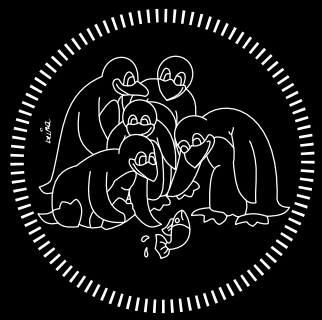
Docker

Corsi Linux 2019

Lorenzo Prosseda

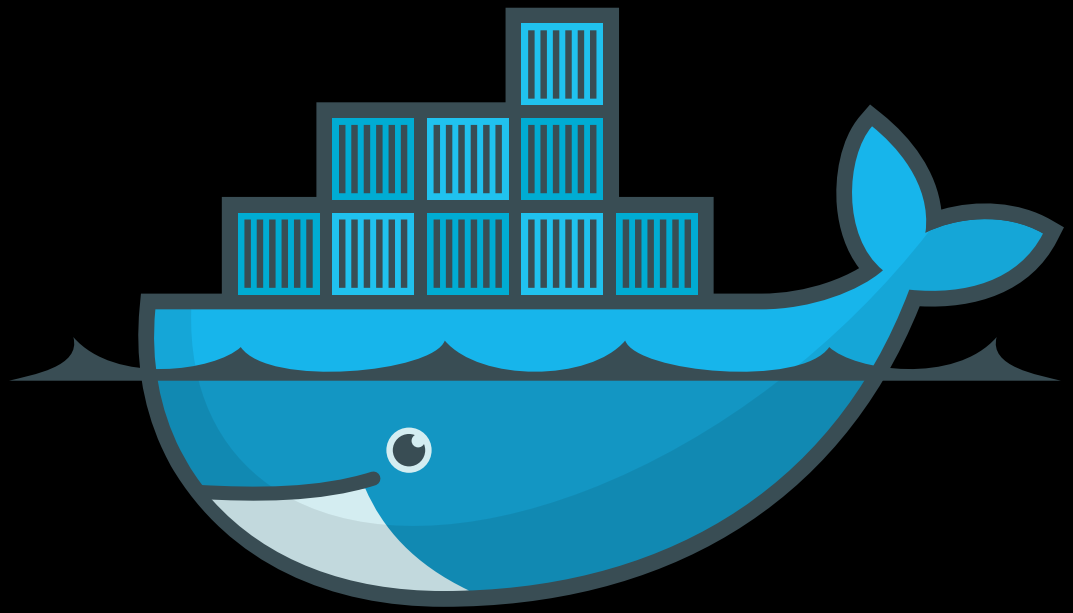
lerokamut@gmail.com

28 marzo 2019



POLITECNICO OPEN
unix LABS

La piattaforma Docker



La piattaforma Docker

A chi può tornare utile?

La piattaforma Docker

A chi può tornare utile?

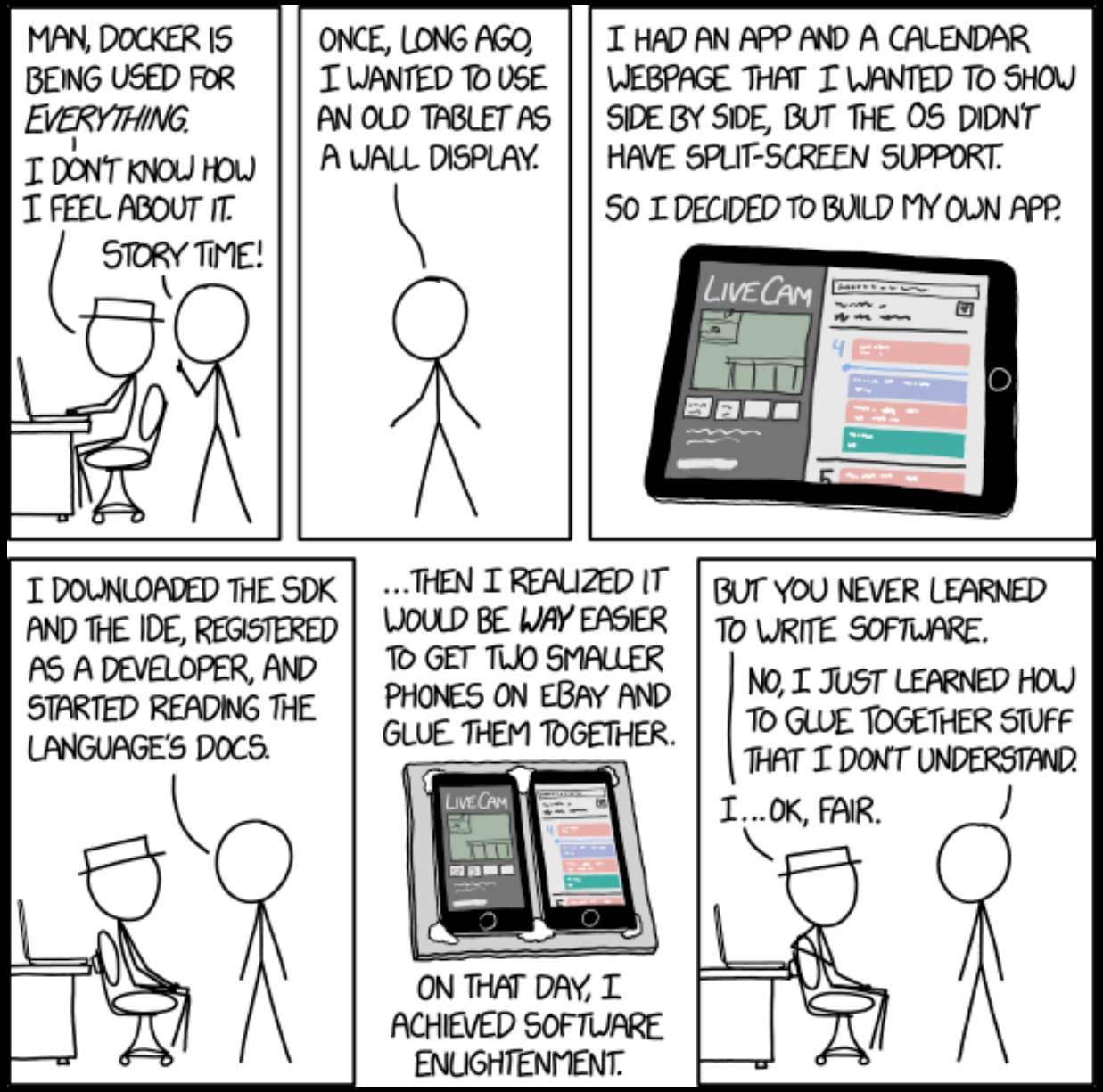
- Sviluppatori

La piattaforma Docker

A chi può tornare utile?

- Sviluppatori
- Amministratori di sistema

La piattaforma Docker



(XKCD)

I container e Linux

In cosa consistono?

I container e Linux

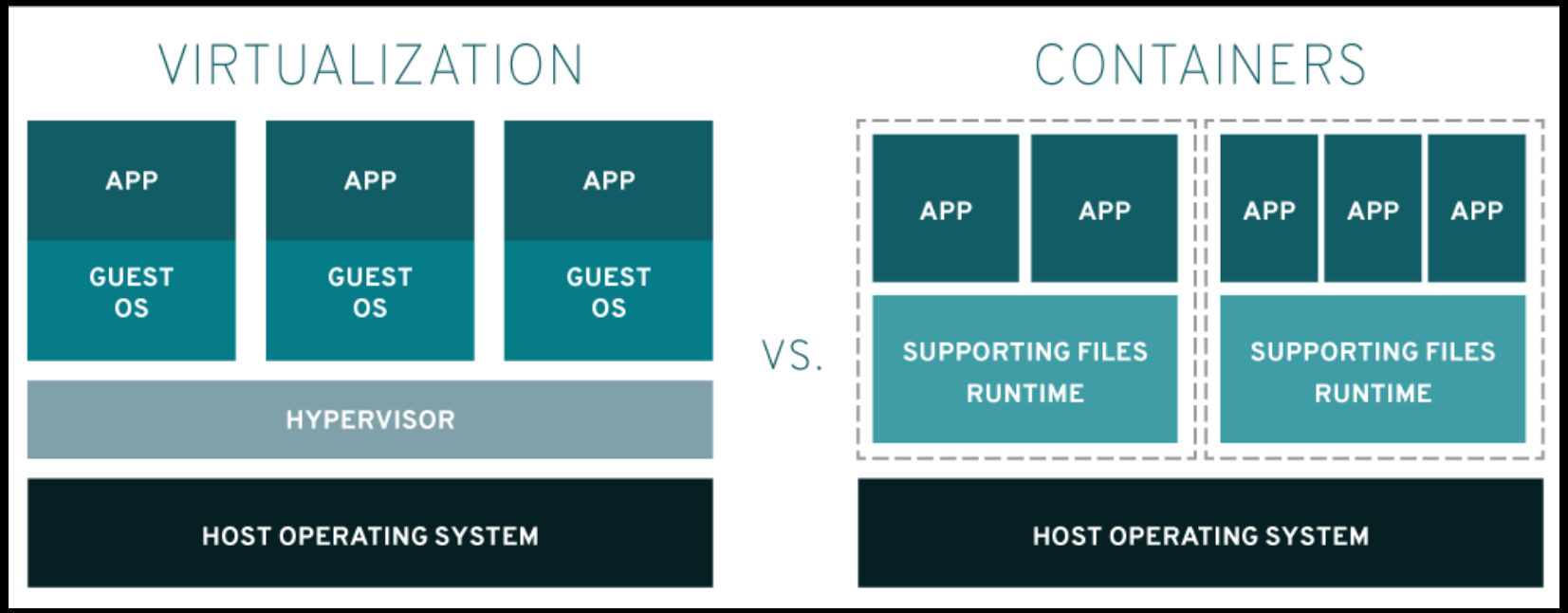
In cosa consistono?

- un processo nel sistema operativo
  - kernel condiviso e nessun hypervisor

I container e Linux

In cosa consistono?

- un processo nel sistema operativo
  - kernel condiviso e nessun hypervisor



I container e Linux

Da dove si pescano?

I container e Linux

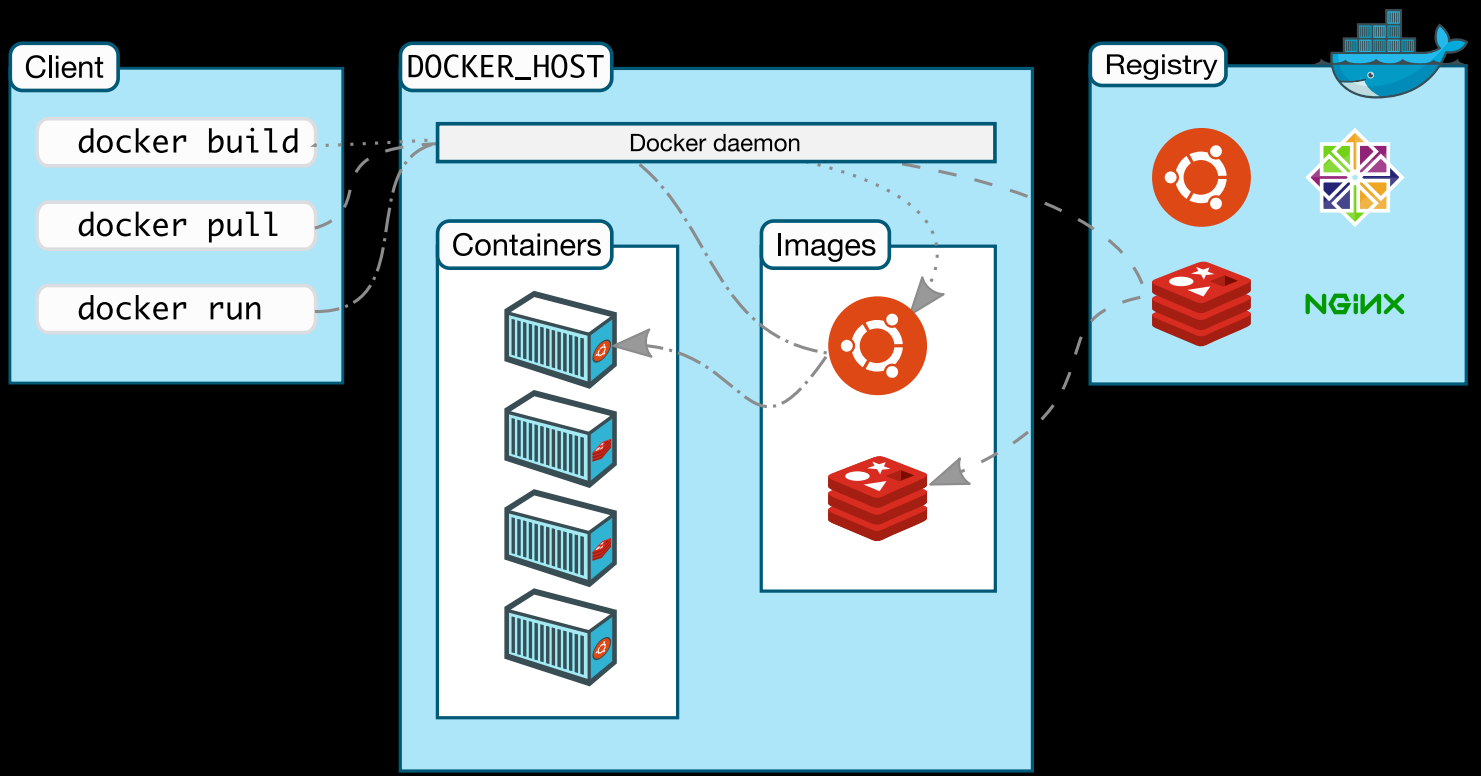
Da dove si pescano?

- immagini Docker
  - registro Docker

I container e Linux

Da dove si pescano?

- immagini Docker
  - registro Docker



I container e Linux

Caratteristiche:

I container e Linux

Caratteristiche:

- flessibili: qualunque applicativo, semplice o complesso, è scomponibile in unità funzionali;

I container e Linux

Caratteristiche:

- flessibili: qualunque applicativo, semplice o complesso, è scomponibile in unità funzionali;
- leggeri: usano la capacità di computazione dell'host in modo efficiente;

I container e Linux

Caratteristiche:

- flessibili: qualunque applicativo, semplice o complesso, è scomponibile in unità funzionali;
- leggeri: usano la capacità di computazione dell'host in modo efficiente;
- effimeri: possono essere sostituiti «a caldo» e «sul posto»;

I container e Linux

Caratteristiche:

- flessibili: qualunque applicativo, semplice o complesso, è scomponibile in unità funzionali;
- leggeri: usano la capacità di computazione dell'host in modo efficiente;
- effimeri: possono essere sostituiti «a caldo» e «sul posto»;
- portabili: formato comune e isolamento;

I container e Linux

Caratteristiche:

- flessibili: qualunque applicativo, semplice o complesso, è scomponibile in unità funzionali;
- leggeri: usano la capacità di computazione dell'host in modo efficiente;
- effimeri: possono essere sostituiti «a caldo» e «sul posto»;
- portabili: formato comune e isolamento;
- scalabili: ottima integrazione con sistemi di bilanciamento del carico.

I container e Linux

Condividere il kernel con tutti i container crea un punto di rottura:

I container e Linux

Condividere il kernel con tutti i container crea un punto di rottura:

- vulnerabilità del kernel utilizzabile da un container qualsiasi

I container e Linux

Condividere il kernel con tutti i container crea un punto di rottura:

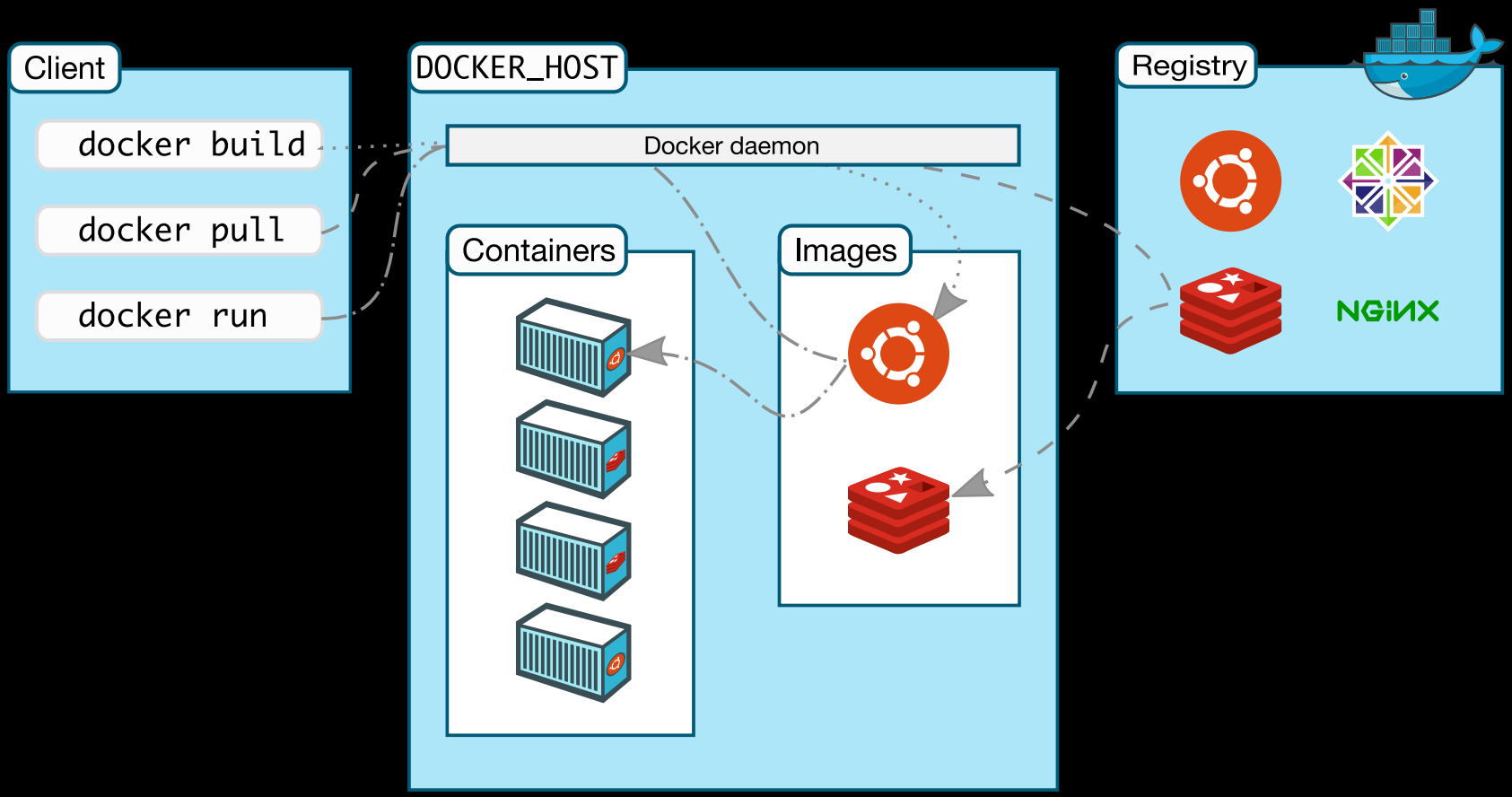
- vulnerabilità del kernel utilizzabile da un container qualsiasi
- tutti i container devono usare lo stesso kernel

Installazione e primi passi

Usando i repository ufficiali di Docker si otterrà sempre l'ultima versione disponibile (Link)

- per controllare in numero di versione di Docker usare il comando `$ docker -v`;
- prima di cominciare, assicuriamoci che il processo di Docker sia in esecuzione, col comando `$ docker info`;
- il comando `$ docker run hello-world` permette di avviare un'immagine predisposta per testare la propria installazione.

Installazione e primi passi

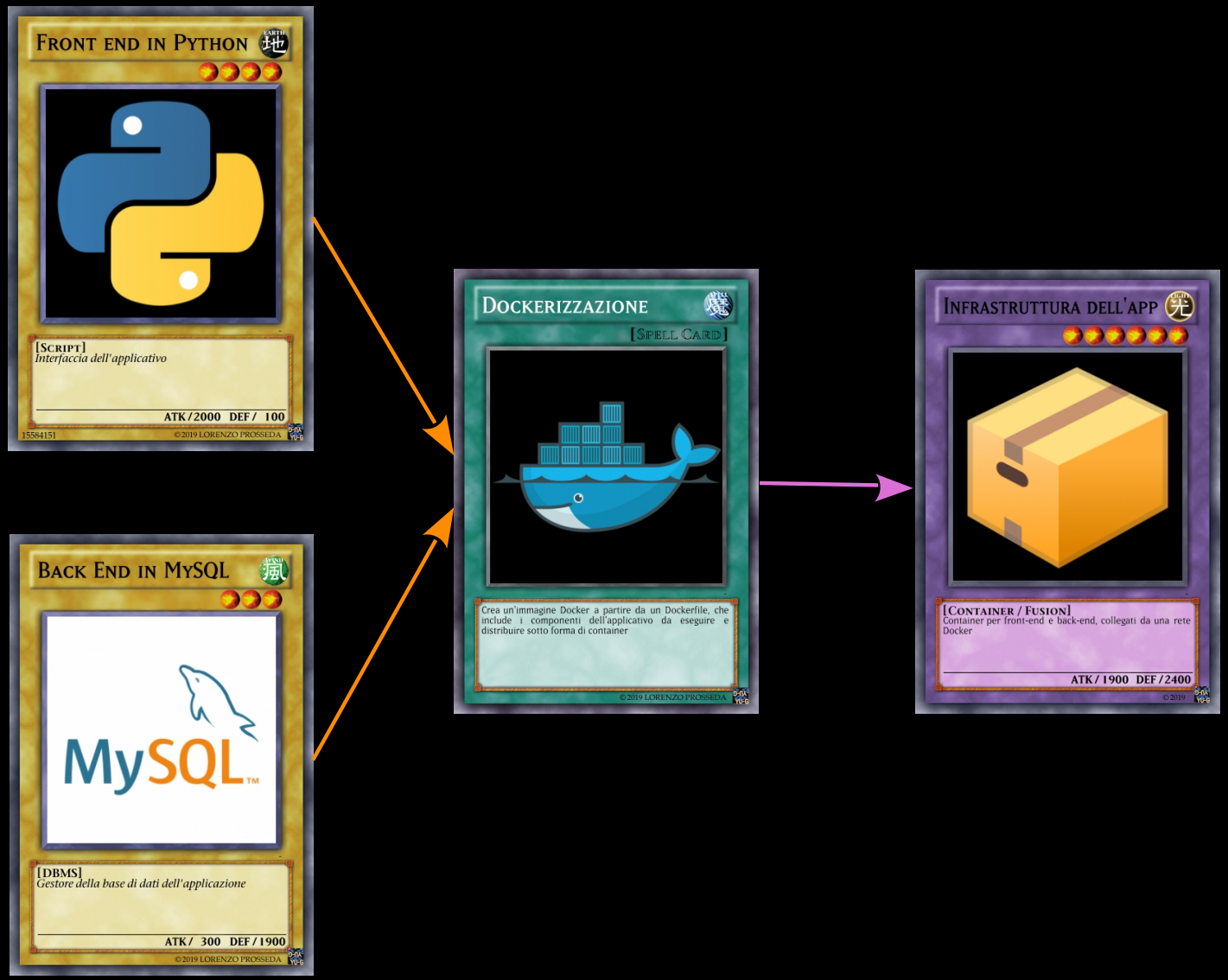


Installazione e primi passi

Gestiamo i container tramite i sottocomandi di Docker:

- visualizzare tutti i container creati: `$ docker ps -a`;
- controllare le immagini scaricate: `$ docker image ls`;
- arrestare un container: `$ docker stop <nome>`;
- eliminare un container non in esecuzione:
  `$ docker rm <nome>`;
- cancellare un'immagine non eseguita da nessun container:
  `$ docker rmi <nome>`;

Dockerizzazione



(Immagini realizzate tramite yugiohcardmaker)

Dockerizzazione

Creiamo le immagini di cui abbiamo bisogno tramite Dockerfile

- file di testo che contiene istruzioni per ottenere un'immagine Docker e configurare il suo ambiente di lavoro
- i comandi nel Dockerfile possono essere eseguiti via CLI dall'utente

Dockerizzazione

Immagine per front-end Python:

- Scarichiamo l'immagine docker con Python 3.7:
  `$ docker pull python:3.7`;
- Avviamo il container col comando:

```
$ docker run -it --rm --name echo-srv \
-v "$PWD":/usr/src/echo-srv:z \
-w /usr/src/echo-srv \
-p 127.0.0.1:8080:8080 \
python:3.7 python main.py;
```

Dockerizzazione

Cosa diavolo è quel comando?

- `-it` → mantiene aperto stdin, alloca uno pseudo-tty;
- `--rm` → rimuove il container al termine dell'esecuzione del comando;
- `--name` → nomina il container che sarà creato dall'immagine;
- `-v` → monta una cartella locale all'interno dell'ambiente del container;
- `-w` → imposta la cartella di lavoro, dalla quale sarà avviato il comando;
- `-p` → mappa esplicitamente una porta dell'host su una del container;
- `python:3.7` → nome dell'immagine da usare;
- `python main.py` → comando da eseguire;

Dockerizzazione

Come possiamo rendere automatico e ripetibile questo procedimento?

1. scriviamo un Dockerfile che contenga le stesse direttive appena usate
2. creiamo un'immagine a partire dal dockerfile, col comando `$ docker build --tag=echo-srv .`;
3. avviamo un container dall'immagine creata: `$ docker run -ti --rm -p 0.0.0.0:8080:8080 echo-srv`.

Dockerizzazione

Contenuto del Dockerfile per front-end Python:

```
FROM python:3.7

RUN mkdir /usr/src/echo-srv
COPY . /usr/src/echo-srv
VOLUME /usr/src/echo-srv

WORKDIR /usr/src/echo-srv

EXPOSE 8080

ENTRYPOINT python main.py
```

Dockerizzazione

Contenuto del Dockerfile per back-end MySQL:

```
FROM mysql:8

COPY template.sql /docker-entrypoint-initdb.d

ENV MYSQL_ROOT_PASSWORD="meme"

EXPOSE 3306
```

- creiamo l'immagine col comando:
  `$ docker build --tag=db-srv .`
- avviamo l'immagine creata tramite questo file col comando:
  `$ docker run -ti --rm -p 0.0.0.0:33060:3306 db-srv`

Rete bridge

Creiamo una rete Docker: *bridge*

Rete bridge

Creiamo una rete Docker: *bridge*

- migliore isolamento e gestione dei container nella rete

Rete bridge

Creiamo una rete Docker: *bridge*

- migliore isolamento e gestione dei container nella rete
- risoluzione DNS tra container

Rete bridge

Creiamo una rete Docker: *bridge*

- migliore isolamento e gestione dei container nella rete
- risoluzione DNS tra container
- i container nel bridge di default condividono le variabili di ambiente

Rete bridge

Creiamo una rete Docker: *bridge*

- migliore isolamento e gestione dei container nella rete
- risoluzione DNS tra container
- i container nel bridge di default condividono le variabili di ambiente
- ogni bridge utente può essere configurato come un'interfaccia di rete a sé stante

Rete bridge

Usiamo i sottocomandi di `$ docker network`:

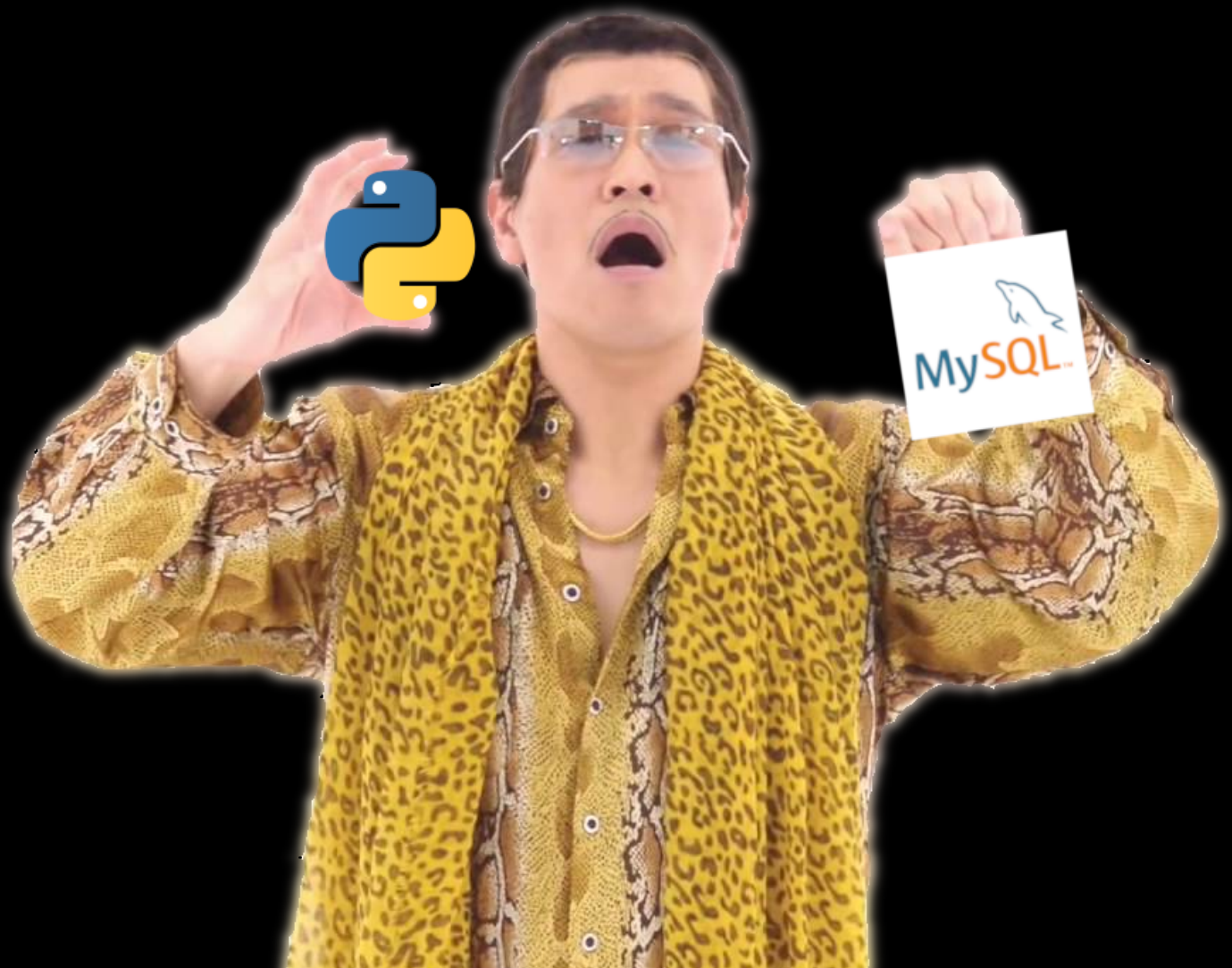
1. creazione del bridge:
   ```
   $ docker network create echo-net
   ```
2. ricerca nome container:
   ```
   $ docker container ls
   ```
3. aggiunta container alla rete utente:
   ```
   $ docker network connect echo-net
   <nome_container>
   ```
4. disconnetto container dal bridge di default:
   ```
   $ docker network disconnect bridge
   <nome_container>
   ```

Compose

I've got front-end, I've got back-end, mmmmmh...
docker-compose!

Compose

I've got front-end, I've got back-end, mmmmmh...
docker-compose!



Compose

Docker-compose (Link)

- tool che permette di gestire più facilmente ambienti con container multipli
- sostituisce i comandi `$ docker run <...>`
- basta lanciare `$ docker-compose up`, e per terminare i container avviati CTRL + C

Compose

Contenuto del file *docker-compose.yml*:

```
version:  "3"
  services:
    echo:
      depends_on:
        - db
      image:  python:3.7
      ports:
        - "127.0.0.1:8080:8080/tcp"
      volumes:
        - ./python:/usr/lib/socket-app:z
      entrypoint:  python /usr/lib/socket-app/main.py
    db:
      image:  db-srv
      expose:
        - "3306"
      environment:
        - MYSQL_ROOT_PASSWORD=meme
```

Compose

Cosa contiene questo file YAML chiamato docker-compose?

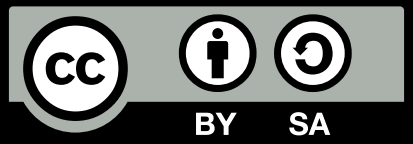
- `version` → versione del tool compose a cui è destinato il file;
- `services` → permette di definire servizi, dichiarati come lo stato desiderato delle componenti dell'applicazione;
- `depends_on` → definisce una relazione di dipendenza tra servizi, influenzando il loro ordine di avvio e di arresto;
- `image` → immagine da cui creare i container associati al servizio;
- `ports` → esponi una o più porte del container sull'host;
- `volumes` → monta una cartella dell'host su un percorso del container (Link);
- `entrypoint` → comando da eseguire all'interno del container una volta avviato; sovrascrive l'entrypoint di default;
- `expose` → esponi una porta solo sulla rete docker (non accesibile dall'host);
- `environment` → inizializza una o più variabili d'ambiente;

Fonti

- Slide 2016
- Slide 2018
- Docker overview
- Wikipedia - Docker
- RedHat Forum - What is a linux container

Fine

Grazie per l'attenzione!



Queste slide sono licenziate Creative Commons Attribution-ShareAlike 4.0

http://www.poul.org