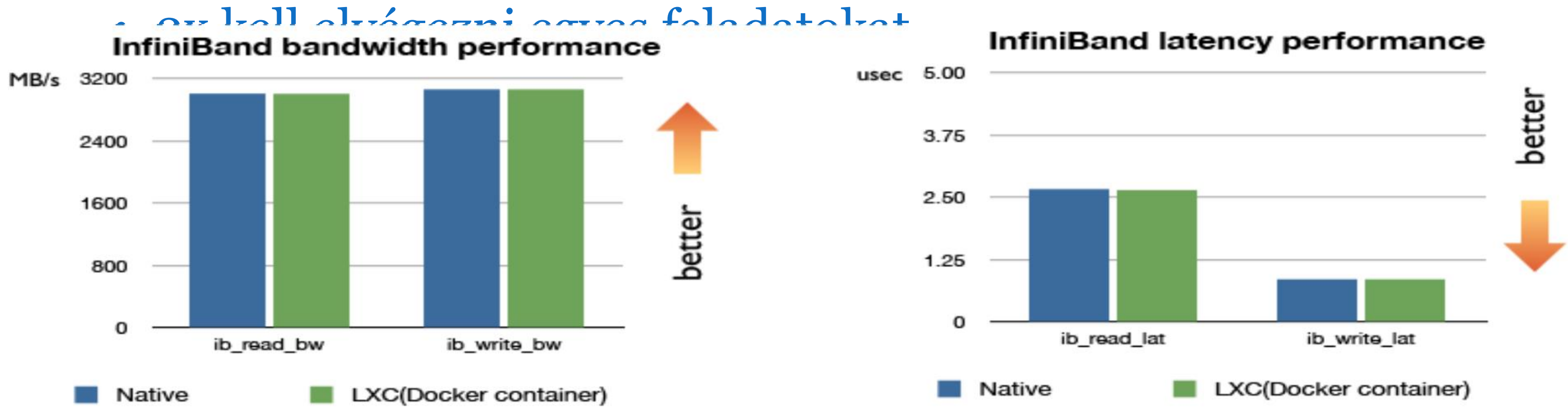
Felhő alapú hálózatok

Simon Csaba
TMIT, HSNLab
2019

Konténerek

Virtualizáció és teljesítmény?

- Motiváció:
  - Virtualizáció = valamiben(fut_valami)
    - ez kell elvégezni egyes feladatokat



Konténer metafora: áruszállítás probléma

- Logisztikai (menedzsment) kérdés
  - Sok szállítási platform, sok árutípus
  - Egy közös csomagoló-egység

	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?

Konténer metafora: inter-modális konténer

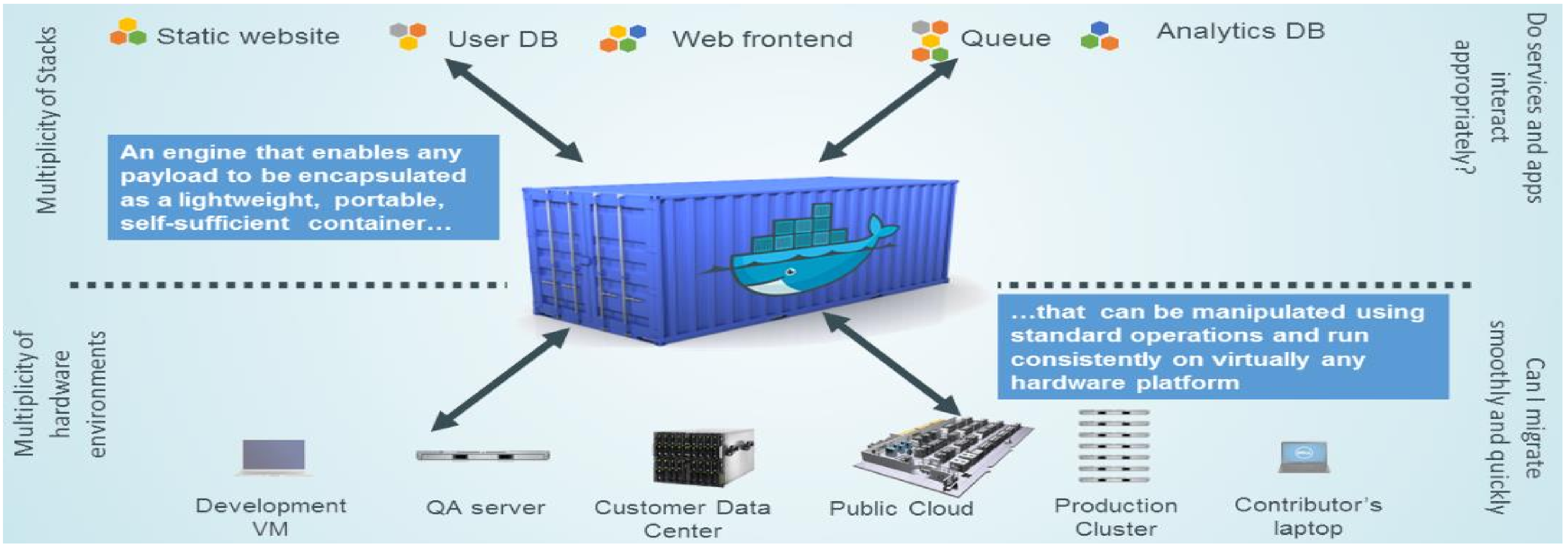
- Logisztikai (menedzsment) kérdés
  - Sok szállítási platform, sok árutípus
  - Egy közös csomagoló-egység
  - KONTÉNER (egységes, köztes szállítási egység)

Kódok „szállítása” virtualizációs megoldásokhoz

- Szállítási platform => végrehajtási környezet
- Árutípus => számítási feladat

	?	?	?	?	?	?
NODE JS	?	?	?	?	?	?
MySQL RAILS	?	?	?	?	?	?
php	?	?	?	?	?	?
	?	?	?	?	?	?
cassandra	?	?	?	?	?	?

Alkalmazás-konténerek



Zárójel: esettanulmány (miért is választják a Dockert a felhő helyett?)

Running Your Services On Docker

Robert Bastian: An experience report

Webinar Series 2015

Why Docker?

My World Needed To Change

- 5+ individual teams building “micro services” in Java and Scala
- Frictionless deployment of “micro-services” using Chef & AWS
- 25+ separate “micro-services” deployed in the previous 18 months
- Each service is typically deployed to a single AWS virtual machine
- Each service is deployed 6x - dev, test, staging (2x) and production (2x)
- 25+ “micro-services” became nearly 150 AWS virtual machines

Why Docker? COST!

The AWS bill is too damn high!

- Decline in the global price of oil causing churn in our business
- 6 AWS virtual machines per service isn’t sustainable with our budget
- AWS monthly bill started to gain visibility from sr. management and ***the board***

Why Docker? WASTE!

We weren't using the compute and memory resources purchased from AMZN!

- Nearly all "micro-services" were at 1% CPU utilization
- Nearly all "micro-services' were only using 40% of memory (JVM)
- 150+ virtual machines essentially sitting idle

Why Docker? LOCK IN!

How would we leave AMZN if we wanted to?

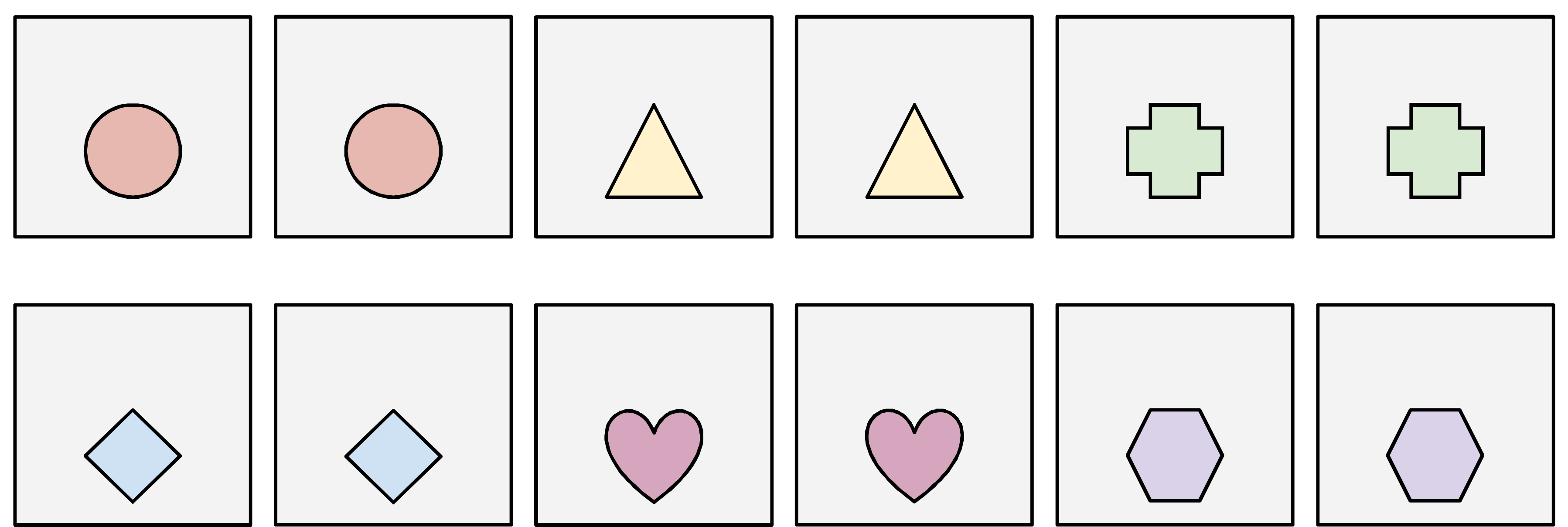
- Could we use Drillinginfo IT's Openstack platform?
- What about alternate IaaS providers like Rackspace or Azure?
- What about Container as a Service (CaaS) providers like Joyent, Tutum or Profitbricks?
- What about using Amazon's Container Service?

My World Needs To Change - Problem Statement

"How can we ***deploy fewer*** virtual machines while ***increasing the density and utilization*** of services per machine ***without locking*** us into a specific IaaS provider?"

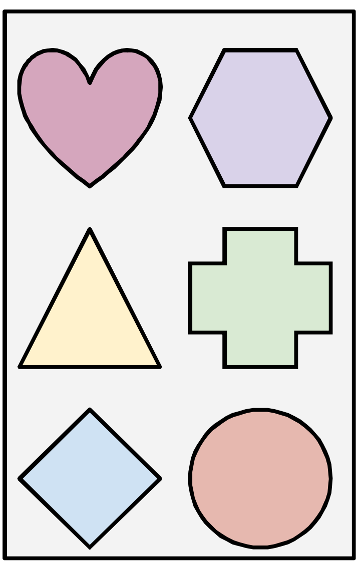
Why Docker Is Important - Before Containers

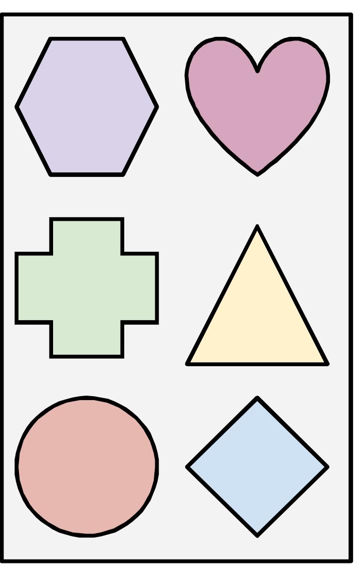
Very inefficient use of memory and CPU resources



Why Docker Is Important - After Containers

Isolated services in fewer VMs…



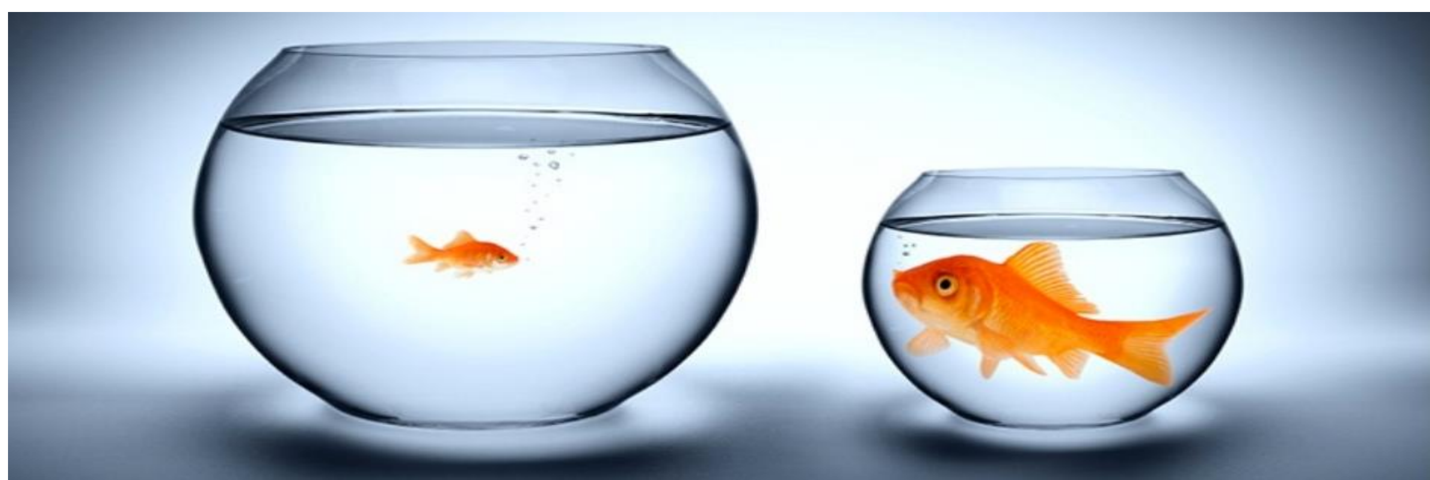


… and use VMs more efficiently.

Why Is Docker Important?

Docker container technology provides our “micro-services” platform:

- Increased ***density*** of ***isolated*** “micro-services” per virtual machine (9:1!)
- Containerized “micro-services” are ***portable*** across machines and providers
- Containerized “micro-services” are much ***faster*** than virtual machines



Zárójel vége

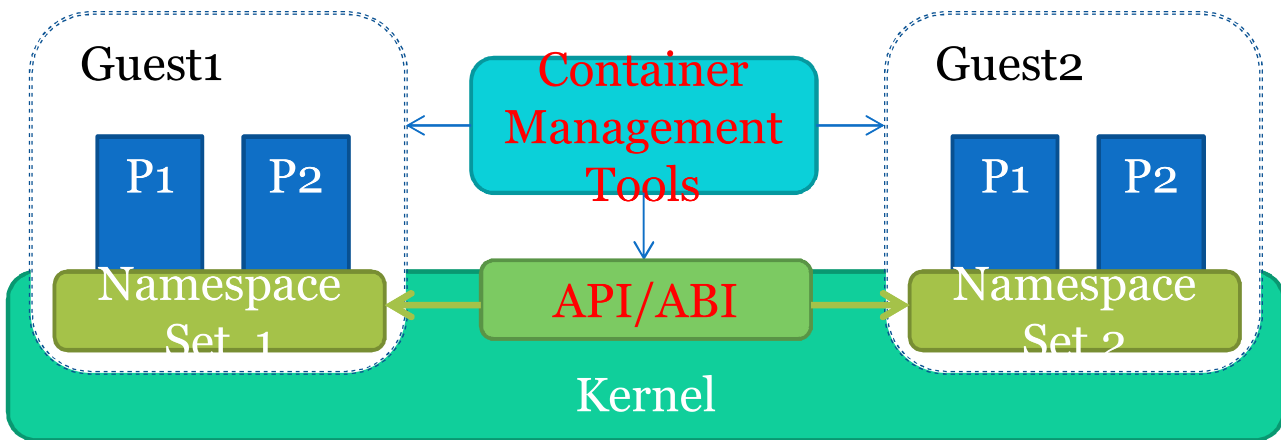
Running Your Services On Docker

Robert Bastian: An experience report

Webinar Series 2015

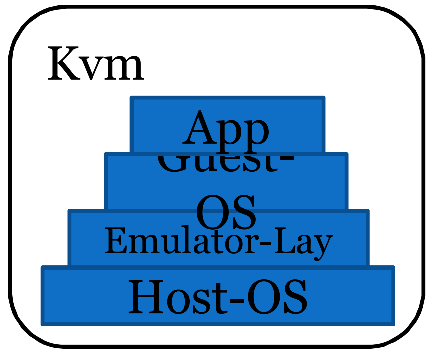
Bevezető: Linux konténerek

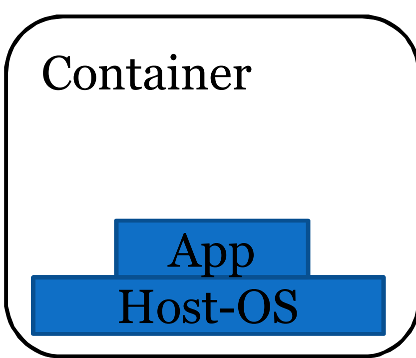
- Konténer = Operation System Level virtualization method for Linux
  - Operációs rendszer (Linux) szintű virtualizációs megoldás



Bevezető: motiváció

- Miért van szükség rá?
  - Jobb teljesítmény





- Több-bérlős virtualizációs kötnyezet
  - multi-tenant

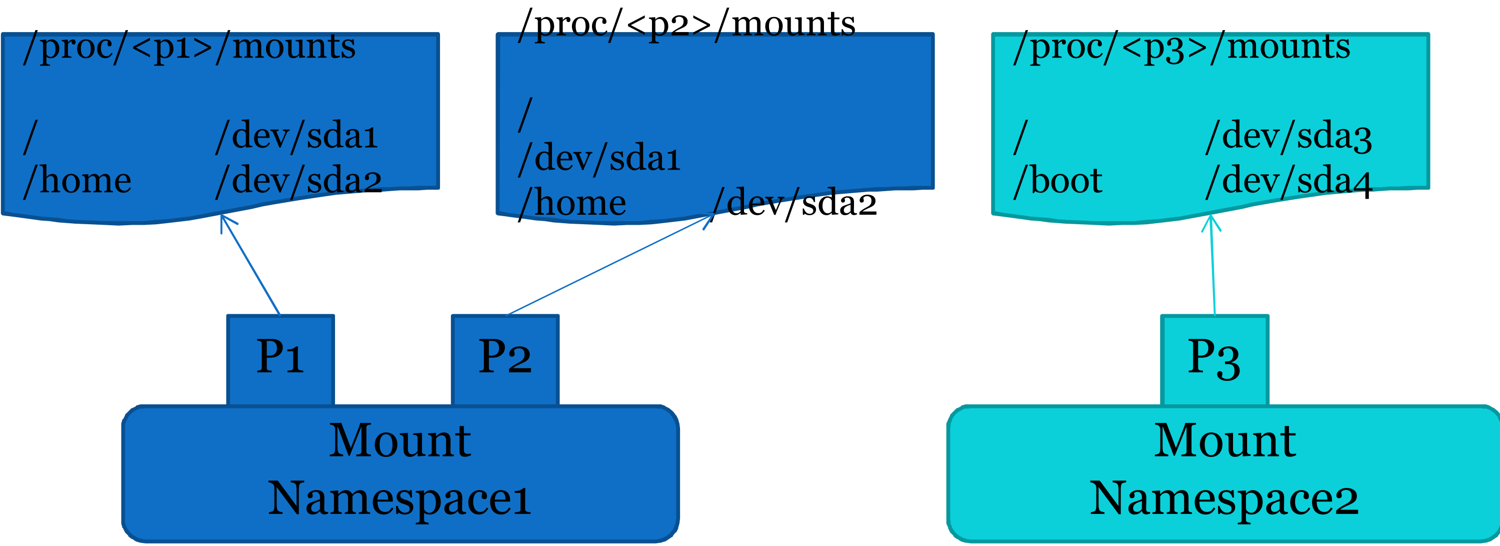
Linux NévtErEK

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

Változatlan

Módosítható

ostype: Linux
osrelease: 3.8.6
version: ...

hostname: uts1
domainname: uts1

UTS namespace1

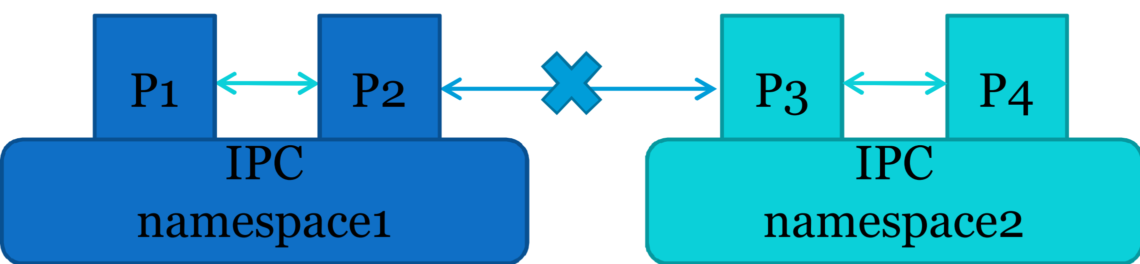
ostype: Linux
osrelease: 3.8.6
version: ...

hostname: uts2
domainname: uts2

UTS namespace2

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 devices: eth0
IP address: 1.1.1.1/24
Route
Firewall rule
Sockets
Proc
sysfs
…

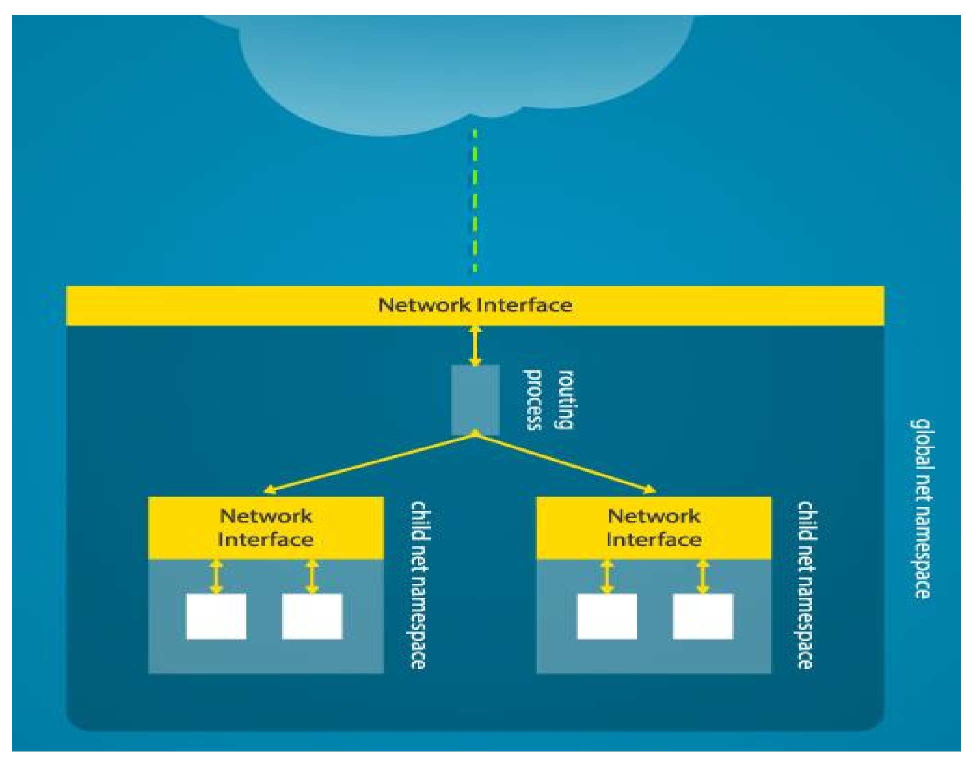
Net Namespace1

Net devices: eth1
IP address: 2.2.2.2/24
Route
Firewall rule
Sockets
Proc
sysfs
…

Net Namespace2

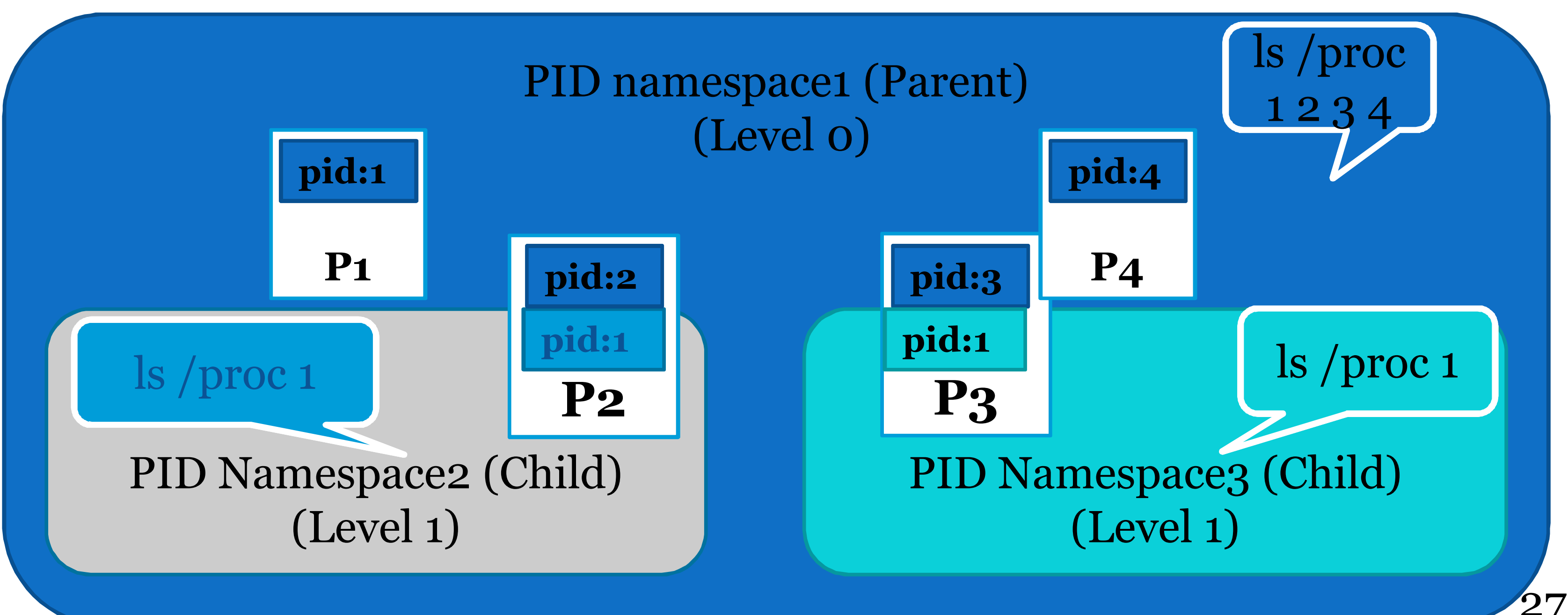
Net Namespace 2/2

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



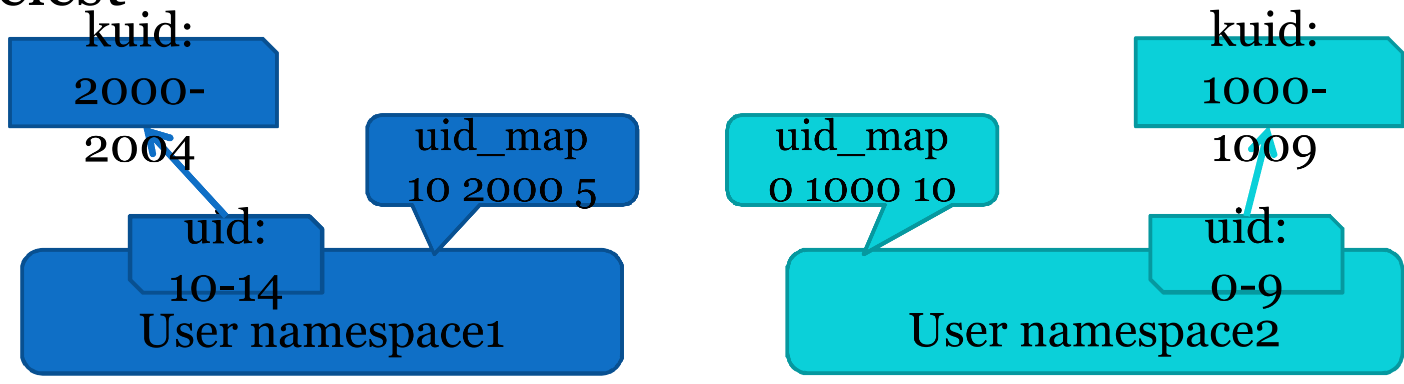
PID Namespace

- PID: Process ID
- Hierarchikus rendszerben virtualizálja



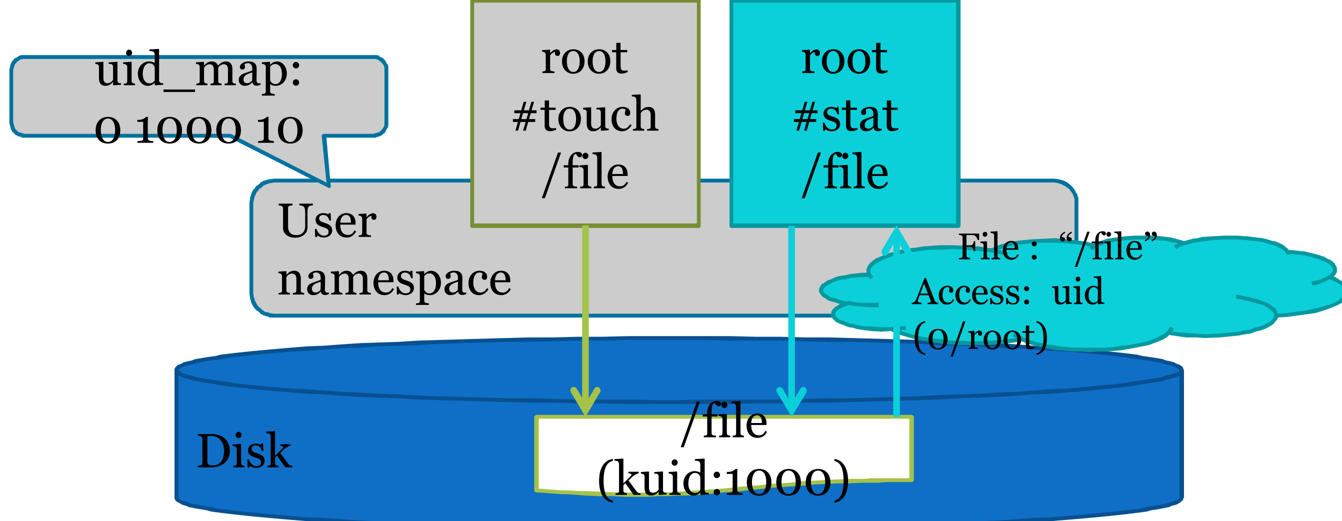
User Namespace

- 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

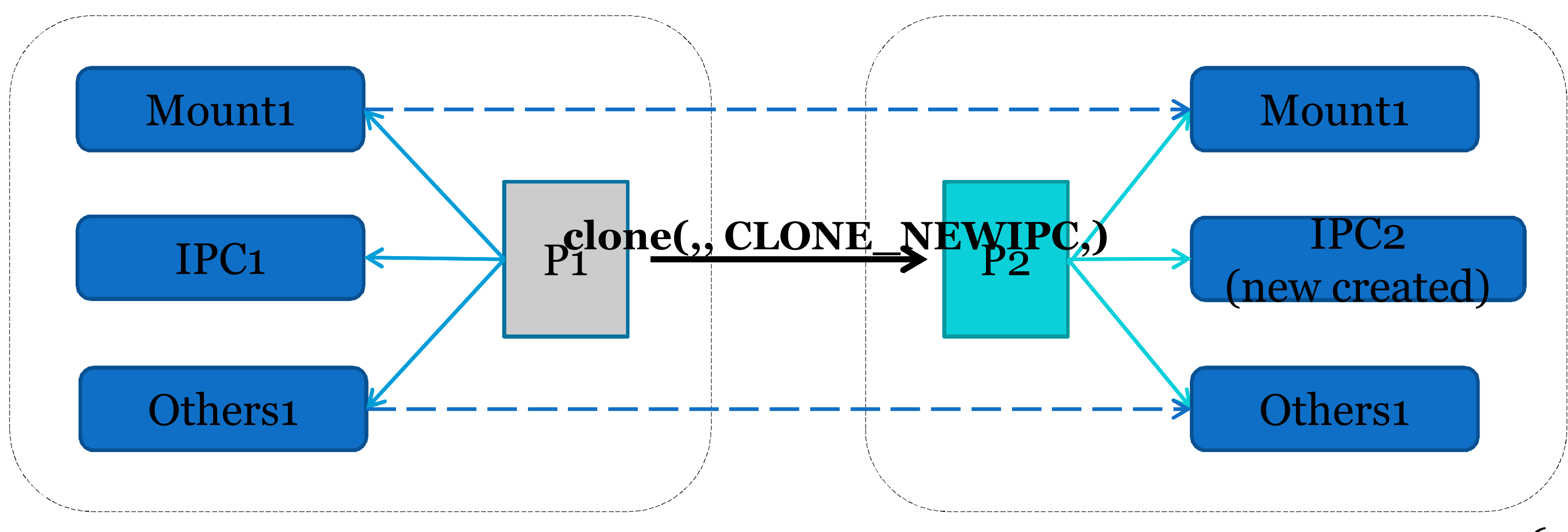
```
int clone(int (*fn)(void *), void *child_stack,
                int flags, void *arg, ...);
```

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

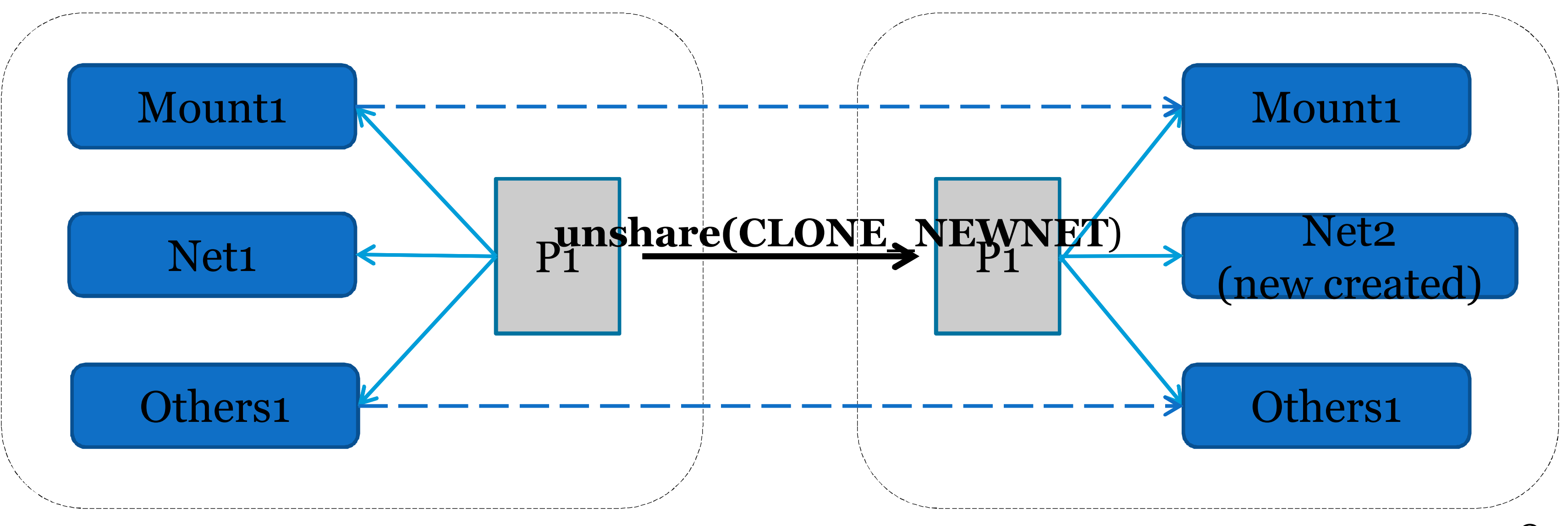
  int unshare(int flags);

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

Rendszerhívások

- unshare

net namespace2 létrehozása



Rendszerhívások

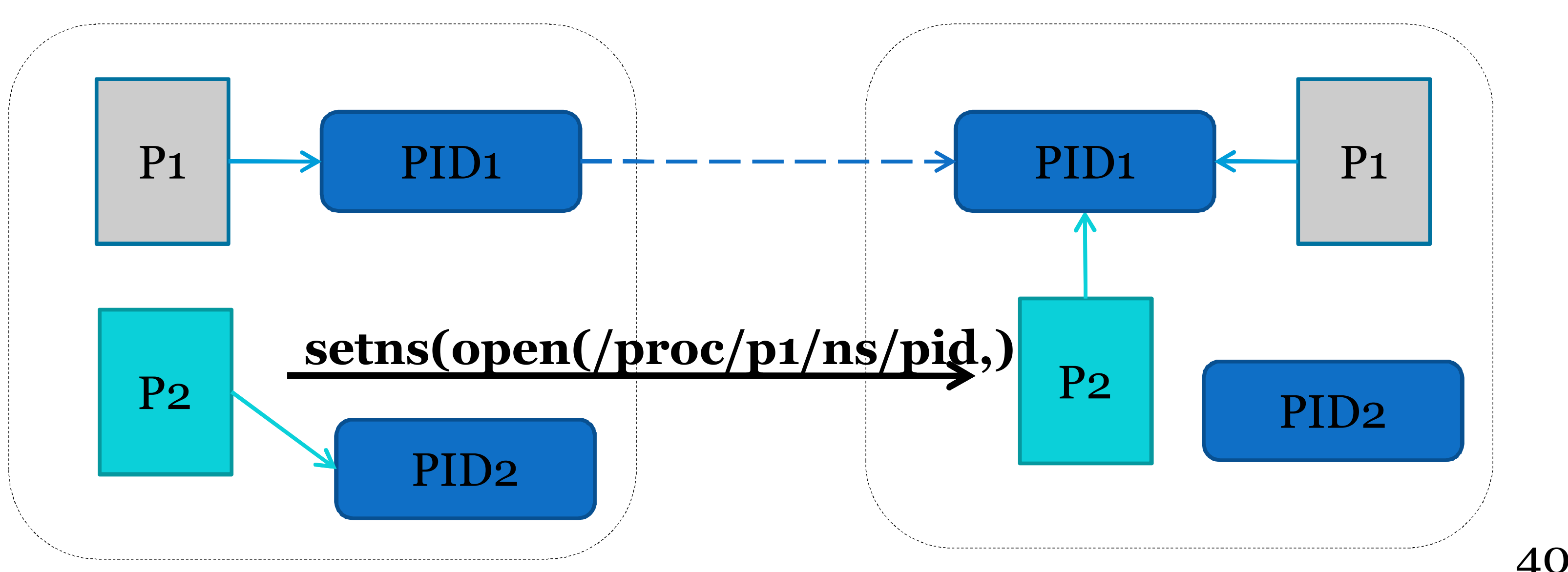
- setns
  int setns(int fd, int nstype);

  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)

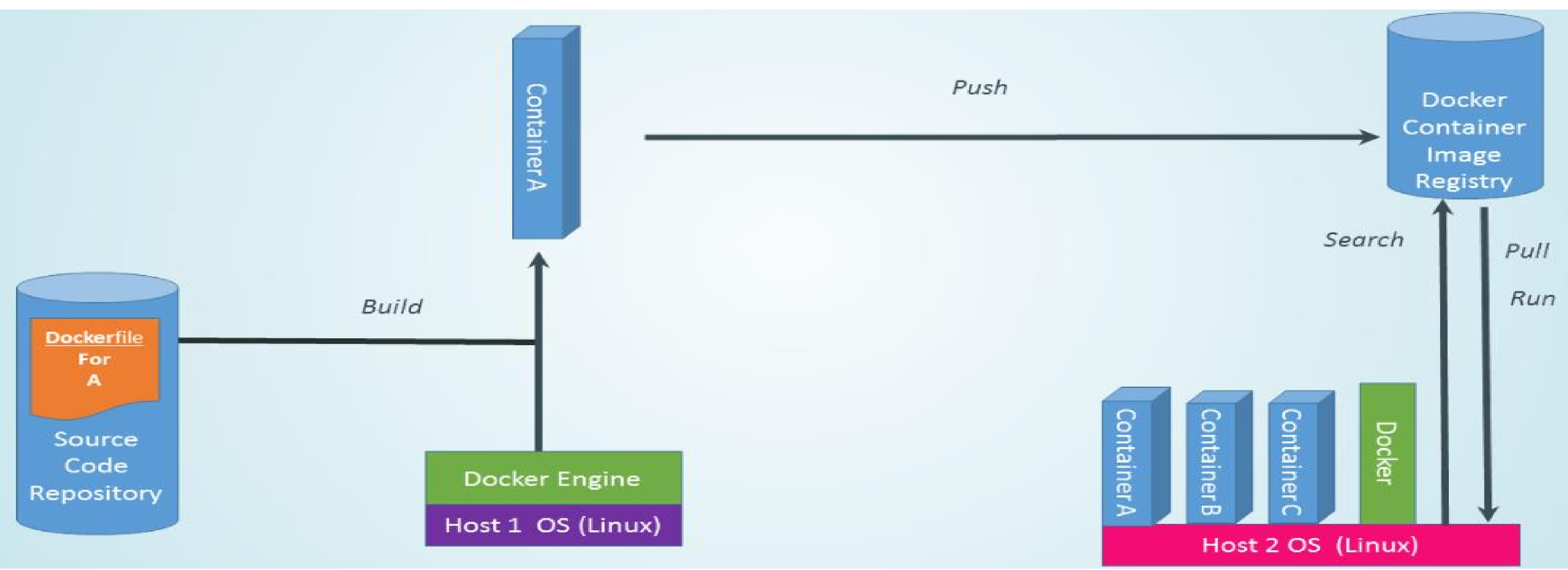
Mi a Docker?

- 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

Docker terminológia

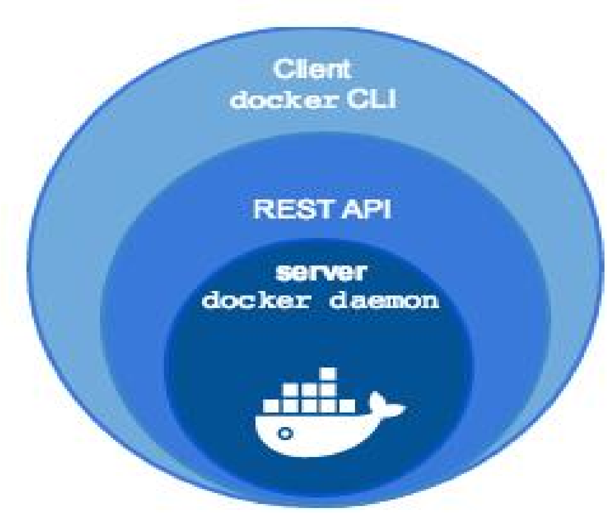
- Képfájl (image) = egy VM-nek megfelelő fájl együttes, amely tartalmaz minden olyan kiegészítést (lib, db, config, stb), ami szükséges az igényelt alkalmazás futtatásához
- Konténter (container) = egy Docker image futtatott példánya
- Tárház (registry) = képfájlok tára
  - Alapesetben helyi (on-host)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

Mi a Docker?



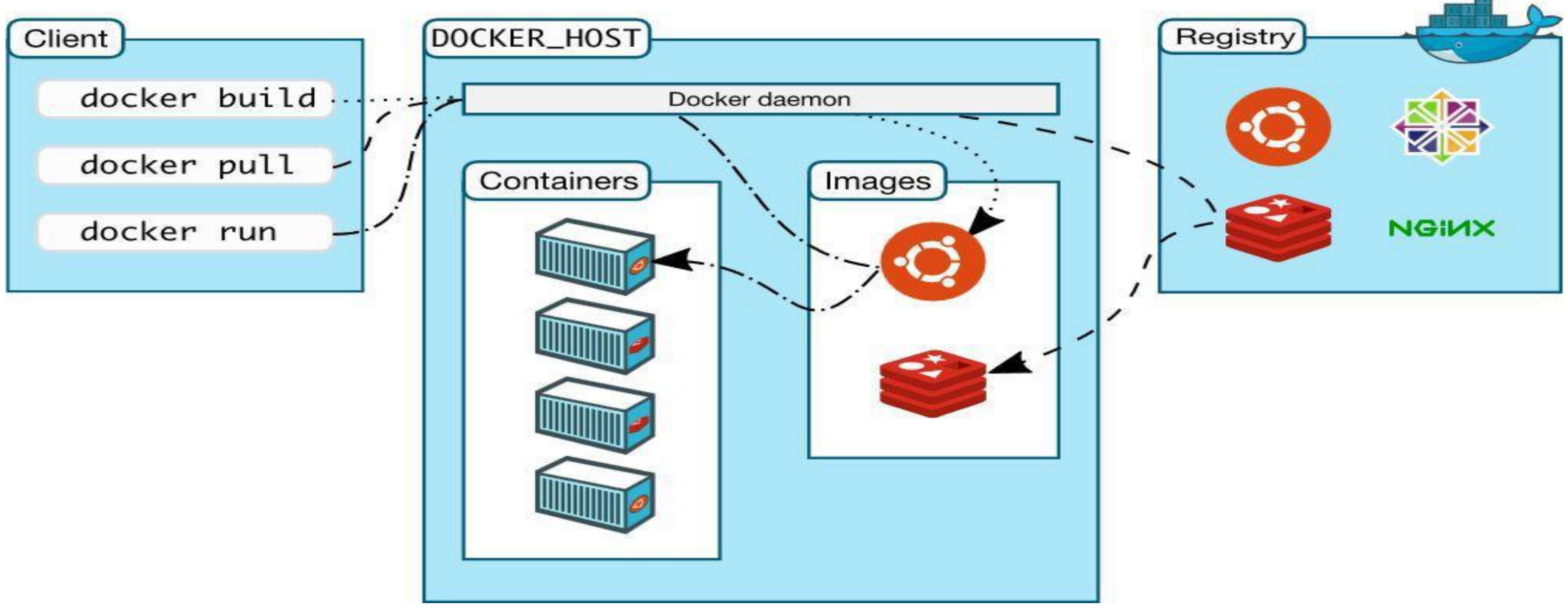
Docker Engine

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



Docker rendszer áttekintés

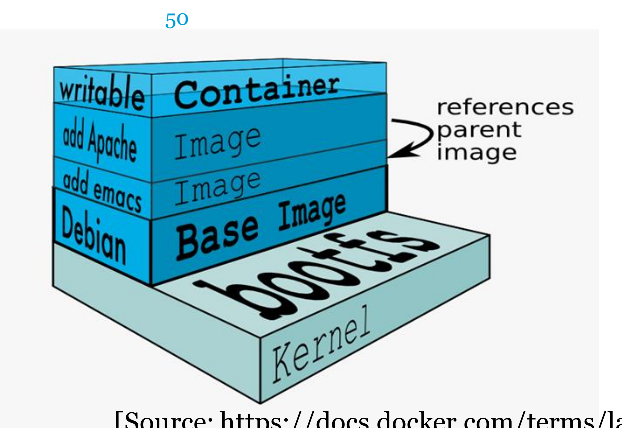
49



Docker képfájlok

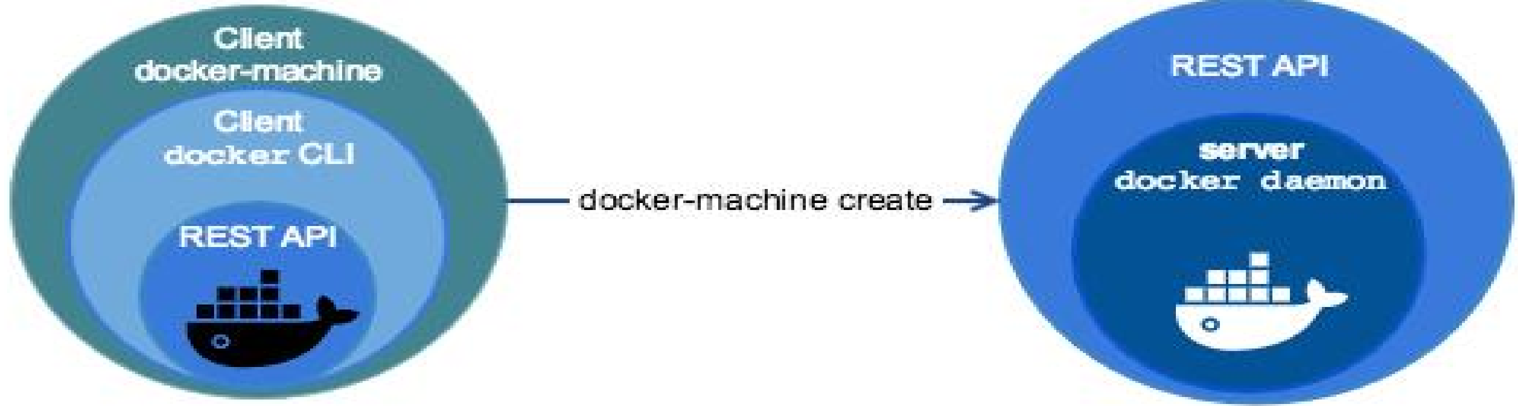
- Rétegektből áll
- Unió típusú fájlrendszer
  - union file system
  - Az összes rétegből egy képfájlt generál
  - A rétegeket tömörítve tárolja
- Sablon alapján létrehozva
  - Dockerfile
  - Kiinduálsi pint: base image (e.g. ubuntu, fedora, stb.)
  - Saját szinatxis az újabb rétegek hozzáadásához
-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

50



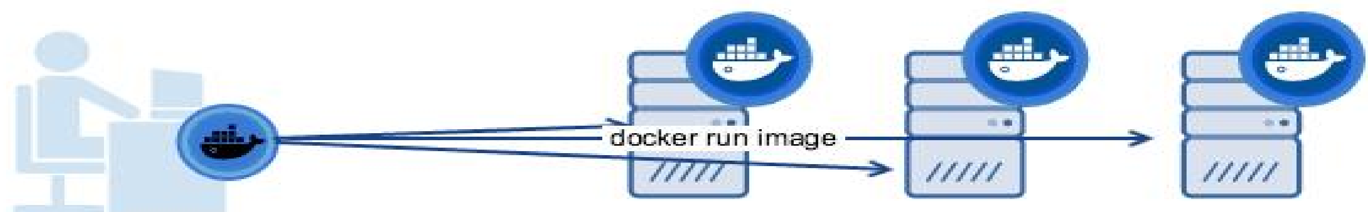
[Source: https://docs.docker.com/terms/layer/]

Docker Machine



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

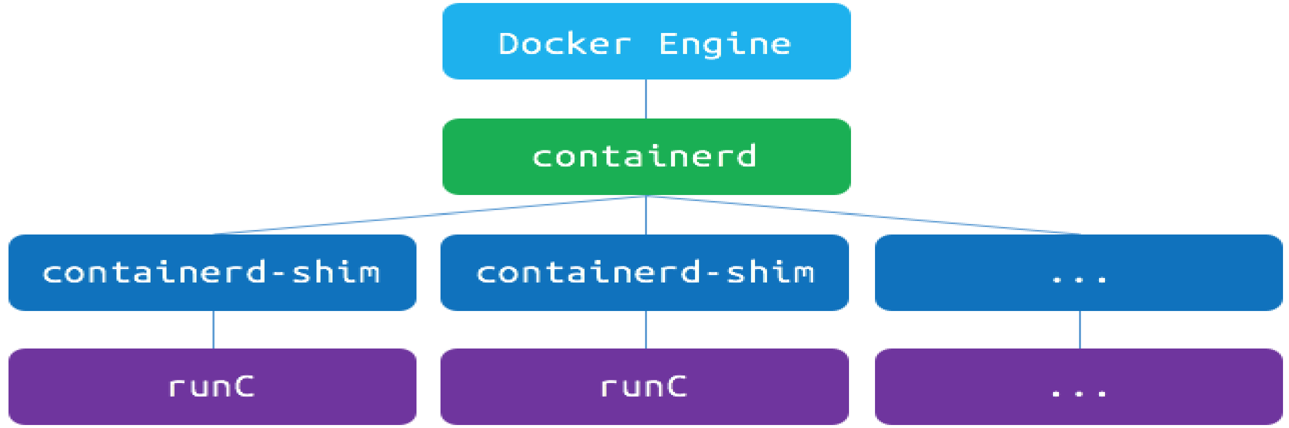
Docker Machine



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

53

Runtime



54

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

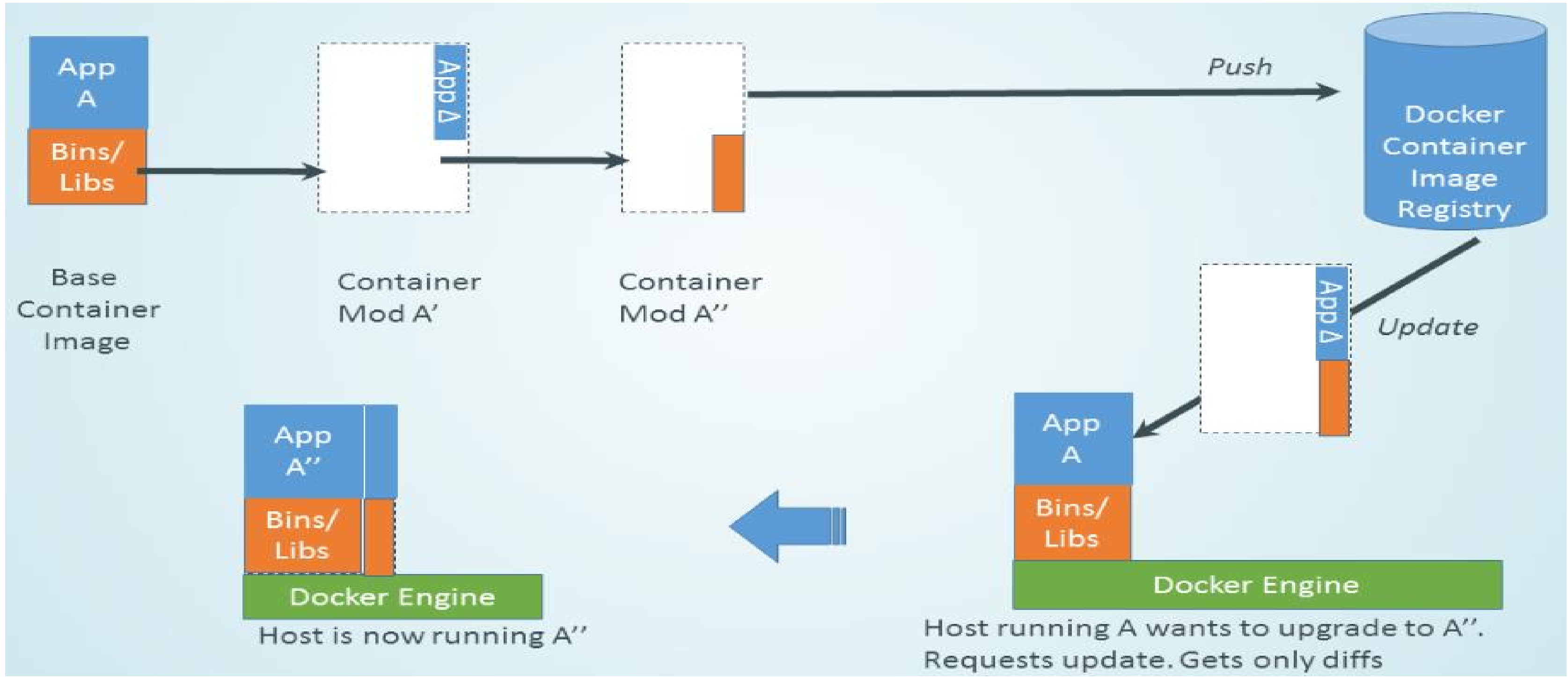
- FROM - egy már létező képfájlból indul ki az új képfájl; ez lesz az alap réteg
  - pl. egy Linux disztribúció
  - Gyakran egy lecsupaszított Linux képfájlt használnak (alpine, busybox)
- COPY – fájlokat másol át a host adott könyvtárából (directory) a képfájlba (pl. konfig, forráskód, szkript)
- RUN – 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?



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 vs. VM

63

- **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

Biztonság?

66

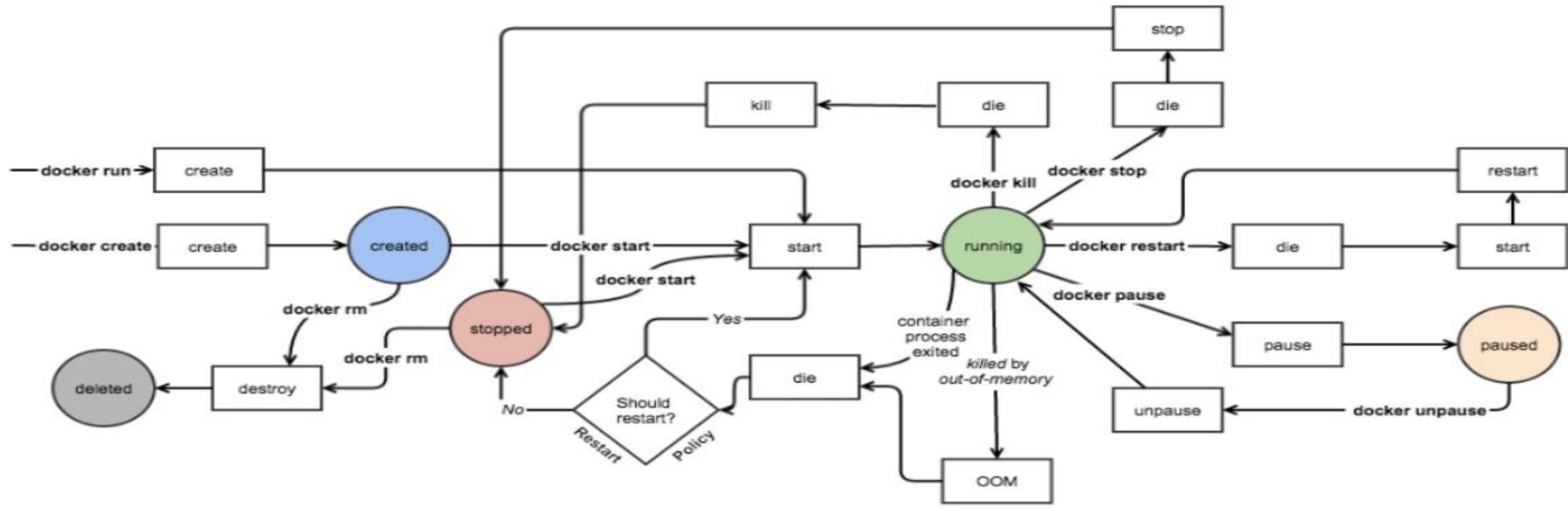
- Docker elérése REST API-n keresztül HTTP felett?
  - Authentikáció!
- Docker démon root jogokkal fut
  - A konténerek már OK
  - „visszanyúlhatnak”?
- docker-1.3 --cap-add, --cap-drop
  - man capabilities
  - „overview of Linux capabilities”
  - „Starting with kernel 2.2”
  - „per-thread attribute”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

68



- die, kill ≠ destroy