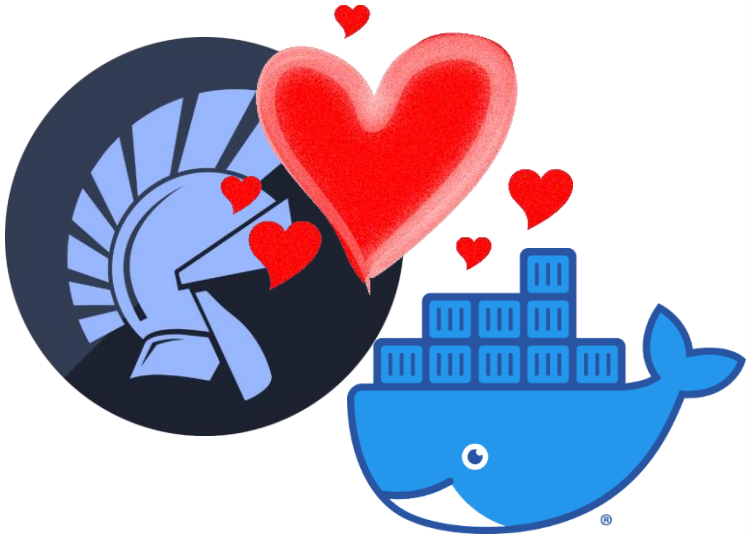
iTDEVCON
EUROPEAN DELPHI CONFERENCE

Docker per sviluppatori Delphi

Marco Breveglieri



ROME NOV 3/4

Marco Breveglieri

Sviluppatore software, trainer e consulente

Compila Quindi Va
www.twitch.tv/compilaquindiva

Delphi Podcast
www.delphipodcast.com

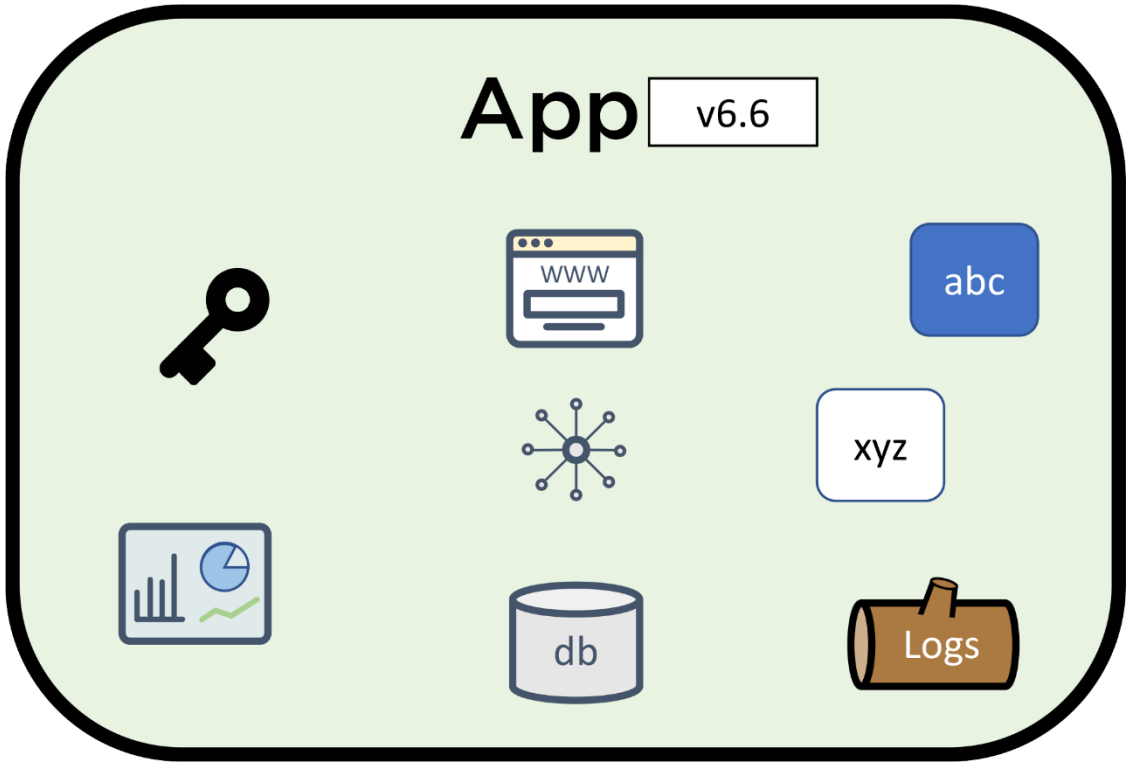
Blog personale
www.compilaquindiva.com

Social & more 👉 **www.breveglieri.it** 👈

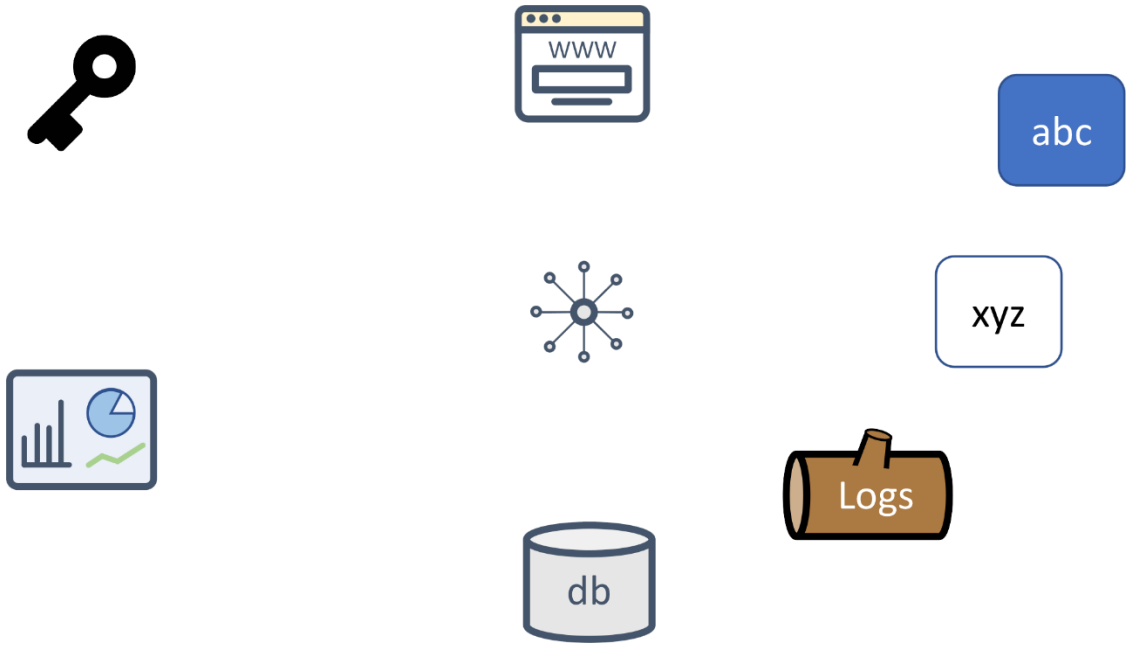
Premessa

Dal monolite...

- Eccessiva interdipendenza tra i moduli software
- Scalabilità e ridondanza inapplicabili
- Resilienza (la «parola magica») molto limitata
- Integrazione di tecnologie e linguaggi ostacolata
- Sostituzione e/o aggiornamento difficoltoso
- Basso sfruttamento abilità del team di sviluppo



...ai microservizi!

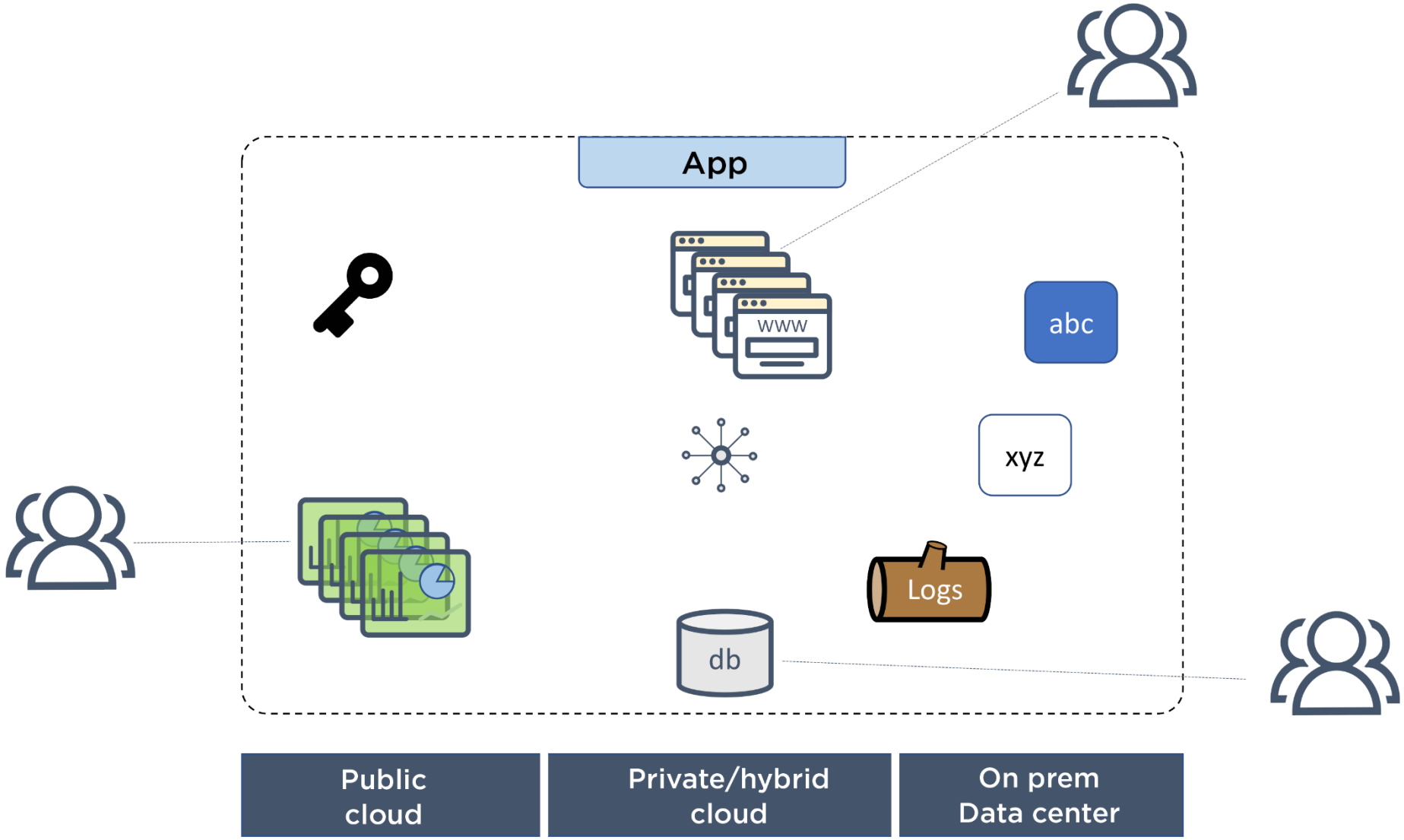


- Moduli del sistema software isolati e indipendenti
- Elevata scalabilità orizzontale in tempi rapidi
- Facile ridondanza dei sistemi critici o più richiesti
- Resilienza garantita da sistemi di messaggistica
- Integrazione aperta ad altri linguaggi e/o piattaforme
- Aggiornamento e sostituzione facilitato delle parti
- Alto sfruttamento abilità del team di sviluppo

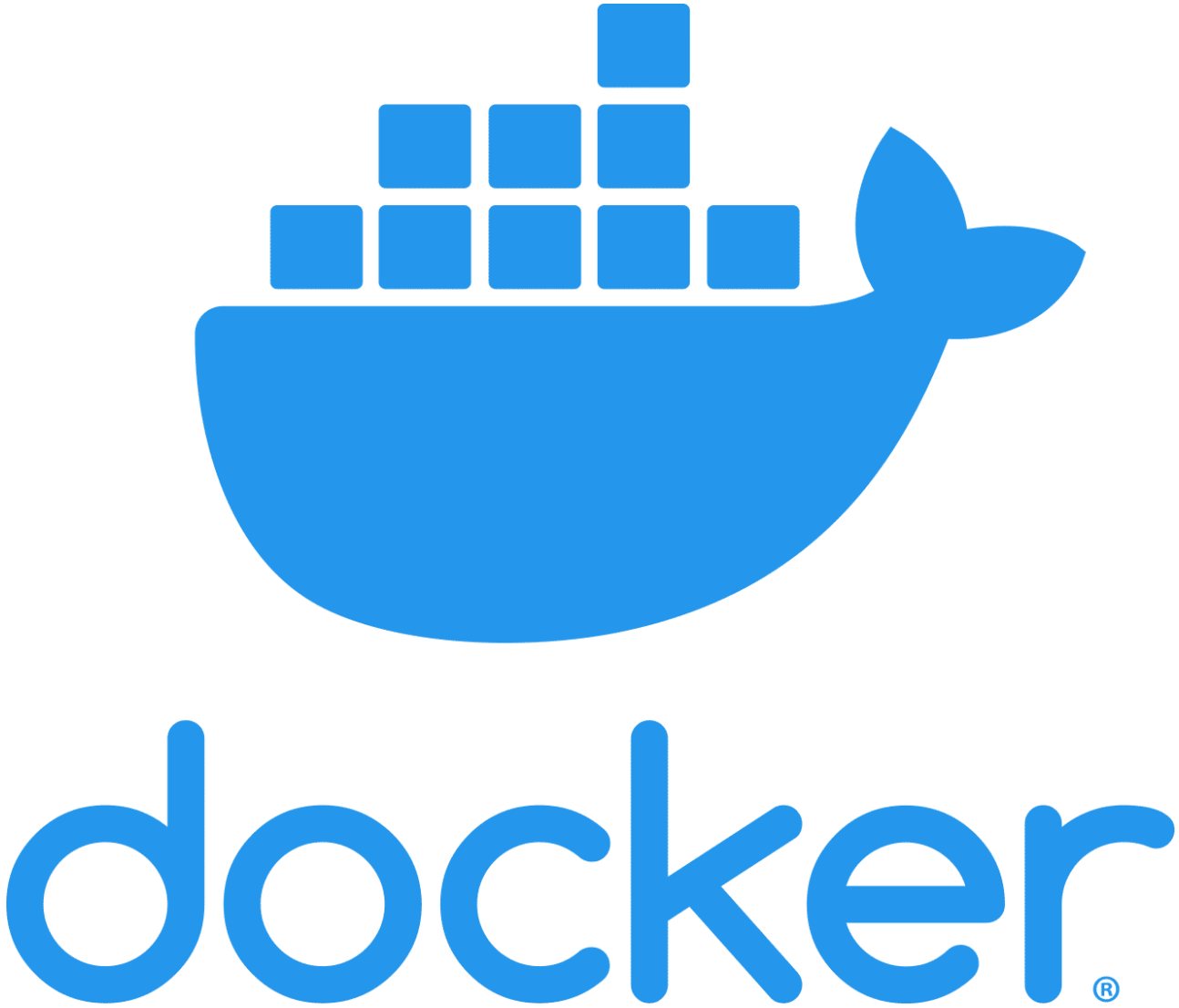
Come facciamo il deploy?

- *Come installare il software all'interno dell'infrastruttura di produzione?*
- *Come gestire e aggiornare le configurazioni dei vari componenti?*
- *Come impostare le connessioni di rete se si usano più macchine per scalabilità e ridondanza?*
- *«Last but not least»… e se voglio migrare tutto nel Cloud?*

Il «desiderata»

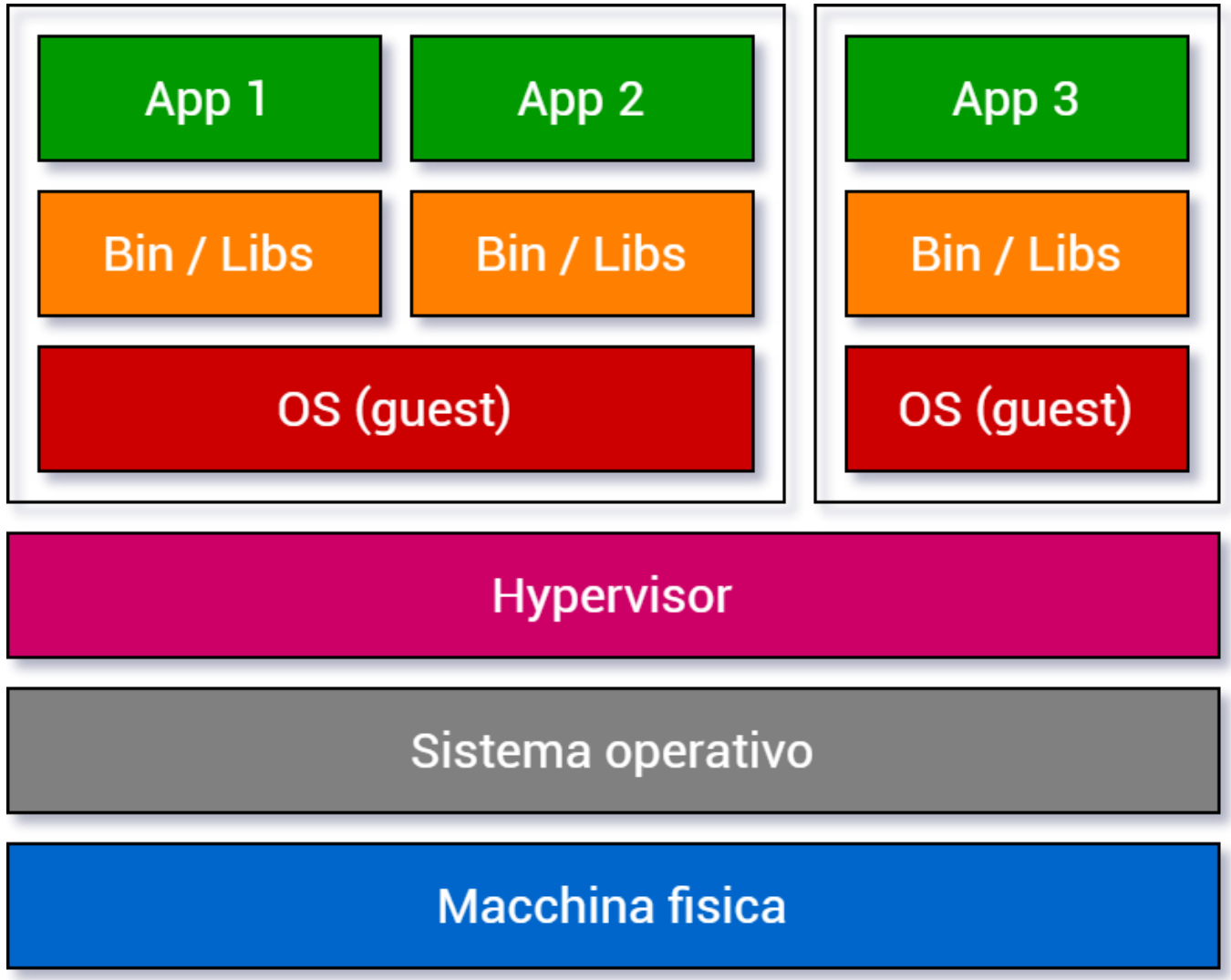


Welcome Docker!

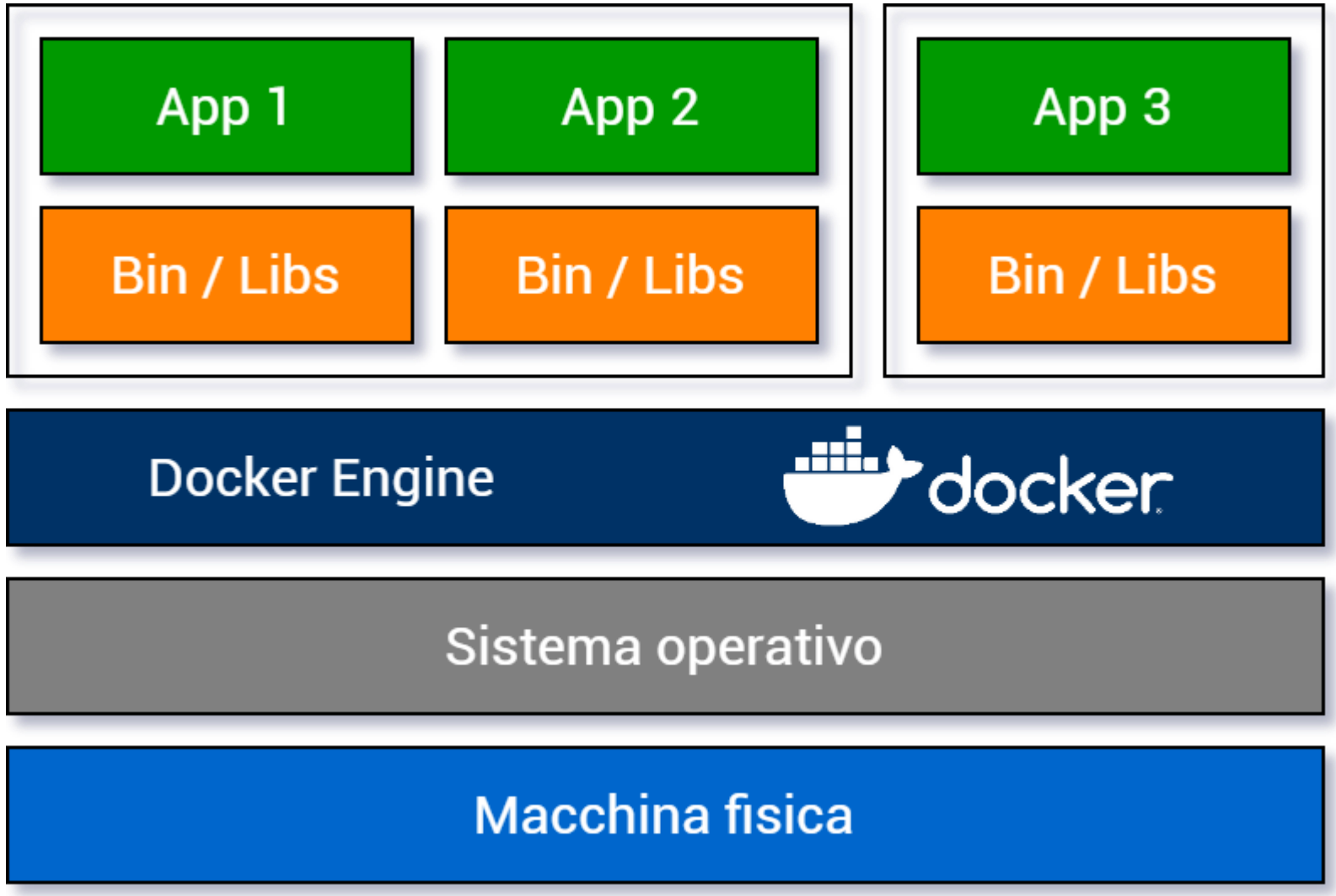


Dalle VM ai Container

Macchine virtuali



Container (in Docker)



Container *vs* VM

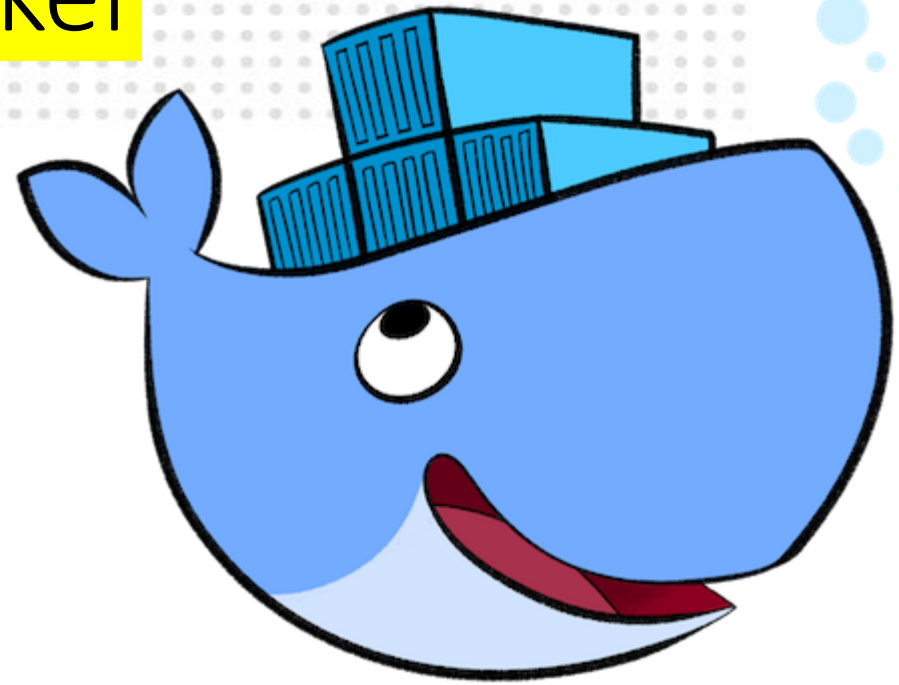
VM

- Isolamento a livello hardware
- Un OS per ogni macchina
- Boot richiede alcuni minuti
- Dimensioni: qualche GB
- Portabilità limitata
- Uso ingente di risorse

Container

- Isolamento a livello software
- OS host condiviso tra container
- Boot richiede pochi secondi
- Dimensioni: qualche MB
- Portabilità elevata
- Uso limitato di risorse

Panoramica di Docker



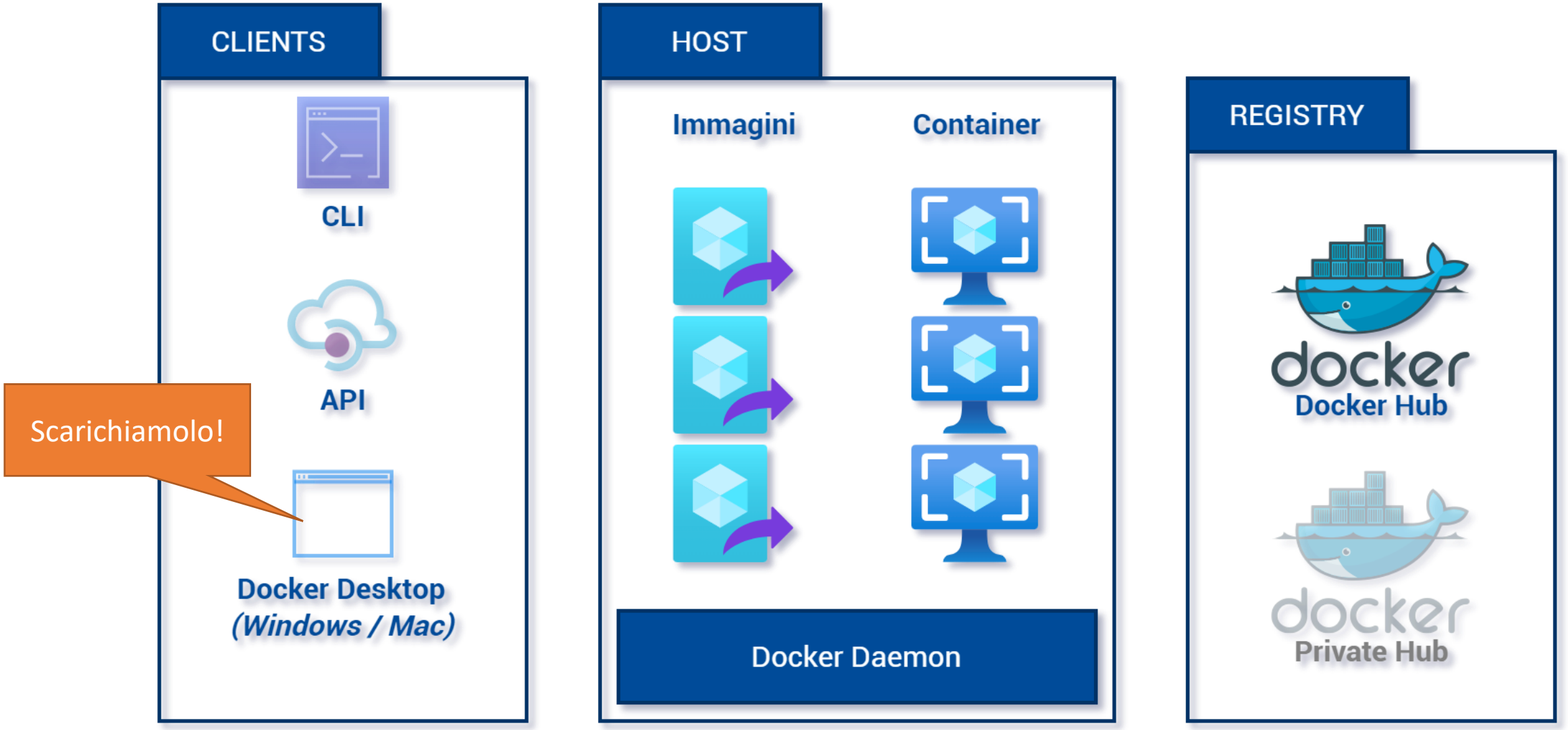
Che cos'è Docker

- Progetto nato nel 2013
  - Annunciato durante la Python Developers Conference, in California
- Inizialmente liquidato come «iniquo»
  - 2014: **100 milioni** di immagini scaricate
  - 2017: oltre **8 miliardi** di immagini scaricate
- E' usato da svariate aziende e professionisti
  - Startup, aziende di medie/grandi dimensioni, numero sempre più crescente di realtà
  - Ha rivoluzionato le architetture di Spotify, Expedia, BBC News
- E' open source ed è (parzialmente) gratuito
- Semplifica e ottimizza l'uso dei Container

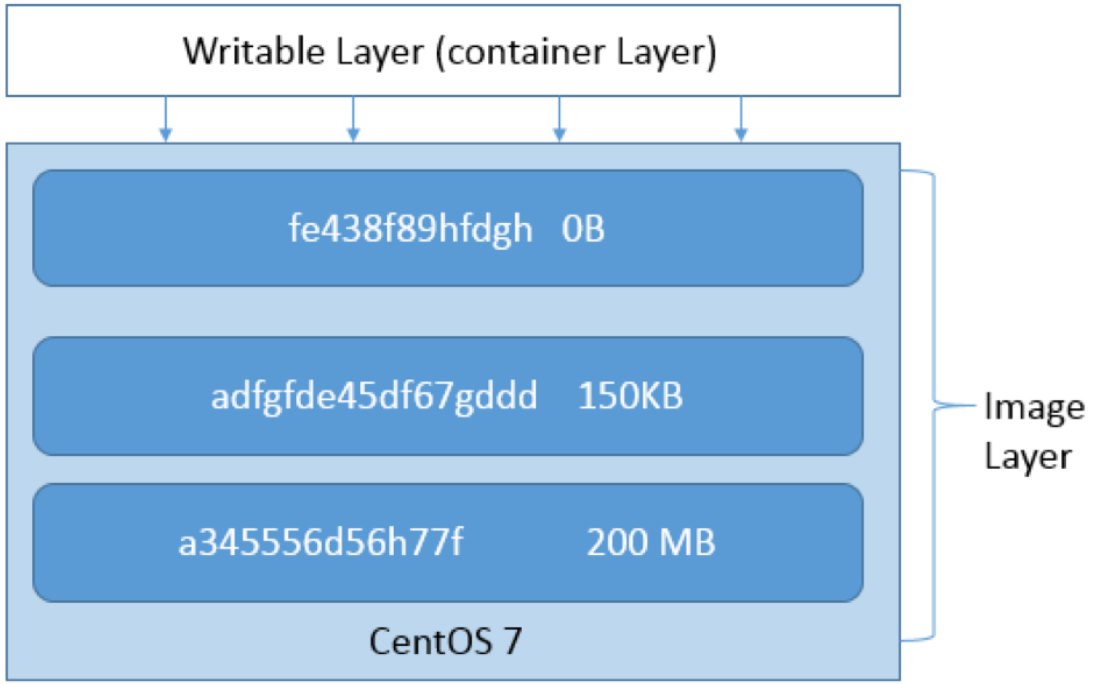
Vantaggi di Docker

- Semplifica la gestione dell'infrastruttura software
  - Usa in modo intelligente i Container (concetto già esistente, ma usato con innovazione)
  - Non richiede necessariamente l'esperienza di diverse figure (sistemista IT, amministratore di rete)
- Rende flessibile il networking
  - Si possono implementare topologie di rete di qualsiasi tipo
  - Le reti sono definite (e configurabili) a livello software
- Abbassa i requisiti hardware
  - Meno risorse richieste, sia fisiche sia virtualizzate
- Riduce i costi di funzionamento delle architetture
  - Grazie ai minori requisiti, riduce i costi di ownership (On Premise e in Cloud)

Architettura

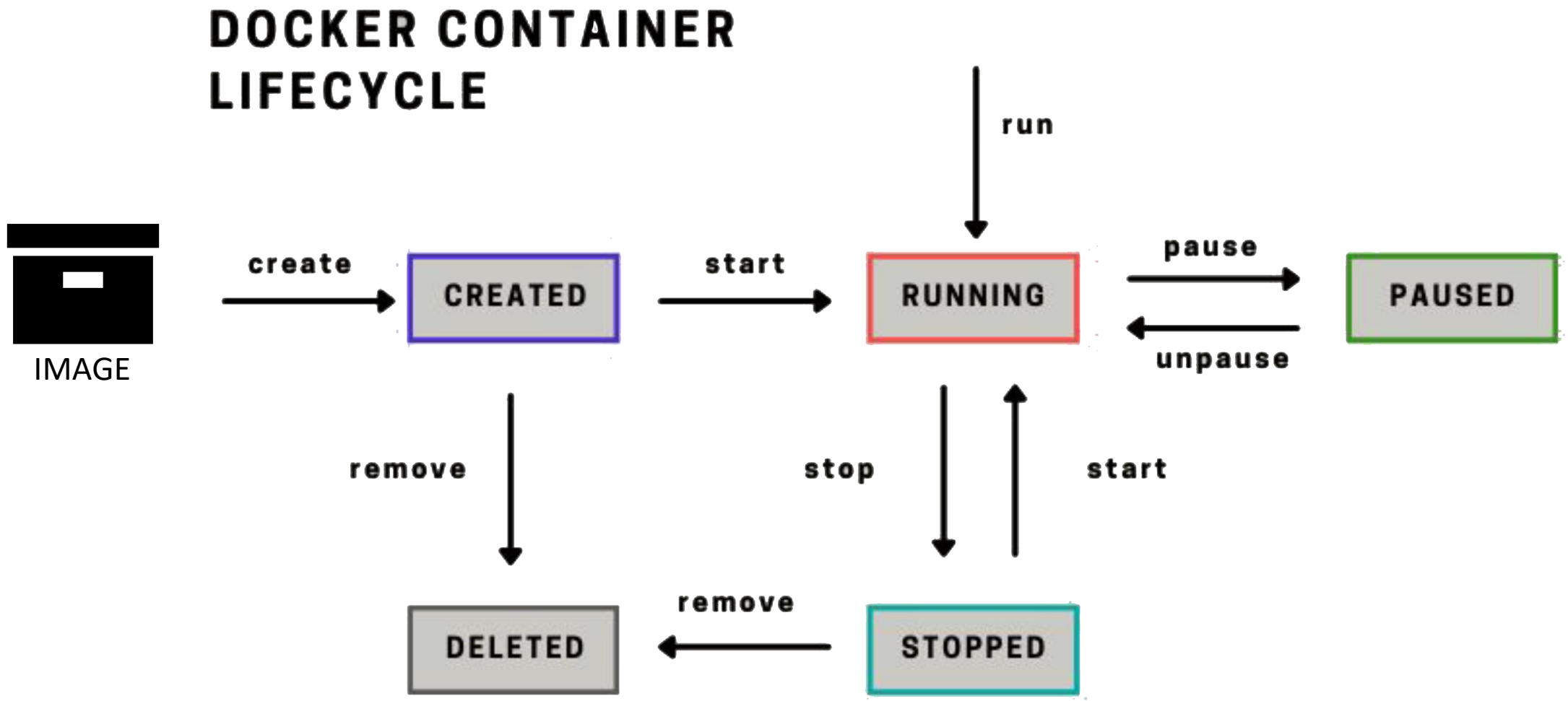


Le immagini



- Sono i «mattoni» costitutivi del mondo Docker
- Permettono di creare e lanciare i Container
- Sono altamente portabili, indipendenti dall'host, e possono essere condivise, archiviate e aggiornate
- Sono basate su un tipo particolare di file che, in fase di build, utilizza in modo ben stratificato i file system, che vengono creati passo dopo passo utilizzando una serie di istruzioni
- Possono essere compilate a partire da un «Dockerfile»
(ne parleremo più avanti)

Il container



Docker CLI

Comandi: le immagini

Scarichiamo una immagine dal registro di Docker Hub

```
docker pull <image-name>:<tag-name>
```

Visualizziamo un elenco delle immagini presenti su disco

```
docker images
docker image ls
```

Visualizziamo i dettagli di una immagine

```
docker inspect <image-name>
```

Rimuoviamo una immagine

```
docker rmi <image-id>
```

Comandi: i container

Creiamo un container a partire da una immagine esistente

```
docker container create --name <container-name> <image-id>
docker create --name <container-name> <image-id>
```

Eseguiamo un container (specificando un set di opzioni)

```
docker run <flags> <container-name> <shell>
```

Eseguiamo comandi all'interno del container avviato

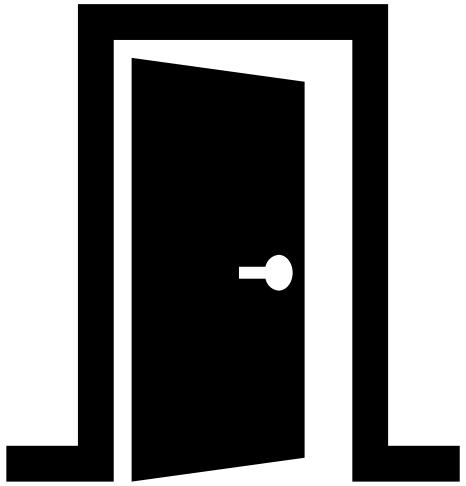
```
docker exec <flags> <container-name> <shell>
```

Visualizzazione dei container in esecuzione (e pregressi)

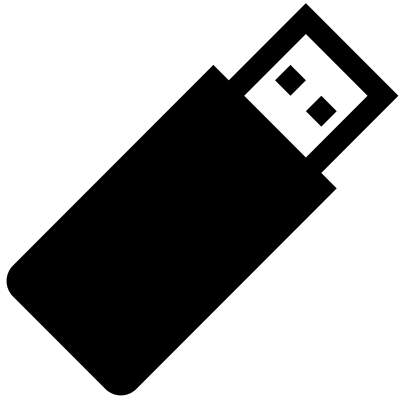
```
docker ps
docker ps -a
```

Scateniamoci alla tastiera! ⌨

Le risorse dei container



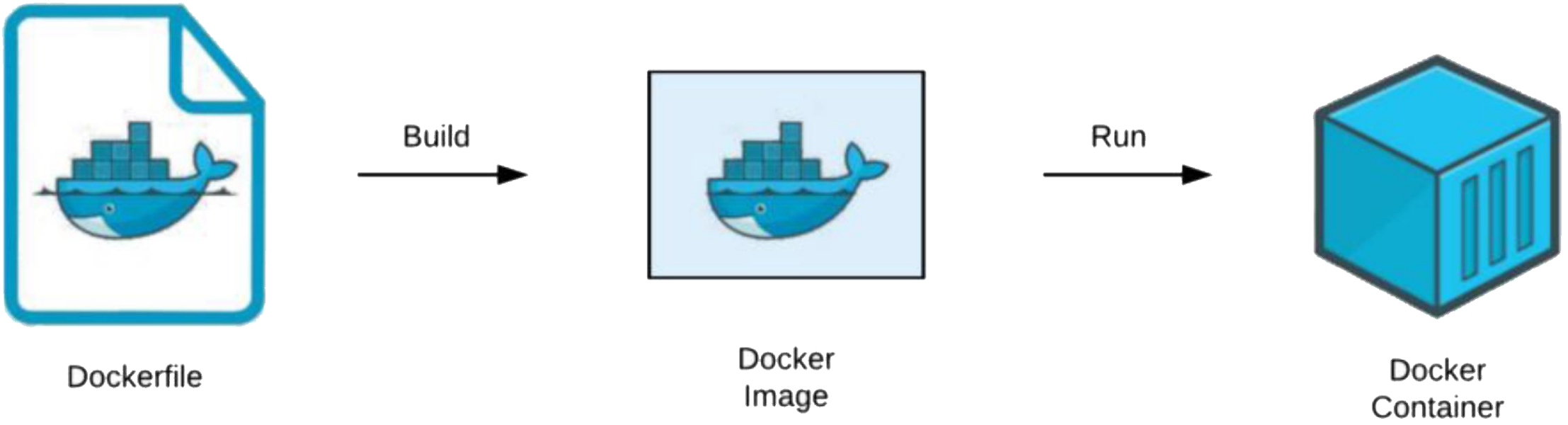
Porte



Volumi

Docker a nostro uso e consumo

Il famigerato «Dockerfile»



Esempio di Dockerfile /1

```
# FROM defines the base image, means it defines from
what image we want to build it from
# Create a layer from the node:8 Docker Image
FROM node:8

# A LABEL is a key-value pair, used to store
information
# Set one/multiple labels on one/multiple lines
LABEL com.example.version="0.1.1-beta"
com.example.release-date="2018-12-
25"  com.example.version.is-production=""
LABEL vendor1="Medium Incorporated"
```

Esempio di Dockerfile /2

```
# Create the app directory for storing, running package manager and launch app
# WORKDIR /app
WORKDIR /usr/src/app

# Copy app dependencies
# from both package.json AND package-lock.json
# to the root of the docker image directory created above
COPY package*.json ./
```

Esempio di Dockerfile /3

```
# Install app dependencies
RUN npm install
# For production, the command should be
# RUN npm install --only=production

# Bundle app source inside Docker image
COPY . .

#  EXPOSE informs Docker that the container listens on the specified network ports at runtime
#  provide access to this isolated docker container
EXPOSE 8080

# Lastly, define the command to run the app
CMD [ "npm", "start" ]
```

Comandi: pubblicazione immagini

Costruiamo una immagine usando il nostro *Dockerfile*

```
docker image build –t <image-name:tag-name> dir
docker build –t <image-name:tag-name> dir
```

Accediamo al nostro account su Docker Hub (e usciamo)

```
docker login
docker logout
```

Etichettiamo la nostra immagine creata in precedenza

```
docker tag <image-name> <repo-name/image-name:tag-name>
```

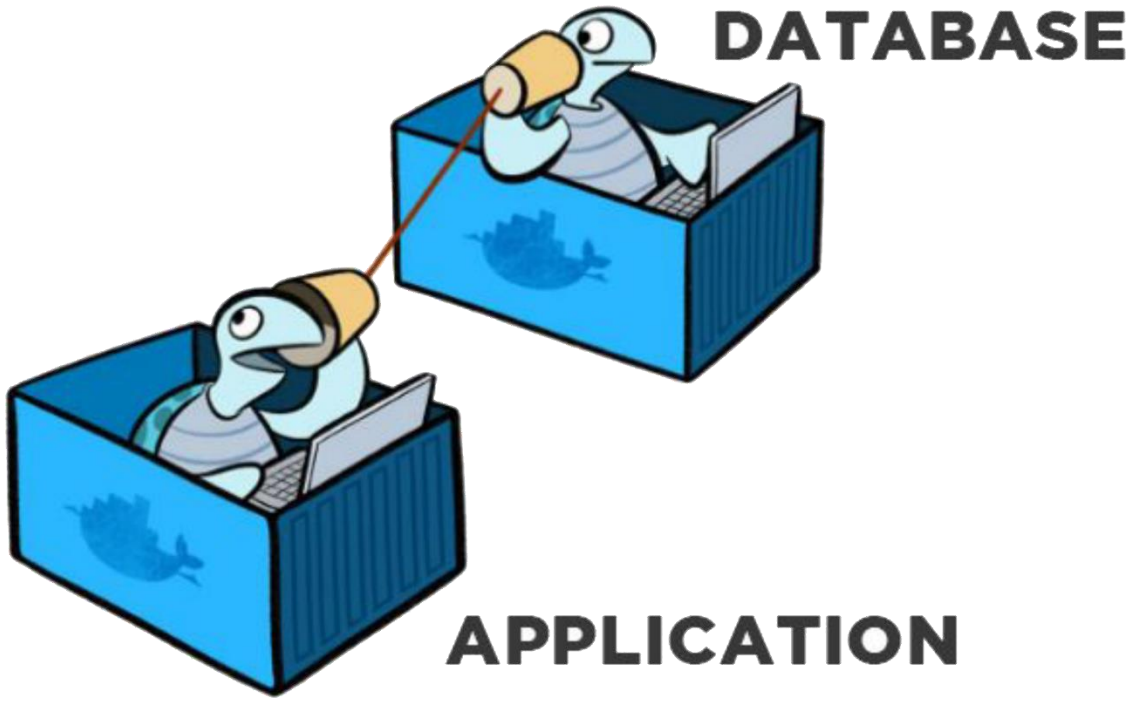
Carichiamo l'immagine sul registro di Docker Hub

```
docker image push <repo-name/image-name>
docker push <repo-name/image-name:tag-name>
```

Scateniamoci alla tastiera! ⌨

Networking

- Docker crea una rete (network) predefinita alla quale appartengono i container: *bridge*
- Possiamo creare nuove network usando differenti topologie
- Per rendere visibili i container tra di loro, possiamo aggiungerli alla rete di appartenenza
- L'accesso al container dall'OS host avviene tramite esposizione e mapping delle porte (vedi comandi **docker run**)



Comandi: pubblicazione immagini

Creiamo una nuova rete per i nostri container

```
docker network create <network-name>
```

Connettiamo il container desiderato alla rete

```
docker network connect <network-name> <container-id>
```

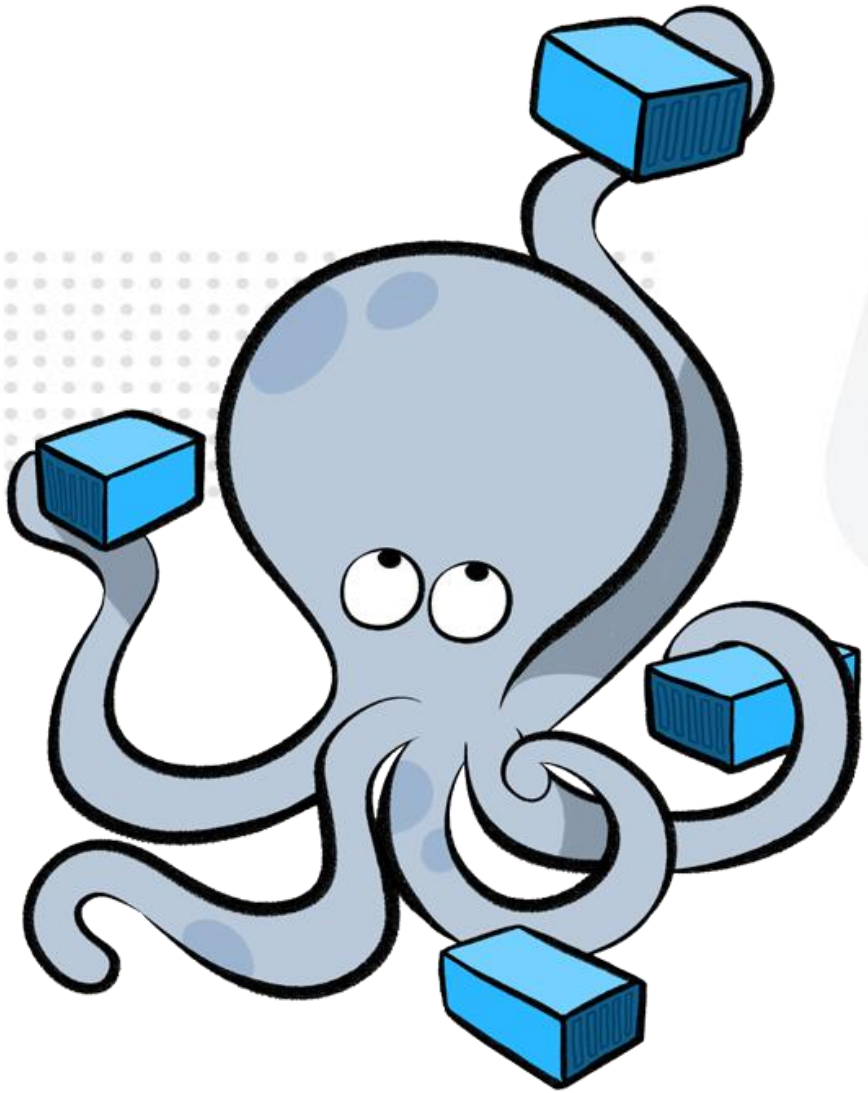
Visualizziamo la lista delle reti disponibili

```
docker network list
```

Ispezioniamo le caratteriche di una specifica rete

```
docker network inspect <network-name>
```

Docker Compose



Docker Compose

- Tool in grado di costruire uno o più container seguendo una configurazione prestabilita
- Il file di configurazione ha nome «docker-compose.yml» ed è in formato YAML
- La configurazione prevede la definizione di più servizi, ciascuno nel proprio container (idealmente), con l'indicazione di *porte*, *volumi*, *network*, ecc.

Esempio

```yaml
version: '3.9'

services:

  service1:
    container_name: node_cont
    build: .
    depends_on:
      - db
    ports:
      - '5000:5000'
    volumes:
      - .:/app
      - /app/node_modules

  db:
    image: 'postgres:latest'
    container_name: db_cont
    ports:
      - '5432:5432'
    environment:
      POSTGRES_USER: 'myuser'
      POSTGRES_PASSWORD: 'mypassword'
```

Comandi: Docker Compose

Costruiamo le immagini relativi alla nostra architettura

```
docker-compose --project-name <name> build --no-cache
```

Avviamo tutti i servizi della nostra architettura

```
docker compose --project-name <name> up
```

Immagini transitorie e/o inutilizzate (*dangling*) dopo tutte le build?

```
FOR /f "tokens=*" %i IN ('docker images -q -f "dangling=true"')
DO docker rmi %i
```


Demo?

Risorse e approfondimenti

Libri consigliati

Docker - Sviluppare e rilasciare software tramite container

di ***Serena Sensini***

Argomenti in breve

- *Installare Docker.*
- *Comprendere i concetti di immagini e container.*
- *Costruire un'immagine Docker.*
- *Avviare, arrestare e analizzare i container.*
- *Gestire, condividere e copiare i dati.*
- *Esporre in rete un container.*
- *Configurare un bridge.*
- *Pubblicare un'immagine su DockerHub.*
- *Conoscere le best practice per lavorare con Docker.*
- *Estendere la gestione dei container con Kubernetes.*
- *Usare Docker su cloud.*

https://www.apogeonline.com/libri/docker-serena-sensini/

Tool correlati: Kubernetes & C.

Q & A


Thanks!