

Hálózati rendszerek és szolgáltatások fejlesztése

Lucienne Kachichian, Mikecz Kálmán, Németh Márton

2018. április 13.

1. Docker használata

1.1. Docker telepítése

`pacman -S docker`

1.2. Docker image

A fájlrendszer és a paraméterek összessége. Rétegekből épül fel, ezek az image létrehozása után csak olvashatóak.

1.3. Docker container

Amikor az image-et containerre alakítjuk (pl. `docker run`), akkor a docker egy írható/olvasható réteget helyez a csak olvasható rétegek felé.

1.4. Dockerfile

A dockerfile egy image leírását tartalmazza. Ez alapján a docker el tud készíteni egy image-et.

2. Container vs VM

IDE MÉG ÍRNI KÉNE

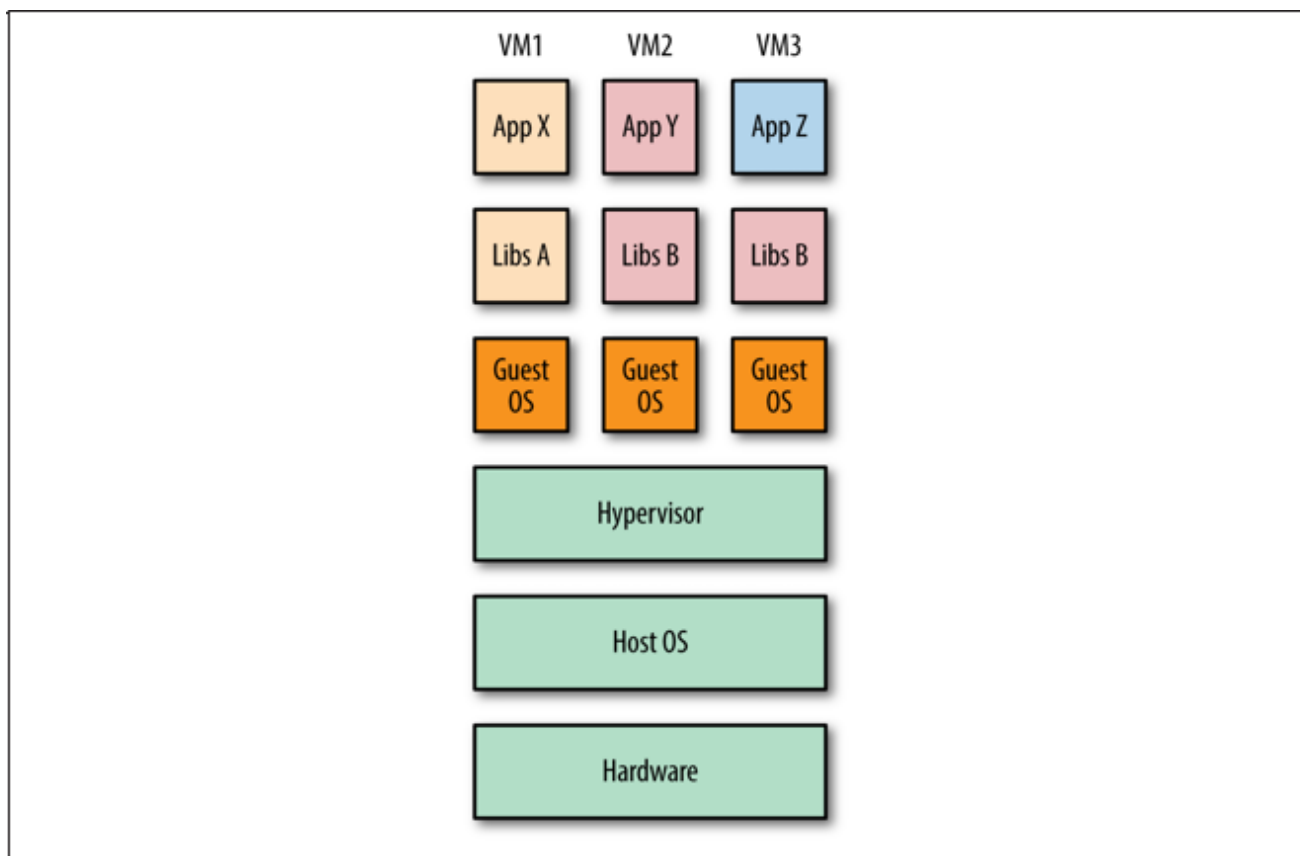


Figure 1-1. Three VMs running on a single host

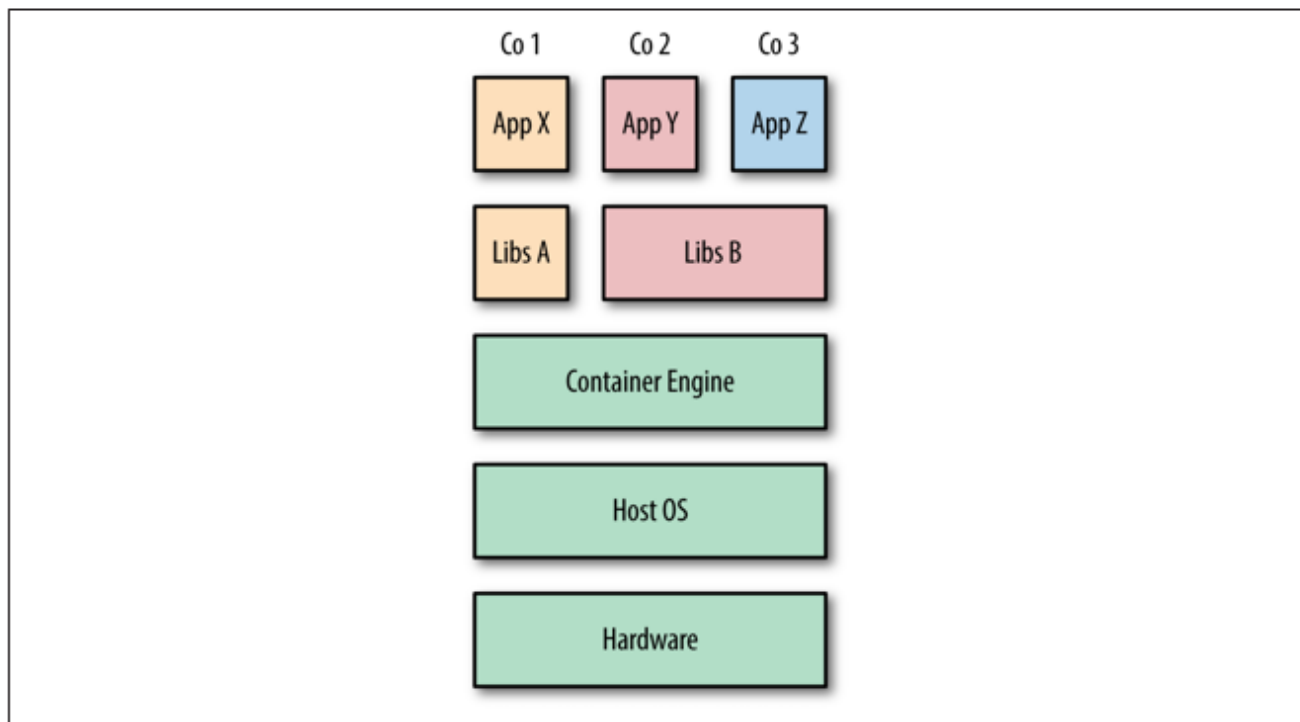


Figure 1-2. Three containers running on a single host

3. Példák docker futtatására

3.1. Hello World

```
docker run debian echo "Hello World"
```

Ez a parancs futtatja a debian image-et, azon belül elindítja az echo programot, ami kiírja, hogy "Hello World".

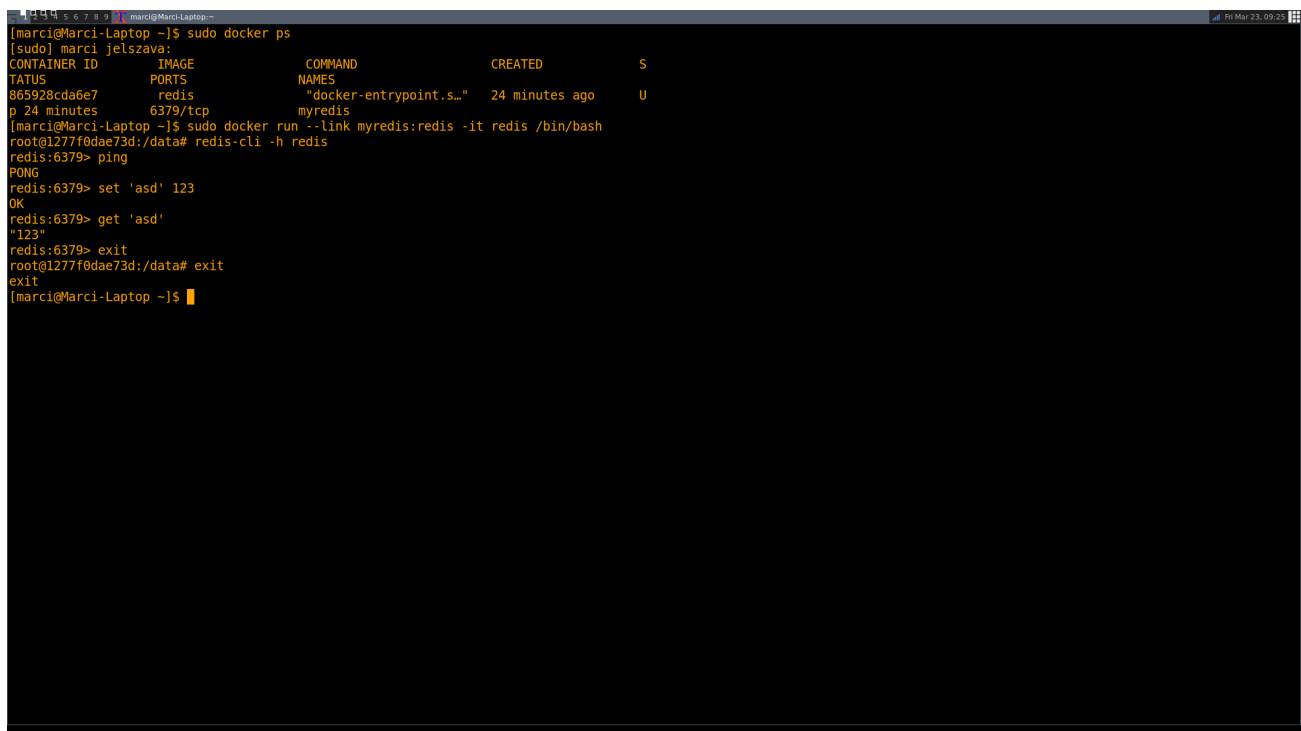
3.2. Két docker container kommunikációja

Az első parancs elindítja a redis containert a háttérben, és elnevezi "myredis"-nek:

```
docker run -name myredis -d redis
```

A következő parancs elindít még egy redis containert. A link kapcsoló segítségével összekapcsolhatjuk a két containert, így a második container "redis" néven látja az elsőt. Ezt úgy éri el, hogy a `/etc/hosts` fájlba beírja az első, háttérben futó container IP-címét. A containerben elindul egy bash shell, itt a megfelelő parancsokat kiadva tudunk kommunikálni a másik containerrel.

```
docker run -rm -it -link myredis:redis redis /bin/bash
```



```
[marci@Marci-Laptop ~]$ sudo docker ps
[sudo] marci jelszava:
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
865928cda6e7   redis     "docker-entrypoint.s..." 24 minutes ago Up
p 24 minutes   6379/tcp   myredis
[marci@Marci-Laptop ~]$ sudo docker run --link myredis:redis -it redis /bin/bash
root@1277f0dae73d:/data# redis-cli -h redis
redis:6379> ping
PONG
redis:6379> set 'asd' 123
OK
redis:6379> get 'asd'
"123"
redis:6379> exit
root@1277f0dae73d:/data# exit
exit
[marci@Marci-Laptop ~]$
```

2. ábra.

4. Egyszerű docker webalkalmazás

Az alábbi leírás alapján kipróbáltunk néhány egyszerű docker alkalmazást:

<https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md>

4.1. Statikus Nginx webalkalmazás

Először egy statikus weboldalt jelenítettünk meg egy dockerben futó Nginx webszerver segítségével. Először letöltöttük és futtattuk a megfelelő docker image-et:

```
docker run -name static-site -e AUTHOR="Your Name" -d -P dockersamples/static-site
```

A run parancs letöltötte a dockersamples/static-site image-et, mert lokálisan nem találta meg. A kapcsolók szerepe:

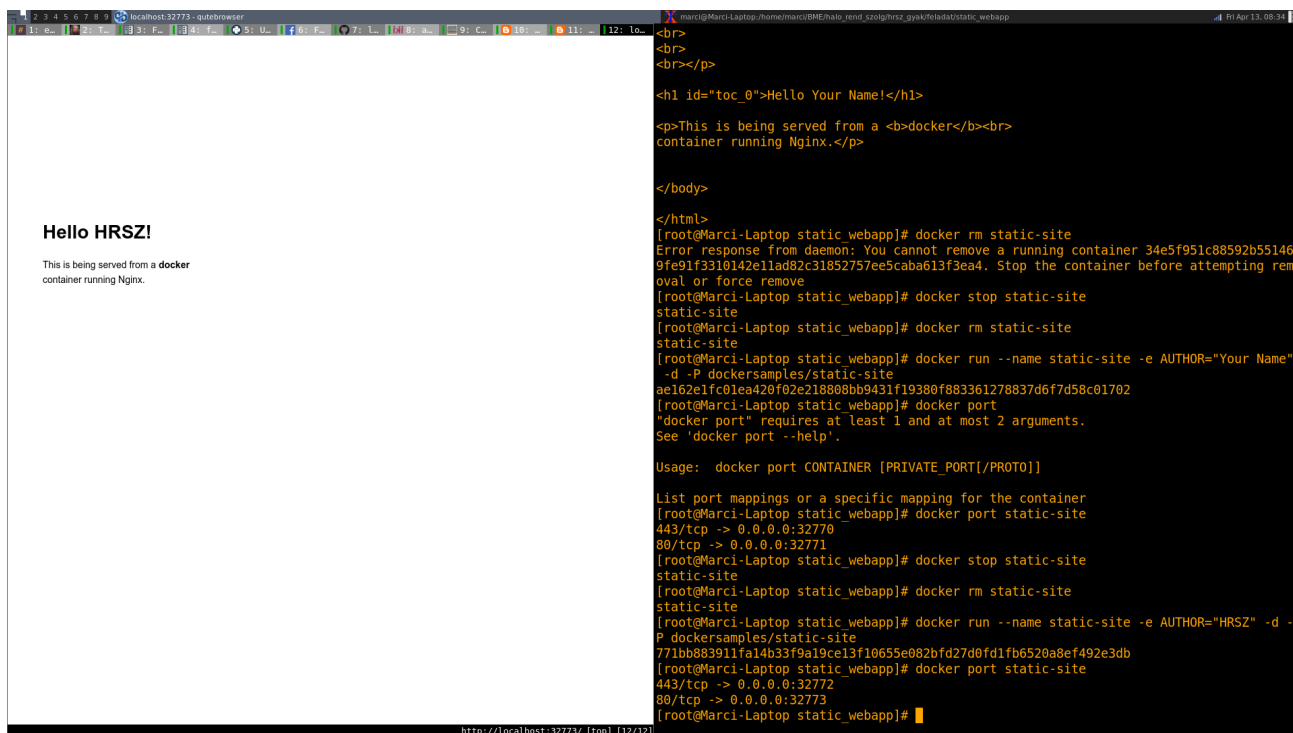
- **name:** megadja a container nevét, később így tudunk rá hivatkozni
- **e:** környezeti változókat állít be a containerber
- **d:** háttérben futtatja a containert
- **-P:** a container nyitott portjait a host egy-egy random portjára átirányítja

Ahhoz, hogy a host oldalról elérjük a weboldalt, tudnunk kell, hogy melyik porton érjük el a containert. Ezt az alábbi paranccsal kérdezhetjük le:

```
docker port static-site
```

Itt a `static-site` a container neve, amit korábban a `run` parancsnál megadtunk.

Ezután a webszervert elérhetjük a `localhost:port` címen, ahol a port helyére azt a portot kell írni, amit a `docker port` parancs megadott (a containeren belül a webszerver a 80-as porton fut). A webszerver működése a 3. ábrán látható.



3. ábra.

4.2. Python Flask webalkalmazás létrehozása Dockerfile segítségével

Az alábbi Dockerfile segítségével hoztuk létre a docker image-et:

```
FROM alpine:3.5
RUN apk add --update py-pip
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/
EXPOSE 5000
CMD ["python", "/usr/src/app/app.py"]
```

A Dockerfile sorainak magyarázata:

- **FROM alpine:3.5:** ezt a docker image-et használjuk kiindulásként. Az összes többi sor ehhez fog újabb rétegeket hozzáadni
- **COPY:** a hoston levő fájlt a docker image-be másolja a megadott helyre
- **RUN:** lefuttat egy parancsot a docker image build-elése közben
- **EXPOSE:** kinyitja a megadott portot
- **CMD:** megadja a container elindításakor lefuttatandó parancsot

A docker image által használt fájlok:

- **requirements.txt:** ebben a fájlban van benne annak a python modulnak a neve, amit a pip csomagkezelővel fel kell telepíteni.
- **app.py:** ez maga a webszerver

- `templates/index.html` ez egy html template fájl, amit a webserverver felhasznál

Az alábbi paranccsal létrehozhatjuk az image-et:

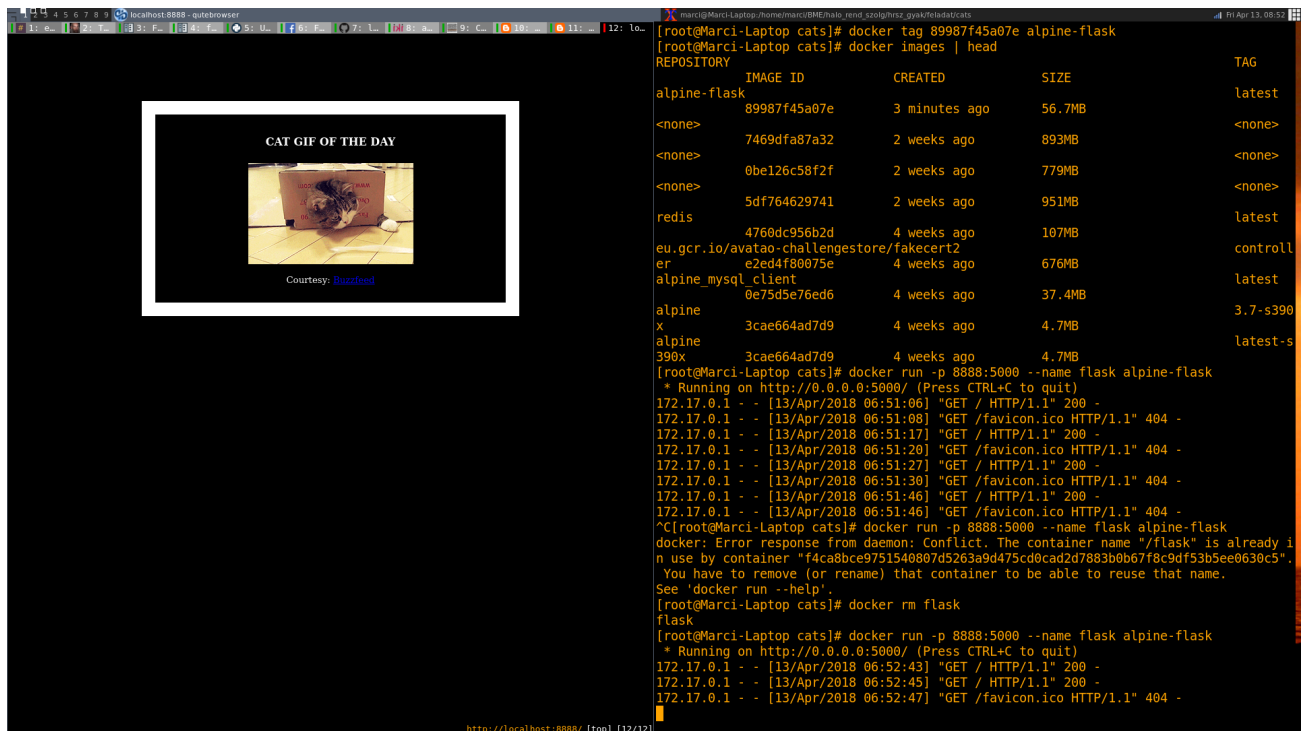
`docker build -t alpine-flask .`

A `t` kapcsoló az image tag-et adja meg, később ezzel tudunk hivatkozni rá. A végén levő pont a `Dockerfile` helyét adja meg (jelen esetben az éppen aktuális könyvtár).

Ezután ugyanúgy futtathatjuk a containert, mint az előző példában:

`docker run -p 8888:5000 -name myfirstapp alpine-flask`

A `p` kapcsoló segítségével konkrétan megadhatjuk, hogy a container melyik portja a host melyik portjára legyen átirányítva. Jelen esetben a container 5000-es portját a host 8888-as portján keresztül érhetjük el. A webserverver működése a 4. ábrán látható.



4. ábra.