# SPRC
# Laboratorul #4 Docker și Microservicii

Versiunea 9

Responsabil: **Dorinel FILIP**

## Cuprins

# Obiectivele laboratorului

În urma parcurgerii laboratorului, studenții:

- își vor însuși conceptul de microserviciu și vor fi capabili să descrie avantajele și provocările acestei abordări de dezvoltare software;
- vor cunoaște și vor putea utiliza principalele funcții ale tehnologiei Docker pentru a rula o aplicație formată din microservicii.

# Microservicii

În aplicațiile moderne, componentele software utilizate în rețea sunt adesea decuplate și reprezentate sub forma de componente independente, mici, responsabile de un set foarte restrâns de responsabilități, care se numesc **microservicii**.

Un avantaj al acestei abordări constă în faptul că fiecare microserviciu este suficient de puțin complex astfel încât să poată fi pus în responsabilitatea unei echipe mici de oameni (uneori chiar a unei singure persoane), dar acest lucru face ca, de multe ori, dezvoltarea fiecăruia dintre microservicii să se facă folosind limbaje de programare și tehnologii diferite, lucru care, la prima vedere, face dificilă administrarea aplicațiilor compuse din multe servicii diverse.

Fiecare microserviciu având propriile cerințe privind configurarea mediului software în care rulează (care depinde de tehnologia și limbajul de programare folosite), o primă abordare pentru cuprinderea acestei complexități este crearea de mașini virtuale configurate potrivit pentru fiecare dintre microservicii.

Totuși, dat fiind faptul că microserviciile sunt adesea componente cu consum mic de resurse, abordarea de a folosi mașini virtuale are dezavantajul de a nu fi foarte eficientă în utilizarea resurselor:

- Ce facem dacă avem 5 microservicii mici, dar scrise folosind 5 tehnologii dificil de configurat împreună? Rulăm sistemul de operare în 5 mașini viruale, doar pentru a scăpa de asta?
- Cel mai probabil, NU, deoarece există tehnologii mai potrivite. **Docker** este una dintre ele.

# Docker

### Ce este docker?

Docker este o platformă de **containere** software, folosită pentru a împacheta și rula aplicații atât local, cât și pe sisteme Cloud, eliminând probleme de genul *„pe calculatorul meu funcționează”*.

### Ce este o imagine?

Containerele Docker au la bază **imagini**, care sunt pachete executabile lightweight de sine stătătoare ce conțin tot ce este necesar pentru rularea unor aplicații software, incluzând cod, runtime, biblioteci, variabile de mediu și fișiere de configurare.

### Ce sunt containere?

Un **container** reprezintă o **instanță a unei imagini**, adică ceea ce imaginea devine atunci când este executată. El rulează izolat de mediul gazdă, accesând fișiere și porturi ale acestuia doar dacă este configurat să facă acest lucru. Containerele ruleaza aplicații nativ pe kernel-ul mașinii gazdă, având performanțe mai bune decât mașinile virtuale, care au acces la resursele gazdei prin intermediul unui hipervizor. Containerele

au acces nativ, fiecare rulând într-un proces discret, necesitând tot atât de multă memorie cât orice alt executabil.

### Instalarea Docker-CE

Docker este disponibil în două variante: Community Edition (CE) și Enterprise Edition (EE). Docker CE este util pentru dezvoltatori și echipe mici care vor să construiască aplicații bazate pe containere. Pe de altă parte, Docker EE a fost creat pentru dezvoltare enterprise și echipe IT care scriu și ruleaza aplicații critice de business pe scară largă. Versiunea Docker CE este gratuită, pe când EE este disponibilă cu subscripție. În cadrul laboratorului de SPRC vom folosi Docker Community Edition.

Pentru a instala rapid Docker-CE pe un sistem cu Ubuntu cea mai utilă resursă este Tutorialul Oficial.

Pentru început, trebuie să ne asigurăm că avem instalate câteva pachete ajutătoare:

```
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common
```

Adaugăm repository-ul oficial:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

Actualizăm lista de pachete și instalăm Docker-CE:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Adăugăm userul curent în grupul docker pentru a putea folosi Docker fără sudo:

```
sudo usermod -aG docker $USER
```

### Testarea Docker

Pentru a testa docker putem folosi următoarea comandă:

```
sudo docker run hello-world
```

Rezultatul ar trebui să fie de forma:

```
root@potato:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
```

```
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

În spate, s-au întâmplat următoarele lucruri:

1. S-a descărcat imaginea hello-world de pe repository:
2. S-a creat un container cu imaginea descărcată;
3. S-a executat comanda default a containerului;
4. Comanda a afisat continutul si s-a închis;
5. Containerul a trecut în starea oprit.

O sursă importantă de imagini Docker este repository-ul https://hub.docker.com. Aici găsim imagini pentru a utiliza foarte multe componente OSS si nu numai.

### Crearea de imagini

Crearea unei imagini de Docker se face pe baza unui fișier Dockerfile, care descrie pașii ce trebuiesc executați pentru obține acea imagine.

Mai jos aveți un exemplu de Dockerfile, pentru o aplicație web scrisă în Python:

```
FROM python:3.6
COPY requirements.txt /tmp
RUN pip install -r /tmp/requirements.txt
COPY /src /app
WORKDIR /app
EXPOSE 80
CMD ["python", "main.py"]
```

Arborescența de fișiere este una simplă:

```
root@potato:~/sprc/image-example# tree
.
|── Dockerfile
|── requirements.txt
+── src
    +── main.py

1 directory, 3 files
```

Pentru a crea, plecând de la aceasta o imagine care se numește **numeImagine**, vom executa următoarea comandă:

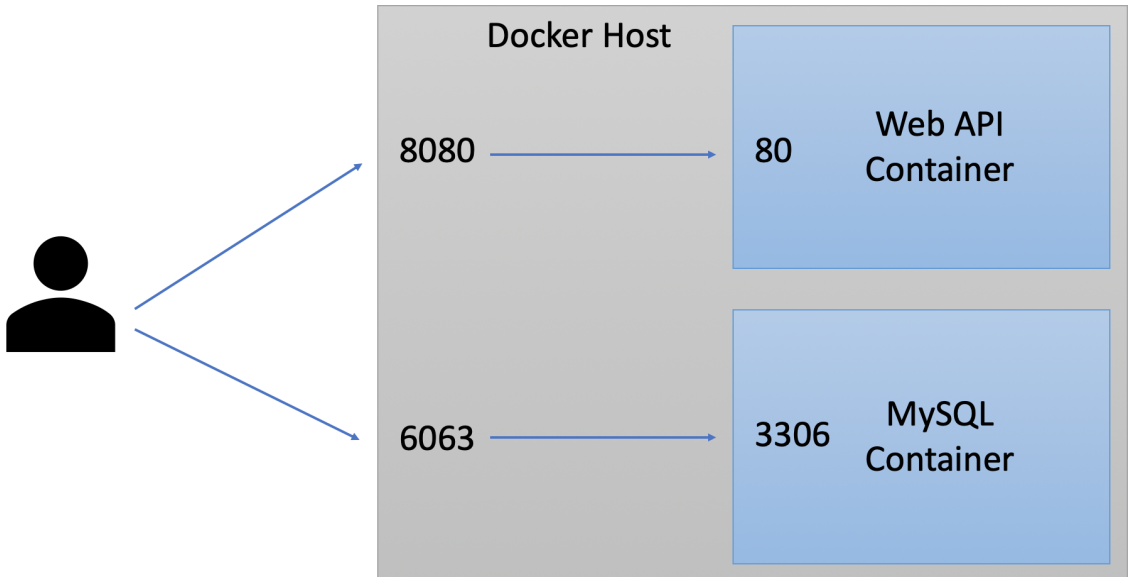
```
docker build . -t numeImagine
```

Mai jos detaliem comenzile din Dockerfile:

- Folosim `FROM` pentru a indica imaginea de la care pornim. În cazul de față, pornim de la imaginea oficială de Python:3.6;
- Comanda `COPY` face copierea de fișiere / foldere de pe mașina gazdă în cadrul imaginii;
- Comanda `RUN` face rularea unei comenzii shell **în cadrul imaginii**;
- Comanda `WORKDIR` schimbă directorul de lucru al containerului;
- Comanda `EXPOSE` indică porturile prin care un container cu această imagine vrea să comunice cu exteriorul;
- Comanda `CMD` specifică cu ce comandă va fi creat procesul containerului.

### Expunerea de porturi

Orice container de docker este conectat la rețeaua internă a sistemului gazdă pe care ruleaza Docker. Pentru ca traficul extern să ajungă în interiorul containerului, este necesar ca **portul** rețelei interne să fie mapat la **portul** de intrare în container.



Acest lucru se poate realiza utilizand flag-ul `-p` sau `--port` și o asociere între **PORTUL EXTERN** și **PORTUL INTERN**.

Exemplu de comandă cu asociere de port:

```
docker run -it -p 8080:80 web_api
docker run -it -p 6603:3066 mysql_image
```

De multe ori, portul extern coincide cu portul intern. Însă, sunt situații în care un port este ocupat pe gazdă și, din acest motiv, se dorește accesarea containerului prin alt port.

### Ce sunt volumele?

Un **container** de Docker își menține starea doar pe parcursul rulării sale. Când container-ul se închide, orice dată aflată in container este ștearsă. Câteodată însă, ne dorim să menținem informația, chiar daca un container se închide (din motive mai mult sau mai puțin controlate de noi).

Să considerăm următorul exemplu:

Avem un container de docker în care rulează *MySQL*. Atunci când containerul este repornit, orice informație din baza de date este ștearsă. Totuși, ne dorim ca baza de date să fie persistentă, adică informația să fie salvată chiar dacă containerul a fost închis și deschis din nou.

Soluția propusă de Docker este **volumul**.

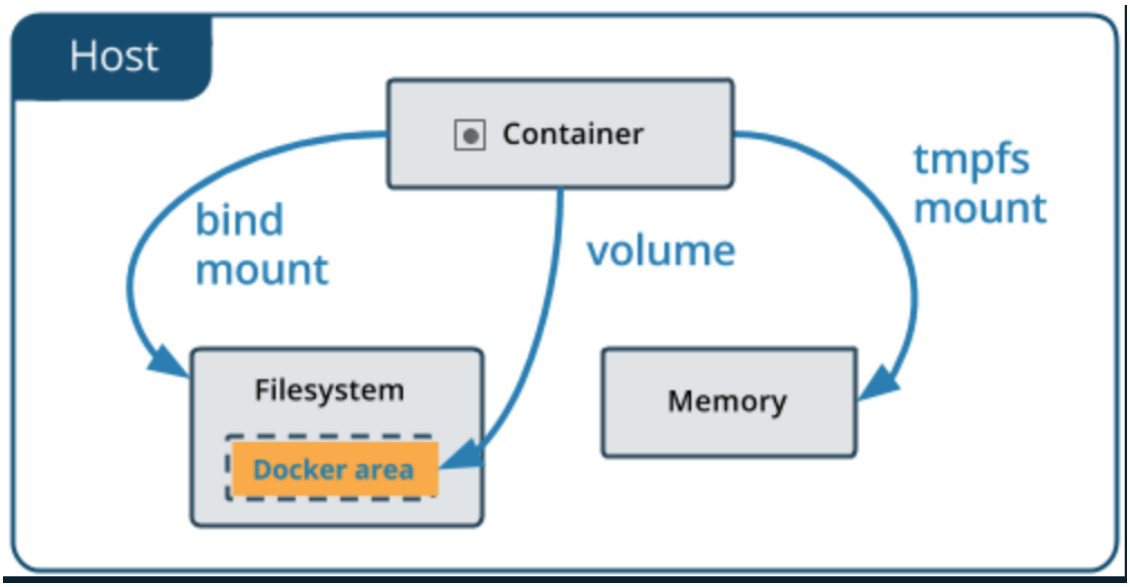



Figura 1: Volume.
Sursa imaginii: `https://docs.docker.com/storage/volumes/`

Volumele, așadar, sunt puncte de acces în sistemul de fișiere al gazdei, care leagă un folder din container cu un folder din gazdă, prin care containerul de docker poate salva informații.

Acestea pot fi definite în două moduri, folosind `-v` sau `--volume` sau folosind `--mount`. În ambele cazuri, trebuie legat un folder din sistemul gazdă de un folder din container.

În exemplul nostru cu *MySQL*, ne dorim să creăm un volum care să conecteze folderul în care este reținută

baza de date cu un folder din sistemul gazdă.

### Ce sunt variabilele de mediu?

Orice proces are variabile de mediu care reține informații sub formă `CHEIE:VALOARE`. Acestea sunt folosite de către programe și pot fi setate înainte sau în timpul lansării in execuție a programelor.

Deoarece un container care rulează este un proces lansat in execuție, și acesta poate folosi variabile de mediu. Variabilele de mediu setate containerului vor fi folosite, apoi, de programul (sau programele) care rulează în interiorul său.

Să reluăm exemplul cu *MySQL*. Orice bază de date trebuie securizată printr-o parolă. Totuși, nu e util ca parola să fie hardcodată în interiorul *Dockerfileului* aferent imaginii de MySQL, deoarce orice om care vrea sa ruleze un container bazat pe imaingea de MySQL cu altă parolă, va trebui sa modifice, în *Dockerfile*, parola.

Variabilele de mediu sunt utile, deoarece o parolă poate fi trimisă containerului în momentul în care acesta este rulat.

### Exemplu de comandă

La link-ul următor puteți găsi un tutorial pentru rularea unui container de *MySQL*: `https://hub.docker.com/_/mysql`. În cadrul lui, veți putea observa următoarea comandă:

```
docker run --name some-mysql -v /my/own/datadir:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=
    my-secret-pw -d mysql:tag
```

În comanda precedentă, regăsim atât o creare de **volum**, cât și o injectare de **variabilă de mediu**.

Volumul este definit de următoarele:

1. flagul `-v`
2. numele folderului din sistemul gazdă unde vor fi stocate informații
3. semnul de legatură `:`
4. numele folderului din container de unde vor fi preluate informații

Variabila de mediu este injectată astfel:

1. flagul `-e`
2. numele variabilei de mediu din interiorul containerului
3. semnul de legatură =
4. valoarea variabilei de mediu

## Docker-compose

Docker-compose este un utilitar care permite gruparea mai multor microservicii sub forma unei soluții unificate, sub forma unui fișier YML.

Cu ajutorul lui, putem face pornirea mai multor microservicii care constituie o soluție.

Mai multe informații despre cum se alcătuiește un Docker-compose puteți găsi la următoarea adresă: `https://docs.docker.com/compose/gettingstarted/`.

### Instalarea Docker-compose

Dacă avem Docker instalat, configurarea Docker-compose este trivială:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Observație: În versiunile mai noi ale Docker, Docker-compose este un plugin, care poate fi folosit direct din executabilul docker (ex. `docker compose up`) fără a mai fi necesar să aveți un executabil separat.

### Utilizarea Docker-compose

Presupunând că în folderul curent există un fișier site.yml, putem folosi următoarele comenzi:

```
docker-compose -f site.yml up         # porneste stack-ul de servicii
docker-compose -f site.yml up -d        # porneste stack-ul de servicii in background
docker-compose -f site.yml down         # opreste toate containerele din stack-ul de servicii
docker-compose -f site.yml down --volumes       # opreste toate containerele din stack-ul de servicii si sterge continutul volumelor prezente.
```

## Sarcini de lucru

**Task-ul 0**: Folosind tutorialul, instalați Docker pe calculatorul pe care lucrați.

**Task-ul 1**: Plecând de la această arhivă, care reprezintă exemplul prezentat la build-ul de imagini:

- Faceți build unei imagini cu numele **task1**;
- Porniți un container cu acest microserviciu și asigurați-vă că serviciul este expus prin portul 8080 al mașinii gazdă;
- Testați în browser că serviciul este disponibil la `http://localhost:8080`;
- Opriți și ștergeți containerul creat.

**Task-ul 2**: Plecând de la următorul Docker-compose:

```
version: '3.3'

services:
   mysql:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - mysql
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: mysql:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
    wp_data: {}
```

Identarea este importantă la redactarea fișierelor .YML. Acesta poate fi descărcat de la adresa: `https://static.dfilip.xyz/sprc_lab_docker.yml`.

- Modificați docker-compose astfel încât site-ul să fie vizibil pe portul `8081`.
- Porniți soluția folosind docker-compose;
- Accesați `localhost:8081` și configurați site-ul;
- Opriti stack-ul folosind comanda `docker-compose down`, reporniți și arătați că site-ul a rămas configurat.
- În fișierul docker-compose oferit se setează deja persistența datelor pentru MySQL (motiv pentru care site-ul a rămas configurat). Faceți modificările necesare astfel încât să se asigure și persistența datelor salvate în WordPress (**hint:** fișierele Wordpress sunt stocate la `/var/www/html` în cadrul containerului);
  - Pentru a verifica persistența, creați o postare care conține și o imagine (fișier .jpg sau .png încărcat din calculatorul vostru), opriți și reporniți serviciile și verificați că imaginea se află încă pe site.

Pentru a primi notă, va trebui să verificați cu asistentul cel puțin funcționalitatea task-ului 2.

Dacă faceți greșeli pe care nu reușiți să le reparați, având în vedere că nu există date importante pe site, puteți șterge tot și să reveniți la starea inițială folosind o comandă de tipul `docker-compose down --volumes`.

## Comenzi utile

```
$ docker image pull <IMAGE>                 # descarca o imagine dintr-un registru
$ docker container run <NAME> [COMMAND]     # ruleaza un container
$ docker image ls                           # arata o lista cu imaginile prezente local
$ docker container run -it <NAME>           # ruleaza un container in mod interactiv
$ docker container run -d <NAME>            # ruleaza un container in background (ca
    daemon)
$ docker container ls -a                    # arata o lista cu toate containerele
$ docker attach <ID>                        # se ataseaza la un container
$ docker stop <ID>                          # opreste un container
$ docker rm <ID>                            # sterge un container
$ docker build -t <TAG> .                   # construieste o imagine
$ docker images                             # afiseaza o lista a imaginilor locale
$ docker image inspect <TAG>                # afiseaza informatii despre o imagine
```