# Konténerek és Docker

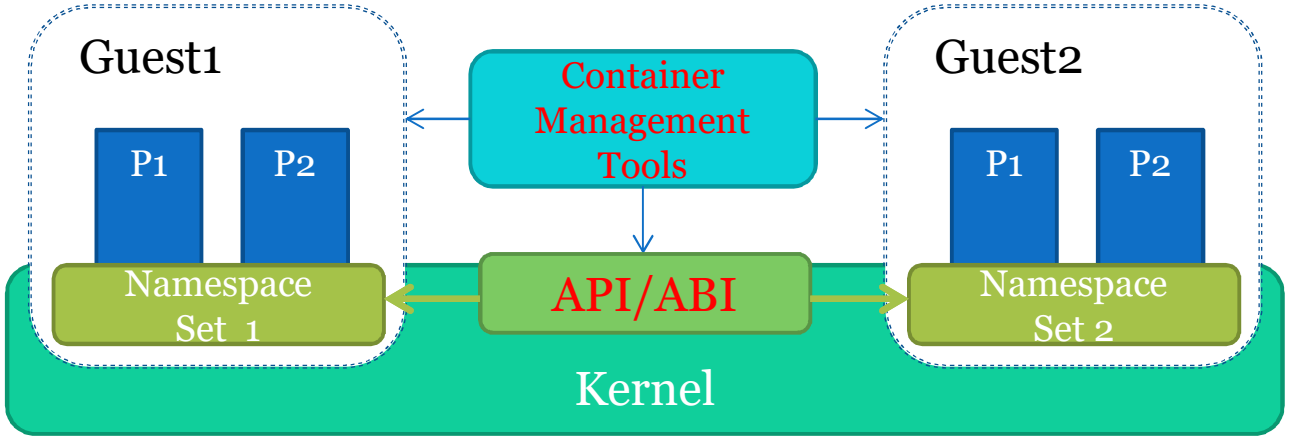
Simon Csaba

2020 február

# Konténerek

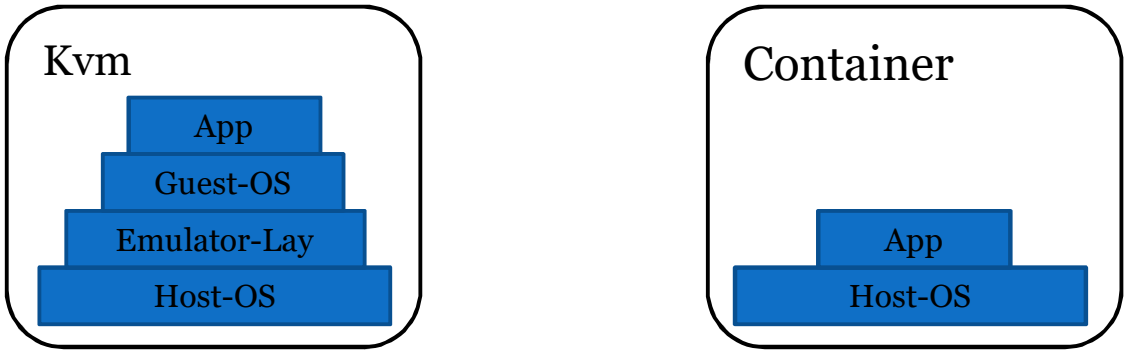
# Bevezető: Linux konténerek

- Konténer = Operation System Level virtualization method for Linux
  - Operációs rendszer (Linux) szintű virtualizációs megoldás



# Bevezető: motiváció

- Miért van szükség rá?
  - Jobb teljesítmény




- Több-bérlős virtualizációs kötnyezet
  - multi-tenant

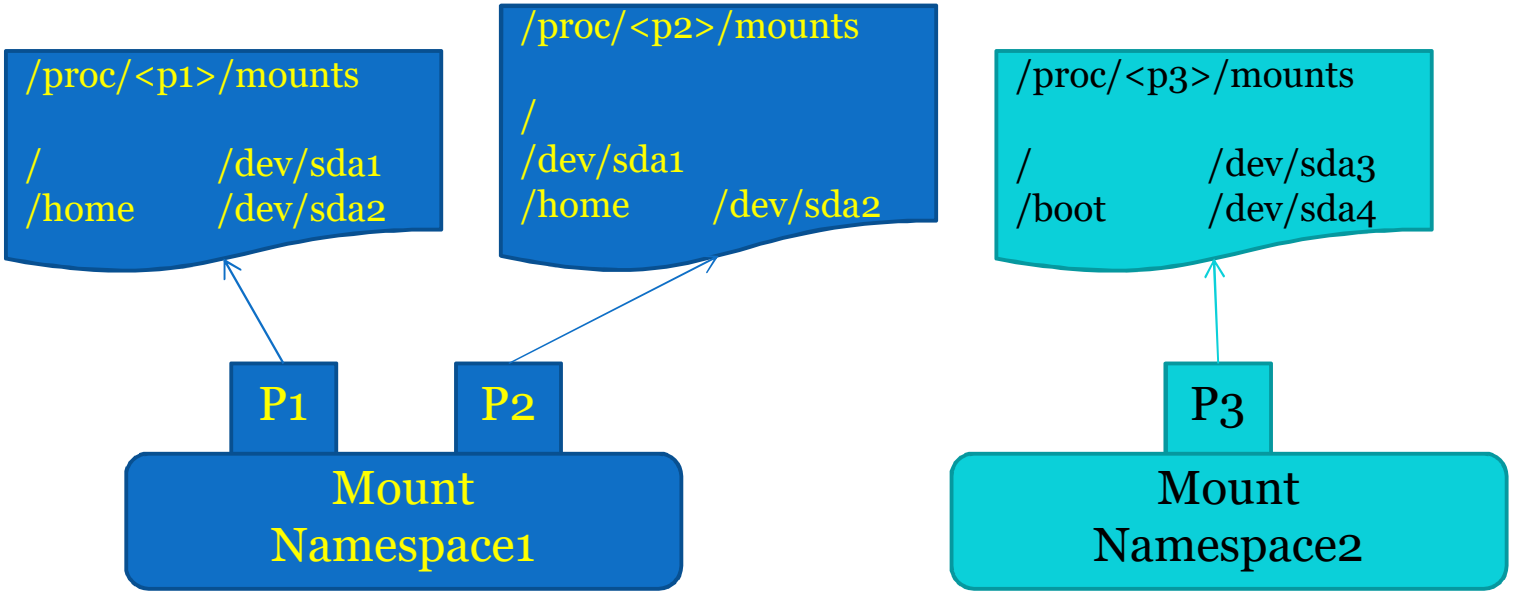
# Linux NévtereK

# Namespaces (névterek)

- Rendszer erőforrásainak elszigetelése
- 6 névtér van a Linux Kernelben
  - Mount
  - UTS
  - IPC
  - Net
  - Pid
  - User

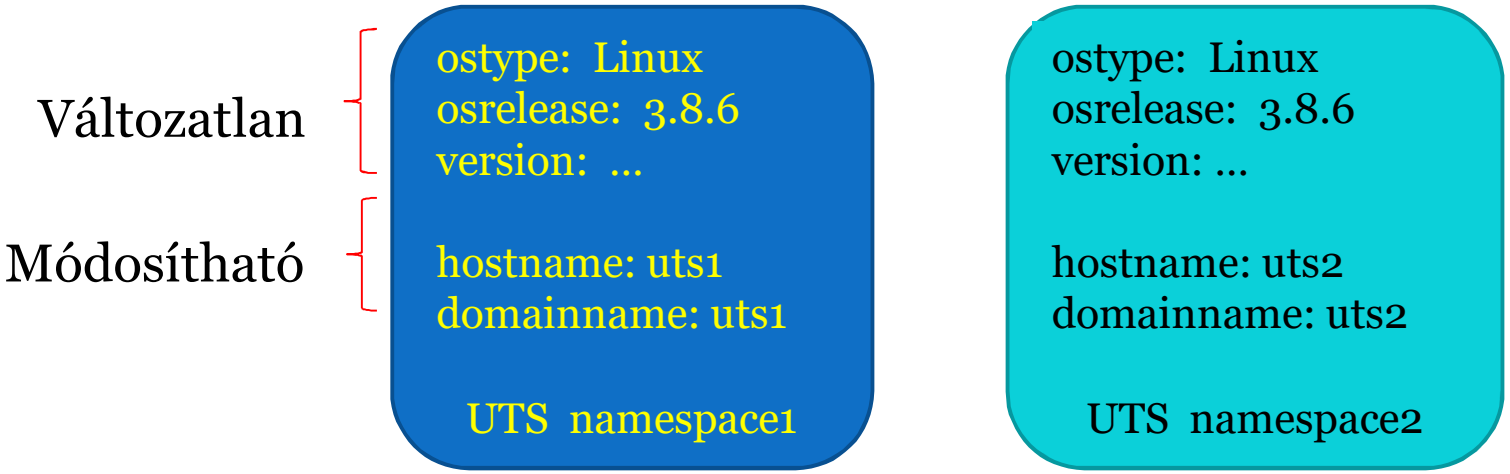
# Mount Namespace

- Saját fájlrendszer



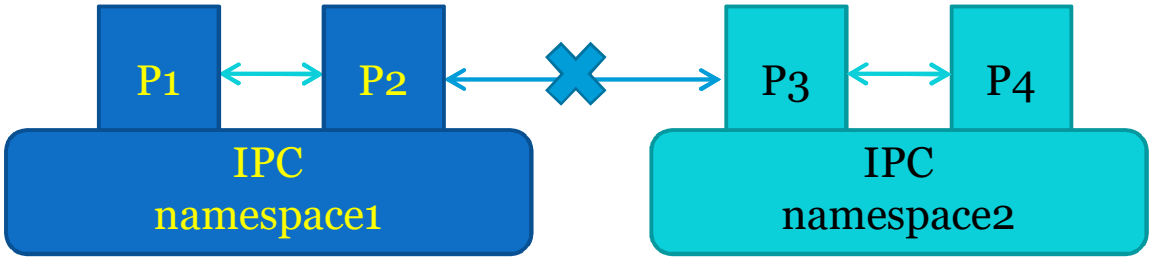
# UTS Namespace

- UTS = UNIX Timesharing System
- Saját uts-infó



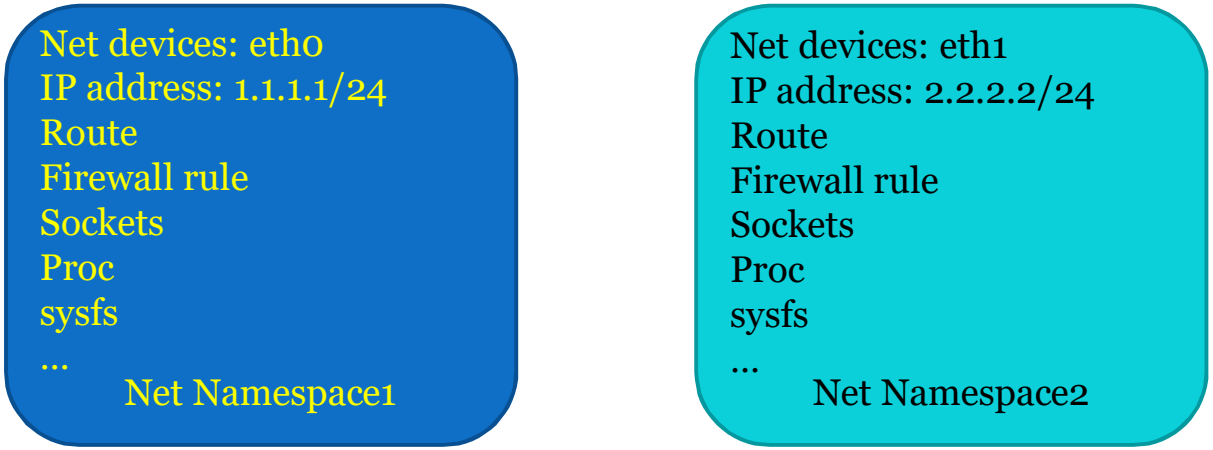
# IPC Namespace

- IPC: InterProcess Communication
- Processzek közti kommunikációt izolál:
  - Shared memory
  - Semaphore
  - Message queue



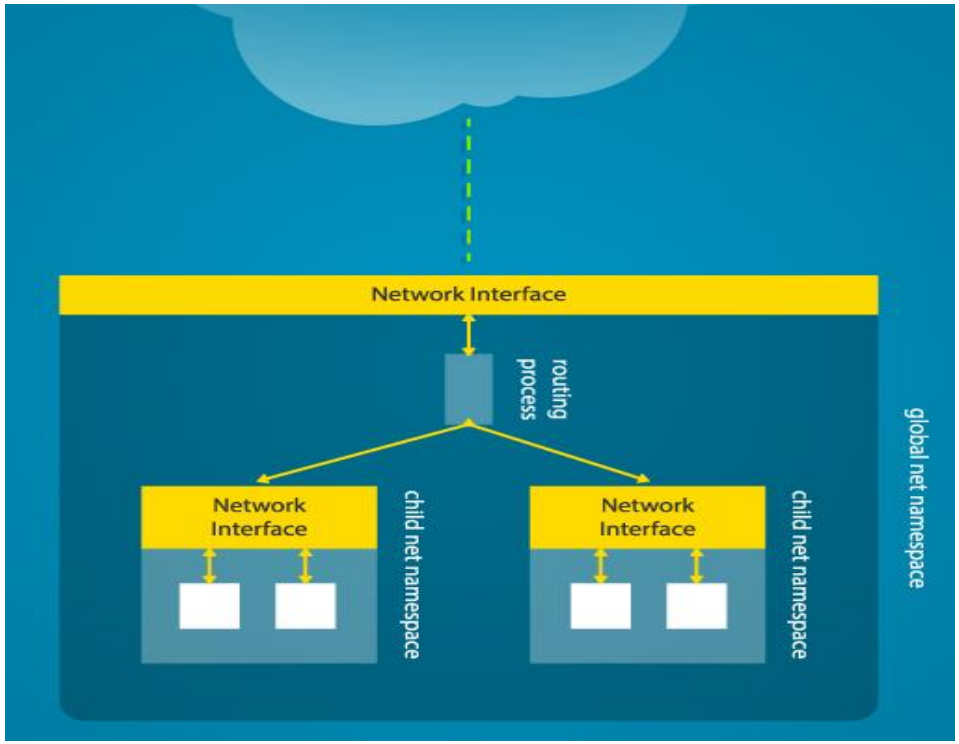
# Net Namespace 1/2

- Net namespace: a hálózati erőforrásokat rejti el



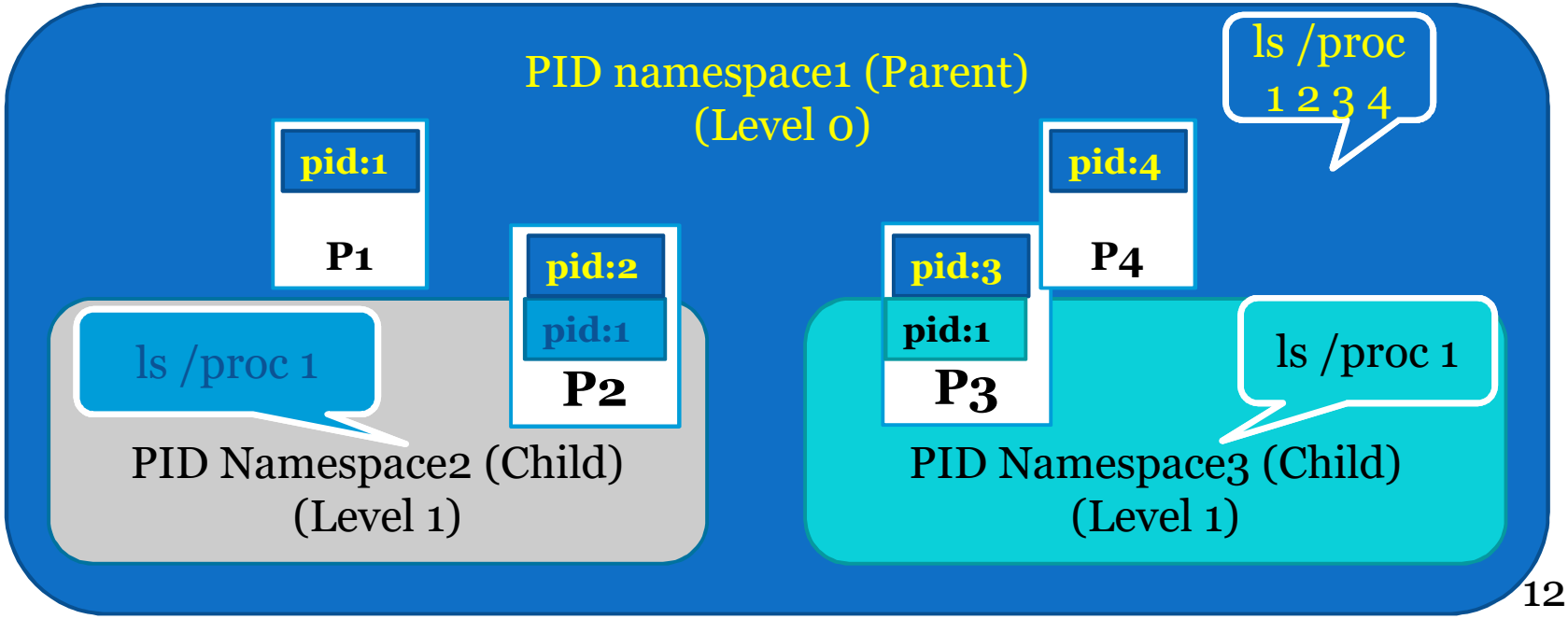
# Net Namespace 2/2

- A kernel elrejti egymástól a két külön hálózati névteret
- Ha át kell hidalni a fizikai interfész és a névtér közötti rést
  - routing



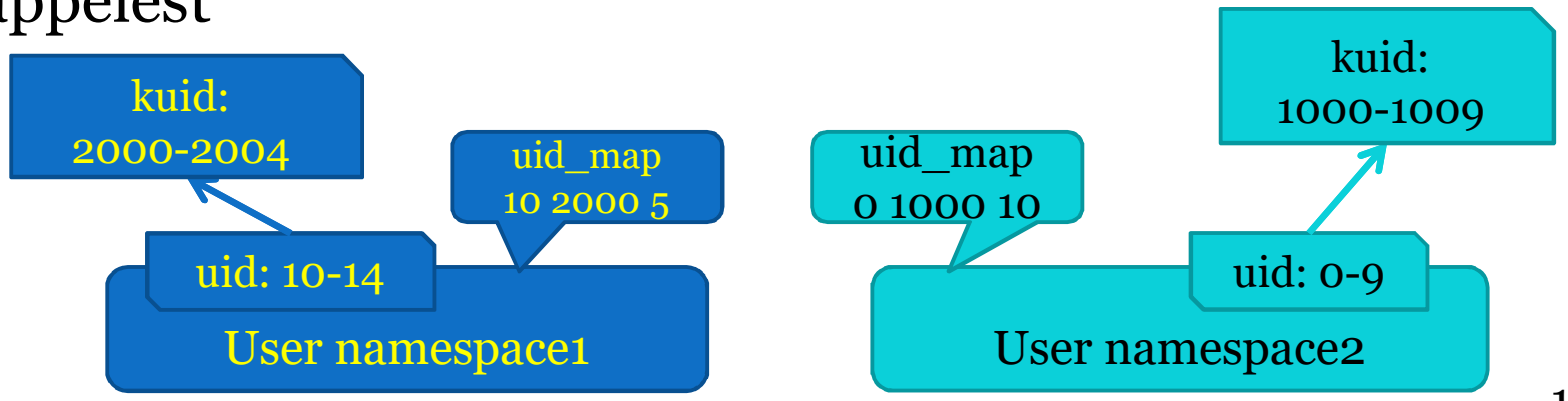
# PID Namespace

- PID: Process ID
- Hierarchikus rendszerben virtualizálja



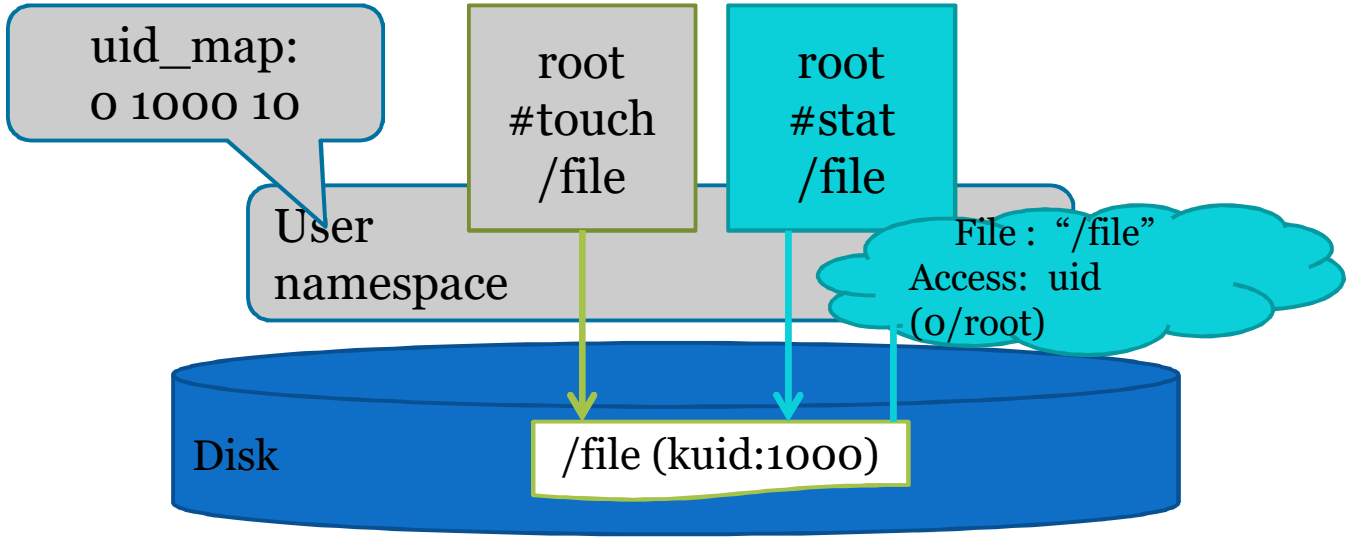
# User Namespace

- Biztonsághoz köthető felhasználói attribútomok izolálása
  - kuid/kgid: Original uid/gid, Global
  - uid/gid: user id a „user” namespaceből a kuid/kgid attribútumokba lesz átfordítva
- Csak a szülő felhasználó (parent user) NS állíthat be mappelést



# User Namespace

- Create, stat file



# CGROUPS

# Linux cgroups

- Erőforrás-felhasználás korlátozása
  - Tárolás (mem)
  - Számítás (cpu)
  - Kommunikáció (blkio)
  - Eszköz (dev)

# LXC

# System API/ABI

- Proc
  - /proc/<pid>/ns/
- System Call
  - clone
  - unshare
  - setns

# Proc

- /proc/<pid>/ns/ipc: ipc namespace
- /proc/<pid>/ns/mnt: mount namespace
- /proc/<pid>/ns/net: net namespace
- /proc/<pid>/ns/pid: pid namespace
- /proc/<pid>/ns/uts: uts namespace
- /proc/<pid>/ns/user: user namespace

- Ha adott processznek a proc fájlja ugyanaz, akkor a két processz ugyanabban a névtérben van

# Rendszerhívások

- clone

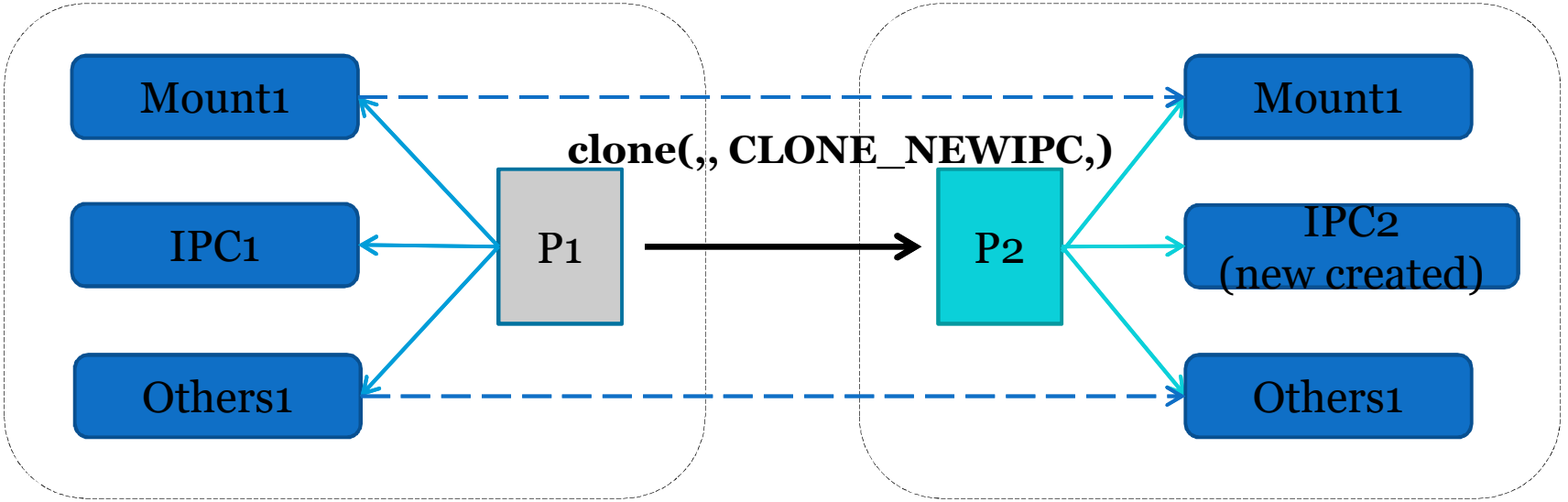
```
int clone(int (*fn)(void *), void *child_stack,
                  int flags, void *arg, ...);
```

6 flag:

CLONE_NEWIPC,CLONE_NEWNET,
CLONE_NEWNS,CLONE_NEWPID,
CLONE_NEWUTS,CLONE_NEWUSER

# Rendszerhívások

- clone
  új processz (process2) és IPC a namespace2-ben



# Rendszerhívások

- unshare

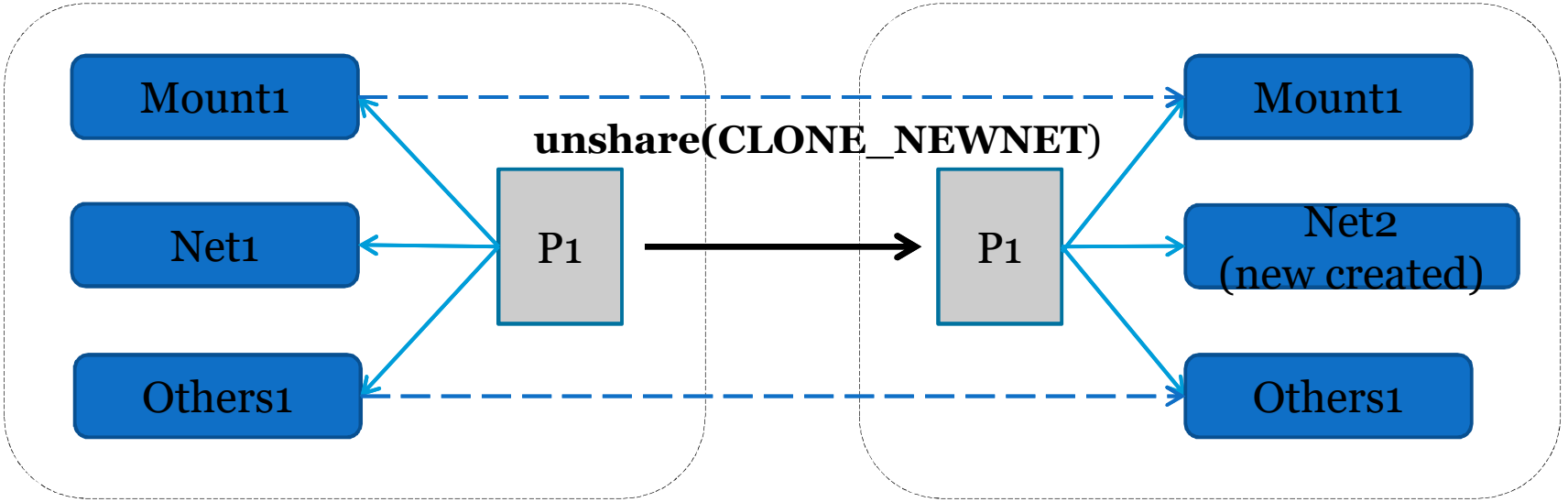
  int unshare(int flags);

  „user space”-ből új névtér hozható létre, új névtérbe lehet átlépni

# Rendszerhívások

- unshare

net namespace2 létrehozása



# Rendszerhívások

- setns

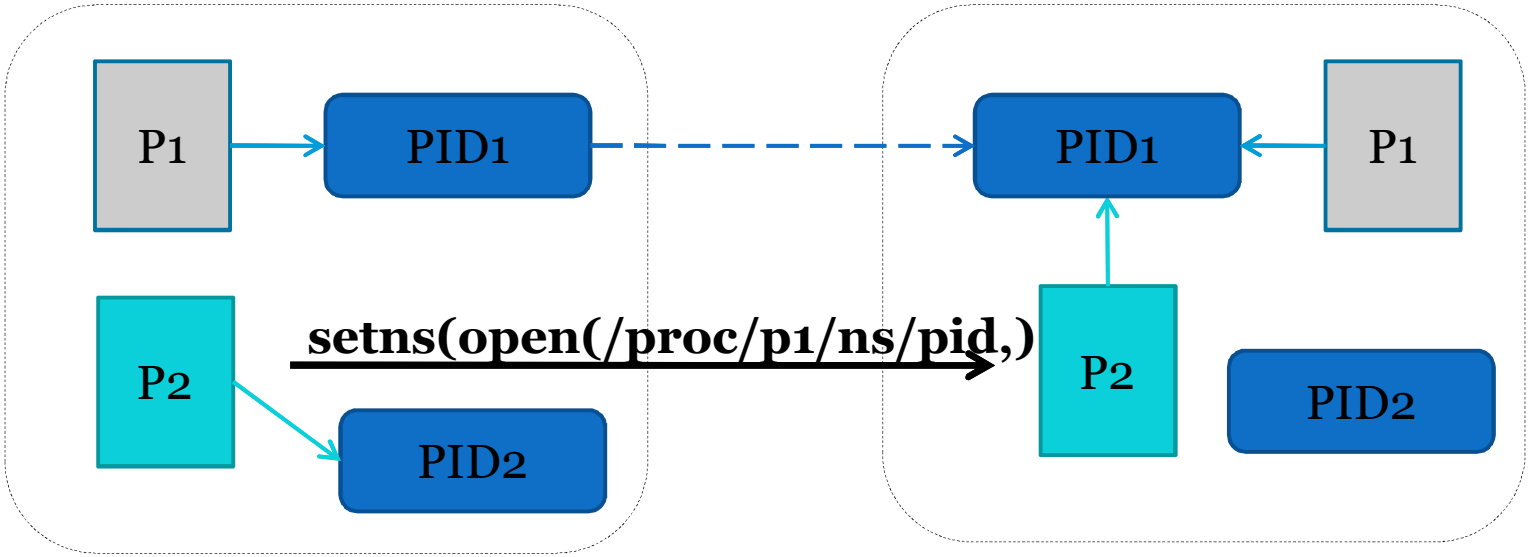
  int setns(int fd, int nstype);

  Új rendszerhívás
  Megadja, milyen névtérbe tartozzon a processz

  @fd: file descriptor of namespace(/proc/<pid>/ns/*)
  @nstype: type of namespace.

# Rendszerhívások

- setns
  A P2 PID namespace megválzoztatása



# Libvirt LXC

- Libvirt LXC: userspace container management tool
  - libvirt driver-ként megvalósítva
  - Konténer menedzsment
  - Névtér létrehozás
  - Privát fájlrendszer kezelése a konténeren belül
  - Konténer eszközeinek létrehozása
  - Cgroup által vezérelt erőforrások

# Összehasonlítás

- Vékony (lightweigth) virtualizáció, csak egy OS van (= ugyanaz a kernel)
  - „host share the same kernel with guest”

| | **Container** | **KVM** |
|---|---|---|
| performance | Great | Normal |
| OS support | Linux Only | No Limit |
| Security | Normal | Great |
| Completeness | Low | Great |

# Felmerülő kérdések

- /proc/meminfo, cpuinfo...
  - Kernel space (cgroup)
  - User space (gyenge hatékonyság)
- Új névtér
  - Audit (user namespace-hez rendelni?)
  - Syslog (szükség van rá egyáltalán?)

# Felmerülő kérdések

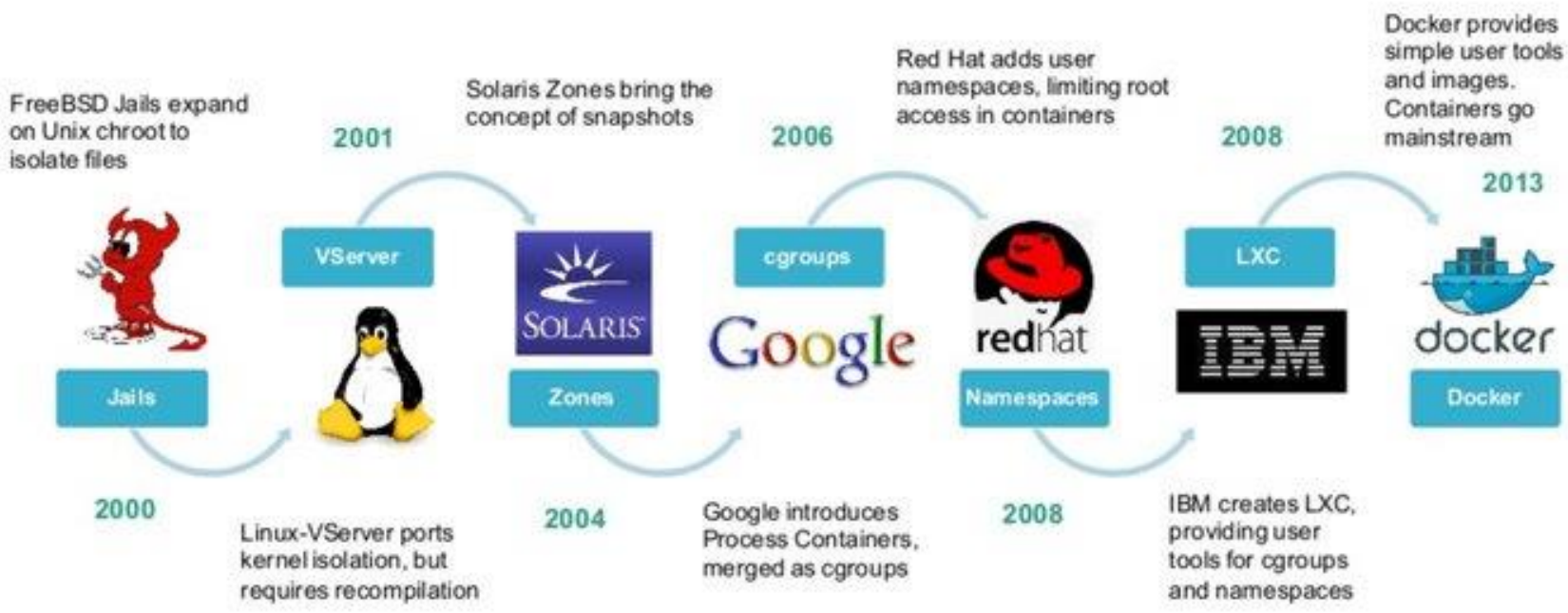
- Bandwidth (sávszélesség kezelése)
  - TC Qdisc
    - On host (hogy rendeljük hozzá a NIC-et a konténerhez)
    - On container (felhasználó módosíthatja)
  - Netfilter
    - Ingress bandwidth kezelése?
- Disk quota
  - Uid/Gid Quota (sok felhasználó)
  - Project Quota (xfs-re OK)

# DOCKER KONTÉNEREK

# Mi a Docker?

- Docker = Linux container engine
- Open Source project
  - első verzió: 3/2013 by dotCloud
  - átnevezték Docker Inc-re
- Eredetileg Python kód, később Go
- https://www.docker.io/
- git repository: https://github.com/dotcloud/docker.git

# Konténerek a Docker korszak előtt



# Mi a Docker? – official version (docker.com/blog/what-is-containerd-runtime)

2017

MICHAEL CROSBY

Aug 07

- „containers are various kernel features tied together”

# A Docker rendszer

- docker-ce és docker-ee
  - A korábbi ingyenes docker helyett docker-ce
  - Fizetős, felhasználói támogatással: docker-ee
- Multi-arch, multi-OS
- Stabil kontroll API
- Stabil plugin API
- Hibatűrés (resiliency)
- Aláírással ellátott
- Klaszeterezhető

# Docker EE → Mirantis tranzakció


- Docker „eladás” okai

> Congratulations to Docker on spinning off that pesky part of their business that makes money. twitter.com/TechCrunch/sta…

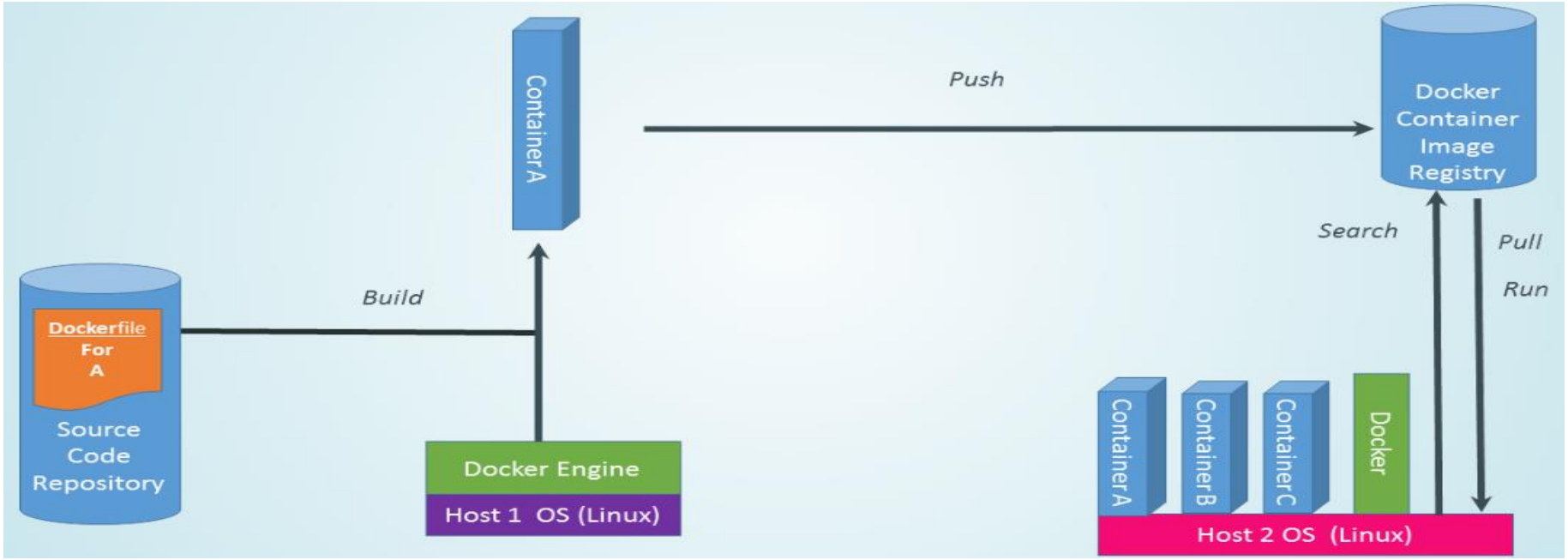
  - *~300 mill. USD befektetői tőké és nem sikerült nyereségessé válni*
  - *2019. November*
- Mirantis: a leg ígéretesebb OpenStack-alapú startup volt
- Mirantis Kubernetes alapokra váltott, on-premises Kubernetes platform
  - Docker Enterprise Engine, Docker Trusted Registry, Docker Unified Control Plane and Docker CLI
  - Fortune 100 cégek 1/3-a, a Global 500 cégek 20%-a Docker Enterprise-t (is) használ

- Mi marad a Docker Inc.-nél?
  - „expanding Docker Desktop and Docker Hub to help developers with modern workflows.”

# Docker terminológia

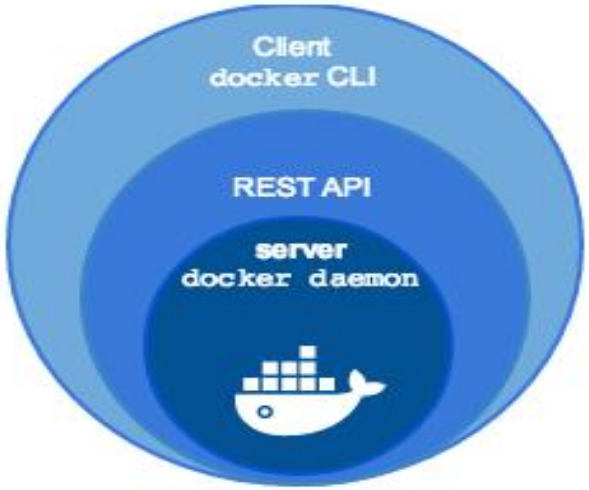
- Képfájl (image) = egy VM-nek megfelelő fájl együttes, amely tartalmaz minden olyan kiegészítést (lib, db, config, stb), ami szükséges az igényelt alkalmazás futtatásához
- Konténter (container) = egy Docker image futtatott példánya
- Tárház (registry) = képfájlok tára
  - Alapesetben helyi (on-host)
  - A Docker cég fenntart egy nyilvános, globális, on-line adatbázist (github-hoz hasonló)

# Mi a Docker?


Container A
Push
Docker Container Image Registry
Search
Pull
Run
Build
Dockerfile For A
Source Code Repository
Docker Engine
Host 1 OS (Linux)
Container A
Container B
Container C
Docker
Host 2 OS (Linux)

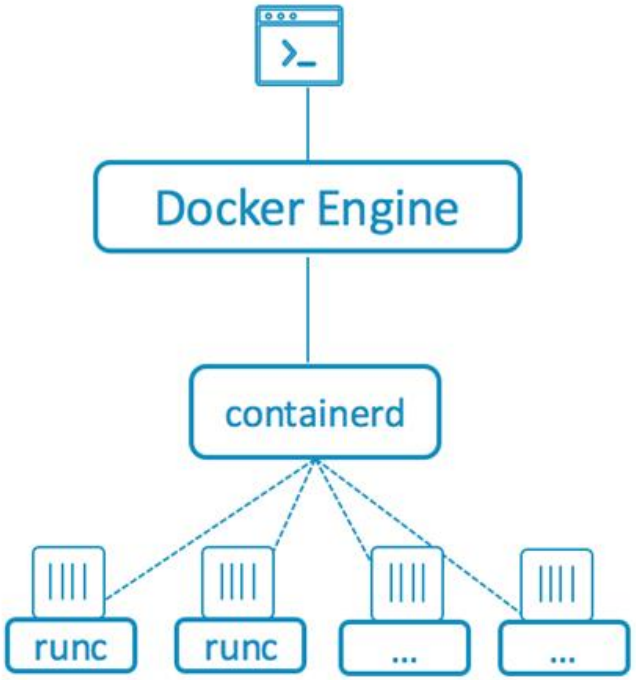
# Docker Engine

- start docker service
- Minden „docker parancsot” ez hajt végre
- Nyilvántartja a lokális hoston levő képfájlokat



# Docker „rétegek”

- Eredetileg monolit docker, a szélesebb elfogadottság után kezdték el komponensenként újra írni
- *docker cli – docker engine* felosztás csak az első lépés
- Docker Engine ~= dockerd = docker host-rezidens központi vezérlése



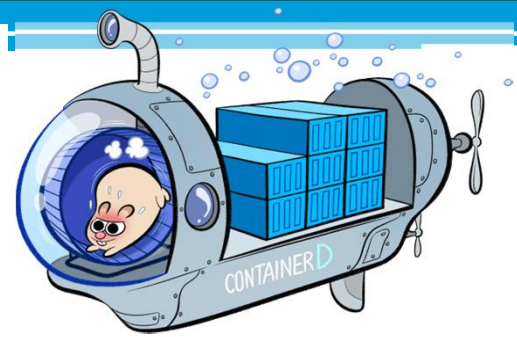

docker blog www.docker.com › blog › docker-engine-1-11-runc

ARNAUD PORTERIE

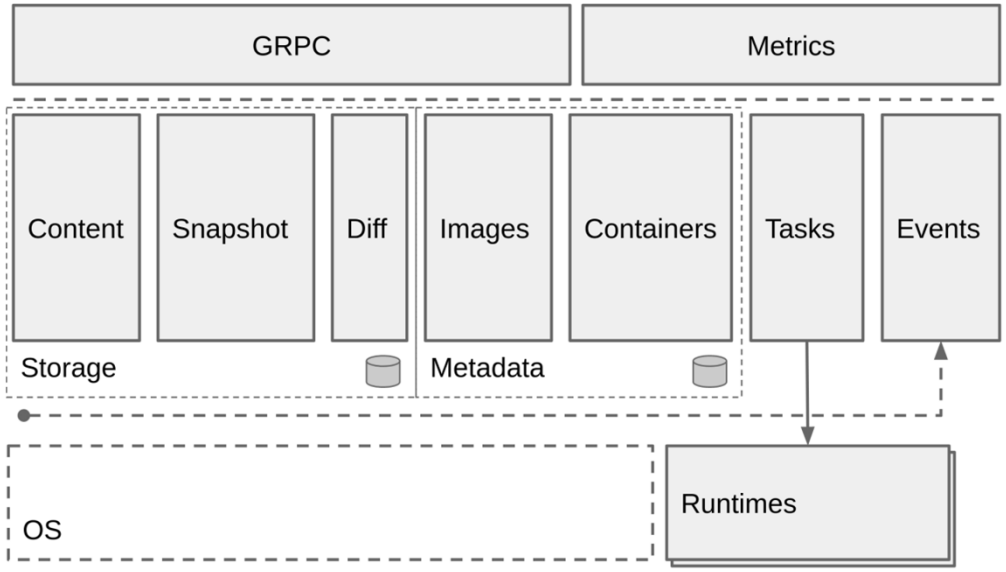
Apr 13

We are excited to introduce Docker Engine 1.11, our first release built on runC ™ and containerd ™. With this release, Docker is the first to ship a runtime based on OCI technology, demonstrating the progress the team has made since donating our industry-standard container format and runtime under the Linux Foundation in June of 2015.

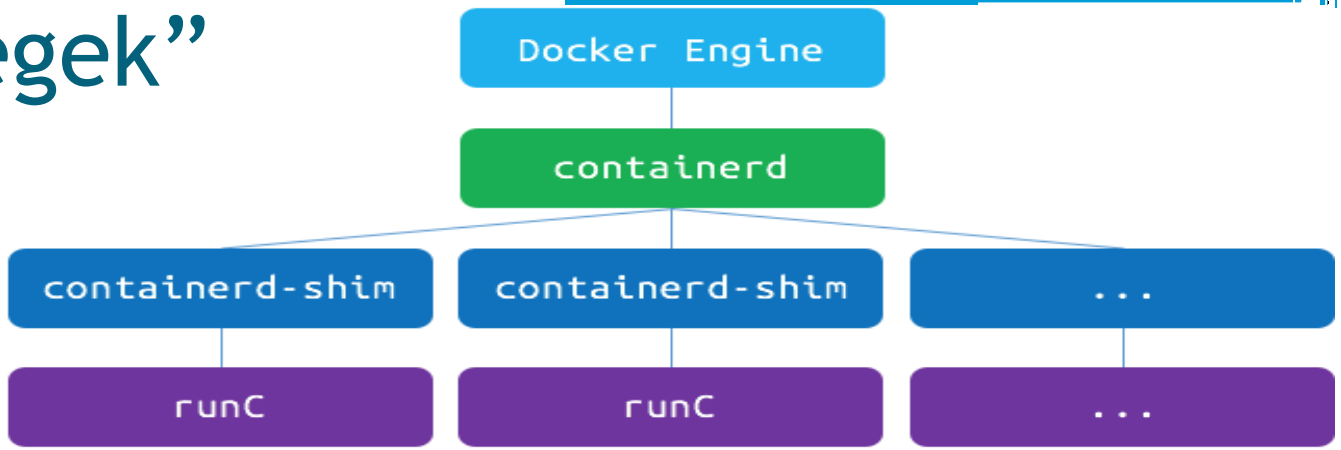
# containerd




- gRPC – lokális UNIX socket API
- „containerd is based on the Docker Engine's core container runtime„
  - Kell egy rendszer, ami az API-n keresztül kezeli
  - Dockeren kívűl is iparági standardnak tekinthető – konténer menedzsment eszköz
- ( containerd-ctr )
  - Fejlesztési célokra
- Runtimes: a konténerek futtatása
  - RunC – gyakorlatilag a containerd komponense
  - De ezt is „ki lehet szervezni", önállóan használni
- Shim layer (később...)



# Docker „rétegek”




Layers

- Ami biztos: van  dockerd (Docker Engine) és a containerd
- A containerd több komponensből áll
  - shim – a „Docker” tulajdonképpen ezt a processzt társítja egy adott konténerhez, ez a „fogantyúja” a konténernek
  - runC – „OCI bundle” : processz + root fs + specifikációk (JSON)

# Docker „rétegek” - shim


- Indításkot: runc processz

```
root@training-System-Product-Name:~# ps ax | grep docker
 1182 pts/4    S+     0:00 sudo docker run -ti geertjohan/gomatrix
 1183 pts/4    Sl+    0:00 docker run -ti geertjohan/gomatrix
 1218 ?        Sl     0:00 containerd-shim -namespace moby -workdir /var/lib/container
d/io.containerd.runtime.v1.linux/moby/5832a6783419c06da06d17971608544eee771afca3ecd9f3
9cdc7a21248f9ea4 -address /run/containerd/containerd.sock -containerd-binary /usr/bin/
containerd -runtime-root /var/run/docker/runtime-runc
 1233 ?        Sl     0:00 runc --root /var/run/docker/runtime-runc/moby --log /run/co
ntainerd/io.containerd.runtime.v1.linux/moby/5832a6783419c06da06d17971608544eee771afca
3ecd9f39cdc7a21248f9ea4/log.json --log-format json create --bundle /run/containerd/io.
containerd.runtime.v1.linux/moby/5832a6783419c06da06d17971608544eee771afca3ecd9f39cdc7
a21248f9ea4 --pid-file /run/containerd/io.containerd.runtime.v1.linux/moby/5832a678341
9c06da06d17971608544eee771afca3ecd9f39cdc7a21248f9ea4/init.pid --console-socket /tmp/p
ty600859098/pty.sock 5832a6783419c06da06d17971608544eee771afca3ecd9f39cdc7a21248f9ea4
 1289 pts/2    S+     0:00 grep --color=auto docker
31913 ?        Ssl    0:19 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cont
ainerd.sock
```

# Docker „rétegek” - shim


- Indításkot: runc processz
- A konténer indítása után a RunC „kilép”,
  - „Átadja” a vezérlést a shim-nek
  - Tkp. a containerd-shim indítja a runc-t, és megartja a file descriptorokat – mgmt, reporting, monitoring feladatok

```
root@training-System-Product-Name:~# ps ax | grep docker
 1182 pts/4    S+     0:00 sudo docker run -ti geertjohan/gomatrix
 1183 pts/4    Sl+    0:00 docker run -ti geertjohan/gomatrix
 1218 ?        Sl     0:00 containerd-shim -namespace moby -workdir /var/lib/container
d/io.containerd.runtime.v1.linux/moby/5852a6783419c06da06d17971608544eee771afca3ecd9f3
9cdc7a21248f9ea4 -address /run/containerd/containerd.sock -containerd-binary /usr/bin/
containerd -runtime-root /var/run/docker/runtime-runc
 1356 pts/2    S+     0:00 grep --color=auto docker
31913 ?        Ssl    0:19 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cont
ainerd.sock
```

# Docker „rétegek” - shim

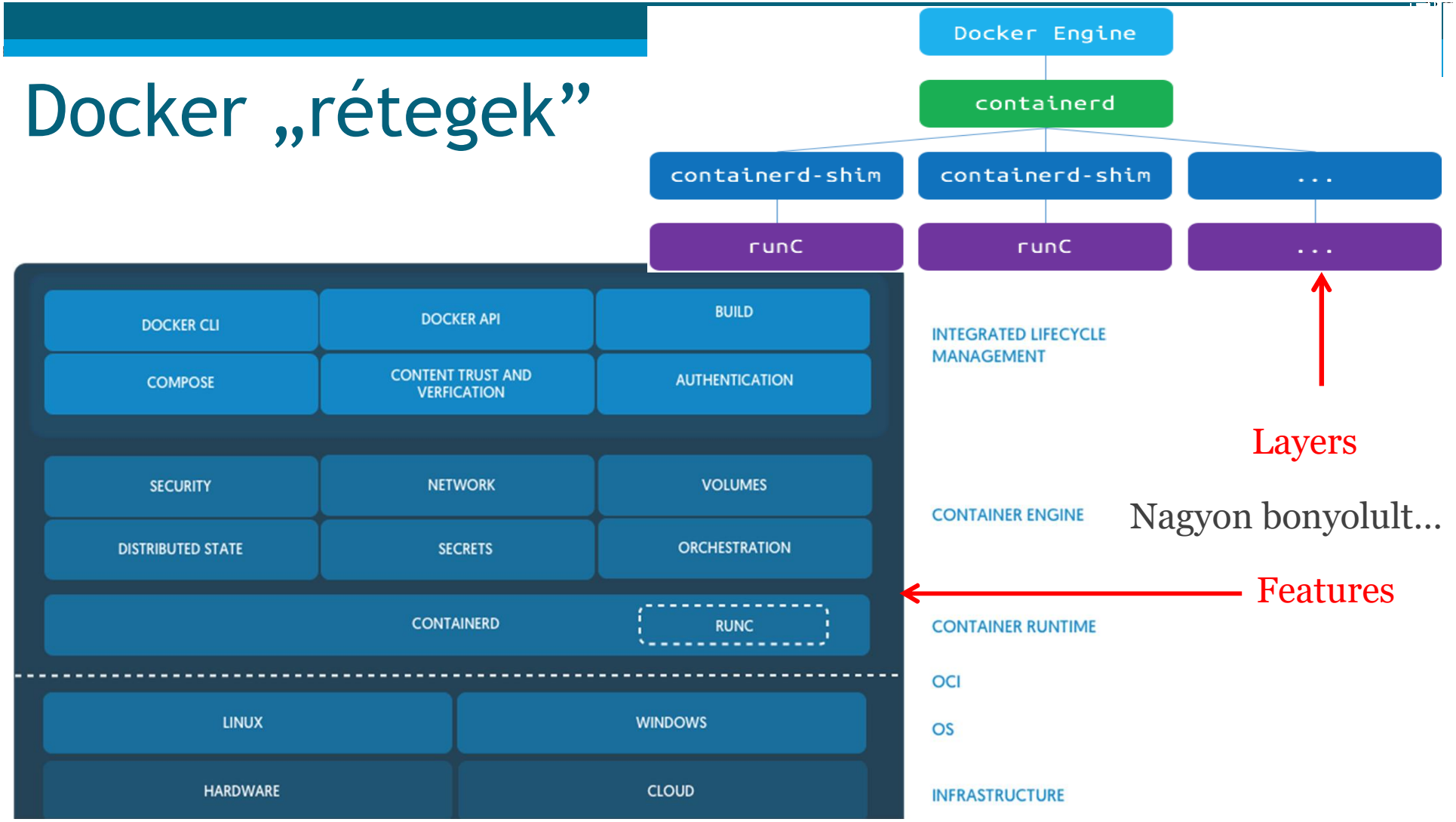
- A futtatási lánc:

dockerd → containerd → containerd-shim →gomatrix

- A runC processz leállt, de a runC által indított gomatrix továbbra is fut

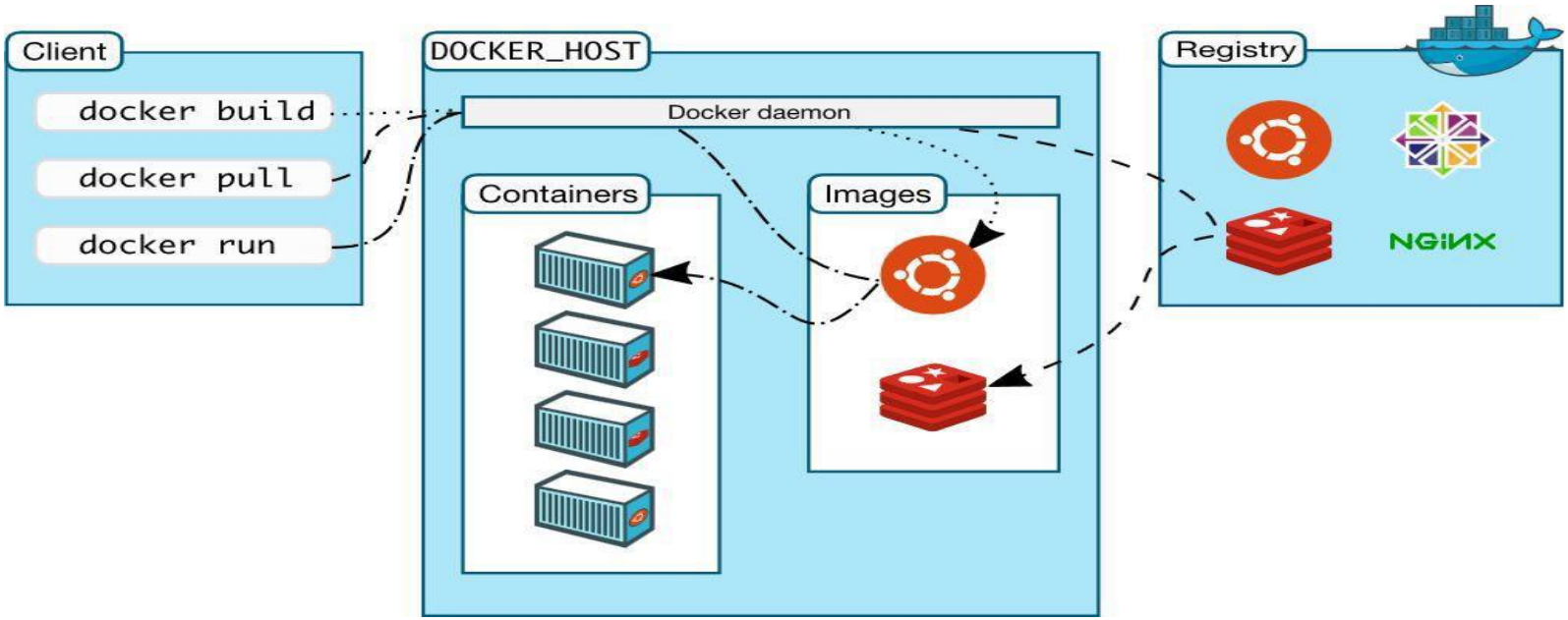
```
root@training-System-Product-Name:~# ps fax | grep docker -A 3
 2696 pts/4    Sl+    0:00  |                   \_ docker run -it geertjohan/gomatrix
 2724 ?        Sl     0:02  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.r
1.linux/moby/39169d54f1f1de83acb30d67b8e38929ee53e5fbd118b08a55c8d2cb0dca9b50 -address /run/containerd/cont
ock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
 2750 pts/0    Ssl+   0:14      \_ app
31913 ?        Ssl    0:26 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

# Docker „rétegek”



# Docker rendszer áttekintés

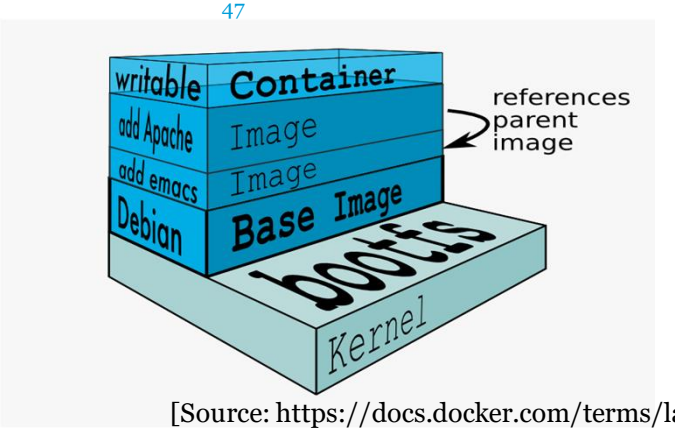
46



# Docker képfájlok

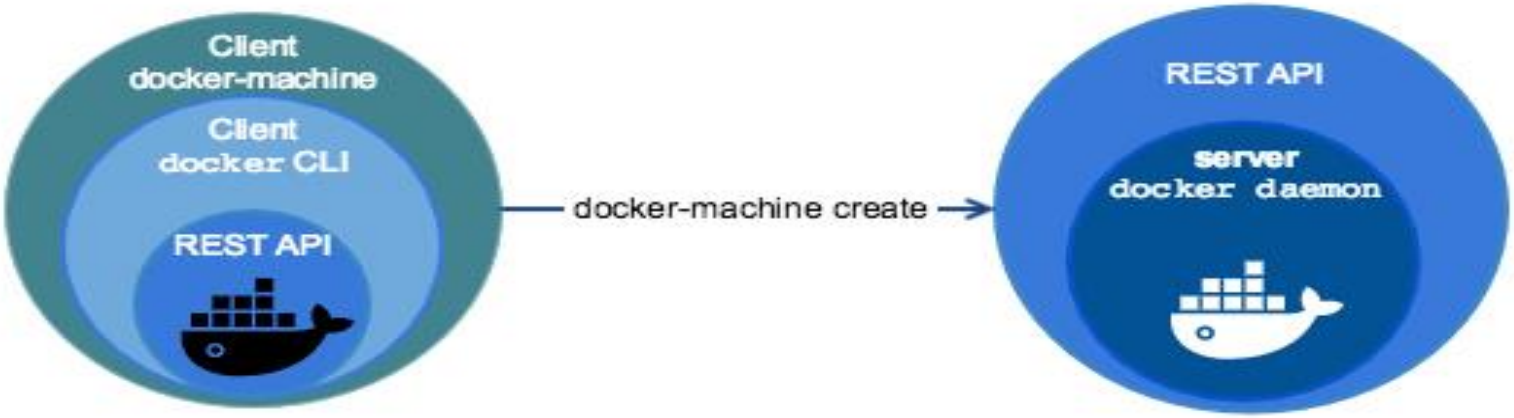
- Rétegektből áll
- Unió típusú fájlrendszer
  - union file system
  - Az összes rétegből egy képfájlt generál
  - A rétegeket tömörítve tárolja
- Sablon alapján létrehozva
  - Dockerfile
  - Kiinduálsi pint: base image (e.g. ubuntu, fedora, stb.)
  - Saját szinatxis az újabb rétegek hozzáadásához
- Adott képfájl rétegeinek látványos megjelenítése:
  - https://imagelayers.io/

47




[Source: https://docs.docker.com/terms/layer/]

# Docker Machine




- Távoli állomásokon is lehet konténereket kezelni
  - Automatikus host létrehozás
  - Docker Engine install
  - docker kliens konfig

# Docker Machine


- Távoli állomásokon is lehet konténereket kezelni
  - saját parancsa van (docker-machine)

50

# Docker Compose

- Összetett feladatok
  - Saját kliens a Docker ökoszisztémán belül
- Több szolgáltatást (konténert) indít egyszerre
- Dockerfile -> alkalmazások sajátosságai
- docker-compose.yml
- docker-compose up

```
version: '2'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

# Docker workflow 1/2

- Fejlesztés egy dev környezetben (local machine v. container)
- Konténerben futtatni a többi szolgáltatás (services) (pl. adatbázisok)
  - És ugyanúgy működik minden más gépen
    - Kernel verziók
- A „valós” működés tesztelése során :
  - *Másodpercek* alatt fordul (build)
  - *Azonnal* fut

# Docker workflow 2/2

- Ha a lokális build OK, akkor
  - Feltölthető a registry-be (public/private)
  - Automatikusan futtatható
  - Üzemi (production, enterprise) környezetben
  - Egyszerű átjárást biztosít a dev és production környezet között
- Hiba esetén: Rollback
  - Vissza lehet térni egy korábbi verzióra

# a.) Képfájlok (Docker images) készítése (run/commit megoldással)

- 1) docker run ubuntu bash
- 2) apt install <this> <and that>
- 3) docker commit <containerid> <imagename>
- 4) docker run <imagename> bash
- 5) git clone git://.../mycode
- 6) pip install -r requirements.txt
- 7) docker commit <containerid> <imagename>
- 8) GOTO 4 (ha kell)
- 9) docker tag <imagename> <user/image>
- 10) docker push <user/image>

# a.) Előnyök/hátrányok

- Előnyök

– Kényelmes, ismert lépések
– roll back/forward – szükség szerint

- Hátrányok

– Kézi-vezérelt folyamat
– Iteratív változások „felgyűlnek”
– Teljes újrafordítás (rebuild) folyamata sok hibalehetőséget tartogat

# b.) Dockerfiles

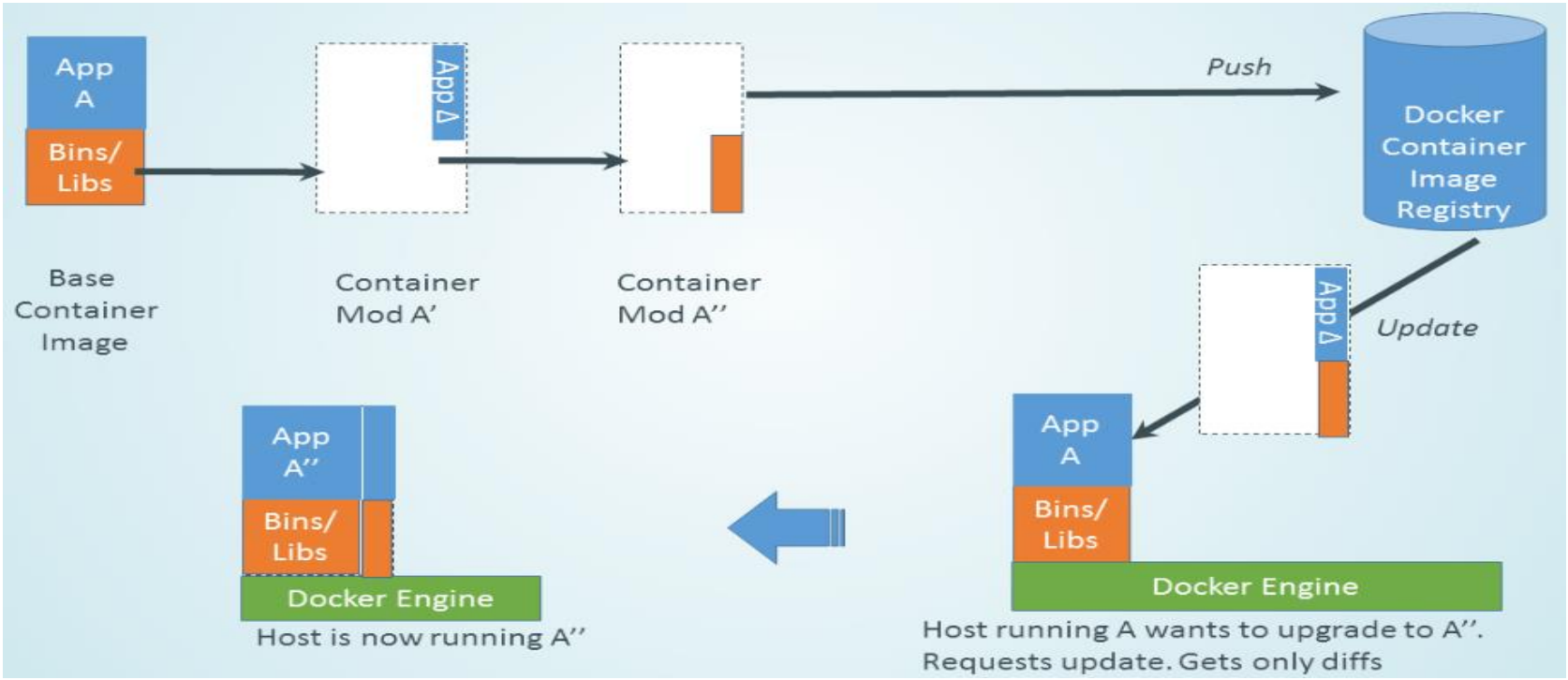
- FROM - egy már létező képfájlból indul ki az új képfájl; ez lesz az alap réteg
  - pl. egy Linux disztribúció
  - Gyakran egy lecsupaszított Linux képfájlt használnak (alpine, busybox)
- COPY – fájlokat másol át a host adott könyvtárából (directory) a képfájlba (pl. konfig, forráskód, szkript)
- RUN – a képfájlba telepítendő, előkészítendő feladatok futtatása
  - Pl. apt update, apt install <program_csomag>, apt
  - Minden külön sor egy külön réteget hoz létre
  - Egymás után felfűzött parancsok („&&” segítségével) egy réteget képeznek
    - Pl. apt update && apt install –y git
- EXPOSE – egy portot nyit majd a konténer számára
- CMD – a konténer indításakor futtatott parancs

- Docker docs on Dockerfile: https://docs.docker.com/engine/reference/builder/

# b.) Előnyök

- Gyorsan tanulható
- Könnyen újra-fordítható
  - Caching rendszer segíti ezt
- Egy fájlban meghatározható a build folyamat

# Docker – miért gyors?



# Docker vs. VM

58

- **Latency**: Applications with a low tolerance for latency are going to do better on physical. This something we see quite a bit in financial services (trading applications are prime example).
- **Capacity**: VMs made their bones by optimizing system load. If your containerized app doesn't consume all the capacity on a physical box, virtualization still offers a benefit here.
- **Mixed Workloads**: Physical servers will run a single instance of an operating system. So, you if you wish to mix Windows and Linux containers on the same host, you'll need to use virtualization
- **Disaster Recovery**: Again, like capacity optimizations, one of the great benefits of VMs are advanced capabilities around site recovery and high availability. While these capabilities may exist with physical hosts, the are a wider array of options with virtualization.
- **Existing Investments and Automation Frameworks** : A lot of the organizations have already built a comprehensive set of tools around things like infrastructure provisioning. Leveraging this existing investment and expertise makes a lot of sense when introducing new elements.
- **Multitenancy**: Some customers have workloads that can't share kernels. In this case VMs provide an extra layer of isolation compared to running containers on bare metal.
- **Resource Pools / Quotas**: Many virtualization solutions have a broad feature set to control how virtual machines use resources. Docker provides the concept of resource constraints, but for bare metal you're kind of on your own.
- **Automation/APIs**: Very few people in an organization typically have the ability to provision bare metal from an API. If the goal is automation you'll want an API, and that will likely rule out bare metal.
- **Licensing Costs**: Running directly on bare metal can reduce costs as you won't need to purchase hypervisor licenses. And, of course, you may not even need to pay anything for the OS that hosts your containers.

# Docker előnyei

- Könnyű installációs folyamat
- Minden alkalmazás fut rajta, sok környezetben
- Ismételhető build folyamat
- Nagy hype, erős közösség, gyors javítások
- Új virtualizációs folyamatok

- **Hátrány**
- A Docker konténer típusa
  - A gazdarendszer OS-e határozza meg
- „Orchestration”
- Hálózati kommunikáció

# Docker hátrányai

- A Docker konténer típusa
  - A gazdarendszer OS-e határozza meg
- „Orchestration”
- Hálózati kommunikáció

- De: jelentős és még mindig aktív fejlesztések az elmúlt félévben is
  - A hype és erős közösség előnye

61

# Biztonság?

- Docker elérése REST API-n keresztül HTTP felett?
  - Authentikáció!
- Docker démon root jogokkal fut
  - A konténerek már OK
  - „visszanyúlhatnak”?
- docker-1.3 --cap-add, --cap-drop
  - man capabilities
  - „overview of Linux capabilities”
  - „Starting with kernel 2.2”
  - „per-thread attribute”
- Várhatóan további változások lesznek
  - Docker démon

# Források

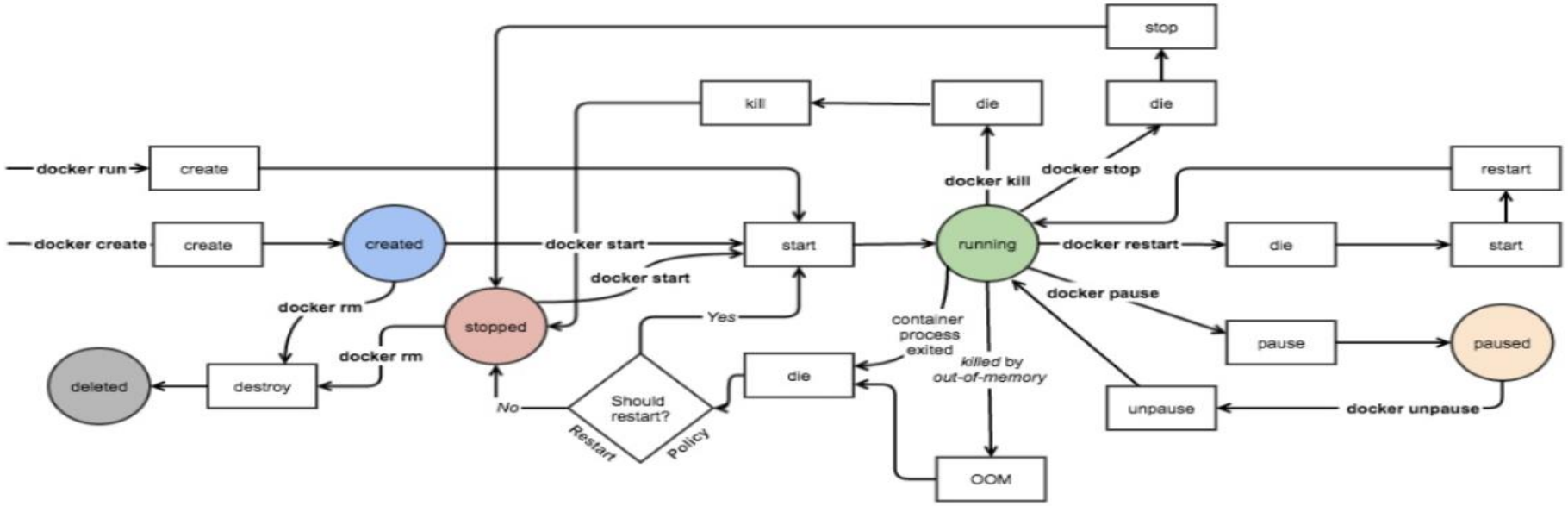
- Docker történet dióhéjban:
  http://www.infoworld.com/article/3025870/paas/the-sun-sets-on-original-docker-paas.html
- Docker áttekintés:
  http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment
- „The Docker Book”
  http://www.dockerbook.com/#toc
- Docker Meetup @Budapest
  http://www.ustream.tv/recorded/60277876

# Docker gyakorlat – parancsok, info, egyebek:

- **https://board.net/p/HakapDock**

64

# Docker állapotgép




- die, kill ≠ destroy