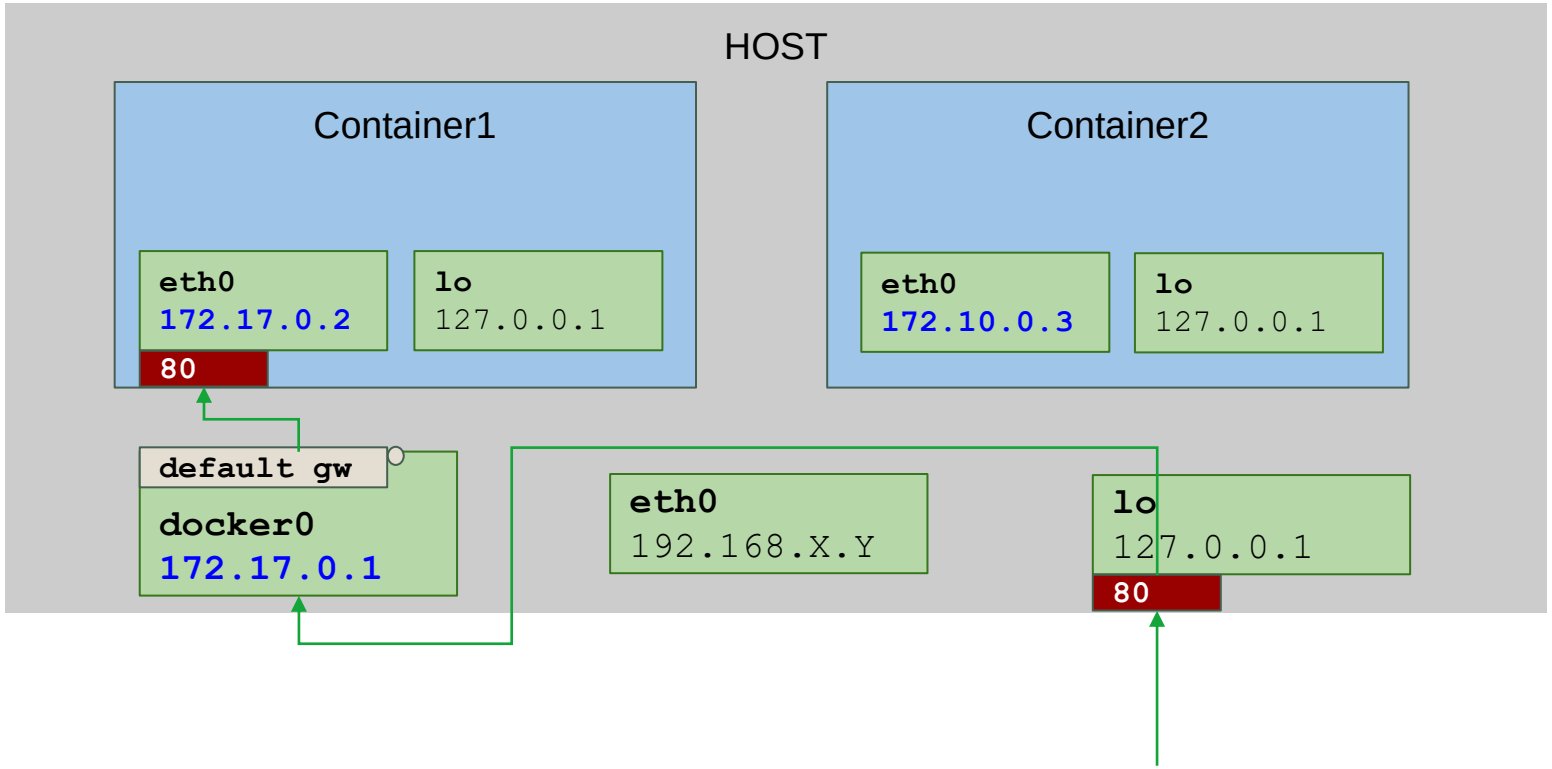
- Dockerfile should specify at least one of CMD or ENTRYPOINT commands.
- ENTRYPOINT should be defined when using the container as an executable.
- CMD should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container.
- CMD will be overridden when running the container with alternative arguments

| | No ENTRYPOINT | ENTRYPOINT [“entry_s1”, “entry_s2”] | ENTRYPOINT entry_s1 entry_s2 |
|---|---|---|---|
| No CMD | error, not allowed | entry_s1 entry_s2 | /bin/sh -c entry_s1 entry_s2 |
| CMD [“cmd_s1”, “cmd_s2”] | cmd_s1 cmd_s2 | entry_s1 entry_s2 cmd_s1 cmd_s2 | /bin/sh -c entry_s1 entry_s2 |
| CMD cmd_s1 cmd_s2 | /bin/sh -c exec_cmd p1_cmd | entry_s1 entry_s2 /bin/sh -c exec_cmd p1_cmd | /bin/sh -c entry_s1 entry_s2 |

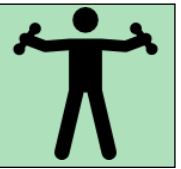
# Docker networking

- docker networking provides full isolation for containers
- isolation can be overwritten to make containers communicate with each other
- docker engine creates **3 default networks**
  - **bridge** → **default network** for containers; points to **docker0** (virtual) network interface
  - **none** → container lacks network interfaces; only **loopback address** is available
  - **host** → adds container to the host network stack
- docker allows users to **create user-defined networks**

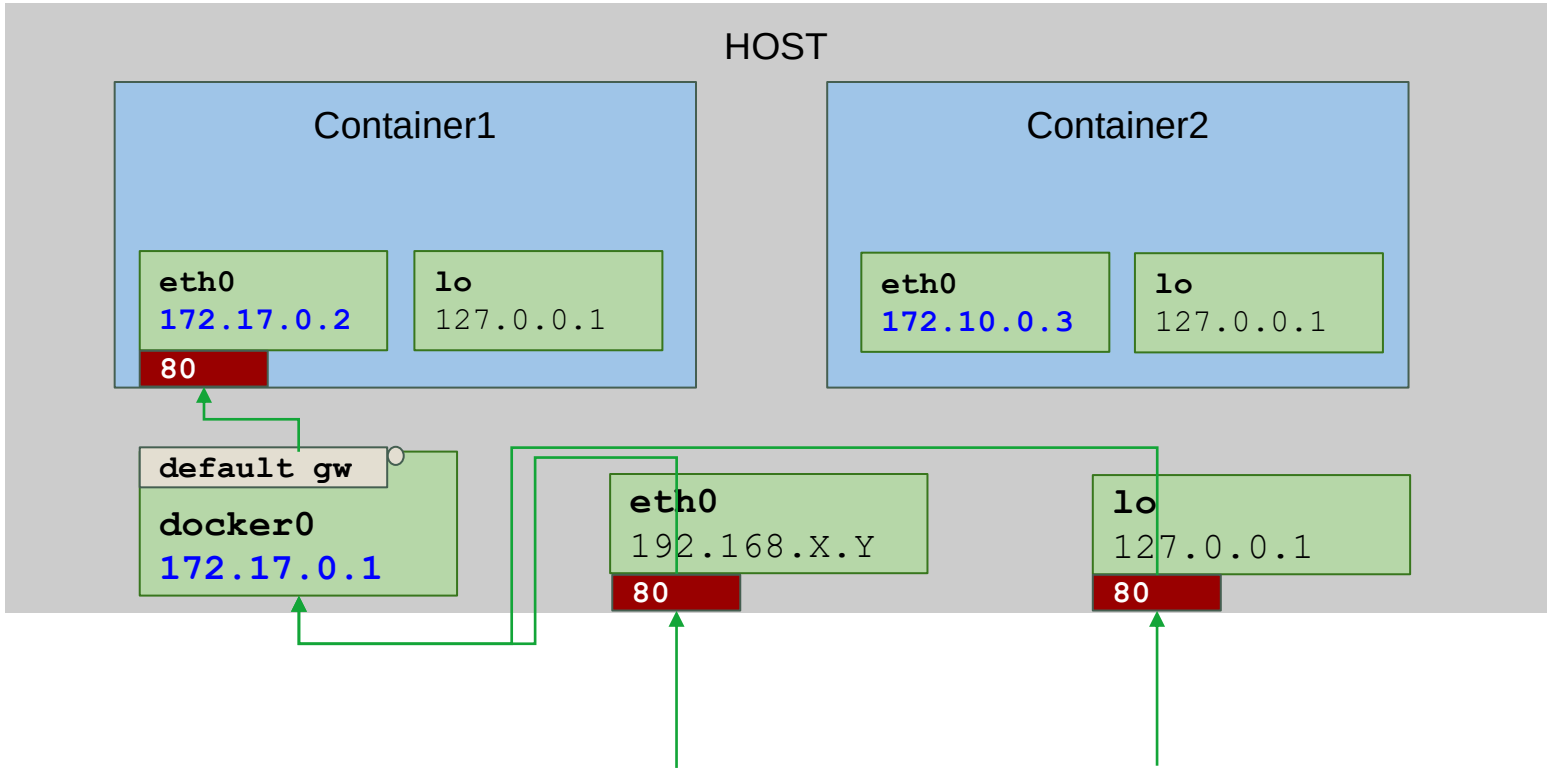
# Docker networking - port forwarding




```
docker run -d -p 127.0.0.1:80:80 httpd:2.4
```



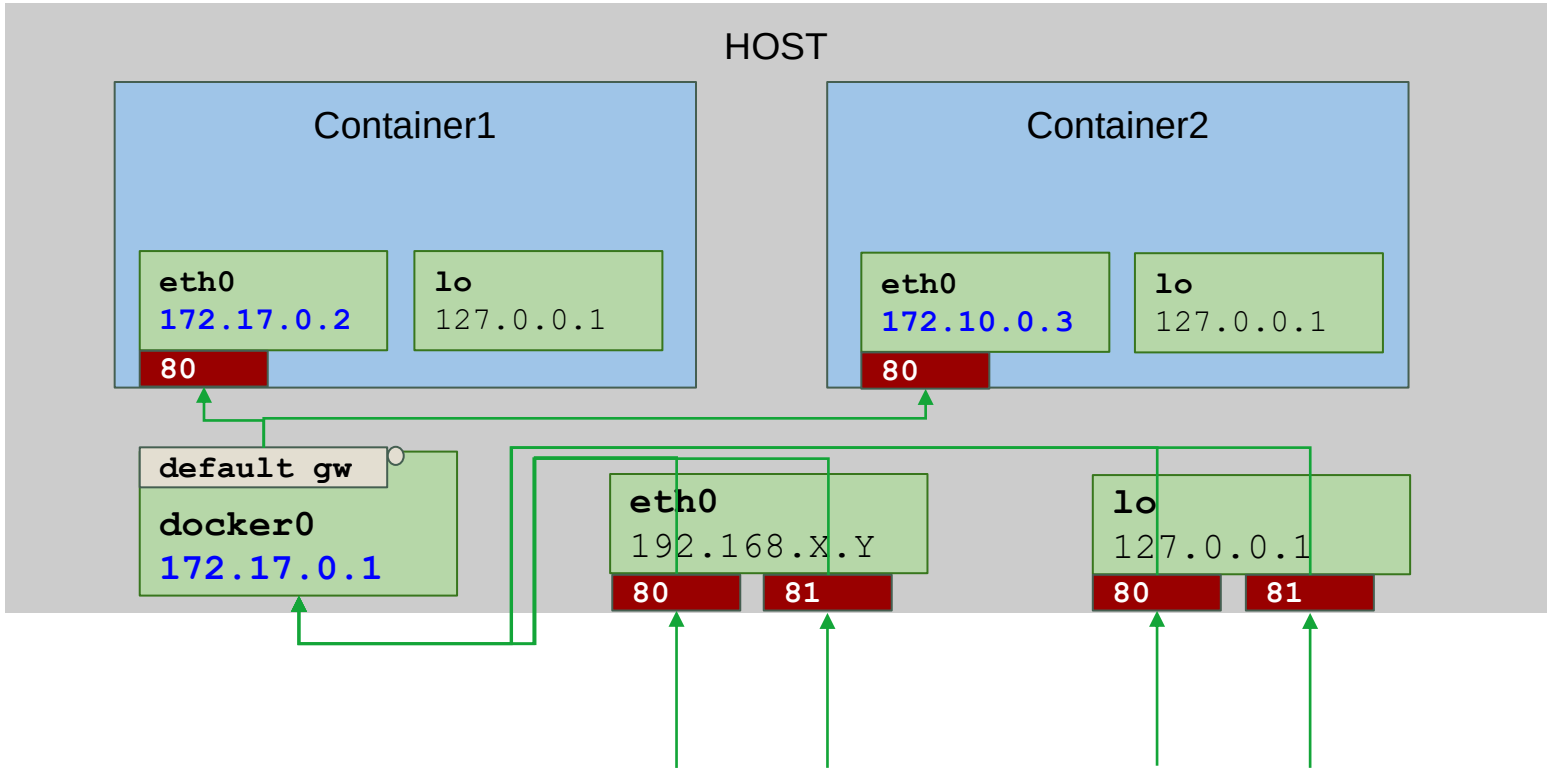
# Docker networking - port forwarding




```
docker run -d -p 80:80 httpd:2.4
```

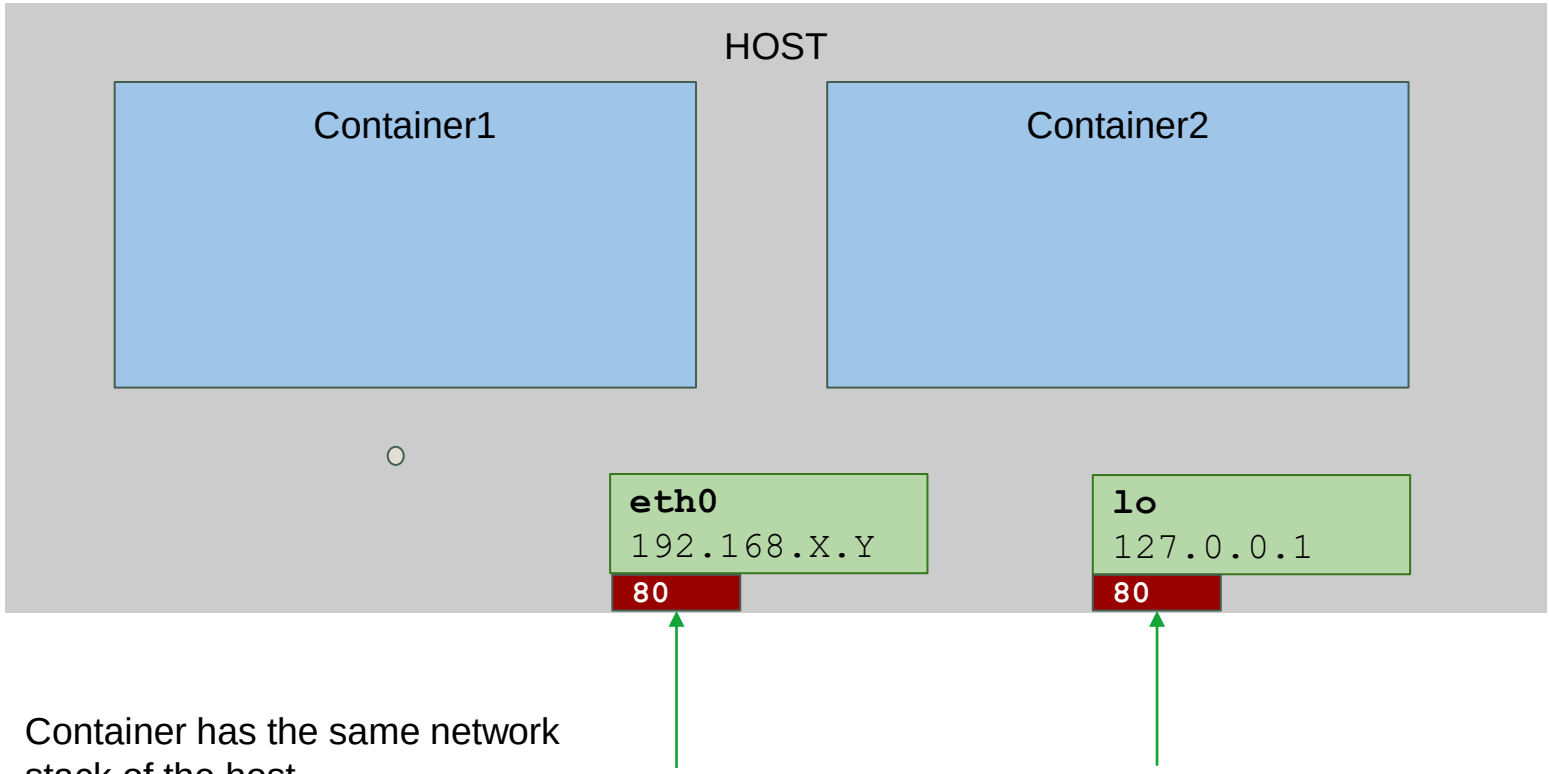
imola informatica

# Docker networking - port forwarding




```
docker run -d -p 80:80 --name container1 httpd:2.4
docker run -d -p 81:80 --name container2 httpd:2.4
```

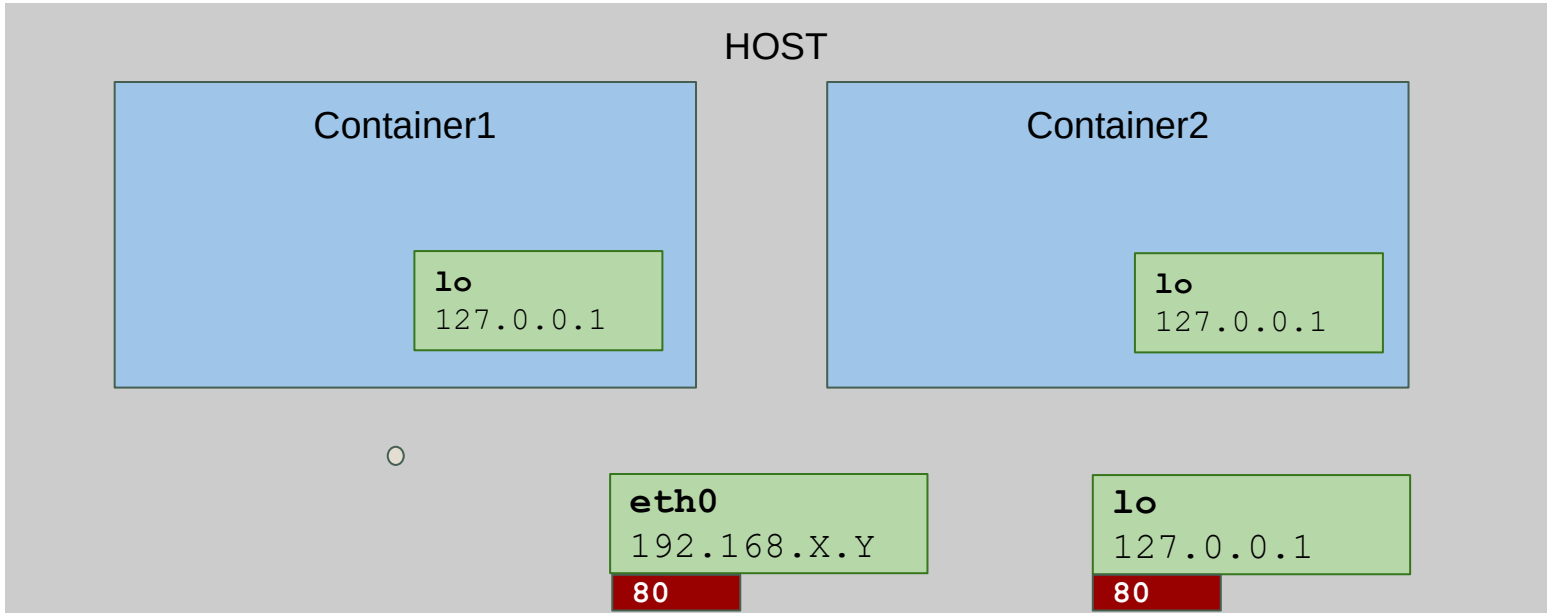
# Docker networking - host



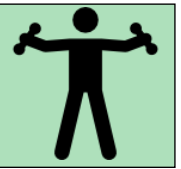

Container has the same network stack of the host

```
docker run -d --network host httpd:2.4
```

# Docker networking - none




```
docker run -d --network none httpd:2.4
```



# Docker Network

```
# From host
docker network ls
docker network inspect networkInterface

# From container
docker exec -it containerId /bin/bash
ip a
```

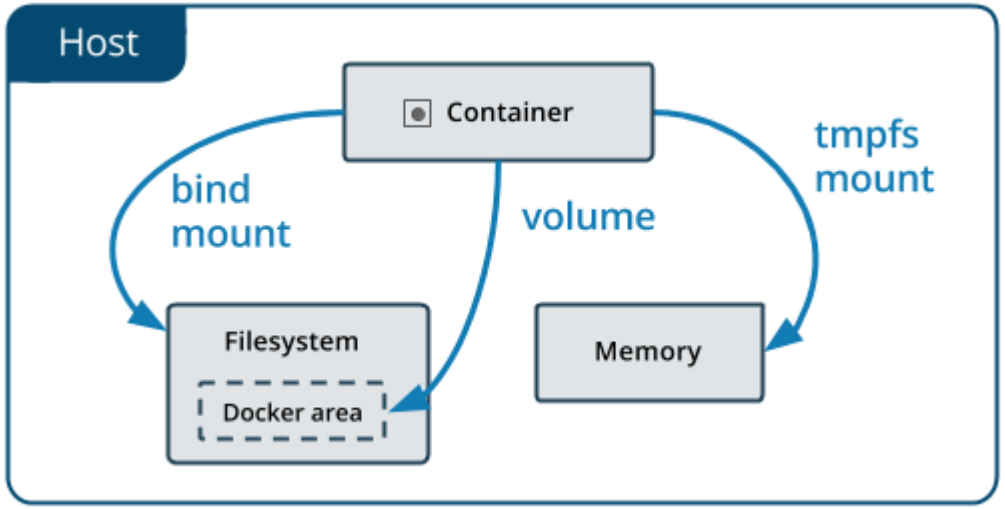
# Container data persistence

**Volumes** are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker. Volumes may be named or anonymous.

**Bind mounts** may be stored anywhere on the host system. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

**tmpfs mounts** are stored in the host system's memory only, and are never written to the host system's filesystem.

**Note:** You can mount even a single file in container filesystem.



# Good use cases for volumes

**Sharing data among multiple running containers**. If you don't explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.

When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you **decouple the configuration of the Docker host from the container runtime**.

When you want to store your container's data on a remote host or a cloud provider, rather than locally.

When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice. You can stop containers using the volume, then back up the volume's directory (such as /var/lib/docker/volumes/<volume-name>).

# Good use cases for bind mounts

**Sharing configuration files from the host machine to containers**. For example Docker by default provides DNS resolution to containers by default, by mounting /etc/resolv.conf from the host machine into each container.

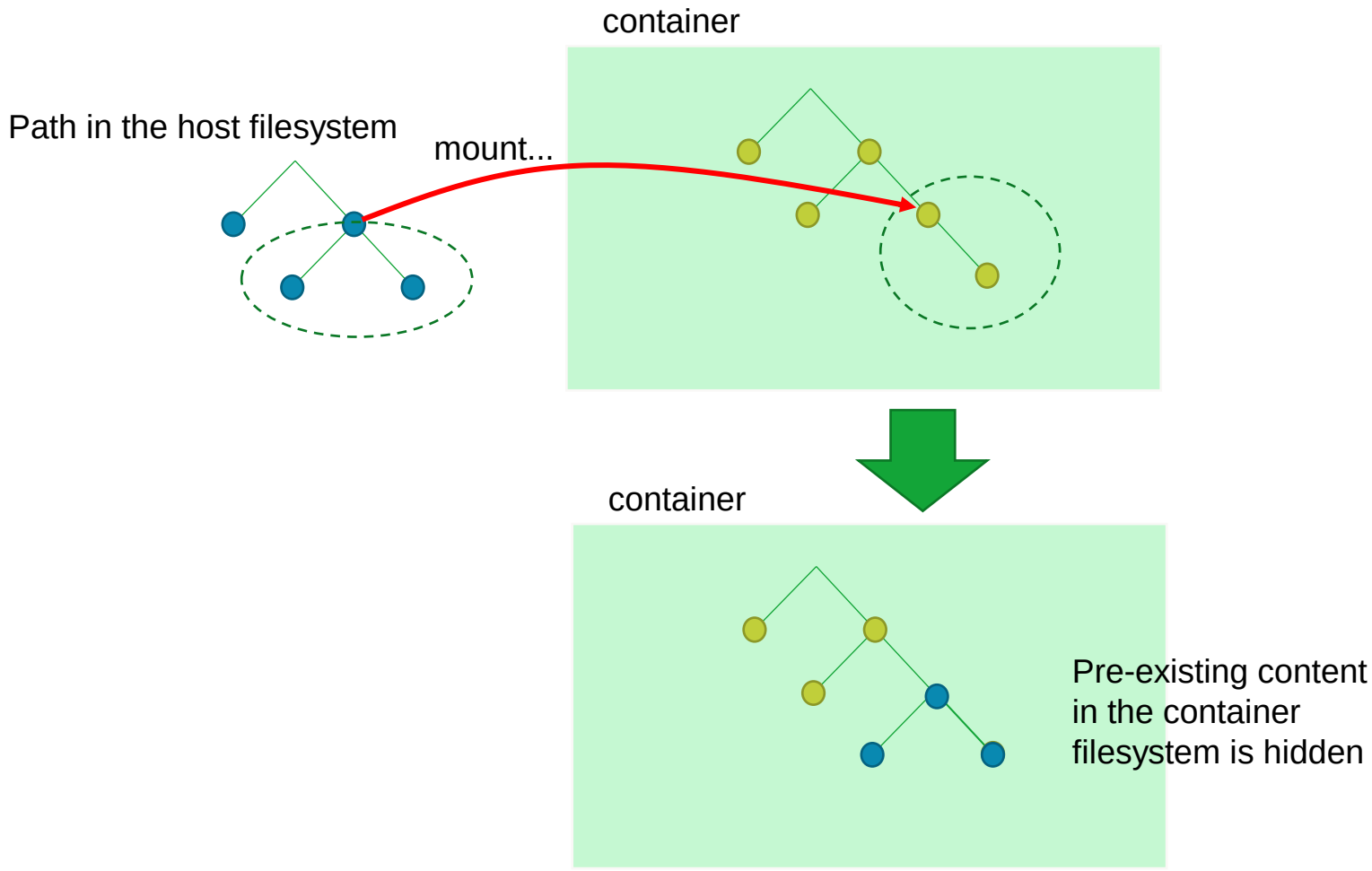
Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts. **Don't use this modality in production environment (embed the artifact in the image).**

When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

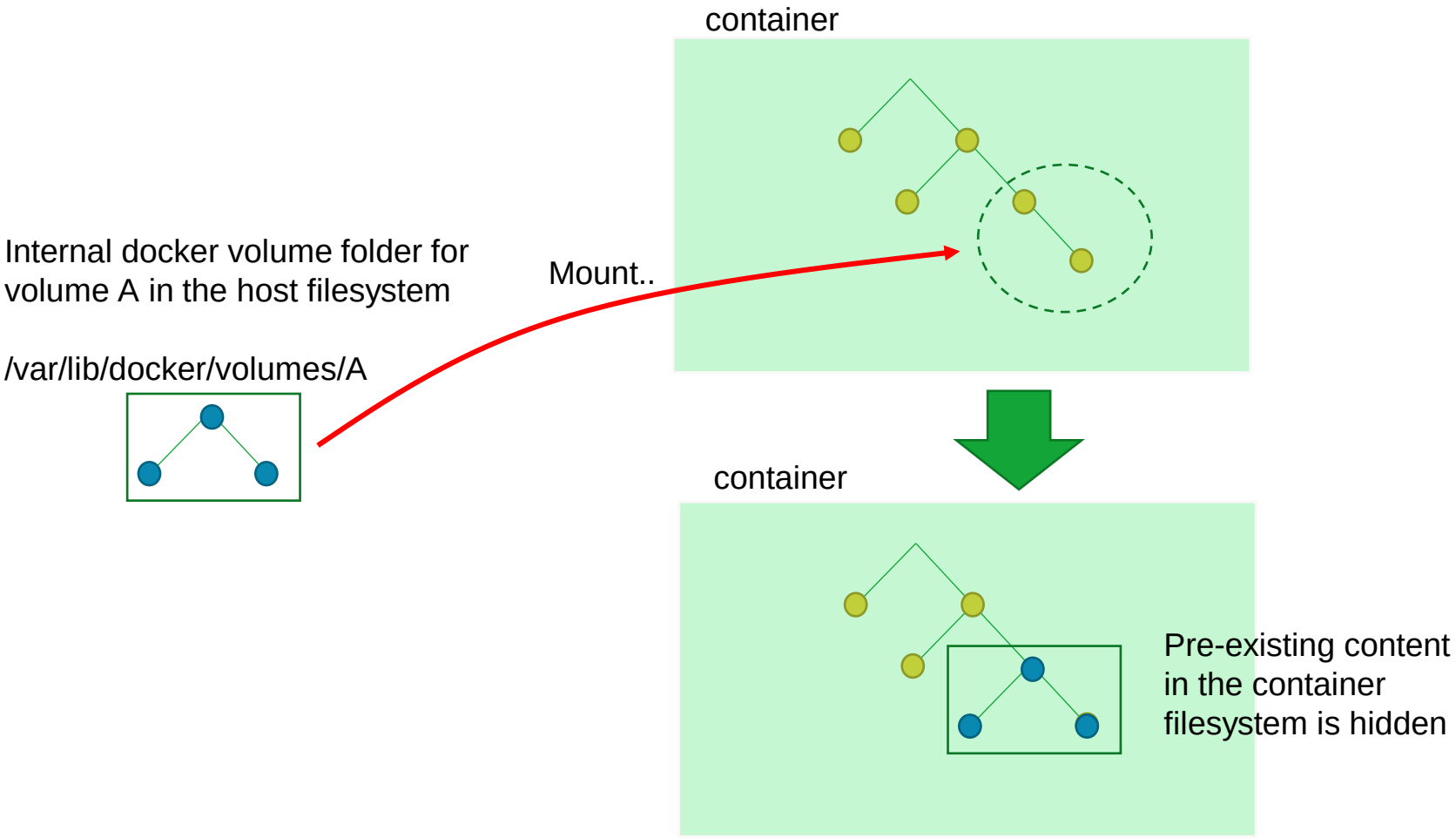
# Good use cases for tmpfs mounts

tmpfs mounts are best used for cases **when you do not want the data to persist either on the host machine or within the container**. This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.

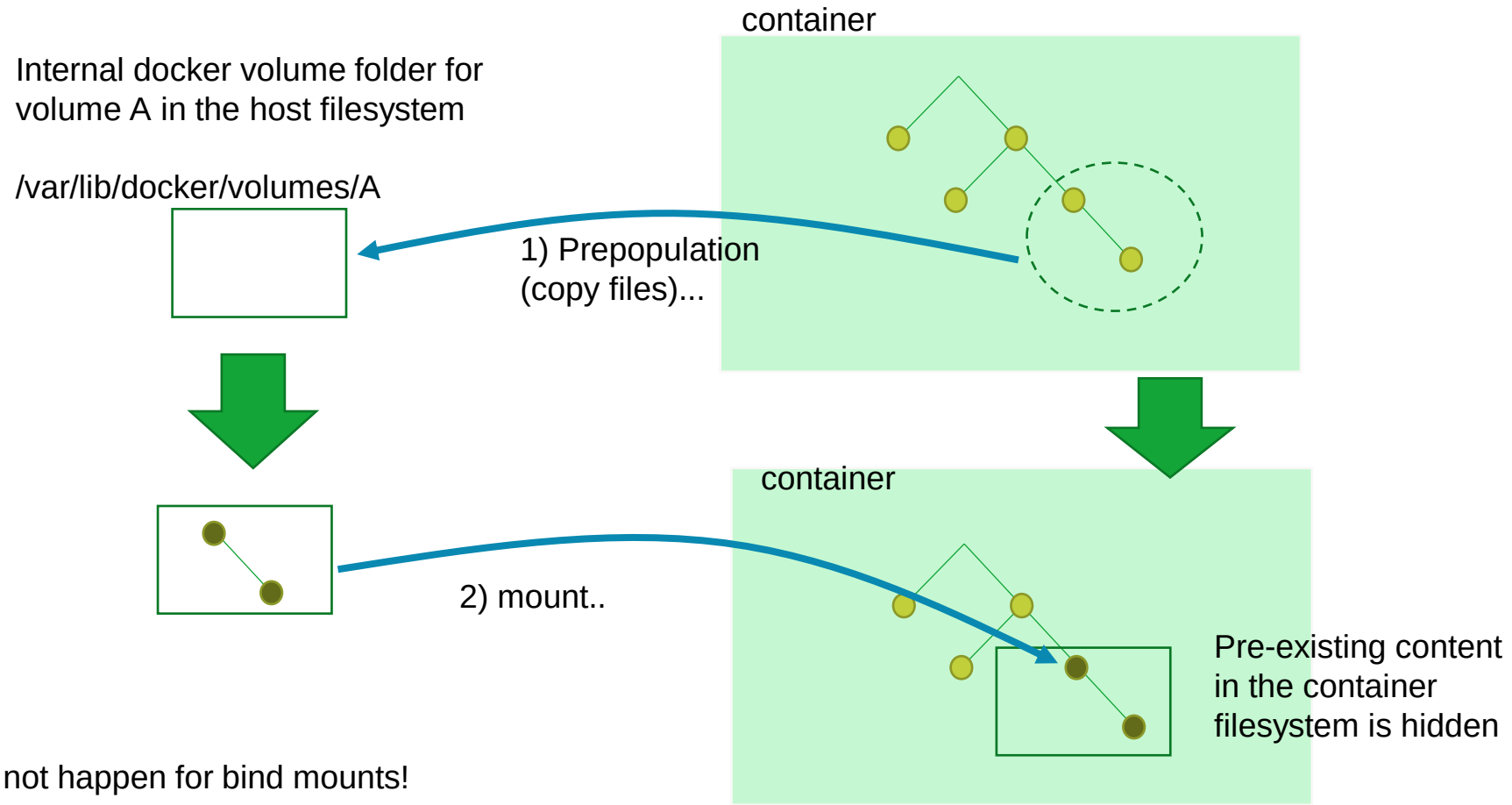
# Bind mount


container
Path in the host filesystem
mount...
container
Pre-existing content
in the container
filesystem is hidden

# Not-empty named/anonymous volume



# Mounting empty named/anonymous volumes




NOTE: prepopulation does not happen for bind mounts!

# Docker volumes - container data persistence

- **Container filesystem** is visible and persistent as long as the container is available (running/stopped/restarted).
- **Docker volumes**
  - can be shared/reused among different containers
  - persist even after container deletion

```
# mounts a specific host directory (usually, in the /var/lib/docker/… FS tree)
# to /webapp mountpoint within the container
docker run -d -v /webapp tomcat:8.0

# mounts /host_fs_folder host directory to /webapp mountpoint within the container
docker run -d -v /host_fs_folder:/webapp tomcat:8.0

# create and mount a named volume
docker volume create tomcat-webapps
docker run -d -v tomcat-webapps:/webapp tomcat:8.0
```

# File permissions

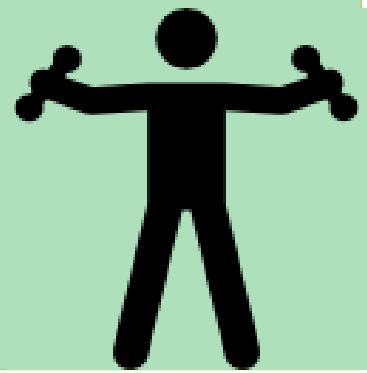
Pay attention to file **owners/group/permissions** (rwx) for the files inside the volumes/bind mounts: the UID/GID of the processes/files inside the containers are usually different from the host

In the Dockerfile

- You may need to chmod/chown file after COPYing them
- You can set the UID inside container with USER directive

# Docker Volume

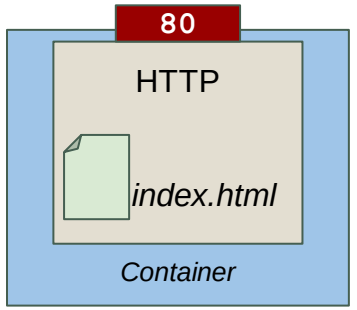
```
docker volume ls
docker volume inspect volumeName


docker volume prune
```



# Agenda

**Docker hands-on**

# Docker - Hands-on




```
cd Docker-Esercizio1
```

**1 - Web Hello World**

**Goals**

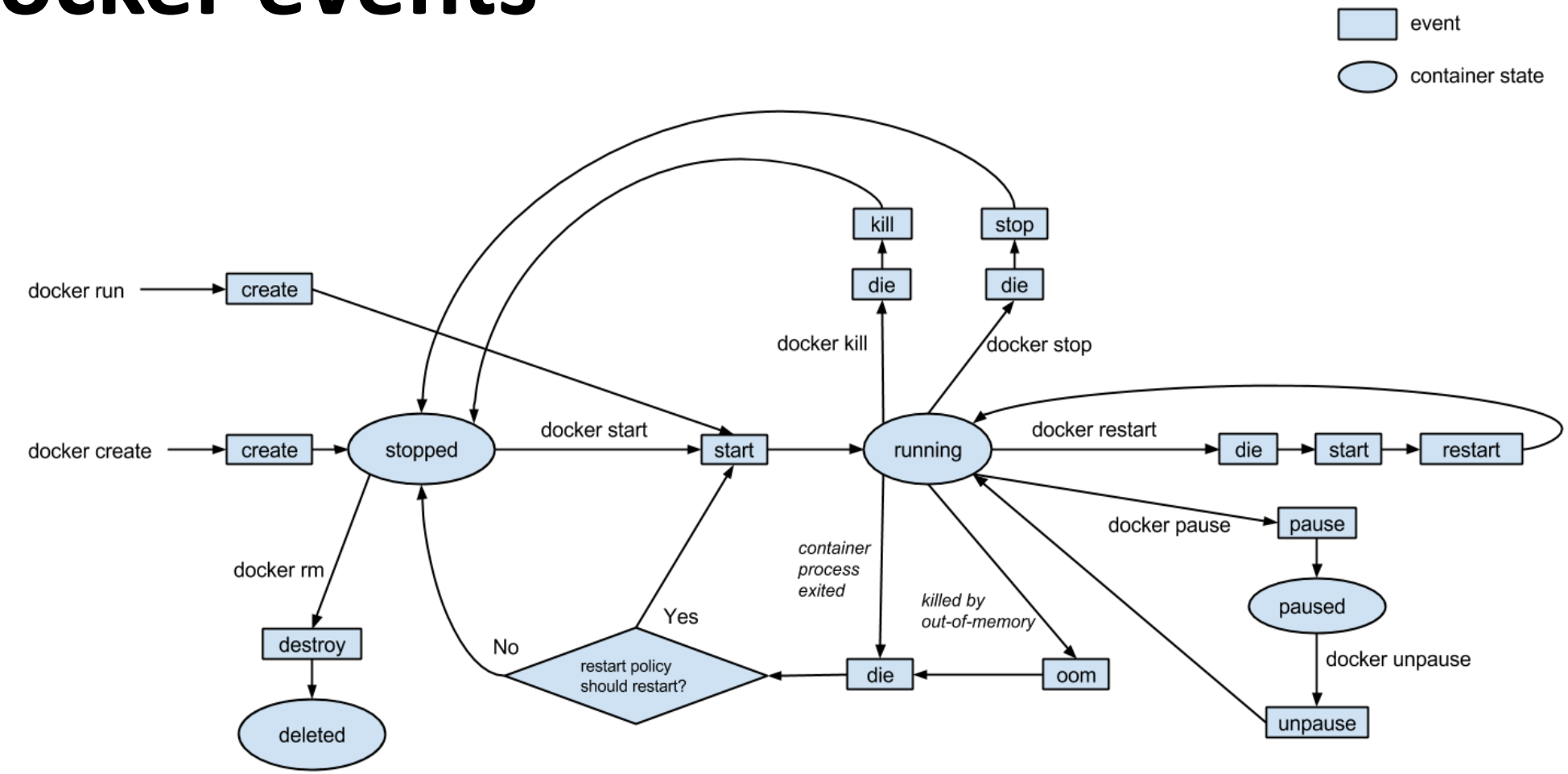
- HTTPD (a.k.a. APACHE) Web Server up and running on standard HTTP port 80, and host-accessible
- the default HTML page (index.html) greets users with a HELLO WORLD

**Hints**

- Docker Hub hosts publicly available images

COPY statement in a Dockerfile allows to copy content from host to container filesystem

# Docker events




```
docker system events
```

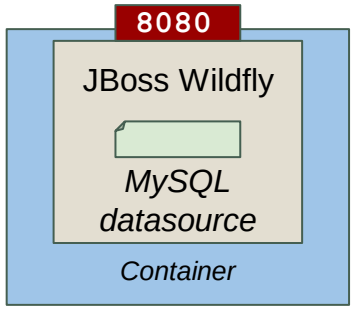
# Docker - Hands-on

**2 - Real-world JEE Application Server**

**Goals**

- JBoss **Wildfly** JEE AS Server up and running on standard HTTP port 8080, and host-accessible
- **MySQL datasource** configured

check datasource connectivity via CLI




```
cd Docker-Esercizio2
```

**Hints**

- Docker Hub hosts publicly available images

default JBoss Wildfly image comes with a stock configuration file that uses an embedded database

→ **example configuration files** are provided in the exercise template

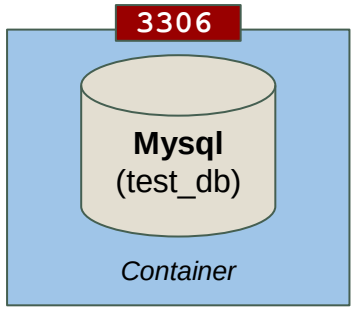
COPY statement in a Dockerfile allows to copy content from host to container filesystem

# Docker - Hands-on

**3 – Mysql DB Server**

**Goals**

- **Mysql** DB Server up and running on standard HTTP port 3306, and host-accessible
- **Database** created
- **Table** created and popolated
- Manage data persistence




```
cd Docker-Esercizio3
```

**Hints**

- Docker Hub hosts publicly available images

Mysql data are stored in **/var/lib/mysql**

When a container is started for the first time will be executed files with extensions .sh, .sql and .sql.gz that are found in /docker-entrypoint-initdb.d

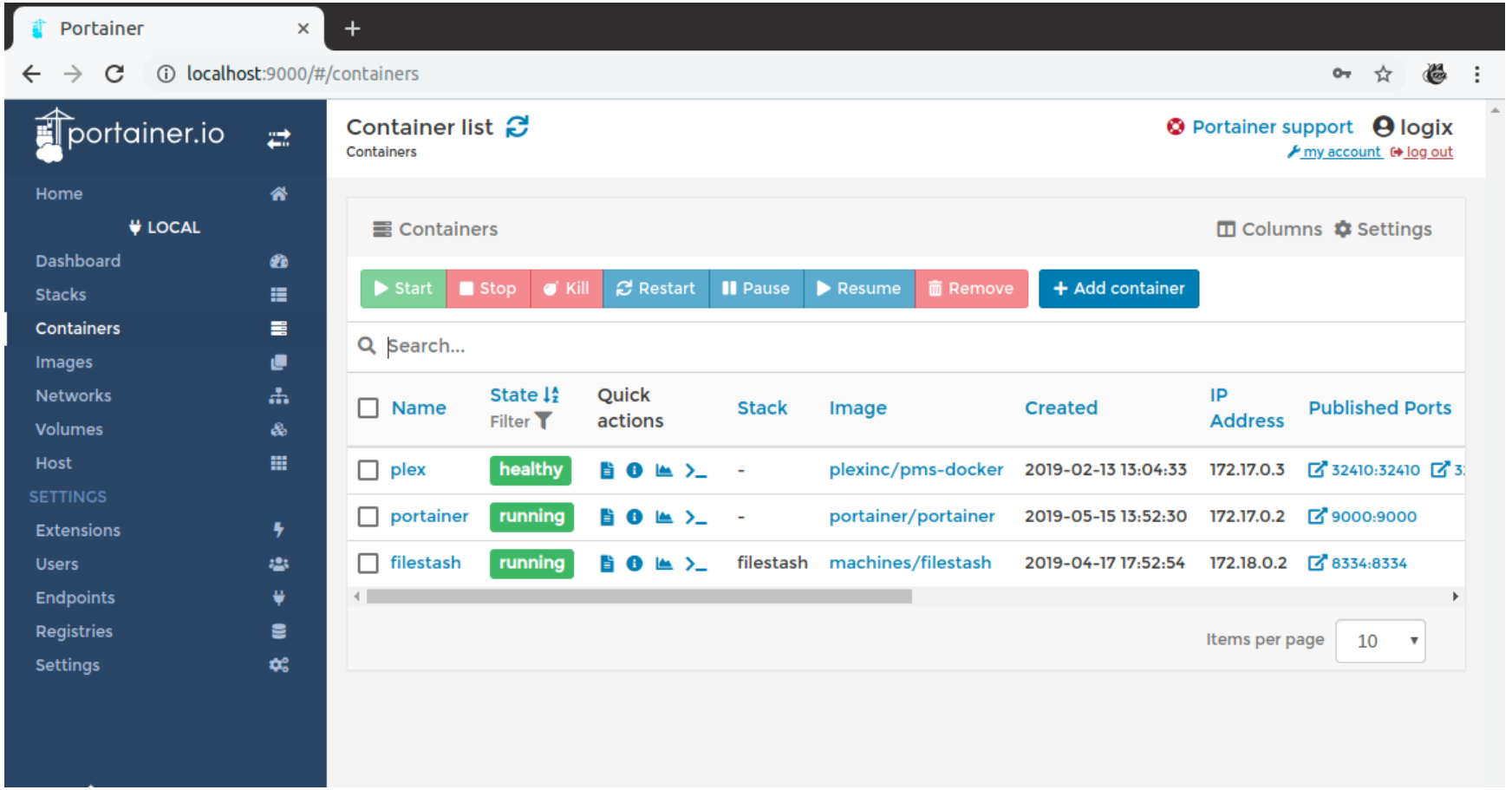
ENV statement in a Dockerfile allows to set environment variable

# Agenda

**Portrainer**

**Web interface**

# Portainer




```
docker run -d -p 9000:9000 --restart always -v /var/run/docker.sock:/var/run/docker.sock -v /opt/portainer:/data portainer/portainer
```

imola informatica

# Debugging running container

```
# Copy file from container
docker cp containerId:containerFilePath hostFilePath

# Copy file into container
docker cp hostFilePath containerId:containerFilePath

# Install tool via yum/apt if package manager is installed into container:
apt update && apt install iputils-ping -y              # ping
apt update && apt install iproute2 -y                  # ip
apt update && apt install net-tools -y                 # netstat

# elsewhere copy file:
docker cp /bin/less containerId:/bin/less              # less
docker cp /bin/ping containerId:/bin/ping              # ping
docker cp /bin/ip containerId:/bin/ip                  # ip
```

**Other techniques: nsenter –** https://stackoverflow.com/a/40352004

imola informatica

# Container Restart policy

```
docker update --restart=flag containerId
```

Available flag:

**no** - Do not automatically restart the container. (the default)

**on-failure** - Restart the container if it exits due to an error, which manifests as a non-zero exit code.

**unless-stopped** - Restart the container unless it is explicitly stopped or Docker itself is stopped or restarted.

**always** - Always restart the container if it stops.

# Lorenzo Patera

IT Consultant

lpatera@imolainformatica.it

@LorenzoPatera

lorenzopatera

imola informatica

imola informatica

IMOLA · BOLOGNA · MILANO

www.imolainformatica.it | blog.imolainformatica.it

# Introduzione alla blockchain

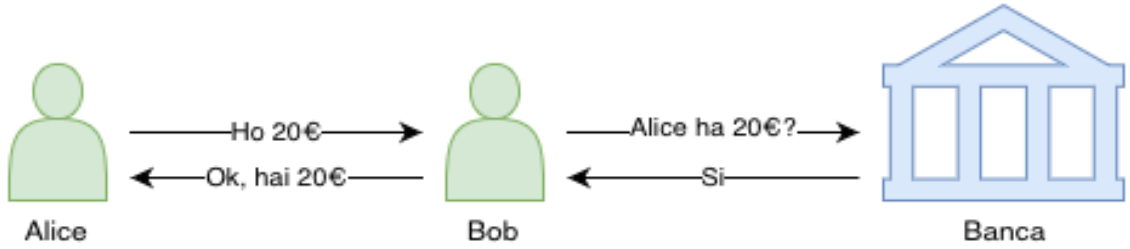
imola informatica

25/11/2020

# Il problema della fiducia

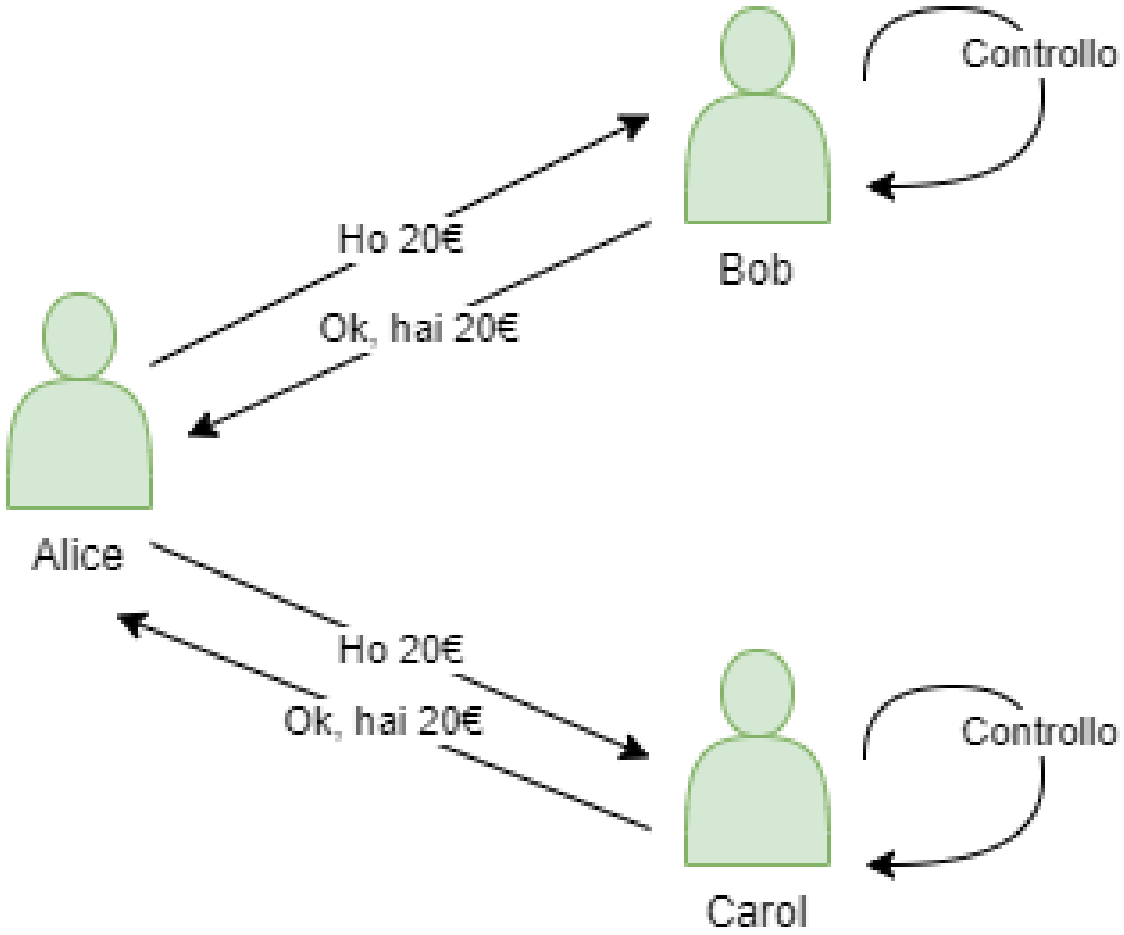
Modello di scambio monetario
basato su **terze parti fidate**:

- Opacità
- Costi
- Ritardi
- Errori
- Problemi di proprietà dei dati



# Distributed Ledger Technology

- Distribuito
- Paritario
- Contenuto concordato
- Trasparente

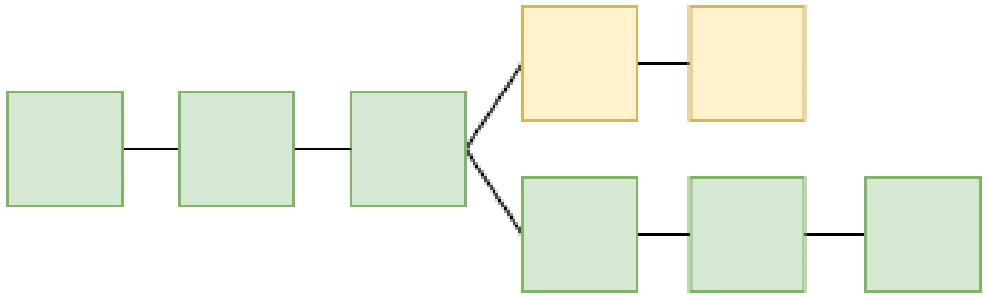


# Caratteristiche dei DLT

| | | Distributed | | Localized |
|---|---|---|---|---|
| | | Decentralized | Centralized | Centralized |
| Permissioned | Tokenized | Pointless | Appl. unclear | WoW Gold |
| | Tokenless | | Financial appl. | Wikipedia |
| Permissionless | Tokenized | Bitcoin, Ethereum | Ripple, Stellar | Appl. unclear |
| | Tokenless | Subject to spam | | |

# Blockchain

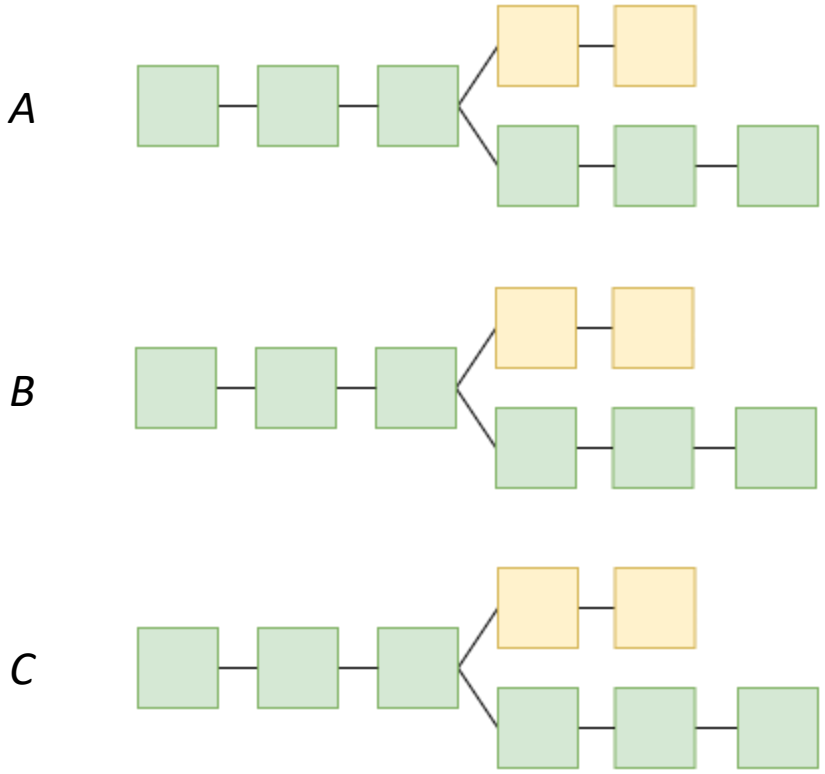
- Dati raggruppati in blocchi
- Blocchi concatenati tramite funzioni matematiche
- Rappresentazione di **tutto** lo storico di transazioni



Rappresentazione grafica blockchain Bitcoin

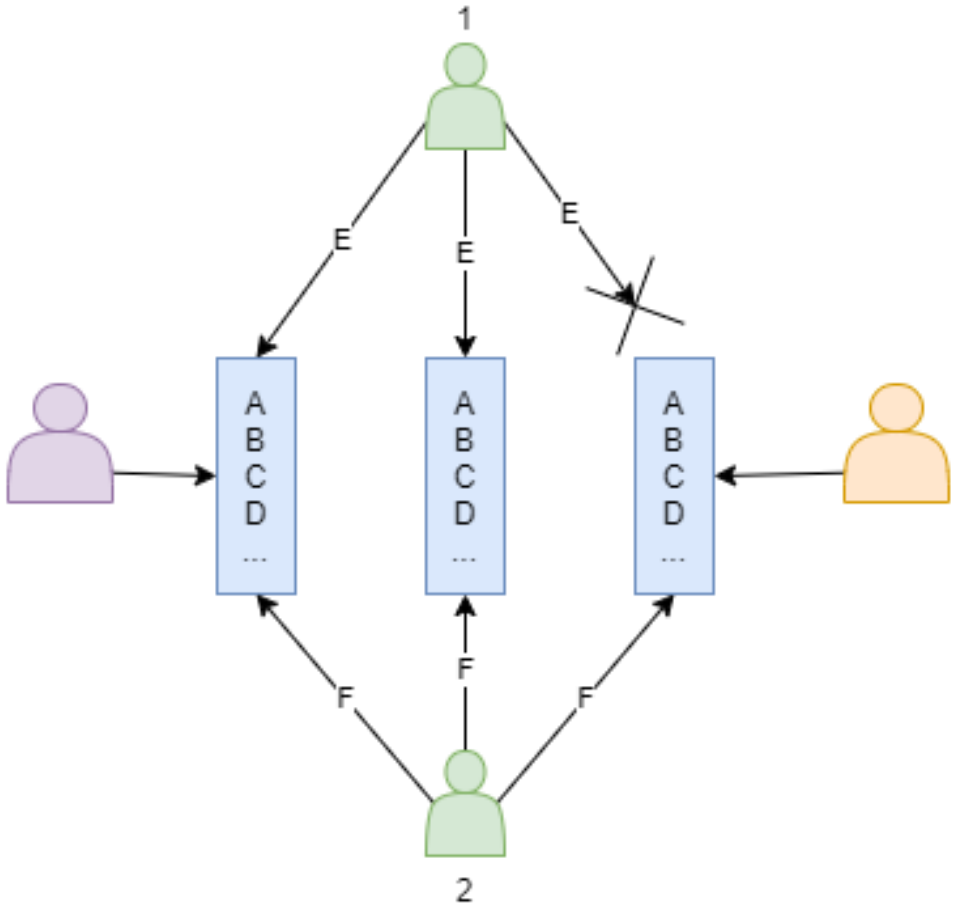
# Blockchain e fiducia

- Più copie della blockchain
- Copie distribuite
- Copie accessibili direttamente
- Consenso sulla validità delle copie



# Il problema del consenso

- Transazioni **incompatibili**
- Ritardi
  nella comunicazione tra
  copie
- Necessità di un
  meccanismo per ottenere
  **consenso**



# Funzioni hash

```
H("imola")       = 88ca0da7398e805a7d7cc167598558abcf61c43d

H("informatica") = f0879b5d26a224f963e51cc543555e00a1c34205
```

**Proprietà:**

- **Assenza di collisioni**, ovvero due diversi valori di input non devono portare allo stesso output
- **Non invertibilità**, ovvero deve essere computazionalmente impossibile risalire all'input partendo dall'output

Approfondimento su Wikipedia

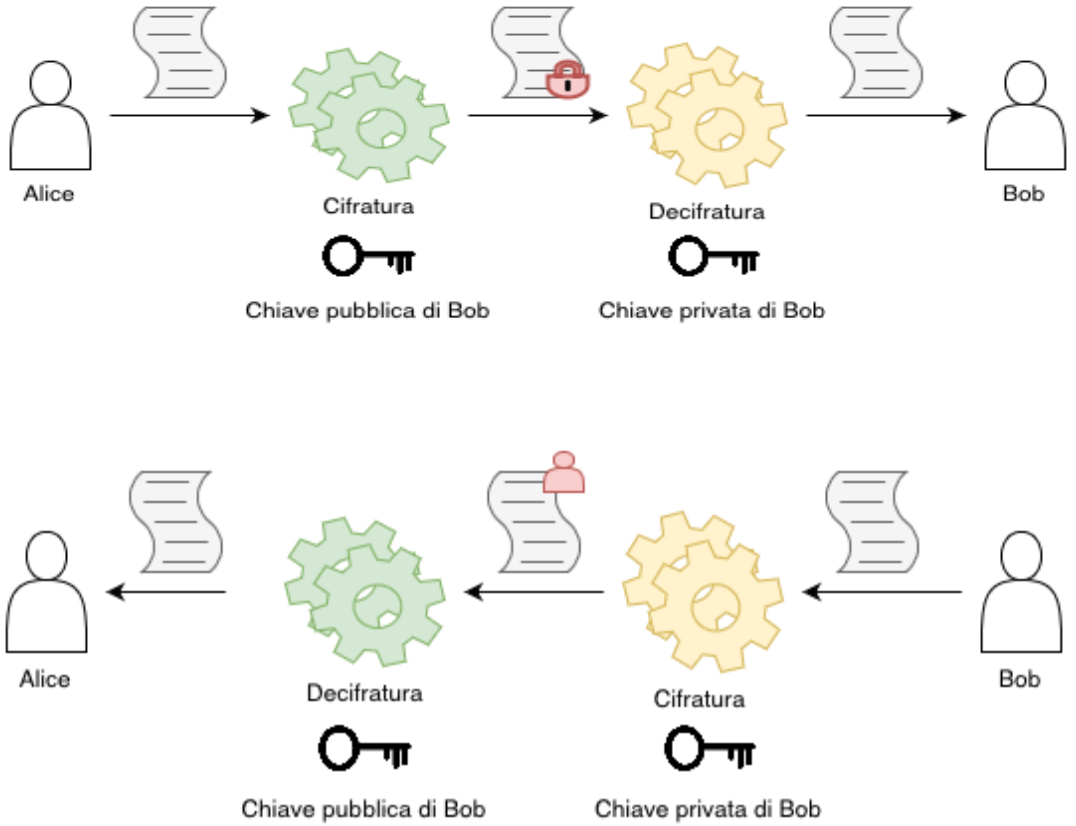
# Crittografia asimmetrica

Due **Tipologie** di chiavi:

- La chiave pubblica, che deve essere distribuita
- La chiave privata, da mantenere segreta

**Operazioni**:

- Cifrare messaggi con la chiave pubblica del destinatario per garantire **confidenzialità**
- Cifrare messaggi con la propria chiave privata per garantire **autenticazione**



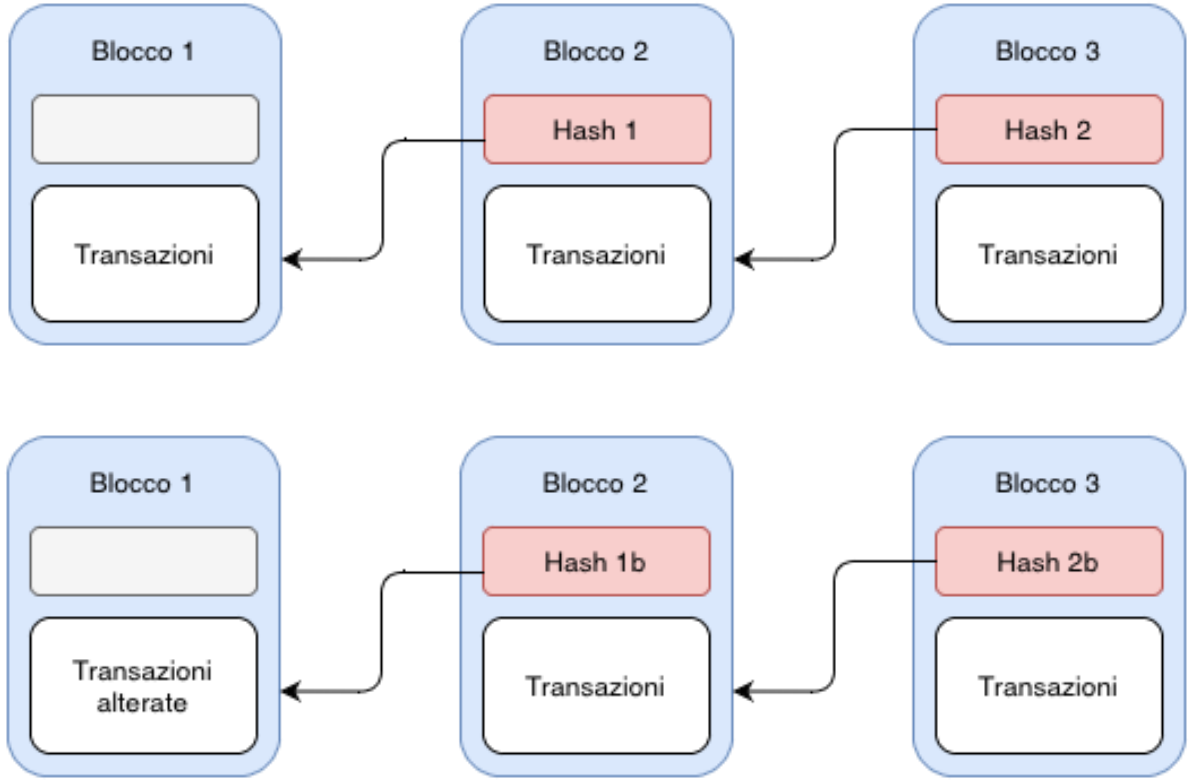

Approfondimento su Wikipedia

# Struttura di una blockchain

Ogni **blocco** della catena contiene:

- Da 1 a N transazioni
- L'hash del blocco precedente
- Metadati variabili

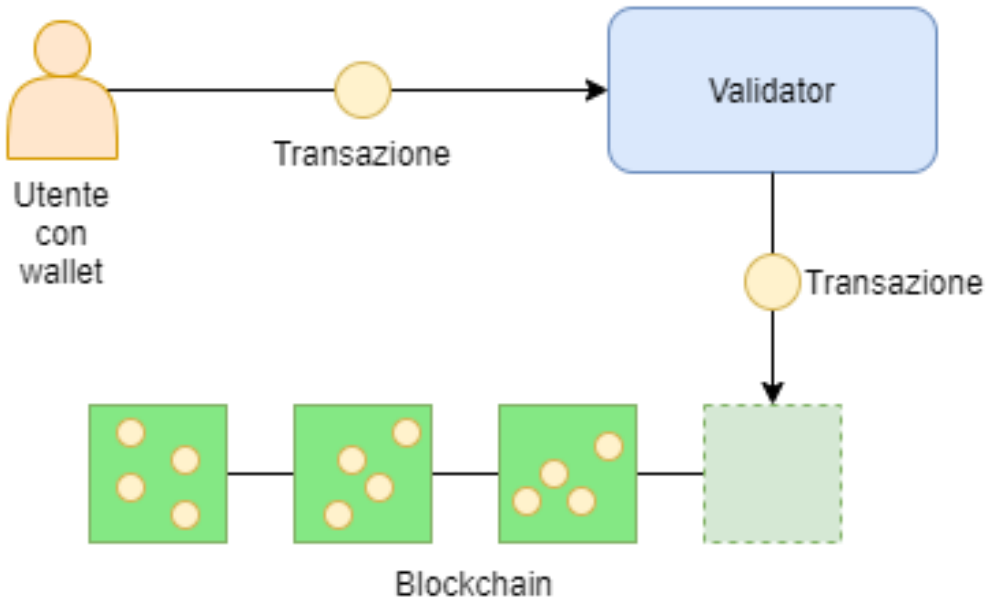
Una volta inserito nella catena, un blocco risulta praticamente **immutabile**.



# Principi di funzionamento

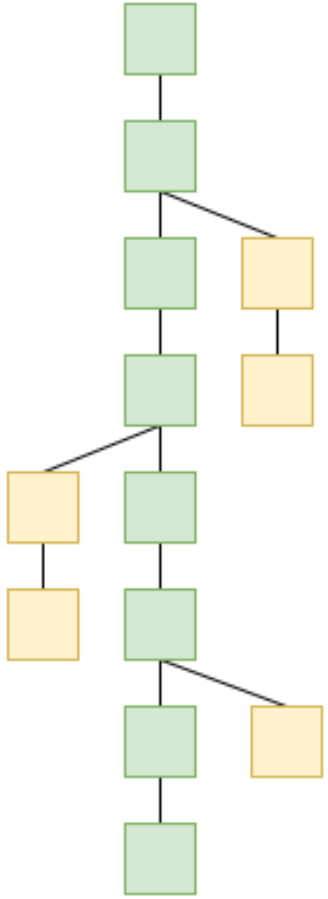
La blockchain ha due diversi **attori**:

- Coloro che effettuano transazioni con un **wallet**
- Coloro che validano le transazioni e creano i nuovi blocchi



# Algoritmo di consenso

- Dipende dall'implementazione
- Permette di accordarsi su quale sia la diramazione della catena valida
- Si basa su un meccanismo di validazione



# Meccanismi di validazione delle transazioni

**Proof of Work**, basato sulla potenza computazionale messa a disposizione

**Proof of Stake**, basato sulla quantità di token che si possiedono


Per approfondire

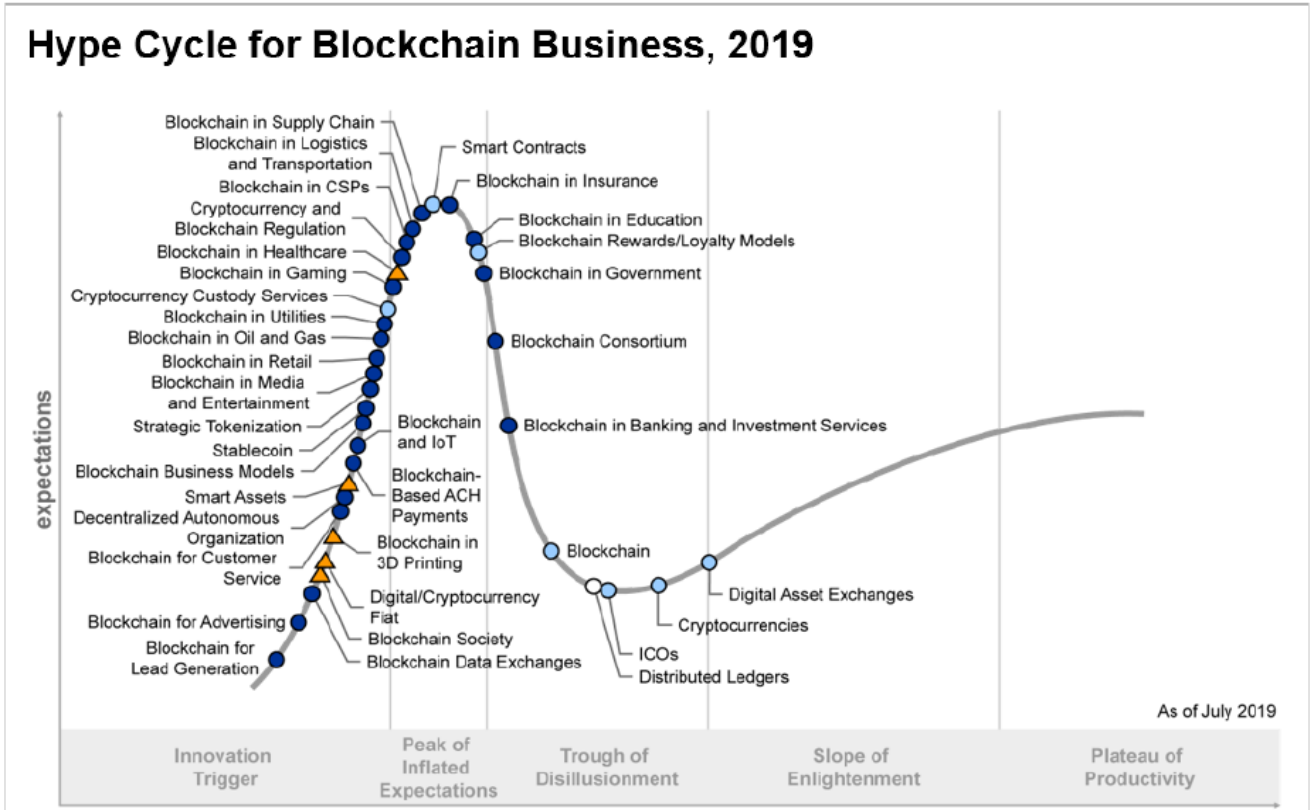
# PoW e PoS a confronto

| | Proof of Work | Proof of Stake |
|---|---|---|
| Aumento richiesta potenza computazionale | Si | No |
| Scalabilità (troughput di transazioni) | Bassa | Media |
| Rischio di centralizzazione | Nullo | Alto |
| Attacco del 50% + 1 | Basato sul possesso di nodi | Basato sul possesso di token |

# Punti di dibattito sulla blockchain

I principali punti di dibattito sulla blockchain sono:

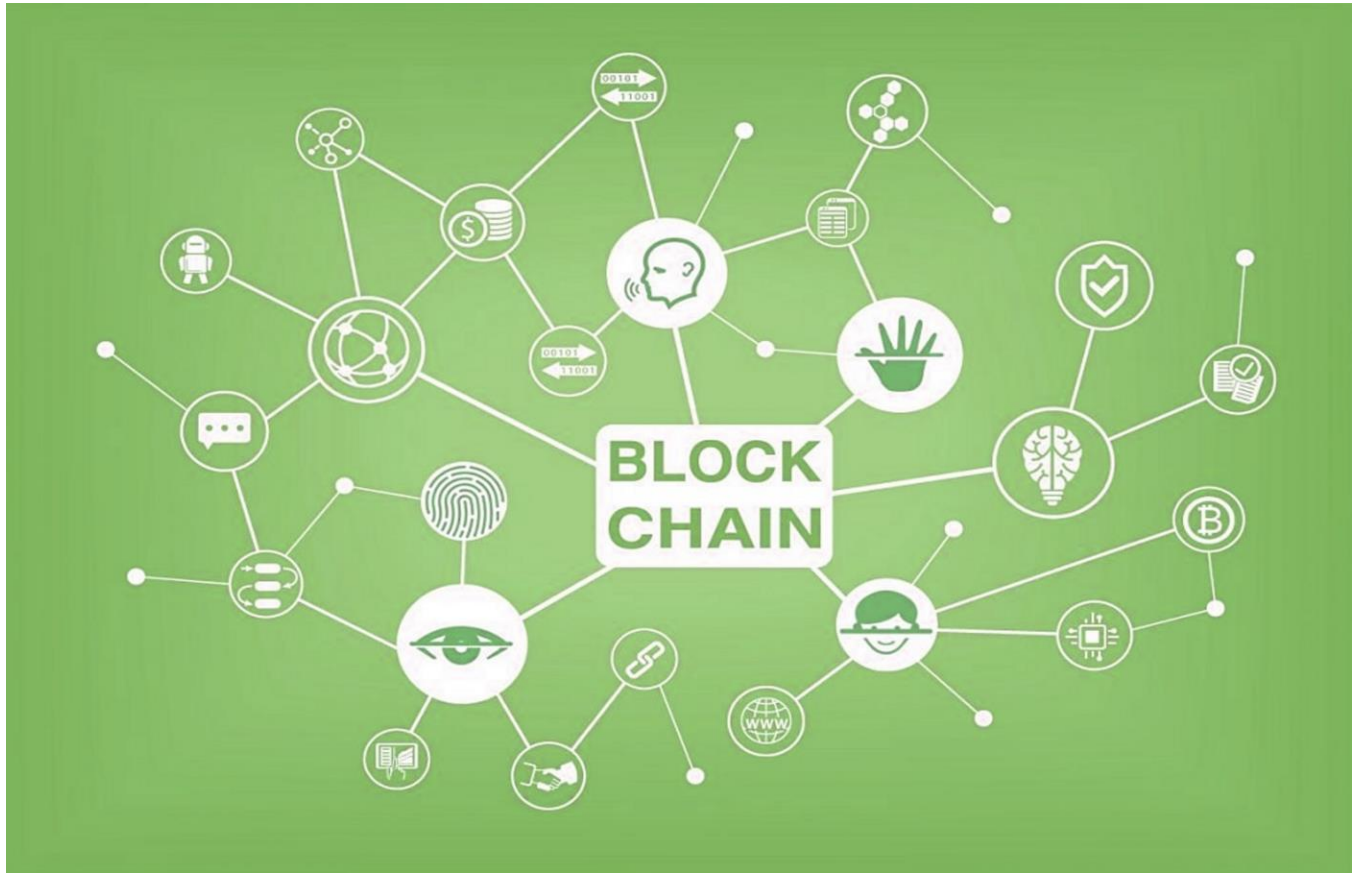
- Scalabilità (dimensioni del registro, throughput delle transazioni, costi in funzione del traffico)
- Costo ambientale del mining
- Sicurezza
- Privacy
- Hype




fonte immagine

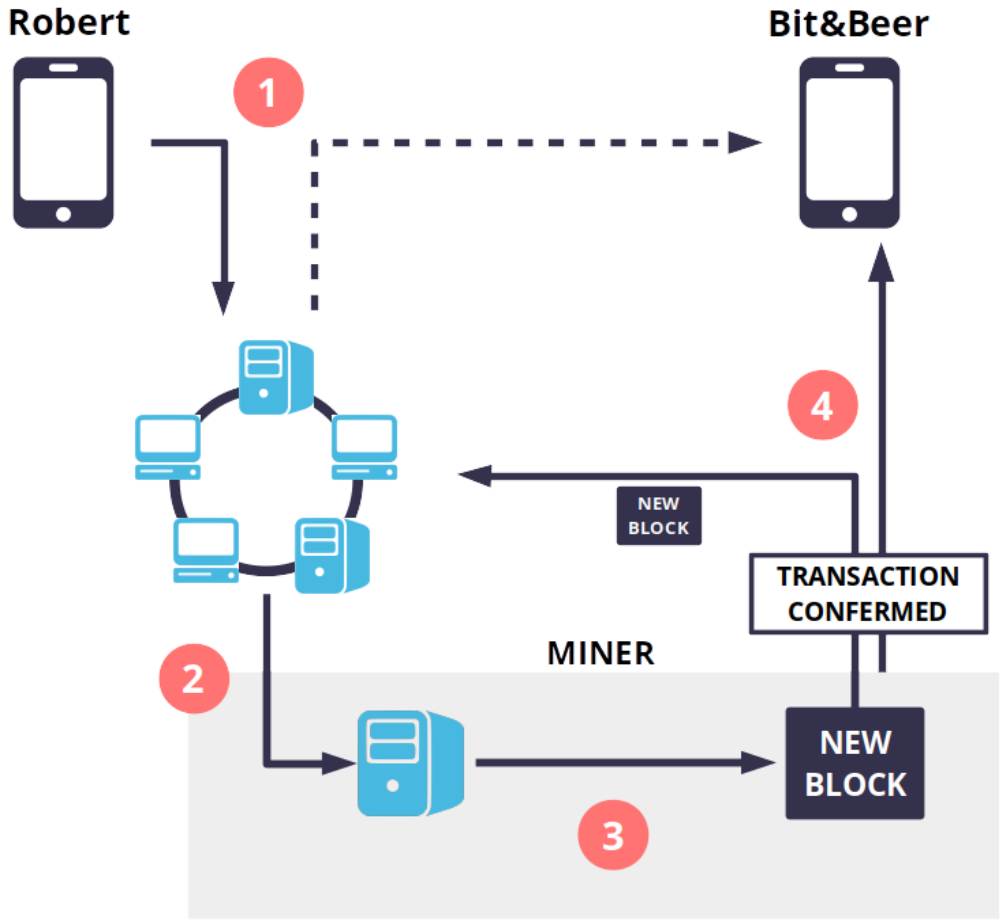
# Possibili applicazioni

- Criptovalute
- Smart Contract
- Distributed File System
- Self Sovereign Identity
- Notarizzazione



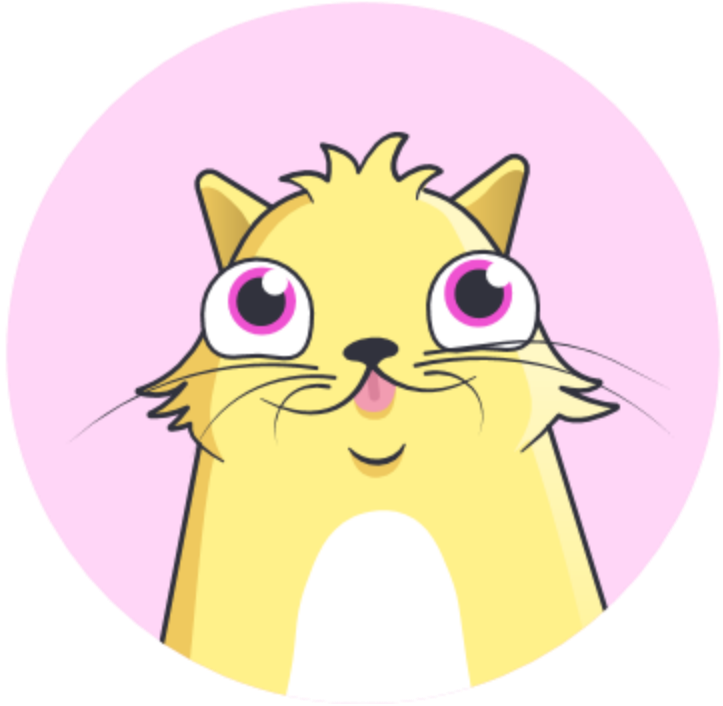
# Bitcoin e criptovalute

- Prima criptovaluta
- Inizialmente descritta e implementata da **Satoshi Nakamoto**
- Scopo finanziario
- Proof of Work



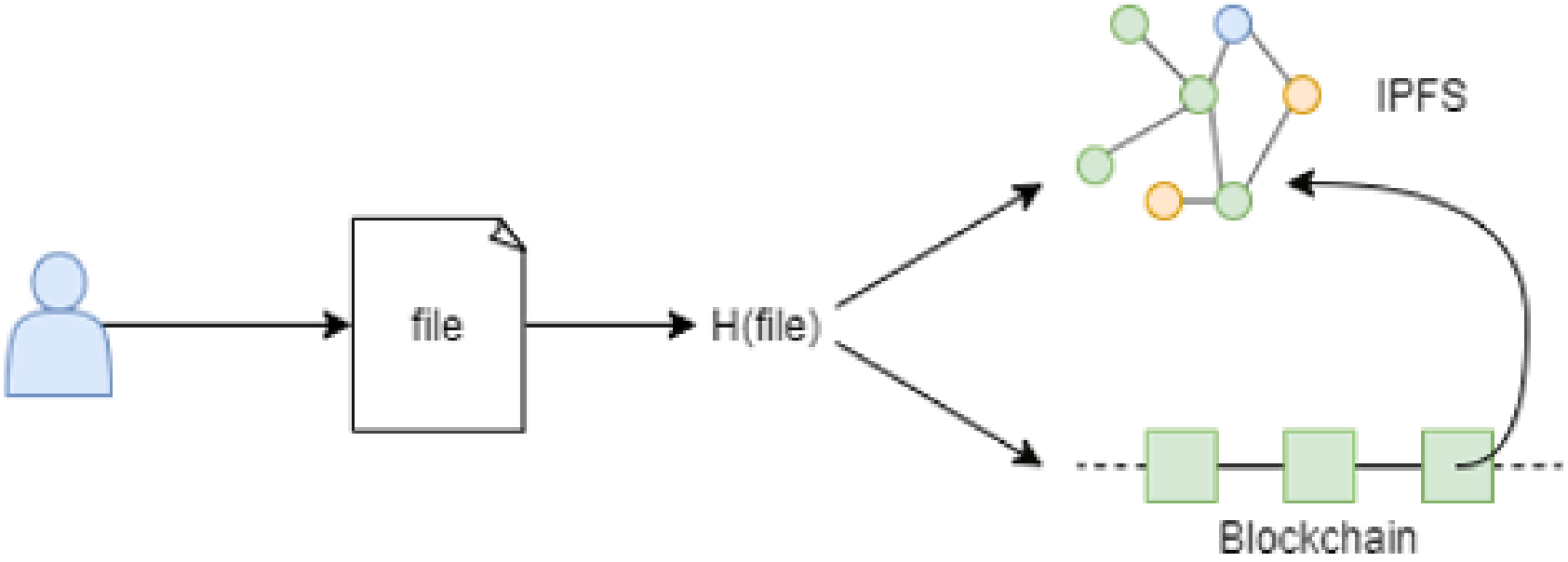
# Ethereum e smart contracts

- Posteriore a Bitcoin
- Maggiore potere espressivo
- **Smart contracts** con linguaggio Turing-completo
- Previsto passaggio a Proof of stake



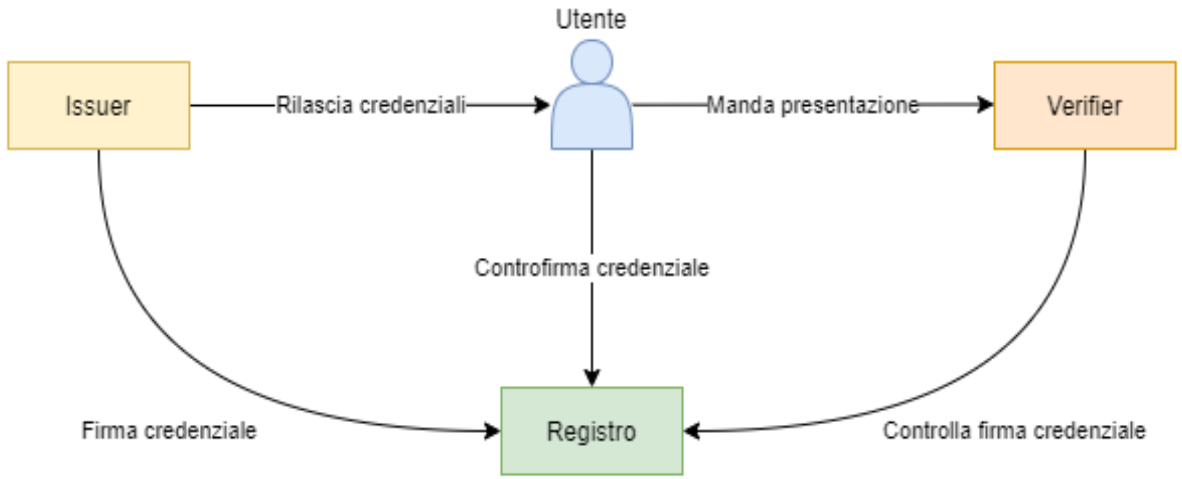
Cryptokitties

# InterPlanetary File System



# Self Sovereign Identity

- Controllo completo sulla propria identità digitale
- Rilascio selettivo di informazioni
- Registrabile su blockchain e altri DLT



# Notarizzazione su blockchain: è legale?

## In Italia:

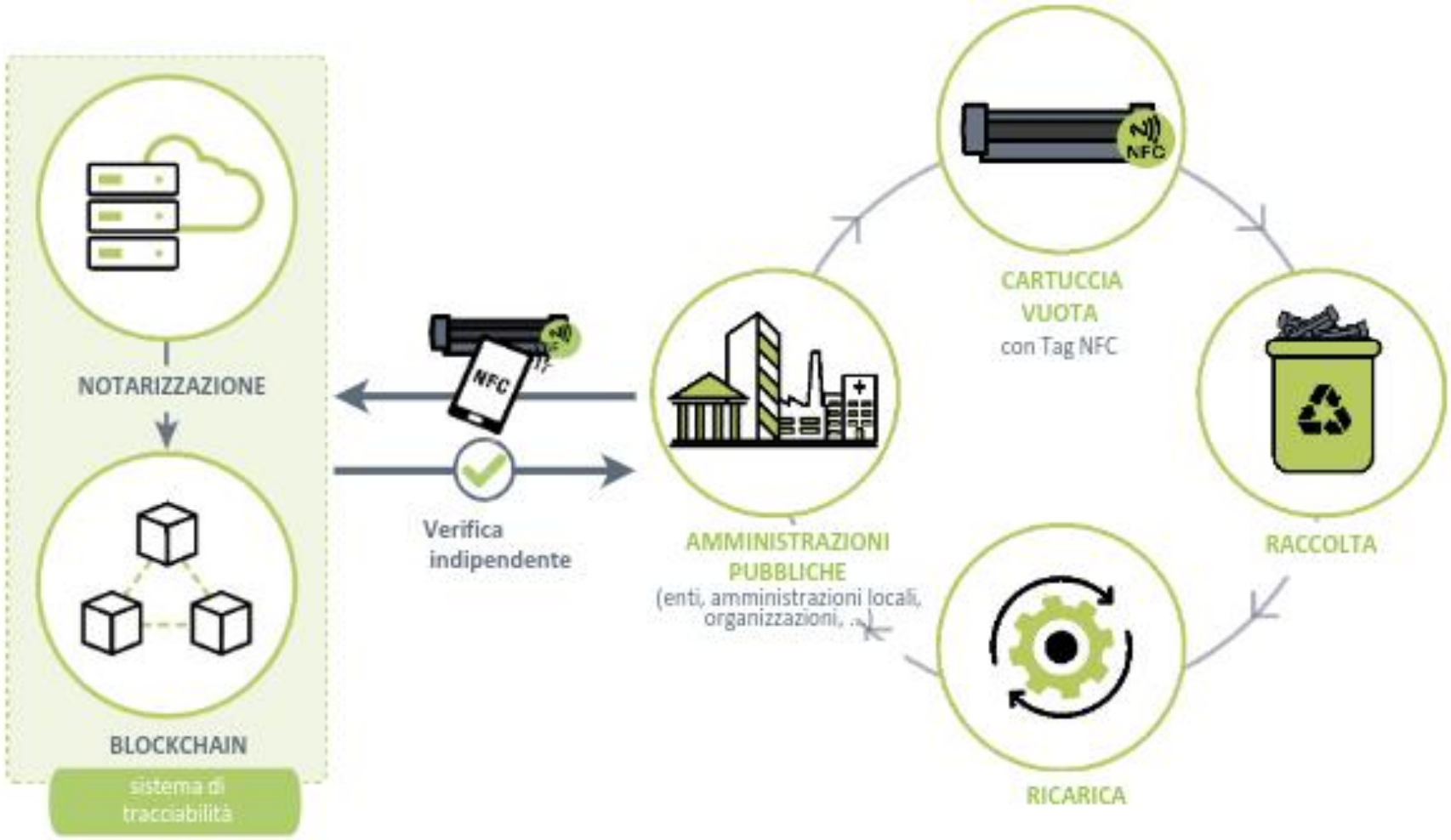
Articolo 8 *ter* del Decreto legge 135/2018 DL Semplificazioni, Gazzetta Ufficiale 12 Febbraio 2019

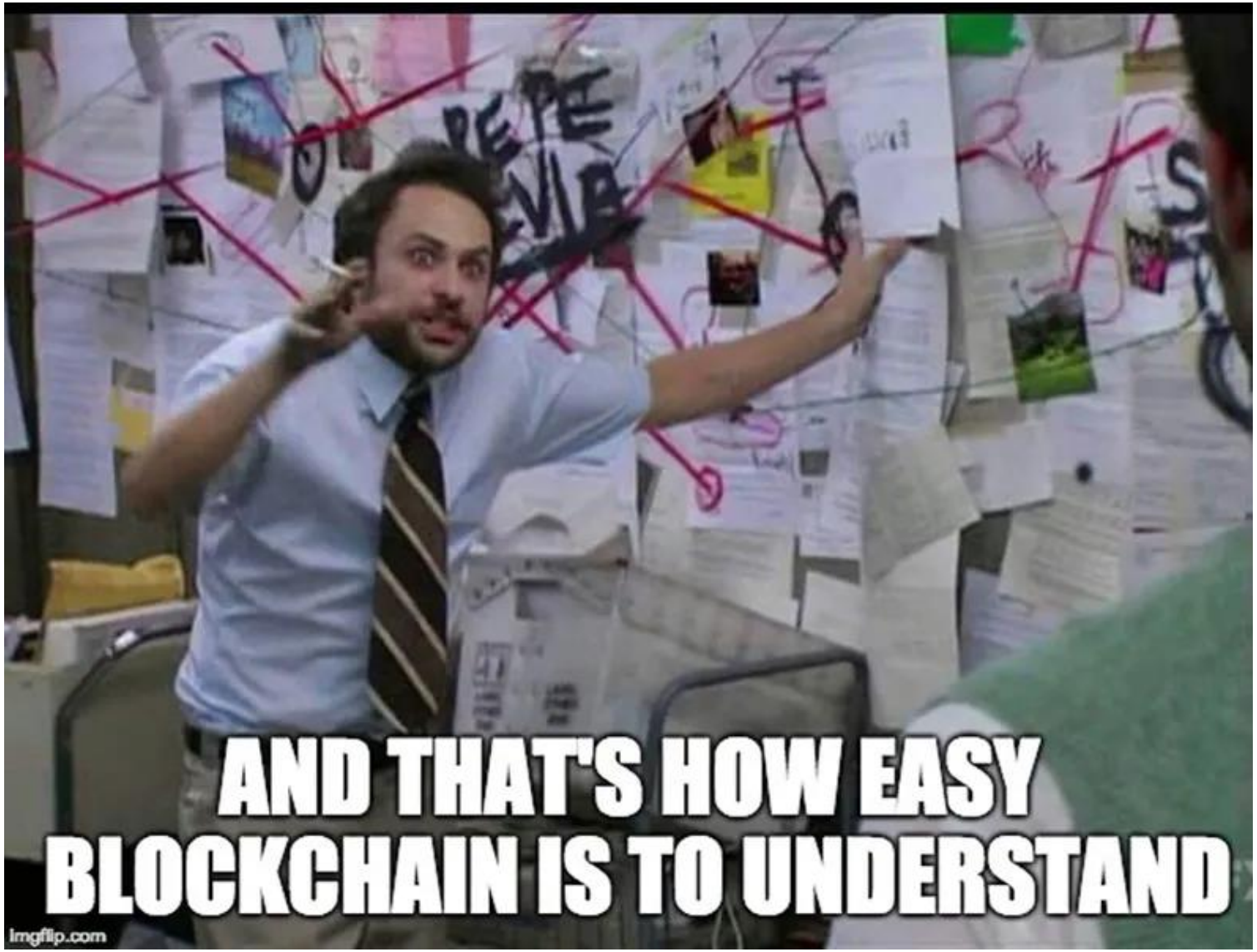
- Definizione legale di smart contract e DLT/blockchain
- Validazione temporale elettronica: i documenti notarizzati con tecnologie basate su sistemi distribuiti tramite timestamp acquisiscono valore legale

## In Europa:

- Il 3 Ottobre 2018, il Parlamento Europeo (EP) ha emesso un ordinanza per riconoscere le tecnologia basate su database distribuiti (DLT) e blockchain europee.
- Validazione temporale elettronica: valida per tutti gli stati membri secondo l'Articolo 41 del Regolamento (UE) N. 910/2014 del Parlamento Europeo

# Imola Informatica: case study




AND THAT'S HOW EASY
BLOCKCHAIN IS TO UNDERSTAND
imgflip.com

imola informatica

IMOLA · BOLOGNA · MILANO

www.imolainformatica.it | blog.imolainformatica.it