

TARTALOMJEGYZÉK

1. VM-ek és konténerek	1
1.1 Virtuális gép (VM)	1
1.2 Konténer	2
2. Docker	3
3. Linux (Debian) telepítése és konfigurálása	5
3.1 A Sudo beállítása.....	7
3.2 IP címzés konfigurálása.....	7
3.3 DNS konfigurálása	8
3.4 A „Guest Additions” kiegészítő telepítése	8
3.5 Időzóna konfigurálása	9
4. A Docker telepítése	9
5. A Docker Hub és használata	10
5.1 Keresés a Docker Hub-on.....	10
5.2 Kép (image) letöltése a Docker Hub-ról	11
5.3 Architektúrák	12
5.4 Tag-ek.....	13
6. Egyszerű Docker konténerek futtatása	13
6.1 Egyszerű „Hello, World!” konténer	13
6.2 Python interaktív konzol futtatása konténerben	14
6.3 Egyedi parancs futtatása konténerben	14
6.4 Java alkalmazás futtatása konténerben	14
6.5 Interaktív Bash indítása egy konténerben.....	16
6.6 Webszerver telepítése konténerbe	16
6.7 A konténerek és képek kezelése	18
7. Volume-ok.....	20
8. Környezeti változók (Environment Variables).....	21
9. A Docker és a Visual Studio Code összekapcsolása	23
9.1 Konténerek kezelése VSCode-ban	29
10. Szolgáltatások összekapcsolása (több konténer egyidejű futtatása).....	33
10.1 Joomla! CMS rendszer telepítése Docker konténerbe.....	33
11. A Docker Compose	38

11.1 YAML fájl létrehozása és konfigurálása (WordPress telepítése).....	38
11.2 A Docker Compose kezelése.....	43
12. A Dockerfile	44
12.1 A Dockerfile felépítése.....	45
12.2 Egyszerű Shell script futtatása, image készítése	47
13. Feladatok	50

A telepítések nél az operációs rendszer újabb verzióját is használhatjuk! Mindig ellenőrizzük, hogy a hivatalos letöltési oldalon van-e újabb megjelenés!

A segédletet a készítő engedélye és beleegyezése nélkül felhasználni és másolni szigorúan tilos!

1. VM-ek és konténerek

A konténerek és a VM-ek közötti különbség:

A virtuális gépek (VM) és a konténerek két népszerű technológia az informatikai rendszerek virtualizációjához, de eltérő módon működnek és különböző felhasználási célokra alkalmasak.

A legfőbb különbségeik az architektúrában, az erőforrás-kezelésben és a hordozhatóságban rejlenek.

1.1 Virtuális gép (VM)

Architektúra:

A virtuális gépek a fizikai hardvert teljes mértékben virtualizálják.

Minden VM saját operációs rendszert futtat (guest OS), amely egy hypervisor (például VirtualBox, VMware, Hyper-V, vagy KVM) segítségével fut a gazdarendszer felett.

Erőforrás-kezelés:

A VM-ek erőforrás-igénye általában nagyobb, mivel minden VM-nek szüksége van egy teljes operációs rendszerre, amely memória- és tárhelyigényteljes.

Nehezebben skálázhatók, mivel az operációs rendszer indítása időigényes lehet.

Hordozhatóság:

A VM-eket egyszerűen lehet más rendszerekre áthelyezni, de a nagy méretük miatt a folyamat lassabb.

Izoláció:

A VM-ek teljes izolációt biztosítanak, mivel minden VM-nek saját operációs rendszere és kernelje van, ami nagyobb biztonságot jelenthet.

Felhasználási terület:

Ideális meglévő rendszerek futtatására, különösen, ha eltérő operációs rendszerek szükségesek.

1.2 Konténer

Architektúra:

A konténerek az operációs rendszer szintjén működnek, és megosztják a gazdagép (host OS) kernelét.

Egy konténerben csak az alkalmazás és annak futtatásához szükséges függőségek találhatók, nincs saját operációs rendszerük.

Erőforrás-kezelés:

A konténerek könnyűsúlyúak, gyorsan indíthatók és kevesebb erőforrást igényelnek, mivel nem futtatnak teljes operációs rendszert.

Kiválóan skálázhatók, különösen dinamikus környezetekben, például mikroszolgáltatásoknál.

Hordozhatóság:

A konténerek is könnyen hordozhatók, mert ugyanúgy futtathatók a fejlesztői gépen, mint a gyártási környezetben, ha ugyanaz a konténer runtime (pl. Docker) érhető el.

Izoláció:

Kevésbé izoláltak, mivel osztóznak a gazdarendszer kernelén. Ez kisebb biztonságot nyújthat, de gyakran megfelelő izolációt biztosítanak a legtöbb alkalmazás számára.

Felhasználási terület:

Ideális modern, felhőalapú alkalmazásokhoz és mikroszolgáltatás-alapú architektúrákhoz.

Összehasonlítás:

Jellemző	Virtuális gép (VM)	Konténer
OS szükséglet	Saját operációs rendszer	Host OS-t használja
Erőforrás-igény	Magas	Alacsony
Indítási idő	Lassú	Gyors
Izoláció	Erős	Mérsékelt
Hordozhatóság	Nagy fájlméret miatt lassabb	Könnyű és gyors
Skáláhatóság	Kevésbé rugalmas	Nagyon rugalmas
Biztonság	Jobb izoláció miatt magasabb	Függ a host kernelétől

A választás a konkrét használati esettől függ: ha teljes izolációra és eltérő operációs rendszerekre van szükség, a VM lehet a jobb választás. Ha pedig gyors indítás, könnyű hordozhatóság és skálázhatóság a cél, a konténerek előnyösebbek.

2. Docker

A Docker egy nyílt forráskódú platform, amely lehetővé teszi az alkalmazások és azok függőségeinek konténerekbe csomagolását, futtatását és kezelését. A konténerek könnyűsúlyú, izolált környezetek, amelyek az alkalmazások futtatásához szükséges összes komponenssel rendelkeznek (pl. kód, könyvtárak, konfigurációk), és azonos módon működnek a fejlesztői gépen, a tesztelési környezetben és a gyártásban.

Főbb összetevői:

- **Docker Engine:** A konténerek futtatásáért, kezeléséért és létrehozásáért felelős motor.
- **Docker Daemon:** A háttérben futó szolgáltatás, amely a konténerek kezelését végzi.
- **CLI (Command Line Interface):** Egy parancssoros eszköz, amely a Docker műveletek végrehajtására szolgál.
- **Docker Images:** A konténerek sablonjai, amelyek tartalmazzák az alkalmazás kódját és függőségeit. Ezekből indulnak el a konténerek.
- **Docker Containers:** Az image-ek futó példányai, amelyek izolált környezetet biztosítanak az alkalmazások számára.
- **Docker Hub:** Egy felhőalapú regisztrációs szolgáltatás, ahol előre elkészített Docker image-eket lehet tárolni és megosztani.

Előnyei:

1. Hordozhatóság

A Docker konténerek azonos módon futnak minden platformon, amely támogatja a Docker Engine-t. Ez megkönnyíti az alkalmazások átvitelét a fejlesztői, tesztelési és gyártási környezetek között.

2. Gyors indítás

A konténerek könnyűsúlyúak, és nincs szükségük külön operációs rendszerre, így gyorsabban indíthatók, mint a virtuális gépek.

3. Erőforrás-hatékonyság

A konténerek megosztják a gazdarendszer kernelét, így kevesebb erőforrást fogyasztanak, mint a virtuális gépek.

4. Modularitás

A Docker támogatja az alkalmazások mikroszolgáltatás-alapú felépítését, amelyben az alkalmazás kisebb, különálló konténerekből áll, amelyek egy adott funkciót valósítanak meg. Ez javítja a skálázhatóságot és a hibák elkülönítését.

5. Könnyű skálázás

A Docker konténerek gyorsan klónozhatók és több példányban is futtathatók, ami ideális a nagy terhelésű alkalmazásokhoz.

6. Függetlenség a gazdarendszertől

A konténerekben futó alkalmazások függetlenek a gazdarendszer konfigurációjától, mivel minden szükséges függőséget tartalmaznak.

7. Egyszerű telepítés

A Docker használatával az alkalmazások és azok futtatási környezete "együtt csomagolható", így könnyen és gyorsan telepíthetők.

8. Széles körű támogatás

A Docker az iparági szabvány lett a konténerizáció terén, és számos nyílt forráskódú eszköz, például Kubernetes, támogatja a használatát.

Tipikus felhasználási esetei:

- *Fejlesztői környezetek beállítása:* Gyorsan és következetesen előállíthatók fejlesztői környezetek.
- *CI/CD (folyamatos integráció/folyamatos szállítás):* Automatizált tesztelési és telepítési folyamatok.
- *Mikroszolgáltatások:* Különálló szolgáltatások futtatása külön konténerekben.
- *Monolitikus alkalmazások modernizálása:* Régi alkalmazások becsomagolása konténerekbe a korszerűsítéshez.

- *Többkörnyezetű alkalmazásfejlesztés:* Azonos konfiguráció biztosítása a különböző környezetekben (pl. fejlesztés, tesztelés, gyártás).

Miért népszerű a Docker?

A Docker leegyszerűsíti az alkalmazások fejlesztését, tesztelését és telepítését, miközben minimalizálja a "működik a gépemen, de máshol nem" típusú problémákat. Ez az egyszerűség, rugalmasság és erőforrás-hatékonyság tette a Dockert a konténerizáció egyik legnépszerűbb eszközévé.

3. Linux (Debian) telepítése és konfigurálása

Hozzunk létre a VirtualBox-ban egy új virtuális gépet az alábbiak szerint:

Name: linux_docker

Type: Linux

Subtype: Debian

Version: Debian 12 Bookworm (64 bit)

Base Memory: 4GB

Processors: 2

A memória mennyisége és a CPU magok száma a gazdagépben lévő fizikai RAM mennyiségének és CPU magok számának függvénye!

Disk Size: 30 GB

A virtuális gép konfigurálása:

General/Advanced → Shared Clipboard: Bidirectional | Drag'n'Drop: Bidirectional

System/Motherboard → Boot Order: floppy-t vegyük ki a boot sorrendből

Storage: helyezzük be az optikai meghajtóba a Debian ISO-t, a vdi lemezképre kapcsoljuk be a *Solid-state Drive*-ot (amennyiben SSD-re telepítünk)

Network/Adapter 1: NAT

Network/Adapter 2: Host-only Adapter

A telepítés menete és paraméterei:

Install

Select a language | Language: *English*

Select your location | Country, territory or area: *United Kingdom*

Configure the keyboard | Keymap to use: *Hungarian*

Configure the network | Primary network interface: *enp0s3*

Configure the network | Hostname: *linuxdocker*

Configure the network | Domain name: (hagyjuk üresen, vagy törljük) → *Continue*

Set up users and passwords

Root password: *#Aa123456789@*

Full name for the new user: *LinuxDockerAdmin*

Username for your account: *linuxdockeradmin*

Choose a password for the new user: *#Bb123456789@*

Partition disks | Partitioning method → Manual

1. New partition size: 25 GB | Type: Primary | Location: Beginning | Use as: Ext4 | Mount point: / | Label: linuxdocker | Bootable flag: on
2. New partition size: 3.6 GB | Type: Logical | Location: Beginning | Use as: Ext4 | Mount point: /home | Label: home | Bootable flag: off
3. New partition size: 3.6 GB | Type: Logical | Use as: swap area | Bootable flag: off

Configure the package manager

Scan extra installation media? → *No*

Debian archive mirror country → *United Kingdom*

Debian archive mirror: *deb.debian.org*

HTTP proxy information (blank for none): hagyjuk üresen → *Continue*

Configuring popularity-contest | Participate in the package usage survey? → *No*

Software selection | Choose software to install:

- *SSH server*
- *standard system utilities*

Configuring grub-pc | Install the GRUB boot loader to your primary drive? → *YES*

Configuring grub-pc | Device for boot loader installation: */dev/sda*

Finish the installation → *Continue*

Az újraindulás után a **root** felhasználóval lépjünk be!

3.1 A Sudo beállítása

```
apt install sudo -y
```

```
usermod -aG sudo linuxdockeradmin
```

```
getent group sudo
```

A konfigurált, emelt joggal rendelkező felhasználóra való váltás:

```
su - linuxdockeradmin
```

3.2 IP címzés konfigurálása

```
sudo mv /etc/network/interfaces /etc/network/interfaces.backup
```

```
sudo systemctl enable systemd-networkd
```

```
sudo nano /etc/systemd/network/10-nat.network
```

Másoljuk az alábbi sorokat a fájlba:

```
[Match]
Name=enp0s3
```

```
[Network]
Address=10.0.2.15/24
Gateway=10.0.2.2
DNS=8.8.8.8
DNS=8.8.4.4
```

Mentsük a fájlt és lépjünk ki!

```
sudo nano /etc/systemd/network/20-host-only.network
```

Másoljuk az alábbi sorokat a fájlba:

```
[Match]
Name=enp0s8
```

```
[Network]
Address=192.168.56.99/24
```

Mentsük a fájlt és lépjünk ki!

```
sudo systemctl restart systemd-networkd
```

3.3 DNS konfigurálása

```
sudo apt update && sudo apt upgrade -y  
sudo apt install resolvconf -y  
sudo systemctl status resolvconf.service  
sudo systemctl start resolvconf.service  
sudo systemctl enable resolvconf.service
```

```
sudo nano /etc/resolvconf/resolv.conf.d/head
```

Másoljuk az alábbi sorokat a fájl végére:

```
nameserver 8.8.8.8  
nameserver 8.8.4.4
```

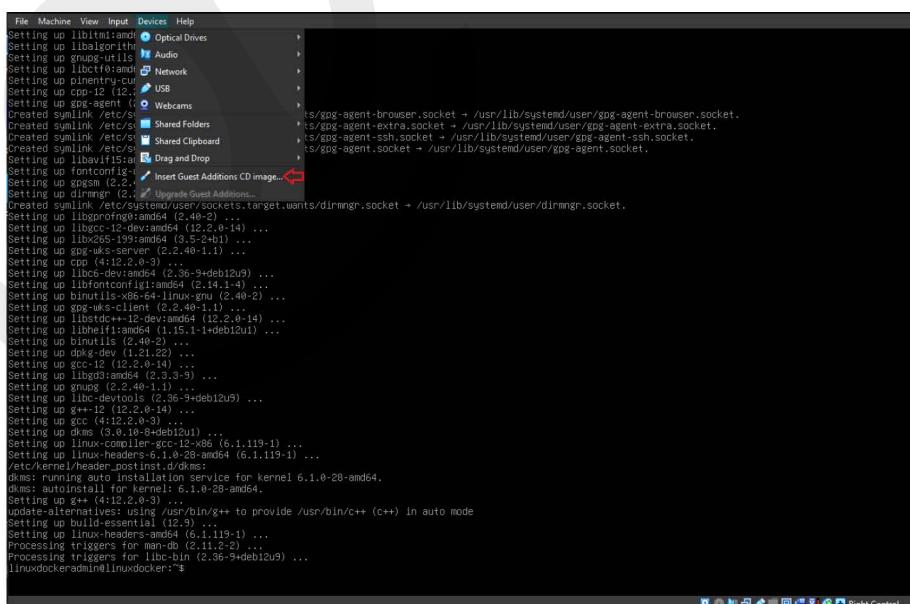
Mentsük a fájlt és lépjünk ki!

```
sudo resolvconf --enable-updates  
sudo resolvconf -u  
sudo systemctl reboot
```

Az újraindítás után jelentkezzünk vissza a **linuxdockeradmin** felhasználóval!

3.4 A „Guest Additions” kiegészítő telepítése

```
sudo apt install build-essential dkms linux-headers-$(uname -r) -y
```



csatlakoztassuk a „Guest Additions” CD image fájlt

```
sudo mkdir /mnt/cdrom  
sudo mount /dev/cdrom /mnt/cdrom  
cd /mnt/cdrom  
sudo sh ./VBoxLinuxAdditions.run --nox11  
sudo systemctl reboot
```

Az újraindítás után **Putty**-val jelentkezzünk vissza a **Host-only** kártya IP címét használva! A csatlakozás után a **linuxdockeradmin** felhasználóval lépjünk be!

3.5 Időzóna konfigurálása

```
sudo timedatectl set-timezone Europe/Budapest  
date
```

4. A Docker telepítése

```
sudo apt install ca-certificates curl gnupg lsb-release -y
```

Adjuk hozzá a Docker csomag GPG kulcsát a repository-hoz:

(A GPG-kulcs egy üzenetek és fájlok aláírására és titkosítására használható nyilvános kulcs.)

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o  
/usr/share/keyrings/docker-archive-keyring.gpg
```

Adjuk hozzá a Docker csomag repository-t a Debian-hoz:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-  
keyring.gpg] https://download.docker.com/linux/debian ${lsb_release -cs} stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt update
```

Telepítsük a Docker-CE csomagot a következő parancssal:

```
sudo apt install docker-ce docker-ce-cli containerd.io -y
```

Adjuk hozzá a felhasználót (linuxdockeradmin) a docker csoporthoz:

```
sudo usermod -aG docker $(whoami)  
sudo systemctl reboot
```

Az újraindítás után jelentkezzünk vissza Putty-val!

Ellenőrizzük a telepített Docker verzióját: `docker version`

5. A Docker Hub és használata

A Docker Hub egy nyilvános és privát tárolókat (repository) biztosító platform, amelyet a Docker fejlesztett ki. Ez a platform egy központi hely, ahol a fejlesztők tárolhatják, oszthatják meg és kezelhetik a konténerképeiket (Docker Images).

Alapvetően egy "konténer kép piactér", amely megkönnyíti az alkalmazások gyors telepítését és megosztását.

Mire használható a Docker Hub?

Kész képek (images) letöltése: Nyilvános tárolókban elérhetőek népszerű alkalmazások hivatalos Docker képei (pl. Nginx, MySQL, Python).

Saját képek megosztása: Feltölthetünk saját készítésű képeket, és megoszthatjuk azokat más fejlesztőkkel.

Automatikus képek építése: Kapcsolhatjuk a GitHub vagy Bitbucket tárolóinkhoz, hogy automatikusan létrehozza és frissítse a képeket.

Privát tárolók: Saját képek tárolása, amelyeket csak meghatározott személyek érhetnek el.

5.1 Keresés a Docker Hub-on

`docker search <kulcsszó>`

Pl.:

`docker search httpd`

A parancs kilistázza a httpd-hez kapcsolódó publikus képeket a Docker Hub-ról.

Hasznos opciók:

--filter: A keresési eredményeket szűkíti különböző szempontok alapján.

pl.:

Csak hivatalos képek keresése:

`docker search --filter "is-official=true" httpd`

Népszerű képek keresése (csillagok alapján):

`docker search --filter "stars=1000" httpd`

A parancs csak azokat a képeket mutatja, amelyeknek legalább 1000 csillagjuk van.

--limit: Meghatározza, hogy hány találatot jelenítsen meg.

Pl.:

`docker search --limit 5 httpd`

A parancs csak az első 5 találatot mutatja meg.

--no-trunc: A teljes leírást jeleníti meg (alapértelmezés szerint a hosszabb leírások rövidítve vannak)

Pl.:

`docker search --no-trunc httpd`

Mikor használjuk?

- Ha egy adott technológiához keresünk előre elkészített Docker image-et.
- Ha szeretnénk összehasonlítani az elérhető képek népszerűségét és megbízhatóságát (csillagok, hivatalos jelölés).
- Ha új image-eket szeretnénk felfedezni egy adott cérla.

A docker search egy gyors és egyszerű módja a Docker Hub felfedezésének!

5.2 Kép (image) letöltése a Docker Hub-ról

`docker pull <kép_neve>`

Pl.:

`docker pull httpd`

Ez a parancs letölti az httpd hivatalos Docker képét a Docker Hub-ról.

Saját képek feltöltése:

Hozzunk létre egy Dockerfile-t, majd készítsük el a képet:

`docker build -t <felhasználónév>/<kép_neve>:<verzió> .`

Pl.:

`docker build -t username/myapp:1.0 .`

Ha már meglévő képet szeretnénk feltölteni, tag-eljük:

```
docker tag <helyi_kép> <felhasználónév>/<kép_neve>:<verzió>
```

Töltsük fel a Docker Hubra:

```
docker push <felhasználónév>/<kép_neve>:<verzió>
```

Pl.:

```
docker push username/myapp:1.0
```

Privát tárolók kezelése:

A Docker Hub lehetővé teszi privát tárolók létrehozását, amelyekhez csak meghívott felhasználók férhetnek hozzá.

Létrehozás:

Lépjünk be a Docker Hub weboldalára, majd a Repositories menüben válasszuk a "Create Repository" lehetőséget, és állítsuk a tárolót "Private"-ra.

Használunk hivatalos képeket, amikor csak lehetséges, mivel ezek biztonságosabbak és jól karbantartottak!

Ha nagyobb projekten dolgozunk, érdemes verziószámokat hozzáadni a képeinkhez (pl. 1.0, latest stb.).

5.3 Architektúrák

A Docker Hub lehetővé teszi, hogy különböző CPU-architektúráakra készült image-eket használunk (pl. amd64, arm64, arm/v7). Ez fontos, ha különböző hardvereken futtatjuk a konténereket.

Ellenőrzés:

Az image támogatott architektúráit az adott image oldalán találjuk:

Menjünk a Docker Hub-on az image-re.

Nézzük meg a "Supported architectures" részt.

A httpd pl. támogatja az alábbi architektúrákat:

amd64, arm32v5, arm32v6, arm32v7, arm64v8, i386, mips64le, ppc64le, riscv64, s390x

A Docker automatikusan letölti a megfelelő architektúrájú image-et, amely megfelel a rendszerünknek.

pl.:

```
docker pull httpd
```

Ha konkrét architektúrát akarunk kiválasztani:

```
docker pull --platform linux/amd64 httpd
```

5.4 Tag-ek

A Docker image-ek különböző verzióit a tagek segítségével kezelhetjük. Ezek általában verziószámokat vagy környezeti specifikációkat jelölnek.

Pl.:

httpd image:

httpd:2.4.62 - A httpd 2.4.62 verzió.

httpd:2.4.62:alpine - Kis méretű, Alpine Linux alapú image.

httpd:2.4.62-bookworm - Debian 12 Bookworm image.

A Docker Hub-on, az image oldalán a „Tags” fülön találjuk az összes elérhető tag-et.

Ha pl. egy konkrét tag-et szeretnénk letölteni:

```
docker pull httpd:2.4.62-bookworm
```

Ha nem adunk meg taget, a **latest** (az alapértelmezett tag) kerül letöltésre.

A Docker image-ek gyakran támogatják környezeti változók beállítását, hogy testreszabassuk a konténer működését.

6. Egyszerű Docker konténerek futtatása

6.1 Egyszerű „Hello, World!” konténer

```
docker pull hello-world:latest
```

A **docker pull** parancssal letöljük a **hello-world** image-et. A hello-world kép egy egyszerű alkalmazást tartalmaz, amely üzenetet ír ki, ha sikeresen lefut.

A **:latest** tag az image legutóbbi (legfrissebb) verzióját jelöli. Elhagyható, abban az esetben is a legfrissebb verzió kerül letöltésre.

```
docker run --name helloworld-app hello-world
```

A **docker run** parancs elindítja a letöltött képfájlt.

A **--name** kapcsolóval nevet tudunk adni a konténernek.

A parancs végrehajtása után a konténer kiírja, hogy a Docker sikeresen működik, és példát ad a konténerek működésére.

6.2 Python interaktív konzol futtatása konténerben

```
docker run --name python-console -it python:latest
```

Ez a parancs egy **interaktív** Python konzolt nyit a legfrissebb Python képből.

Ha nincs letöltve a Python image, a parancs letölти a legfrissebbet!

Írunk Python kódot, például:

```
print("Hello from Docker!")
```

Kilépés: Ctrl+D, vagy írjuk be az **exit()** parancsot.

6.3 Egyedi parancs futtatása konténerben

```
docker run --name busybox-app busybox:latest ping -c 4 google.com
```

A **BusyBox** egy olyan program, amely egy futtatható bináris fájlban valósítja meg a Unix rendszerekből ismert egyszerűsített parancsok használatának lehetőségét.

A **-c 4** opcióval meghatározhatjuk, hogy 4 ICMP kérést küldjön.

6.4 Java alkalmazás futtatása konténerben

Először létre kell hoznunk a Java alkalmazásunkat, amit a konténerben fogunk futtatni:

```
sudo apt update && sudo apt upgrade  
sudo apt install openjdk-17-jdk -y  
mkdir docker  
mkdir docker/javaapp  
cd docker/javaapp/
```

Hozzuk létre az alábbi fájlt:

```
nano main.java
```

Másoljuk a fájlba az alábbi kódot:

```
public class main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

Mentsük a fájlt és lépjünk ki!

A javac eszközzel fordítsuk le a .java fájlt:

```
javac main.java
```

Ez létrehoz egy **main.class** fájlt a könyvtárban.

Készítsünk egy MANIFEST.MF fájlt:

A **.jar** fájlhoz szükség van egy MANIFEST.MF fájlra, amely megadja az induló osztály nevét.

```
echo "Main-Class: main" > MANIFEST.MF
```

A Main-Class értéke annak az osztálynak a neve, amely tartalmazza a main metódust.

Használjuk a jar eszközt a .jar fájl létrehozásához:

```
jar cfm helloworld.jar MANIFEST.MF main.class
```

c - új .jar fájl készítése.

f - az eredményfájlt a helloworld.jar-ba írja.

m - hozzáadja a MANIFEST.MF fájlt.

Ez létrehozza a helloworld.jar fájlt.

Teszteljük a .jar fájlt:

```
java -jar helloworld.jar
```

Futtassuk a Java alkalmazásunkat egy OpenJDK konténerben:

```
docker run --name helloworld-java -v $(pwd)/helloworld.jar:/app/helloworld.jar openjdk:17  
java -jar /app/helloworld.jar
```

- Az **openjdk:17** kép egy Java környezetet biztosít.
- A **-v** opcióval a helyi helloworld.jar fájlt csatoljuk a konténerbe.
- A **java -jar** parancssal futtatjuk a Java alkalmazást.

6.5 Interaktív Bash indítása egy konténerben

```
docker run --name intbash -it ubuntu:latest bash
```

Ez a parancs elindít egy Ubuntu alapú konténert, nevet ad neki, és egy interaktív Bash shell-t biztosít a konténeren belül. Adjuk ki az **exit** parancsot a kilépéshez!

A konténer újraindítása leállítás után:

```
docker start -ai intbash
```

6.6 Webszerver telepítése konténerbe

Töltsük le a httpd (Apache2 webkiszolgáló) Docker image fájlt:

```
docker pull httpd:latest
```

```
docker run -d --name staticwebsite -v $HOME/docker/apache2-static:/usr/local/apache2/htdocs -p 8080:80 httpd
```

Az új konténerünk létrejött!

Elemezzük a fenti parancsot:

```
docker run
```

A docker run parancs elindít egy új konténert az adott Docker image-ből (ebben az esetben az httpd-ből).

```
-d
```

Detached mód: A konténer a háttérben fut, így a terminálunk nem lesz lefoglalva.

```
--name staticwebsite
```

Ez a paraméter a konténer nevét adja meg, ebben az esetben: **staticwebsite**. Ha nem adunk meg nevet, a Docker véletlenszerű nevet generál.

```
-v $HOME/docker/apache2-static:/usr/local/apache2/htdocs
```

Kötet csatolása:

- A -v opció egy helyi mappát (\$HOME/docker/apache2-static) csatol a konténeren belüli útvonalhoz (/usr/local/apache2/htdocs). Ha nem létezik ez a mappa, a parancs létrehozza.
- Ebben az esetben a helyi gép \$HOME/docker/apache2-static könyvtára tartalmazza a statikus weboldal fájlokat (HTML, CSS, JavaScript stb.).

- A konténer belülről ezekhez a fájlokhoz az Apache szerver által kiszolgált tartalomként fér hozzá.

```
-p 8080:80
```

Porttovábbítás:

- A helyi gép 8080-as portját a konténer 80-as portjára irányítja át.
- A gazdagép böngészőjében a **http://192.168.56.99:8080** URL-en érjük el az Apache szerver által kiszolgált weboldalt.

```
httpd
```

Ez a használt Docker image neve, amely az Apache HTTP szerver hivatalos képe.

A parancs elindít egy Apache HTTP szervert tartalmazó konténert, amely a helyi fájlokat a megadott mappából (\$HOME/docker/apache2-static/) szolgálja ki.

Hozunk létre egy egyszerű weboldalt a **/home/linuxdockeradmin/docker/apache2-static/** nevű mappában:

```
sudo nano ~/docker/apache2-static/index.html
```

Másoljuk az alábbi sorokat a fájlba:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Welcome to Apache2</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is a simple static HTML site served by Apache2.</p>
</body>
</html>
```

Mentsük és zártuk be a fájlt!

A linuxdocker virtuális gép böngészőjében így érhetjük el a weboldalt:

A gazdagépről **http://192.168.56.99:8080** címmel érjük el a weboldalt!

6.7 A konténerek és képek kezelése

A letöltött image fájlok listázása:

`docker images`

A létrehozott konténerek listázása:

`docker ps -a`

`docker ps`: Alapértelmezés szerint csak az éppen futó konténereket listázza ki.

-a (vagy --all) opció: Az összes konténert megjeleníti, beleértve azokat is, amelyek leálltak vagy kiléptek.

A kimenet általában a következő oszlopokat tartalmazza:

CONTAINER ID: A konténer egyedi azonosítója.

IMAGE: A konténer létrehozásához használt Docker image neve.

COMMAND: A konténerben futó parancs.

CREATED: Mikor lett a konténer létrehozva.

STATUS: A konténer aktuális állapota (pl. Up, Exited, stb.).

PORTS: A konténer által használt portok.

NAMES: A konténer neve.

Mire használható?

- *Leállt konténerek azonosítása:* Azokat a konténereket is mutatja, amelyek már nem futnak.
- *Hibaelhárítás:* Információt ad arról, hogy egy konténer miért állt le (pl. kilépési kódok).
- *Konténerek törlése:* A nem szükséges, leállt konténereket könnyebben megtalálhatjuk és törölhetjük (pl. `docker rm <CONTAINER ID>`).

Ez a parancs kulcsfontosságú, ha egy Docker-környezetet menedzselni vagy hibaelhárítani szeretnénk!

A futó konténerek listázása:

`docker container ls`

Ez a parancs alapértelmezés szerint csak a futó konténereket jeleníti meg. Nem mutatja azokat a konténereket, amelyek leálltak, vagy kiléptek.

A futó konténer leállítása:

```
docker container stop <futó konténer neve>
```

Pl.:

```
docker container stop staticwebsite
```

A konténer neve az előző parancs "NAMES" oszlopában lévő név.

Konténer átnevezése:

```
docker container rename <jelenlegi konténernév új konténernév>
```

Pl.:

```
docker container rename staticwebsite staticwebsite-apache2
```

A leállított konténer újraindítása:

```
docker container start <konténer neve>
```

```
docker container start staticwebsite-apache2
```

Futó konténer újraindítása:

```
docker container restart <konténer neve>
```

pl.:

```
docker container restart staticwebsite-apache2
```

Konténer(ek) törlése:

A konténer és az image törlését csak akkor hajtsuk végre, ha ténylegesen szeretnénk a konténert és az image fájlt törölni!

```
docker container rm -f staticwebsite-apache2
```

A -f (force) kapcsoló a konténer kényszerített törlésére szolgál.

Összes konténer törlése:

```
docker container rm -f $(docker ps -aq)
```

docker ps -aq:

Ez a parancs kilistázza az összes konténer azonosítóját, beleértve a futó és leállt konténereket is. Az -q opció (quiet) csak az azonosítókat adja vissza, nem jeleníti meg a teljes részleteket.

\$(...):

A parancsbehelyettesítés segítségével a docker ps -aq kimenetét adja át argumentumként a docker container rm -f parancsnak. Gyakorlatilag a docker ps -aq által visszaadott konténerazonosítókat beilleszti a docker container rm -f parancsba.

docker container rm -f:

Ez kényszerítve törli a megadott konténereket, beleértve a futókat is. Az -f opció automatikusan leállítja a futó konténereket a törlés előtt.

Összességében a parancs minden konténert töröl a rendszerből, függetlenül attól, hogy azok futnak, leálltak, vagy kilépett állapotban vannak.

Image fájl(ok) törlése:

```
docker image rmi -f <image neve>
```

pl.:

```
docker image rmi -f httpd:latest
```

Átmenetileg állítsuk le a staticwebsite-apache2 nevű konténert!

7. Volume-ok

A Docker volume egy Docker által kezelt tárolómechanizmus, amely lehetővé teszi, hogy tartósan tároljunk adatokat a konténereken kívül. A volume-ok az adatokat megőrzik akkor is, ha a konténer leáll vagy újraépül, így ideálisak adatbázisok, fájlrendszerben tárolt adatok vagy bármilyen más állandó adatok kezelésére.

1. Volume létrehozása:

```
docker volume create my_volume
```

2. Volume csatlakoztatása egy konténerhez:

```
docker run -v my_volume:/data <képfájl neve>
```

A docker run parancs segítségével csatlakoztathatjuk a volume-ot egy konténerhez.

- my_volume a volume neve.

- /data az útvonal a konténer fájlrendszerében, ahol a volume elérhető lesz.

3. Adatok megosztása több konténer között:

```
docker run -v my_volume:/shared_data <képfájl1 neve>
docker run -v my_volume:/shared_data <képfájl2 neve>
```

Több konténer is használhatja ugyanazt a volume-ot, így az adatok könnyen megoszthatók.

4. Volume-ok kezelése:

Listázás:

```
docker volume ls
```

Törlés:

```
docker volume rm my_volume
```

Használaton kívüli volume-ok törlése:

```
docker volume prune
```

Mikor használjuk a volume-okat?

- Ha adatokat szeretnénk tartósan tárolni a konténereken kívül.
- Amikor az adatoknak túl kell elnie a konténer leállítását, törlését vagy újraépítését.
- Több konténer közötti adatmegosztáshoz.
- Olyan esetekben, amikor a gazdagép fájlrendszerének absztrakcióját akarjuk használni.

A Docker volume-ok egyszerű és hatékony módot nyújtanak az adatok kezelésére konténeres környezetben.

8. Környezeti változók (Environment Variables)

A Docker környezeti változók (environment variables, env) olyan változók, amelyek értékei dinamikusan beállíthatók, és a konténer futtatása során a konténer környezetében elérhetők. Ezeket az alkalmazások futtatásához szükséges konfigurációk (pl. adatbázis URL, jelszó, portok) meghatározására használják.

Mire jók a környezeti változók?

- **Konfigurálhatóság:** Alkalmazásaink futási paramétereit egyszerűen módosíthatjuk, anélkül, hogy újra kellene építeni az image-et.
- **Biztonság:** Érzékeny információk (pl. jelszavak) kezelésére is alkalmasak, bár ezt óvatosan kell kezelni.
- **Rugalmasság:** Ugyanaz az image eltérő környezetekben (pl. fejlesztés, teszt, éles) más-más konfigurációkkal futtatható.

pl.:

MySQL image:

A mysql image-nél néhány hasznos környezeti változó:

MYSQL_ROOT_PASSWORD: Az admin felhasználó jelszava.

MYSQL_DATABASE: Az inicializálálandó adatbázis neve.

MYSQL_USER és MYSQL_PASSWORD: Egy új felhasználó létrehozása és annak jelszava.

Környezeti változók használata:

Beállítás futtatáskor:

A docker run parancsnál az -e opcióval adhatunk meg környezeti változókat:

pl.:

```
docker run -e DATABASE_URL=mysql://user:password@host/db <képfájl neve>
```

Környezeti változók fájlból:

Ha több környezeti változót szeretnénk kezelní, használhatunk egy .env fájlt:

A .env fájl tartalma:

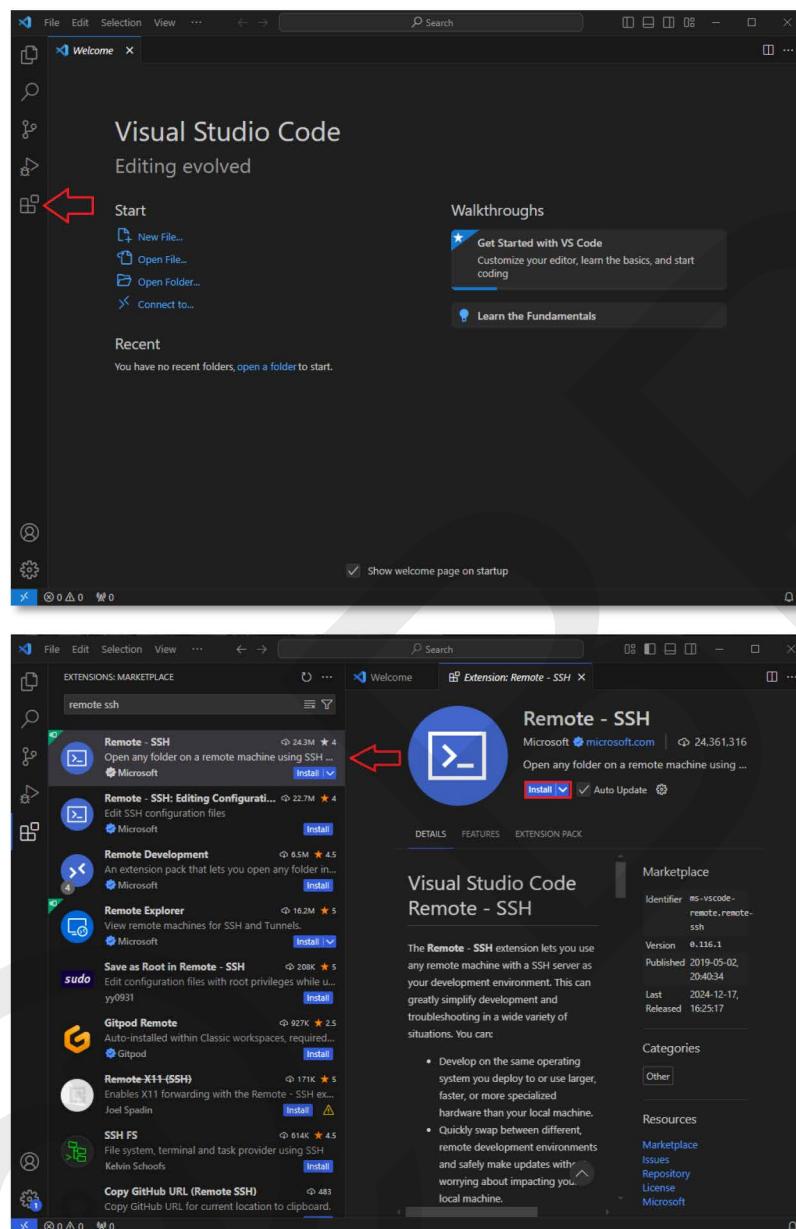
```
DATABASE_URL=mysql://user:password@host/db  
API_KEY=12345
```

Használat futtatáskor:

```
docker run --env-file .env <képfájl neve>
```

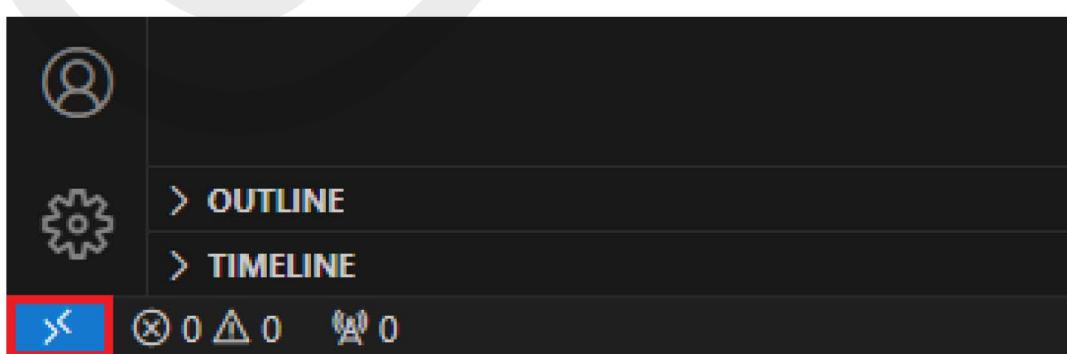
A környezeti változók egyszerű, mégis erőteljes eszközt kínálnak a konténerizált alkalmazások konfigurálására és testreszabására.

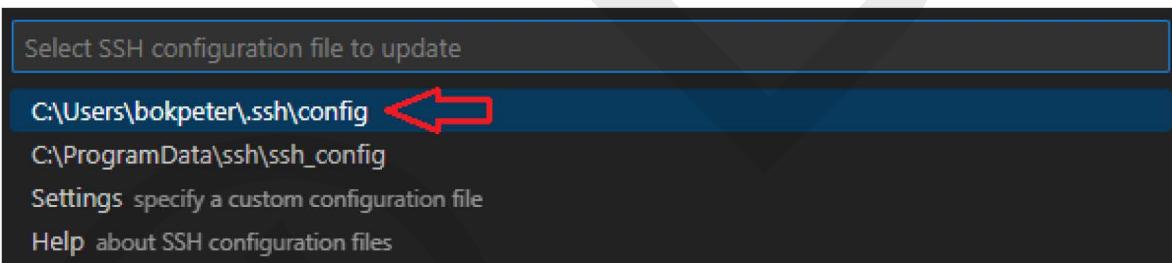
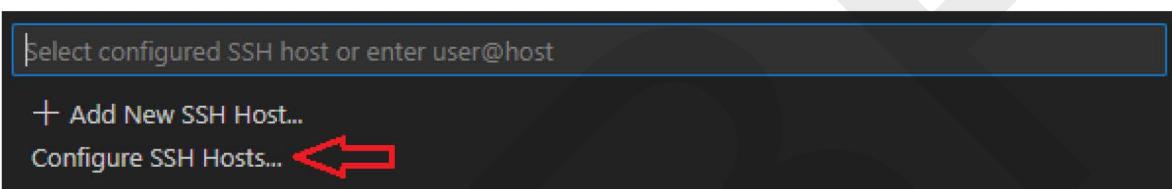
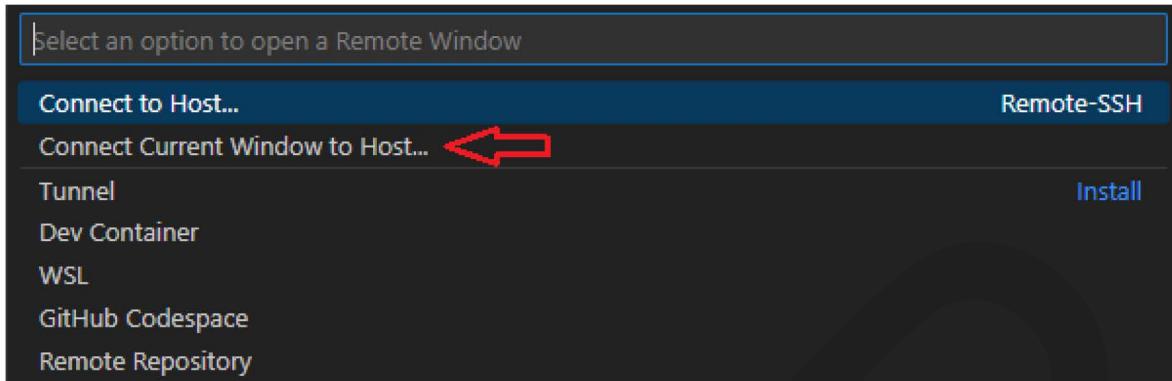
9. A Docker és a Visual Studio Code összekapcsolása



töltsük le és telepítsük VSCode-ban a Remote-SSH beépülő modult

A Remote-SSH beépülő modul konfigurálása és csatlakozás a távoli számítógéphez:

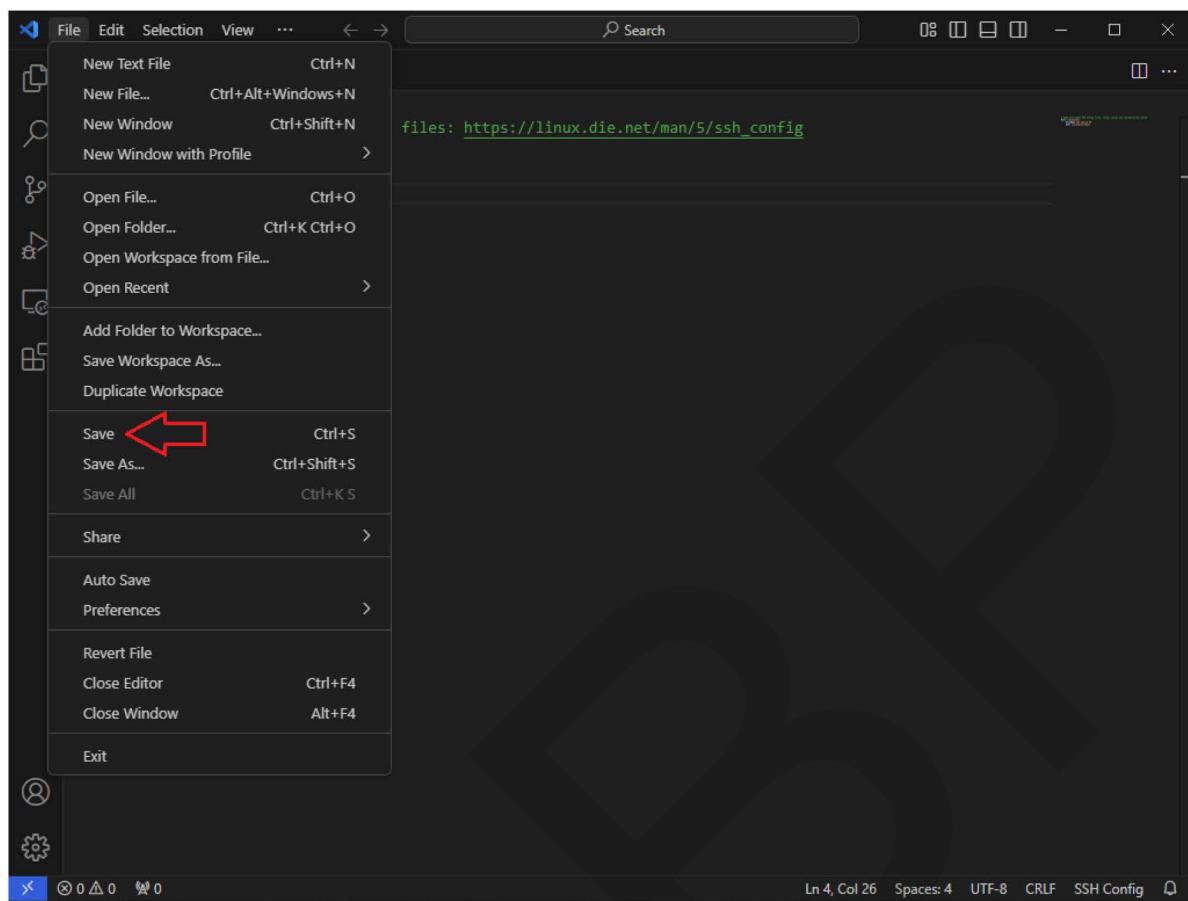




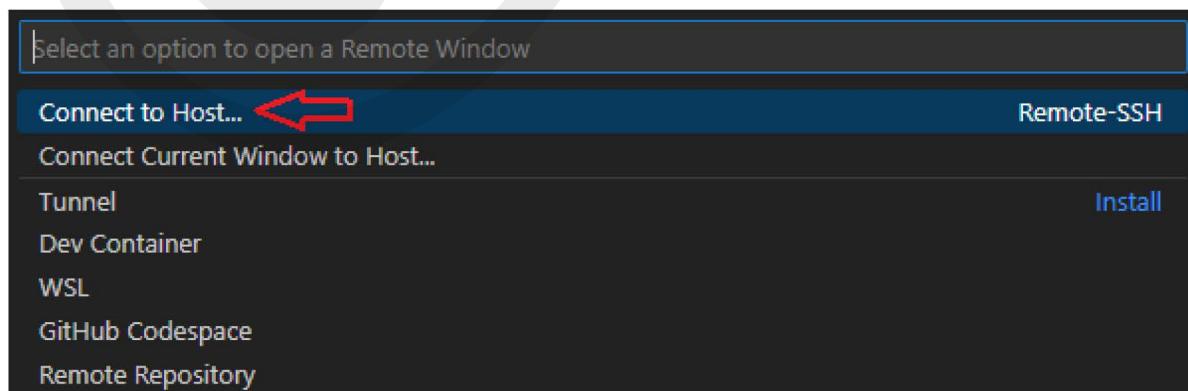
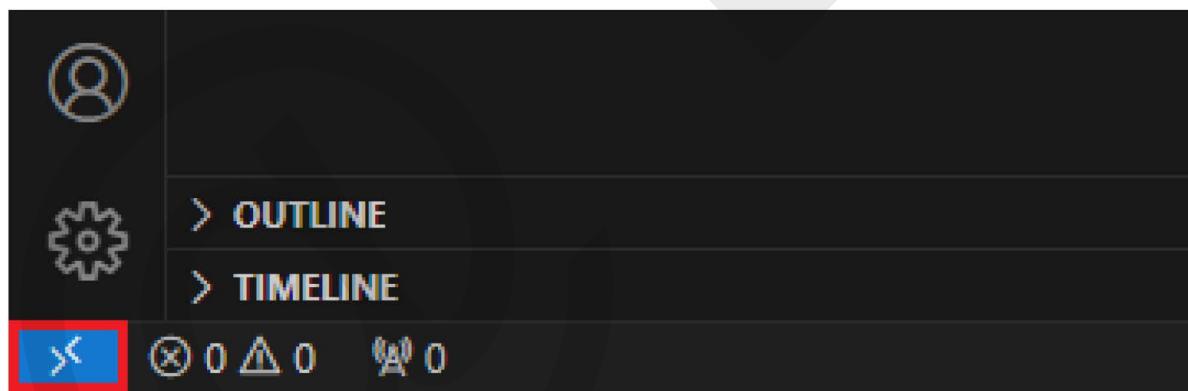
The screenshot shows a code editor window with the file 'config' open. The file contains the following content:

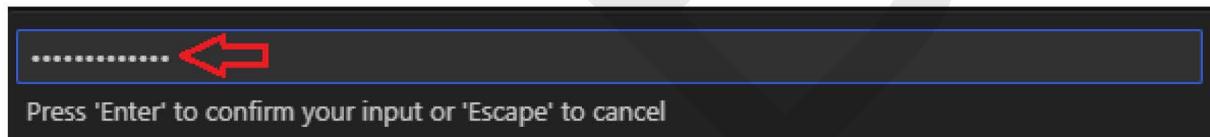
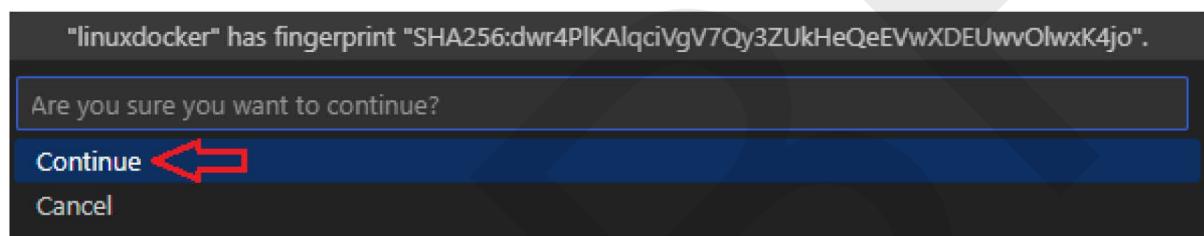
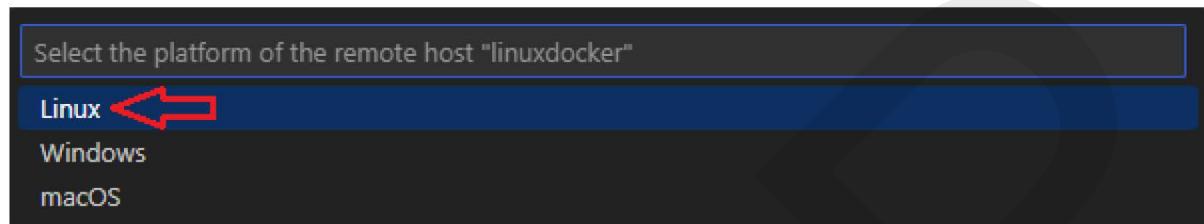
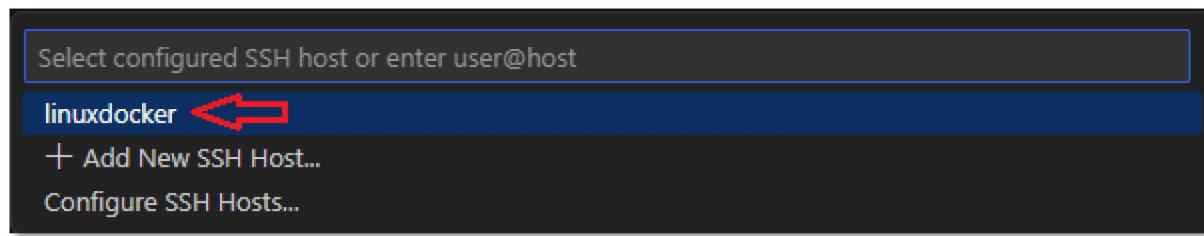
```
C: > Users > bokpeter > .ssh > config
1 # Read more about SSH config files: https://linux.die.net/man/5/ssh\_config
2 Host linuxdocker
3   HostName 192.168.56.99
4   User linuxdockeradmin
```

Host: **linuxdocker**
HostName: **192.168.56.99**
User: **linuxdockeradmin**

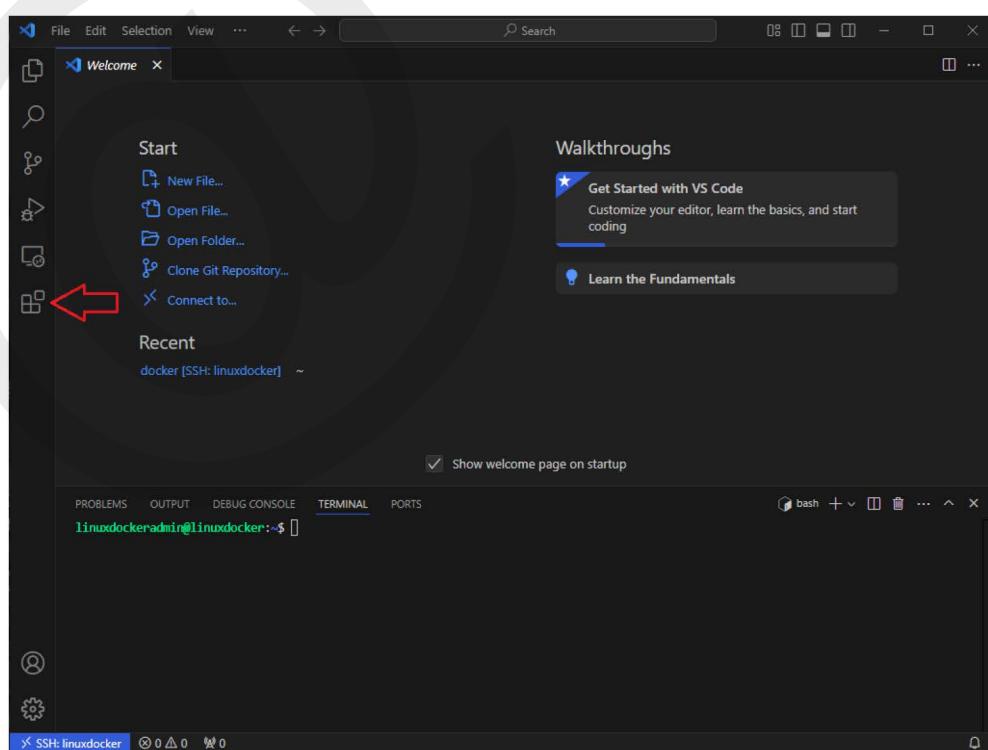


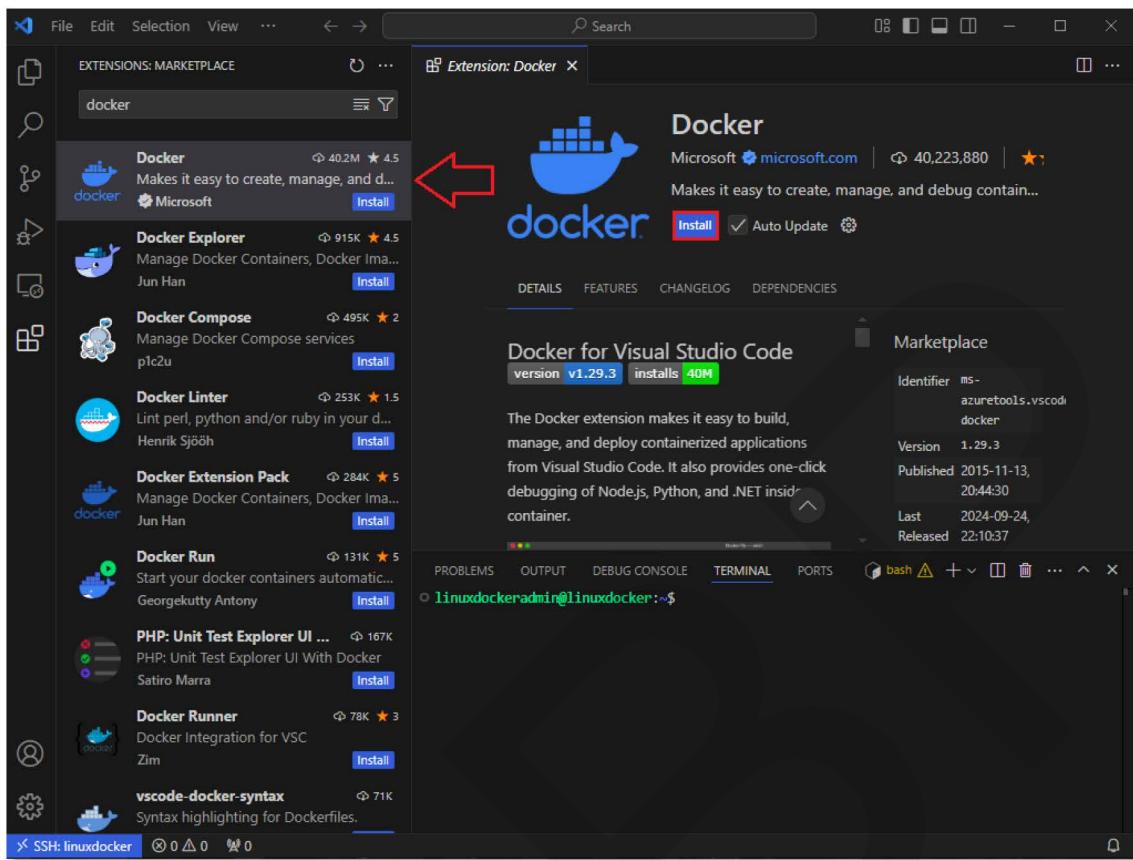
mentsük el a konfigurációt





jelszó: #Bb123456789@





telepítsük a Docker modult

Getting Started with Docker

Learn about Docker and the Docker extension for Visual Studio Code

Open a Workspace Folder

Open a workspace folder to get started with Docker extension features.

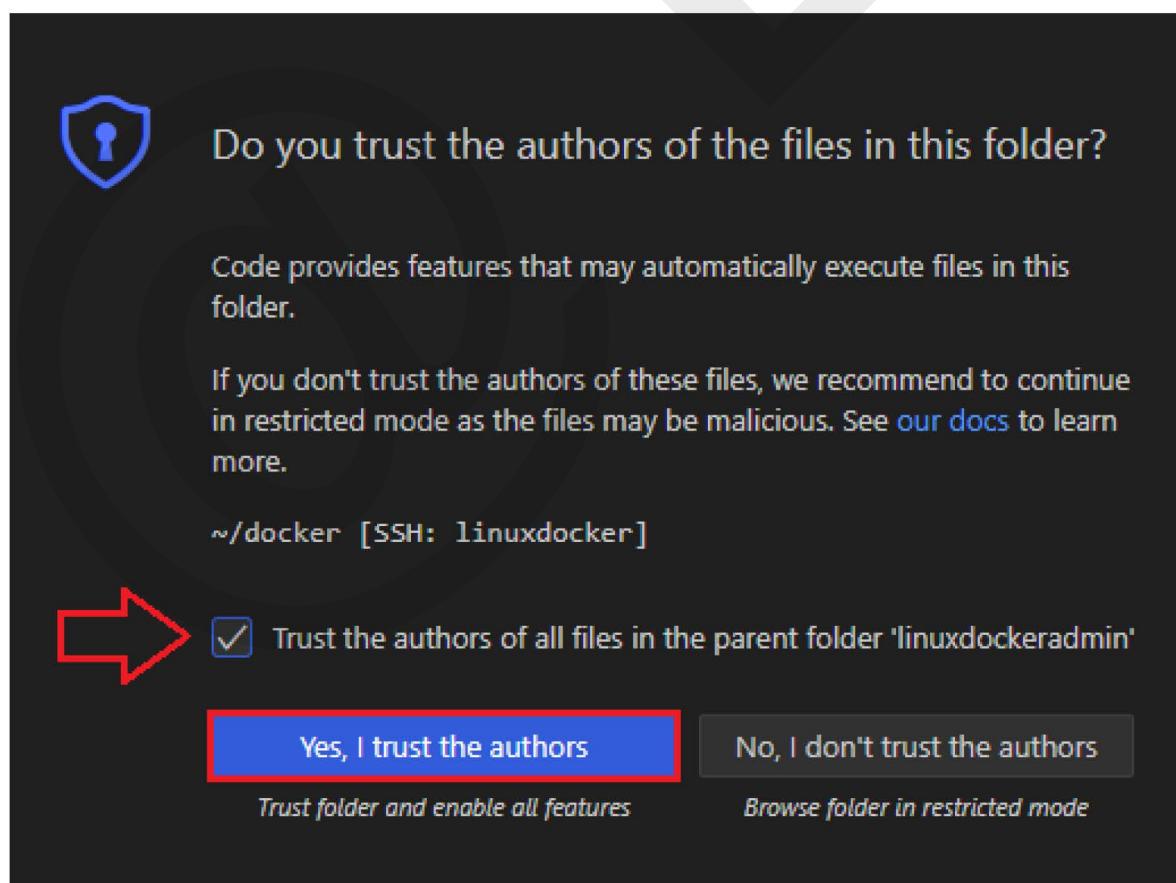
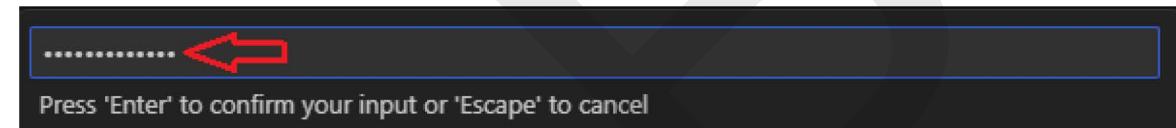
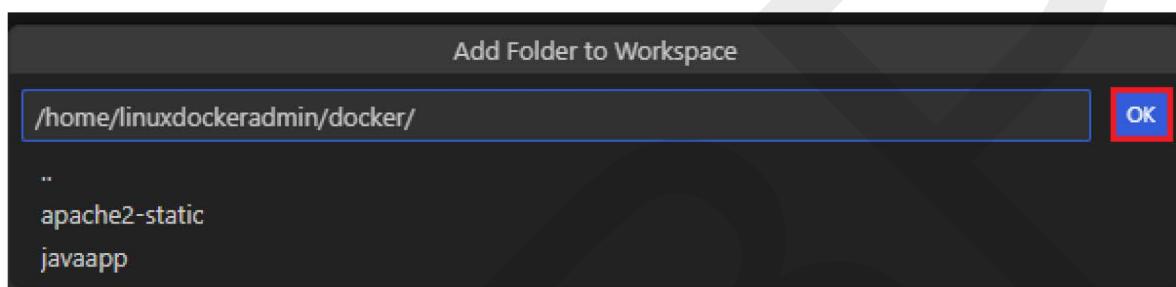
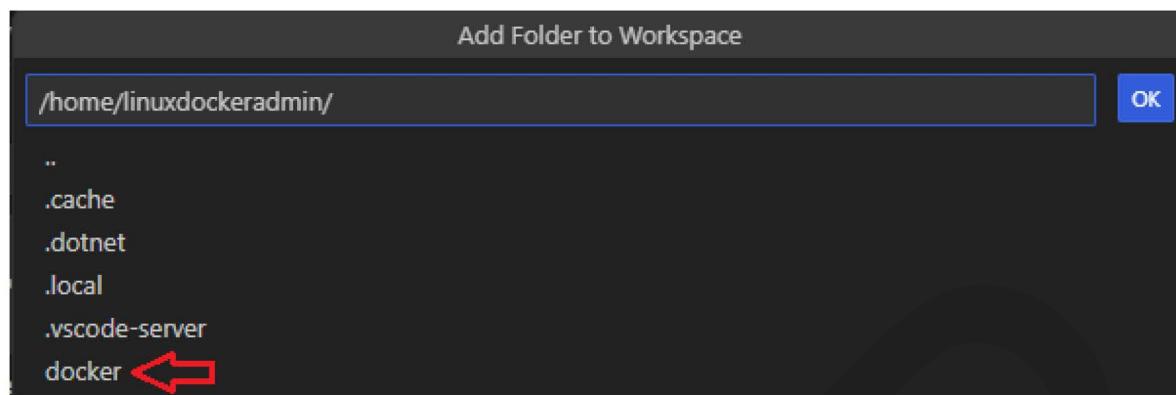
Open Folder

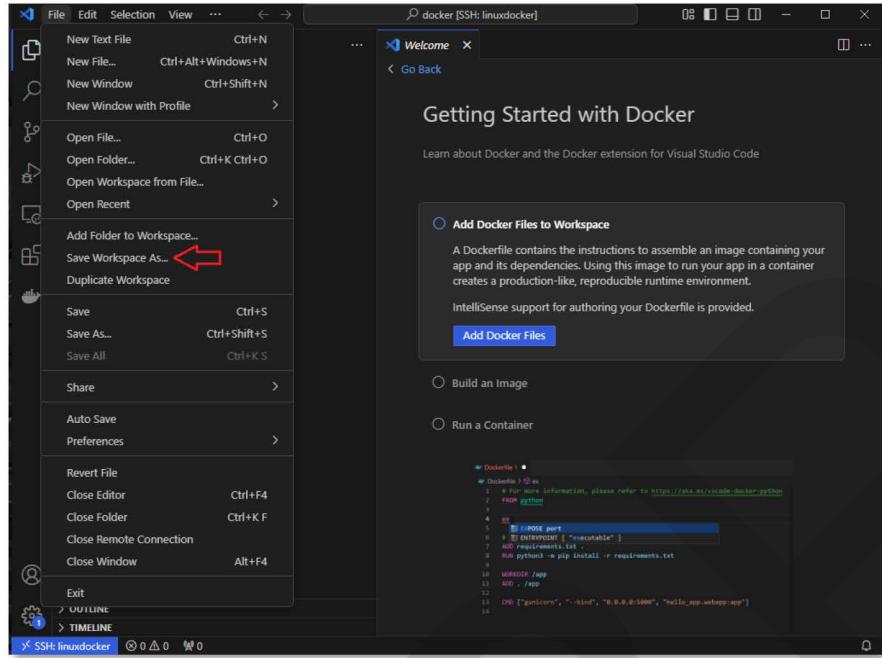
Tip: Use keyboard shortcut **Ctrl + K** **Ctrl + O**

Run a Container

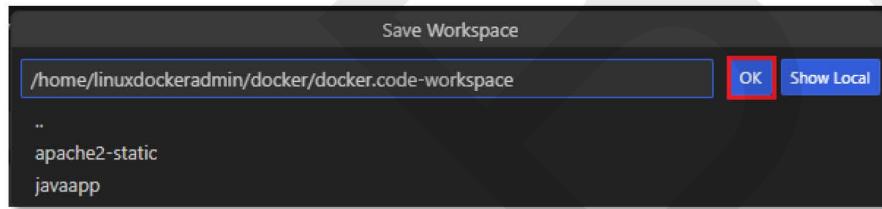
Use the Docker Explorer

válasszuk ki a munkamappánkat



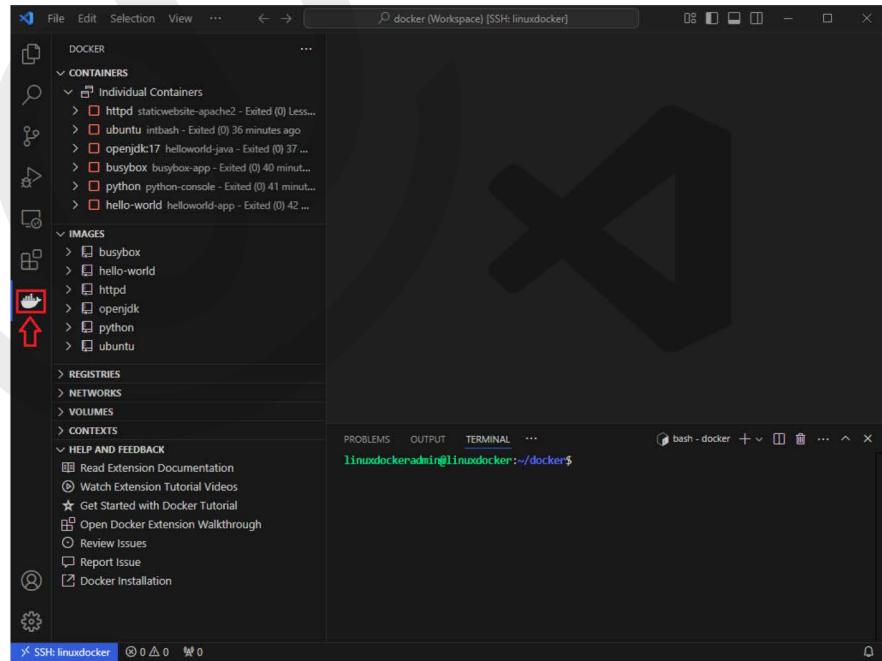


mentsük el a munkamappánkat



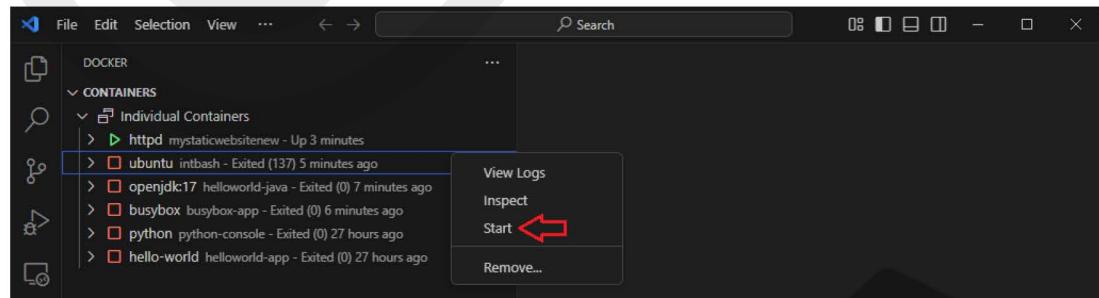
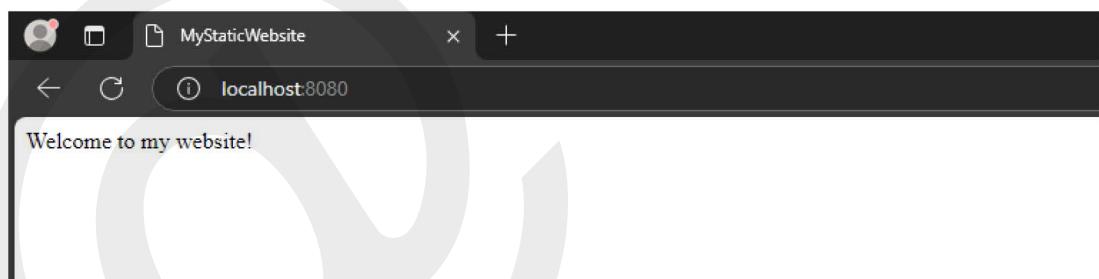
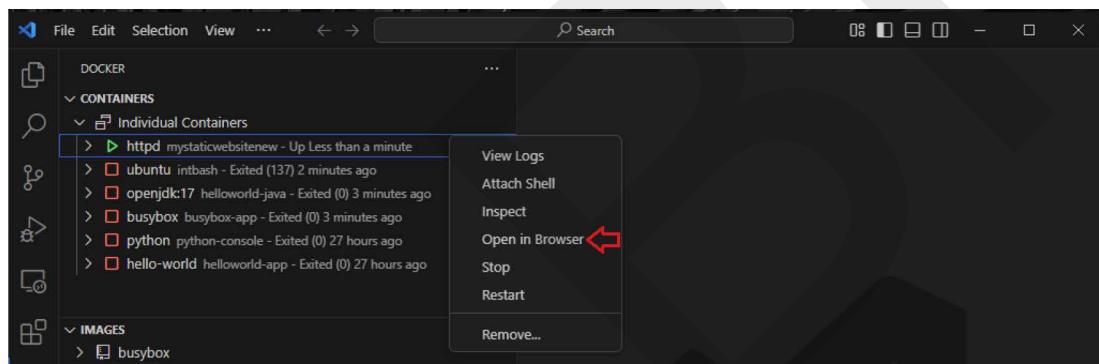
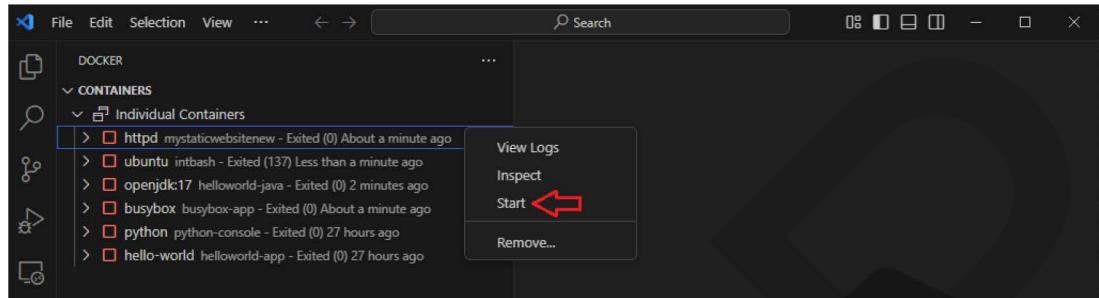
ha az OK után megszakad a kapcsolat, csatlakozzunk újra

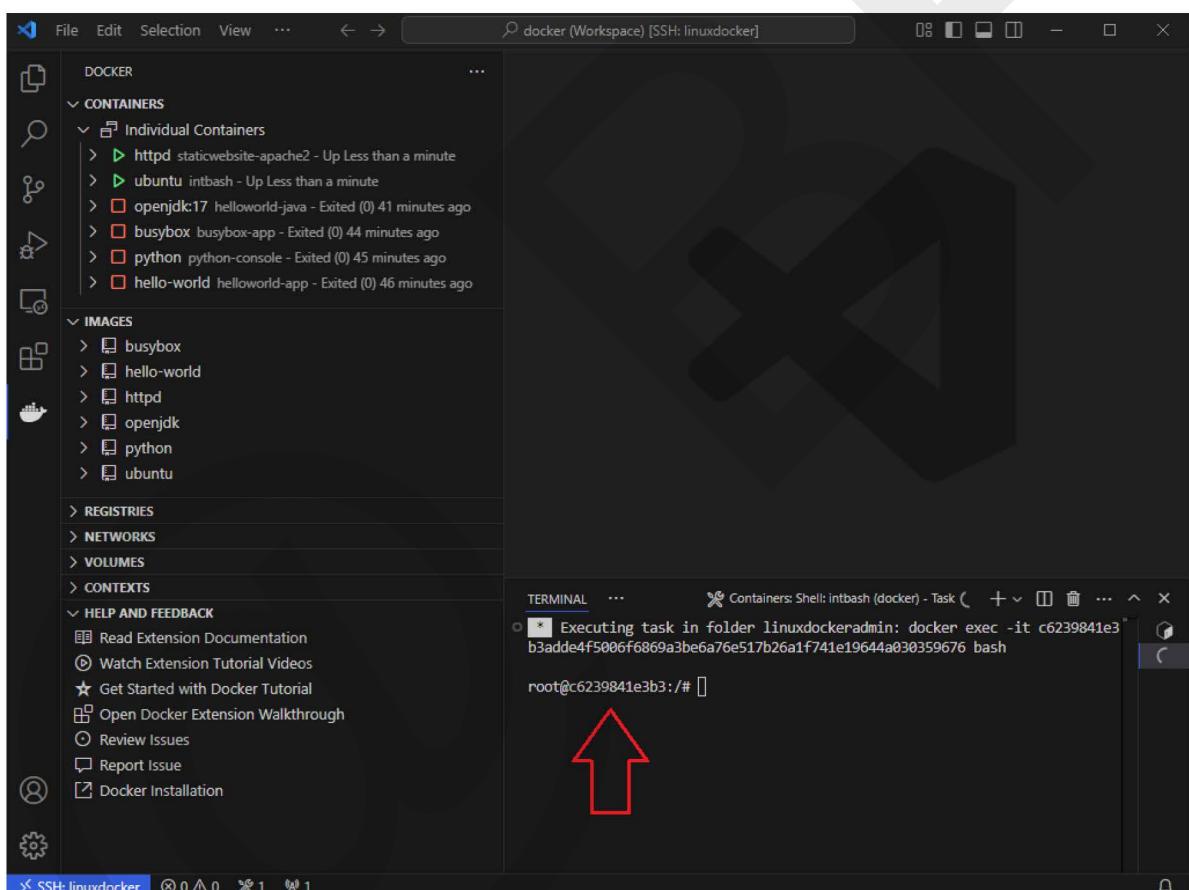
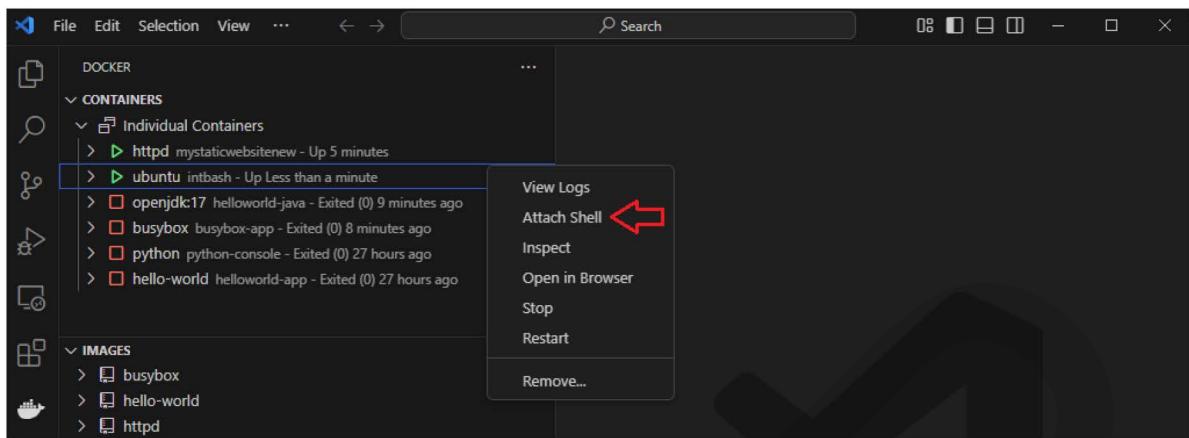
9.1 Konténerek kezelése VSCode-ban



Bal oldalon a Docker ikonjára kattintva kezelhetjük a konténereket, képeket, hálózatokat, köteteket stb.

Indítsunk el konténereket:

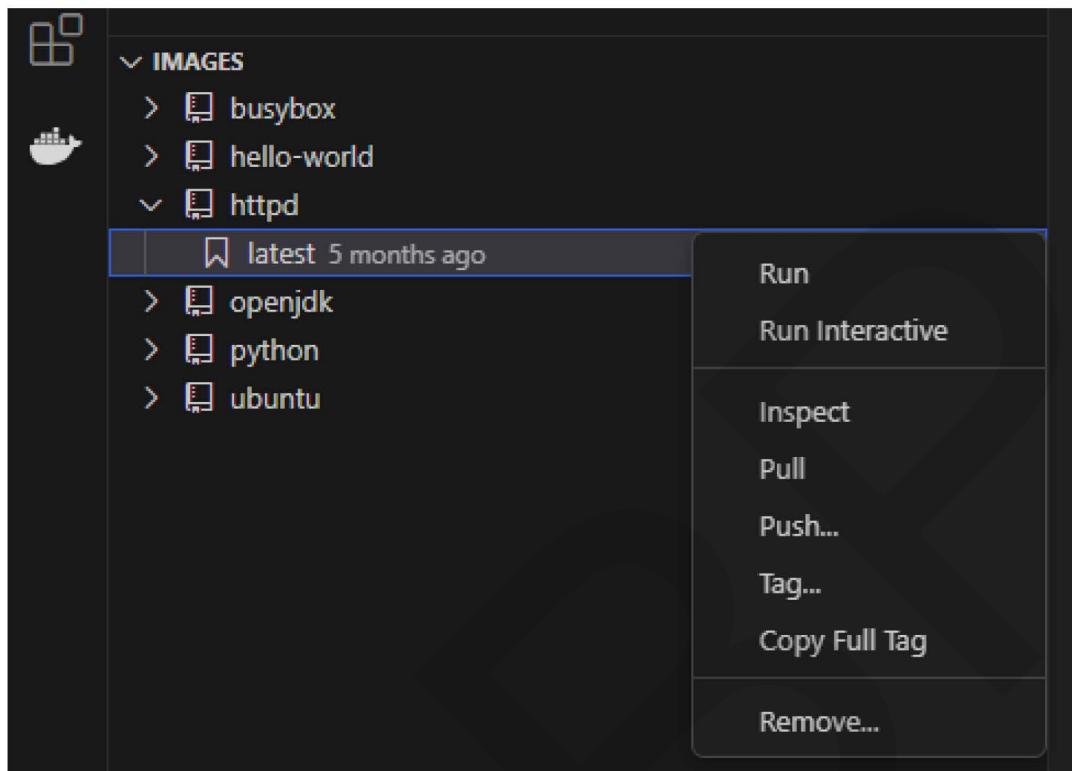




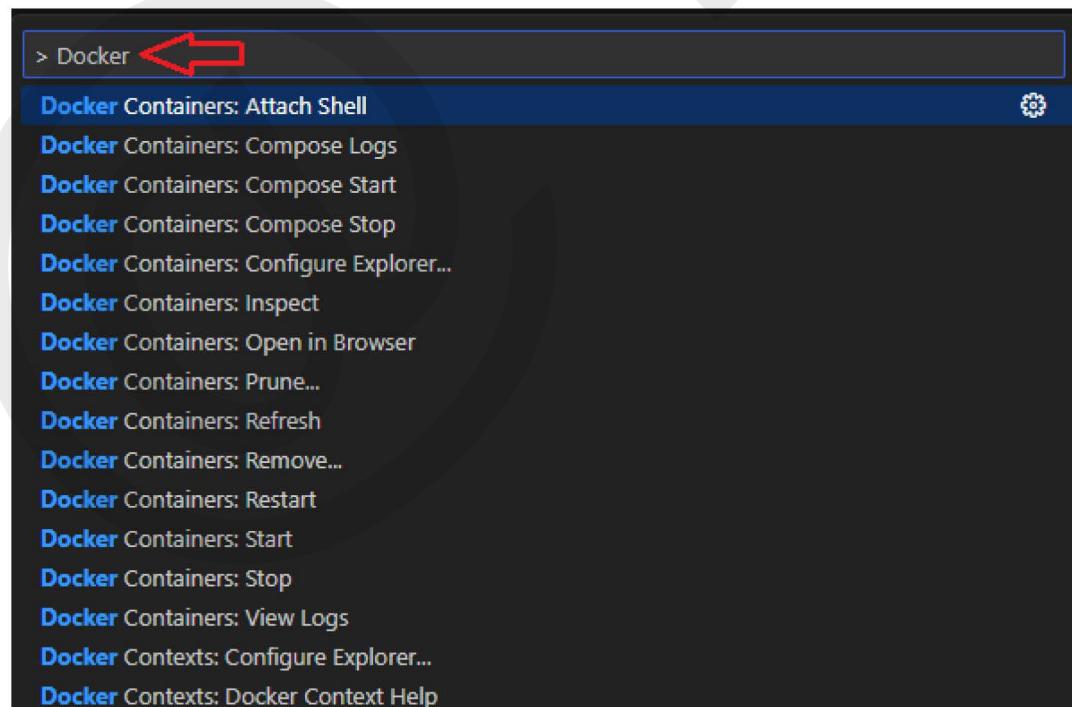
elindult a konténeren belüli terminál
az **exit** parancccsal tudunk kilépni, majd bármelyik billentyű lenyomása után bezáródik a terminál ablaka

A konténereket újra tudjuk indítani, le tudjuk állítani, valamint megtekinthetjük a log bejegyzéseket, a konténereken belüli fájlokat stb.

Képek (images) kezelése:



a képek helyi menüjében tudjuk futtatni a képet (interaktív módban is), letölteni (a meglévő image legújabb változatát), feltölteni (megosztani) a Docker Hub-on, címkézni, törölni stb.



a keresősávban számos további Docker funkciót elérhetünk

10. Szolgáltatások összekapcsolása (több konténer egyidejű futtatása)

10.1 Joomla! CMS rendszer telepítése Docker konténerbe

A Joomla! alapú weboldal futtatásához Docker konténerekben szükség van egy Joomla! konténerre (amely tartalmazza a Joomla! és a webszerver környezetet), valamint egy MySQL vagy MariaDB konténerre az adatbázis kezeléséhez.

Hozzunk létre egy egyedi Docker-hálózatot:

A külön hálózat biztosítja, hogy a Joomla! és a MySQL konténerek kommunikálni tudjanak egymással.

```
docker network create joomla_network
```

Ez a parancs virtuális hálózatot hoz létre, amelyhez a Joomla! és a MySQL konténerek csatlakozhatnak.

Indítsunk egy MySQL konténert:

A Joomla! adatbázisának kezeléséhez szükségünk lesz egy MySQL vagy MariaDB konténerre.

Az alábbi parancs elindítja a MySQL konténert, és biztosítja az adatbázis perzisztenciáját egy Docker-volume segítségével:

```
docker run --name joomla_db --network joomla_network -e  
MYSQL_ROOT_PASSWORD=#Aa123456789@ -e MYSQL_DATABASE=joomla -e  
MYSQL_USER=joomla_user -e MYSQL_PASSWORD=#mysql12345@ -v  
mysql_data:/var/lib/mysql -d mysql:latest
```

- docker run: Ez egy új konténert indít a megadott beállításokkal.

--name joomla_db: A konténer neve. Ezt használjuk majd a Joomla! adatbázis-kapcsolat konfigurálásához.

--network joomla_network: A konténer csatlakozik a korábban létrehozott Docker-hálózathoz.

-e opciók (környezeti változók): Beállítják a MySQL konfigurációt:

- MYSQL_ROOT_PASSWORD: A MySQL root felhasználó jelszava.
- MYSQL_DATABASE: Létrehoz egy adatbázist, amelyet a Joomla! fog használni.
- MYSQL_USER és MYSQL_PASSWORD: Létrehoz egy új felhasználót az adatbázishoz.

-v mysql_data:/var/lib/mysql: Egy volume-ot csatol a MySQL adatbázis tárolására. Ez biztosítja, hogy az adatok megmaradjanak a konténer újraindítása után is.

mysql:latest: A hivatalos MySQL Docker-kép legfrissebb verzióját használja.

Indítsuk el a Joomla! konténert:

```
docker run --name joomla_website --network joomla_network -p 8082:80 -v  
joomla_data:/var/www/html -e JOOMLA_DB_HOST=joomla_db -e  
JOOMLA_DB_USER=joomla_user -e JOOMLA_DB_PASSWORD=#mysql12345@ -e  
JOOMLA_DB_NAME=joomla -d joomla
```

1. docker run

Ez egy új konténert indít a megadott beállításokkal.

2. --name joomla_website

A konténer neve joomla_website lesz. Ez egyedi névvel azonosítja a konténert, amely később használható például parancsok futtatásához vagy naplók lekéréséhez.

3. --network joomla_network

A Joomla! konténer a joomla_network nevű Docker-hálózathoz csatlakozik. Ez lehetővé teszi, hogy a konténer kommunikáljon más konténerekkel (például egy MySQL konténerrel) ezen a hálózaton keresztül.

4. -p 8082:80

Porttovábbítás a host és a konténer között:

A **8082**-as port a host gépen a Joomla! weboldalhoz lesz hozzárendelve.

A 80-as port a konténerben futó Apache webszerver alapértelmezett portja. A gazdagép böngészőjében elérhetjük a Joomla! oldalt a **http://192.168.56.99:8082** címen.

5. -v joomla_data:/var/www/html

Ez egy volume csatolása:

A joomla_data Docker volume lesz használva a Joomla! weboldal fájljainak tárolására.

A konténeren belül a Joomla! fájlrendszer a **/var/www/html** mappában található. Ez azt jelenti, hogy a fájlok tartósak maradnak, még akkor is, ha a konténert újraindítjuk.

6. Környezeti változók: -e

JOOMLA_DB_HOST=joomla_db

A Joomla! számára itt megadjuk az adatbázis hosztját. joomla_db a MySQL konténer neve a Docker-hálózaton belül.

JOOMLA_DB_USER=joomla_user

Az adatbázis-felhasználó neve, amely a Joomla! telepítéséhez szükséges. Ez megegyezik a MySQL konténer indításakor létrehozott felhasználónévvel.

JOOMLA_DB_PASSWORD=#mysql12345@

Az adatbázis-felhasználó jelszava. A Joomla! ezt használja az adatbázishoz való csatlakozáshoz.

JOOMLA_DB_NAME=joomla

Az adatbázis neve, amelyet a Joomla! használ. Ennek meg kell egyeznie a MySQL konténerben létrehozott adatbázis nevével.

7. -d

Ez a parancs "detached" módban futtatja a konténert, vagyis a háttérben futtatja azt, anélkül, hogy a terminálhoz lenne csatolva.

8. joomla

Ez a Docker Hub-ról letöltött Joomla! képet jelöli, amelyet a konténer futtat.

Ha a futó konténerünk fájljait szeretnénk megnézni, módosítani, akkor adjuk ki az alábbi parancsot:

docker exec -t -i joomla_website /bin/bash

A parancs egy konténeren belül egy interaktív Bash shell-t indít.

docker exec: Ez a Docker parancs lehetővé teszi, hogy egy már futó konténerben parancsot futtassunk.

-t: Ez az opció egy "terminált" (pszeudo-TTY) nyit a parancshoz, így interaktív munkavégzésre alkalmas környezetet hoz létre.

-i: Ez az opció lehetővé teszi az interaktivitást, vagyis a billentyűzetről érkező bemenet továbbítását a konténeren belüli parancshoz.

joomla_website: Ez a konténer neve vagy azonosítója, amelyben a parancsot futtatni szeretnénk. Ebben az esetben egy "joomla_website" nevű konténerről van szó.

/bin/bash: Ez a parancs a konténerben. A /bin/bash indításával egy Bash shell-t indítunk, amely lehetővé teszi a konténer belsejében lévő fájlrendszer és környezet felfedezését, illetve parancsok futtatását.

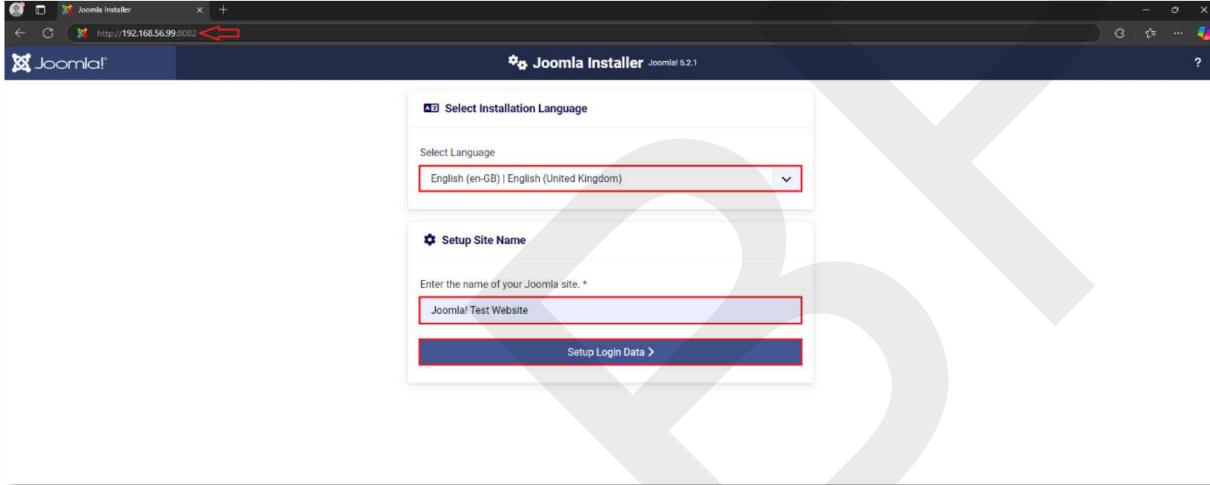
```
ls  
apt update  
apt install nano -y
```

(Ezután tudjuk szerkeszteni a konténerben lévő fájlokat.)

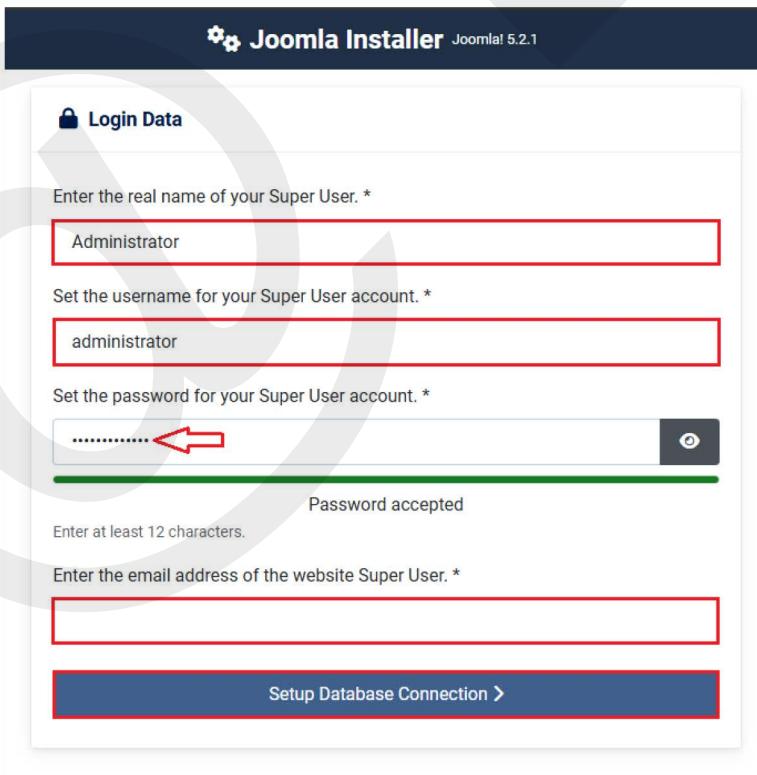
exit

Konfiguráljuk a Joomla!-t a böngészőben:

A gazdagépről **http://192.168.56.99:8082** címmel érjük el a telepítőt:



The screenshot shows the Joomla Installer interface. In the top left, the URL bar displays `http://192.168.56.99:8082`. The main window title is "Joomla Installer Joomla! 5.2.1". The first step, "Select Installation Language", has "English (en-GB) | English (United Kingdom)" selected. The second step, "Setup Site Name", has "Joomla! Test Website" entered into the "Enter the name of your Joomla site." field. A red arrow points to the URL bar.



The screenshot shows the "Login Data" configuration screen. It includes fields for the Super User's real name ("Administrator"), username ("administrator"), and password ("....."). A red arrow points to the password field. Below the password field, a message says "Password accepted" and "Enter at least 12 characters.". The next step, "Setup Database Connection >", is visible at the bottom. A red arrow points to the password field.

jelszó: #Aa123456789@

e-mail cím: ha valós környezetben fut a weboldal, akkor az adminisztrátor e-mail címe

 Joomla Installer Joomla! 5.2.1

 Database Configuration

Select the database type. *

MySQLi

Enter the host name, usually "localhost" or a name provided by your host. *

joomla_db

Enter the database username you created or a username provided by your host. *

joomla_user

Enter the database password you created or a password provided by your host.

Enter the database name. *

joomla

Enter a table prefix or use the randomly generated one. *

uvfkp_

If you are using an existing database with tables with the same prefix, Joomla will rename those existing tables by adding the prefix "bak_".

Connection Encryption *

Default (server controlled)

Install Joomla >

jelszó: #mysql12345@
(valós környezetben erősebb jelszót adjunk)

 Congratulations!

Congratulations! Your Joomla site is ready.

Install Additional Languages >

Open Site

Open Administrator

CASSIOPEIA

You are here: Home

Home

Main Menu

[Home](#)

Login Form

Username



Password



Remember Me

Log in

[Forgot your password?](#)

[Forgot your username?](#)

11. A Docker Compose

A Docker Compose egy eszköz, amely lehetővé teszi több Docker-konténer egyidejű kezelését. Az alkalmazásokat egy *compose.yaml* fájlban konfiguráljuk, amely tartalmazza a konténerek definícióit, például:

- képek
- hálózati beállítások
- környezeti változók
- volume-ok

A Docker Compose telepítése:

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

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
docker compose version
```

11.1 YAML fájl létrehozása és konfigurálása (WordPress telepítése)

A Docker Compose YAML fájl beállításához és a Docker Compose működésének bemutatásához egy, a Joomla!-hoz hasonló CMS rendszert, a WordPress-t konfiguráljuk és telepítjük.

```
mkdir ~/docker/wordpress
nano ~/docker/wordpress/compose.yaml
```

Másoljuk a fájlba az alábbi konfigurációt:

```
services:  
  db:  
    image: mariadb:latest  
    container_name: wordpress_db  
    volumes:  
      - db_data:/var/lib/mysql  
    restart: always  
    environment:  
      - MYSQL_ROOT_PASSWORD=#Aa123456789@  
      - MYSQL_DATABASE=wp_db  
      - MYSQL_USER=wp_mysql_user  
      - MYSQL_PASSWORD=#mysql12345@  
    expose:  
      - 3306  
      - 33060  
  wordpress:  
    image: wordpress:latest  
    container_name: wordpress_website  
    volumes:  
      - wp_data:/var/www/html  
    ports:  
      - 8084:80  
    restart: always  
    environment:  
      - WORDPRESS_DB_HOST=db  
      - WORDPRESS_DB_USER=wp_mysql_user  
      - WORDPRESS_DB_PASSWORD=#mysql12345@  
      - WORDPRESS_DB_NAME=wp_db  
  volumes:  
    db_data:  
    wp_data:
```

Mentsük a fájlt és lépjünk ki!

Ez egy Docker Compose konfigurációs fájl, amely két szolgáltatásból (konténerből) áll: egy MariaDB (MySQL-kompatibilis) adatbázisszerverből és egy WordPress CMS rendszerből.

1. services:

Ez határozza meg a különböző szolgáltatásokat, amelyeket Docker-konténerekként fogunk futtatni.

Szolgáltatás: db

Ez a szolgáltatás egy MariaDB (MySQL-kompatibilis) adatbázisszert futtat:

image: mariadb: A **mariadb** Docker-képet használja, amely tartalmazza a MariaDB adatbázis-szert.

container_name: wordpress_db: A konténer neve **wordpress_db** lesz. Ez egyedi azonosításra szolgál.

volumes:

A **db_data** nevű Docker-volume-ot csatolja a konténer **/var/lib/mysql** könyvtárához.

Ez biztosítja, hogy az adatbázis-állományok (pl. táblák, adatok) tartósak maradjanak, még akkor is, ha a konténer törlődik vagy újraindul.

restart: always: Ez a konténer automatikusan újraindul, ha leáll vagy ha a Docker gazdagép újraindul.

environment: Környezeti változók, amelyek az adatbázis-konfigurációt adják meg:

MYSQL_ROOT_PASSWORD: Az adatbázis root jelszava (#Aa123456789@).

MYSQL_DATABASE: Az adatbázis neve (**wp_db**), amelyet a WordPress használni fog.

MYSQL_USER: Az adatbázis-felhasználó neve (**wp_mysql_user**).

MYSQL_PASSWORD: Az adatbázis-felhasználó jelszava (#mysql12345@).

expose: Csak a Docker-hálózaton belül teszi elérhetővé a következő portokat:

3306: Az alapértelmezett MariaDB/MySQL port.

33060: MariaDB SQL protokollokkal való kapcsolódáshoz használt port.

Szolgáltatás: wordpress

Ez a szolgáltatás egy WordPress webalkalmazást futtat:

image: wordpress: A **wordpress** Docker-képet használja, amely tartalmazza a WordPress alkalmazást és annak futtatásához szükséges környezetet (PHP, Apache).

container_name: wordpress_website: A konténer neve **wordpress_website** lesz.

volumes:

A **wp_data** nevű Docker-volume-ot csatolja a konténer **/var/www/html** könyvtárához.

Ez biztosítja, hogy a WordPress fájlok (például feltöltött képek, bővítmények) tartósak maradjanak.

ports:

A konténer belső 80-as portját a gazdagép 8084-es portjára irányítja át.

restart: always: Ez a konténer is automatikusan újraindul, ha leáll vagy ha a Docker gazdagép újraindul.

environment: Környezeti változók, amelyek a WordPress adatbáziskapcsolatát konfigurálják:

WORDPRESS_DB_HOST: Az adatbázis-hoszt neve (**db**). Ez a **db** szolgáltatás Docker-hálózati neve, így a WordPress ezen keresztül éri el az adatbázist.

WORDPRESS_DB_USER: Az adatbázis-felhasználó neve (**wp_mysql_user**), amely megegyezik az adatbázis szolgáltatásban beállított értékkel.

WORDPRESS_DB_PASSWORD: Az adatbázis-felhasználó jelszava (#mysql12345@), amely szintén megegyezik a **db** szolgáltatásban megadott értékkel.

2. Volume-ok:

A fájl implicit módon definiál két Docker-volume-ot:

db_data: Az adatbázis fájljainak tartós tárolására szolgál.

wp_data: A WordPress fájljainak tartós tárolására szolgál.

Ezek a volume-ok automatikusan létrejönnek a **docker compose up** parancs futtatásakor.

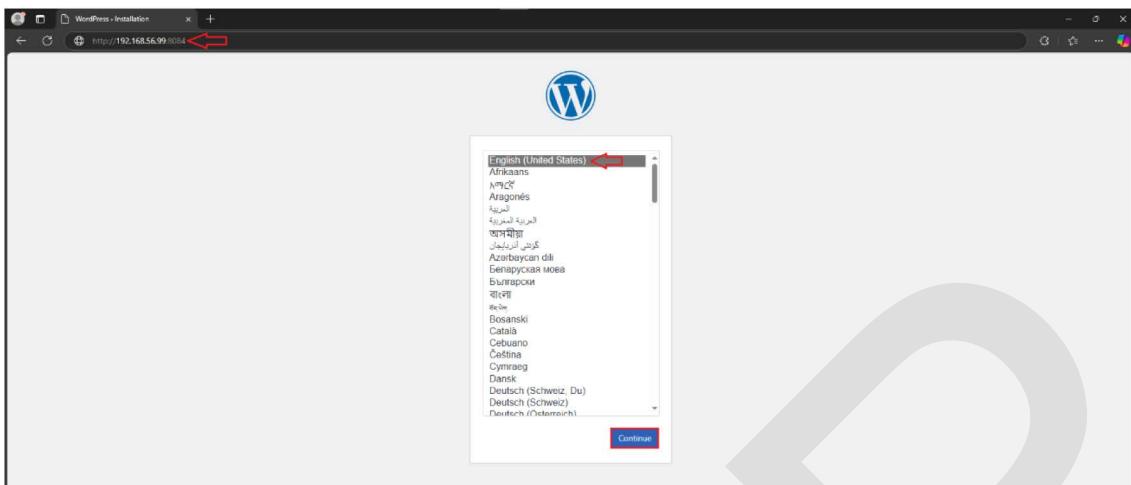
A fenti konfiguráció egy WordPress alapú weboldal telepítésére alkalmas, amelyhez egy MariaDB adatbázis-szerver tartozik. Az adatbázis és a webalkalmazás a Docker-hálózaton keresztül kommunikál egymással. A volume-ok segítségével az adatok és fájlok megőrződnek, még ha a konténereket le is állítják vagy törlik.

A futtatás lépései:

```
cd ~/docker/wordpress/  
docker compose up -d
```

Konfiguráljuk a WordPress-t a böngészőben:

A gazdagépről **http://192.168.56.99:8084** címmel érjük el a weboldalt (a WordPress telepítójét):



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Do not worry, you can always change these settings later.

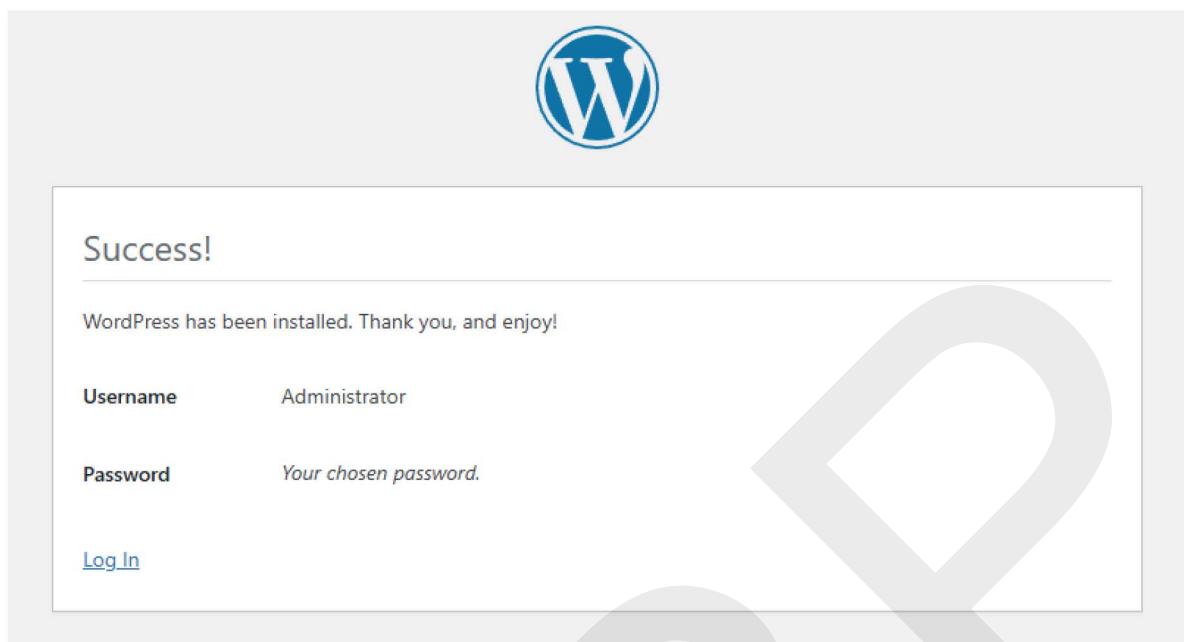
Site Title	WordPress Test Website
Username	Administrator
Password	***** Medium <input type="button" value="Show"/>
Your Email	
Search engine visibility	<input type="checkbox"/> Discourage search engines from indexing this site It is up to search engines to honor this request.

Important: You will need this password to log in. Please store it in a secure location.

Double-check your email address before continuing.

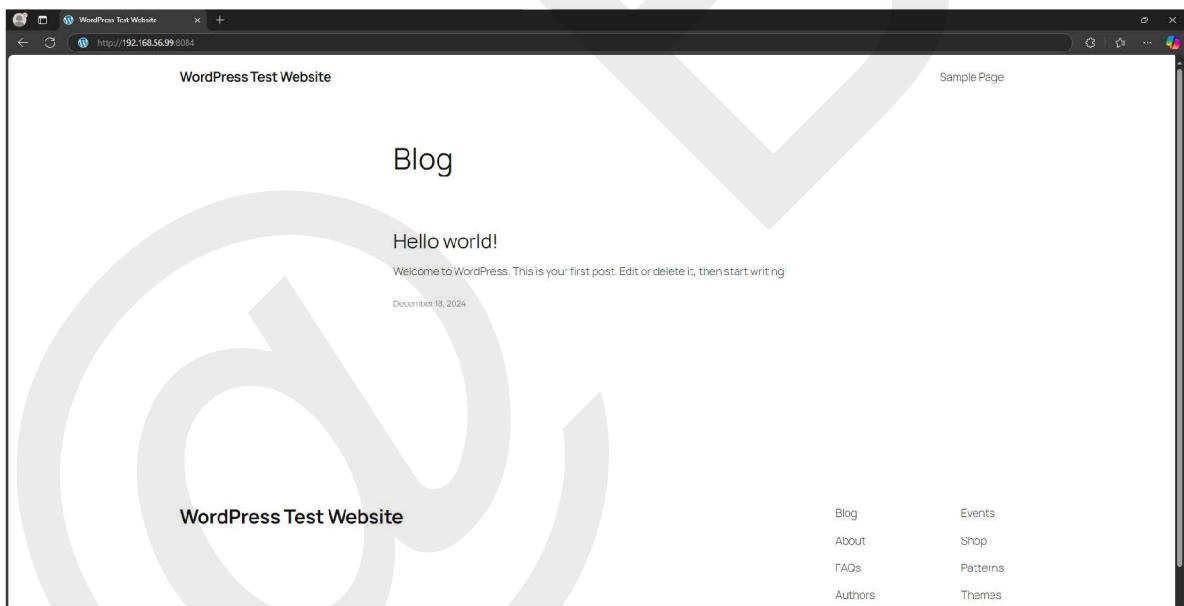
Install WordPress

jelszó: #Aa123456789@
(valós környezetben erősebb jelszót adjunk)



a „log In”-re kattintva beléphetünk a WordPress adminisztrációs felületére

Nyissuk meg a weboldalt:



11.2 A Docker Compose kezelése

Konténer indítása - alapértelmezett fájl (compose.yaml) - esetében:

docker compose up

Háttérben történő futtatás:

docker compose up -d

A konténerek futásának szüneteltetése:

`docker compose pause`

A konténerek futásának folytatása:

`docker compose unpause`

A konténerek futásának leállítása:

`docker compose stop`

A konténerek futásának indítása:

`docker compose start`

Konténerek leállítása (az összes futó szolgáltatás leállítása és törlése):

`docker compose down`

Szolgáltatás újraindítása (egy adott szolgáltatás újraindítása):

`docker compose restart <szolgáltatás neve>`

Állapot ellenőrzése:

`docker compose ps`

Naplók megtekintése (egy adott szolgáltatás esetében):

`docker compose logs <szolgáltatás neve>`

Az összes szolgáltatás naplójának figyelése:

`docker compose logs -f`

Volume-ok, hálózatok törlése (leállítás után):

`docker compose down -v`

A Docker Compose verzió ellenőrzése:

`docker compose version`

12. A Dockerfile

A Dockerfájl (Dockerfile) egy egyszerű szöveges fájl, amely tartalmazza az utasításokat és parancsokat, amelyek egy Docker-kép (Docker image) létrehozásához szükségesek. A

Docker-kép pedig egy futtatható, csomagolt környezet, amely tartalmaz minden szükséges komponenst (alkalmazást, könyvtárakat, konfigurációkat stb.) egy konténer futtatásához.

Mire használjuk a Dockerfájlt?

- **Automatizált képkészítés:** Leírhatjuk egy konténer környezetének pontos felépítését (pl. használt operációs rendszer, szükséges függőségek, alkalmazásfájlok, konfigurációk).
- **Reprodukálható környezetek:** Ugyanaz a Dockerfájl minden azonos környezetet fog előállítani, függetlenül attól, hogy hol futtatjuk (fejlesztői gép, szerver, CI/CD pipeline).
- **Egyszerűbb telepítés és fejlesztés:** Nem kell kézzel beállítanunk a függőségeket és konfigurációkat, a Dockerfájl minden definiál.
- **Portabilitás:** A Docker-képek futtathatók különböző rendszereken, ha ott telepítve van a Docker.

12.1 A Dockerfile felépítése

1. Alap image kiválasztása (FROM)

Ez az utasítás kötelező, és meghatározza, hogy melyik image-ból induljon az építés. Ez lehet egy hivatalos image (pl. ubuntu, node, python) vagy egy egyedi, előzőleg létrehozott image.
pl.:

```
FROM ubuntu:latest
```

2. Karakterkódolás beállítása (opcionális)

Az alapértelmezett karakterkészlet beállítása (általában hasznos, ha nem UTF-8 az alapértelmezett).

pl.:

```
ENV LANG C.UTF-8
```

3. Függőségek és eszközök telepítése (RUN)

A RUN utasítás olyan parancsokat futtat, amelyek módosítják az image-et az építés során. Általában csomagok telepítésére és konfigurálására használják.

pl.:

```
RUN apt-get update && apt-get install -y \
    curl \
    vim
```

4. Munkakönyvtár beállítása (WORKDIR)

Meghatározza azt a könyvtárat, ahol a következő parancsok végrehajtódnak.

pl.:

```
WORKDIR /app
```

5. Fájlok másolása a konténerbe (COPY vagy ADD)

A COPY és ADD utasítások a fájlokat a host gépről a konténer image-be másolják. Az ADD támogatja a távoli URL-ekről való letöltést is.

pl.:

```
COPY . /app
```

6. Környezeti változók beállítása (ENV)

Környezeti változók definiálása a konténer számára.

pl.:

```
ENV APP_ENV production
```

7. Portok kitettsége (EXPOSE)

Jelzi, hogy a konténer melyik porton fogad bejövő forgalmat.

pl.:

```
EXPOSE 8080
```

8. Fő futtatási parancs beállítása (CMD vagy ENTRYPOINT)

A CMD vagy az ENTRYPOINT határozza meg, hogy mi fut le, amikor a konténer elindul.

CMD: Parancsok megadására szolgál. Több CMD is megadható, de csak az utolsó érvényesül.

pl.:

```
CMD ["python", "app.py"]
```

ENTRYPOINT: Inkább olyan esetekben használjuk, ahol a konténer mindenkor egy konkrét alkalmazást futtat.

pl.:

```
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

12.2 Egyszerű Shell script futtatása, image készítése

Hozzunk létre egy Bash script fájlt az alábbi tartalommal:

```
mkdir ~/docker/scripts  
cd ~/docker/scripts/  
nano osszeg.sh
```

Másoljuk bele a fájlba az alábbi sorokat:

```
#!/bin/bash  
  
SZAM=1  
SZUM=0  
  
while [ $SZAM -ne 0 ]  
do  
    echo -n "Szam: "  
    read SZAM  
    let SZUM=$SZUM+$SZAM  
done  
  
echo "Osszeg: " $SZUM
```

Mentsük a fájlt és lépjünk ki!

Ez a script egy számösszegző program, amely:

1. Folyamatosan bekér számokat a felhasználótól.
2. Ezeket a számokat összeadja.
3. Amikor a felhasználó 0-t ír be, a program leáll, és kiírja az összeget.

A script csak számokat tud kezelni, szöveget nem, és nem is kezeli a hibás bemenetet!

Változók inicializálása:

```
SZAM=1  
SZUM=0
```

- **SZAM**: Az aktuális beírt számot tárolja. Kezdeti értéke 1, hogy a while ciklus először mindenképpen lefusson.
- **SZUM**: Az eddig beírt számok összegét tárolja. Kezdetben 0.

A ciklus indítása:

```
while [ $SZAM -ne 0 ]
```

A while ciklus addig fut, amíg a SZAM értéke nem egyenlő 0-val (-ne az „not equal” operátor a Bash-ben).

Szám bekérése:

```
echo -n "Szam: "
read SZAM
```

- A script bekér egy számot a felhasználótól, amelyet a SZAM változóba helyez.
- Az echo -n üzenet kiírása nem ugrik új sorba, így a felhasználó ugyanabban a sorban láthatja a promptot.

Összeg frissítése:

```
let SZUM=$SZUM+$SZAM
```

- A let parancs matematikai műveleteket végez Bash-ben.
- A SZUM értékéhez hozzáadja a SZAM aktuális értékét.

A ciklus vége:

- A ciklus addig ismétlődik, amíg a SZAM értéke nem 0.
- Amikor a felhasználó 0-t ír be, a while feltétele hamissá válik, és a ciklus kilép.

Összeg kiírása:

```
echo "Osszeg: " $SZUM
```

A script kiírja a végső összeget, amely az összes korábban megadott szám összege.

Megjegyzések:

Nulla, mint kilépési feltétel:

- A 0 megadása nem adódik hozzá az összeghez, hanem jelzi a scriptnek, hogy vége.

Negatív számok:

- A script kezeli a negatív számokat is, azokat hozzáadja az összeghez.

Hozzunk létre egy Dockerfile-t:

```
cd ~/docker/scripts
nano Dockerfile
```

Másoljuk a fájlba az alábbi sorokat:

```
# Debian alapú image használata
FROM debian:latest

# Másoljuk a Bash scriptet a konténerbe
COPY osszeg.sh /usr/local/bin/osszeg.sh

# Bash telepítése (ha nincs alapértelmezésben)
RUN apt-get update && apt-get install -y bash

# Script futtatási engedély beállítása
RUN chmod +x /usr/local/bin/osszeg.sh

# Indítási parancs
CMD ["bash", "/usr/local/bin/osszeg.sh"]
```

Mentsük a fájlt és lépjünk ki!

Docker image építése:

`docker build -t osszeg .`

Konténer futtatása:

`docker run -it --name osszegscript osszeg`

--name: Ez határozza meg a konténer nevét (jelen esetben osszegscript).

-it: Interaktív mód a bemenetek fogadásához.

osszeg: Az image neve.

Ha azt szeretnénk, hogy a konténer automatikusan töröljön a futás után, akkor a **--rm** opciót adjuk hozzá a parancshoz!

A konténer újraindítása leállítás után:

`docker start -ai osszegscript`

-a: A "attach" (csatlakozás) kapcsoló. Ez azt jelenti, hogy a parancs futtatása után a konténer standard kimenete és bemenete a terminálhoz lesz csatolva. Így látni fogjuk a konténer által generált kimenetet, és interakcióba léphetünk vele.

- i: Az "interactive" (interaktív) kapcsoló. Ez lehetővé teszi, hogy a konténer indításakor bemenetet adjunk a konténernek a terminálon keresztül, interaktív módon.

13. Feladatok

- Készítsünk Docker Compose segítségével egy Nginx alapú szervert, amely statikus HTML fájlokat szolgál ki:

Hozzuk létre az alábbi mappaszerkezetet:

```
mkdir ~/docker/nginx-static && cd ~/docker/nginx-static  
mkdir site && cd site  
nano index.html
```

Másoljuk bele az fájlba az alábbi kódot:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Welcome to Nginx</title>  
</head>  
<body>  
    <h1>Hello, World!</h1>  
    <p>This is a simple static HTML site served by Nginx.</p>  
</body>  
</html>
```

Mentsük a fájlt és lépjünk ki!

1.1 Hozzuk létre a YAML fájlt:

```
cd ~/docker/nginx-static  
nano compose.yaml
```

Másoljuk bele az fájlba az alábbi sorokat:

```
services:  
  web:  
    image: nginx:latest  
    container_name: staticwebsite-nginx  
    volumes:  
      - ./site:/usr/share/nginx/html:ro  
    ports:  
      - "8086:80"
```

Mentsük a fájlt és lépjünk ki!

1.2 Docker Compose indítása:

docker compose up -d

A gazdagépről **http://192.168.56.99:8086** címmel érjük el a weboldalt!

1.3 Docker Compose leállítása:

docker compose down

2. Készítsünk Docker Compose segítségével MySQL adatbázist phpMyAdmin adminisztrációs felülettel:

Hozzuk létre az alábbi mappaszerkezetet:

```
mkdir ~/docker/mysql-phpmyadmin  
nano ~/docker/mysql-phpmyadmin/compose.yaml
```

Másoljuk a fájlba az alábbi konfigurációt:

```
services:  
  mysql:  
    image: mysql:latest  
    container_name: mysql_server  
    environment:  
      MYSQL_ROOT_PASSWORD: rootpassword  
      MYSQL_DATABASE: testdb  
      MYSQL_USER: testuser  
      MYSQL_PASSWORD: testpassword  
    ports:  
      - "3306:3306"  
    volumes:  
      - db_data:/var/lib/mysql  
  phpmyadmin:  
    image: phpmyadmin/phpmyadmin:latest  
    container_name: phpmyadmin  
    environment:  
      PMA_HOST: mysql  
      PMA_USER: testuser  
      PMA_PASSWORD: testpassword  
    ports:  
      - "8088:80"  
    depends_on:  
      - mysql  
  volumes:  
    db_data:
```

Mentsük a fájlt és lépjünk ki!

2.1 Docker Compose indítása:

```
cd ~/docker/mysql-phpmyadmin  
docker compose up -d
```

A gazdagépről **http://192.168.56.99:8088** címmel érhetjük el a phpMyAdmin felületet!

2.2 Docker Compose leállítása:

```
docker compose down
```

3. Hozzunk létre egy egyszerű Node.js alkalmazást, amely egy HTTP szervert indít és egy üzenetet jelenít meg:

Hozzuk létre az alábbi mappaszerkezetet:

```
mkdir ~/docker/nodejs  
nano ~/docker/nodejs/webserver.js
```

Másoljuk a fájlba az alábbi kódot:

```
const http = require('http');  
  
const PORT = 3000;  
  
const server = http.createServer((req, res) => {  
    res.statusCode = 200;  
    res.setHeader('Content-Type', 'text/plain');  
    res.end('Hello, Docker from Node.js!\n');  
});  
  
server.listen(PORT, () => {  
    console.log(`Server running at http://localhost:${PORT}/`);  
});
```

Mentsük a fájlt és lépjünk ki!

3.1 Készítsünk egy Dockerfile-t:

```
cd ~/docker/nodejs  
nano Dockerfile
```

Másoljuk a fájlba az alábbi sorokat:

```
# 1. Használunk egy hivatalos Node.js bázis image-t  
FROM node:14
```

```
# 2. Állítsuk be a munkakönyvtárat  
WORKDIR /app
```

```
# 3. Másoljuk be a helyi fájlokat a konténerbe  
COPY webserver.js /app
```

```
# 4. Állítsuk be az alapértelmezett parancsot  
CMD ["node", "webserver.js"]
```

Mentsük a fájlt és lépjünk ki!

- Node.js bázis image:** A node:14 tartalmazza a Node.js és npm alapvető futtatási környezetét.
- Munkakönyvtár:** A WORKDIR az /app mappát állítja be a konténer belsejében.
- Fájlok másolása:** A COPY parancs átmásolja a server.js fájlt a konténerbe.
- Parancs:** A CMD határozza meg a konténer futtatásakor elinduló alapértelmezett parancsot.

3.2 A konténer futtatása:

Docker image létrehozása:

```
docker build -t node-docker-app .
```

A konténer futtatása:

```
docker run --name nodejswebserver -d -p 3000:3000 node-docker-app
```

A gazdagépről **http://192.168.56.99:3000** címmel érjük el a weboldalt!

4. Hozzunk létre egy konténert, amely számolja az 1-től 10-ig terjedő számok összegét:

Hozzuk létre az alábbi mappaszerkezetet:

```
mkdir ~/docker/calculator
```

```
nano ~/docker/calculator/Dockerfile
```

Másoljuk a fájlba az alábbi sorokat:

```
# 1. Használunk egy Alpine Linux bázis image-et  
FROM alpine:latest  
  
# 2. Telepítsük a Bash shell-t  
RUN apk add --no-cache bash  
  
# 3. Alapértelmezett parancs egy Bash szkript futtatása  
CMD ["bash", "-c", "echo $(seq 1 10 | paste -sd+ - | bc)"]
```

Mentsük a fájlt és lépjünk ki!

Docker image létrehozása:

```
cd ~/docker/calculator  
docker build -t sum-calculator .
```

A konténer futtatása:

```
docker run --name bash_calculator sum-calculator
```

5. Saját Python alkalmazás futtatása Dockerben

Írunk egy Python alkalmazást, ami az első 20 Fibonacci számot írja ki a képernyőre!

Készítsünk ebből a programból egy Docker image-et, majd futtassuk konténerként!

Hozzuk létre az alábbi mappaszerkezetet:

```
mkdir ~/docker/python  
nano ~/docker/python/fibonacci.py
```

Másoljuk az alábbi kódot a fájlba:

```
def generate_fibonacci(n):  
    if n <= 0:  
        return []  
    fib = [0, 1]  
    while len(fib) < n:  
        fib.append(fib[-1] + fib[-2])  
    return fib[:n]  
n = 20  
print(f"Az első {n} Fibonacci-szám:", generate_fibonacci(n))
```

Mentsük a fájlt és lépjünk ki!

Hozzunk létre egy Dockerfile-t:

```
cd ~/docker/python  
nano Dockerfile
```

Másoljuk az alábbi sorokat a fájlba:

```
FROM python  
COPY fibonacci.py /app/fibonacci.py  
WORKDIR /app  
CMD ["python", "fibonacci.py"]
```

Mentsük a fájlt és lépjünk ki!

Készítsük el az image fájlt:

```
docker build -t fibonacci-python-app .
```

Futtassuk a konténeret:

```
docker run --name fibonacci-py fibonacci-python-app
```