

Széchenyi István Egyetem

GKLB_INTM006
Modern szoftverfejlesztési eszközök

Docker bemutatása

Készítette:
Debnárik Roland József
DJLF47
Mérnökinformatikus

Esztergom, 2019

Tartalomjegyzék

Téma választás indoklása:	3
Történelmi áttekintő:	3
Előismeretek	4
Docker terminológia	4
Képfájl (image).....	4
Konténer (Container).....	4
Tárház (registry)	4
Docker képfájlok.....	4
Előnyök	5
Hátrányai	6
Rendszer követelmények:	6
Docker telepítése	7
Docker beépülők	7
Docker Daemon	7
Docker CLI.....	7
Konténer telepítése lehetőségek	8
Docker Compose	8
version:	8
services:	8
image:	8
restart:	8
ports:	8
container_name:	9
volumes:	9
env_file:	9
Dockerfile	9
From	9
RUN.....	9
COPY	9
WORKDIR	9
EXPOSE	10
CMD	10
Elindítás	10
Build cache	10
Docker Taggelés.....	10

docker build -t roland.debnarik/docker:latest	10
Parancsok	11
Program parancsok felépítése.....	11
A parancs összetétele:.....	11
Docker parancsok	12
Docker ps	12
-a	12
Docker rm	12
Docker container prune.....	12
Docker system prune.....	12
Docker start	12
Docker logs	12
Docker stop	12
Docker kill	13
Docker exec	13
Docker exec -it.....	13

Téma választás indoklása:

Azért választottam Docker témát, mert folyamatosan próbálok up-to-date lenni az informatikába, mert nagyon gyorsan fejlődik az IT ipar és nem lehet megmaradni egy adott technológián, mert mindig lesz gyorsabb/olcsóbb.

A másik indok, hogy a jelenlegi munkaadóm egy új szoftver bevezetését tervezi külsős céggel lefejllesztve és már ők is Docker technológiát használnak, és miután elkészül a szoftver így nekünk kell üzemeltetni, és a hatékony üzemeltetésnek az egyik feltétele, hogy ismerjük a technológia sajátosságait.

Illetve, ezzel a tudással, akár a meglévő rendszereinket is lehet konténerizálni, amivel nagyon sok problémát meg lehet oldani (lentebb az esettanulmányba).

Történelmi áttekintő:

Solomon Hykes és Sebatien Pahl 2010-ben alapították a Docker vállalatukat, de még dotCloud néven, 2013-ban nevezték át magukat a jelenlegi nevükre.

A Docker debütálásuk 2013-ban történt, és egy nyílt forráskódú projekt lett, amit kezdetben Pythonban, majd Go-ban fejlesztették.

A Docker egy platform szolgáltatás (PaaS) ami Go nyelven fejlesztenek.

Honlapjuk: <https://www.docker.io>

Git repository: <https://github.com/docker/docker> de ez a link már átirányít a ./moby/moby-ra.

Előismeretek

Docker előjárójában meg kell értetnünk a programok és az operációs rendszerek kapcsolatát, amikor valaki az asztalán rákattint egy ikonra akkor az az ikon nem maga a program, ami elindul, hanem az egy link a számítógép háttértárolóján található programhoz, amit el fog indítani, a háttérben ilyenkor a parancsok értesítik az operációs rendszert, hogy keresse elő a háttértáron található programot, majd át kell töltenie a memóriába és ez után tudja lefuttatni a programot.

A programokban nem csak annyi lehetőség van, hogy dupla kattintással rákattintunk és futtatjuk a programot, a programoknak vannak különböző paramétereik, amik azt hivatottak előidézni, hogy ha a fejlesztő felkészítette a programot paraméter fogadására akkor adott paraméter esetén más adatot fog visszaküldeni a program, például ha kiadjuk terminálban az „ls” parancsok, akkor a rendszer ki fogja listázni a fájlokat és a mappákat, azonban ha azt írjuk be, hogy „ls -a” akkor minden olyan állományt is ki fog listázni ami „.”-al kezdődik. Pusztán paraméterekkel tudjuk befolyásolni a program működését.

A programoknak 3 darab csatornája van, ezek az:

- Input: ezzel paramétereket lehet átadni a programnak
- Output: ez a program kimenete az esetleges paraméterrel
- Error: Itt érkezik vissza minden olyan üzenet, ami hiba

Docker terminológia

Képfájl (image)

Az egy VM-nek megfelelő fájl együttese, amely tartalmaz minden olyan futtatáshoz szükséges kiegészítőt (lib, db, config...stb) ami szükséges az igény kielégítő alkalmazás futtatásához.

Konténer (Container)

Egy Docker image futtatott példánya.

Tárház (registry)

Képfájlok tára, alapesetben a helyi hoston, de a Docker cég fenntart egy nyilvános adatbázis ezek tarolására és elérésére.

Docker képfájlok

Az elkészült Docker képfájl rétegekből épül fel, ahol az alsóbb rétegek mindig nagyobb 'felületet' foglalnak el.

A képfájlok azok uniós típusú fájlrendszert alkotnak, ami azt jelenti, hogy az összes egymásra helyezett rétegből egy darab képfájlt generál, és a rétegeket tömörítve tárolódnak.

A Dockereket legtöbb esetben bizonyos sablonok alapján hozzuk létre (Dockerfile) ez általában egy Base image indul ki pl.: ubuntu. Az újabb réteg hozzáadásához ismernünk kell vagy a Dockerfile, vagy a hozzá adott réteg szintaxisát.

Előnyök

A technológia egyik előnye, hogy felismerte azt a gyakori fejlesztési és implementálási hibát, hogy amikor egy szoftverfejlesztő részleg vagy cég, akár Java, C# vagy web alapú alkalmazást készít, nem minden esetben tudják a programoknak a futtatását garantálni, mert Java esetén rossz java verzió lehet fent az adott futtató szerveren, vagy már nem támogatott az az eljárás, vagy esetleg C# esetén nincs feltelepítve a megfelelő .Net keretrendszer és ezeknek a problémáknak kiküszöbölésére rengeteg emberi órát kellett fordítani, de a Docker ezt a problémát ismerte fel és létrehozott egy olyan rendszert, ahol a programjainkat és a környezetünket különböző konténerekbe szervezhetjük amit fel is tölthettünk a felhőbe, és az adott helyen egy parancs segítségével azt a felhőben elmentett képfájlt letölti a rendszer és az ott tárolt parancsokkal létrehozta a megfelelő környezeti változókkal szükséges környezetet, és így már nem volt probléma egy automatikus frissítés ami tönkre is teheti a programok működését.

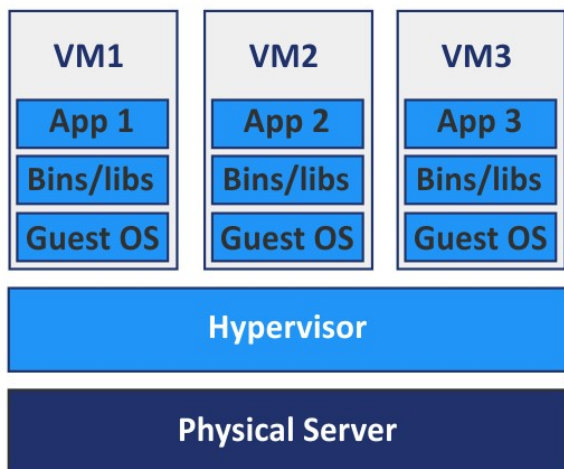
A technológia egy másik problémára hozott megoldást, amit a webfejlesztésben jelentkezett, hogy különböző webalkalmazásokat az üzemeltetők csak úgy tudtak külön szeparálni, ha külön szervereket üzemeltettek, ezek lehettek külön fizikai gépeken vagy virtualizáltan, de ezzel az volt a probléma, hogy költségben és mind erőforrásban nagy terhet jelentett, mivel magán a host szerveren lévő operációs rendszer fölé kellett még egy elszeparált számítógépet telepíteni, ami ugyan úgy rendelkezett operációs rendszerrel, ami csökkentette a szolgáltatás teljesítményét.

Ezzel szemben a Docker technológia Windows-os környezetben egy nem betekinthető réteget képez az operációs rendszeren és így éri el az utasítás készletét és így nincs szükség köztes Hypervisor rétegre.

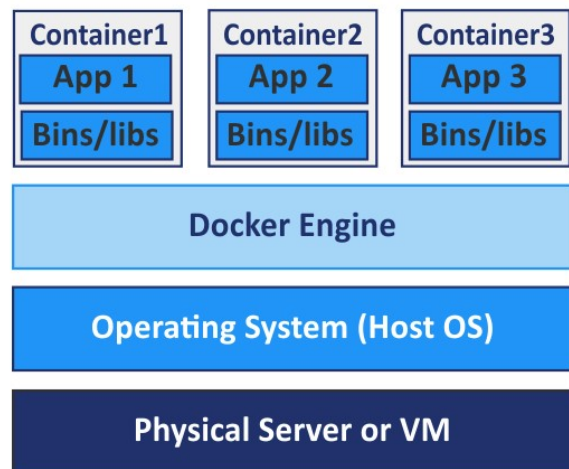
Ez mellett előfordulnak olyan eseteket, egy hoston futatott elkülönített mappákban elérhető weboldalak működéséhez környezeti változók más-más módon változnak és kihatással vannak a másik alkalmazásra, ami akár el is ellehetetleníti az alkalmazás futtatását/elérhetőségét.

Docker előnyei még, hogy bármely operációs rendszeren, Windows, Linux és macOS olyan környezetet tud létrehozni, ami az összes operációs rendszeren el tudja futtatni a lefejlesztett alkalmazást.

Virtual Machines



Containers



Hátrányai

Egy egyszerűbb programnak a teljes újra fordítása sok időt vehet igénybe, ha olyan környezetben van telepítve az adott alkalmazás vagy szolgáltatás.

A Docker konténerek típusát a gazdarendszer operációs rendszere határozza meg.

Rendszer követelmények:

- Windows:
 - Windows 10
 - 64 bit
 - Pro
 - Enterprise
 - Education (build 15063 vagy későbbi)
 - 64 bites processzor
 - 4GB memória
 - Hardveres virtualizációs támogatás
- Linux
 - 3.10 vagy magasabb Linux kernel
 - CS Docker Engine 1.13 vagy EE Daemon 17.03 vagy magasabb
 - 4Gb memória
 - 3GB szabad hely
 - Statikus IP cím
- macOS
 - 2010 után hardver
 - Legalább 10.13-as macOS
 - Legalább 4GB memória
 - 4.3.30 előtti verziójú VirtualBox nem megléte

Docker telepítése

Windows-ra és macOS-re a docker-com-ról tudjuk letölteni a konténer kezelő alkalmazást, majd egy egyszerű telepítéssel telepíteni az alkalmazást.

Linux esetén pedig a már megszokott 'sudo apt-get install dokcer.io' paranccsal tudjuk feltelepíteni az alkalmazást.

Fontos, hogy a telepítés befejezése után szükséges egy újraindítás, vagy legalább egy kijelentkezés.

Docker beépülők

A Docker telepítés után települ a gépre kettő fontosabb modul, ami a Docker CLI és a Docker Daemon.

Docker Daemon

Ez a beépülő játszik szerepet a CLI-től érkezett parancsot értelmezésére és végrehajtására, ez mellett ő van kapcsolatba a Docker HUB-al ahonnan már meglévő képfájlokat tud letölteni és abban az esetben le is fog tölteni, ha nem elérhető az Daemon cache-ben a telepíteni kívánt képfájl.

Ez a Daemon fogja telepíteni a megadott képfájl alapján a konténert és igényel erőforrást az operációs rendszertől.

Későbbiekben, ha bármi már egyszer lefutott kérés érkezik a Daemon irányába, akkor gyorsabb lesz a lefutása, mert már a helyi cache-ből fog gazdálkodni a folyamat.

Docker CLI

Ez az operációs rendszer függvényében telepített parancssor/terminálba épül be és ennek segítség tudunk kiadni Docker parancsokat, amit közvetlenül a Docker Daemon fogadja-értelmezi és hajtja végre.

Konténer telepítése lehetőségek

Docker Compose

Ezzel az egyik módszerrel lehet fájlok segítségével telepíteni a konténereket. Szükséges létrehozunk egy docker-compose.yml fájlt, aminek a tartalma a következők lehetnek.

Fontos, hogy a docker-compose.yml fájlban a különböző részeket, TAB-al vagy SPACE karakterrel kell eltolnunk.

```
*docker-compose - Jegyzettömb
Fájl Szerkesztés Formátum Nézet Súgó
version: "3"

services:

  postgresdb:
    image: postgres:11
    restart: always
    ports:
      - 5432:5432
    container_name: "pg-db-11"
    volumes:
      - C:/docker/data:/var/lib/postgresql/data
    env_file:
      - POSTGRES.env
```

version:

Ezzel meghatározzuk a Docker motorunk verziószámát, hogy milyen verziószámmal legyen kompatibilis az adott .yml fájl, ez azért fontos, mert vannak olyan műveletek, amit bizonyos verziós számú Docker-ek nem támogatnak, míg másik igen.

Részletesebben a Docker verziókról: <https://docs.docker.com/compose/compose-file/>

services:

Itt soroljuk fel, hogy milyen szolgáltatásokat telepítünk majd a megadott .yml fájl segítségével.

Jelen esetben egy postgresdb, definiáltunk szolgáltatásnak, de ez mellett van lehetőség több szolgáltatást is definiálni, de ezeket azonos sorbehúzással szükséges írni.

image:

Ezzel a résszel mondjuk meg, hogy a helyi gépről vagy a Docker hub-ról milyen képfájlt szeretnénk telepíteni és a „:” (kettőspont) után mondjuk meg, hogy az adott képfájlból melyik verzió számot akarjuk telepíteni.

restart:

Ezzel a paranccsal tudjuk definiálni, hogy egy esetleges gazdagép újraindítás után mit tegyen a már meglévő konténerünk, a példa esetében, ha a gazdagép újraindul akkor automatikus elindul a konténer is.

ports:

A docker-compose fájlban itt tudjuk megadni, hogy az adott konténer, milyen porton legyen elérhető a többi konténer számára, illetve, hogy a külső felhasználó, milyen porton tudja elérni az adott szolgáltatást. Természetesen több portot is megadhatunk.

Az első port szám azt jelöli, hogy a host melyik portot nyissa ki a konténer irányába, a „:” után rész, meg, hogy a konténer melyik portját nyissa ki a host irányába.

container_name:

Ez opcionális, hogy későbbiekben a rendszerüzemeltető, milyen néven tud hivatkozni, vagy keresni az adott konténert.

volumes:

Ezzel a paranccsal tudjuk beállítani, hogy bizonyos konfigurációs adatokat, vagy fontos adatokat a konténer honnan másolja át, vagy működése közben hova másolja az adatokat.

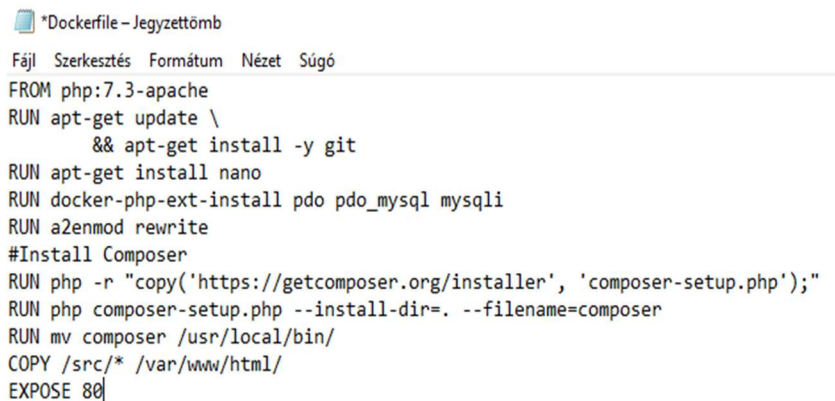
Ebben az esetben nem tárolódnak kettő helyen az adatok, ez csak egy hivatkozás a konténer számára, hogy hova tegye ki az adatokat, vagy honnan olvassa fel.

env_file:

Itt tudjuk megadni a feltelepülő konténernek, hogy milyen környezeti változókat állítson be az adott konténer

Dockerfile

A Dockerfile egy kiterjesztés nélküli fájl, amiben egy adott alap képfájlból indulunk ki és számunkra megfelelő komponenseket telepítünk fel, az adott képfájl nyelvének megfelelően.



```
*Dockerfile - Jegyzettömb
Fájl Szerkesztés Formátum Nézet Súgó
FROM php:7.3-apache
RUN apt-get update \
    && apt-get install -y git
RUN apt-get install nano
RUN docker-php-ext-install pdo pdo_mysql mysqli
RUN a2enmod rewrite
#Install Composer
RUN php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
RUN php composer-setup.php --install-dir=. --filename=composer
RUN mv composer /usr/local/bin/
COPY /src/* /var/www/html/
EXPOSE 80
```

From

Ebben az esetben is van egy Base image, amiből ki kell indulnunk, ezt szintúgy, mint a docker-compose.yml fájlban is definiálva volt egy alap képfájl.

RUN

Itt a kiválasztott Base imagenek, script nyelvének megfelelően tudjuk telepíteni a számunkra fontos alkalmazásokat, jelen esetben, git (revízió követő) nano (szövegszerkesztő) pdo (adatbázis kezelő), composer (függőség kezelő a php-hoz).

Ez a parancs a végleges képfájl létrehozása előtt fognak lefutni, az első parancsok lefuttatását tudjuk itt megadni.

COPY

A host gépről át tudjuk másolni a konténer megadott helyére a különböző meglévő forrás fájlainkat.

WORKDIR

Ezzel a paranccsal tudjuk definiálni, hogy a konténereb belül melyik könyvtárban dolgozunk, ez azért fontos, mert így a „COPY innen ide” parancsból az 'ide' rész elhagyható, mert a Docker mindent a WORKDIR könyvtárba fog másolni, ha nem adunk meg célt a másolásnak.

EXPOSE

Itt tudjuk definiálni, hogy az adott konténert a külső felhasználó hol tudja elérni, milyen port segítségével.

CMD

Ebben a részben tudjuk azokat a parancsokat definiálni, amik a konténer létrehozása után fognak lefutni.

Elindítás

Ezt a Dockerfile-t a „docker build” paranccsal tudjuk lefuttatni, ha abban a mappában vagyunk, ahol található a Dockerfile akkor csak egy „.” (pontot) kell a parancs után tennünk és a megadott paraméterekkel feltelepíteni a mi specifikációnknak megfelelő konténert.

A képfájlok létrehozása közben a közbülső képfájlokat a végleges képfájl elkészítése után törli a Docker.

Ezután a docker run (képfájl neve) paranccsal el tudjuk indítani az elkészült képfájlt.

Build cache

A képfájlok építése közben a Docker mind a Dockerfile és docker-compose fájl esetén a végleges képfájl elkészítése közben minden lépést cache-el, ez azért előnyös, mivel az egyszer elkészített konténerünket, vélhetően újra fogjuk építeni vagy fognak hasonlóképpen épülni és ebben az esetben a Dockernek nem szükséges az internetre csatlakozni, hanem a helyi cache-ből létre tudja hozni a képfájlt. Azonban, ha a Dockerfile vagy a docker-compose fájlba besúruunk egy új parancsot, és előtte már lefutott az a fájl, akkor a Docker csak addig képes cache-ből dolgozni, ameddig nem érkezik az új parancshoz, mivel a lefutása után már egy teljesen új fajta képfájlt hoz létre és a további parancsoknak meg szükséges lefutni az újonnan létre hozott képfájla.

Docker Taggelés

Ez a funkció azért fontos a Docker-ben mert így egy elkészült képfájlt nem a rendszer által generált imageID szerint kell kikeresnünk és elindítanunk, hanem van lehetőségünk elnevezni a végleges képfájlt, amit az elnevezett néven tudunk későbbiekben hivatkozni.

`docker build -t roland.debnarik/docker:latest .`

A Docker és a build paranccsal már foglalkoztunk számunkra a -t paraméter lesz a fontos.

-t

Ennek a paraméterrel jelezzük a Dockernek, hogy egyedi címkét szeretnénk regisztrálni erre a képfájla, a parancs után meg kell adnunk a docker.com-ra regisztrált nevünket, majd „/” után a projekt nevét, majd a „:” után elláthatjuk verzió számmal a képfájlt, vagy a „latest” paranccsal jelezzük, hogy ez az utolsó verzió az adott képfájlból, a „.” pedig a Dockerfile-ra utal, hogy abból építsen képfájlt.

A Docker build parancs után elég kiadnunk roland.debnarik/docker:latest parancsot (latest esetén nem szükséges utána írni, mert alapértelmezetten azt keresni) és így el is fogja készíteni a Docker a képfájl alapján a konténerünket, és ha ki listázzuk a konténereket, akkor már a képfájlnál az elnevezett neve fog megjelenni.

Parancsok

Program parancsok felépítése

Mint a legtöbb programban itt is minden szónak és szókapcsolatban fontos jelentősége van az értelmezés és lefutás miatt.

Egy ilyen például a „docker run busybox ls”

Ez a parancs egy konténert fog készíteni, amit a docker hub-ról másol le, ha még nem szerepelne a Daemon cache-be.

A parancs összetétele:

docker

Ezzel a paranccsal az operációs rendszer segítségével a docker programot szólítjuk meg, ezzel fogunk dolgozni.

run

Ez a parancs, egy Docker specifikus parancs, hogy futtasson le valamit, ami „utána” következik.

create

A run command az egy összevont parancs, mert az magában foglalja a create parancsot, ami létrehozza a konténert a már kapott erőforrásokon.

start

Ez a parancs a create parancs után fut le, miután létre jött a konténer, akkor lefut a start parancs is.

Ebben az esetben, ha bármilyen parancsot futtatunk, nem fogja kiírni számunkra a kimenetet.

Abban az esetben, ha a docker run parancsot adtuk ki akkor a start parancs után a „-a” paraméter is bekerül, ami visszaadja a konténer által küldött üzenetet a terminálunkra.

busybox

Ez magának a képfájlnak a neve, ezt nem szükséges ismernie a helyi Docker programunknak, elég, ha a Docker Daemon elő tudja keresni a cache-ből, vagy a Docker Hub-on van ilyen bejegyzés.

/s

Ha a képfájl után adunk ki parancsok, akkor az a parancs már magán a telepített konténer rendszerén belül fog lefutni, jelen esetben a busybox által telepített Linux fájlrendszer gyökér könyvtárát fogja kilistázni.

Docker parancsok

Docker ps

Ennek a programnak a segítségével ki tudjuk listázni, az éppen futó konténereket és tudjuk látni a fontosabb információkat róla.

-a

Ha az előbb említett parancs után írjuk a „-a” kapcsolót, akkor minden olyan konténert ki fog listázni, ami az adott operációs rendszer indítás óta elindult, vagy leállt konténer.

Docker rm

Ezzel a paranccsal tudjuk a már meglévő futó vagy leállított konténereinket törölni, a parancs után a cointanerID-t kell megadni, amit a „docker ps” paranccsal ki tudjuk listázni. Vigyázat, ebben az esetben minden törlődik a konténerből és nincs lehetőség visszaállítani, ez alól kivétel a volumes, mert akkor az adatok megmaradnak.

Docker container prune

Ez a parancs az összes futó és leállított konténereket törölni fogja, így nem kell egyesével törölni őket.

Docker system prune

Ezzel a paranccsal ki tudjuk üríteni a teljes Docker által kezelt képfájlokat és konténereket.

Docker start

Ezzel a paranccsal el tudjuk indítani a létrehozott, vagy már lefutott konténerünket, a konténertől nem fogunk ebben az esetben visszajelzést kapni, mert ebben az esetben is szükséges a „-a” paraméter.

Ha már egy létrehozott konténert akarunk elindítani akkor már nincs lehetőségünk a konténernek parancsot átadni, az elsőnek létrehozott indító parancs fog rajta lefutni, ez az újrahasznosítás hátránya.

Docker logs

Ezzel a paranccsal megkapunk minden olyan adatot, amit az output csatornára írt a konténer az indulása óta a parancs lefutásáig.

Docker stop

Ezzel a paranccsal le tudjuk állítani a kiválasztott konténerünket, a parancs után meg kell adni a konténer ID-t.

Ez egy barátságosabb leállítási módszer a Dockerek számára, mert itt az operációs rendszer kezdeményezi a rendes leállítást a konténeren, amennyiben nem tud a konténer leállni a parancs kiadása után 10 másodperccel, akkor a Docker a háttérben meghívja a docker kill parancsot.

Docker kill

Ennek a parancs végére ugyan úgy a konténer ID-t kell beírni, amit a docker ps-el tudunk kilistázni.

Ennek a parancsnak a kiadása után az operációs rendszer azonnal megszüntetni a konténer működését, ilyen esetben előfordulhatnak inkonzisztens adatok, ha az adatbázis nem tudta befejezni a tranzakciókat.

Docker exec

Ennek a parancsnak a paraméterezése a következő „docker exec konténerID” új parancs

Ezzel a parancssal a már futó konténerünkbe tudunk plusz parancsokat/programokat bejuttatni,

Docker exec -it

Ezzel a plusz kapcsolóval nem csak parancsokat tudunk bejuttatni a konténerbe, hanem ezzel a kapcsolóval jelezzük a Dockernek, hogy csatlakoztasson minket rá arra a parancsra, amit végrehajtunk az adott konténeren, így interakcióba léphetünk a konténerrel és esetleges inputokat is meg tudunk adni.

-i ennek segítségével köt rá a Docker az adott parancsra.

-t ez csak a formázott szintaktikáért felelős.