Konténer alapú virtualizáció Docker segítségével

Cloud Computing

A felhő alapú számítástechnika (Cloud Computing) létrejötte és rohamos fejlődése forradalmasította az informatikai szolgáltatások üzemeltetését. A Cloud Computing a 2010-es évektől kiemelkedő területe a számítástechnikának. Fontos előnye a számítási kapacitás igény szerinti elérhetősége, valamint a felhőszolgáltató által biztosított adattárolási lehetőségek és számítási teljesítmény igénybevétele a hardverek közvetlen, felhasználó általi karbantartása nélkül. Infrastrukturális szinten a felhőszolgáltatók virtuális gépeket bocsátanak rendelkezésre, és támogatják ezek későbbi, dinamikus skálázását (pl. memória és tárhely mennyiségének növelése).

A felhő alapú számítástechnika lehetővé teszi a vállalatok számára, hogy minimalizálják az IT-infrastruktúra felépítésének kezdeti költségeit, valamint alkalmazásaikat gyorsabban üzembe helyezzék, jobb kezelhetőség és kevesebb karbantartás mellett.

Lehetővé teszi továbbá, hogy a szolgáltatók valamely publikus felhőszolgáltatás felhasználásával gyorsan adaptálják saját erőforrásaikat az ingadozó, vagy csak időszakosan kiugró igényekhez (pl. Black Friday egy webshopnál, sorozat premier egy streaming szolgáltatónál). Ezt „cloud bursting”-nek is nevezzük.

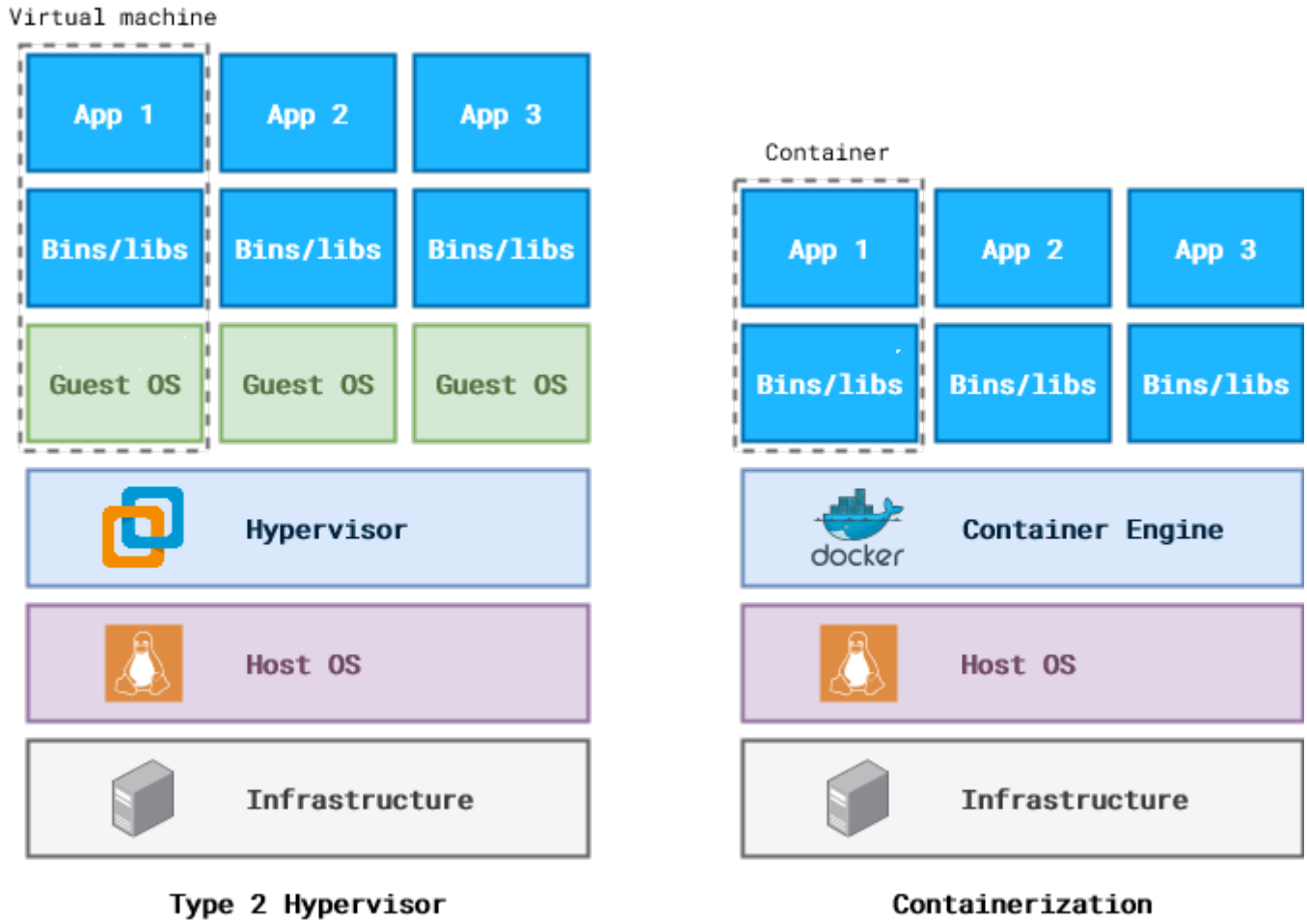
Konténerek

A natív felhő alapú számítástechnika (Cloud Native Computing) egy olyan modern szoftverfejlesztési megközelítés, amely a felhőt használja fel jól skálázható alkalmazások létrehozására és futtatására, dinamikus üzemeltetési környezetekben. Ennek a megközelítésnek gyakori elemei az olyan technológiák, mint a konténerek, mikroszolgáltatások, „serverless” függvények és a deklaratív kódon keresztül telepített infrastruktúra (Infrastructure as Code).

A natív felhő alapú alkalmazások gyakran Docker-tárolókban futó konténerek által nyújtott mikroszolgáltatásokból épülnek fel, amelyeket Kubernetes-ben irányítanak, és CI/CD, valamint DevOps munkafolyamatok segítségével telepítenek és üzemeltetnek.

A **konténerizáció** (containerization) jelentős hatást gyakorol a natív felhő alapú számítástechnikára, mert önálló telepítési egységek szabványosított formában történő létrehozásának lehetőségét biztosítja, továbbá energia-, költség-, erőforrás- és tárhely-hatékony, emellett a hagyományos virtuális gépekétől jelentősen gyorsabb rendszerindítást tesz lehetővé. Ezen tulajdonságai megkönnyítik a terheléselosztást, a rendszerkarbantartást, valamint a konténerek földrajzi régiók közötti replikációját a jobb hibatűrés és az alkalmazások megbízhatóságának növelése érdekében.

A konténer technológia, mint OS-szintű virtualizáció jelen van, és egyre inkább teret nyer a modern szoftverek üzemeltetésében. A számítási felhő platformok következő generációja nem a hardverek, hanem az alkalmazások virtualizációján alapul.



Type 2 Hypervisor — Containerization

A klasszikus, |hypervisor alapú virtuális gépek úgy működnek, hogy a hypervisor több virtuális gépet (VM) felügyel, melyek egyenként külön operációs rendszert futtatnak. A konténerek esetében minden alkalmazás ugyanazon az operációs rendszeren fut, viszont konténerenként külön-külön, elszeparált környezetekben. A konténerek között tehát processz szintű izoláció valósul meg, melyet a hoszt gép kernele biztosít.

Mivel egy-egy alkalmazás elindításakor nem kell a kernel elindulására várni, a rendszerindítási folyamat sokkal gyorsabb, mint a hagyományos VM-ek esetében. Emellett az erőforrás kihasználás jelentősen jobb (közel natív teljesítmény elérésére van lehetőség), mint a virtuális gépeknél.

A konténer technológia további előnye, hogy az alkalmazáskörnyezetet kisméretű önálló telepítési egységekbe, „képfájlokba” szervezik. Ezek a képfájlok kis méretükből adódóan előállhatnak akár automatizáltan, egy CI/CD folyamat eredményeképpen is. Az elkészült képfájl módosítások nélkül azonnal futtatható megfelelő konténer motor segítségével.

Ennek jelentősége van a szoftverfejlesztés során is, a Docker Hub-on közel 1 millió konténer-képfájl található, melyek egy jelentős része a fejlesztéshez is használt, sokszor komplex módon üzemeltethető eszközöket (pl. adatbázisokat, gyorsítótárakat, webszervereket) tartalmaz, melyek így egyszerűen, bonyolultabb telepítési lépések nélkül indíthatók és távolíthatók el a fejlesztő gépéről.

A konténerek emellett kisebb tár- és memóriaigénnyel rendelkeznek, mint a VM-ek, emiatt könnyebben lehet őket alkalmazni multi-cloud környezetben, ahol a szolgáltatás egyszerre több felhőben (pl. céges privátfelhő, Google Cloud Platform, Microsoft Azure, Amazon Web Services) van üzemeltetve. Jól alkalmazhatók továbbá cloud bursting során, amikor a megnövekvő igények miatt a

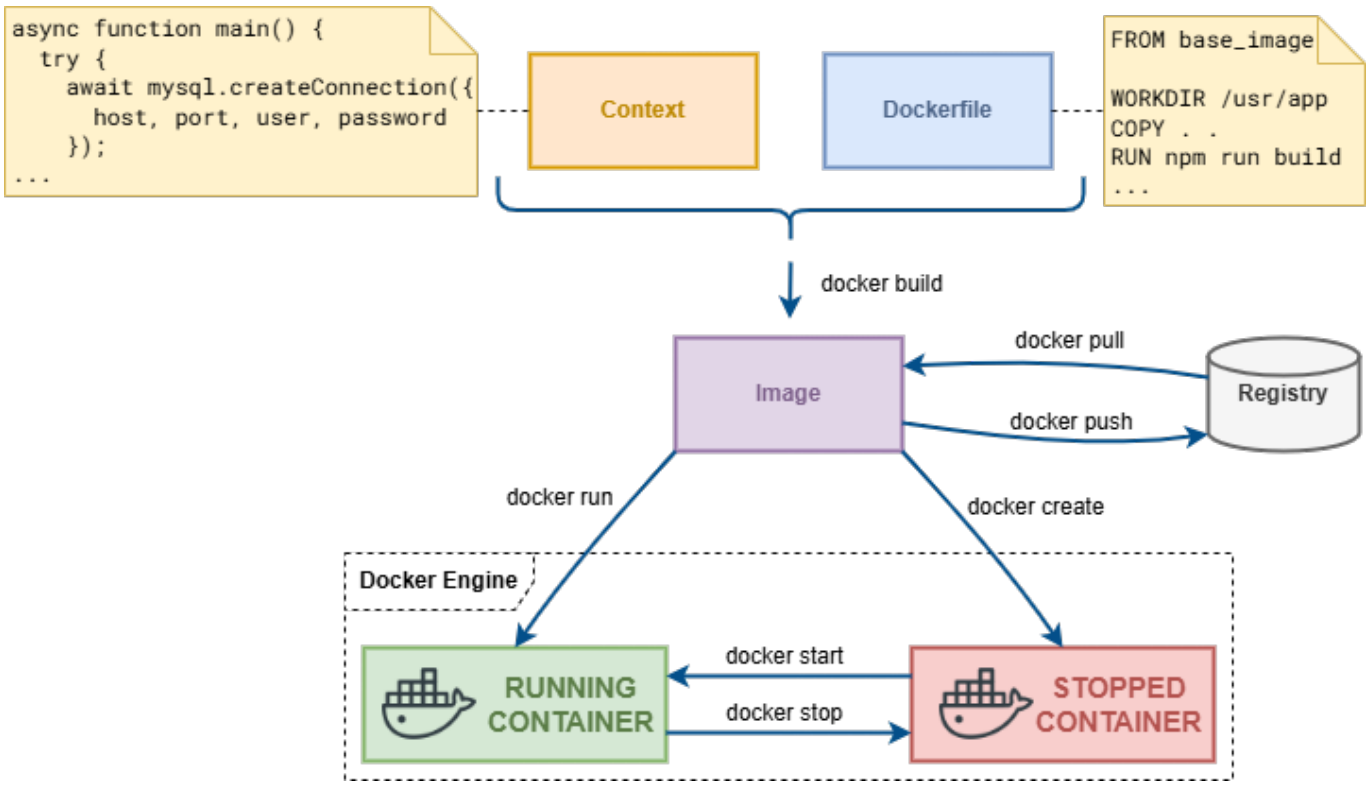
saját infrastruktúra mellett igénybe kell venni egy publikus felhőszolgáltatást is az üzemeltetett alkalmazás számára.

A konténerek számos előnnyel rendelkeznek, de architektúrájukból adódóan sebezhetőbb megoldásnak számítanak, mint a klasszikus virtuális gépek, ugyanis minden konténer egyetlen kernelen fut, és csupán ennek a kernelnek (vagy a konténer motornak) a hibáiból adódóan (Single Point of Failure) előfordulhat nem kívánt adatszivárgás a konténerek között.

Docker

Napjainkban a Docker platform de-facto standardnak számít a konténerizáció gyakorlati megvalósításában. Segítségével az alkalmazásokat egyszerűen csomagolhatjuk konténerekbe, majd telepíthetjük őket szinte bármilyen számítógépre vagy szerverre.

Architektúra



A Docker architektúrális felépítését, valamint legfontosabb parancsait a fenti ábra mutatja be.

A Docker platform építőelemei a következők:

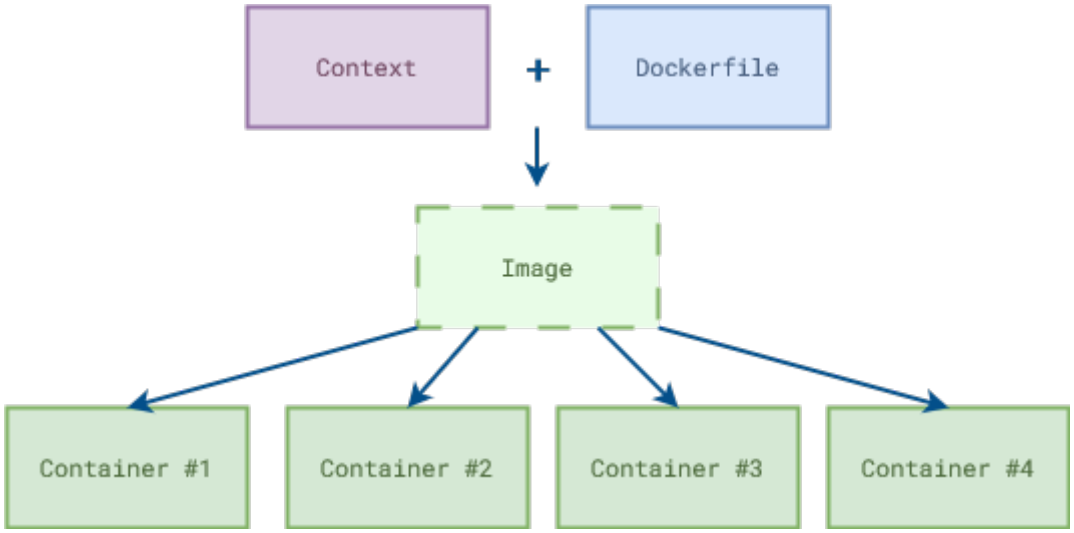
- **Image** (képfájl): Állapotmentes, futtatható egység, amely tartalmazza az alkalmazást és annak összes szükséges függőségét, konfigurációját, akár forráskódját. A képfájl tulajdonképpen az alkalmazás-környezet statikus kezdőállapota. Docker képfájlok kontextus és Dockerfile alapján építhetőek.
  - **Kontextus**: Alkalmazásunk forráskódja és függőségei. Minden olyan fájl, információ, és egyéb erőforrás, melyre szükség van a szoftverünk működéséhez.
  - **Dockerfile**: A Docker-képfájl létrehozásához szükséges parancsokat tartalmazó fájl.

- **Container** (konténer): A Docker-képfájl egy éppen futó példánya. Minden konténer rendelkezik képfájllal. Egy képfájl több példányban (azaz több konténerként) is elindítható.
- **Registry** (tároló): Előre elkészített Docker-képfájlokat tároló szerver. Lehet publikus (pl. Docker Hub) vagy privát (pl. céges) elérésű. A registry-be feltölthetők (`push`), illetve onnan letölthetők (`pull`) a képfájlok.

Konténerek és képfájlok

A Docker konténerek alapjául szolgáló képfájl a kontextus és egy Dockerfile segítségével állítható elő.

A képfájl a konténer statikus váza, egy recept arra vonatkozóan, hogy hogyan tudjuk a konténert létrehozni és elindítani.



Egy képfájlból több konténer is létrehozható, például abban az esetben lehet erre szükség, ha alkalmazásunk válaszidejét csökkenteni szeretnénk, például az alábbi módok valamelyikén:

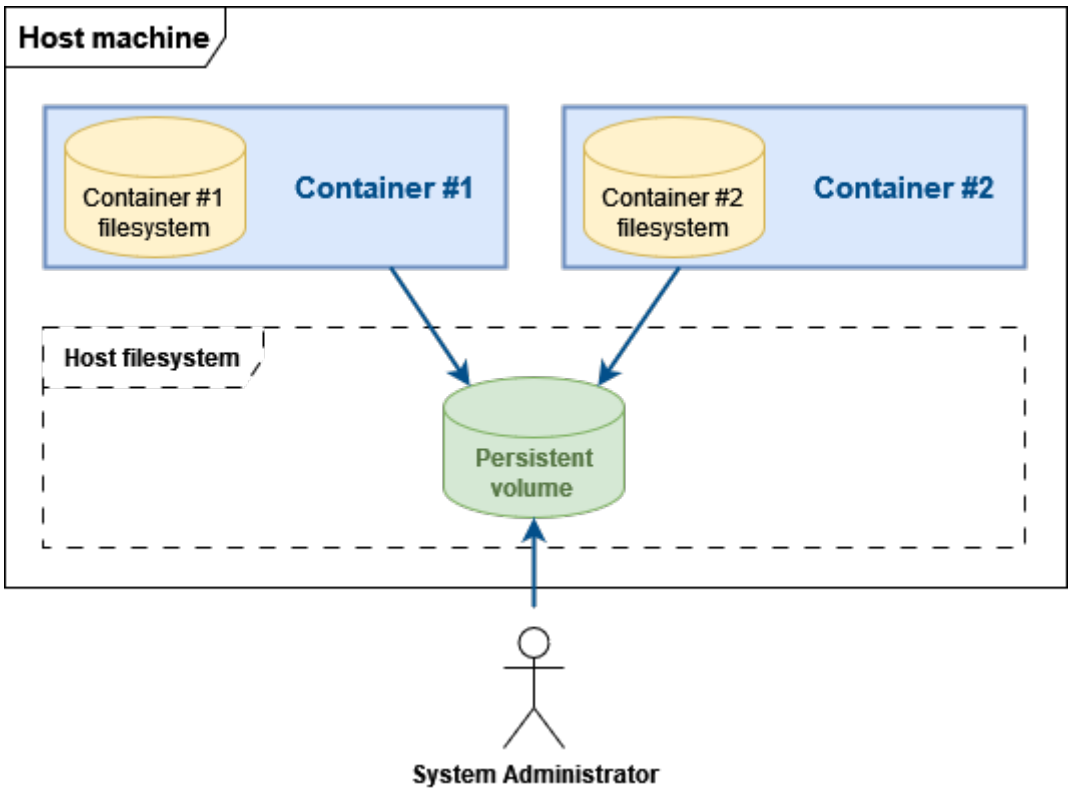
- Adott szerverközpontban több konténerben indítjuk el ugyanazt az alkalmazást, a kliensek kéréseit egy load balancer mindig a legkevésbé leterhelt konténerhez irányítja.
- Ugyanazt az alkalmazást több szerverközpontban indítjuk el (pl. egyet Európában, egyet az USA-ban), majd a kliensek kéréseit egy load balancer mindig a geográfiailag legközelebbi konténerhez irányítja.

Perzisztens tárolók

A konténereket alapvetően „állapotmentes” működésre tervezték, ez azt is jelenti, hogy bármikor eldobhatónak és egy kezdeti, tiszta állapotból újraindíthatónak kell lenniük, anélkül, hogy ez a rendszer egészére komoly hatást gyakorolna. Nem célszerű tehát, ha olyan állapotot tartalmaznak, amit fontos lenne hosszabb ideig megőrizni.

A fentiek alapján érezhetjük, hogy ez a fajta állapotmentesség egyes alkalmazások, pl. adatbázisok esetében nem állja meg a helyét, hiszen ezek célja éppen az, hogy a tárolt adatokat hosszabb távon, perzisztens módon megőrződjenek és szükség esetén hozzáférhetőek legyenek.

Ennek a problémának a megoldása miatt van lehetőségünk ún. **perzisztens tárolók** (volume-ok) létrehozására. A perzisztens tárolók nem a konténer virtualizált – és kívülről közvetlenül elérhetetlen - fájlrendszerén, hanem közvetlenül a hoszt gépen kerülnek tárolásra, ahogy az az alábbi ábrán is látható.



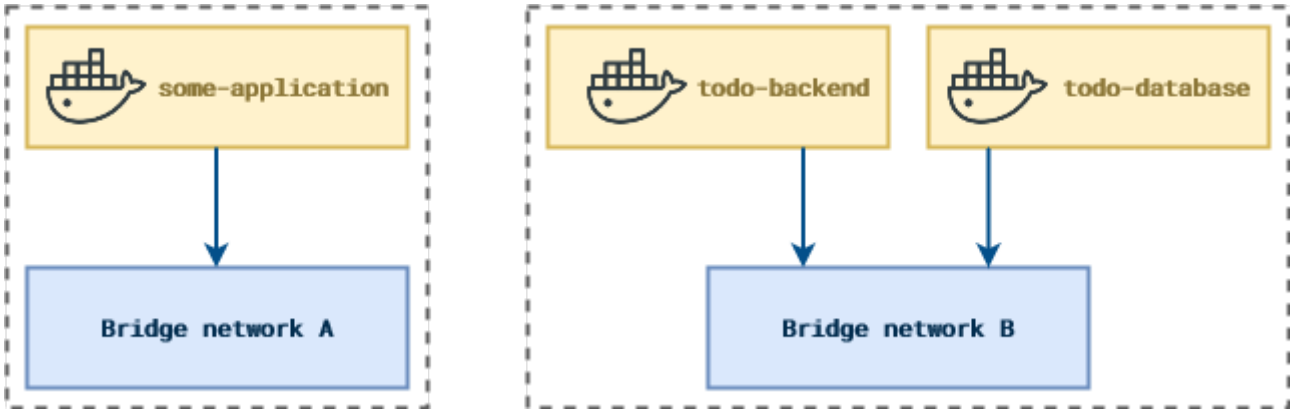
Perzisztens tárolókat nevük és opcionálisan egyes jellemzőik (pl. méretük) megadásával hozhatunk létre. A tároló nevének egyedinek kell lennie.

A konténer elindításakor meghatározhatjuk, hogy a konténer fájlrendszerének mely könyvtárát rendeljük hozzá a tárolóhoz. Így ezek a fájlok nem a konténer, hanem a hoszt gép fájlrendszerében lesznek tárolva, innen lesznek elérhetők és módosíthatók. Ebben az esetben a tárolt fájlokhoz nem csak a konténerek, hanem a hoszt gép felhasználói is hozzáférhetnek. Ez egyebek mellett előnyös lehet például biztonsági mentések készítésekor is.

Ezt a folyamatot a Docker Engine kezeli, a konténer számára teljesen transzparens módon.

Hálózat virtualizáció

Az alkalmazások gyakran több konténerre oszthatóak, ezek a konténerek a hálózaton keresztül képesek kommunikálni egymással. A Docker többféle virtualizált hálózatkezelési módszert támogat, melyek közül leggyakrabban bridge network-öket használnak.



Ahogy az a képen is látható, az azonos hálózathoz hozzákapcsolt konténerek (`todo-backend`, `todo-database`) egymással kommunikálni tudnak, azonban a különböző hálózaton lévő konténerek (pl. a `some-application` és a `todo-backend`) nem érhetik el egymást. Egy konténer egyébként akár több hálózathoz is kapcsolódhat.

Amennyiben az elindított konténert nem kapcsoljuk hálózathoz, az alapértelmezett bridge hálózathoz fog tartozni. Fontos megjegyezni, hogy ezen az alapértelmezett hálózaton a konténerek kizárólag közvetlenül, IP-cím alapján érhetik el egymást, a DNS szolgáltatás nem biztosított számukra (a DNS a névfeloldásért szerepel, pl. az `uni-miskolc.hu` domainből előállítja a `193.6.10.2` IP-címet, ezt ki is lehet próbálni, pl. itt: https://toolbox.googleapps.com/apps/dig/#A/).

A DNS szolgáltatás hiánya bonyolult feladattá teszi a konténerek kommunikációjának megvalósítását, hiszen például egy újraindított konténer esetében semmi nem garantálja azt, hogy újra a korábbi IP-címét kapja meg a hálózaton.

Amennyiben saját hálózatot („user-defined bridge”) hozunk létre, azon belül a névfeloldás automatikusan biztosított, az egyes konténerekre konkrét IP-címük helyett elegendő elnevezésükkel (pl. `todo-database`) hivatkoznunk a hálózati kommunikáció során, így az esetleges IP-cím váltás sem okoz gondot az alkalmazás üzemeltetésében.

Feladatok

- Flask szerver és Redis cache beüzemelése
- Terhelés elosztás és monitorozás HAProxy segítségével
- Todo webalkalmazás beüzemelése + Forráskód

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:informatikai_rendszerek_epitese:docker_ismerteto

Last update: **2025/03/26 13:39**