

Hálózati rendszerek és szolgáltatások fejlesztése

Kachichian Lucienne, Mikecz Kálmán, Németh Márton

2018. május 10.

1. Dockerről általában

1.1. Mi az a Docker, mire jó?

A Docker jelenleg az egyik legelterjedtebb container framework, ami megkönnyíti az elosztott alkalmazások fejlesztését, szállítását (hordozhatóságát) és futtatását. A technológia már régóta létezik, ám ahhoz, hogy széleskörben elterjedhessen, a következő tulajdonságok együttes jelenlétére volt szükség: könnyű használhatóság, sebesség, modularitás és skálázhatóság.

A Dockerrel (más eszközökhöz képest) nagyon egyszerűen tudunk alkalmazásokat létrehozni, illetve menedzselni. Segítségével nem kell többet órákat eltöltenünk a különböző környezetek közötti alkalmazásátvitellel.

A gyorsaság abból fakad, hogy nem virtuális gépekről, hanem konténerekről van szó, amik tömören sokkal kisebbek és egyszerűbbek, felépítésüknek köszönhetően pedig kevesebb erőforrást használnak (erről később bővebben is lesz szó).

1.2. Hogyan működik?

Egy fájlrendszerbe csomagolja a szoftvert, ami mindent - kódot, futtatókörnyezetet, rendszereszközöket, rendszerkönyvtárakat - tartalmaz, ami a futtatáshoz szükséges. Ezek teszik lehetővé, hogy az alkalmazásunk minden környezetben azonos módon futhasson függetlenül attól, hogy ez az asztali gépünket jelenti, vagy egy nyilvános felhőt.

2. Docker használata

2.1. Docker telepítése

```
pacman -S docker
```

2.2. Docker image

A fájlrendszer és a paraméterek összessége. Rétegekből épül fel, ezek az image létrehozása után csak olvashatóak. Igény esetén több hivatalos és közösség által feltöltött ingyenes image letöltésére van lehetőség a Docker Hub-ról (<https://hub.docker.com/>).

2.3. Docker container

Amikor az image-et containerré alakítjuk (pl. `docker run`), akkor a docker egy írható/olvasható réteget helyez a csak olvasható rétegek felé. Fontos megemlíteni, hogy a container-ek nem másolatai az image-nek, tehát ha egy image-et 5-ször indítunk el, akkor az nem eredményez ötszörös helyfoglalást. A container-ek mérete csak a fájlok módosításakor, hozzáadásakor nő, törlés esetén pedig nem csökken, mivel az eredeti image file-ban ugyanúgy jelen vannak, csak a container-ben válnak láthatatlanná.

2.4. Dockerfile

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

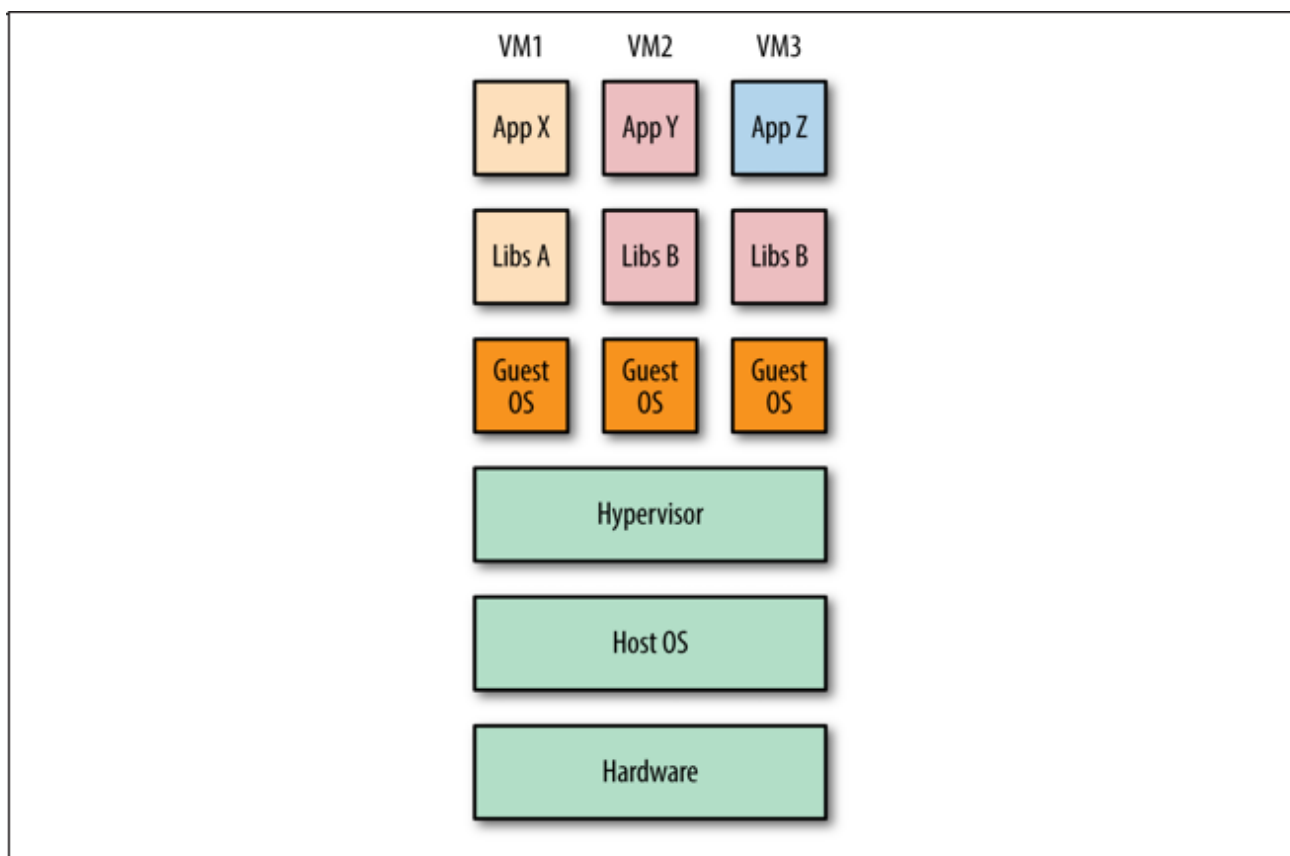


Figure 1-1. Three VMs running on a single host

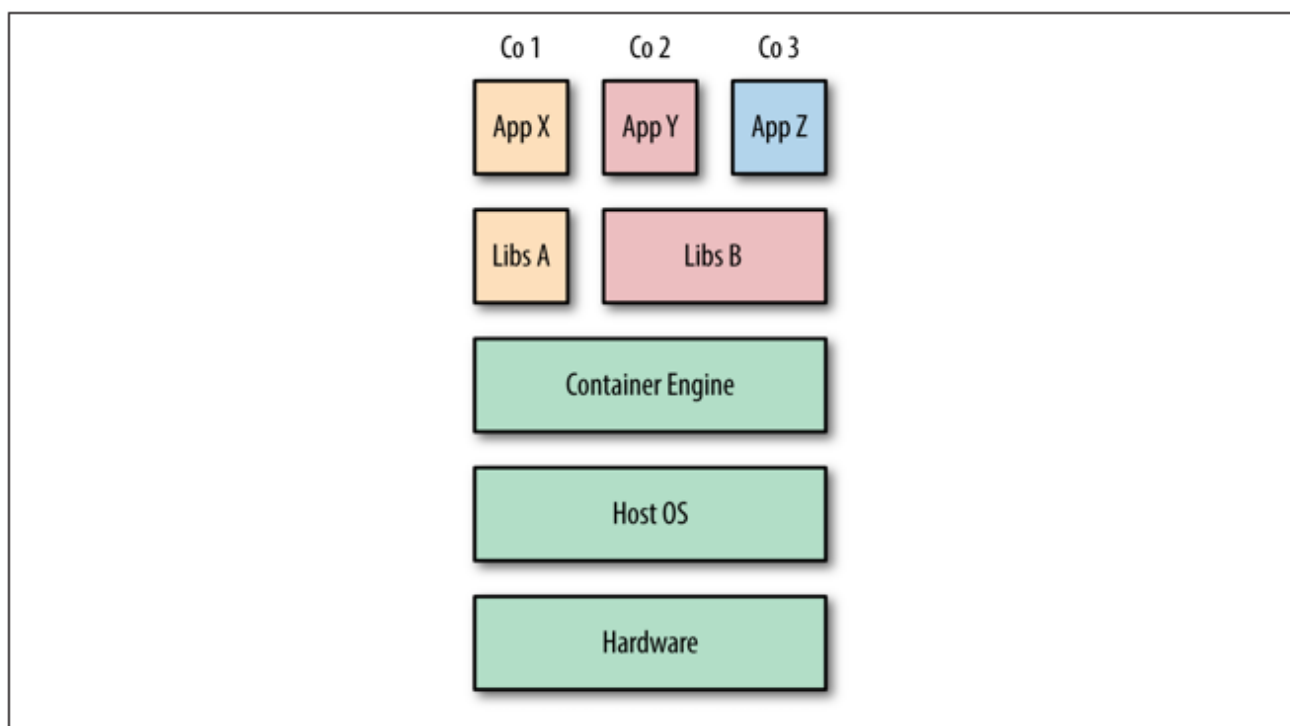


Figure 1-2. Three containers running on a single host

3. Container vs VM

A konténerek és a virtuális gépek is virtualizációs formák. Bár első benyomásra nagyon hasonlíthatnak egymáshoz, közelebbről szemügyre véve őket alapvető tulajdonságokban is különböznek.

Virtuális gép esetében hardver virtualizációról beszélünk, tehát ideális esetben az operációs rendszer nem is tudja, hogy igazából nem fizikai hardveren fut. Ehhez azonban szükség van a host operációs rendszer felett futó hipervizorra. Ezen már több vendég operációs rendszer is futtatható, egymástól teljesen elkülönítve. Mivel minden egyes virtuális gép egy különálló egység, ezért számottevő erőforrást igényel. A teljes elszigeteltségnek azonban vannak előnyei is, kisebb a biztonsági kockázat, illetve egy virtuális gép összeomlása nem vonja magával a többiét is.

Ezzel szemben a konténerek egy új ötleten alapulnak, miszerint nincs szükség a teljes virtualizációra, elég a konténeres virtualizáció, ami az operációs rendszer kernelében való folyamatelkülönítést jelenti. Ebből kifolyólag nincs szükség plusz erőforrásokra és a virtuális gépekhez képest sokkal kevesebb erőforrást igényelnek. Fontos azonban megemlíteni, hogy a gazda operációs rendszer összeomlásakor a konténerek veszélybe kerülhetnek, illetve a biztonsági kockázat is magasabb.

A virtuális gép és a container felépítése az 1. ábrán látható.

4. Példák docker futtatására

4.1. 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".

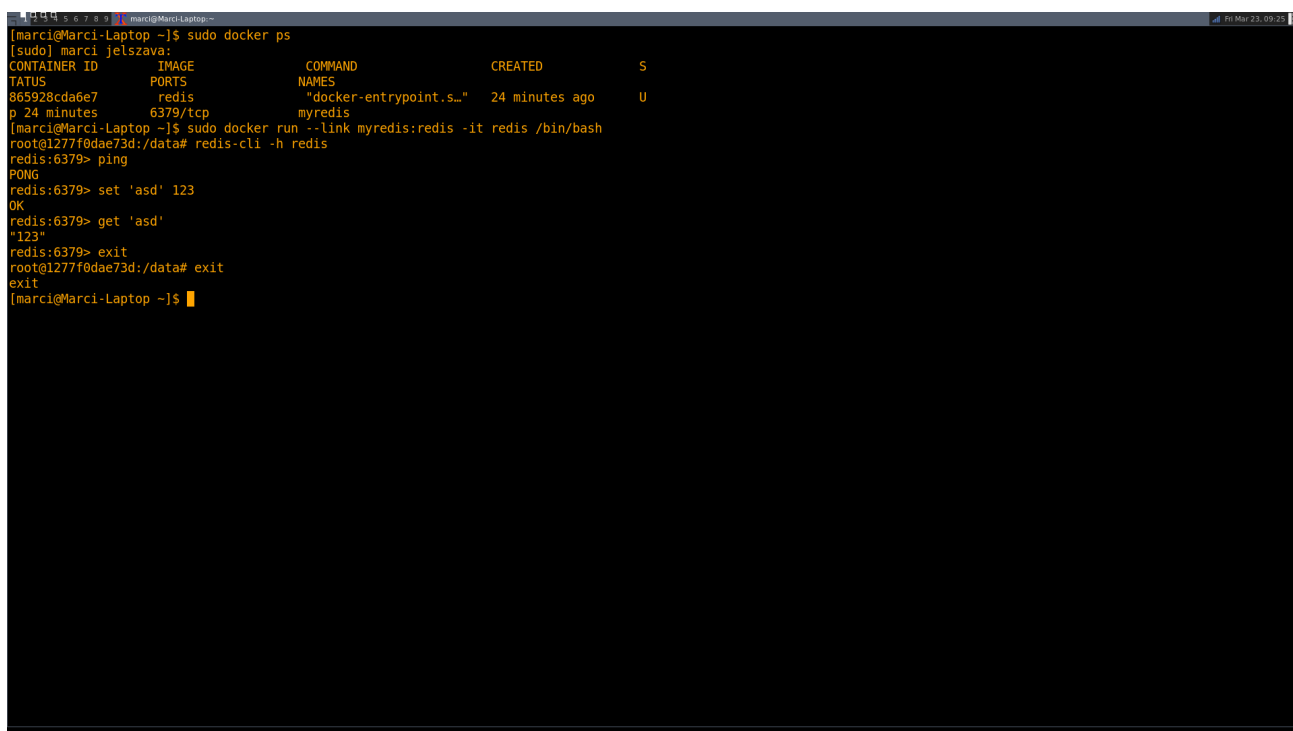
4.2. Két docker container kommunikációja

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 első. 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.

5. Egyszerű docker webalkalmazás

Az alábbi leírás alapján kipróbáltunk néhány egyszerű docker alkalmazást:

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

5.1. 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:

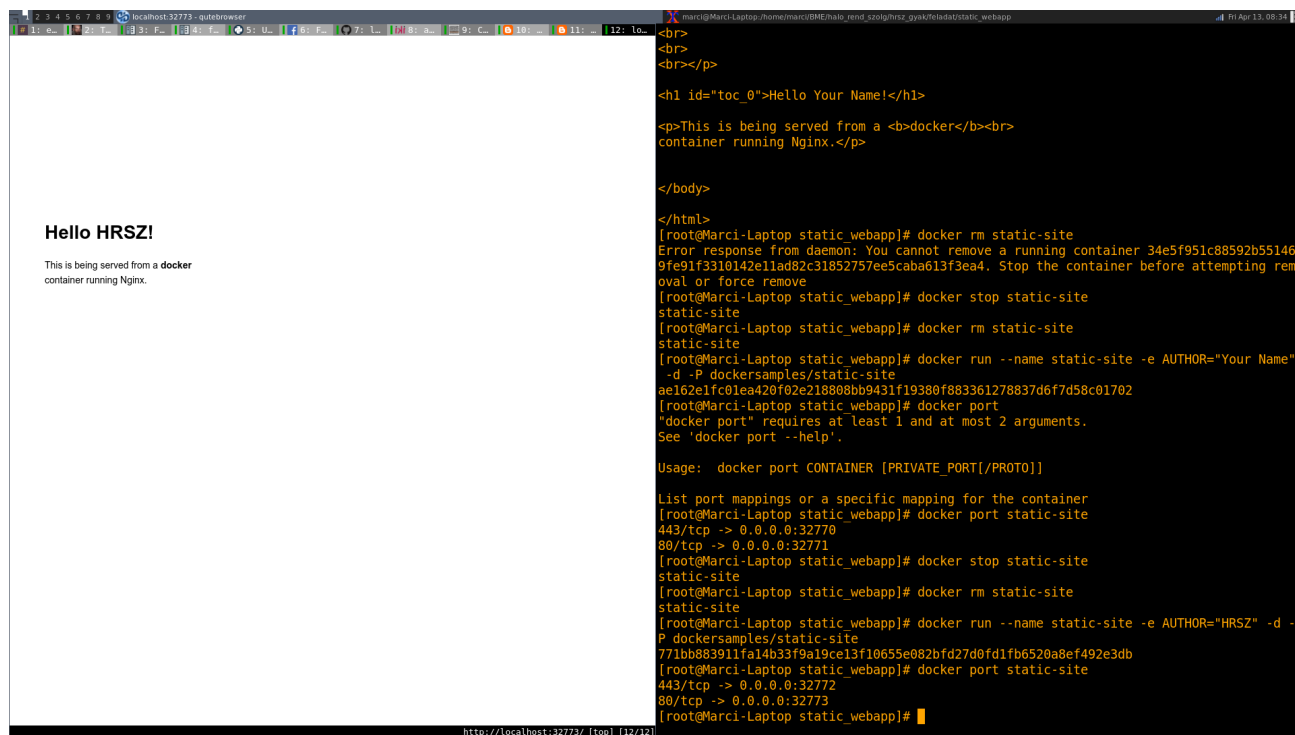
- **name:** megadja a container nevét, később így tudunk rá hivatkozni
- **e:** környezeti változókat állít be a containerber
- **d:** háttérben futtatja a containert
- **-P:** a container nyitott portjait a host egy-egy random portjára átirányítja

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 paranccsnál 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.

5.2. 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 webszerver felhasznál

Az alábbi paranccsal létrehozhatjuk az image-et:

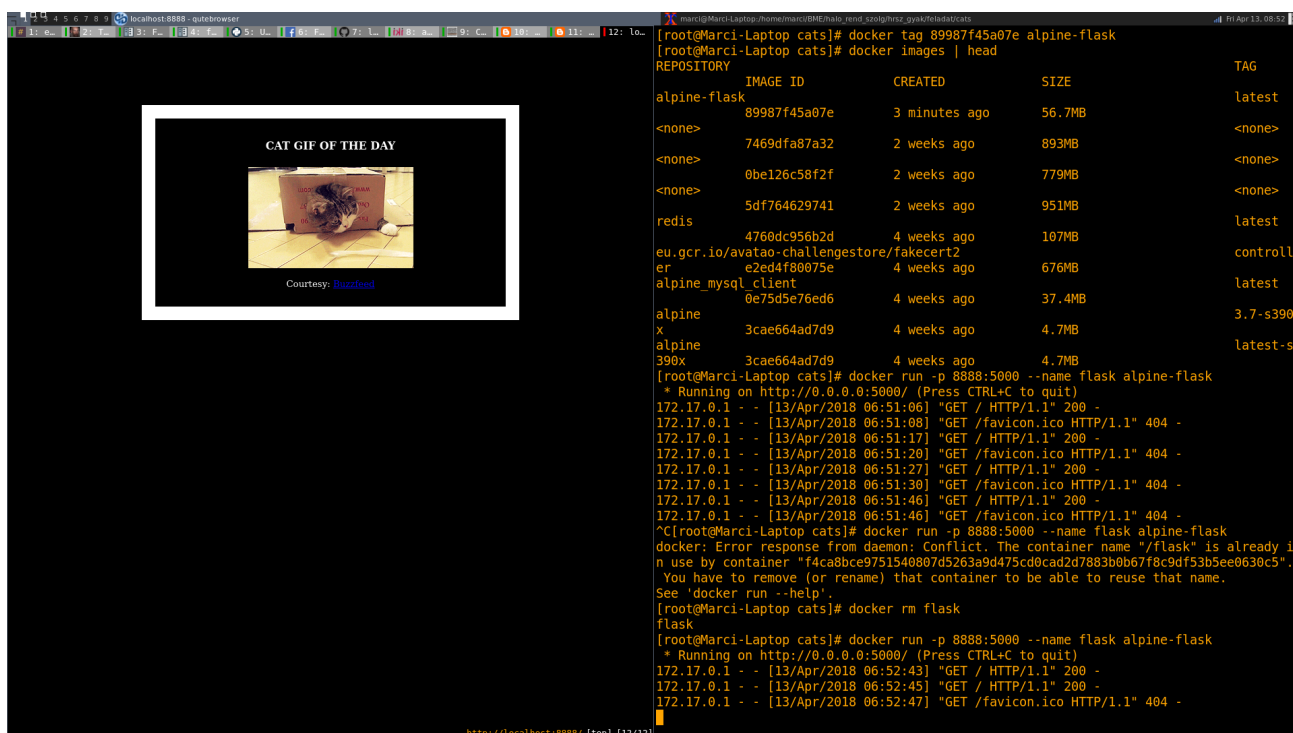
`docker build -t alpine-flask .`

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 webszerver működése a 4. ábrán látható.



4. ábra.

6. Docker swarm

A docker swarm lényege, hogy egy alkalmazást több containerben tudunk futtatni, így az erőforrásokat meg tudjuk osztani. A swarm tagjai containerek, amiknek két szerepük lehet: a manegerek irányítják a swarm működését, a workerek pedig elvégzik a feladatokat.

A swarm működtetéséhez először létrehoztunk két virtuális gépet, az egyikben a manager, a másikban a worker fog futni. Ezt az alábbi paranccsal tehetjük meg:

```
docker-machine create -driver virtualbox myvm1
```

```
docker-machine create -driver virtualbox myvm2
```

A `docker-machine ls` paranccsal listázhatjuk a futó virtuális gépeket, és az IP-címüket is megnézhetjük.

A swarmot az alábbi paranccsal hozhatjuk létre:

```
docker-machine ssh myvm1 "docker swarm init -advertise-addr <myvm1 ip>"
```

Ez a parancs a `myvm1` nevű gépben elindítja a swarmot. Ez a gép manager lesz. Létrehozáskor a swarm generál egy token, amire szükség van, amikor egy másik gép akar csatlakozni a swarmhoz. A swarmhoz workerként az alábbi paranccsal csatlakozhat a `myvm2` gép.

```
docker-machine ssh myvm2 "docker swarm join -token <token> <ip>:2377"
```

A docker swarm-ban service-eket futtathatunk. Egy service áll egy docker image-ből, valamint egyéb paraméterekből, például: containerek száma, CPU használati korlát, RAM használati korlát, port forwarding beállítások, stb. A mi példánkban egy korábban létrehozott, a docker hubra feltöltött image-et használtunk, amit 5 példányban futtattunk a swarmon.

A service-t a `docker-compose.yml` fájl írja le. Egy ilyen fájl például így nézhet ki:

```
1 version: "3"
2 services:
3   web:
4     # replace username/repo:tag with your name and image details
5     image: nemarci/get-started:part2
6     deploy:
7       replicas: 5
8       resources:
9         limits:
10          cpus: "0.1"
11          memory: 50M
12       restart_policy:
13         condition: on-failure
14     ports:
15       - "80:80"
16     networks:
17       - webnet
18 networks:
19   webnet:
```

5. ábra.

Ezt a service-t az alábbi paranccsal tudjuk futtatni a korábban létrehozott swarmon:

```
docker-machine ssh myvm1 "docker stack deploy -c docker-compose.yml getstartedlab"
```

Ezután a swarm elkezd futtatni a fenti fájlban megadott service-t.