

1. **Pe baza la .jar, cream un docker file**

* A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image
* Il cream direct in folderul proiectului, si numele trebuie sa fie neaparat **Dockerfile**
* Docker file mereu incepe cu **FROM**, pentru a arata care e base sau parent image pentru ceea ce urmeaza sa facem
* Trebuie sa punem ca imagine de baza JDK, de ex, amazaoncoretto:17, unde 17 e versiunea la Java

FROM amazoncorretto:17

* Acum, definim metadata cu LABEL. Putem pune orice key-value si oricate cu LABEL

LABEL mentainer="Eduard"  
LABEL scope="test"

* Cream un directory in care se va gasi aplicatia:

WORKDIR /app

Se va crea un folderul app in container si orice comanda mereu va incepe cu el ca CD

* Copiem fisierele ce vrem sa fie puse in container cu COPY:

COPY Folder NumeInContainer

COPY /target/springboot-docker-0.0.1-SNAPSHOT.jar /app/application.jar

Deci, luam .jar din proiectul nostru si il mutam in container in app directoy si i dam numele application.jar

* Definim comanda de rulare la pornirea la container, prin ENTRYPOINT [“”,””,””]. Mare atentie la space intre ENTRYPOINT si []

Se va pune space intre fiecare element

* Deci, Dockerfile va fi asa:

FROM amazoncorretto:17  
  
LABEL mentainer="Eduard"  
LABEL scope="test"  
  
WORKDIR /app  
  
COPY /target/springboot-docker-0.0.1-SNAPSHOT.jar /app/application.jar  
  
ENTRYPOINT["java","-jar","application.jar"]

1. **Cream imaginea**

* Deschidem consola la docker si punem proiectul undeva ca sa il putem accesa cu cd
* De ex, in Ubuntu/mnt si dam cd ../ pana ajungem acolo
* Acum in consola la docker dam cd NumeFolderApp
* Folosim

docker build -t NumeAlesLaImage AdresaLaFisierulDockerfile

De ex:

docker build -t springboot-docker-demo .

Acel punct inseamna ca nu punem nicio adresa la fisier, el deja este in directory actual

* Pentru a pune un tag,ce il vedem cand scrim docker images, mai punem la nume :
* docker build -t springboot-docker-demo:RELEASE-1.0.0 .
* Acum, imaginea a fost creata si pusa in docker. O putem vedea cu

docker images

* Putem folosi si **docker build -t username/repository\_name**

1. **Rulam imaginea**

* docker run -p 8080:8080 springboot-docker-demo
* -p e pentru a expune porturile
* De ex, -p 5231:8561

8561 – asta e port din container

5231 – asta e port ce va fi afisat in OS la noi. Deci, daca in container avem 8561, in browser la noi vom folosi 5231. Chiar daca vrem sa lasam port 8561, scriem -p 8561:8561, caci portul asa va fi expus, anume asta si face -p. Container are un private network si cu -p accesam acel port din network, adica dam acces la host OS la el

* **-d – detached**. Daca folosim run si -d, container va rula in detached mode, adica nu se va deschide direct consola lui.
* **docker logs -f id** – arata logurile din consola la container

**Push image**

* folosim docker login pentru a ne conecta cu contul
* apoi trebuie sa dam tag la o imagine. docker tag creaza o referinta la acea imagine si gata, si referinta va avea asa nume username/reponame. Dupa username/ pune ce nume vrem sa aiba repository pentru imagine

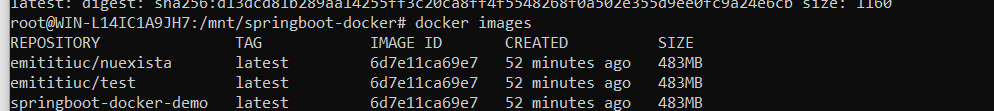
docker tag local-image:tag-name username/reponame:tagname

docker tag springboot-docker-demo emititiuc/test



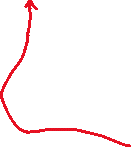
**Atentie sa punem mereu username!!!**

Si daca scriem docker images vedem:





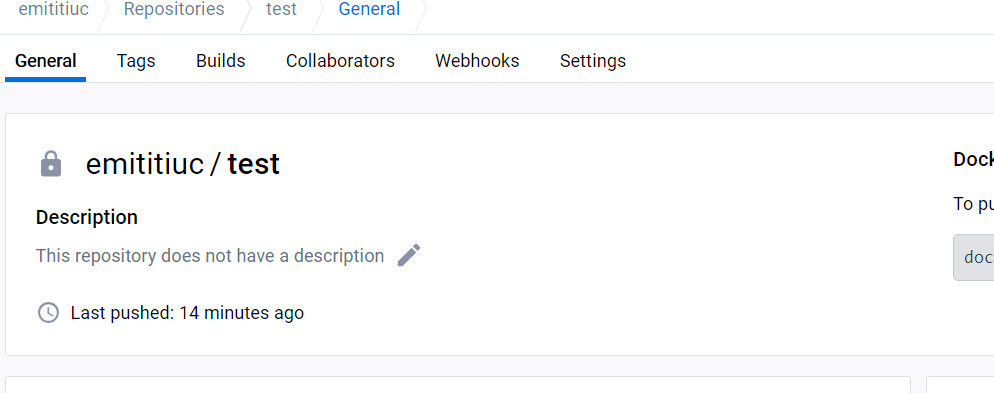
Vedem ca s-a mai creat o imagine cu numele emititiuc/test, dar asta nu e o imagine noua, ci doar o referinta la springboot-docker-demo, asta o vedem si dupa Image ID



* Acum i dam push:

docker push <username>/<reponame>:<tagname>

docker push emititiuc/test



**Pull image**

* Preluam numele complet al imagii, adica username/numeRepo

**Tag**

* Tag este pus la final de nume, gen mysql:**latest**
* Putem vedea lista de tags chiar in remote repo
* Daca nu punem niciunul, :latest e automat pus
* Acum, daca de ex cream o imagine,si apoi facem o noua versiune, putem sa folosim tags pentru a o pune tot in acelasi repo, doar ca cu alt tag. De ex:

docker tag springboot-docker-demo emititiuc/test

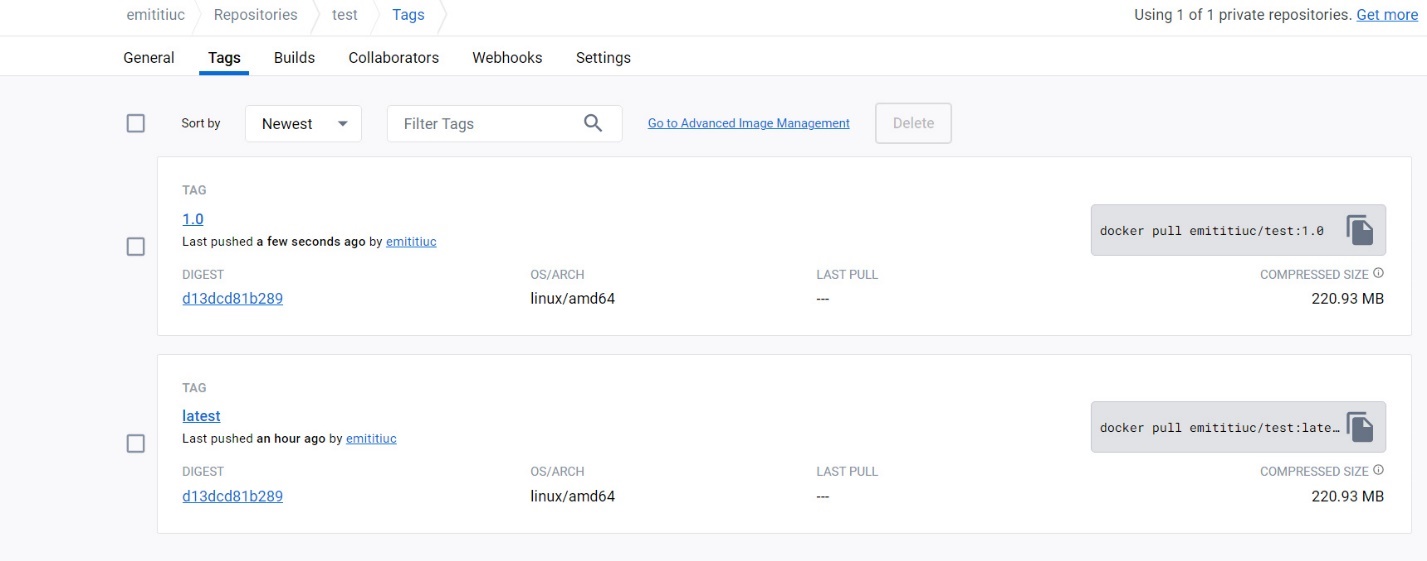
docker pull emititiuc/test

Apoi, daca folosim

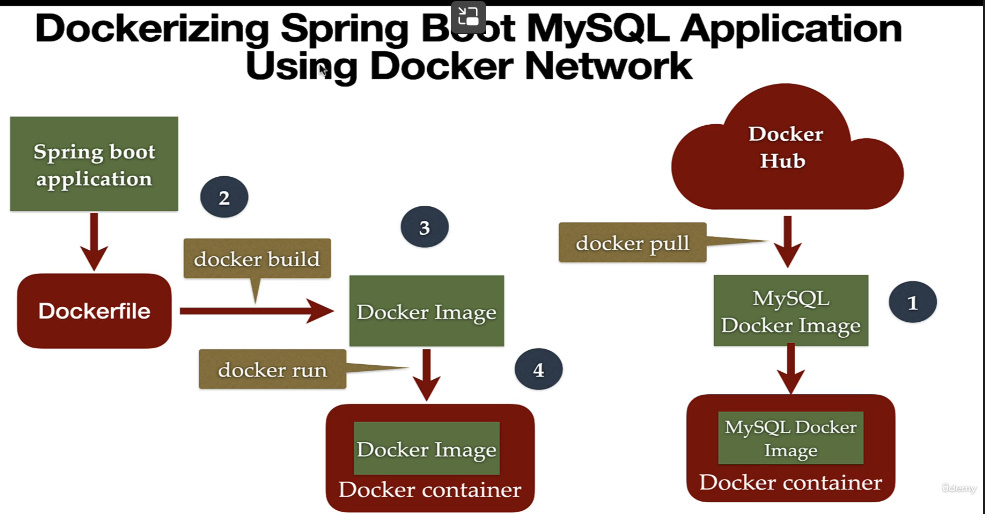
docker tag springboot-docker-demo emititiuc/test:1.0.0

docker push emititiuc/test:1.0.0

imaginea va fi pusa tot in test repository, doar ca va avea tag 1.0.0



**Docker network**

* Daca vrem ca containers sa lucreze unele cu altele, trebuie sa cream un network ce sa le ofere acces sa comunice intre ele
* 
* Fiecare container ruleaza intr-un network separat, dar le putem face un network comun
* Comanda:

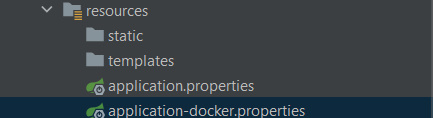
**docker network create NumeNetwork**

* **docker network ls** – arata toate networks
* Acum, cand vom folosi docker run sau docker create, vom mai aadauga si **--network NumeNetwork**
* pentru a accesa consola la un container:

**docker exec -it IdConainter bash**

**Use profiles**

* Cream profilul prin a crea alta application.properties, asa:

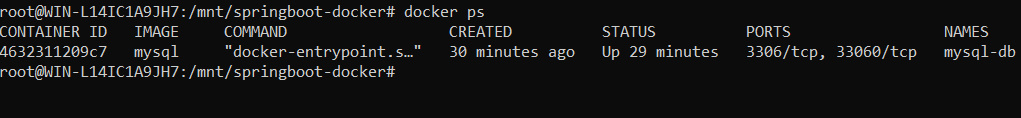


avem profilul docker

**docker run --name mysql-db -e MYSQL\_ROOT\_PASSWORD=test -e MYSQL\_DATABASE=test --network myapp mysql**

MYSQL\_DATABASE – numele la database default creata

* Acum, atentie, url la baza de date nu va mai avea ca host localhost, ci numele la container, gen:





spring.datasource.url=jdbc:mysql://mysql-db:3306/test  
spring.datasource.username=root  
spring.datasource.password=test  
  
management.info.env.enabled=true  
management.endpoints.web.exposure.include=\*  
  
management.endpoint.health.show-details=*always*

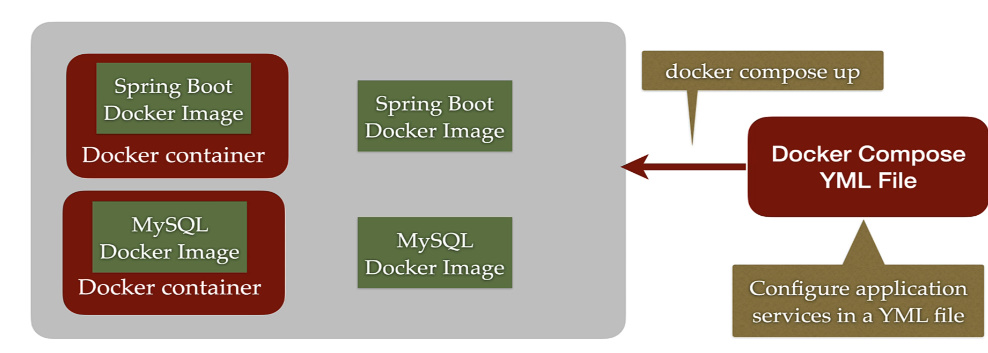


* In **application.properties** folosim asta ca sa activam docker profile:

spring.profiles.active=docker

**Docker compose**

* **Docker compose** – tool pentru a da deploy(start) la mai multe containere



* Daca avem o aplicatie care depinde de mai multe containere, gen pentru MySQL, Kafka, etc. nu e deloc comod mereu sa cream cate un container separat, apoi sa le unim prin network. Scriem o gramada de comenzi ce se tot repeta
* In loc sa tot repetam aceste comenzi, putem sa le rulam toate impreuna intr-un singur fisier. Deci, cu un singur fisier cream si rulam mai multe containere
* Fisierul pentru a crea docker compose este **YML**

**Docker compose yml file**

1. Cream fisierul **docker-compose.yml** in project

El incepe cu “version”, desi nu e obligatoriu

version: "3.8"

Apoi, in services: definim containerele

punem:

- numele la service

- container\_name: NumeContainer

- image: NumeImagine

- environment: pentru a specifica toate variabilele de sistem

- networks pentru a crea si defini un network

Acum, trebuie sa cream un image pentru proiectul nostru, care inca nu are imagine. Pentru asta folosim Dockerfile pentru project si

- build:

context: folder sau punem ./ daca e aici

dockerfile: nume la fisier

- ports:

- “8080:8080”

- restarrt: on-failure – daca nu se poate conecta la mysql,container de care depinde, aplicatia da restart. E important caci mysql container va lau ceva timp sa se porneasca si aplicatia poate da eroare ca nu se poate conecta la db, si va reincerca iar si iar

- depends\_on – container de care depinde

version: "3.8"  
  
services:  
 mysqldb:  
 container\_name: mysqldb  
 image: mysql  
 environment:  
 MYSQL\_ROOT\_PASSWORD: test  
 MYSQL\_DATABASE: test  
 networks:  
 test-network:  
 my-app:  
 container\_name: my-app  
 build:  
 context: ./  
 dockerfile: Dockerfile  
 ports:  
 - "8080:8080"  
 depends\_on:  
 - mysqldb  
 networks:  
 test-network:  
 restart: on-failure  
  
networks:  
 test-network:

2. Rulam acest fisier asa:

cd project

docker-compose up --build

Vedem ci docker ps -a ca s-au creat 2 containers

1. Alte comenzi:

docker-compose down – opreste toate containerele din background