
# Konténer alapú virtualizáció

Docker alapok

# Tematika

- Fizikai gépektől a konténerekig
- Linux konténerek – technológiai háttér
- Docker alapok és hiányosságok
- Elosztott konténerkezelés - Kubernetes

# Pár devops fogalom

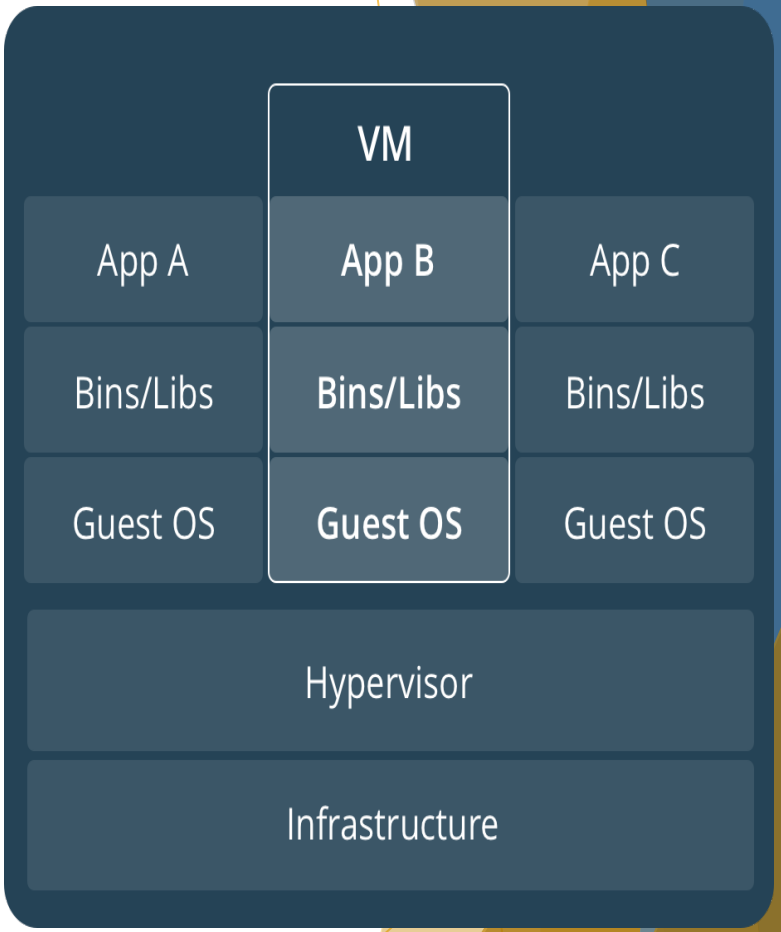
- Immutable infrastructure
- Infrastructure as Code
- Continuous Integration
- Continuous Delivery/Deployment
- Automation, automation, automation!

# Fizikai gép

- “Vas”
- Egy gépen több alkalmazás
- Szeparáció: külön OS userek, virtualenv, stb.
- Például: web hosting
- Probléma: inkompatibilis csomag igények, erőforrás-kihasználtság

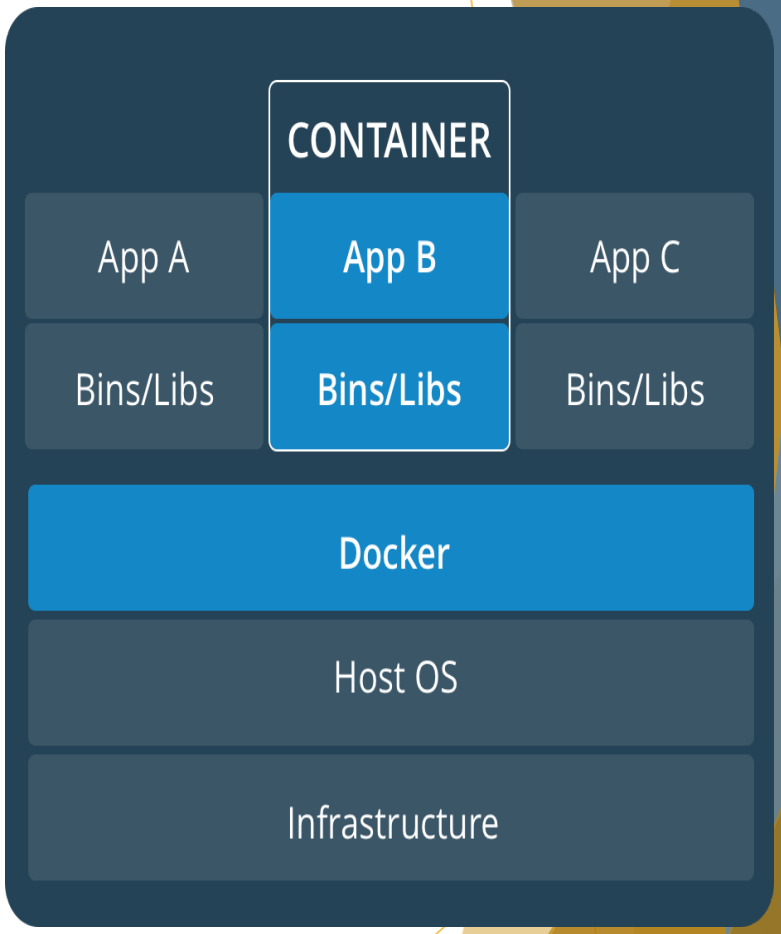
# Virtuális gép

- Egy fizikai gépen több
- Szeparáció: saját kernel
- Példa: Infrastucture as a Service (IaaS)
- Probléma: magas önköltség (1 app / VM?)



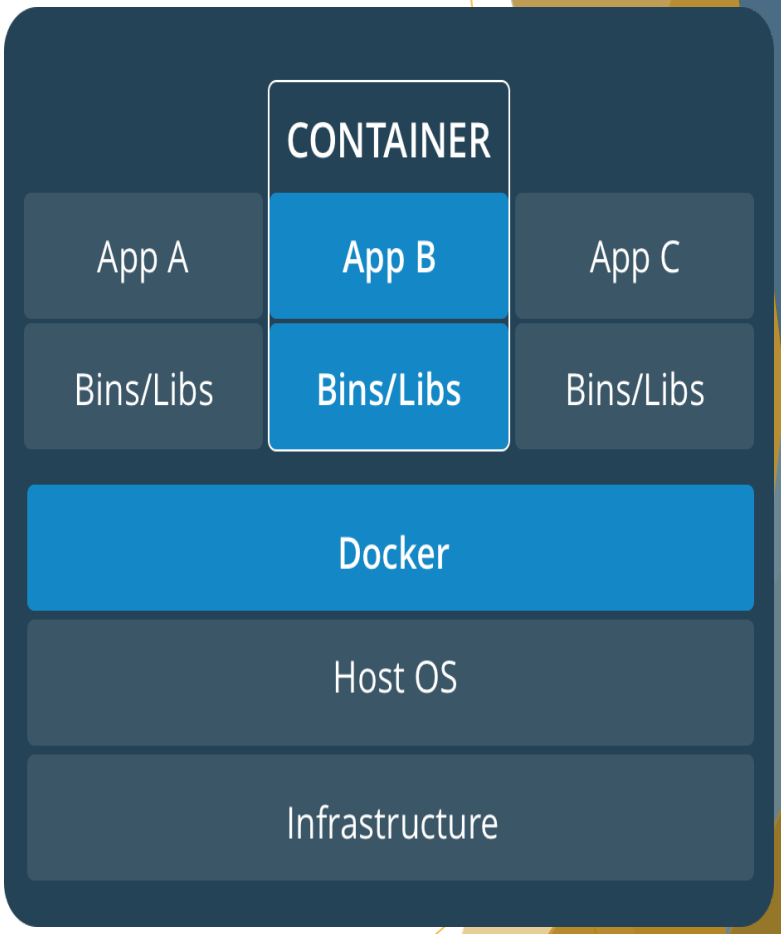

# Konténer

- Egy kernelben több
- Minimalista: csak az alklalmazás és annak függőségei
- Jobb erőforrás-kihasználtság

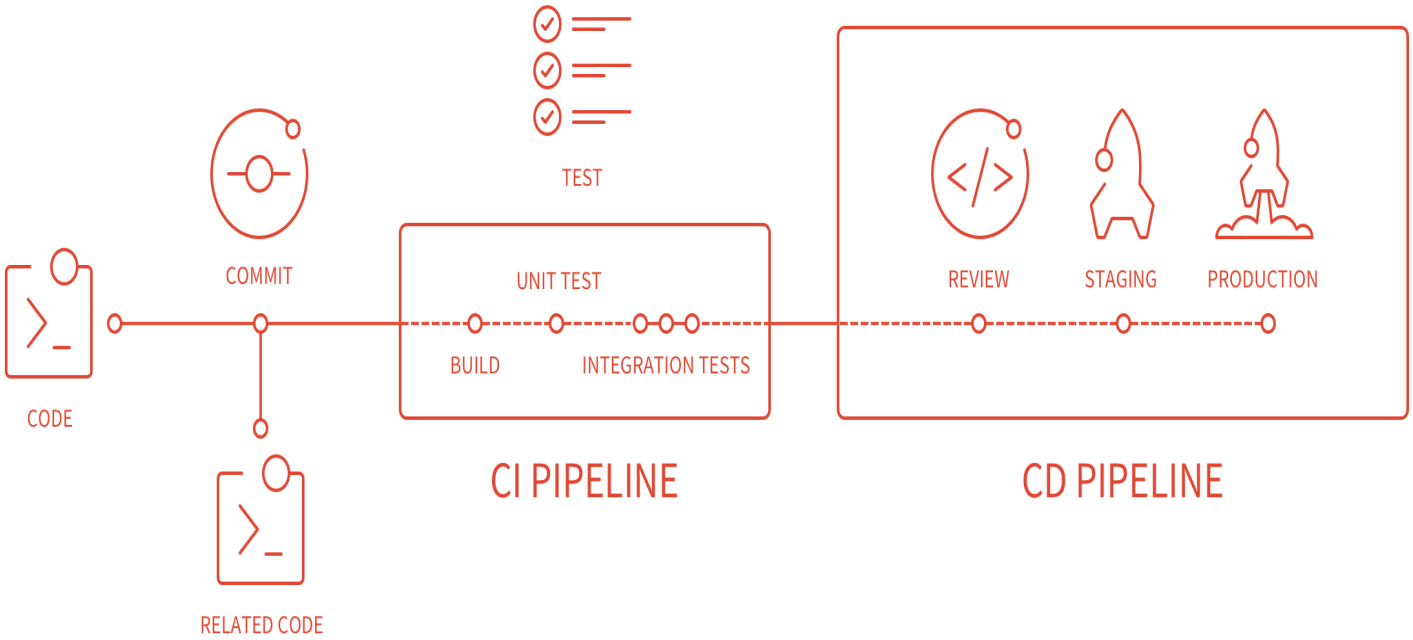



# Docker konténer

- “A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.”




# Konténer: munkadarab




# Alklalmazás vs. OS konténer

- OS konténerek
  - Linux Containers (LXC)
  - Solaris Zones
- Alkalmazás konténerek
  - Docker
  - Rocket (rkt)

# Technológiai háttér

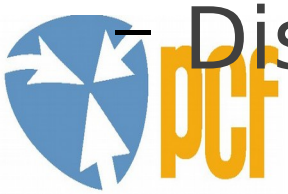
- Google fejlesztések (saját konténer technológia)
- Linux kernel feature-ök:
  - cgroups (irányítás)
  - namespaces (szeparáció)
- “Kézzel hajtós” konténer: https://youtu.be/sK5i-N34im8

# Miért Docker?

- Elterjedt (6 milliárd letöltés 2016-ban)
- Könnyen használható (főleg fejlesztői oldalról)
- Hatékonyan automatizálható

# Mit ad a Docker?

- Egységes API
  - Konténer irányítása (CLI, REST)
  - Konténer létrehozásra (Dockerfile)
- Konténerek csomagolása
  - Újrahasználhatóság
  - Verziókezelés
  - Disztibúció (registry)



# Docker Image

- A “becsomagolt” konténer
  - Image vs konténer ~ bináris fájl vs processz
- Base image: minimális OS eszközök
- Verziózás: tagek segítségével
- Rétegekből áll
- Előállítás: Dockerfile alapján

# Rétegek (layers)

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

```
IMAGE       ...     CREATED BY                      SIZE

fdd93d9c2c60  ... /bin/sh -c CMD ["npm" "start"]
 0 B

e9539311a23e  ... /bin/sh -c EXPOSE 8080/tcp
  0 B

995a21532fce  ... /bin/sh -c COPY dir:50ab47bff7
  760 B

ecf7275feff3  ... /bin/sh -c npm install  3.439 MB

334d93a151e ... /bin/sh -c COPY file:551095e67
265 B

86c81d89b023 ... /bin/sh -c WORKDIR
/usr/src/app  0 B

7184cc184ef8  ... /bin/sh -c mkdir -p /usr/src/app
0 B

530c750a346e  ... /bin/sh -c CMD ["node"]
0 B
```

# Registry vs Hub

- Image-ek disztribúciójára szolgál
- Publikus image tároló: https://hub.docker.com/
- Image le- és feltöltés: pull, push
- Lokális tárolás: docker images
- Verziókezelés: tag-ek (latest, 1.0)
- Privát hub: registry

# Konténer indítás

- Letöltésre kerül a megfelelő taggel ellátott image
- A konétner saját belső fájlstruktúrája leképzeződik a fájlrendszerre
  - Tipikusan: /var/lib/docker/...
- Minden változás ebbe a könyvtárba kerül (copy-on-write), az image NEM változik

# Perzisztencia

- Van lehetőség perzisztens tárolásra
- Példa: adatbázis konténer (mysql, postgres)
- Volume-ok csatolásával
  - Lokális fájlrendszer
  - Hálózati megosztás
  - Másik konténer

# Konténer konfiguráció

- Csak default értékeket érdemes image-be építeni
- Bemenetként környezeti változókkal (valamilyen külső kulcs-érték tárolóból – pl. Redis)
- Konfigurációs könyvtár becsatolása volume-ként

# Konténer összeállítása

- CLI-n keresztül (docker run ...)
- Függőségek kezelése nehézkes (scriptelési feladat)
- Megoldás: docker-compose (YAML)
- Deklaratív leírás
- Újra felhasználható
- De: külső verziókezelés, tárolás szükséges

# Logolás

- Hibakereséshez, auditáláshoz
- Alkalmazások: alapértelmezetten fájlrendszerre
- Kivezetés STDOUT-ra/STDERR-re
- Ezek automatikusan gyűjtésre kerülnek
- Használható külső loggyűjtő rendszer is (pl. syslog)

# Docker hiányosságok

- Egy gépes megoldás
- Monitorozás: van, de fapados
- Online konténer frissítés: nincs
- Skálázás, terheléselosztás: nincs
- A használt portok karbantartása nehézkes

# Elosztott Docker

- Docker Swarm – 1.12-től
  - A problémák jelentős részére megoldást jelent
  - Egyszerű, de fapados
- Kubernetes
  - Jól bejáratott technológia
  - Jól konfigurálható, de összetett

# Kubernetes alapok

- Moduláris felépítés (szinte minden plug-in)
- Elvárt állapot alapú megközelítés
- Állapot leírása: YAML
- Namespace alapú láthatóság

# Pod

- Ütemezési egység
- Akár több konténer is lehet egyben
- Saját IP cím
- Elérés: service-eken keresztül

# Deployment

- Konténerek számának skálázására
- Lehet automatikus (pl. CPU használat alapján)
- Beépített, kiesés nélküli frissítés:
  - Új példányok indítása
  - Régi példányok leállítása

# Service

- Pod-ok címzése és terheléselosztás
- Label-ek alapján
- Elérés
  - NodePort: a node-ok saját címén is elérhető
  - ClusterIP: saját IP cím
  - LoadBalancer: cloud környezetben

# ConfigMap és Secret

- Kulcs-érték párok tárolására
- A Pod-okban hivatkozhatóak
  - Érték alapján, környezeti változóban
  - Fájlként becsatolva
- Konfiguráció és alkalmazás szétválasztása

# Ingress

- A clusterben futó szolgáltatások külső eléréséhez
- Jellemzően HTTP forgalom
- Név alapú azonosítás (~VirtualHost)
- Ingress → Service → Pod

# Helm

- Összetett alkalmazás architektúra leírására
- Verziókezelés
- Sablon támogatás

# Kubernetes telepítés

- Összetett feladat
- kubeadm: egyszerű, megismételhető
- Fejlesztői környezet: minikube
- Ajánlott “irodalom”:

  Kesley Hightower: Kubernetes The Hard Way

  https://github.com/kelseyhightower/kubernetes-the-hard-way