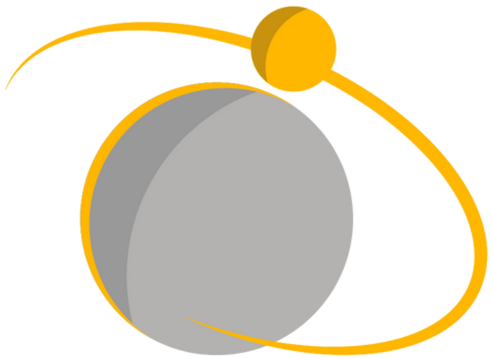
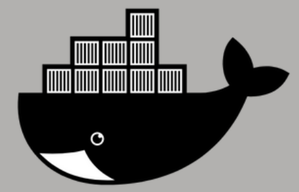


PERIGEA

Docker



Docker
Agenda

- Cos'è Docker
- Dockerfile & Docker images
- Docker volumes
- Docker networks
- Docker compose

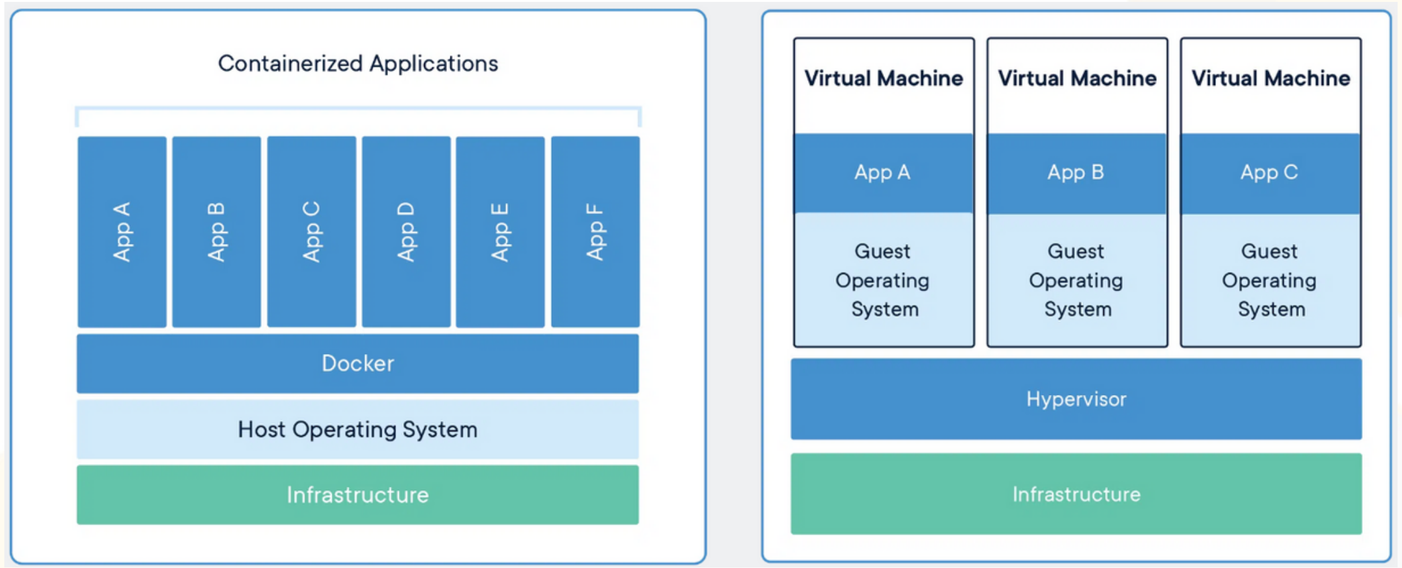
Cos'è Docker?
Container

Docker è una piattaforma Open Source per la creazione, la gestione e il deploy di **container**

Un **container** è un'unità standardizzata contenente tutto il necessario (librerie, utilities di sistema, codice...) per eseguire una determinata applicazione in maniera consistente tra diversi ambienti

Cos'è Docker?
Container

- Condivide kernel e risorse con il sistema host
- Più leggero di una VM (che ospita l'intero sistema al suo interno)
- Docker utilizza **runC** per la gestione di namespace, delle capabilities, dei cgroups etc
- Lo standard è definito dalla W3C attraverso l'**Open Container Format**



Cos'è Docker?
Container

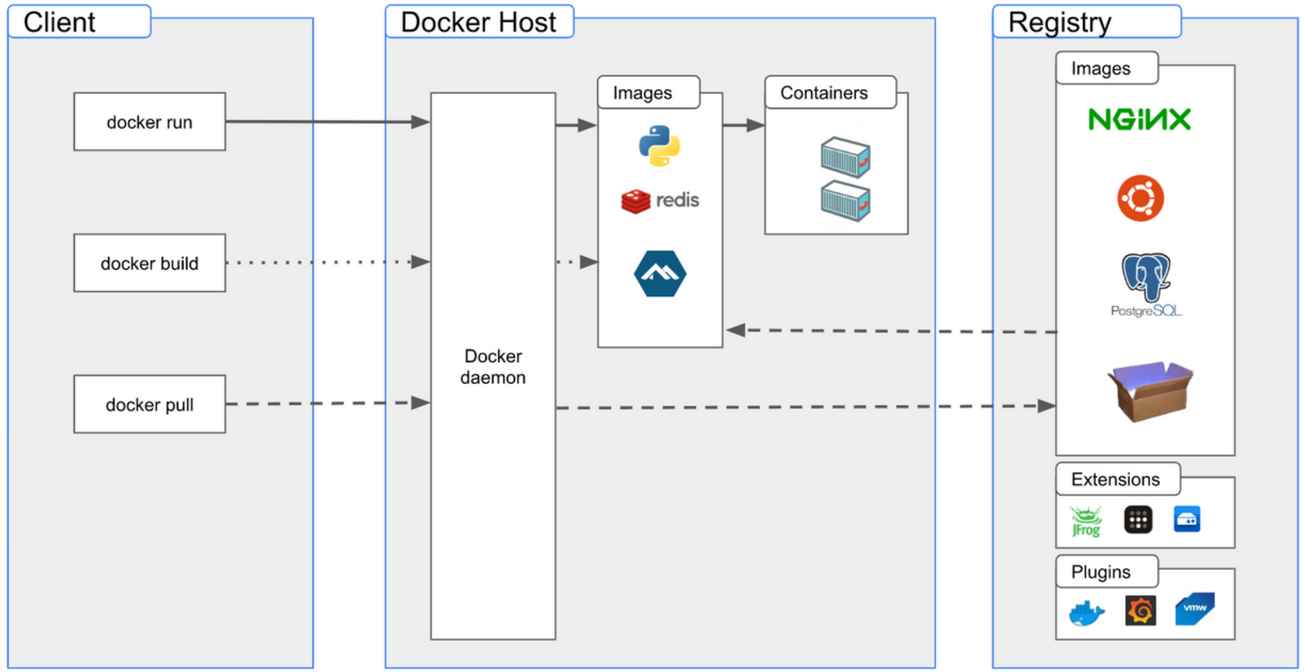
Il kernel utilizzato da un container Docker è il medesimo del sistema host su cui viene lanciato

```
beard@laniakea:~$ uname -r
5.10.16.3-microsoft-standard-WSL2
beard@laniakea:~$ docker run alpine uname -r
5.10.16.3-microsoft-standard-WSL2
```

Cos'è Docker?
Architettura

Architettura *client-server* composta da 3 elementi principali:

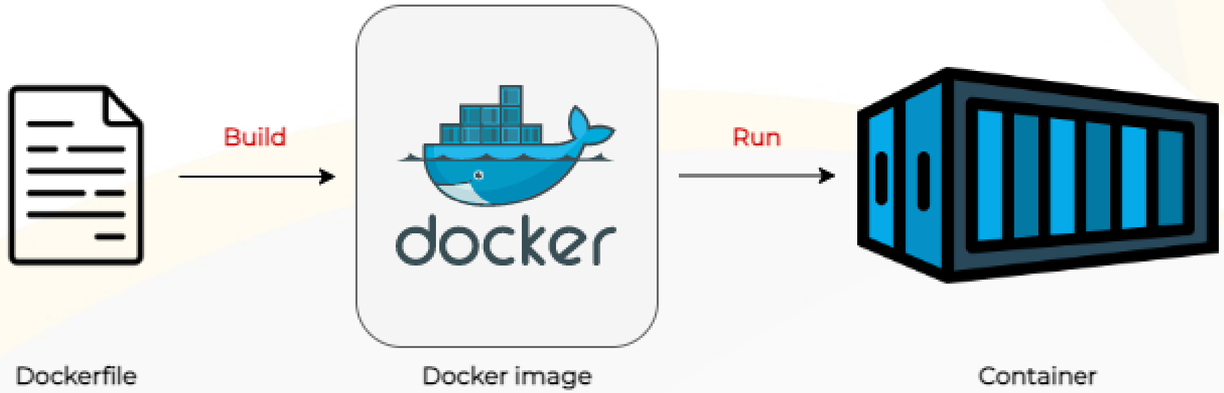
- **Deamon:** (*dockerd*) gestisce le richieste API
- **Client:** (*docker* o *docker-compose*) interagisce con le API esposte dal server inviando comandi al deamon
- **Registry:** è un raccoglitore di immagini docker, utilizzato ad esempio in caso di *pull*



Docker Images
Dockerfile

I **Dockerfile** descrivono **come** un'immagine vogliamo che sia costruita, andandone a specificare i principali parametri quali:

- Immagine di base
- Label varie
- Directory di lavoro
- Comandi da eseguire
- Porte da esporre
- ...



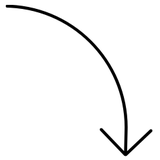
Docker Images
Dockerfile

1. **FROM:** specifica la *base-image*, ovvero l'immagine di partenza (priva di parent)
2. **LABEL:** possibilità di specificare labels di vario genere, tra cui autore o versione
3. **RUN:** esegue un comando in un layer al di sopra dell'ultimo e genera un'immagine per il layer successivo
4. **WORKDIR:** specifica la working directory del container
5. **COPY**
6. **ENTRYPOINT:** permette di rendere il container "eseguibile", a partire dal comando specificato
   CMD: specifica un argomento di default per ENTRYPOINT, può essere soltanto uno per Dockerfile

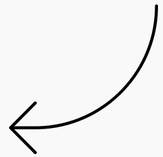
```
# Image pulling
FROM ubuntu:23.04                                   1

# Author label
LABEL Author="andrea.guzzon@perigea.it"             2

# Update packages
RUN apt update && apt upgrade -y                    3

# Set the working directory
WORKDIR /home                                       4

# Copy the shell script
COPY hello.sh .                                     5

# Run the shell script
ENTRYPOINT [ "/bin/bash" ]                          6
CMD [ "./hello.sh" ]
```

Docker Images
Dockerfile

```
docker build -t ubuntu-test . --no-cache
```



```
[+] Building 13.3s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:23.04
 => CACHED [1/4] FROM docker.io/library/ubuntu:23.04@sha256:008c0f6712067722f42f9685476d37b0b7b689e31d66e5787d1920c7ac230849
 => [internal] load build context
 => => transferring context: 29B
 => [2/4] RUN apt update && apt upgrade -y
 => [3/4] WORKDIR /home
 => [4/4] COPY hello.sh .
 => exporting to image
 => => exporting layers
 => => writing image sha256:e508000832ffbf8f13fc6ba154e9ecfc9415989cebd6f8cf0f9863cb462c747c
 => => naming to docker.io/library/ubuntu-test
```



```
beard@laniakea:~$ docker image ls | grep ubuntu-test
ubuntu-test                 latest      e508000832ff
```

Docker Images
Layers

I **layer** sono i "mattoncini" che compongono un'immagine. Possiamo immaginare ogni layer come una specie di "diff" rispetto all'immagine precedente
Viene generato un nuovo layern **ad ogni istruzione che apporta modifiche all'immagine precedente**

Solitamente, i layer vengono salvati in una cache locale per facilitare il rebuild delle immagini.

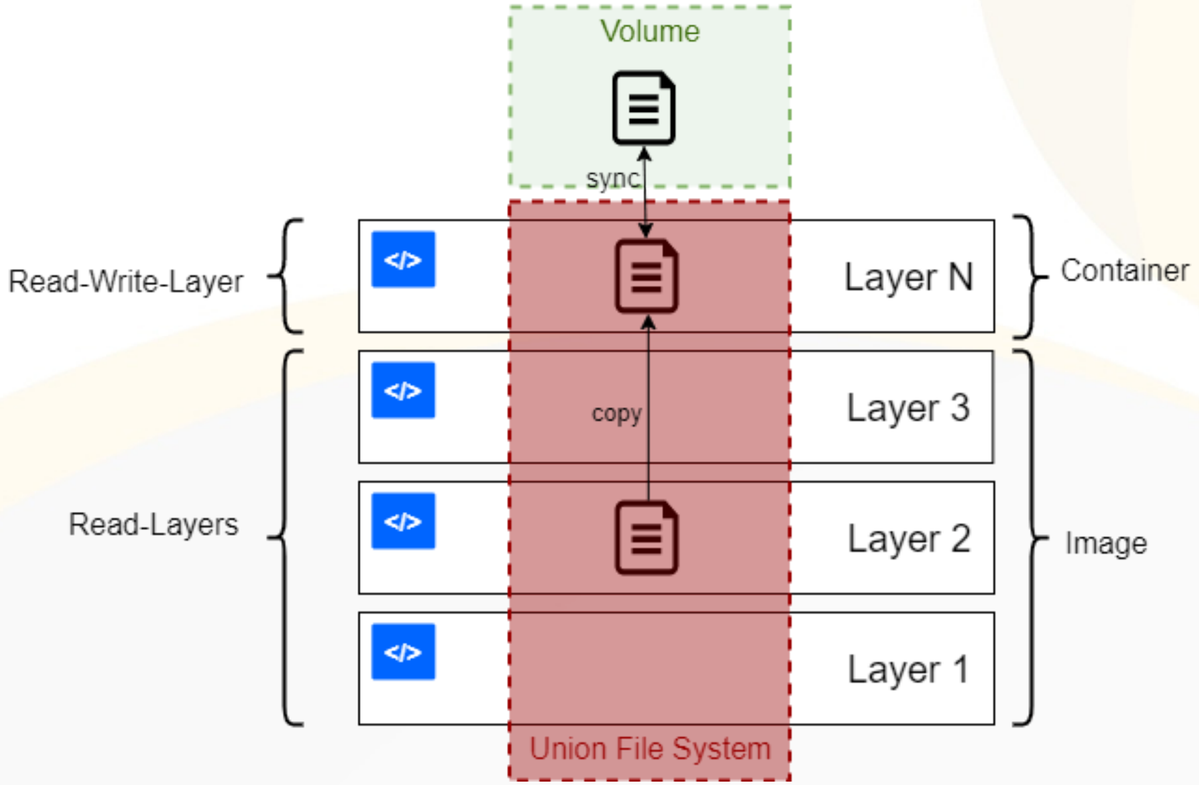
È possibile fare lo **squash** di un'immagine (prende le modifiche dei singoli layer e li compatta in un'unica immagine)

```
[+] Building 13.3s (9/9) FINISHED
 => [internal] load build definition from
 => => transferring dockerfile: 32B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.i
 => CACHED [1/4] FROM docker.io/library/u
 => [internal] load build context
 => => transferring context: 29B
 => [2/4] RUN apt update && apt upgrade -
 => [3/4] WORKDIR /home
 => [4/4] COPY hello.sh .
 => exporting to image
 => => exporting layers
 => => writing image sha256:e508000832ffb
 => => naming to docker.io/library/ubuntu
```

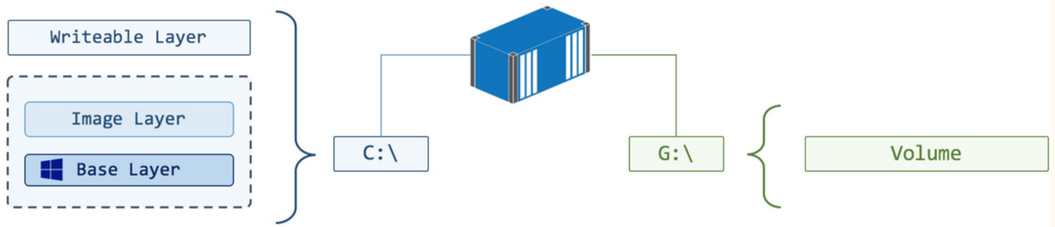
Docker Volumes
Cosa sono i volumi?

Sono il modo in cui possiamo **persistere** dati dal container
al sistema Host (e a differenza del *bind mount* NON sono dipendenti dal filesystem del sistema host)

Docker crea in automatico un layer r/w al di sopra dei layer read-only nella generazione delle immagini che **viene eliminato alla terminazione del container**



Docker Volumes
Creare un volume



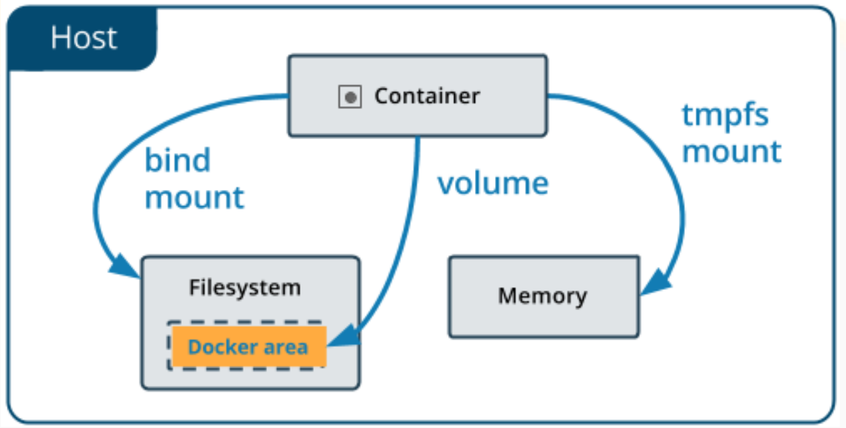
Possiamo creare un volume tramite il comando:

```
docker volume create my_volume
```

E andare a montarlo quando eseguiamo un container attraverso il comando

```
docker run -v my_volume:/path/to/folder my_container
```

Dove, separati da **:**, abbiamo il nome del volume e il path del container verso cui il volume verrà montato



Docker Networks
Cos'è una rete?

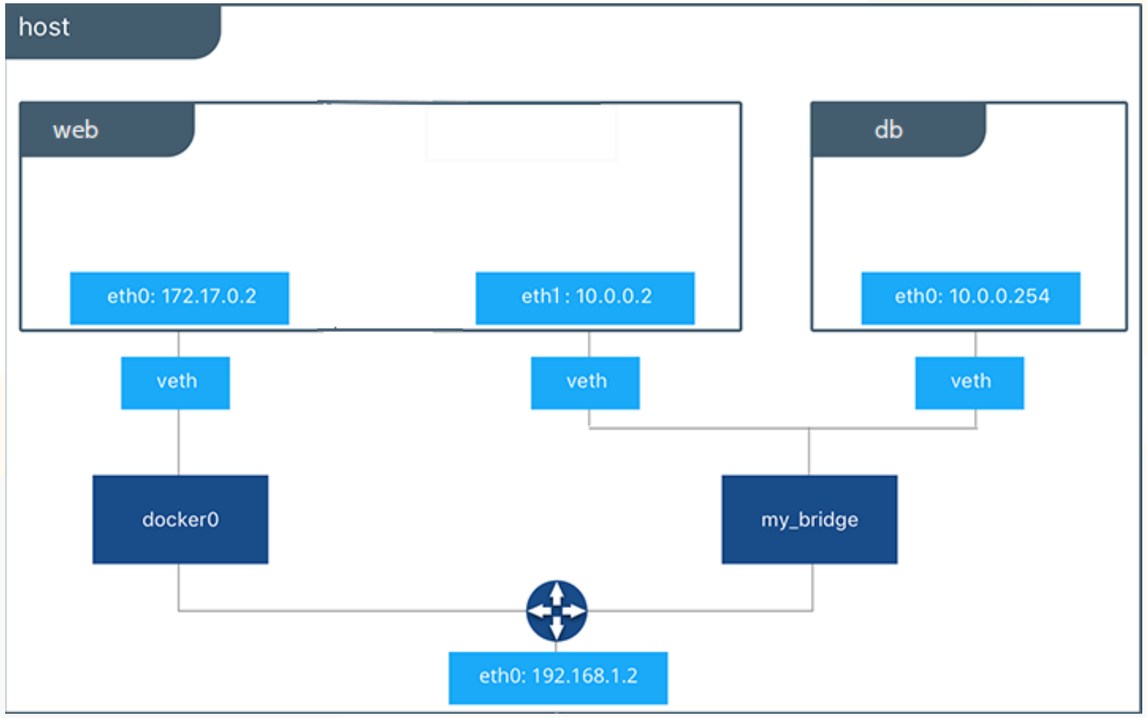
È l'interfaccia che permette ai container di "vedersi", isolarsi e comunicare tra di loro o con l'host (attraverso l'utilizzo di *veth*, ovvero Virtual ETHernet devices)

Possiamo creare una rete con il comando:

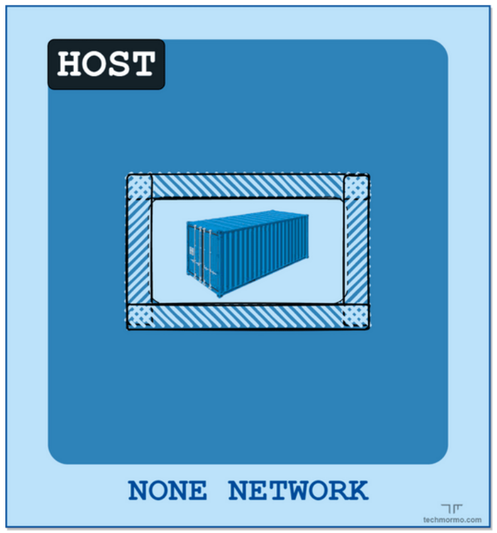
```
docker network create my_network
```

Esistono **4** interfacce di rete in Docker:

- **None**
- **Host**
- **Bridge**
- **Overlay**

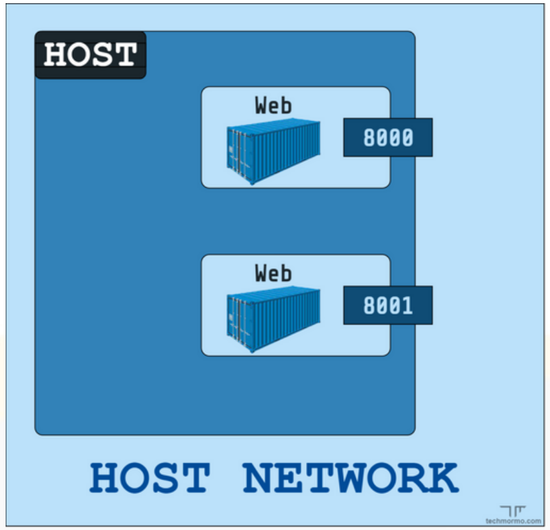


Docker Networks
Tipi di rete



Disabilita la rete per un container, che pertanto risulta isolato dal sistema

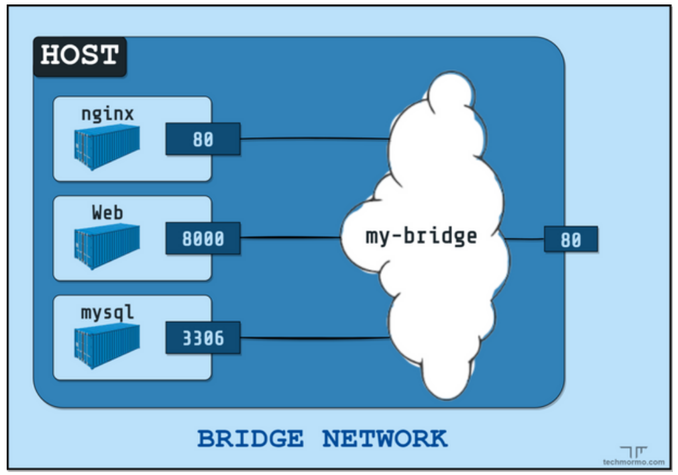
```
docker network create --driver=none my_network
```



Condivide la rete con l'host system, apparendo a tutti gli effetti come se il container stesso fosse parte della rete del sistema host

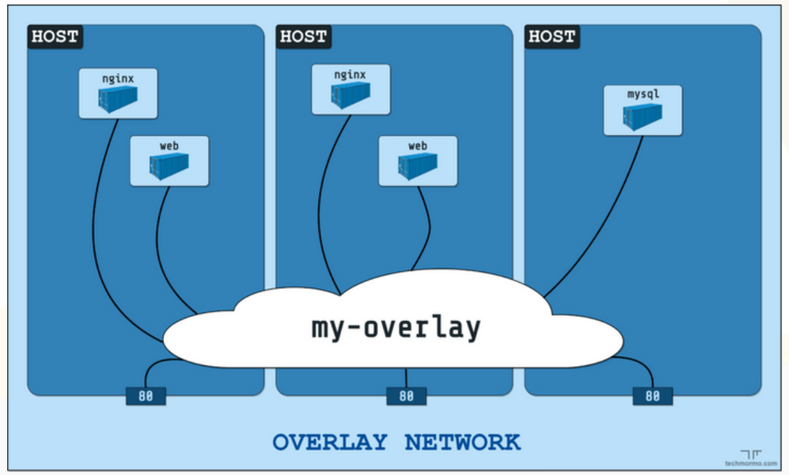
```
docker network create --driver=host my_network
```

Docker Networks
Tipi di rete - (2)



Crea una sottorete all'interno del sistema host, in cui i container *interni* alla suddetta si vedono ma sono isolati dai container esterni

```
docker network create --driver=bridge my_network
```



Crea una rete distribuita che permette la comunicazione anche di diversi host system, pertanto è la via preferibile nel caso di cluster multi-host

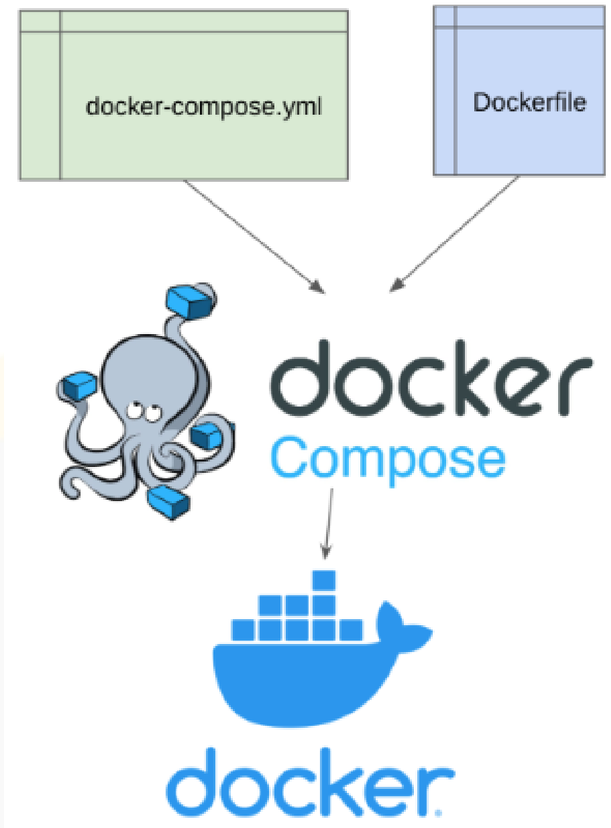
```
docker network create --driver=overlay my_network
```

Docker Compose
Cos'è docker-compose

È un tool per la creazione e gestione di applicazioni Docker *multicontainer* (è infatti possibile specificare più immagini e servizi nello stesso file)

Utilizza **yml** come formato e permette di

- salvare la configurazione di uno (o più) container associati ad una (o più) immagini
- eseguire comandi
- creare/associare reti e volumi
- passare variabili al container stesso, permettendo di differenziare gli ambienti
- ...



Docker Compose
Dockerfile vs docker-compose

DOCKERFILE

- File plaintext per definire come **buildare** una immagine
- Partendo solitamente da una **base-image** permette di aggiungere layer ed eseguire comandi base
- **NON** ha possibilità di invocare il docker compose
- L'immagine creata può essere eseguita con opzioni attraverso *docker run <options>*

DOCKER-COMPOSE

- **Tool** per la gestione di applicazioni multicontainer
- File **yml** per la configurazione di servizi e comandi da eseguire
- Permette di definire oggetti comuni a più container **nello stesso file** (*networks, volumi etc)*
- **Ha accesso** alle immagini create tramite Dockerfile
- Posso utilizzare la stessa immagine per più docker-compose

Docker Compose
Esempio di docker-compose.yml

1. **version:** specifica la versione e la compatibilità del docker-compose
2. **services:** specificano in forma astratta una risorsa applicativa all'interno del docker-compose. DEVO dichiarare un elemento SERVICES radice
3. **volumes:** specificano i/il volumi/e che i container andranno ad utilizzare, mappandoli verso una directory
4. **ports:** mappano la porta esposta dal container con la porta dell'host (di modo che sia visibile esternamente)

```yaml
version: '3.6'                                          1
services:
  docker-spring-service:                                2
    container_name: docker-spring
    image: docker-spring
    volumes:
      - example-app:/data/docker-demo                   3
    ports:
      - "8080:8080"                                     4
    networks:
      - workshop-network                                5

networks:                                               6
  workshop-network:
    external: true
    driver: bridge
volumes:                                                7
  example-app:
```

Docker Compose
Esempio di docker-compose.yml (2)

5 **networks:** connette il container ad una rete, specificata

6 **networks:** fornisce le specifiche della rete a cui vengono connessi i services (e.g. il tipo di driver da utilizzare)

7 **volumes:** specifica le caratteristiche dei volumes che vengono montati/creati in fase di creazione del container

```yaml
version: '3.6'                                          1
services:
  docker-spring-service:                                2
    container_name: docker-spring
    image: docker-spring
    volumes:
      - example-app:/data/docker-demo                   3
    ports:
      - "8080:8080"                                     4
    networks:
      - workshop-network                                5

networks:                                               6
  workshop-network:
    external: true
    driver: bridge
volumes:                                                7
  example-app:
```

Docker Compose
Live demo


```
@RestController
public class simpleController {

    @GetMapping("/greetings/")
    public String getGreeted(@RequestParam String name) {
        return String.format("Hello %s!", name);
    }
}
```

Docker
Domande