HAKAP ZH

(Docker, Kubernetes, Severless)

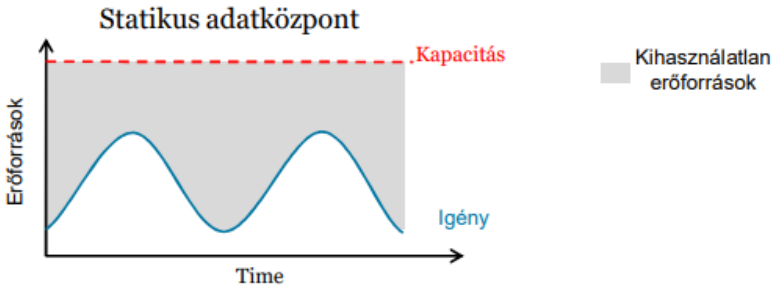
Hálózati erőforrásmegosztás

Korai hálótatmegosztás: mi az, milyen topológiák, nagyméretû erõforrás rendszereket alakítottak ki, Amdahl törvénye 20-23 fóliák

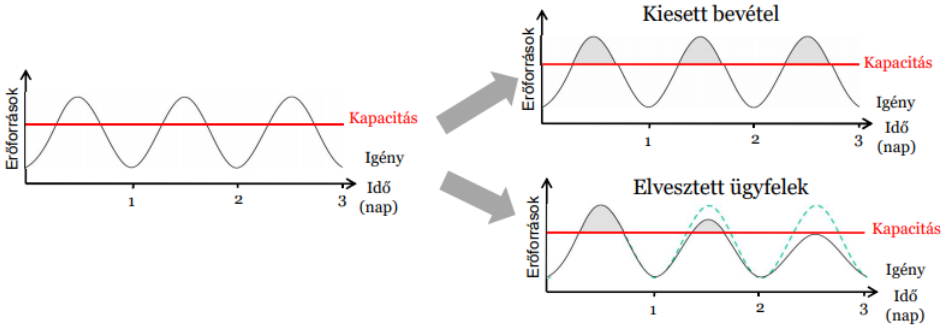
- **„Shared resource”:** Számítógépek közösen használják az erőforrást (programok, fájlok, nyomtatók, Intranet, LAN)
  - Topológiák:
  - Workgroup (P2P)
  - Kliens szerver
- Korai LAN-nál több alternatíva:
  - Apple Filing Protocol / AppleTalk (Apple)
  - SMB / TCP/IP (Microsoft)
  - Network File System (NFS) / TCP/IP (Unix)
  - NetWare Core Protocol (NCP) / SPX (Novell)
- Nagyméretű számítási erőforrás rendszerek:
  - Clusterek
  - Nagysebességű helyi hálózaton keresztül összekötött ugyanolyan számítógépek
  - **Amdahl törvénye** megadja a párhuzamosítással elérhető **nyereséget**:
    A nem párhuzamosítható része a feladatnak korlátozza a gyorsítás mértékét

Statikus adatközpontok hátrányai túlbiztosítás és alultervezés esetén - 25-27 fóliák

- Túlbiztosítás = alul kihasználás



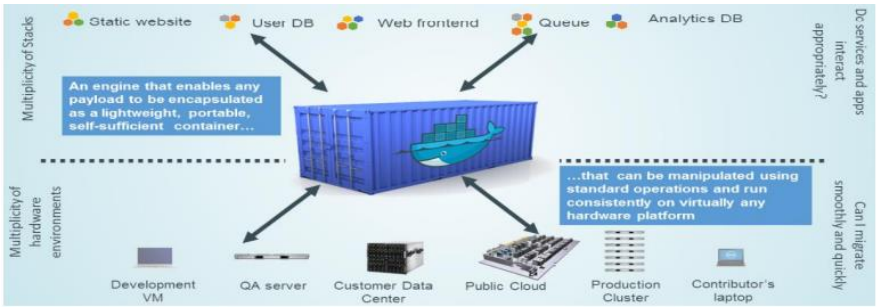
- Alultervezés hátrányai



Linux konténerek

Konténerek, mi az, mire jó, mi a konténer-metafora... - 3-7, 19

- **Konténer metafora:** áruszállítás probléma
  - Logisztikai (management) kérdés merül fel a hétköznapi szállítási feladatok kivitelezése során is. Sok szállítási platform van, sok árutípusra → egy közös csomagoló-egység létrehozása szükséges→ **KONTÉNER** (egységes, köztes szállítási egység)



  - Szállítási platform → végrehajtási környezet
  - Árutípus → számítási feladat

- **Konténer:**
  - OS szintű virtualizáció, elkülönített erőforrások, de a kernel közös
  - Pro:
    - Könnyed, gyorsan deployolható, hordozható, skálázható, flexibilis
  - Con:
    - biztonsági problémák: deamon-t futtat, aminek root jog kell
    - default user a root a konténerben
    - nincs hardver izoláció, mint VM-eknél
  - Alkalmazások csomagszintű terjesztésére használják (packaging)
  - Microservices architektúra támogatására:
    - Komplett alkalmazások kisebb részekre bontva melyek együttműködnek
    - Komponensként skálázható
    - Sok új gond (együttműködés, komplexitás, nehezebb debug-golás)
  - SOA - Service Oriented Architecture
  - CMT (Container Management Tools): ezen keresztül nyúlunk bele

Linux névterek, cgroups - 21-29, 32

- **Névterek (namespace):** rendszer erőforrásainak elszigetelése (MIT használhatsz)
  - Egy mindenki által elérhető rendszererőforrást úgy mutat egy folyamat számára, mintha saját különálló példánya lenne abból. Vagy egyszerűbben: meghatározza, hogy egy folyamat mit láthat. Egy névtér meg is osztható folyamatok között.
  - Amikor elindítunk egy konténert, a rendszer létrehoz jó pár névteret és ellenőrzőcsoportot. A névterek segítenek két konténer elkülönítésében.
    - **Mount**
      - csatolások, létrehozáskor lemásolódik
      - saját fájlrendszer
    - **UTS névtér**
      - UTS = UNIX Timesharing System
      - a folyamat más hostnevet és domainnevet kaphat (UTS-infó)
    - **IPC névtér**
      - IPC = InterProcess Communication
      - folyamatok közti kommunikáció
    - **Hálózati (Net)**
      - Hálózati erőforrásokat rejti el
      - létrehozáskor csak egy visszacsatoló interfészt tartalmaz. minden interfész csak egy névtérben lehet, de át lehet őket mozgatni. saját irányítótábla, tűzfal, IP címtartomány

        - **PID**
            - PID = Process ID
            - Hierarchikus rendszerben virtualizálja
            - folyamatazonosítók, egymásba ágyazott szerkezet
        - **felhasználó (User)**
            - Biztonsághoz köthető felhasználói attribútomok izolálása
            - egy folyamat gondolhatja hogy ő admin felhasználó úgy, hogy közben kívülről nézve nem az
            - Csak a szülő felhasználó (parent user) NS állíthat be mappelést

- **Cgroups:** erőforrás felhasználás korlátozás és annak számontartása (MENNYIT használhatod)
    - Tárolás (mem)
    - Számítás (CPU)
    - Kommunikáció (blkio)
    - Eszköz (dev)

A három rendszerhívás (clone, unshare, setns) és egy-egy mondatban, hogy mire való - 33-41

Linux névterek az unshare, clone illetve setns rendzserhívások segítségével lehet létrehozni, illetve manipulálni.

- **Unshare**
    - A folyamat a végrehajtási kontextus azon részeit szétválasztja, amelyeket jelenleg más folyamatokkal osztanak meg.
- **Clone**
    - A clone segítségével, más részeket (például virtual memory),meg lehet osztani, miközben létrehozunk egy folyamatot
    - jelzők, amelyek megadják, hogy melyik új névteret kell áttelepíteni, megosztani
- **Setns**
    - Belép a fájleíró által megadott névtérbe

LXC – 41

Linux Container, Az LXC egy userspace interfész a Linux kernel elszigetelési funkciókhoz. Egy erőteljes API-n és egyszerű eszközökön keresztül lehetővé teszi, hogy a Linux-felhasználók könnyen létrehozhassanak és kezelhessenek rendszer- vagy alkalmazási konténereket.

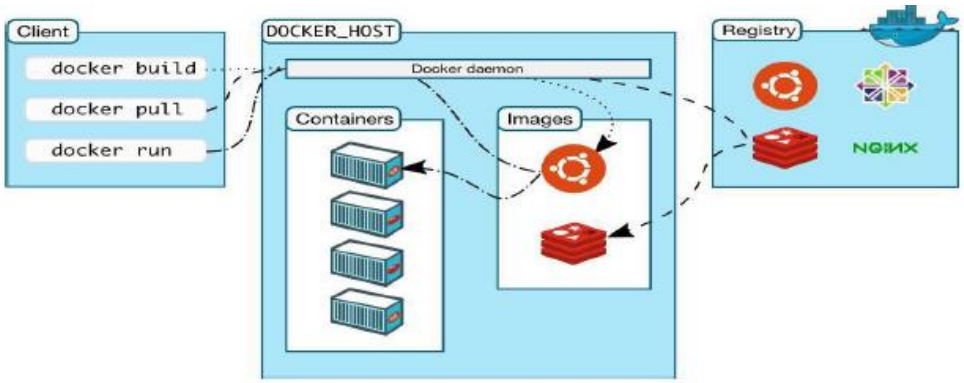
- **Libvirt LXC:** userspace container management tool
    - libvirt driver-ként megvalósítva
    - Konténer menedzsment
    - Névtér létrehozás
    - Privát fájlrendszer kezelése a konténeren belül
    - Konténer eszközeinek létrehozása
    - Cgroup által vezérelt erőforrások

Docker

Docker, architektúra, elõny-hátrány - 2-4, 19, 21-23

- **Docker** = Linux container engine, minden docker parancsot ez az engine hajt végre
    - Open Source project
    - 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 architektúra
    - A Docker client-server architektúrát használ. A Docker kliens a Docker deamon-nal beszél, amely a Docker konténerek építésének, futásának és létrehozásának nagyfokú kezelését teszi lehetővé. A Docker kliens és a deamon ugyanazon a rendszeren futtatható, vagy csatlakoztathat egy Docker klienst egy távoli Docker deamon-hoz. A Docker kliens és a deamon REST API, UNIX socket-ek vagy hálózati interfész segítségével kommunikál.

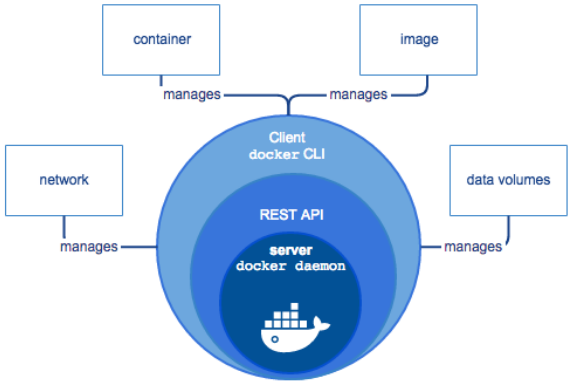


- **Képfájl (image)** = egy VM-nek megfelelő fájl együttes, amely tartalmaz minden olyan kiegészítést (lib, db, config, stb), ami szükséges az igényelt alkalmazás futtatásához
- **Konténter (container)** = egy Docker image futtatott példánya
- **Tárház (registry)** = képfájlok tára
    - Alapesetben helyi (on-host)
    - A Docker cég fenntart egy nyilvános, globális, on-line adatbázist (github-hoz hasonló)
- **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 rendszer alapjai (a kliens pull, build, run parancsait a Docker hoston a Docker daemon hajtja végre, a Docker host-on találhatók a futtatott konténerek, az imagek amelyekbõl létrejöttek), Docker daemon, Docker machine - 5, 6, 9

Docker Engine főbb részei:

- **Szerver**: egy hosszú futású program, ami a docker deamon
- **REST API**: megadja az interfészeket, amit a program használhat, hogy a deamon-nal kommunikáljon, és utasítsa őt, hogy mit tegyen
- **CLI:** parancssori interfész
- A CLI a Docker REST API-t használja a Docker deamon vezérléséhez vagy interakciójához szkriptek vagy közvetlen CLI parancsok segítségével



Docker deamon:
A Docker deamon (dockerd) hallgatja a Docker API kéréseit és kezeli a Docker objektumokat, például a image-k, konténereket, hálózatokat. A deamon más deamon-okkal is kommunikálhat a Docker szolgáltatások kezelésére.

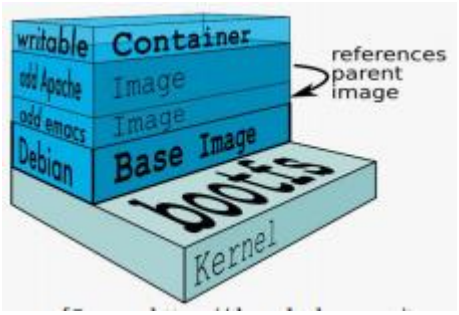
- **Docker** run: letölt MINDENT és futtatja a konténerünket
- **Pull:** lehúzza az image-t a docker registry-ből, és menti a saját rendszerünkbe
- **Build:** saját image létrhozása

} a hoston a deamon hajtja végre

Docker machine:
A Docker Machine olyan eszköz, amely lehetővé teszi a Docker Engine telepítését virtuális gépeken, és menedzselését a hostoknak a `docker-machine` parancs segítségével. Távoli állomásokon konténer kezelés.

Docker képfájlok (aka image) – 7

A Dockerben minden image-en alapul. Az image fájl rendszerek és paraméterek kombinációi.



- Rétegekből állnak
- Csak olvasható template-ek
- Unió típusú fájlrendszer
  - union file system
  - Az összes rétegből egy képfájlt generál
  - A rétegeket tömörítve tárolja
- Dockerfile-ban van definiálva, ahol megadjuk neki azokat a lépéseket ami a létrehozásához szükségesek
- Saját magunk is létrehozhatunk (de csak a registry-ből)
  - Gyakran egy image egy másik image-on alapszik, csak további testreszabásokkal
  - Minden instrukció egy új réteget hoz létre
  - Mikor megváltoztatom a dockerfile-t, és rebulid-elem az image-t, akkor csak azok a rétegek változnak
    →Emiatt a technológia miatt gyors, kicsi, könnyen kezelhető

Docker compose, orkesztráció – 11

Docker compose:
A compose egy olyan tool, amivel egy több konténert tartalmazó Docker alkalmazást hozhatunk létre vagy futtathatunk. A YAML fájllal konfigutálhatjuk az alkalmazás szolgáltatásait, majd egyetlen paranccsal létrehozni, illetve futtatni az összes szolgáltatást a konfigurációnkból.
Álatlában 3 fő lépésből áll:

1. Definiálni az applikáció környezetét egy Dockerfile-ban, tehát így mindenhol lehet majd reprodukálni
2. Definiálni a szolgáltatást, ami elkészíti az applikációta docker-compose.yml fájlban, így tudn majd több, izolált környezettel is együtt futni
3. Futtatni a `docker-compose` up-ot és a Compose elindul és futtatja az egész alkalmazást

Orkesztráció:
Menedzseli a konténerek élteciklusát, különösképpen nagy, dinamikus környezetkben. Számos feladat ellenőrzésére és automatizálására használják (Docker Swarm, Kubernetes):

- Konténerek biztosítása és telepítése
- Redundancia és elérhetőség
- A konténerek méretének növelése vagy eltávolítása az alkalmazás terhelésének egyenletes elosztása érdekében a host infrastruktúrában
- A konténerek mozgatása az egyik gépről a másikra, ha a host-ban nincs erőforrás, vagy ha a host meghal
- Erőforrások elosztása a konténerek között
- A konténerek közötti szolgáltatások felfedezése, terheléskiegyenlítése
- Alkalmazás konfigurálása a futó konténerekhez képest

Docker "workflow", Dockerfile + mi a különbség egy manuálisan létrehozott képfájl készítés és a Dockerfile használata között - - 12-13, 15-17

Workflow:

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

Dockerfile:
Egy olyan fájl, ami az image megépítse szükséges lépéseket definiálja. Egy olyan text fájl, amit a Docker fenntről lefelé olvas be. Egy csomó instrukciót tartalmaz, ami infromálja a Dockert, hogy HOGYAN kéne a Docker image-nak felépülnie (bulid).
(Mint pl sütéskor (Image=torta, Dockerfile=recept) A recept megmond minden hozzávalót, és leírja a lépéseket, ahhoz, hogy el tudjuk készíteni a tortát).
A docker image build-elve lesz, azáltal hogy futtatjuk a Docker parancsot (Dockerfile)

- FROM - egy már létező képfájlból indul ki az új képfájl; ez lesz az alap réteg
    - 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

Előnyök: könnyen újra-fordítható (Caching rendszer), egy fájlban meghatározható a build folyamat

Kubernetes

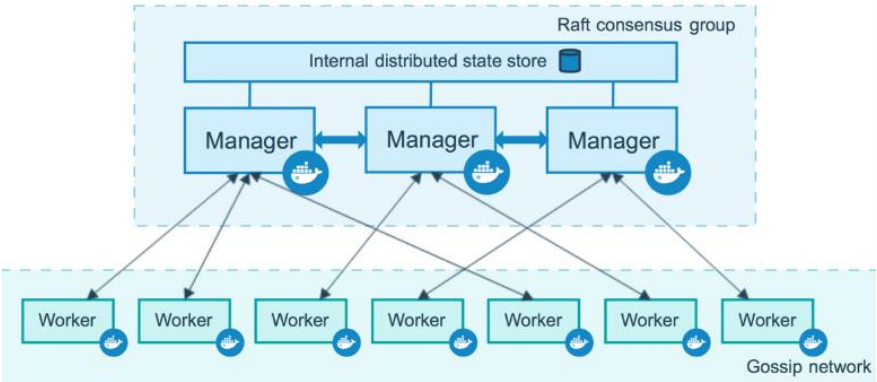
Konténer orkesztráció, motiváció - 2-3

Docker konténerek docker parancsokkal kezelhetőek az adott host gépen. Nehézkes hálózati kapcsolatok, multi-hosting
→ **Orchestration:** automatizált konténer telepítés és menedzsment multi-host környezetben (incl. skálázódás vezérlése)
Megoldások:

- Egyik megoldás: nyilvános felhőkben Docker
    - Amazon Web Services, Google Cloud, Microsoft Azure
- Másik megoldás: Docker + OpenStack
    - OpenStack Magnum
- Harmadik megoldás: Docker orkesztráció
    - Apache Mesos (2010)
    - Google Kubernetes (2014)
    - Docker Swarm Mode (2016)

Docker Swarm Mode + Hálózat - 6-7 + 8

Swarm: gépek összessége, amik Dockert futtatnak, és egy cluster-be kapcsolódnak. Ezután ugyanúgy futtatjuk a Dockert, csak a parancsok a **swarm manager** által lesznek elvégezve ezen a clusteren.
Cél: szolgáltatások (service) indítása ebben a clusterben.
Lehet fizikai vagy virtuális is a gép a swarm-ban,amikre a csatlakozás után, node-ként referálunk. Legalább egy master node van, és több worker node.



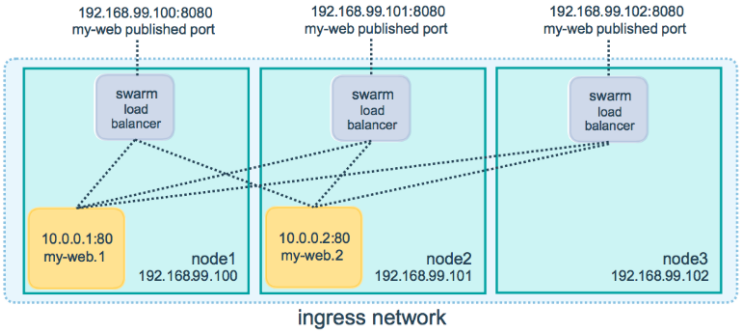
Swarm manager: minden parancs hozzá fut be, ő menedzseli azokat. Továbbítja a feladatokat a worker node-ok felé. ez engedélyezi más gépek csatlakozását is a swarm-hoz. Feladatai: cluster manager, API biztosítása, erőforrás ütemezés.

Node (manager, worker): task-ok futtatása. Manager egyúttal worker is lehet.
Minden worker node egy agentként fut, ami visszajelez a master node-nak a task állapotáról, tehát így a manager node nyomon tudja követni azokat.

Swarm mode: Docker engine (node) futtatási mód (Amenniyben az engine-k közös clusterbe vannak szervezve). Mikor létrehozunk egy service-t, akkor megadjuk annak optimális állapotát. Ha egy node elérhetetlenné válik, akkor a Docker átszervezi azt a task-ot (futó konténer) egy másik node-ra.

Task: egy futó konténer, ami része a swarm-nak és a swarm manager menedzseli. Végrehajtják a számukra kiadott feladatokat a service-ben. A manager node feladatokat ad a worker node-oknak, amik ezután nem mozgathatók át másik node-ra. Ha a feladat megbukik, akkor a manager készít egy új verziót, és azt adja át egy másik node számára.

Hálózatkezelés:
Routing mesh: Minden node fogadhat kapcsolatot a hirdetett porton, a swarm load balancer egy aktív konténerhez routol-ja a kérést.



Minden host-on kell futnia egy terheléselosztó funkciót (load balancer) is végző modulnak

- Swarm mode routing mesh része
- Ez juttatja el a kérést a megfelelő konténerhez
- Akkor is, ha az a konténer más host-on fut
- Akkor is, ha a kérést először fogadó host-on nem is fut olyan task

Kubernetes + Hálózat - 10-13 + 14

Alkalmazások elosztása konténerekbe, ezek ütemezése, skálázása, menedzselése, változások karbantartása. Ez stateless alkalmazásoknál jó, ami teljesen egyforma másolatokkal skálázható.

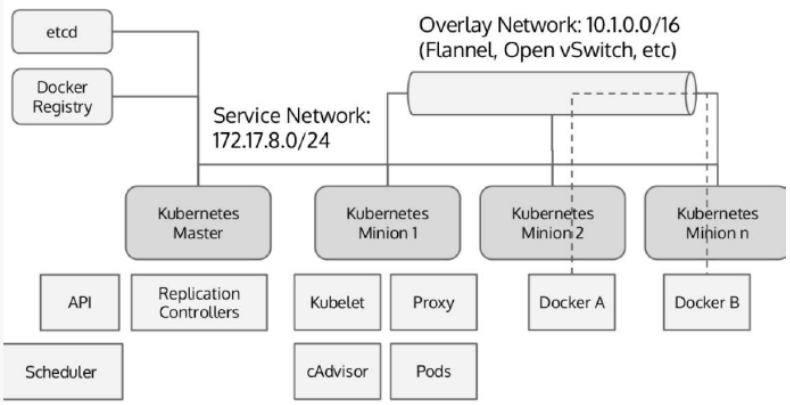
Cluster: Node-ok összessége, ahol legalább egy master node és több worker node van, amik lehetnek fizikaiak vagy virtuálisak is.

Node: A kért, hozzájuk rendelt feladatokat végzik el. A Kubernetes master vezérli őket.

Kubernetes master: Ő kezeli az applikációk ütemezését és deployement-jét a node-ok között. A master a node-okkal az API serveren kereszetül kommunikál. Az ütemező node-okat rendel a pod-okhoz(egy vagy több konténer), a megadott erőforrás-és policy korlátozásoktól függően.

Kubelet: minden node futtat egy agent folyamatot, ami a node állapotának menedzselésért felelős (start, stop...) A kubelet minden információját a Kubernetes API server-ről kapja.

Pods: Egy vagy több konténer, amely egyetlen node-ra van elhelyezve. A podban lévő összes konténer megoszt egy IP-címet, IPC-t, host nevet és egyéb erőforrásokat. Az absztrakt hálózat és a tároló elkülönül az alatta található konténerektől. Ezáltal könnyebben mozgatható a konténer a cluster körül. A Kubernetes mint egy csoport, úgy indít, leállít és replikál minden konténert a pod-ban. Megadható a kívánt konténer állapot a pod-ban egy YAML vagy JSON fájlban.



Labels: podokat azonosítja

Proxy: Load balancer a po d-oknak

Etcd: metadata sevice

cAdvisor: konténer felügyeű, ami erőforrás használatot és teljesítmény statisztikákat szolgál

Schelduler: Ütemezi a pod-okat worker node-okba

Hálózatkezelés:

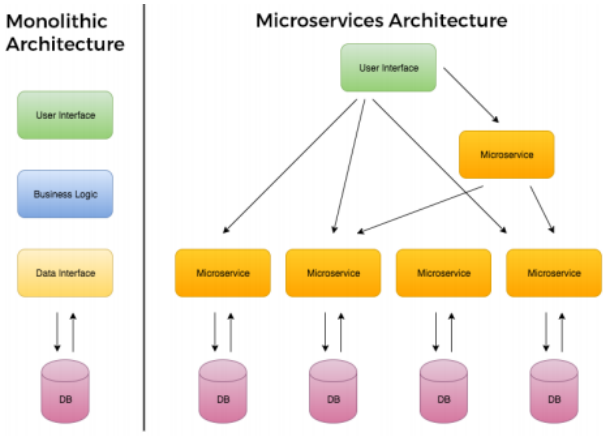
POD hatáskörében alkalmaz IP címeket. Egy PODon belül a konténerek megosztoznaka hálózati névtéren.

- Előny: localhost-on tudják egymást elérni
- Következmény: vigyázni kell a podon belüli konténerek portkiosztására (két konténer nem használhatja ugyanazt)
- Host-ok felé is van elvárás: NAT nélkül kell kommunikálni a konténerekkel

Serverless

Mi a mikroszervíz? - 2-3

Egy alkalmazást külön processzben futó, egymással valamilyen, viszonylag egyszerű mechanizmussal kommunikáló (ez legtöbbször egy HTTP API) kisebb szolgáltatások összességéből építünk fel.
(Konténer != microservice)
A komponensek mind teljesen különálló folyamatok, melyek egymással például web service kérésekkel vagy távoli függvényhívásokkal kommunikálnak. Ennek a megközelítésnek a legnagyobb előnye, hogy így a különböző komponensek egymástól függetlenül publikálhatóak, frissíthetőek. Ezzel szemben egy klasszikus alkalmazásban ha például frissítjük az egyik általunk használt libraryt, az magával vonja az egész alkalmazás frissítését.

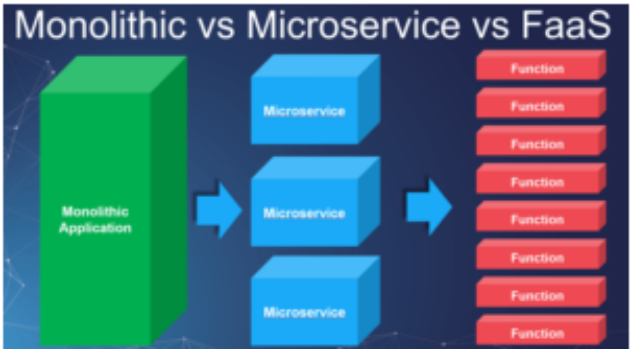


- Monolitikus: horizontálisan vannak elosztva a Prem/Cloud/Hybrid-eken
- Microservice: Vertiálisan és horizontálisan is el vannak osztva

A serverless model

A munka egység csak akkor fogyaszt erőforrást, amikor használja azt.
A serverless egy felhőalapú számítási execution modell, ahol a felhő szolgáltató dinamikusan kezeli a kiszolgálók elosztását és ellátását. A serverless alkalmazás esemény nélküli, számtalan konténerben fut, amelyek esemény-kiváltottak, rövid életűek (lehetnek egy hívásig), és amelyeket a felhő szolgáltató teljes mértékben kezel. Számlázás futásidő, tárhely és memória után történik, „virtuálisan határtalan”, automatikusan skálázódó tárhely és computing kapacitás.
Kód centrikus, skálázható, állapotmentes

Mi a "function"?



Function as a Service (FaaS): a fejlesztő számára lehetővé teszi, hogy futtassa az elkészült kódot, egyből válaszolva az eseményre bármilyen komplex infrasturktúra megépítése nélkül. Ez azt jelenti, hogy simán csak fel lehet tölteni a részletet a felhőbe, ahol egyből lehet is futtatni. Ahelyett hogy skálázni kellene a monolitikus modellben, most csak felosztjuk a szervert több function-ra, amik azonnal és automatikusan skálázhatóak.

Előnyök:

- Kevesebb fejlesztői ligisztika – szerver infrastruktúra menedzselése valaki más feladata
- Több idő jut kódolásra
- Vele járó a skálázhatóság
- Soha nem kell idle erőforrásokért fizeti
- Beépíthető és hibatűrő
- Kis, kezelhető egységek

Hátrányok:

- Csökkenti az átláthatósgot (más kezeli az infrastruktúrát, nehéz egyben megérteni az egész rendszert)
- Nehéz debug-golni
- Auto-scaling gyakran költség skálázást is jelent, így nehezebbé teszi az üzleti költségvetés felmérését
- Nehéz követni, a sok, szeparált részt