

Virtualizációs Technológiák

Operációs rendszer szintű virtualizáció

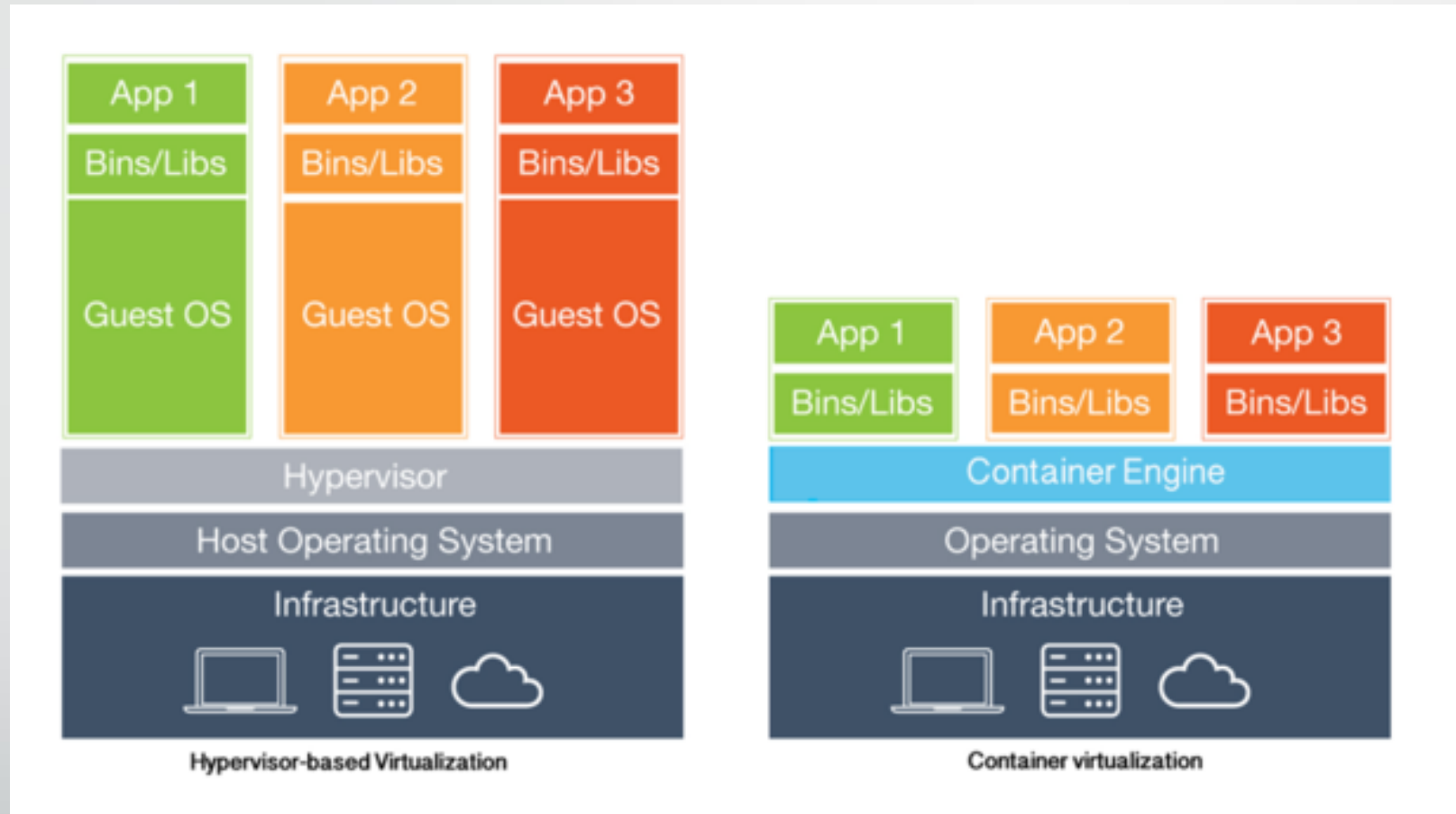
Konténerek

Forrás, BME-VIK

Virtualizációs technológiák

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

Konceptió



- 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 kerneltől !!!!!!!!!!!
 - Külön rendszer létrehozása debootstrappal
 - Nem privilégizált linux processzként fut
- Jail/chroot
 - Egy adott processz izolálása a rendszertől
 - Minden amire szükség lehet újra létrehozni egy külön könyvtárban
 - Chroot -> a könyvtárban voálá
- Konténer = egy szimpla jail szteroidokkal 😊

Megvalósítás

- Kezdetben volt a *chroot*...
 - A fájlrendszer gyökerét átirányíthatjuk egy alkönyvtárra (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: virtualizáció és izoláció
 - Cgroups: erőforrás menedzsment
 - 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)
 - Ún. Beancounter rendszer, ami csoportok szerint összesíti a foglалást
- A memóriafoglалás elég sokrétű probléma:
 - Alkalmazás virtuális/fizikai memórialapok
 - Cache lapok
 - Pufferek (socketek, hálózati kapcsolatok)
 - Megosztott lapok „igazságos” költségelszámolása
 - Néhány alkalmazás elég „zűrös” memóriafoglалó (Java VM) – kézi beállítást igényelhet

Checkpointing/Migration

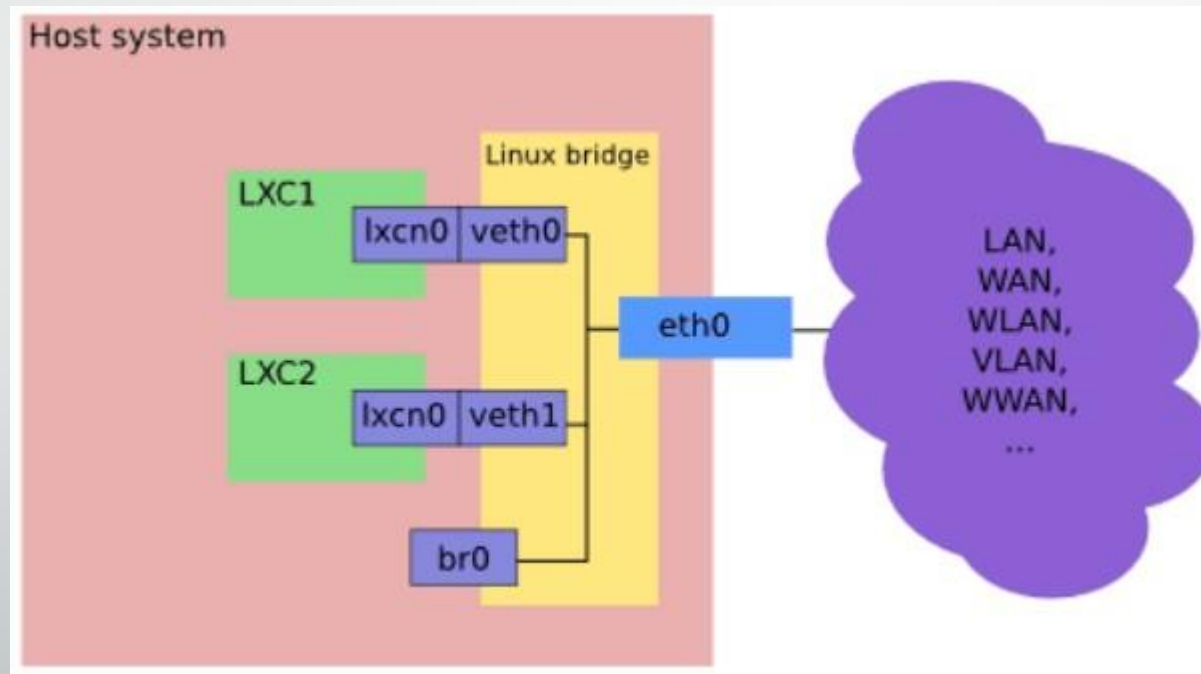
- Teljes konténer állapot
 - Menthető
 - Futó folyamatok
 - Megnyitott fájlok
 - Hálózati kapcsolatok
 - Memória szegmensek
 - Visszaállítható
 - Visszaállítható másik szerveren

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



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)

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