



**PEOPLE COME FIRST**  
INFORMATIKAI SZAKÉRTŐK EGYESÜLETE

# Docker & Kubernetes

Konténer alapú virtualizáció

# Tematika

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

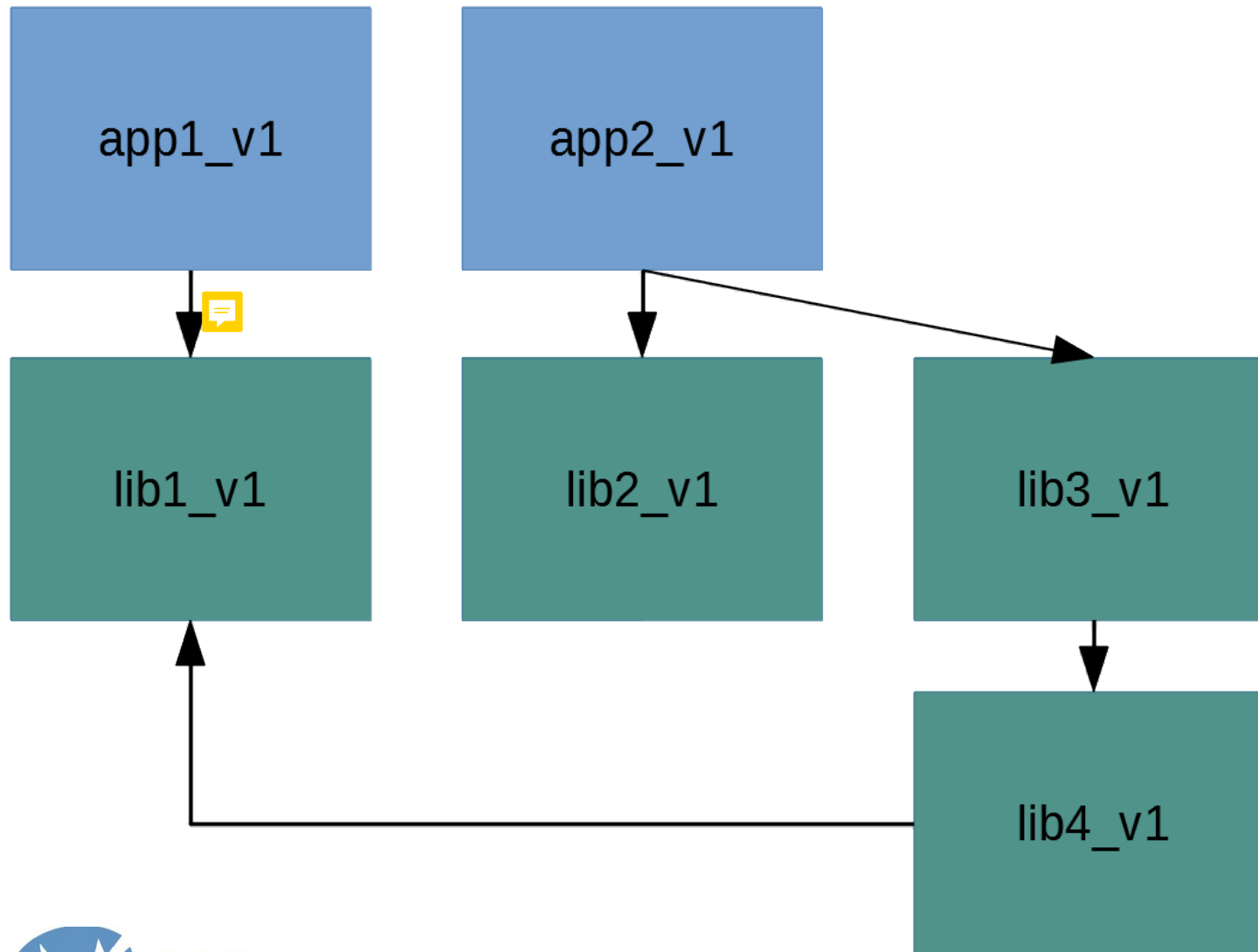


# Fizikai gép

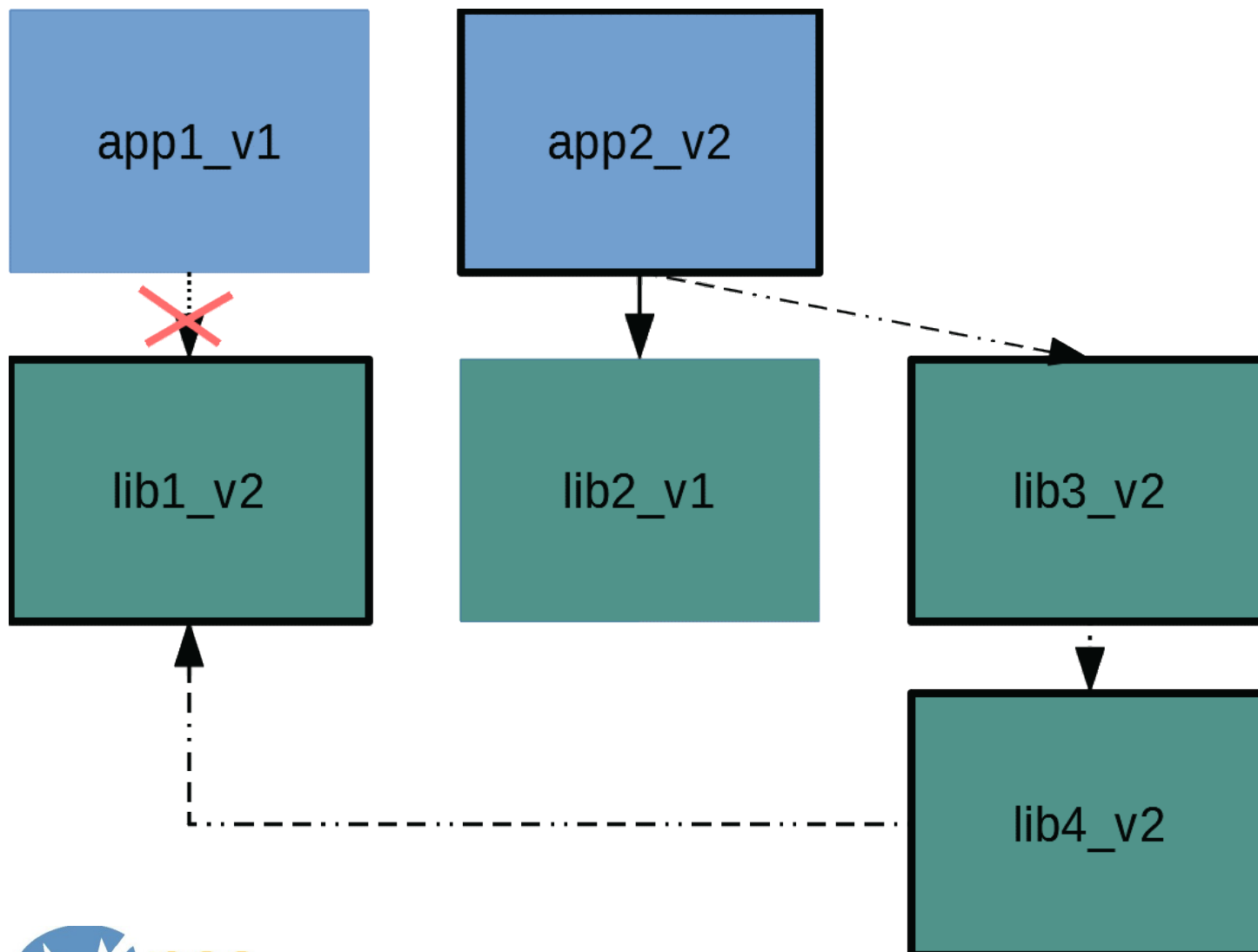
- ▶ “Vas”
- ▶ Egy gépen több alkalmazás
- ▶ Szeparáció: külön OS userok, ~~virtualenv~~, stb.
- ▶ Például: web hosting
- ▶ Probléma: inkompatibilis csomag igények, erőforrás-kihasználtság



# Függőségek



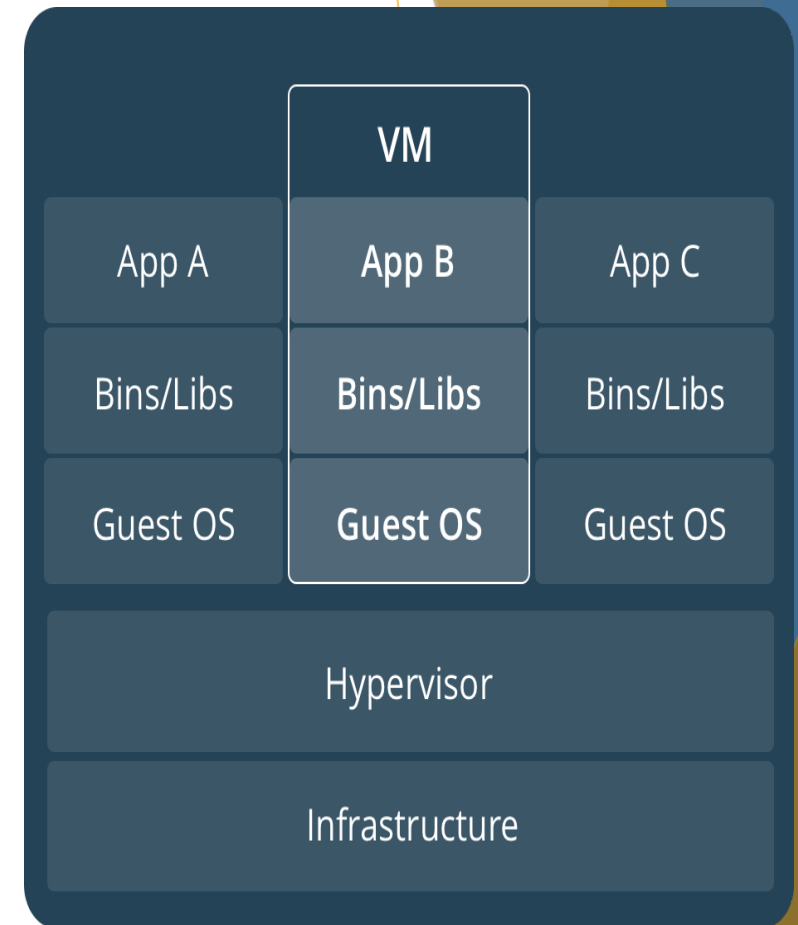
# Függőségek





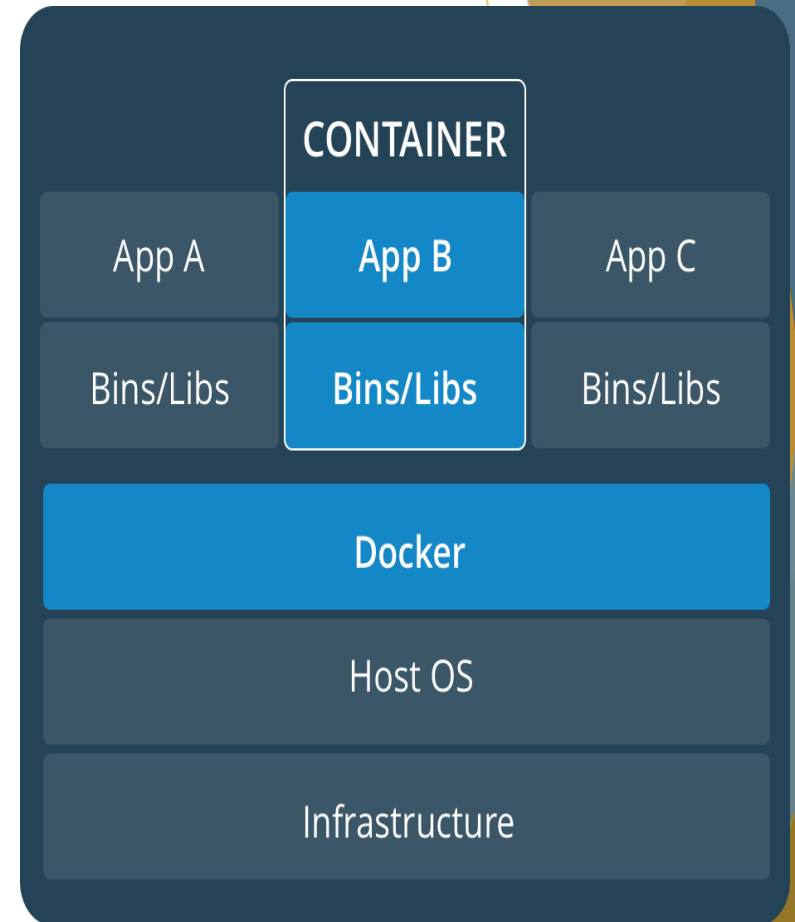
# Virtuális gép

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



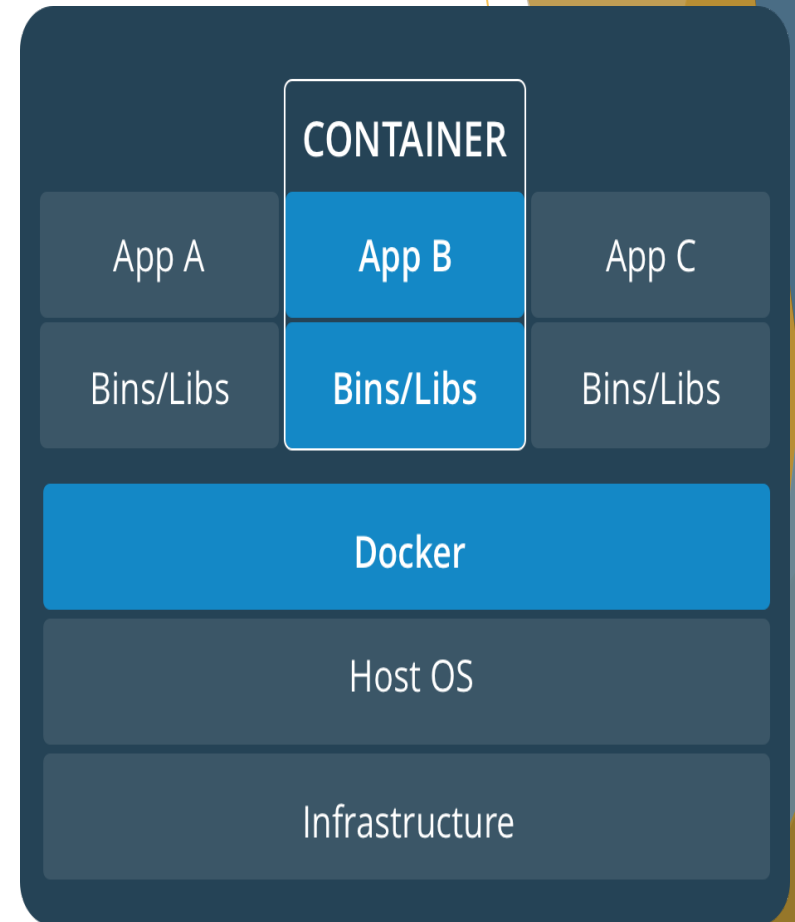
# Konténer

- ▶ Egy kernelben több
- ▶ Minimalista: csak az alkalmazás 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.”








# App vs. OS konténer

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




# Docker 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 vezérlése (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	IMAGE	...	CREATED BY	SIZE
# Create app directory RUN mkdir -p /usr/src/app WORKDIR /usr/src/app	fdd93d9c2c60	...	/bin/sh -c CMD ["npm" "start"]	0 B
# Install app dependencies COPY package.json /usr/src/app RUN npm install	e9539311a23e	...	/bin/sh -c EXPOSE 8080/tcp	0 B
# Bundle app source COPY . /usr/src/app	995a21532fce	...	/bin/sh -c COPY dir:50ab47bff7	760 B
EXPOSE 8080 CMD ["npm", "start"]	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






# Hogy indul egy konténer?

- ▶ Letöltésre kerül a megfelelő taggel ellátott image
- ▶ A konténer saját belső fájlstruktúrája leképeződik 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
- ▶ Bemenetként környezeti változókkal, de valamilyen külső kulcs-érték tárolóból (pl. Redis)
- ▶ Konfigurációs könyvtár becsatolása volume-ként 




# Konténerek összeállítása

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




# Logolás

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

# Docker hiányosságok



- ▶ Egy gépes megoldás (kernelen belüli)
- ▶ Monitorozás: van beépített, de fapados
- ▶ Online konténer frissítés: nincs
- ▶ Skálázás, terheléselosztás: nincs
- ▶ A használt portok karbantartása nehézkes

# Elosztott Docker

- ▶ Docker Swarm - 1.12-től beépített
  - A problémák jelentős részére megoldást jelent
  - Egyszerű, de fapados
- ▶ Kubernetes 
  - Jól bejáratott technológia (Google)
  - Jól konfigurálható, de összetett




# Miért Kubernetes?

- ▶ Elterjedtség és támogatás
- ▶ Kiváló Docker integráció 
- ▶ Nagyon jól testreszabható 
- ▶ A nehézségek nagy részére már van megoldás
- ▶ Erős felhő támogatás (GKE stb.)



# Kubernetes alapok

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





# Kubernetes Node

- ▶ Két node típus:
  - Worker: alkalmazás konténerek
  - Master: a cluster vezérléséhez szükséges komponensek
- ▶ Kubelet: a node-on futó komponens
  - Konténerek indítása/leállítása
  - Állapot jelentése a Master felé

# Master komponensek

- ▶ API szerver: komponens kommunikáció
- ▶ Etcd: állapot tárolása  
<https://raft.github.io/>
- ▶ Kube-scheduler: hol fusson a Pod?
- ▶ Kube-controller-manager: az elvárt állapot alapján az aktuális állapot módosítása




# Kubernetes networking

- ▶ Minden konténernek tudnia kell kommunikálnia bármely más konténerrel NAT használata nélkül
- ▶ Minden node-nak tudnia kell kommunikálnia bármely konténerrel (és vissza) NAT használata nélkül
- ▶ A konténer saját magát ugyanazon az címen látja, mint bármely másik konténer vagy Node





# Pod

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




# Perzisztencia

- ▶ Háttértárolók elérésére: volume
- ▶ Ha nem csak időlegesen kell:  
PersistentVolume 
- ▶ További absztrakció:  
PersistentVolumeClaim 
- ▶ Ezeket az objektumokat vagy a Pod definícióban kell hivatkozni (adott csatolási ponthoz)



# Service



- ▶ Pod-ok címezése és terhelésselosztás
- ▶ 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
  - Fájlként becsatolva
- ▶ Konfiguráció és alkalmazás szétválasztása



# Deployment

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





# Helm



- ▶ Összetett alkalmazás architektúra leírására
- ▶ Verziókezelés
- ▶ Sablon támogatás



# Kubernetes telepítés

- ▶ Összetett feladat (rengeteg komponens)
- ▶ Könnyű és megismételhető telepítés  
kubeadm segítségével
- ▶ Ajánlott “irodalom”:

Kesley Hightower: Kubernetes The Hard Way

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



# Kubernetes telepítés

- ▶ Összetett feladat (rengeteg komponens)
- ▶ Könnyű és megismételhető telepítés kubeadm segítségével
- ▶ Ajánlott “irodalom”:

Kesley Hightower: Kubernetes The Hard Way

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



Köszönöm a figyelmet!

