



Docker in Delphi

Un tool ormai essenziale per lo sviluppo



Marco Brevaglieri

Sviluppatore software
Trainer e Consulente



Homepage & social links

👉 www.brevaglieri.it



Compila Quindi Va

👉 www.twitch.tv/compilaquindiva



Delphi Podcast

👉 www.delhipodcast.com



AGENDA

- Introduzione a Docker: cos'è e come è fatto
- Primi passi con Docker
- Esercitazione pratica con Delphi (e altri servizi)
- Docker Compose: cos'è e come si usa
- Risorse e approfondimenti
- Q & A

begin

Che cos'è Docker?

- **Progetto nato nell'ormai lontano 2013**
 - Annunciato nella Python Developer Conference (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
 - Ha rivoluzionato le architetture software di Spotify, Expedia, BBC News
- **E' open-source e (parzialmente) gratuito**
- **Semplifica e ottimizza l'uso dei «Container»**

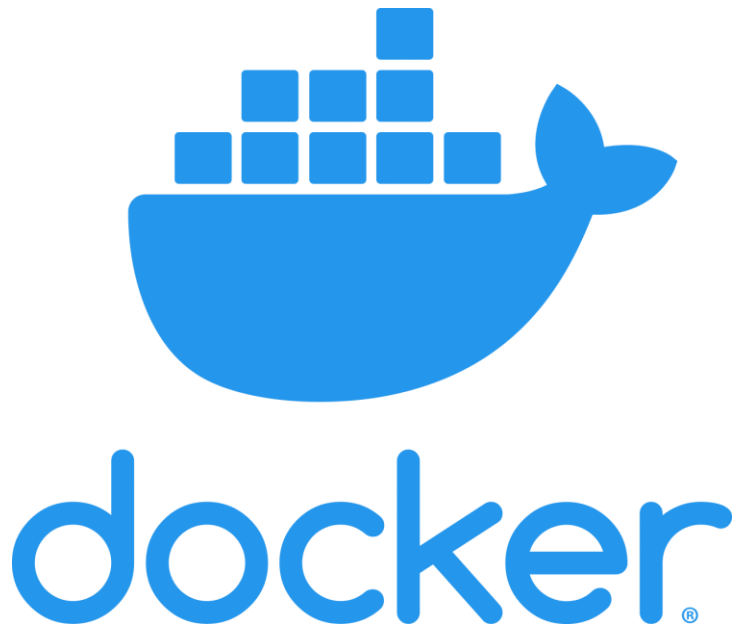
Caratteristiche di Docker

- **Semplifica la gestione dell'infrastruttura software**
 - Usa in modo «intelligente» la tecnologia dei Container
 - Non richiede l'esperienza di diverse figure oltre a quella dello sviluppatore (es. sistemista IT, amministratore di rete, ecc.)
- **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 necessari**
 - Meno risorse richieste, sia fisiche sia virtualizzate
- **Riduce i costi di funzionamento delle architetture**
 - Grazie ai minori requisiti, abbatte i costi di ownership

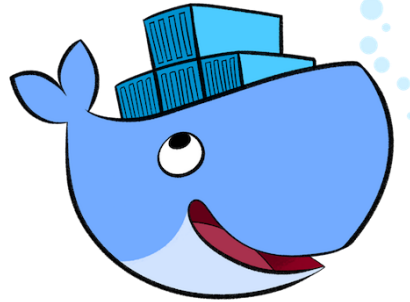
Perché Docker?

- Strumento ideale per sistemi software a (micro) servizi
- Deploy isolato e indipendente delle applicazioni e dei servizi
- Facile replicabilità degli ambienti, con relativa configurazione
- Installazione «side by side» di versioni differenti di specifici tool
- Ideale per esigenze «usa e getta»

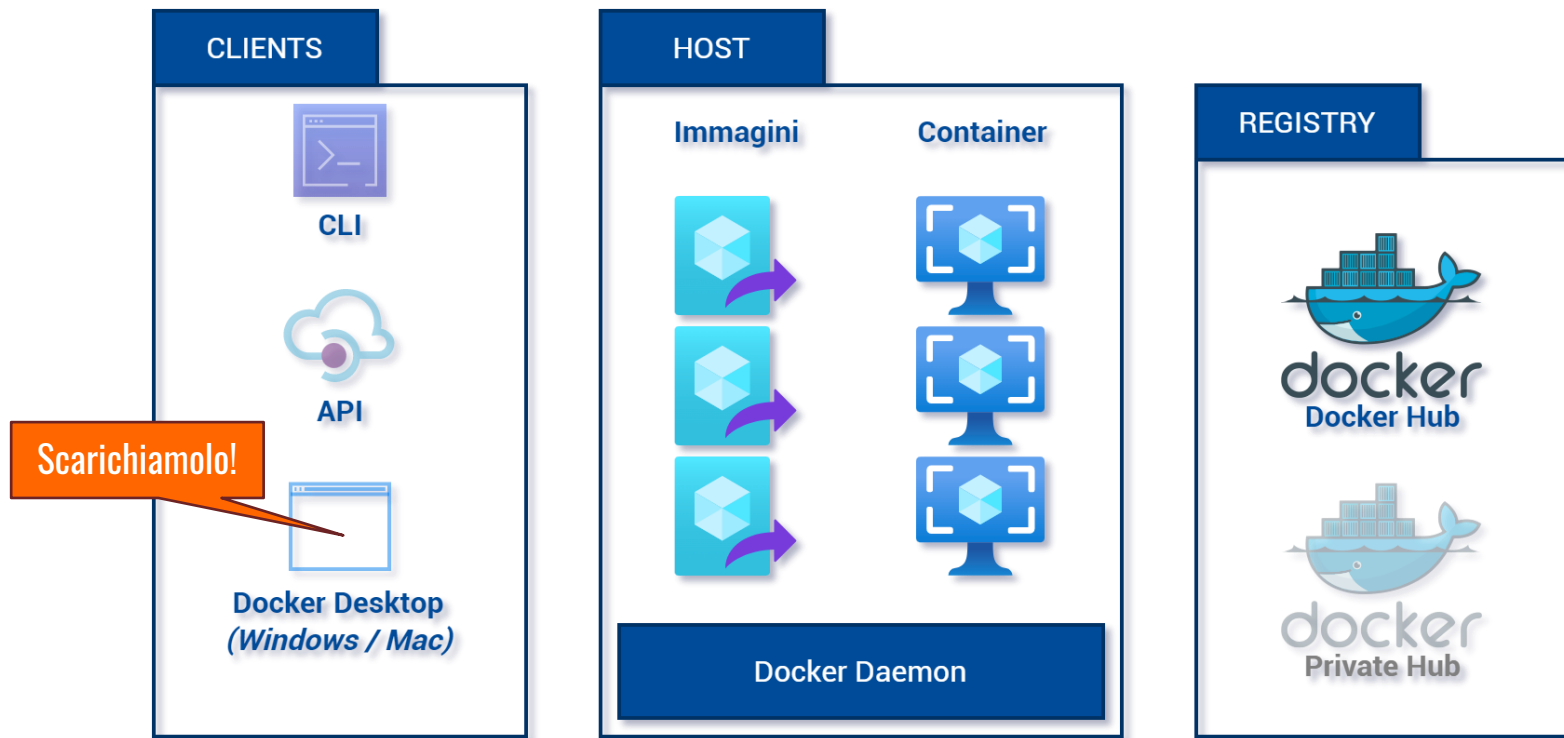
...e altro ancora!



Come è fatto Docker

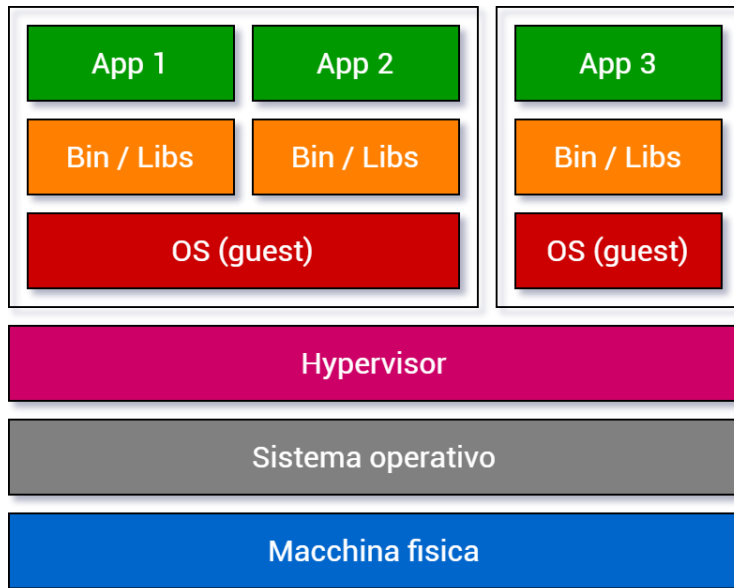


Architettura di Docker



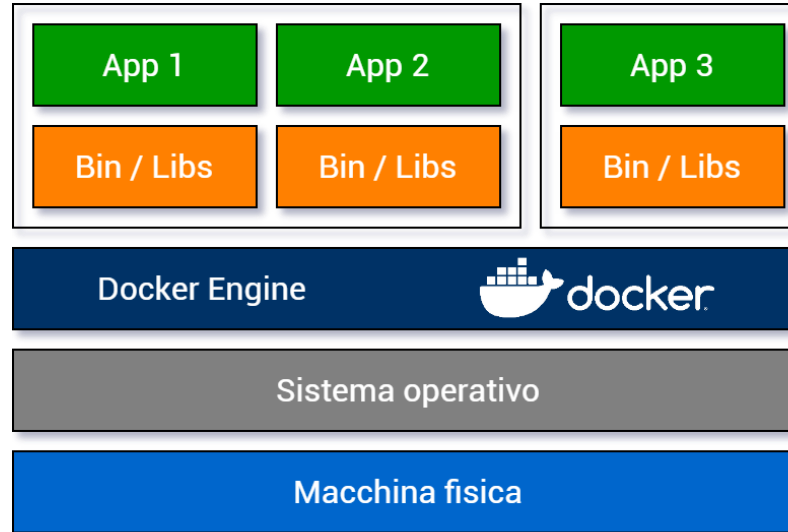
Dalle VM ai Container /1

Virtual
Machine



Dalle VM ai Container /2

Container



Container vs VM

VM

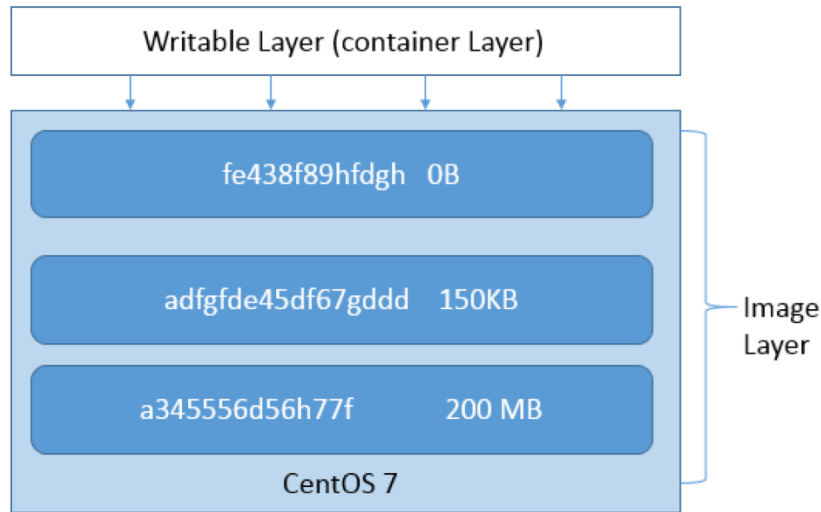
- Isolamento a livello HW
- Un OS per ogni macchina
- Boot richiede tot minuti
- Dimensioni di qualche GB
- Portabilità limitata
- Uso ingente di risorse

Container

- Isolamento a livello SW
- OS host condiviso
- Boot richiede secondi
- Dimensioni di qualche MB
- Portabilità elevata
- Uso limitato di risorse

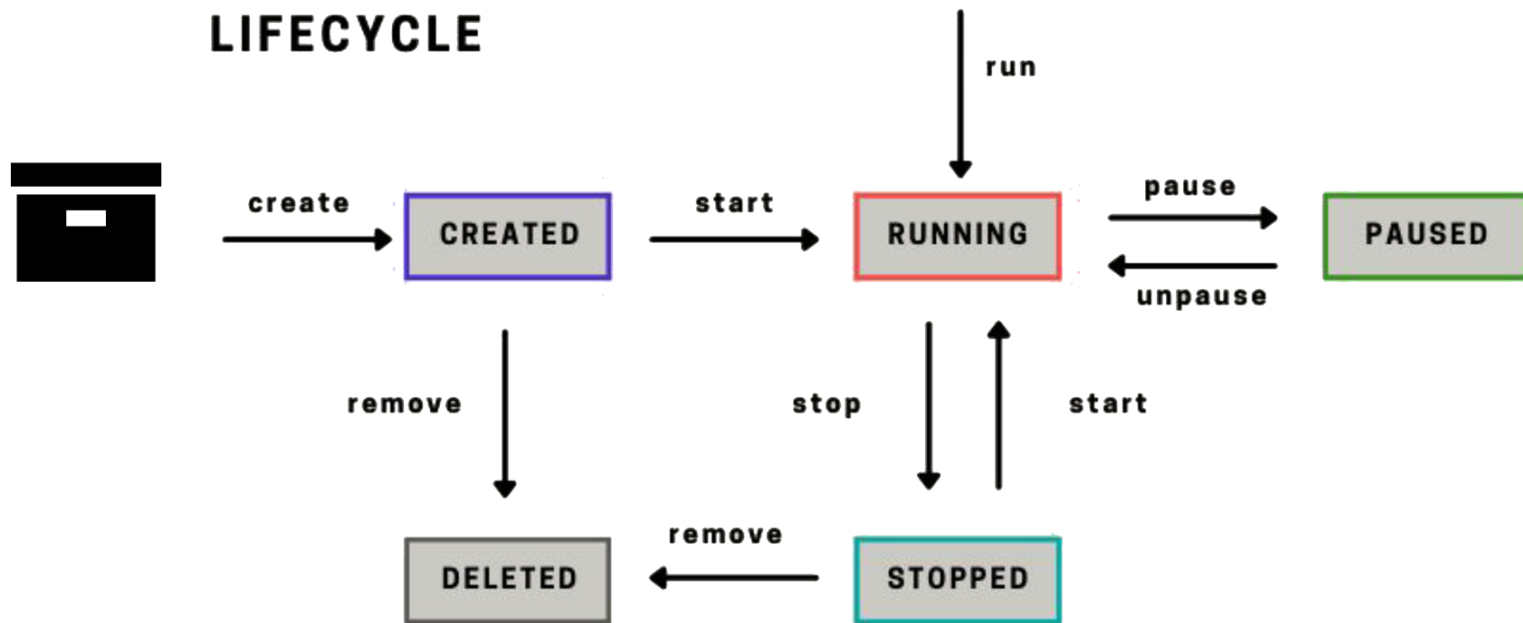
Immagini

- Permettono di creare, preparare e lanciare i Container
- Sono altamente portabili: indipendenti dall'host, possono essere condivise, archiviate e aggiornate
- In fase di build, creano layer di un file system passo per passo
- Possono essere create, partendo da zero o basandosi su immagini già esistenti

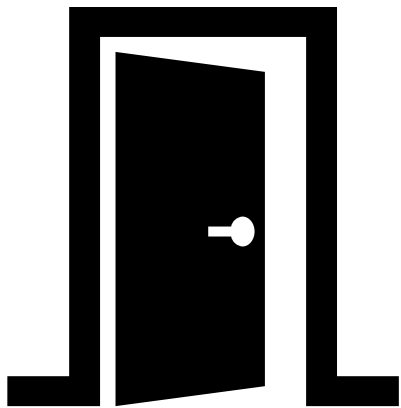


Da Image a Container

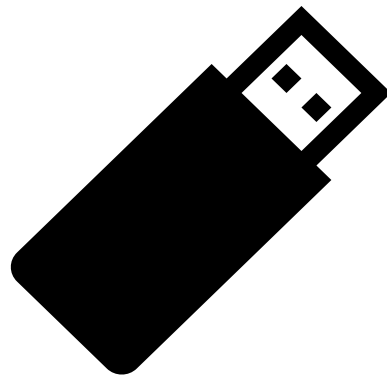
DOCKER CONTAINER LIFECYCLE



Risorse dei Container

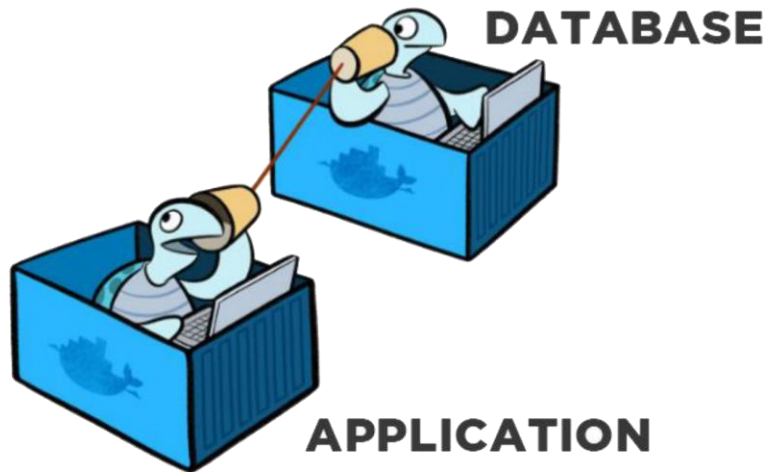


Porte



Volumi

Networking

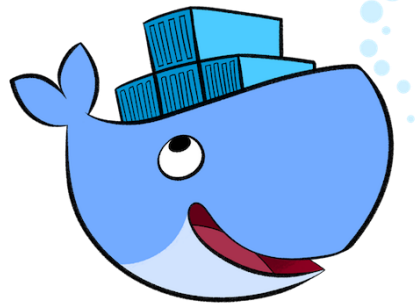


- Docker crea una rete predefinita alla quale appartengono i container: bridge
- Possiamo creare network personalizzate usando differenti topologie
- Per consentire ad alcuni Container di comunicare tra loro, possiamo aggiungerli alla stessa network
- L'accesso al Container dall'OS host avviene tramite mapping delle porte esposte

Scateniamoci alla tastiera!



Primi passi con Docker



Prerequisiti



- **Installare WSL (Windows Subsystem for Linux)**
Pannello di Controllo: Aggiungi/rimuovi feature Windows
- **Aggiornare WSL alla versione 2**
https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi
- **Installare il tool Docker Desktop**
<https://docs.docker.com/desktop/install/windows-install/>
- **Creazione di un account su Docker Hub**
<https://hub.docker.com/>

Docker CLI: 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>
```

Docker CLI: 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

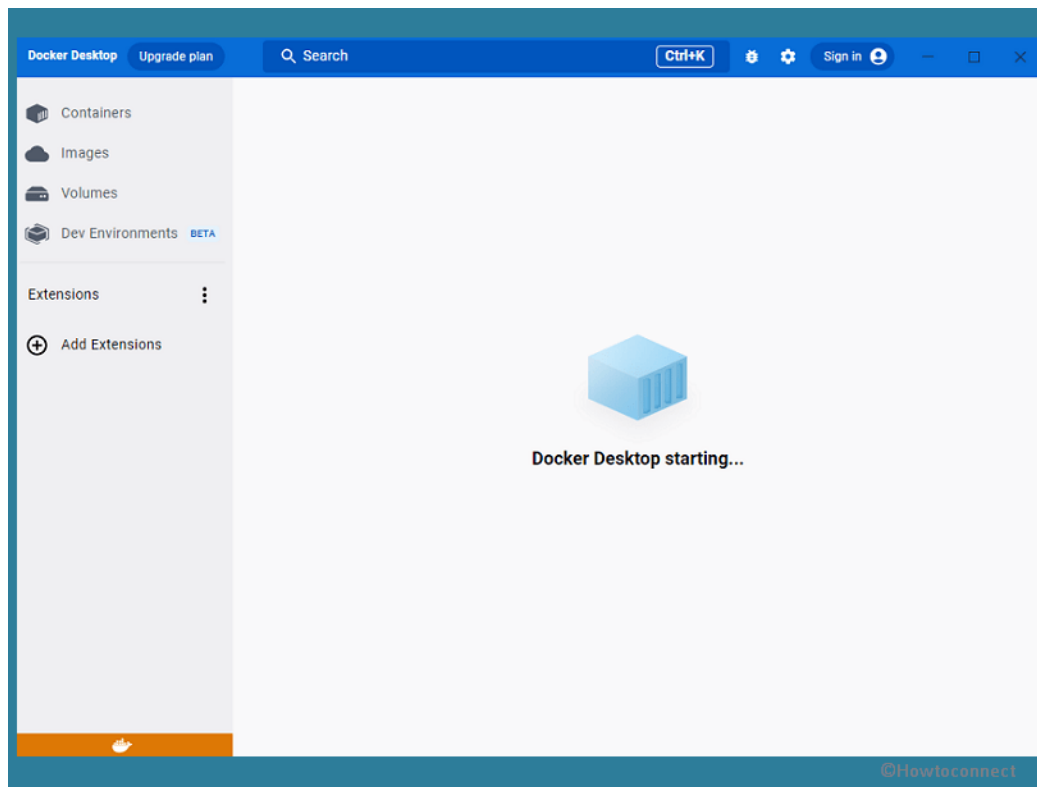
```
docker exec <flags> <container-name> <shell>
```

Visualizzazione dei container in esecuzione (e pregressi)

```
docker ps
```

```
docker ps -a
```

Usiamo «Docker Desktop»



Docker CLI: networking

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>`

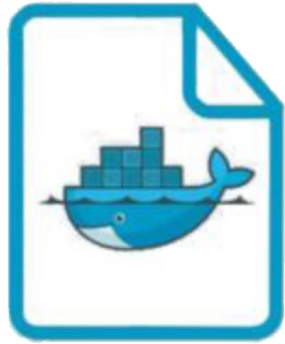
Visualizziamo la lista delle reti disponibili

`docker network list`

Ispezioniamo le caratteristiche di una specifica rete

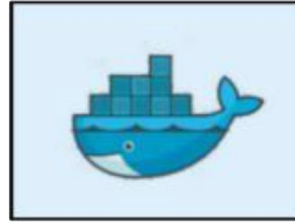
`docker network inspect <network-name>`

Creiamo un «Dockerfile»



Dockerfile

Build →



Docker
Image

Run →



Docker
Container

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" ]
```

Docker CLI: publish di 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

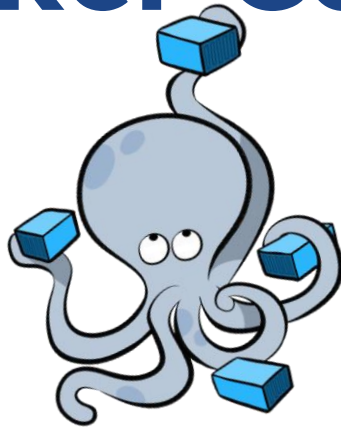
```
docker tag <image-name> <repo-name/image-name:tag-name>
```

Carichiamo l'immagine sul registro di Docker Hub

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

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

Docker Compose



Cos'è 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
- Ciascun servizio viene eseguito nel proprio Container
- E' possibile configurare porte, volumi, network, ecc.

Esempio di file

```
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'
```

Docker Compose CLI

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 le build?

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


Risorse e approfondimenti

Licenze e costi

Personal

Ideal for individual developers, education, open source communities, and small businesses.

\$0

- Docker Desktop ⓘ
- Unlimited public repositories
- Docker Engine + Kubernetes ⓘ
- 200 image pulls per 6 hours
- Unlimited scoped tokens ⓘ

Pro

Includes pro tools for individual developers who want to accelerate their productivity.

\$5 /month

- ← Everything in Personal plus:
- Unlimited private repositories
 - 5,000 image pulls per day
 - 5 concurrent builds ⓘ
 - 300 Hub vulnerability scans

Team

Ideal for teams and includes capabilities for collaboration, productivity and security.

\$9 /user/month
max 100 users
Start with minimum
5 users for \$25

- ← Everything in Pro, plus:
- Unlimited teams
 - 15 concurrent builds ⓘ
 - Unlimited Hub vulnerability scans
 - Add users in bulk
 - Audit logs ⓘ

Business

Ideal for medium and large businesses who need centralized management and advanced security capabilities.

\$24 /user/month
Start with minimum
5 users for \$120
Only available with an
annual subscription

- ← Everything in Team, plus:
- Hardened Docker Desktop
 - Enhanced Container Isolation
 - Settings management
 - Centralized management
 - Registry Access Management
 - Image Access Management
 - Single Sign-On (SSO)
 - SCIM user provisioning
 - VDI support
 - Purchase via invoice
 - Volume Pricing Available

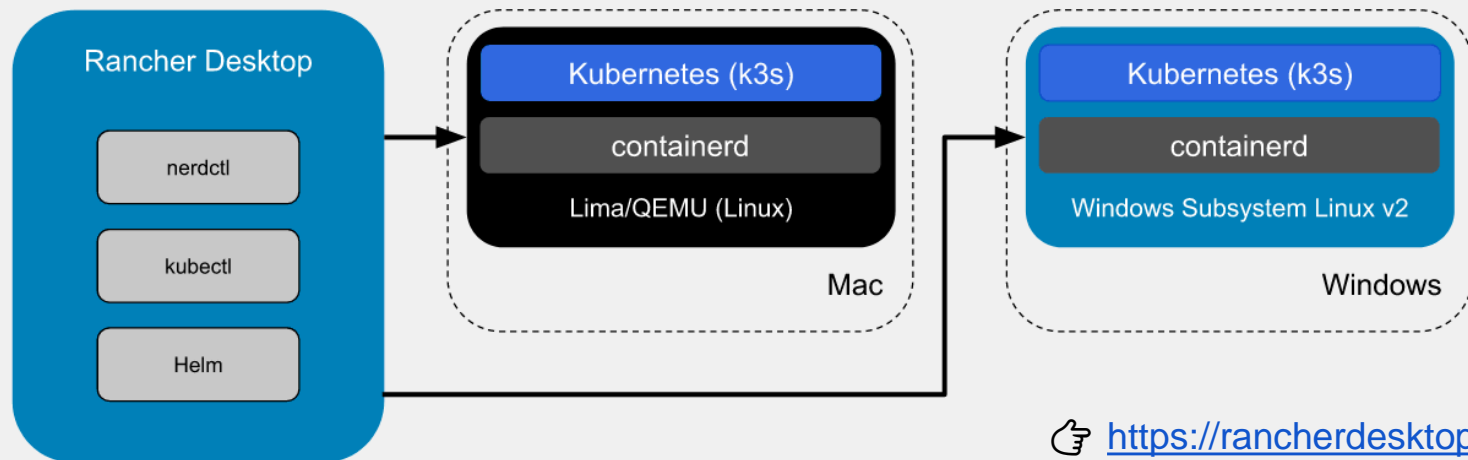
Tool aggiuntivi



Una alternativa: *Rancher Desktop*

How it Works

Rancher Desktop is an electron based application that wraps other tools while itself providing the user experience to create a simple experience. On MacOS Rancher Desktop leverages a virtual machine to run containerd and Kubernetes. Windows Subsystem for Linux v2 is leveraged for Windows systems. All you need to do is download and run the application.



👉 <https://rancherdesktop.io/>

Corsi e formazione



1 giornata

DevOps

Introduzione a Docker

Docker è una piattaforma per container per costruire e gestire la più ampia scala di applicazioni dalla fase di sviluppo a quella di messa in produzione sia onpremise che nel cloud.

con Claudio Piffer

Dettagli Corso ➔



CORSI DEVOPS

IN PRESENZA E ONLINE

aws

wintech italia

Azure

👉 <https://www.wintech-italia.it/corsi/devops/docker-base/>

Libri consigliati



end.

Q & A





THANK YOU!