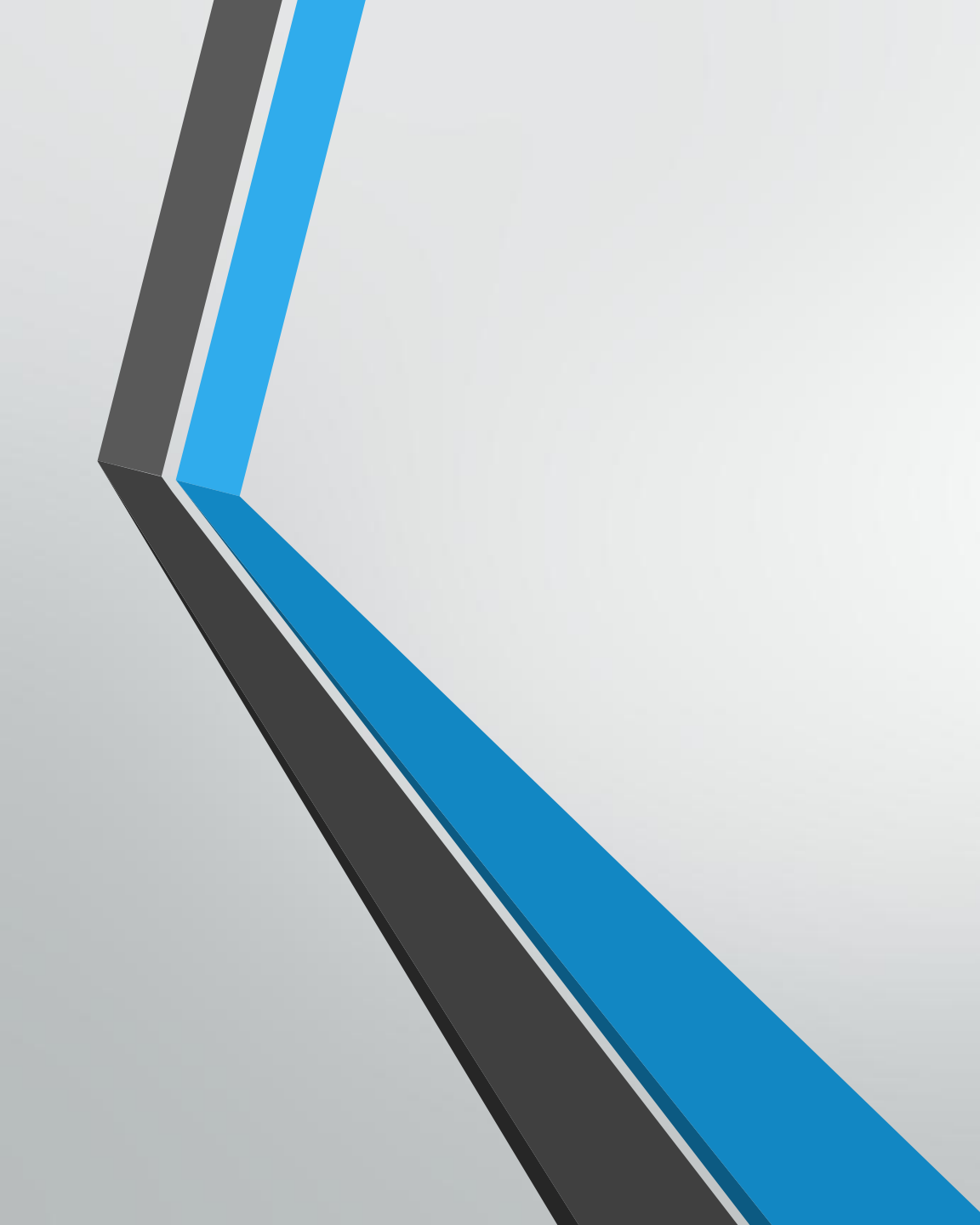




Virtualizációs Technológiák

Operációs rendszer szintű virtualizáció

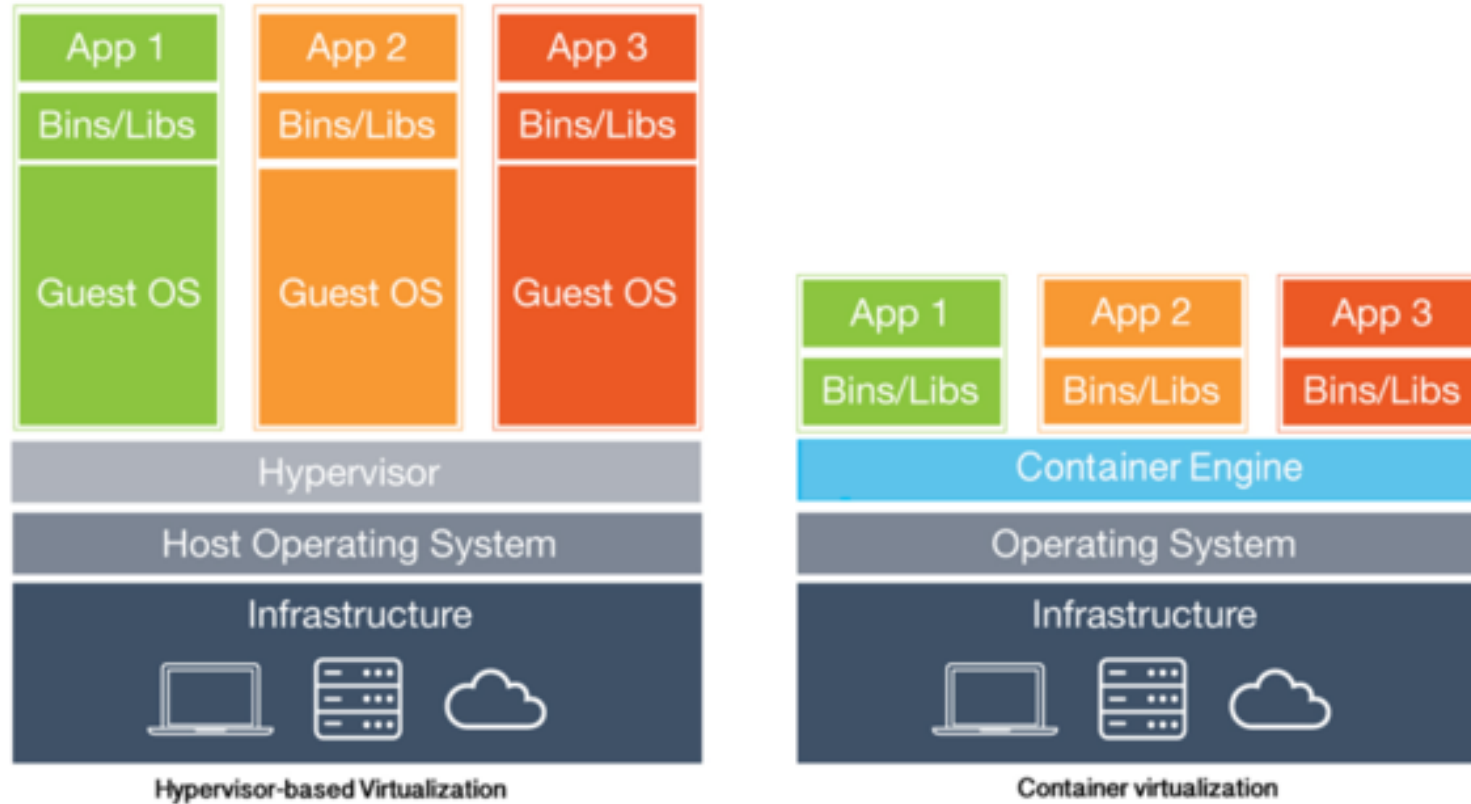
Konténerek

Forrás, BME-VIK

Virtualizációs technológiák

<https://www.vik.bme.hu/kepzes/targyak/VIMIIV89/>

Koncepció



- Ha megfelel, hogy azonos az OS kernel a vendég gépek között
- Ha csak izoláció, erőforrás-korlátozás kell
- nem kell VMM, elég az OS alkalmazás szétválasztása

Nincs új a nap alatt

- Operációs rendszer szintű virtualizáció – más néven konténer alapú virtualizáció
 - BSD-ken: Jail
 - Solaris: Containers, Zones
 - Linux alatt: OpenVZ (Parallels Virtuozzoból a Linux kernel módú részei), Linux VServer
 - AIX: WPAR (workload partitions)
 - Windows: Parallels Virtuozzo

Konténerek

- nagyon kis költségű
 - elvileg nem 0 – csak éppen nem lehet kimérni olyan kicsi 😊
 - konténerenként nincsenek további vendég kernelek
- erőforrás virtualizációs és
- erőforrás gazdálkodási szempontból problémamentes
 - nincs fixen lefoglalt memória, nem kell trükközés ballonozással stb.
 - nincs fixen lefoglalt háttértár – a hoszt fájlrendszere fájl szinten elérhető
- biztonsági szempontból kevésbé jó izoláció
- közös kernellel kell élni (azonos verzió, fordítási paraméterek)

Hypervisor vs. konténerek

- Hypervisor
 - Egy fizikai, számos virtuális hardver
 - Vegyes operációs rendszerek
 - Kisebb sűrűség
 - Kisebb teljesítmény
- OS szintű virtualizáció
 - Egy fizikai hardver, nincs virtuális
 - Egy kernel, userspace példányok
 - Nagy sűrűség
 - Közel natív teljesítmény

Réges-rég egy messzi-messzi galaxisban

- UML (User Mode Linux) 2.2.0-ás kernelről !!!!!!!!!
 - Külön rendszer létrehozása debootstrappal
 - Nem privilegizált linux processzként fut
- Jail/chroot
 - Egy adott processz izolálása a rendszertől
 - Minden amire szükség lehet újra létrehozni egy külön könyvtárban
 - Chroot -> a könyvtárban való
- Konténer = egy szimpla jail szteroidokkal ☺

Megvalósítás

- Kezdetben volt a *chroot*...
 - A fájlrendszer gyökerét átirányíthatjuk egy alkönyvtárba (egy processzre vonatkozik!)
 - Cél:
 - Általában: az abszolút fájl elérési útvonalak módosítása nélkül tudunk több példányt fenntartani egy-egy fájlból
 - Kicsit „antipattern” alkalmazás: biztonság növelése
 - Ez nem teljes körű izoláció, de sok esetben működik
 - Kernel minden adatszerkezete közös (processz lista, hálózati interfész, IP, routing, sysctl beállítások...)
 - A chrootból ráadásul ki is lehet navigálni a VFS adatszerkezeten keresztül...
 - Hogy is néz ki:
 - egy teljes alap OS installációt készítünk egy alkönyvtárba, ami kicsit eltérő is lehet az eredetitől
 - Általában véve a programkönyvtáraknak, konfigurációs fájloknak meg kell lenniük a chroot-on belül is
 - Problémás globális könyvtárak: /proc, /sys, /dev, /tmp, /var, ...
 - Lehet mount binddal trükközni, de nem lesz tökéletes...

Alkotóelemek

- Kernel
 - Névterek (namespaces): virtualizáció és izoláció (külön hálózat, user ID, mountpoint-ok)
 - Cgroups: erőforrás menedzsment (erőforráslimit adható meg)
 - Checkpoint/Restart: live migration
- Userspace eszközök
 - Konténer menedzsment parancssori eszközök
- OS sablonok

Erőforrások

- CPU – a kernel beépített ütemezője, prioritáskezelője
- Memória – a kernel beépített memóriakezelője, *rlimit*-tel megadható maximális foglalások)
- Háttértár – a fájlrendszer egy alkönyvtára, *quota* rendszerrel korlátozható foglalás
- Hálózat – a kernel beépített Ethernet hídja vagy routing táblája, pl. IPtables QoS paraméterekkel korlátozható
- Egyéb perifériák – a kernelben lévő meghajtón keresztül

Erőforrás-gazdálkodás

- Azt szeretnénk, ha a konténerre vonatkozna a korlátozás, nem pedig az egyes folyamatokra
- Hierarchikus erőforrás foglалás:
 - A folyamatok fa hierarchiába szervezettek (nemcsak Unix, Windows alatt is)
 - Ün. Beancounter rendszer, ami csoportok szerint összesíti a foglалást
- A memó riafoglалás elég sokrétű probléma:
 - Alkalmazás virtuális/fizikai memó rialapok
 - Cache lapok
 - Pufferek (socketek, hálózati kapcsolatok)
 - Megosztott lapok „igazságos” költségelszámolása
 - Néhány alkalmazás elég „zűrös” memó riafoglалó (Java VM) – kézi beállítást igényelhet

Checkpointing/Migration

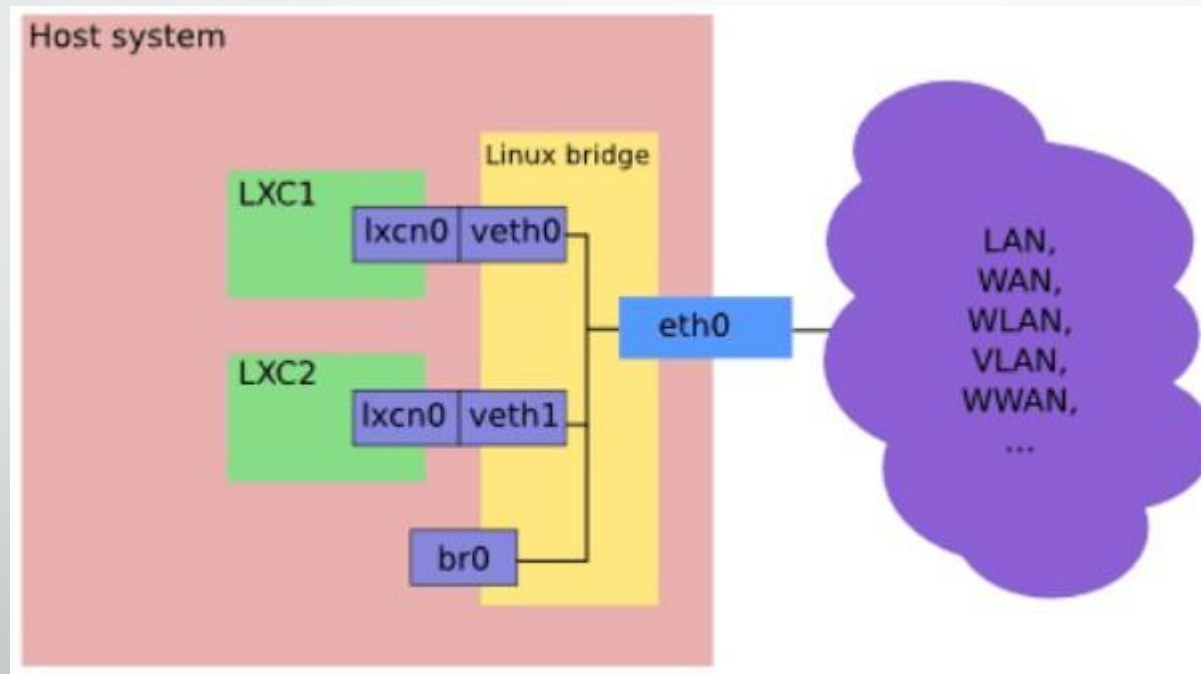
- Teljes konténer állapot
 - Menthető
 - Futó folyamatok
 - Megnyitott fájlok
 - Hálózati kapcsolatok
 - Memória szegmensek
 - Visszaállítható
 - Visszaállítható másik szerveren

Hozzáférés a vendég „gépekhez”

- Szöveges konzol elérés van
- Grafikus felület
 - általában nehéz, mert globális erőforrásokhoz (Unix alatt X Server, socket) igényel hozzáférést
 - Gyakorlatban használt megoldás: távoli elérés szerver indítása a konténerben, tipikusan VNC szerver
- Fájrendszer
 - Konténeren kívülről belátunk a konténer fájlrendszerébe
 - Konténeren belülről nem látunk kifelé
- Folyamatok
 - Mint a fájlrendszernél...

Hálózat

- Pont-pont kapcsolat a konténer és a hoszt között
- A hoszt hálózati interfészei között lehetőség van NAT vagy Bridged jellegű beállításra



Docker

- 2013-ban vált nyílt forrásúvá a Docker nevű konténer platform amely kezdetben a korábban létező LXC (Linux Containers) technológiát használta a konténerek futtatására.
- Később ezt lecserélték egy saját implementációra, de a Docker valódi célja nem ez volt, hanem hogy létrehozzanak egy olyan rendszert amely nagyban leegyszerűsíti a konténerek menedzselését a rendszerüzemeltetők és a végfelhasználók számára.
- Ennek a rendszernek két fő komponense van: egy linux operációs rendszerekre elérhető eszközkészlet és a hozzá tartozó szerverprogram (**docker daemon**) amely a konténerek futtatását végzi, illetve egy web alapú konténer könyvtár (**dockerhub**) ahová a felhasználók által létrehozott konténereket találjuk.

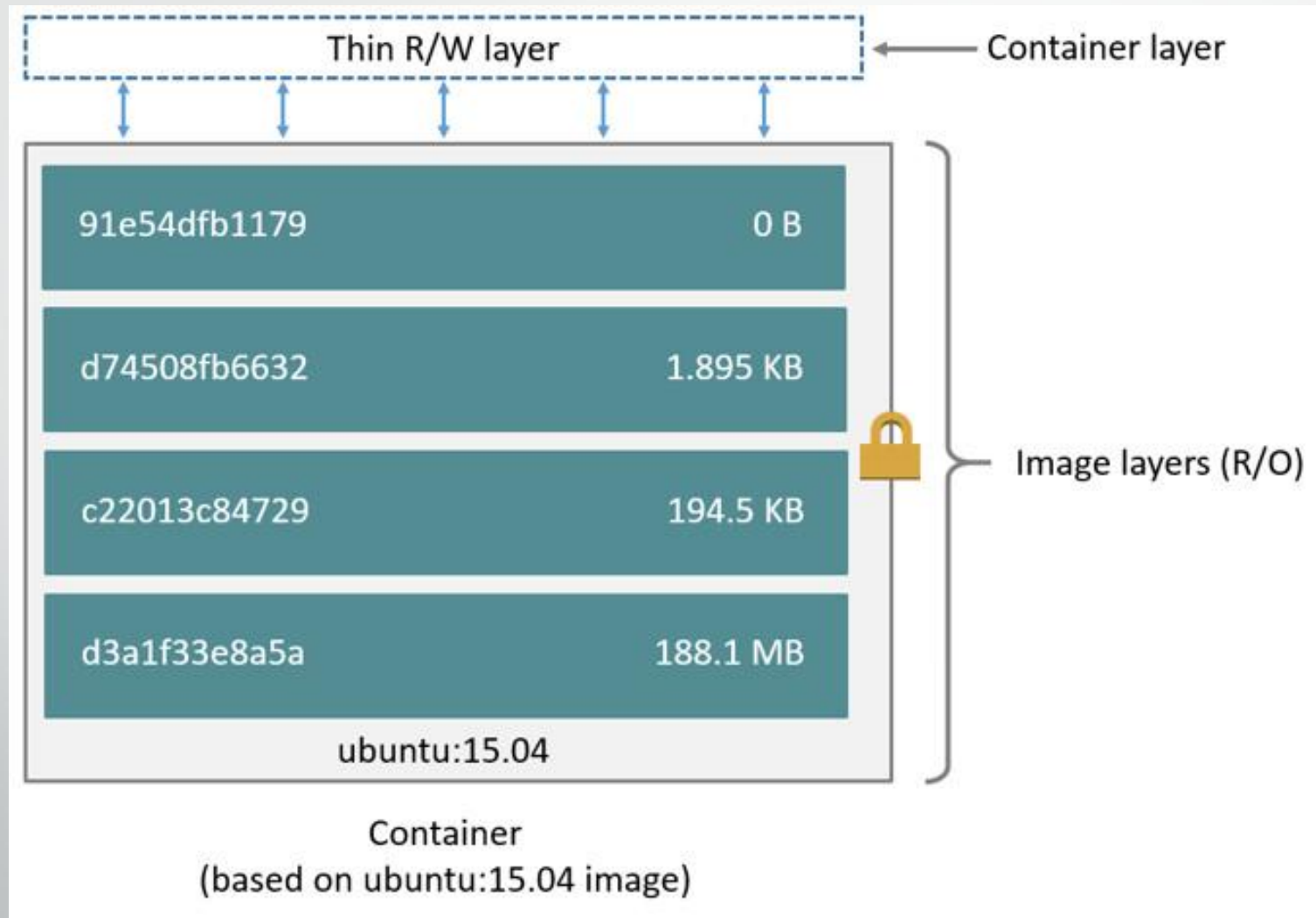
Singularity

- Singularity egy új megközelítése a konténer technológiáknak.
- Futtatásához nincs szükség rendszergazdai jogosultságokra így kiválóan alkalmas megosztott rendszereken való használatra. A kifejezetten HPC (High Performance Computing) környezetbe szánt
- Ez által a speciális gyorsítókártyák és videókártyák valamint a HPC környezetben gyakran használatos interconnect technológiák mint például az Infiniband minden akadály nélkül használhatóak.
- Továbbá a hálózati meghajtója is közvetlenül a fizikai host interfészét használja így nagyobb sebességre képes alapértelmezett beállítások mellett, mint a Docker.
- A dockerhubbal kompatibilis, így a Docker-nek szánt konténerek gond nélkül futtathatóak Singularityvel is.

CGroups

- A control groups-ot a google két mérnöke Paul Menage és Rohit Seth fejlesztette még 2006-ban azzal a céllal, hogy a Linuxon futtatott különböző processzek és folyamatok között megfelelő elszigetelést hozzanak létre. A cgroups a következő főbb feladatokat látja el:
- **Erőforrás limitáció**
 - A group-oknak be lehet állítani memória és processzoridő limiteket
- **Priorizálás**
 - A prioritást élvező group-ok nagyobb részt kaphat processzoridőből és Lemez Írás/Olvasás műveletekből
- **Számvitel**
 - Méri az egyes group-ok erőforráshasználatát amelyet akár későbbi számlázáshoz is fel lehet használni
- **Vezérlés**
 - Megállíthatja, újraindíthatja az egyes processzeket, akár ellenőrzőpontokat is létrehozhat későbbi folytatáshoz

Fájrendszer



Konténer leírófájl

```
FROM ubuntu:16.10 # meghatározzuk a base image-t  
COPY . /app # a fájlrendszerünkből bemásoljuk az app  
mappát  
RUN make /app # elindítjuk az app könyvtárban található  
make fájlt  
CMD python /app/app.py # elindítjuk az app könyvtárban  
található app.py-t
```

“Hátrányok”

- Docker csak egy processzt használ (multiprocess bukta)*
- Több konténer egy szolgáltatáshoz
 - Nginx konténer
 - Mysql konténer
 - PHP konténer stb...
- Nincs login script, SSH daemon, monitoring agent stb.
- Read-only konténer (snapshot jellegű működés)

*: https://docs.docker.com/engine/containers/multi-service_container/

Konténer menedzsment

- Több konténer egy szolgáltatáshoz
 - Docker compose: <https://docs.docker.com/compose/>
 - Konténerek és paramétereik egy leírófájlban: docker-compose.yaml (YAML)
 - docker-compose parancs ami ez alapján készíti/indítja/leállítja

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
  redis:
    image: "redis:alpine"
```

Konténer menedzsment

- Több konténer egy szolgáltatáshoz
- HA, failover stb.
 - Kubernetes <https://kubernetes.io/>
 - 1 szolgáltatás = 1 konténer = 1 POD
 - Node: olyan gép amin futhatnak a podok (VM vagy fizikai vas)
 - Cluster: több Node-ból álló üzemeltetési egység
 - Deployment: 1 futó POD:
 - Automatikus újraindítás
 - HA: legalább x példány mindig fut (node failure-t is kezel)
 - “szabványos” API:
 - több ingyenes és fizetős termék amivel Kubernetes cluster-t lehet kezelni
 - Pl.: RedHat OpenShift
 - Helm (<https://helm.sh/>)



Összefoglalás

- Konténer alapú virtualizáció - eltérő koncepció a platform virtualizációtól
 - Közös kernel
 - Kisebb erőforrás igény
 - Operációs rendszer szintű erőforrásokat virtualizál
 - Jellemzően a host felől nézve a konténer átlátszó, belülről nézve átlátszatlan (!!!!!)
- OpenVZ, Parallels Virtuozzo, LXC, Docker
 - Linux illetve Windows környezetben nagyon hasonló megoldások
 - Konténerek példányosítása sablonokból

Gyakorlat

- <https://gitlab.tilb.sze.hu/teach/virt-tech/docker-example>