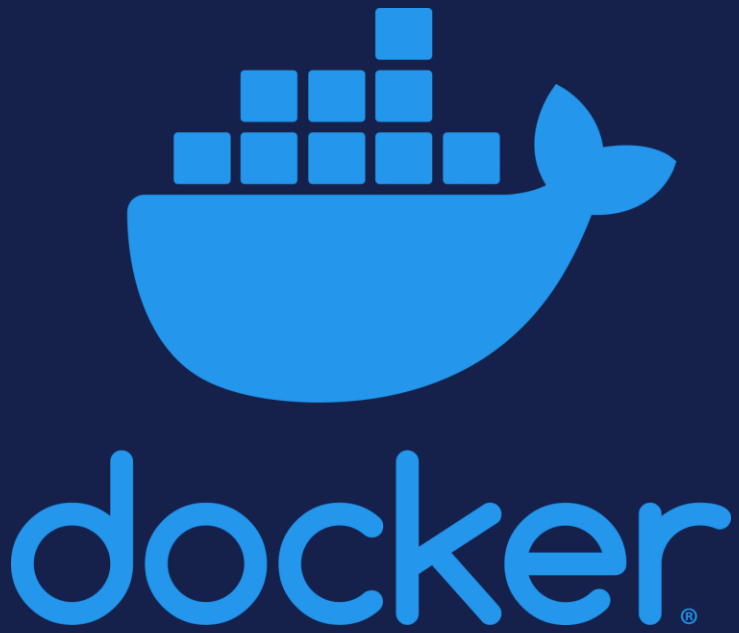
Docker

Ce este docker?

Docker este o platformă de containerizare open source. Permite dezvoltatorilor să împacheteze aplicații în containere . Această tehnologie a fost dezvoltată pentru prima dată în 2013 și este din ce în ce mai răspândită de atunci. Containerizarea este mult mai eficientă din punct de vedere al consumului de resurse hardware decât virtualizarea.



Avantajele docker

Scalabilitate - multe containere pot fi plasate într-o singură gazdă

Eficiență - Containerele rulează ca niște procese și nu consumă multă putere de procesare

Flexibilitate - Ușurință în mutarea și întreținerea aplicațiilor

Securitate mai bună, acces mai mic necesar pentru a lucra cu codul ce rulează în containere. De asemenea, mai puține dependențe de software

Instalare docker

Pentru a instala docker se pot urmării pașii prezentați aici

Se poate verifica faptul că toți pașii au fost instalați corect cu comanda:
sudo docker run hello-world

Pentru a putea rula comenzile fără sudo, de pe userul nostru curent, va fi nevoie să adăugăm acel user pe grupul docker. Putem face asta cu comanda
sudo usermod -aG docker $USER

Instalare docker prin script

O metodă mai simplă de instalare a docker este printr-un script. Tot ce trebuie să facem este să rulăm acel script. Pentru alte detalii se poate consulta documentatia oficială de docker. Pentru a instala se vor rula cele 2 comenzi:

1) curl -fsSL https://get.docker.com -o get-docker.sh
2) sh get-docker.sh

Din nou un singur pas a mai ramas și anume să dam drepturi userului nostru de a rula comenzi de docker.
sudo usermod -aG docker $USER

Ce sunt imaginile Docker ?

Imaginile Docker acționează ca un set de instrucțiuni pentru a construi un container docker (este un șablon)

De exemplu putem avea o imagine de docker pentru nginx, mysql sau chiar ubuntu. Pe baza acestei imagini noi putem porni aplicația
(containerul de docker)

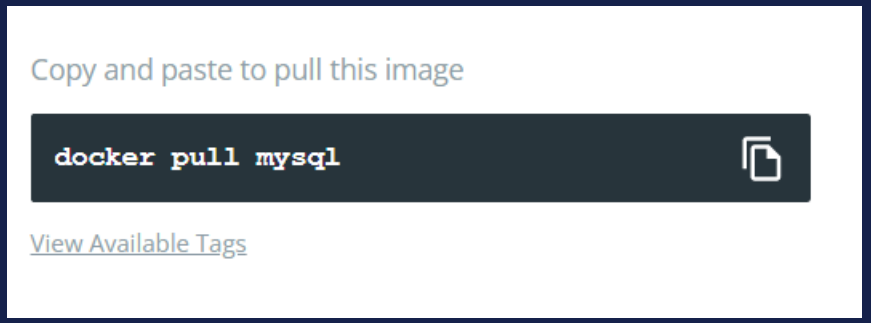
Putem verifica imaginile de docker de pe server cu comanda:
docker images

Cum descărcăm o imagine ?

Se poate descarca cu comanda: **docker pull nume_imagine:tag**

Dacă nu vom preciza un tag anume, implicit se va utiliza tag-ul **latest**. Pentru a vedea toate versiunile este nevoie să apăsăm pe "**view available tags**"

Putem vedea imaginea de docker ca pe un cd cu windows și containerul de docker ca navigarea și utilizarea calculatorului după instalarea acestuia



IT School
FROM ZERO TO HERO!

Cum verificăm imaginile descărcate?

Acestea se pot descarca cu comanda: **docker images ls**

Aici vom vedea mai multe informații precum: id-ul imaginii, data, dimensiunea imaginii

Pentru a șterge o imagine utilizăm: **docker image rm**

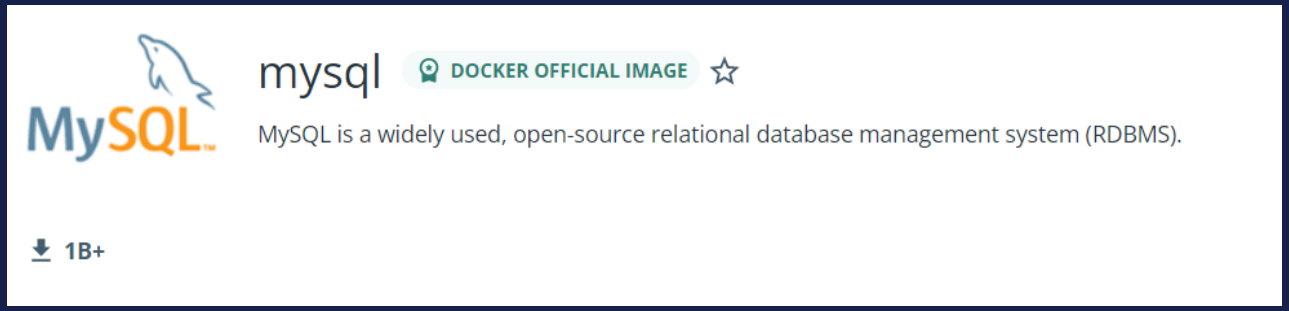
De asemenea, putem adauga un tag unei imaginii cu comanda:
docker image tag existing_image:existing_tag new_image:new_tag

Dockerhub

Putem descărca o imagine dintr-un repository (public sau privat). Cel mai cunoscut repository de docker este Dockerhub.

Acesta este un repository public, unde fiecare poate contribui și crea imagini de docker. Putem găsi de exemplu imagini oficiale de mysql , dar și tot felul de versiuni custom neoficiale .

Pentru a evita orice vulnerabilitate este recomandat să se utilizeze doar imaginile din surse oficiale. Acestea pot fi recunoscut deoarece conține “Docker official image”



IT School
FROM ZERO TO HERO!

Încărcarea unei imaginii pe Dockerhub

Dacă avem un cont de dockerhub putem face upload acelei imagini. Inițial după ce ne creăm un cont va fi nevoie să ne logăm și din serverul nostru. Se poate face asta cu **docker login.**

Putem uploada imagini de docker cu comanda:
docker push [OPTIONS] NAME[:TAG]

Pentru a redenumi o imagine , o putem face cu comanda:
docker tag OldName:tag NewName:tag

IT School
FROM ZERO TO HERO!

Încărcarea unei imaginii pe Dockerhub

Pentru a încărca imaginea în repository-ul nostru de dockerhub este nevoie ca imaginea să fie denumită după userul nostru. De exemplu dacă userul se numește **user513 și imaginea httpd:2.3 vom putea încărca imaginea cu:**

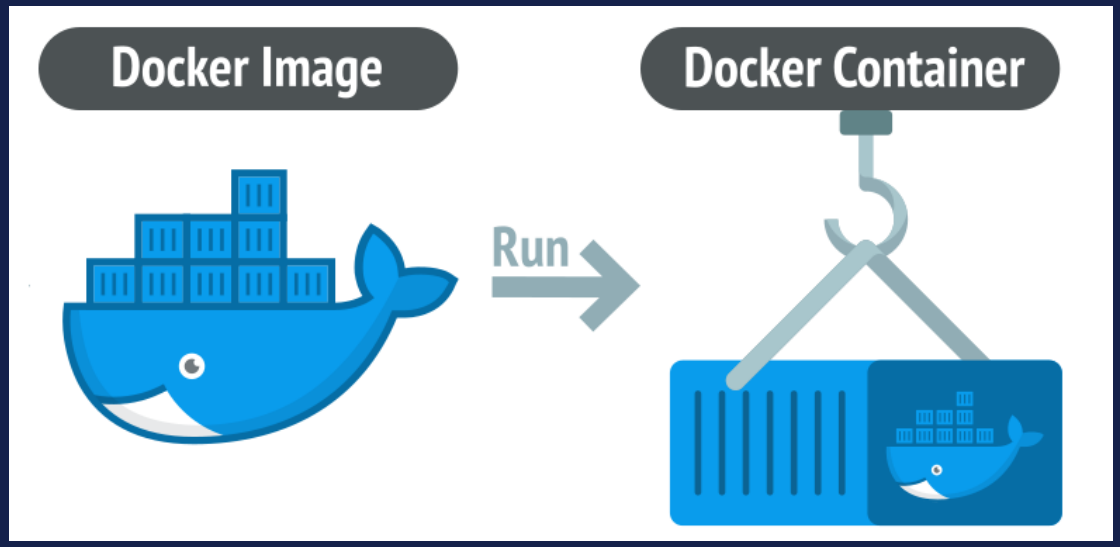
docker push user513/httpd:2.3

Vom putea apoi accesa dockerhub și vedea imaginea noastră.

Containere de docker

Ce este un container? Un container de docker este o instanță a unei imaginii Puteți crea, porni, opri, muta sau șterge un container utilizând API-ul Docker sau CLI .

Putem porni un container cu comanda: **docker run nume_imagine:numar_tag**



Sursă imagine

IT School
FROM ZERO TO HERO!

Containere de docker

De exemplu putem rula comanda**: docker run httpd**

Astfel cu această comandă rulăm containerul și ne conectăm în shell-ul acelui container. Pentru a putea porni containerul și în același timp să rămânem în shell-ul curent trebuie să utilizăm –d (detached)

Putem vedea containele ce rulează cu comanda: **docker container ls**

A apărut vreun proces pentru containerul nou dacă verificăm cu ps –ef ?

Containere de docker

Pentru a putea vedea și containerele care au fost active și care acum sunt oprite cu comanda: **docker container ls -a**

Pentru a opri un container folosim **docker stop id_container/nume_container**

Pentru a porni un container: **docker start id_container/nume_container**

Containere de docker

Atunci când pornim un container în modul prezentat mai sus vom putea vedea faptul că i s-a asignat un nume aleatoriu. Putem totuși specifica și un nume custom pentru acel container .

docker run --name my-redis -d redis

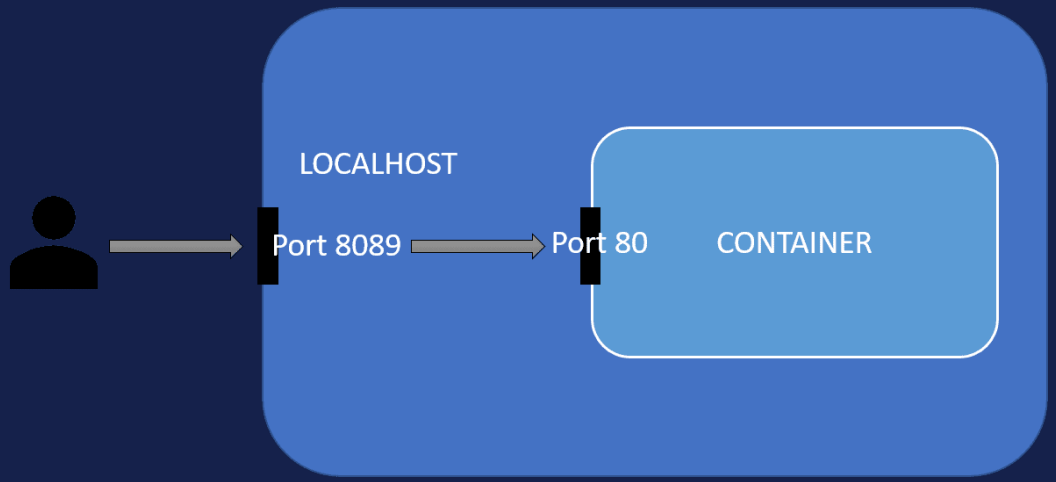
Putem de asemenea să mapăm un port pe acel container (aplicația http rulează pe portul 80, dar acesta va fi în interiorul containerului). Putem mapa portul 8080 de pe hostul nostru local, cu portul 80 din interiorul containerului de docker:

docker run -p 8080:80 httpd

Port mapping

Atunci când rulăm un container putem folosi flag-ul **–p** pentru a face mapare între porturile ce rulează la serverul nostru și cel care este în interiorul containerului .

docker run -p 8089:80 image_name



Sursa: https://www.code4it.dev/blog/run-mongodb-on-docker/

IT School
FROM ZERO TO HERO!

Volume

Să presupunem că avem un container de mysql. Dacă acel container se va închide vom pierde toate datele de pe el. De aceea, în unele cazuri dorim ca acel container să aibă date statice și dacă îl repornim să avem datele intacte. Pentru acest lucru există volumele.

Cum creăm un volum?

docker volume create my-vol

Pentru a lista volumele create:
docker volume ls

Pentru a vedea informații despre acel volum:
docker volume inspect my-vol

Volume

Cum pornim un container la care atașăm un volum?

docker run -d --name devtest -v myvol2:/app nginx:latest

Putem verifica informațiile despre container în care vom vedea și detaliile legate de volum cu comanda inspect:
docker inspect devtest

Bind mount

Bind mounts au o funcționalitate limitată în comparație cu volumele.

docker run -d -it --name devtest -v /home/myuser/Nginx_folder:/app nginx:latest

Networking

Containerele au propriul networking (implicit când veți crea un container va fi din rețeaua bridge). Asta inseamnă că are atașat și un ip din acea rețea. Puteți vedea aceste detalii cu **docker inspect nume_contaier/id_container**

Putem crea o rețea folosim comanda :
docker network create [OPTIONS] NETWORK
docker network create --driver=bridge --subnet=192.168.0.0/16 br0

Pentru a vedea rețelele existente cu comanda:
docker network ls

Putem verifica ip-ul containerului și executa comanda **ping** pentru a vedea daca ne raspunde

IT School®
FROM ZERO TO HERO!

Networking

Cu docker network connect/disconnect putem conecta sau deconecta un containter de la un networking (în acest caz cu numele multi-host-network). La fel și cu disconnect.

docker network connect multi-host-network container1
docker network disconnect multi-host-network container1

Networking

Cum cream un container într-un anumit networking:
docker run -itd --network=multi-host-network busybox

Pentru a vedea informatii despre mai multe tipuri de networking puteți verifica aici https://docs.docker.com/network/

Docker logs

Comanda docker logs afișează informațiile înregistrate de un container care rulează.

docker logs nume_container

```
[node1] (local) root@192.168.0.8 ~
$ docker logs fervent_elion
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/07/31 12:22:13 [notice] 1#1: using the "epoll" event method
2022/07/31 12:22:13 [notice] 1#1: nginx/1.23.1
```

IT School
FROM ZERO TO HERO!

Docker cp

Copiați fișiere/directoare între un container și sistemul de fișiere local. Este asemanator cu scp

docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH

Exemplu:
docker cp file.png mycontainer:/work

file.png va fi fișierul trimis pe containerul cu nume "**mycontainer**" în locația /work

Docker commit

Cu docker commit puteti sa creați o nouă imagine din modificările unui container

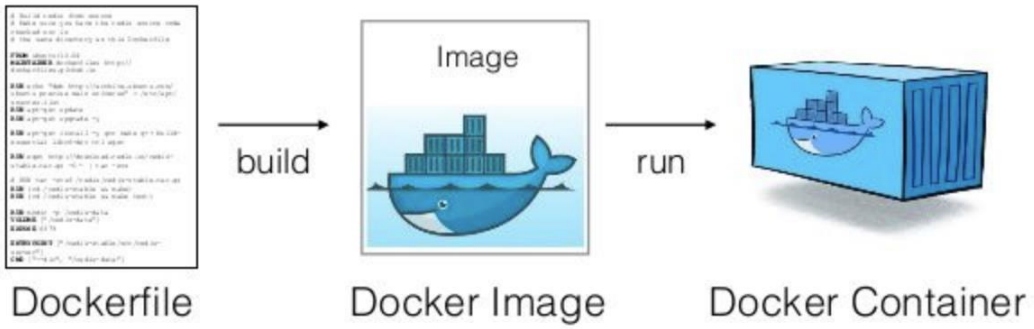
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

docker commit c3f279d17e0a svendowideit/testimage:version3

c3f279d17e0a reprezintă id-ul containerului
svendowideit numele userului de pe dockerhub
testimage numele imaginii
version3 tag-ul folosit

Dockerfile

Scopul dockerfile este acela de a avea imagini customizate de noi. Astfel putem avea aplicația deja configurată după bunul plac și doar trebuie să pornim containerul .



Sursa imagine

IT School
FROM ZERO TO HERO!

Dockerfile

Ce conține un Dockerfile?

FROM -> Cu această comandă specificăm sursa imaginii pe baza căreia dorim să o construim

 # -> Pentru a adăuga comentarii

COPY -> Cu aceasta comanda putem copia fișiere de pe serverul nostru în interiorul imaginii (și implicit să le găsim pe container atunci când îl pornim)

MAINAINER -> Pentru a specifica cine deține (cine este creatorul acelei imaginii)

ADD -> Funcție similară cu comanda COPY, dar în plus poate copia fișiere direct de pe URL-uri

Dockerfile

Ce conține un Dockerfile?

EXPOSE -> Putem expune anumite port-uri și să specificăm pe ce containere ar putea asculta acel container atunci când rulează

ENV -> Setează variabile de environment pe container

WORKDIR -> Setează locația în care vei fi direcționat atunci când pornești containerul

ENTRYPOINT și **CMD** -> setează valori implicite pentru un container care se execută

Dockerfile

```
FROM alpine:3.4

RUN apk update
RUN apk add vim
RUN apk add curl
ENV AWS_PROFILE="production"
COPY config.yml /root
```

Aceste comenzi vor fi într-un fișier numit **Dockerfile**

Care este utilitatea ?
De la o simplă imagine de alpine:3.4 , ne putem crea o imagine custom care să aibă pachetele de vim și curl deja instalate .

Pentru a construi o imagine de docker pe baza acelui fișier putem rula comanda: **docker build -t nume_imagine:numar_tag .**

Docker-compose

Compose este un instrument pentru definirea și rularea aplicațiilor Docker cu mai multe containere. Cu Compose, utilizați un fișier YAML pentru a configura serviciile unei aplicații. Apoi, cu o singură comandă, creați și porniți toate serviciile din acea configurație.

Exemplu:

```
version: "2"

services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./app:/var/www/html/
```

IT School
FROM ZERO TO HERO!

Docker-compose

Salvam textul de mai devreme într-un docker-compose.yml și îl pornim cu **docker compose up –d**

De aici putem se volume, crea un networking , da variabile de environment pentru containere și practic cu un singur fișier putem configura o întreagă infrastructură pentru aplicația dorită

Docker-compose

LINKS: Legăturile vă permit să definiți aliasuri suplimentare prin care un serviciu este accesibil de la un alt serviciu. De asemenea, putem folosi comanda build pentru a construi imaginea containerului web dintr-un dockerfile

```
version: "3.9 "

services:
  web:
    build: .
    links:
      - "db:database"
  db:
    image: postgres:14.1-alpine
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - '5432:5432
```

Docker-compose

Va fi necesar ca un fișier Dockerfile să fie prezent în aceeași locație cu docker-compose.yml

Acesta poate fi și simplu, de tipul: **FROM http**

De pe containerul de httpd , vom putea da ping acum in containerul de postgress prin mai multe metode

Ex ping db , ping database (numele dat de noi prin links) , ping root-db-1 (numele generat de docker compose pentru container)

Sfârșit