UNIVERSITÀ DI PARMA

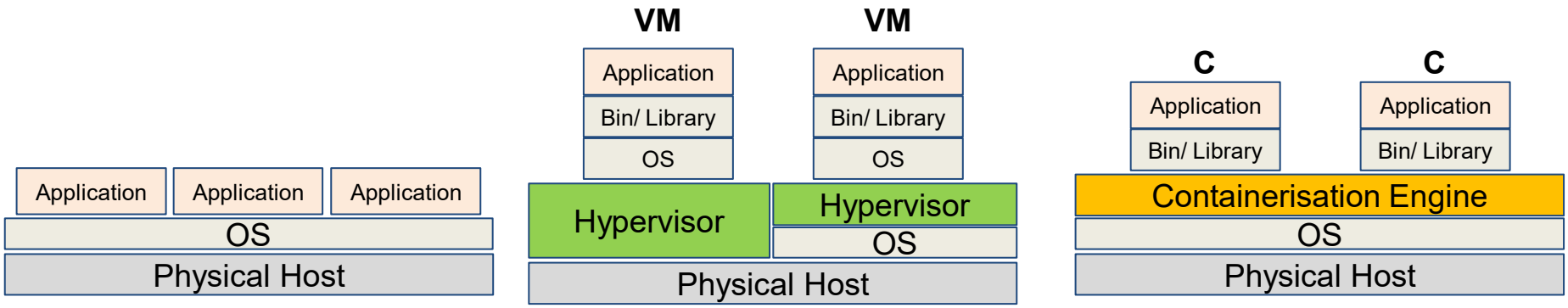
DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE ED INFORMATICHE
Corso di Laurea in Informatica

Docker 1 : Introduzione

LABORATORIO DI RETI DI CALCOLATORI - a.a. 2023/2024

Roberto Alfieri

Container vs Virtualizzazione



- **Virtualizzazione** : l'HV emula l'hardware su cui si appoggia il Sistema Operativo
  - Maggiore flessibilità: diversi sistemi operativi completamente isolati tra loro

- **Container**: sistema operativo è condiviso tra i container
  - Leggeri. Minori tempi di avvio e minore memoria richiesta sull'Host.
  - **Docker** è una piattaforma disponibile come software libero per Linux, MacOsX o Windows, per la gestione ed esecuzione di container. https://www.docker.com/

Il progetto Docker

- Progetto OpenSource https://www.docker.com/ in cui i contenitori possono essere eseguiti il locale o su cloud pubbliche o private.
- Nativo in ambiente GNU/Linux.
- Disponibile anche per MacOS e Windows. L'engine Docker utilizza la virtualizzazione per eseguire un kernel Linux
- L'engine Docker si basa sui moduli Control Group (cgroups) e Namespaces di GNU/Linux
  - Cgroups: E' un resource controller che consente di limitare le risorse associate al ogni container.
  - Namespaces: Garantisce l'isolamento dei processi impedendo ad un container di accedere alla macchina Host e agli altri container.

DOCKER: Architettura

Una immagine è un template read-only che include le informazioni necessarie per creare un container che possa girare sopra una piattaforma docker.

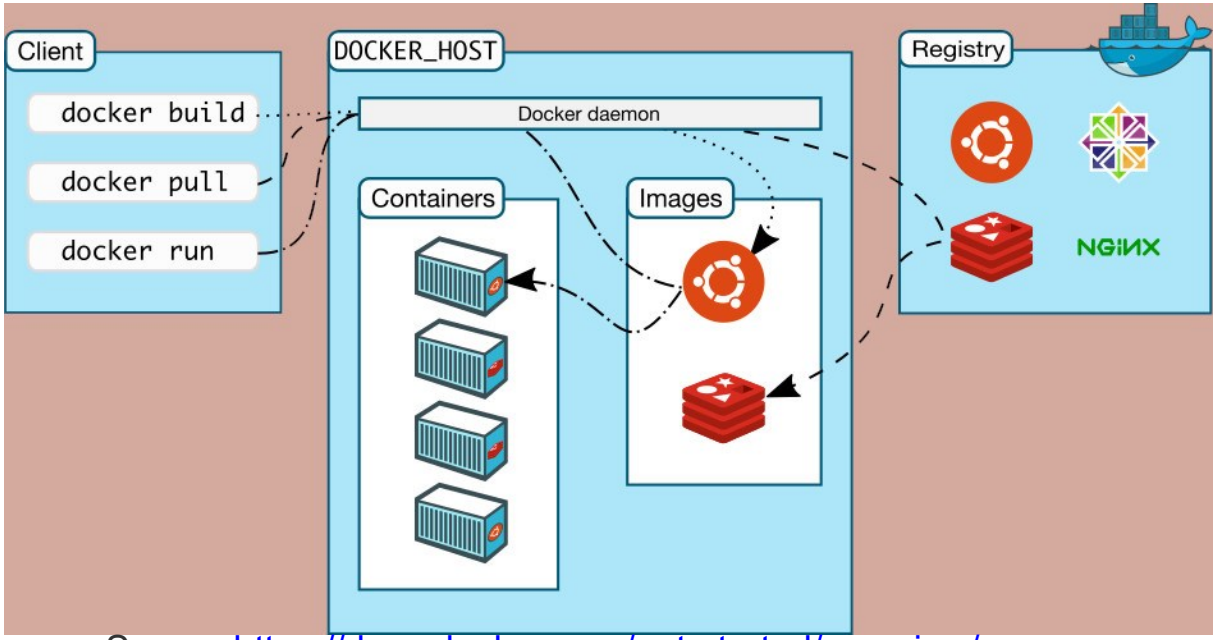
Registry è una applicazione stateless che consente di memorizzare immagini.

Docker Hub è un servizio di Registry fornito da Docker

Un **Docker host (server)** è un computer fisico o virtuale Linux su cui vengono eseguiti i container.

- L'attività richiede l'esecuzione di un **docker daemon**.
- Docker host gestisce anche un **local registry** che ospita le immagini necessarie.

Docker client è l'interfaccia utente. Può coincidere con Docker host.



Source https://docs.docker.com/get-started/overview/

DOCKER: Ciclo di vita

Una immagine salvata può essere utilizzata per **creare** un container (aggiungendo un layer scrivibile all'immagine), che poi verrà avviato con **start.**

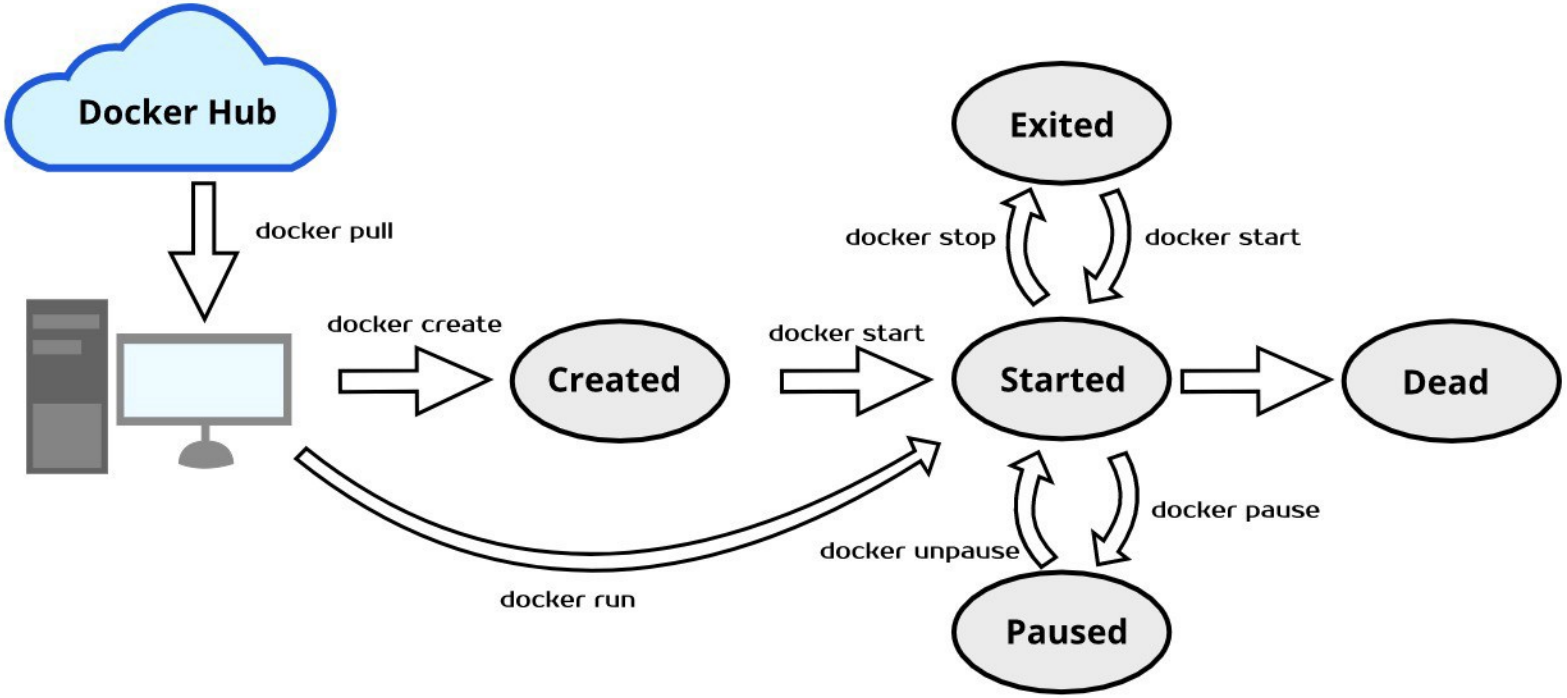
Un container può essere creato e eseguito con il singolo comando **run**

Un container in esecuzione può essere messo in **pausa (pause/unpause)**

Dopo avere eseguito l'entrypoint il container termina **(exited)**

Il container può essere terminato con il comando **stop**

Il container terminato rimane in memoria e può essere riavviato **(start)** o cancellato **(rm)**



Installazione di Docker client e server su Linux

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

Avvio del daemon:

service docker start
service docker status

docker version
Client: Docker Engine - Community
 Version: 20.10.4
 API version: 1.41
 Go version: go1.13.15
…..

Server: Docker Engine - Community
 Engine:
  Version: 20.10.4
  API version: 1.41 (minimum version 1.12)
  Go version: go1.13.15
…..

Gestione delle immagini

Comandi principali con utilizzo dell'immagine ubuntu/nginx :

docker images # lista immagini del local registry

docker search ubuntu/nginx # ricerca in Docker Hub
docker pull ubuntu/nginx # immagine copiata da Docker Hub a local registry

docker inspect ubuntu/nginx # informazioni sul contenuto dell'immagine
….

docker image rm ubuntu/nginx # rimuove l'immagine dal local registry

docker image prune # rimuove tutte le immagini inutilizzate

Gestione dei container

Il modo più semplice per scaricare (se necessario) l'immagine, creare ed eseguire un container è attraverso il comando **run**:

docker run -d ubuntu/nginx # -d modalità detached
docker run -d --name ng1 ubuntu/nginx

docker ps # elenco dei containers attivi
docker ps -a # elenco di tutti i containers (status: running, exited, paused, ..)

Ogni container ha un nome, un comando in esecuzione (entrypoint) e un identificativo (ID)

docker ps -aq # stampa ID di tutti i container
docker stop ng1 # ferma il container
docker start ng1 # riavvia il container
docker exec ng1 ps -ef # eseguire un commando nel container in esecuzione
docker exec -it ng1 bash # shell interttiva nel container in esecuzione

docker rm $(sudo docker ps -aq) # rimuove tutti i container (tranne i running)

docker logs <containter> # stampa i log di un container

Creare immagine nginx-ssh da container

Vogliamo arricchire il containter, con comandi di rete e servizio ssh e creare una nuova imagine che chiameremo nginx-ssh

```
docker exec -it  ng1 bash
ng1>  apt update
ng1>  apt install net-tools nmap traceroute iptables curl nano
ng1>  apt install openssh-server
ng1> /etc/init.d/ssh start            # avvio di ssh
ng1>  ssh ubuntu@localhost       # verifica di ssh
ng1> netstat -tupan                 #  lista porte in ascolto
ng1> ip a                                # visualizza ip address (ad esempio 172.17.0.2)

# per l'avvio automatico di ssh:   editare /docker-entrypoint.sh
# aggiungere     /etc/init.d/ssh start      dopo  #!/bin/sh

ng1> exit                                  # esci dal container

docker commit ng1 nginx-ssh          # crea nuova imagine nginx-ssh  dal container ng1
docker rm -f ng1                          # eventuale eliminazione del container ng1
```

Creare un container da immagine nginx-ssh

attiviamo un container dalla nuova imagine e facciamo qualche verifica:

docker run   -d --name ns2 nginx-ssh
docker exec -it ns2 bash

ns2> netstat -tupan
ns2> ip a

Creare una immagine da dockerfile

Dockerfile è l'insieme di istruzioni necessarie per costruire automaticamente una imagine
Le istruzioni sono composte da righe di testo che iniziano con una parole chiave.
https://docs.docker.com/engine/reference/builder/

Nel seguente esempio creiamo l'immagine nginx-ssh utilizzando dockerfile:
https://www.cyberciti.biz/faq/how-to-install-openssh-server-on-alpine-linux-including-docker/

```
> cat dockerfile
FROM ubuntu/nginx
RUN apt  update
RUN apt   install -y nmap tcpdump iptables curl net-tools whois traceroute tshark
RUN apt   install -y openssh-server  nano
RUN echo -n 'ubuntu:ubuntu' | chpasswd
COPY docker-entrypoint.sh /
RUN chmod +x  /docker-entrypoint.sh
EXPOSE 22
```

Per il file docker-entrypoint.sh vedi slide precedente

```
docker build -t nginx-ssh  .
```

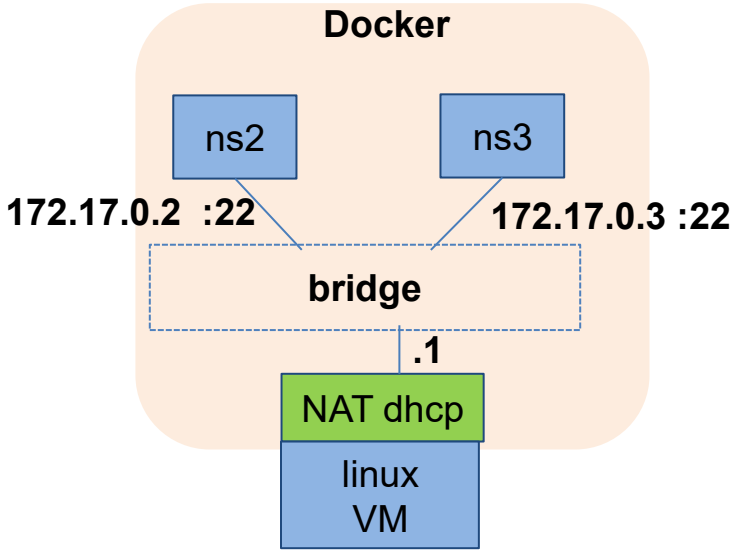
Verifica

Dall'immagine nginx-ssh creiamo due container ns2 e ns3 e verifichiamo il funzionamento del servizio ssh.

docker run -d --name ns2 nginx-ssh
docker run -d --name ns3 nginx-ssh
docker ps
docker exec -it ns2 bash

ns2> ip a
ns2> netstat -tupan
ns2> nmap 172.17.0.0/24

ns2> ssh ubuntu@172.17.0.3



Docker-compose

Docker Compose è un tool che consente di gestire i propri container salvandone la configurazione di istanziamento in un unico file di configurazione in formato YAML che per default prende il nome di docker-compose.yml

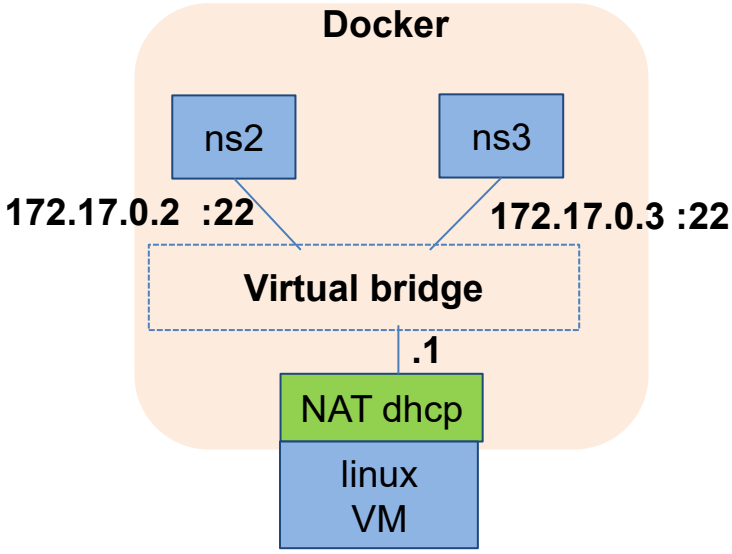
https://docs.docker.com/compose/reference/

docker-compose.yaml: build ns2 e ns3

```
version: "3"

services:
  ns3:
    container_name: ns3
    build:
      context: .
      dockerfile: Dockerfile
    image: nginx-ssh
    network_mode: bridge

  ns2:
    container_name: ns2
    build:
      context: .
      dockerfile: Dockerfile
    image: nginx-ssh
   network_mode: bridge
```



run di ns2 e ns3 con docker-compose

```
docker-compose up -d   # esegui in modo detach
docker-compose ps      # verifica che i due container sono in esecuzione

docker exec -it ns2 sh   #
ip a                     # veridica l'inidirizzo ip
netstat -tupan           # verifica che il servizio ssh e' attivo
ssh ubuntu@172.17.0.3

docker-compose stop    # ferma i container
docker-compose rm      # rimuove i container

docker compose down    # ferma e rimuove
```