K o n t é n e r i z á c i ó

2022. 09. 22.

by Walaki

Tartalomjegyzék

1 Kialakulás

Egy operációs rendszerben több alkalmazás illetve szolgáltatás futhat. Első sorban Linuxos környezetben felmerült az igény, hogy ezeket jó lenne egymástól minél jobban elválasztani, hogy ne férjenek hozzá egymás állományaihoz és ne is tudjanak egymásról. Alapvetően biztonsági kérdésként indult. Első szélesebb körben alkalmazott megoldás a `chroot` illetve a `FreeBSD alapú jail`. A `chroot` inkább csak fájlrendszerbeli szétválasztást végzett, míg a `jail` már adott egyfajta biztonságot is.

A linuxos rendszerek alapban szétválasztották a rendszer szintű (`kernel space`) és felhasználói szintű (`user space`) területekre a szoftver környezetet, a windows-ok csak később kezdték ezt alkalmazni. Ez alapvető biztonságot nyújt a operációs rendszer működése szempontjából, hiszen a felhasználói programok teljesen külön környezetben futnak, elszeparáltan a kerneltől. Megjelent egy olyan igény, hogy kernel szintű műveleteket felhasználói szint alól is el lehessen végezni. Ezen kívül az is igényként jelentkezett, hogy a felhasználói programokat is el lehessen egymástól különíteni. Ehhez az alapvetően monolitikus kernel felépítésén, működésén kellett változtatni.

Megoldották a kernel problémáit a `cgroups` és a névterek (`namespace`) bevezetésével. A 2.6.24-es (2008.01.24) kernel[1] már alkalmas volt a felhasználói szintű szeparálásra is.

Szinte azonnal elkészült az eredeti igénynek megfelelő konténerizációs technológia, az `LXC` – Linux Containers (2008.08.6.)[2]. Eredetileg az `LXC` konténerek nem nyújtottak többet, mint a `chroot`, de az `LXC 1.0` kiadástól kezdődően lehetőség van a konténerek normál felhasználóként való futtatására a gazdagépen "nem privilegizált konténerek" használatával. A nem jogosult tárolók korlátozottabbak, mivel nem férhetnek hozzá közvetlenül a hardverhez. Innentől lehet beszélni konténerizációról. Az `LXC` ma is elérhető a linuxos rendszereken.

A széles körben való alkalmazásához a `Docker`[3] megjelenéséig (2013.03.20) várni kellett. Onnantól azonban viharos gyorsasággal terjedt el. Először csak a fejlesztők oldaláról, azonban az általa nyújtott előnyök miatt már szinte mindenhol konténerekkel dolgoznak az informatikai területeken. Érdekesség, hogy kezdetben a `Docker` is `LXC`-t használt konténer végrehajtási rendszerként.

2 Fogalma

A konténerizáció „egy operációs rendszer szintű virtualizációs módszer több izolált Linux rendszer (tároló) futtatására egy vezérlő gazdagépen egyetlen Linux kernel használatával.” [4]

Ne tévesszen meg senkit itt a „virtualizáció”, itt szó sincs igazi virtualizációról, mégis kezdetben így nevezték el. Abból a szempontból tekinthető esetleg virtualizációnak, hogy ez a technika is egy látszólagos (virtualizált) környezetet biztosít az alkalmazásoknak, mint a hagyományos virtualizáció.

[1] https://kernelnewbies.org/Linux_2_6_24

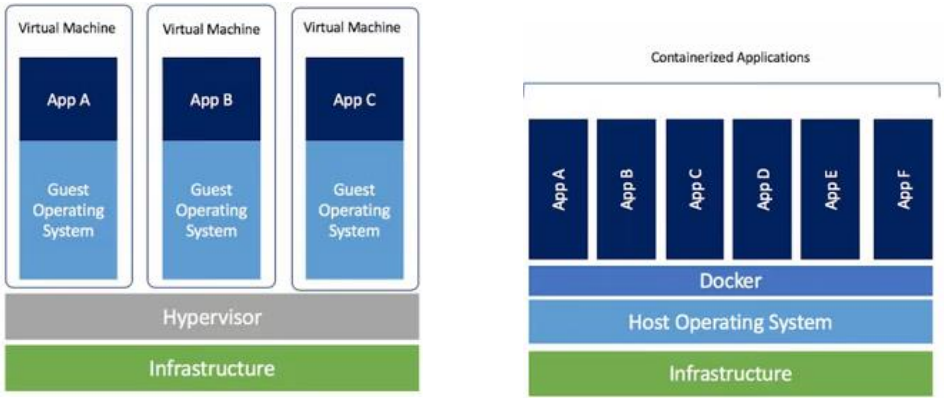
[2] https://en.wikipedia.org/wiki/LXC

[3] https://docs.docker.com/get-started/overview/

[4] https://en.wikipedia.org/wiki/LXC

3 Virtualizáció vs konténerizáció

A következő ábra sokat segíthet a megértésben:



1. ábra Virtualizáció vs konténerizáció - forrás[5]

Észrevehető, hogy konténerek esetén nincs külön vendég operációs rendszer, valamint nincs hypervisor se, vagyis tényleges hardver emuláció vagy virtualizáció sincs. A `Docker` vagy az `LXC` nem végez hardveres virtualizációt, csak az elkülönített tárolókból való futtatást teszi lehetővé, amit a host kernele hajt végre. Ebből adódóan jóval kevesebb erőforrást igényel, mint a hagyományos virtualizáció.

A konténerek jellemzői:

- önálló, izolált környezetet adnak,
- önálló (bár a legtöbb esetben lecsupaszított) operációs rendser a alapjuk,
- kisebb méretűek,
- gyorsabbak (a kernel ugyanaz),
- a fájlrendszerük futás közben maradandóan nem módosítható.

4 Alapfogalmak

4.1 Image

Gyakorlatilag egyetlen tömörített állomány, amely tartalmaz minden olyan fájlt, eszközöket és beállításokat, amelyek az alkalmazás futtatásához szükségesek. Önmagában nem lehet elindítani, ezt a futtató környezet (pl. `LXC` vagy `Docker`) fogja elvégezni.

4.2 Konténer (Container)

A konténer egy éppen végrehajtás alatt álló image.

A konténer futtató környezet (pl. `Docker`) kezeli, biztosítja, hogy a konténerben lévő alkalmazást a host rendszer kernele végrehajtsa, hozzárendel egy portot, amin keresztül kommunikálhat a konténer belsejével, szükség esetén hálózati kapcsolatot teremt a host és a konténerben futó alkalmazás között, valamint biztosít a futó konténer részére egy írható réteget, ahova a konténer alkalmazásai írhatnak. Ez az írható réteg a konténer leállásakor törlődik, vagyis a futás során odaírt adatok nem maradnak meg. A legközelebbi indításnál az image eredeti állapotából indulunk. Nem kell megijedni azonban az adatvesztéstől, mert külön van arra megoldás, hogy a futás közben létrejött adatokat megőrizzük (`Docker` esetén pl. a `Persistence Volumes`).

4.3 Registry

Ez egy olyan szolgáltatás, ami az image fájlokat tárolja és teszi elérhetővé, akár bárki számára.

[5] https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/

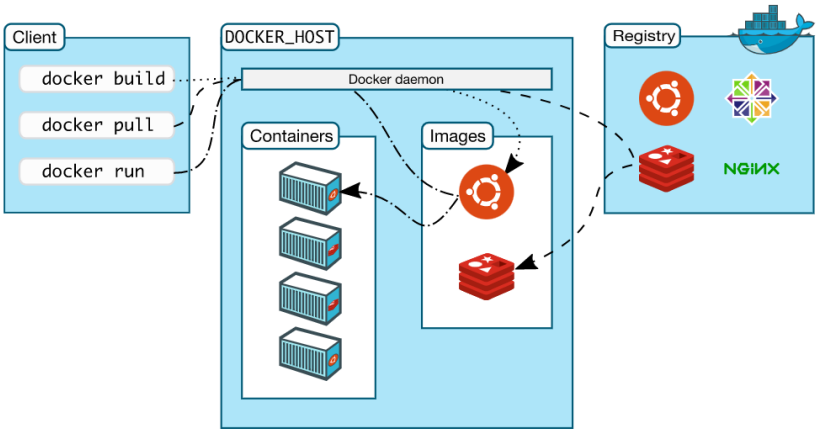
A Docker által kezelt nyilvános ilyen szolgáltatás a Dockerhub[6], amit bárki használhat igyenesen, ide feltöltheti saját image-it, illetve elérheti a fent lévő, akár mások által készített imageket is.

4.4 Konténer orkesztátorok

Az elmúlt években elterjedt mikroszerviz-alapú alkalmazás-fejlesztést a konténerek remekül támogatják, hiszen minden mikroszerviz a saját környezetében futhat, és megoldhatóvá válik az alkalmazás egyes rétegeinek önálló skálázása. A sok konténer együttműködésének vezénylésére konténer-orkesztrátorok állnak rendelkezésre. Ha a konténerünk Docker-alapú, akkor e tekinteteben a Docker Swarm, a Kubernetes és a Mesos áll rendelkezésünkre.

5 A Docker

https://docs.docker.com/get-started/overview/



2. ábra Docker architektúra – forrás[7]

A Docker alapvetően linuxra, linuxos környezetre íródott. Windowson csak 2016-ban lett elérhető és először csak hyperv-vel virtualizált linux környezettel. Most már eljutottak oda, hogy windows-on sem kell virtualizálni a konténerek futtatásához. Ebből adódóan egy image biztosíthat linuxos vagy windowsos környezetet is a konténerben futó alkalmazás részére.

A Docker elérhető

- Linux,
- Windows,
- macOS

alá is és létezik

- szerver
- és desktop változata is.

Docker változatok:

- Linuxos docker, konzolos alkalmazás, ami a docker parancs segítségével kezelhető
- Windowsos docker, konzolos alkalmazás, ami a docker parancs segítségével kezelhető
- macOS-es docker, konzolos alkalmazás, ami a docker parancs segítségével kezelhető
- Docker Desktop for Linux[8], komplett grafikus környezet
- Docker Desktop for Windows[9], komplett grafikus környezet
- Docker Desktop for Mac[10], komplett grafikus környezet

---

[6] https://hub.docker.com/

[7] https://docs.docker.com/get-started/overview/

[8] https://docs.docker.com/desktop/install/linux-install/

[9] https://docs.docker.com/desktop/install/windows-install/

[10] https://docs.docker.com/desktop/install/mac-install/

- Docker Toolbox (régebbi Windowsok esetén – Win7), virtualizáltan futtatja a linuxos docker daemon-t

5.1 Docker telepítése Linuxon

Csomagkezelőkből mindenhol telepíthető, `debian/ubuntu` esetén pl.:

```
apt install docker.io
```

A snap, platformfüggetlen csomagkezelővel pl.:

```
snap install docker
```

A kettő között annyi az eltérés, hogy az `apt` külön felhasználói csoportot is készít a host-on, amibe bármelyik egyszerű felhasználót is felvehetünk, akik innentől kezelni tudják a konténereket, míg a `snap`-el telepített változatnál csak `root` jogokkal kezelhetjük a konténereket.

5.2 Docker telepítése Windowson (kliensen és szerveren is)

Adminisztrátorként `PowerShell`ben:

```
Install-WindowsFeature Containers
Restart-Computer –Force
Install-Module -Name DockerMsftProvider –Force
Install-Package -Name Docker -ProviderName DockerMsftProvider -Force
Restart-Computer –Force
```

Kezelése `PowerShell`ben a docker paranccsal.

Ezzel lehet futtatni Windowsos konténereket, pl.:

```
docker run hello-world:nanoserver
```

Ha Linuxos konténereket is akarunk futtatni, akkor a következő lépéseket is végre kell hajtani:

```
[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", "1", "Machine")
Restart-Service docker
New-Item "C:\ProgramData\docker\config\daemon.json"
Az imént létrehozott .json fájlba helyezzük el a következő tartalmat:
{
  "experimental": true
}
Invoke-Webrequest -Uri "https://github.com/linuxkit/lcow/releases/download/v4.14.35-v0.3.9/release.zip" -UseBasicParsing -OutFile release.zip
Expand-Archive release.zip -DestinationPath "$Env:ProgramFiles\Linux Containers\."
Restart-Computer -Force
```

Az ezután kiadott

```
docker info
```

parancs kimenetén látnunk kell az `lcow (linux)` kiírást is a `windows` mellett. Most már tudunk futtatni Windowsos és Linuxos konténereket is!

5.3 Docker alapgyakorlat- első image és első konténer

A

```
docker run hello-world:linux
```

paranccsal beszerezhetjük és futtathatjuk első konténerünket. Lévén a konténer alapjául szolgáló image helyben nem volt elérhető, a Docker démon elment érte a Docker Hub-ra, letöltötte és indított belőle egy konténert. Ez a konténer üzenetet

írt, ami megjelent a terminálunkban. Tekintve, hogy a repository (`hello-world`) előtt nincs szervernév, ebből tudjuk, hogy az image nem saját üzemeltetésű Docker szerverről származik, hanem magáról a Dockerhubról.

Ha a

```
docker image ls (vagy docker images, esetleg docker image list)
```

parancsot adjuk ki, látni fogjuk a gépünkre jelenleg letöltött image-eket. Az image-eket egy azonosító azonosítja, és látjuk még, hogy van egy tárolójuk (repository), illetve egy címkéjük (tag).

A

```
docker container ls
```

paranccsal megnézhetjük a jelenleg futó konténereinket. Jelenleg egy sem fut, ugyanis az eddigi egyetlen konténerünk az üzenet kiírásával be is fejezte a működését. A komolyabb konténerek többnyire szerver funkciót látnak el, azaz nem lépnek ki a futtatást követően, és itt látni fogjuk őket.

A futást befejezett konténerek is láthatók a

```
docker container ls --all
```

parancsot kiadva. Itt látjuk, hogy a konténereknek neve is van (ellentétben az image-ekkel), ha mi nem adunk meg neki, akkor a Docker választ számára egy – általában meghökkentő – nevet. Amikor egy-egy konténerrel dolgozunk, az azonosítót és a nevet is használhatjuk. A konténereknek megnézhetjük a kimenetét (azaz a szabványos kimenetre, az stdout-ra, illetve a szabványos hibacsatornára, az stderr-re érkező üzeneteket).

```
docker container logs
```

paranccsal, a

```
docker container inspect
```

pedig arra való, hogy a konténer állapotát megszemléljük. Ha az eddigi egyetlen konténerünket nézzük, akkor például kiderül, hogy nem volt hibaüzenete, és az `OOMKilled: false` azt is elmondja nekünk, hogy nem memóriahiány miatt kapcsolta le a `Docker démon`. A kilépett konténerek megtartása azért is jó dolog, mert így meg tudjuk nézni, hogy mi volt az állapotuk a futás végén. A már szükségtelen konténert a

```
docker container rm <ID vagy név>
```

paranccsal távolíthatjuk el (`rm`). Tegyük is ezt most meg. Ha az eddigi egyetlen image-ünkből újabb konténert futtatnánk a

```
docker run --rm hello-world:linux
```

paranccsal, akkor észrevesszük, hogy a letöltés elmarad (hiszen már le van töltve az image), az `--rm` kapcsolónak pedig az a hatása, hogy a konténer nem marad meg a kilépését követően – ellenőrizhetjük is. Az immáron szükségtelen image a

```
docker image rm
```

paranccsal távolítható el – ne habozzunk megtenni.

5.4 Docker alapparancsok

```
docker info
docker version
docker images
docker pull image_neve
docker run image_neve
docker run --rm image_neve
docker run --name kontener_neve image_neve
docker run -d image_neve
docker run --rm -d --name kontener_neve -p 80:8000 image_neve
```

```
docker run -d -ti --name kontener_neve -p 80:8000 image_neve
  futtatando_parancs_es_parameterei
docker ps
docker ps -a
docker stop kontener_neve
docker start kontener_neve
docker logs kontener_neve
docker inspect kontener_neve
docker rm kontener_neve
docker rm -f kontener_neve
docker exec -ti kontener_neve futtatando_parancs
docker image rm image_neve
```

5.5 Összetettebb konténer futtatása- hálózatkezelés

A következő konténerünk egy webszervert futtat majd, ami a konténerben lévő `images` mappában lévő képeket jeleníti meg egy képgaléria formájában. A konténer letöltését és futtatását ezúttal két lépésben végezzük, pusztán a tanulás okán. A

```
docker pull walaki/kepgaleria
```

parancs végzi a letöltést, és a szokásos

```
docker run walaki/kepgaleria
```

indítja el a konténert. Nem ír ki semmit és a konzolt se kapjuk vissza. A konténer egyébként az nginx webszervert futtatja és a 8000-es porton figyel. A konténer az előtérben maradt, és így nem nagyon tudunk mit tenni vele (azért örülünk a lehetőségnek, látjuk, hogy egy napon ez még jó lesz hibakeresésre), úgyhogy a `Ctrl+C` billentyűparanccsal állítsuk meg. Persze nem használtuk az `--rm` kapcsolót a futtatáskor, úgyhogy távolítsuk el az álló konténert kézzel, az oktatás további részében pedig használjuk a `--rm` kapcsolót. Adjuk ki a

```
docker run --rm -d --name kepgal walaki/kepgaleria
```

parancsot. Ahogy vártuk, letöltés nincs, a `--rm` kapcsolóval megismerkedtünk. A `-d` a deatched, azaz lecsatolt futást kéri – ezúttal a háttérben fut a konténerünk, a `--name` pedig arra jó, hogy már most tudjuk, hogy milyen néven hivatkozhatunk a konténerünkre. Nézzük meg, hogy fut-e a konténerünk a

```
docker ps -a
```

paranccsal. Itt azt vegyük észre, hogy a PORTS oszlopban megjelenik a 8000/tcp, vagyis maga a konténer igenis figyel a 8000-es porton. Ha viszont a host kiadjuk az

```
ss -tlpnu
```

parancsot, akkor nem látunk szolgáltatást a 8000-es porton. A host gépről azonban hozzá tudunk férni a konténer adott portjához csak ismernünk kéne a konténer IP címét. Adjuk ki a


```
docker inspect kepgal
```

parancsot és keressük meg az IP-t! A megjelenített információk végén a `NetworkSettings IPAddress` mezőben találhatjuk meg. Itt rendszerint a `172.17.0.2` címet fogjuk találni. A konténerek alapesetben egy különálló hálózatba kerülnek, ahonnan NAT-al érik el a külső hálózatot, ebből adódóan a külvilág felől elérhetetlenek, csak és kizárólag a host-on keresztül lehet elérni őket. Most nézzük meg, hogy a host alól tényleg el lehet-e érni, adjuk ki a

```
curl 172.17.0.2:8000
```

parancsot. Ha minden rendben van, akkor megjelenik egy HTML tartalom, ami a képgaléria alap oldala.

Ha azt szeretnénk, hogy ne csak a host gépről legyen elérhető, akkor a konténer portját publikálnunk kell a gazdagépünkre, azaz először is állítsuk meg a konténert a

```
docker container stop kepgal
```

paranccsal, majd futtassuk újfent, a

```
docker run --rm -d --name kepgal -p 80:8000
  walaki/kepgaleria
```

egysoros paranccsal, ahol a 80-as gazdaportra kötjük a konténer 8000-es portját. A host-n kiadott `ss -tlpnu` parancs eredményeképp pedig megjelenik a 80-as porton is egy szolgáltatás, konkrétan a `docker-proxy`. Ha egy böngészőben a host gépünk IP-címét adjuk meg, látnunk kell a képgaléria.

5.6 A konténer belsejében és adattárolás

Vizsgáljuk meg a konténer belsejét, lépjünk be a konténerünkbe, adjuk ki a

```
docker exec -ti kepgal /bin/sh
```

parancsot, azaz a futtassuk az alapértelmezett parancsértelmezőt `(/bin/sh)` a konténer belsejében.

Amikor beléptünk, az `ls` paranccsal kilistázhatjuk a weboldal fájljait. Láthatjuk, hogy az oldal PHP alapú és a

```
cat index.php
```

paranccsal meg is szemlélhetjük az alapoldalt. A képeket az `images` mappában találjuk.

Lépjünk be az `images` mappába (`cd images`), majd töltsünk le két képet:

```
wget http://walaki.infora.hu/macska1.png
wget http://walaki.infora.hu/macska2.jpg
```

majd frissítsük a böngészőnkben a konténerben futó szerver kiszolgálta oldalt. Nem kell ezekhez a képekhez ragaszkodni, bármilyen képeket letölthetünk. Az újonnan letöltött képeket is látnunk kell!

Ha úgy tarja kedvünk, akár telepíthetünk alkalmazást a konténerbe (elvégre ez egy pici Alpine Linux), az alábbi parancs például a Midnight Commander fájlkezelőt telepíti:

```
apk add mc
```

Elindítani az `mc` paranccsal lehet, az `F10` a kilépés.

Ha kinézelődtünk magunkat, akkor lépjünk ki a konténerből (`exit`, vagy `Ctrl+D`). Az Ubuntunkba visszajutva állítsuk meg a konténert

```
docker container stop kepgal
```

majd indítsuk el újra. A böngészőnket frissítve azt látjuk, hogy a letöltéseink eltűntek – az image-ek fájlrendszere nem módosítható. Lehetőség van azonban a konténer mappáit kivezetni. Állítsuk meg a konténert.

Először a gazdagépen a home könyvtárunkban hozzunk létre egy mappát (`mkdir direktcsatolt`), majd váltsunk bele (`cd direktcsatolt`) és az előző két `wget`-paranccsal töltsük le a két ismert képet (persze bármilyen egyéb kép is jó). Ha megvagyunk, futtassuk ismét a konténerünket a


```
docker run -v ~/direktcsatolt:/app/images --rm -d --
  name kepgal -p 80:8000 walaki/kepgaleria
```

egysoros paranccsal. A parancsban gazda `~/direktcsatolt` mappáját kötjük össze a konténerben lévő `/app/images` mappával, azaz amit a gazda mappájában elhelyezünk, azt a konténer is tudja olvasni.

Figyeljünk, hogy se a helyi, se a konténerbeli elérési utat ne gépeljük el, mert a Docker csendben létrehozza, és azt fogja felcsatolni. Ha minden jól ment, ezúttal nem látszik a két kutyus kép, csak amiket mi letöltöttünk a mappába. Ezzel a

megfigyeléssel választ kaptunk arra, hogy miként lehet bővülő tartalmú webszervereket, vagy például adatbázis-kiszolgálókat üzemeltetni konténerből.

5.7 Saját image készítése kézzel és feltöltése Dockerhub-ra

A Docker nagyok sok kész image-et tesz elérhetővé, de ha van egyedi igényünk, akkor mi is tudunk készíteni egyedi image-eket. Erre több lehetőségünk is van:

- meglévő futó konténer alapján (commit)
- egy konfigurációs fájl (Dockerfile) alapján (build)
- Dockerfile és CM (Config Management) segítségével
- üres image és import tar fájlok segítségével

Az első két lehetőséget a gyakorlatban is megnézzük, ugyanis ezek a leggyakoribb eljárások erre a célra. Ebben a fejezetben pedig az első lehetőséget vizsgáljuk meg. Most a célunk egy olyan webszerver image elkészítése, ami PHP futtatására is képes.

Induljunk ki egy `ubuntu` alap image-ből:

```
docker pull ubuntu
```

Futtassuk interaktív módban, majd csatlakozzunk rá:

```
docker run -d --name ubi -ti ubuntu bash
docker exec -it ubi bash
```

Ezután a konzolban azt csinálunk, amit akarunk, ugyanúgy, ahogy egy linuxos rendszer előtt ülnénk. Telepíthetünk, indíthatunk bármit, ami elérhető az adott környezetben.

```
uname -a
apt udate
apt upgrade
apt install apache2 php
```

Eddigi ismereteink szerint ilyenkor automatikusan elindult a webszerver és hozzá is lehet férni. Hogyan tudjuk ezt most ellenőrizni? Azt várnánk, hogy a konténerben a web szolgáltatás a 80-as porton figyel. A

```
ps aux
```

parancs kimenete azt mutatja, hogy a rendszerben csak a bash fut, semmi más.

Az `ss` paranccsal is szoktuk ellenőrizni a szolgáltatások működését. Próbáljuk meg!

```
ss -tlpnu
```

Azt látjuk, hogy a rendszerben nincs `ss` parancs. A `whereis` parancs sem ad találatot. A letöltött image nem tartalmazza azt a csomagot, amiben az `ss` benne van. Az `ss` az `iprute2` csomagban található. Az oldal megjelenítéséhez a `w3m`-et is használhatjuk. Telepítsük mindkettőt:

```
apt install iproute2 w3m
```

Próbálkozások után kiderül, hogy nem fut az `apache`, nincs `systemd` sem. Indítsuk el az `apache`-ot:

```
/usr/sbin/apachectl start (vagy apachectl start vagy service apache2 start)
```

Most már figyel a 80-as porton, ellenőrizhetjük a tartalmat is:

```
w3m localhost
```

A PHP kód futtathatóságának ellenőrzésére készítsünk egy `index.php` fájlt, az alapértelmezett site könyvtárába:

```
echo "<?php phpinfo(); ?>" > /var/www/html/index.php
```

Ellenőrizzük a `php` működését `apache`-on keresztül:

```
w3m localhost/index.php
```

Ha azt szeretnénk, hogy az indexfájlok sorrendjében az `index.php` előrébb legyen, mint az index.html. Ehhez módosítani kell a default site konfigurációját:

```
sed -i '/<\/VirtualHost>/i\\tDirectoryIndex index.php index.html' /etc/apache2/sites-enabled/000-default.conf
```

Majd indítsuk újra az `apache`-ot és ellenőrizzük:

```
apachectl restart
w3m localhost
```

Lépjünk ki a konténerből:

```
exit
```

A konténer még fut, ellenőrizzük:

```
docker ps
```

Regisztráljunk magunknak egy egyedi `Dockerhub`-os fiókot a https://hub.docker.com/ oldalon. Lépjünk be a böngészővel és ismerkedjünk a felülettel! Az alap oldal tetején az `Explore` menüpontot kiválasztva egy kereshető, szűrhető listát kapunk a nyilvánosan is elérhető image-ekről. A saját, általunk készített image-eket is ide tölthetjük majd fel és azt is meghatározhatjuk, hogy nyilvánosan is elérhetővé tesszük (public) vagy csak mi férhetünk majd hozzá (private).

A továbbiakban a saját azonosítónevünket kell mindenhol használni, ahol itt a példákban a **`username`** szerepel!

A futó konténerből készíthetünk egy saját image-et:

```
docker container commit ubi username/ubiphp:0.1
```

Az `ubi` a futó konténer neve, a **`username`** Dockerhub azonosító, ha még nincs, az sem baj. Ilyenkor a local előtagot is lehet használni, de ha a Dockerhub-ot is szeretnénk használni, akkor célszerű már most a saját azonosítónkkal dolgozni. Az `ubiphp` a létrehozott image neve, a „`:`” utáni rész pedig az úgynevezett tagname, ami gyakorlatilag a verziót jelenti. Ellenőrizzük, hogy létrejött-e:

```
docker images
```

Most ellenőrizzük a létrejött image működését. A futó konténert állítsuk le és töröljük is. Majd indítsuk el az image-ünket úgy, hogy a linuxos host IP-jén is elérhessük:

```
docker rm -f ubi
docker run --rm -d --name ubiphp -p 80:80 username/ubiphp:0.1
```

Azt látjuk, hogy kilép. Nézzük meg konzol alatt, hogy miért nem fut:

```
docker run --rm -d -ti --name ubiphp -p 80:80 username/ubiphp:0.1 bash
docker exec -ti ubiphp bash
ss -tlpnu
cat /var/log/syslog
```

Nem indul el az `apache`. Vegyük észre, hogy a konténerek esetén nincs `init`, nincs `systemd`, nincs automatikus rendszer indítási folyamat, itt mindent nekünk kell megadni. Másik fontos tapasztalat, hogy ha nincs elindítva semmilyen program a konténerben, akkor be fogja fejezni a futását. Az `apache`-ot is az előtérben kell indítani, ha folyamatosan futtatni szeretnénk, ezt pedig az

```
/usr/sbin/apache2ctl -D FOREGROUND
```

paranccsal tehetjük meg. Ezt ki is próbálhatjuk! `CTRL+C` -vel állítsuk, majd lépjünk ki a futó konténerből.

Készítsünk egy másik image-t, ami már „rendesen” fog működni. Ehhez a következő formában kell kiadni a `commit` parancsot:

```
docker container commit -m "Entry foreground-al" --change='ENTRYPOINT ["/usr/sbin/apachectl", "-D", "FOREGROUND"]' ubiphp username/ubiphp:0.2
```

Itt a `-m` paraméter után egy rövid leírást adhatunk az `image`-ünkhöz, a `--change` utáni rész, pedig beállítja a konténer belépési pontját (`ENTRYPOINT`), és meghatározza, hogy milyen szolgáltatás induljon el a konténer futtatásakor. Itt már az image tagname-nél egy másik verziót használtunk (0.2) jelölve, hogy különbözik az előzőtől.

Állítsuk le (ezzel törlődik is a `--rm` miatt) a futó konténert, majd indítsuk el az újabb, 0.2-es változatot:

```
docker stop ubiphp
docker run --rm -d --name ubiphp -p 80:80 username/ubiphp:0.2
```

Ellenőrizzük a webszerverünk elérhetőségét a host alól

```
curl localhost   vagy   w3m localhost
```

és lehetőség szerint távolról, másik gép alól is a böngészőbe a linuxos host gép IP címét beírva!

Ha mindent rendben találunk, akkor töltsük fel a működő image-t a `Dockerhub`ra, ehhez először konzol alól is be kell jelentkezni a

```
docker login -u username
```

paranccsal, ahol a **username** a saját `dockerhub`-os azonosító. A feltöltéshez pedig a következő parancsot kell kiadni:

```
docker push username/ubiphp:0.2
```

Ellenőrizzük a feltöltött `image`-ünket úgy, hogy leállítunk és letörlünk minden `konténert` és minden `image`-t a host-ról, majd töltsük le újból és futtassuk, majd ellenőrizzük is a web szolgáltatást és PHP-s oldal megjelenítést is!

Összefoglalva a készítés lépései a következők:

- Regisztráljunk a `Dockerhub`-ra, ha még nem tettük meg!
- `docker pull ubuntu  //ha még nincs letöltve`
- `docker run -d --name ubi -ti ubuntu bash`
- `docker exec -it ubi bash`
- `apt udate`
- `apt upgrade`
- `apt install apache2 php`
- `echo "<?php phpinfo(); ?>" > /var/www/html/index.php`
- `sed -i '/<\/VirtualHost>/i\\tDirectoryIndex index.php index.html' /etc/apache2/sites-enabled/000-default.conf`
- `exit`
- `docker container commit -m "Entry foreground-al" --change='ENTRYPOINT ["/usr/sbin/apachectl", "-D", "FOREGROUND"]' ubiphp walaki/ubiphp:0.3`
- `docker login -u username  //ha még nem jelentkeztünk be`
- `docker push username/ubiphp:0.2`

5.8 Dockerfile

Az előzőleg készített image-et egyszerűbben is el tudjuk készíteni, egy egyszerű szöveges fájl, a `Dockerfile` segítségével. Hozzunk létre a felhasználó könyvtárunkban egy új mappát, miben pedig egy `Dockerfile` nevű fájlt:

```
mkdir dockerfiles
cd dockerfiles
touch Dockerfile
```

Helyezzük el benne a következő tartalmat:

```
FROM ubuntu:latest
RUN apt update
```

```
RUN apt -y upgrade
RUN echo 'tzdata tzdata/Areas select Europe' | debconf-set-selections
RUN echo 'tzdata tzdata/Zones/Europe select Budapest' | debconf-set-selections
RUN apt update && DEBIAN_FRONTEND="noninteractive"
RUN apt install -y tzdata
RUN apt install -y apache2
RUN apt install -y php
RUN rm -rf /var/lib/apt/lists/*
RUN echo "<?php phpinfo(); ?>" > /var/www/html/index.php
RUN sed -i '/<\/VirtualHost>/i\\tDirectoryIndex index.php index.html' /etc/apache2/sites-enabled/000-default.conf
EXPOSE 80
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```

Figyeljünk oda a másolás után, hogy ne törje meg a sort (`RUN sed` kezdetű sor) és az idézőjelek típusa is megfelelő legyen!

Készítsünk ez alapján egy image-t:

```
docker build -t username/ubiphp:0.4 .
```

Figyeljünk oda, hogy a `build` az aktuális vagy a megadott könyvtárban a többi állományokat is figyeli, és a `COPY` paranccsal akár az image megfelelő pontjára is bemásolhatók. Lényeges, hogy a buildelés mappájában csak azok a fájlok, mappák legyen, amiket használni is fogunk a `build` során. A `.dockeringnore` fájlban megadhatjuk, hogy milyen fájlokat, könyvtárakat szeretnénk kihagyni mondjuk a másolásból.

Ellenőrizzük az elkészült image működését lokálisan a host alól vagy távolról! Ha minden rendben, akár fel is tölthetjük!

Mint látszik a `Dockerfile` használata egyszerű. A `FROM` után megadjuk, hogy mi a kiinduló image. A `RUN` után végrehajtandó parancsokat határozhatunk meg, az `EXPOSE` meghatározza, hogy a konténer melyik porton nyújt szolgáltatást, a `CMD` pedig a belépési pont-hoz hasonlóan a konténer elindulása után végrehajtandó parancsot határozza meg. Itt nagyjából azokkal a parancsokkal találkozhatunk, amiket a kézi image készítés során kiadtunk.

Van azonban egy erős megkötés. Olyan utasításokat kell használnunk, ami nem vár választ, vagyis a futtatás során nem áll meg, hanem kérdés nélkül végrehajtódik. Emiatt vannak itt újabb parancsok, amik ezt a megállást küszöbölik ki. A PHP telepítésekor ugyanis az időzóna beállításokat is meg kellene adni, a 4.-7. sorok kerülik ezt ki.

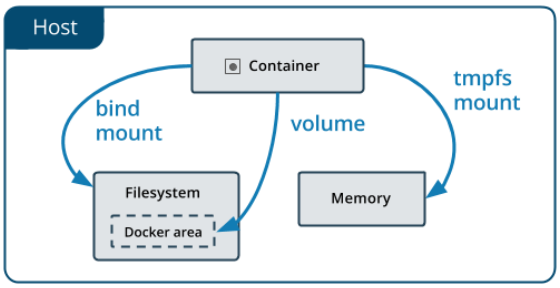
A `RUN rm -rf` kezdetű sor, a csomagkezelő átmeneti fájljait törli ki, így kicsit kisebb méretű lesz az image.

További hasznos információk a Dockerfile használatával kapcsolatban:

- Dockerfile bevált gyakorlatok/követendő pédák: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- Dockerfile reference: https://docs.docker.com/engine/reference/builder/
- MS oktató anyag: https://learn.microsoft.com/hu-hu/training/modules/intro-to-docker-containers/

5.9 Adattárolás - Storage

Ennek a szolgáltatásnak a segítségével valósítható meg, hogy a futó konténerekben létrejött adatok háttértáron megőrzésre kerülhessenek. Alapvetően az image tartalma futás közben nem módosítható, azonban a futó konténernek vagy egy olyan része, ami írható, viszont amint leáll a konténer futása, ez az írható terület tartalom is elvész, törlődik. Ezért van szükség más megoldásra. A Docker több megoldást is kínál.



3. ábra Adattárolás Docker alatt[11]

5.9.1 Bind mounts - direkt csatolás

Ebben az esetben közvetlenül meghatározzuk azt a hoston lévő mappát, ahová a konténer valamely mappáját kapcsoljuk. Ezt a megoldást alkalmaztuk a `direktcsatolt` mappa feladatnál.

```
docker run –v /fizikai/rendszer/fs:/container/fs
docker run –v /home/mount/data:/var/lib/mysql/data
```

Mivel a konténerből hozzáférhetünk a host fájlrendszeréhez, akár a rendszerfájlokhoz is, így ez megoldás biztonsági kockázatokat is rejthet magában. Hatékony ugyan, de a gazdagép fájlrendszerére támaszkodik. A host gépen nem kell hogy létezzen a mappa, ahová csatoljuk a konténer mappáját, automatikusan létrehozásra kerül.

5.9.2 Volumes - kötetek

Ezt a tároló fajtát teljes egészében a Docker maga kezeli és el van választva a host gép alapvető funkcióitól. Ez a megoldás már biztonsági szempontból is megfelelő.

Két fajtája van:

- Anonym - névtelen
- Named - nevesített

Az `anonym` esetében maga a Docker látja el egy véletlen szerű névvel, ami a hoston egyedi név, a `nevesített` esetben, pedig mi adhatunk a kötetnek egy nevet.

Egy kötet létrehozása a

```
docker volume create
```

vagy a

```
docker volume create <kötetnév>
```

paranccsal történik.

A köteteket a

```
docker volume ls
```

paranccsal listázhatjuk és a

```
docker volume inspect <kötetnév>
```

paranccsal kaphatunk róluk bővebb információt.

[11] Forrás: https://docs.docker.com/storage/

A köteteket a konténer indításakor kapcsolhatjuk fel

```
docker run –v <név>:/container/fs
```

és a

```
docker volume rm
```

vagy

```
docker volume prune
```

parancsokkal törölhetők.

A kötetek támogatják a kötet-illesztőprogramok (driverek) használatát is, amelyek lehetővé teszik adataink távoli gazdagépeken vagy felhőszolgáltatókon való tárolását.

A `volumes` a javasolt eljárás adattárolási célra!

5.9.3 tmpfs mounts

A tmpfs csatolás ideiglenes tárterület a host memóriájában, ami a konténerből is elérhető, de nem marad meg a tartalma a konténer leállítása után. Jellemzően ideiglenes adattárolásra és érzékeny adatok konténerekbe való beillesztésére használatos.

```
docker run -d -it --name tmptest --tmpfs /app nginx:latest
```

5.9.4 named pipes

Ezek nevesített kapcsolatok a gazdagép és a konténer közötti kommunikációhoz.

További információk az adattárolásról:

https://docs.docker.com/storage/

5.10 Docker hálózatok

A Docker hálózati alrendszere illesztőprogramok segítségével csatlakoztatható. Alapértelmezés szerint számos illesztőprogram létezik, amelyek alapvető hálózati funkciókat biztosítanak a konténerek részére:

- bridge - a konténerek ugyanabban a host-tól különböző hálózatban vannak, elérhetik egymást és NAT-olva érik el a külső hálózatokat.
- host - a konténer hálózati verme nincs elválasztva a host hálózatától, nincs külön IP címe se, a host IP címével kommunikál és a szolgáltatások is közvetlenül a host-hoz kapcsolva jelennek meg. Ha a konténer a 80-as porton webszolgáltatást biztosít az olyan, mintha a hoston indítottuk volna el.
- overlay - elosztott hálózat több Docker host között, ráépülve a host hálózatára
- ipvlan - összetett lehetőséget biztosít mind IPv4 mind IPv6 alapokon, kiegészülve a 2. és 3. rétegbeli VLAN szolgáltatásokkal
- macvlan - direkt hozzáférést biztosít a fizikai hálózathoz a konténerek számára egy egyedi MAC cím hozzárendeléssel
- none - ebben az esetben a konténer nem fog rendelkezni semmilyen hálózattal, Docker API-n keresztül kommunikálhat
- egyéb beépülő modulok - egyedi fejlesztésű, külön telepíthető kiegészítő modulok is elérhetőek, amelyek a fenti szolgáltatásokat kiegészíthetik vagy akár kombinálhatják is

Az eddigi példákban nem konfiguráltuk külön a konténerek hálózatát, így minden esetben bridge típusú hálózattal rendelkeztek a futtatás során, mivel az alapértelmezett hálózati kapcsolat a bridge mód.

A létező hálózatokat a

```
docker network ls
```

paranccsal listázhatjuk ki, a

```
docker network inspect <név>
```

paranccsal részletes információkat kaphatunk róla.

Új hálózatot a

```
docker network create <név>
```

pl.:

```
docker network create -d "nat" --subnet "192.168.100.0/24" network01
```

paranccsal hozhatunk létre és a

```
docker network rm <név>
```

paranccsal törölhetjük.

A hálózatokat a konténer futása esetén is hozzáadhatjuk

```
docker network connect <hálnév> <konténernév>
```

illetve lecsatlakoztathatjuk a

```
docker network disconnect <hálnév> <konténernév>
```

paranccsal.

Részletes információk:

https://docs.docker.com/network/

5.11 Monitoring Docker környezetben

5.11.1 Hagyományos megoldások

Mivel a konténerekben is rendszerint valamilyen szolgáltatás fut, így a különböző szolgáltatás figyelő, monitorozó rendszer (`Monit, Nagios`) itt is használható. Konténerizált környezetben azonban csak a szolgáltatásokhoz férnek hozzá és ott is csak akkor, ha a konténer hálózatok ezt lehetővé teszik. A részletesebb terhelési adatokhoz csak a host gépen lehet hozzájutni és ott is csak akkor, ha a monitorozó rendszer fel van erre készítve, vagyis képes a `Docker API`-n keresztüli kommunikációra.

5.11.2 Docker stats

A Docker engine futási környezetéről, a konténerek állapotáról jelenít meg információkat, amiket akár monitorozni is lehet:

```
docker stats
```

5.11.3 Prometheus

`A Prometheus` egy nyílt forráskódú rendszerfigyelő és riasztási eszköztár, kifejezetten `Docker` környezetek figyelésére úgy, hogy maga is egy konténerként fut. Az adatokat és beállításokat egy böngészőn keresztül érhetjük el.

Alap információk:

https://docs.docker.com/config/daemon/prometheus/

Bővebb információk:

https://prometheus.io/docs/introduction/overview/

5.12 Docker Composer

A `Docker Compose` a `Docker` olyan plusz szolgáltatása, amely `YAML` fájlok segítségével definiált, jellemzően összetartozó konténereket tud automatikusan futtatni.

A `YAML` fájlok egyszerű szöveges fájlok, amelyekben jellemző módon név és érték párosokat határozunk meg hierarchikus szerkezetben. A `YAML` fájlok felépítése kötött, a sorokban lévő adatok kezdő pozíciójának lényeges szerepe van. A fájl neve rendszerint `docker-compose.yml`. Ettől eltérő nevet is lehet használni, csak akkor azt a parancs futtatásakor meg kell határozni paraméterként.

5.12.1 Telepítés

A `Docker Desktop` változatok alapban tartalmazzák, ekkor nem kell semmit külön telepíteni.

Egyéb esetben telepíteni kell a `docker-compose-plugin` csomagot is külön. Ekkor válik elérhetővé a szolgáltatás.

5.12.2 Parancs formája

Eredetileg a parancs neve `docker-composer`, de az új változatokban már beépült a docker alap környezetébe, ezért kötőjel nélkül (`docker compose`) is használhatók a parancsok. A régebbi ismertetők a `docker-compose` formát használják, de azok, akik most kezdenek ismerkedni, érdemes rögtön a kötőjel nélküli változatot alkalmazni.

5.12.3 Használatának lépései

- Meg kell határozni az alkalmazáshoz tartozó szolgáltatásokat (Web szerver/webapplikáció, PHP, adatbázis, stb), és meg kell tervezni, hogy hogyan tudják majd elérni egymás szolgáltatásait.
- `Dockerfile`-ok előállítása az egyes különálló szolgáltatásokhoz.
- Készíteni kell egy `.yaml` fájlt, amiben meghatározzuk az applikáció szolgáltatásait és meghatározzuk legfontosabb jellemzőiket.
- Ki kell adni a `docker compose up` (vagy `docker-compose up`) parancsot.
- A Docker ennek hatására elkészíti az alkalmazás futtatásához szükséges image-eket, és a yaml fájlban meghatározott módon elindítja a konténereket, majd figyeli a konténerek állapotát és szükség esetén újraindítja őket.

5.12.4 Egyszerű példa

A `Docker` dokumentációjában van egy egyszerű példa a `compose` használatára, ez a következő oldalon érhető el:

https://docs.docker.com/compose/gettingstarted/

Az ott leírtak alapján bárki össze tudja állítani az első komplex rendszerét.

Az oldal alján pedig a további ismerkedéshez is találunk segítséget (`Where to go next - Merre tovább`).

5.12.5 Egy példa YAML fájl

Ez a `Docker` oldalán elérhető minták között található (`Compose and Wordpress`) YAML fájl direkt másolata, ami azt mutatja meg, hogy a `MySQL (MariaDB)` adatbázis és a `WordPress`-t megjelenítő webszerver szolgáltatás külön image-be és így külön konténerbe kerül, de mivel összetartoznak, ezért egyszerűbb egy `YAML` fájlal egyszerre kezelni őket. A `docker-compose.yml` fájl tartalma:

```
services:
  db:
    # We use a mariadb image which supports both amd64 & arm64 architecture
    image: mariadb:10.6.4-focal
    # If you really want to use MySQL, uncomment the following line
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
```

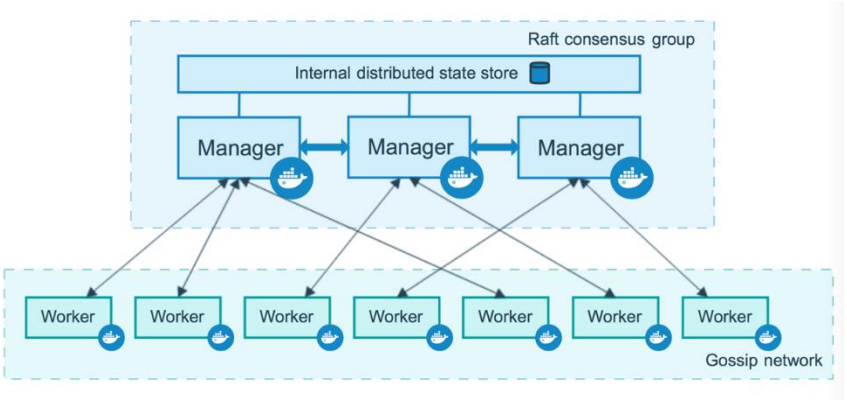
```
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:
```

Bővebben itt lehet róla olvasni:

https://docs.docker.com/samples/wordpress/

6 Docker Swarm

A `Docker Swarm` egy `orkesztrátor`, egy karmester, ami vezérli/kezeli/irányítja a docker konténereket, vagyis segítségével több docker konténereket kezelő gépet (`Worker Node`) lehet összekapcsolni, és dinamikusan menedzselni egy közös hálózaton belül. A gépek ilyen csoportosítását `raj`-nak (`swarm`) is nevezik. Ezen a ponton túl kell lépni az 1-2 gépes hálózatokon, itt akár egyszerre 100 gép kezeléséről is beszélhetünk, amelyek mindegyike több konténert is futtathat.



4. ábra Docker Swarm felépítése (Forrás Docker dok)

A klaszterben lévő gépek lehetnek önálló, fizikai gépek vagy akár virtuális gépek is, illetve ezek tetszőleges kombinációi.

A `Docker Swarm` a `Docker engine` része, vagyis nem kell külön telepíteni, elég csak magát a dockert.

Bővebben majd a Felhőszolgáltatásoknál lesz róla szó.

7 Kubernetes

A `Kubernetes` a másik docker orkesztrátor, amit széles körben használnak felhő rendszerek kialakításához. A `Kubernetest` a `Google` kezdte fejleszteni, amikor a `Docker Swarm` még sehol sem volt, később pedig nyílt forrásúvá tette. Mind a mai napig aktívan fejlesztik, és közben ipari kváziszabvánnyá vált.

Mit tud a `Kubernetes`, amit a `Docker Swarm` nem?

- monitoroz (`health check` a `Dockerben` is van, de nem tudjuk a konténer terhelését monitorozni)
- autoskáláz, azaz a terhelés változására a konténerek számának változásával reagál (szemben a `Docker` manuális skálázásával)
- pod-okban definiáljuk az összetartozó konténereket (hasonló dolgot jelentenek a `Docker Compose` értelmezte `YAML`-fájlok)
- (más konténerekkel is használható)
- a felhőszolgáltatók nyújtanak rá menedzselt megoldást

Mi szól a `Docker Swarm` mellett?

- egy ökoszisztéma, egy telepítendő rendszer
- egyszerűség
- nem kell új CLI-eszközt megtanulni