HAKAP Vizsga

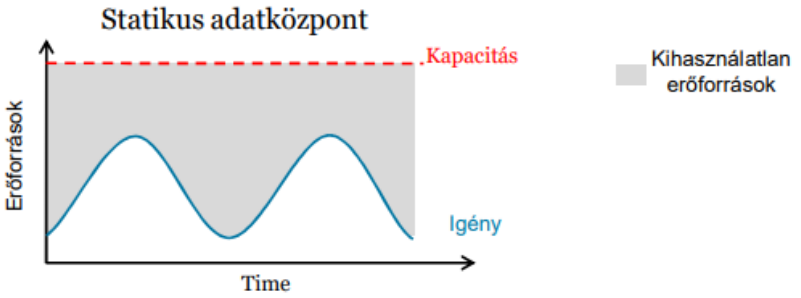
Hálózati erőforrásmegosztás

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

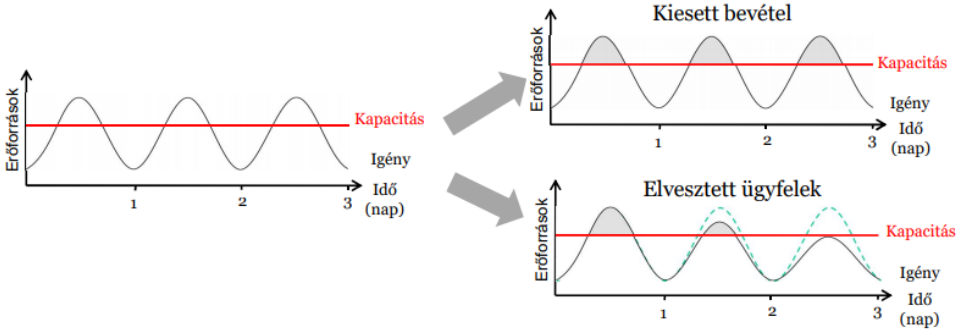
- **„Shared resource”:** Számítógépek közösen használják az erőforrást (programok, fájlok, nyomtatók, Intranet, LAN)
  - Topológiák:
  - Workgroup (P2P)
  - Kliens szerver
- Korai LAN-nál több alternatíva:
  - Apple Filing Protocol / AppleTalk (Apple)
  - SMB / TCP/IP (Microsoft)
  - Network File System (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



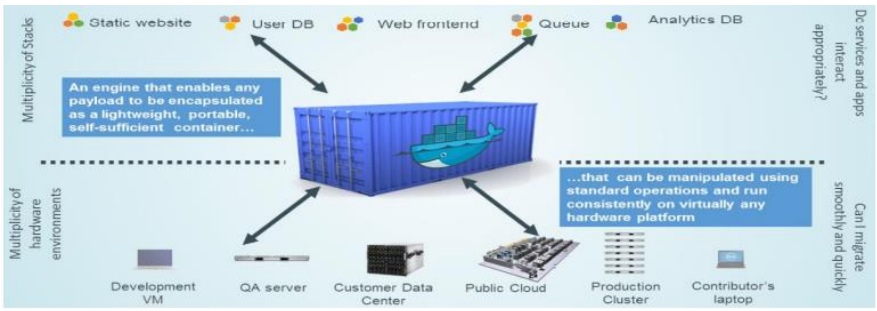
- Alultervezés hátrányai



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.

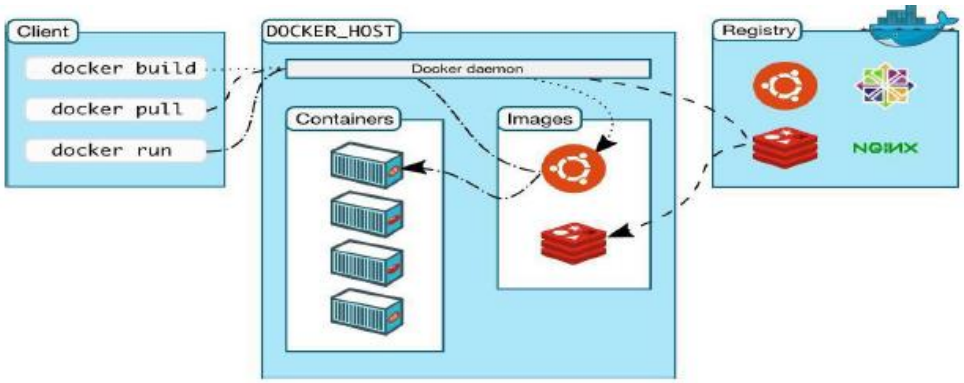
- **Libvirt LXC:** userspace container management tool
    - libvirt driver-ként megvalósítva
    - Konténer menedzsment
    - Névtér létrehozás
    - Privát fájlrendszer kezelése a konténeren belül
    - Konténer eszközeinek létrehozása
    - Cgroup által vezérelt erőforrások

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.

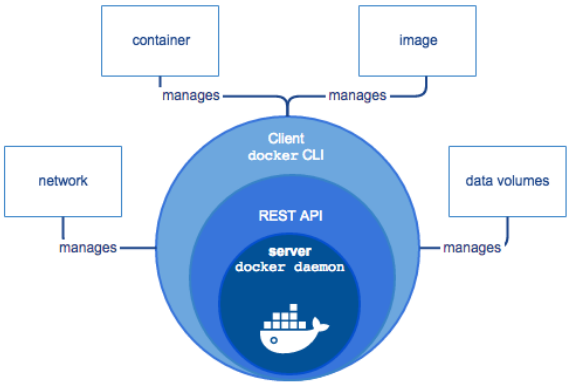


- **Képfájl (image)** = egy VM-nek megfelelő fájl együttes, amely tartalmaz minden olyan kiegészítést (lib, db, config, stb), ami szükséges az igényelt alkalmazás futtatásához
- **Konténter (container)** = egy Docker image futtatott példánya
- **Tárház (registry)** = képfájlok tára
  - Alapesetben helyi (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 amelyekből 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.

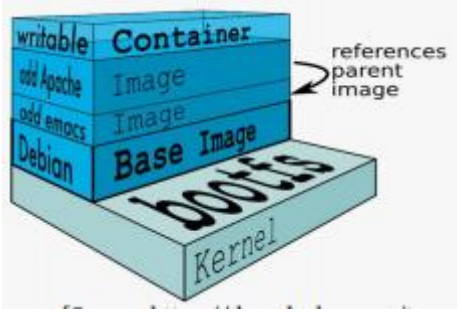
- **Docker** run: letölt MINDENT és futtatja a konténerünket
- **Pull:** lehúzza az image-t a docker registry-ből, és menti a saját rendszerünkbe
- **Build:** saját image létrhozása

} 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

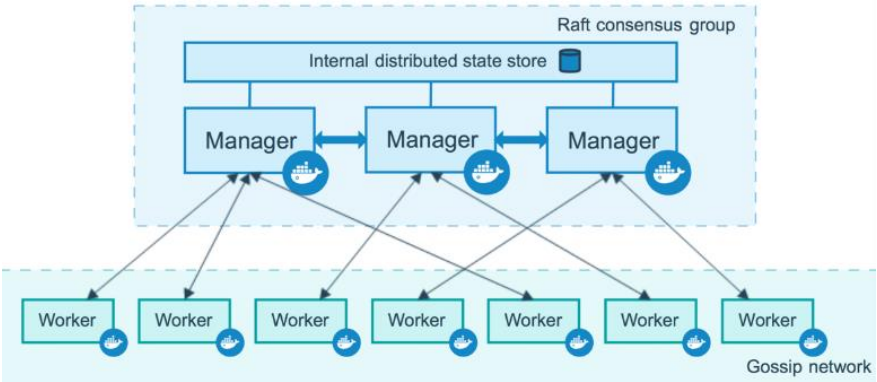
Konténer orkesztráció, motiváció - 2-3

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.



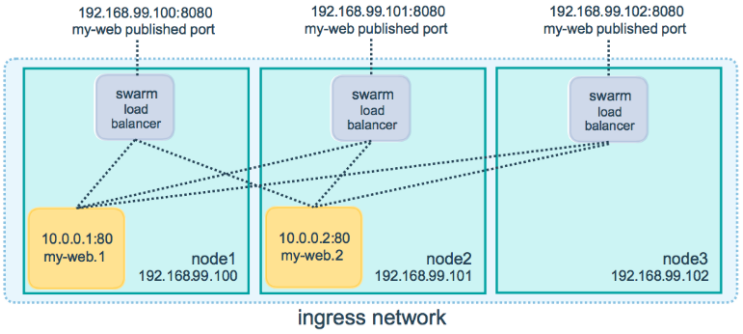
Swarm manager: minden parancs hozzá fut be, ő menedzseli azokat. Továbbítja a feladatokat a worker node-ok felé. ez engedélyezi más gépek csatlakozását is a swarm-hoz. Feladatai: cluster manager, API biztosítása, erőforrás ütemezés.

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.

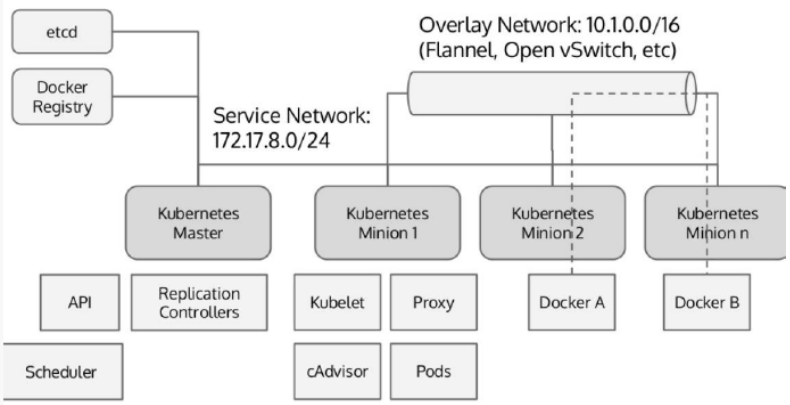
Cluster: Node-ok összessége, ahol legalább egy master node és több worker node van, amik lehetnek fizikaiak vagy virtuálisak is.

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:

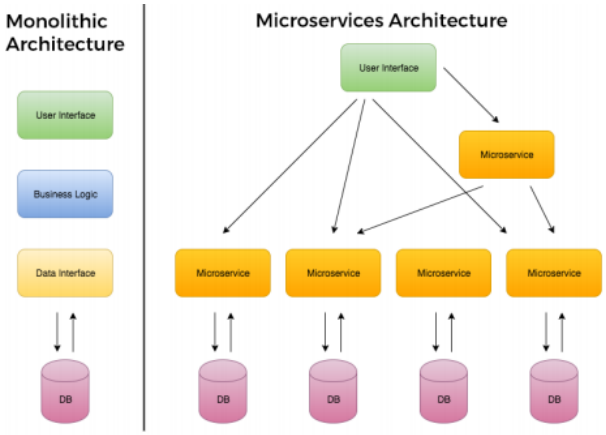
POD hatáskörében alkalmaz IP címeket. Egy PODon belül a konténerek megosztoznaka hálózati névtéren.

- 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.

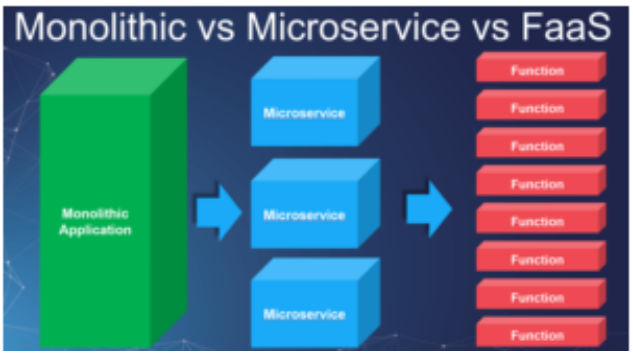


- Monolitikus: horizontálisan vannak elosztva a Prem/Cloud/Hybrid-eken
- Microservice: Vertiálisan és horizontálisan is el vannak osztva

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

ZH

Virtuális hálózatkezelés alapok

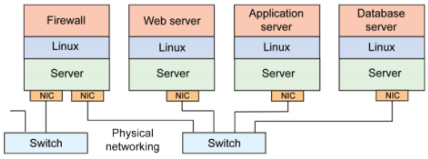
Virtualizálás – 3-4

Virtuális gép (Virtual Machine – VM)

- CPU, memória, háttértár és ezen felül a hálózat virtualizálás kulcsfontosságú!
- több, különböző operációs rendszer ugyanazon a hardveren
- hibák izolálása: egy VM hibája nem teszi tönkre a többit
- operációs rendszer szintű állapot mentés/visszatöltés
- szerver terhelés (workload) optimalizálás
- felhő infrastruktúra szolgáltatás

Virtualizálás

- CPU időbeli megosztása a vendég rendszerek között
- Memória „térbeli” megosztása a vendég rendszerek között
- Lemez, hálózati és egyéb eszközök szimulálása



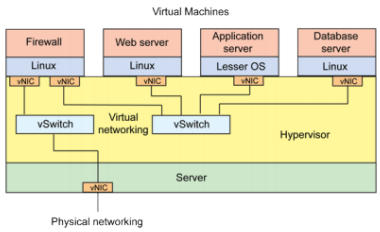
Hálózati virtualizálás – 5-6

Hálózatok: fizikai – virtulális

- NIC: Network Interface Card
- vNIC: virtual Network Interface Card

Virtuális kapcsoló

- a lokális kommunikáció sebessége a memóriaátvitel sebességétől függ
- Implementációk: Linuc bridge, Open Virtual Switch (OVS)



Hálózati kártya virtualizáció

- **QEMU: Quick Emulator**
  - szoftveres platform emulátor, szoftveres NIC emuláció
- **KVM: Kernel-based VM**
  - hardveres gyorsítás
  - egy virtualizációs infrastruktúra a Linux rendszermagba integrálva.
  - Egy Hypervisor váltja a Linux rendszermagot.
  - A KVM natív virtualizációt támogat.
- **virtIO**
  - fő platform az IO virtualizálásának a KVM-ben
  - fő célja, hogy egy közös framework legyen a hypervisior-nak és a virtualizált IO-nak is
  - input/output para-virtualizáció
    - módosított vendég rendszer
    - egyszerűbb és gyorsabb
    - blokk, általános PCI és hálózati eszközök számára
- **TAP: (Test Access Point)**
  - Ethernet szintű (L2) virtuális hálózati meghajtó
- **TUN: tunnel**
  - IP szintű (L3) virtuális hálózati meghajtó

Linux virtualizáció

Hypervisor: QEMU/KVM, Xen, stb..

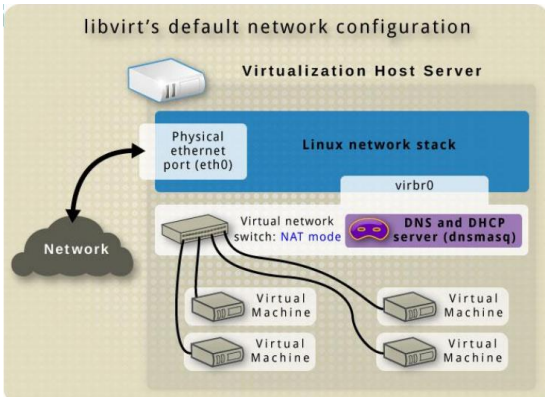
Libvirt: virtualizált API és egy toolkit, ami arra szolgál, hogy menedzselje a virtalizációs host-ot

Libvirt GUI: virt-manager

Libvirt hálózat – 9 -10 -11

Libvirt hálózat: NAT (Network Address Translation)

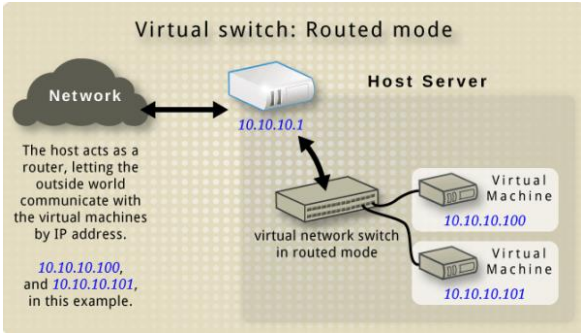
***NAT**: A NAT (hálózati címfordítás vagy hálózati címfordító) az internetprotokoll (IP) címek virtualizációja. A NAT javítja a biztonságot és csökkenti az IP-címek számát a szervezet igényeinek megfelelően. A NAT átjárók két hálózat, a belső hálózat és a külső hálózat között ülnek .*



- virsh net-list --all
  - név: default
- default virtual network: NAT
  - virbr0 – Virtual Bridge 0
    - NAT-hoz, címfordításhoz használják ezt az interfészt
    - Ezt a libvirt könyvtár biztosítja, és a virtuális környezetek néha a külső hálózathoz való csatlakozáshoz használják.
    - nincs hozzá csatolva fizikai interfész, mivel NAT + forwarding segítségével kapcsolódik a külvilághoz
    - ip forwarding engedélyezve
    - iptables szabályok: ki/be forgalom engedélyezése a virbr0–hoz kapcsolódó VM-ek számára, (INPUT, FORWARD, OUTPUT és POSTROUTING láncok)
  - DHCP (dnsmasq program)
  - külső hozzáférés port továbbítással beállítható
  - név alapján csatlakoztatható hozzá VM
- különbségek a VirtualBox NAT módhoz képest
  - kimenő forgalom engedélyezett, VM kívülről elérhetetlen, **hálózaton belül a VM-ek illetve a hoszt kommunikálhatnak**
  - kb. NAT hálózat + Host only vegyítése

További Libvirt hálózati módok:

- **Bridge-elt**
  - full bridging, a vendég rdsz. közvetlenül a LAN-hoz kapcsolódik
  - shared physical device: a hoszt fizikai interfésze csatolva a virtuális bridge-hez
  - megj.: vezeték-nélküli interfész nem csatolható hoszt bridge-hez, csak Ethernet
  - bridge név alapján csatlakoztatható hozzá VM
- **Routed**
  - ha bridge-elt nem kivitelezhető
  - statikus útvonalbeállítás, nincs NAT
  - hálózat név alapján csatlakoztatható hozzá VM
- **Izolált**
  - VM-ek egymással és a hoszttal kommunikálhatnak, de a külvilággal nem



Map Reduce

MapReduce – 15-18 A Big Data technológia születése

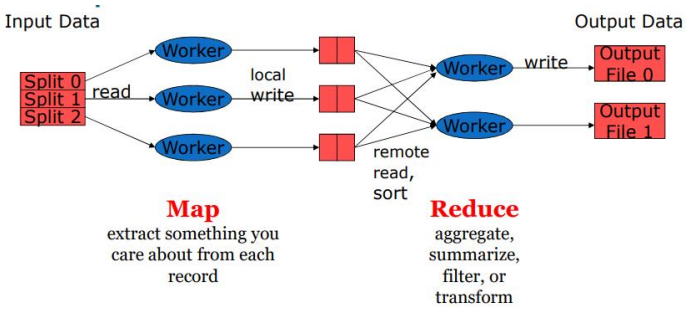
A MapReduce egy programozási modell nagy adathalmazok feldolgozására párhuzamosan és egy klaszteren elosztottan. A MapReduce tartalmaz egy map funkciót, amely szűrést és rendezést végez, valamint egy reduce funkciót, amely összegzi az eredményt. A MapReduce rendszer osztja el a feladatokat a szervereken párhuzamosan futtatva azokat, irányítva minden adatátvitelt, egyúttal hibatűrést is biztosít redundancián keresztül.

MapReduce biztosít számunkra:

- Automatikus parallelizációt, elosztást
- I/O ütemezésú
  - Teherelosztás
  - Hálózat és adat szállítási optimalizálás
- Hibatűrés: gép hibák kezelése
- Több erő: KIskálázás, nem FELskálázás
  - Nem nagyszámú „nyersanyag” (Commodity server) szerverek, hanem magas minőségű speciális szerverek!
- Apache Hadoop: MapReduce nyílt forráskódú megvalósítása

Tipikus problémák, amiket megoldott a MapReduce:

- Rengeteg adat olvasásáa
- **Map**: kiszűr mindent ami éppen érdekel minden rekordból
- Véletlen sorrend és rendezés
- **Reduce**: összesítés, összegzés, szűrés és átalakítás
- Eredmények kiírása



Mappers és Reducers

- Ha több adatot kell kezelni, hozzá lehet adni több Map-et és Reduce-t (Mappers, Reducers)
- Nem kell foglalkozni multiszálas kóddal
  - Mappers és Reducers tipikusan egy szálasak és determinisztikusak.
    - A determinisztikusság lehetővé teszi, hogy újra indítsunk egy hibás munkát (restarting of failed jobs)
    - Mappers and Reducers teljesen függetlenül futnak egymástól (Hadoop-ban külön, szeparált JVM-ekben futnak)

Failures in MapReduce, Summary – 26 -27, 30

Worker meghibásodás:

- Hibák észlelése időszakos „szívverésekkel”
- Újra végrehajtás a folyamatban lévő Map/Reduce feladatoknak

Master meghibásodás:

- Egyetlen meghibásodás → folytatás a végrehajtási naplóból (Execution Log)

Robust

Google tapasztalata: 1800 gépből 1600-at veszített egyszerre, de még így is befejezte rendesen

Hibatűrés: Újra elvégzéssel van kezelve

Ha kész a task, azt a master véglegesíti

Tisztítás: ha egy recordban az egyik worker 2 hibát is talál, szól a következő worker-nek, hogy ugorja át azt a rekordot.

GFS – 32-37

Google File System, Google által kifejlesztett szabadalmaztatott fájlrendszer , amely hatékony és megbízható hozzáférést biztosít az adatokhoz nagy commodity hardwer clusterek használatával

Motiváció: Nagy méretű adat tárolása

- Sok adat manipulálása
- Nagy számú commodity hardware
- Komponens meghibásodása a norma
- CÉL: skálázható, nagy teljesítőképességű, hibatűrő elosztott fájl rendszer

Miért kell új fájl rendszer?

- Egyik sem hibamodellre lett tervezve
- Kevés olyan skálázás ami magasan vagy dinamikusan és könnyen skálázik
- Speciális primitívek hiánya a nagy elosztott számításokhoz

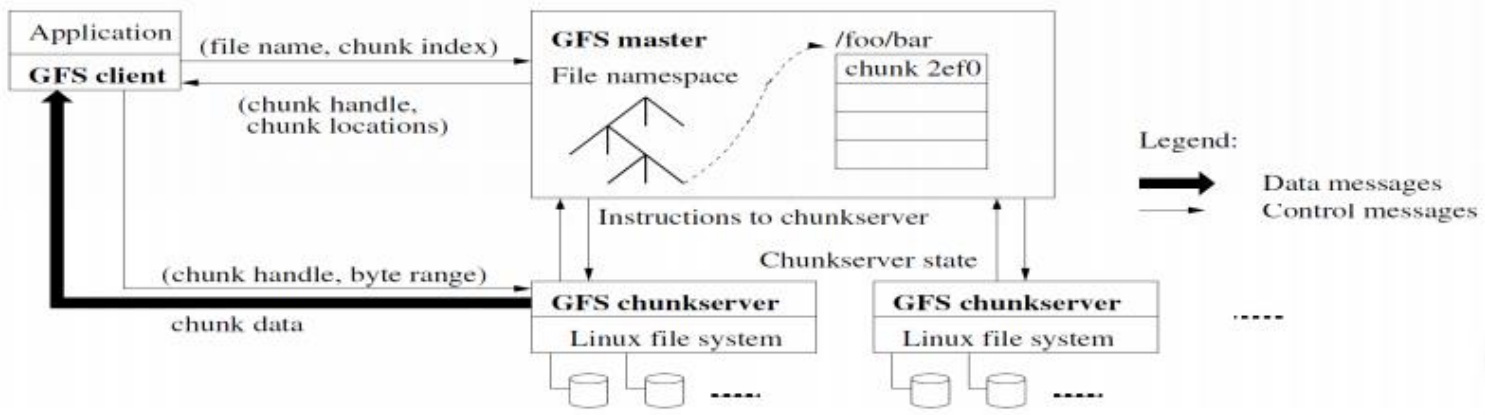
Mit kéne várnunk a GFS-től?

- Google applikációhoz lett tervezve
  - Kontrol mind a fájl rendszer, mind az applikáció felett
  - Az applikációk néhány specifikus ellérési mintát használnak
    - Hozzácsatolva egy nagy fájlhoz
    - Nagy streaming olvasás
  - Nem alkalmas:
    - Alacsony késleltetésű adat eléréshez
    - sok kicsi fájlhoz, többszörös íráshoz, önkényes fájlmódosításokhoz
- Nem POSIX, bár többnyire hagyományos
  - Specifikus műveletek: RecordAppend
- Különböző jellemvonások, mint a tranzakciós vagy a „costumer order” adat: Write Once Ready Many (WORM) – egyszer olvasható, többször írható adat
  - Pl.: web logok, web bejáró adatai, vagy egészségügyi és páciens információk
  - MapReduce progrmozási modellt inspirálta
- Ezeket a jellemzőket a Google hasznosította a GFS-ben
  - Apache Hadoop: nyílt forráskódú projekt (HDFS: Hadoop Distributed Files System)

Komponensek

- **Master(NameNodeI)**
  - Metaadatok menedzselése (namespace)
  - Nem vesz részt az adatátvitelben
  - Kontrolálja a kiosztást, elhelyezést , replikációkat
- **Chnunkserver (DataNode)**
  - Adat csonkokat/darabokat tárol
  - Nincs ismerete a GFS felépítéséről
  - Helyi Linux fájlrendszerre épült

GFS Architektúra



GFS Fault Tolerance, Replica mgmt – 44-47

Hibatűrés

- **Replikáció**:
  - Magas elérés az olvasatoknak (reads)
  - Felhasználó által kontrollálható, alapból 3 (non-RAID)

    - Olvasási/keresési sávszélességet biztosít
    - A master feladata a re-replikáció irányítása, ha az adatcsomópont (DataNode) meghal
- Online ellenőrző összegzés az adatcsomópontokon)DataNode)

Replica management (cloning)

- A chnunk repilika elveszik vagy sérült
- Cél: minimalizálni az applikáció megszakadását és az adatvesztét
    - Kb. prioritás sorrendben:
        - Több replika hiány → prioritás boost
        - Törölt fájl → prioritás csökkentése
        - Kliens blokkolása írásban → nagy prioritás boost
    - Master irányítja az adatok másolását
- Termelési klaszter teljesítménye:
    - Egyszeri hiba, teljes visszaállítás (600GB): 23,2perc
    - Dupla hiba, visszaállítva 2 replikáció: 2perc

Garbage Collection

A Mastern-nek nem kell teljesen tudnia, mit tárol minden egyes adatcsomóponton (DataNode)

- Rendszeresen vizsgálja a namepsace-eket
- GC (GarbageColletion) után, a törölt fájlok kikerülnek a névtérből
- Az adatcsomópont rendszeresen lekérdezi a Master-t, minden egyes chunk-ról, amit tud
- Ha egy darabot elfelejtenek, a Master szól az adatcsomópontnak, hogy törölje azt

Korlátozások

- Master a hiba központ
- Master a skálázhatóság szűk keresztmetszete lehet
- Késleltetés, mikor több ezer fájlt vizsgálunk vagy megnyitunk
- Biztonsági modell gyengesége

Konklúzió

- Az olcsó commodity komponensek lehetnek az alapjai a nagyméretű, megbízható rendszernek
- Szabályozza az API-t (Pl.: ReduceAppend nagy, elosztott applikációkat engedélyezhet)
- Hibatűrés
- Hasznos sok más hasonló apphoz

Google MapReduce → HADOOP

Hadoop – 3-4, 8-9, 11-12

Az Apache Hadoop egy nyílt forráskódú keretrendszer, amely adat-intenzív elosztott alkalmazásokat támogat. Nagy mennyiségű alacsony költségű, általánosan elérhető hardverből épített cluster építését teszi lehetővé. A Hadoop a Google MapReduce és a Google File System leírásaiból készült.

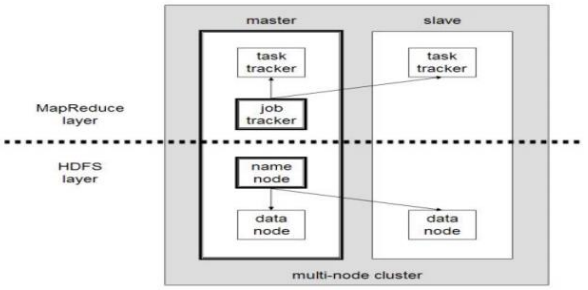
- Apache top-level projekt, nyílt forráskódú keretrendszer a megbízható, skálázható, elosztott számításoknak és adattárolásnak
- Java alapú keretrendszer az eszközöknek a tároláshoz és adatok nagyméretű feldolgozása a clustereken
- Ez egy rugalmas és rendkívül elérhető architektúra a nagyszabású számításokhoz és az adatfeldolgozáshoz a commodity hardver hálózatán
- Apache v2 licenc

Célok/Követelmények

- Összefoglalni és egyszerűsíteni a nagy és/vagy gyorsan növekvő adatok tárolását és feldolgozását
  - Rendezett és rendezetlen adat
  - Egyszerű programozási modellek
- Magas skálázhatóság és elérhetőség
- Commodity (olcsó!!) hardver használata kis redundanciával
- Hibatűrés
- Számítások áthelyezése az adatok helyett

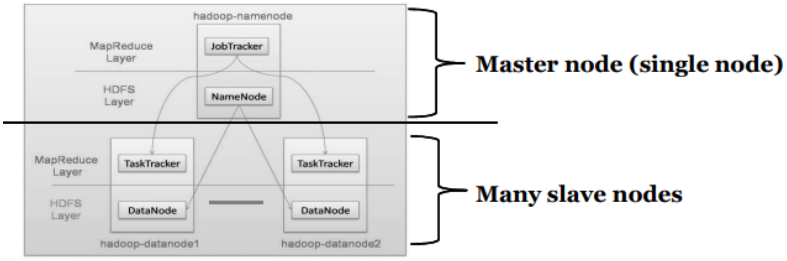
Keretrendszer - rétegek

Két fő réteg:



- Execute engine (MapReduce)
  - **JobTracker**
    - Felosztja az adatokat kisebb taskokra (“map”) és elküldi a TaskTracker folyamatnak minden node-ban
  - **TaskTracker**
    - Jelent a JobTracker-nek és jelentéseket készít a munkafolyamatról, elküldi az adatokat („Reduce”) vagy új munkát kér
- Elosztott fájl rendszer (HDFS - Hadoop Distribution File System)

Hadoop Master /Slave Architektúra



MapReduce korlátai

- **Skálázhatóság**
  - Max cluster méret : 400 nodes
  - Max egyidejű task: 40 000
  - Nyers,durva szinkronizálás a JobTracker-ben
- Egyetlen meghibásodási pont (Single Point of Failure – SPOF)
  - Egyetlen hiba megöli az összes éppen futó és várakozó feladatokat
  - A felhasználóknak újra elő kell állítani minden feladatot
- Az újraindítás nagyon bonyolult a komplex állapot miatt

HDFS – 14-18, 21

Hadoop saját fájlrendszere

Commodity hardvereken futó elosztott fájlrendszer. Sok hasonlóság van a már meglévő elosztott fájlrendszerekkel, de az eltérések nagyon jellegzetesek.

- Rendkívül hibatűrő és alacsony költségű hardveren való alkalmazásra tervezték.
- Nagy áteresztőképességű hozzáférést biztosít az alkalmazásadatokhoz, és alkalmas nagy adatállományú alkalmazásokhoz.
- Lazít néhány POSIX követelményen, hogy engedélyezze a streaming hozzáférést a fájlrendszer adataihoz.
- Az Apache Hadoop Core projekt része

MapReduce with HDFS

- Elosztott, néhány központosítással
- Ahol a legtöbb számítási kapacitás és rendszer tárolója található, ott futnak a fő csomópontok a clusternek
- A fő csomópontok futtatnak TaskTracker-eket, hogy elfogadják és válaszoljanak a MapReduce feladatokra, és a DataNode is tárolja a szükséges blokkokat olyan közel, amilyen közel csak tudja
- Központi vezérlő csomópont futtat NameNode-okat, ahhoz hogy figyelemmel kísérje a HDFS könyvtárakat és fájlokat, és a JobTracker elküldi a számítási feladatokat a TaskTrackernek
- Javában írt, támogatja a Paython-t és Ruby-t is

Tulajdonságai

- MapReduce igényeihez igazítva
- Lehetséges sok olvasás a célja
- Írás sokkal költségesebb
- Nagyfokú adat replikáció (alapból 3x)
- Nincs szükség RAID-re a normál node-okon
- Nagy blokkméret (64MB)
- DataNodes helyének ismerete a hálózaton

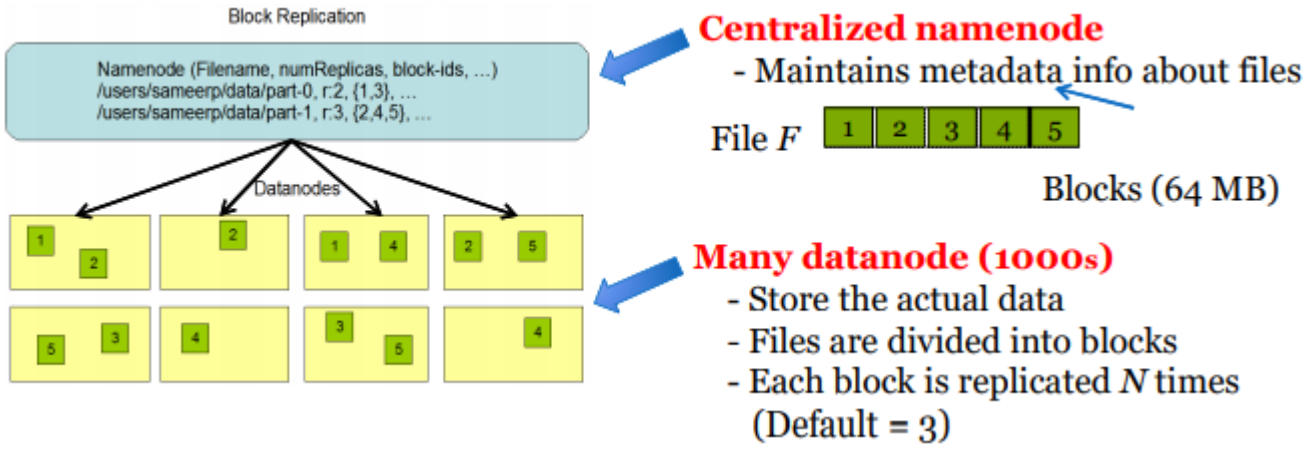
NameNode

- Fájlok metaadat tárolása, mint a pl tipikus FileSystem struktúra
- A szerver tartja a NameNode-ot, ami nagyon fontos mivel csak egy van belőle
- Tranzakciós naplófájl törléséhez, hozzáadásához, stb. Nem használ tranzakciót teljes blokkok vagy file-streams-hez, csak metaadatokhoz.
- DataNode meghibásodása után, szükség esetén több másolati blokk létrehozását kezeli

DataNode

- Aktuális adatokat tárolja a HDFS-ben
- Bármilyen underlaying fájlrendszeren képes futni (ext3/4, NTFS, stb…)
- Értesíti a NameNode-ot, hogy milyen blokkok vannak nála
- Replikálja a blokkokat 2x a lokális rack-be, 1x valahova máshova

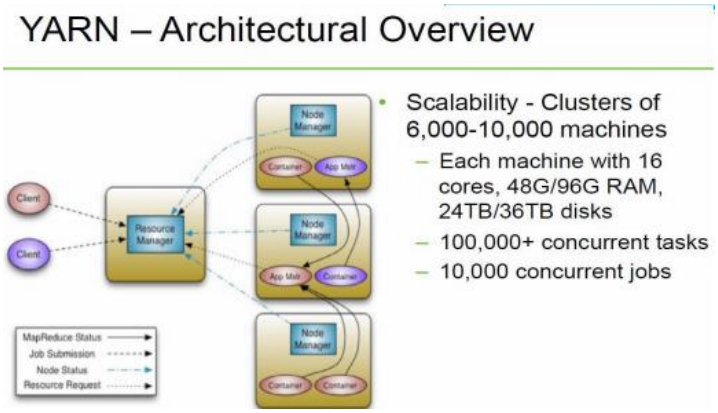
Adatok tárolása:



YARN - 29-32

YARN

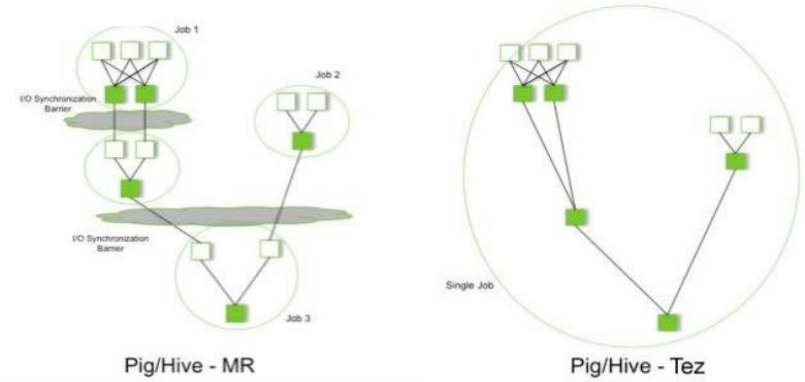
- Yet Another Resource Negotiator („Még egy másik erőforrás közvetítő”)
- YARN Another Resource Negotiator (rekurzív betűszó)
- Megoldja a „klasszikus” MapReduce skálázhatósági hiányosságait
- Többnyire általános célú keretrendszer, melynek egyik alkalmazása a klasszikus MapReduce.
- Felosztja a JobTracker / TaskTracker két fő feladatát különálló egységekké
- **JobTracker**
  - globális erőforrás-kezelő – Klaszter erőforrás-kezelés
  - alkalmazásonként Apllication Master – feladatok ütemezése és felügyelete, az ütemezőtől származó erőforrás konténerek közvetítése, állapotuk és haladásuk nyomon követése
- **TaskTracker**
  - új csomópontonkénti slave Node Manager (NM), amely felelős az alkalmazások konténereinek elindításáért, az erőforrás-felhasználás (cpu, memória, lemez, hálózat) felügyeletéért és az erőforrás-kezelőnek történő jelentésekért
  - (alkalmazásonként egy NodeManager-en futó konténer)
- A YARN fenntartja a kompatibilitást a meglévő MapReduce alkalmazásokkal és felhasználókkal



Criteria	YARN	MapReduce
Type of processing	Real-time, batch, interactive processing with multiple engines	Silo & batch processing with single engine
Cluster resource optimization	Excellent due to central resource management	Average due to fixed Map & Reduce slots
Suitable for	MapReduce & Non – MapReduce applications	Only MapReduce applications
Managing cluster resource	Done by YARN	Done by JobTracker

TEZ – 38-40

- Gyors, azonnali mobilfizetés (indiai felhasználói célközönséggel)
- 2018-ban átnevezték Google Pay-re
- Általános célú, rendkívül testreszabható keretrendszer, amely egyszerűsíti az adatfeldolgozási feladatokat mind a kis léptékű (alacsony késleltetés), mind a nagy léptékű (nagy teljesítményű) munkaterheléseknél a Hadoop-ban.
- Általánosítja a MapReduce paradigmát egy hatékonyabb keretrendszerre, biztosítva a komplex DAG (körmentes) végrehajtásának képességét
- Lehetővé teszi az Apache Hive, az Apache Pig és a Cascading alkalmazását az emberi interaktív válaszidők és az extrém petabyte-skálájú áteresztőképesség követelményeihez



- Az eredeti MapReduce minden folyamat után lemez I / O-t igényel
- Egy sor MapReduce munka egymás után következik be, ami sok I / O-t eredményezne
- Tez megszünteti ezeket a közbenső lépéseket, növeli a sebességet és csökkenti az erőforrás-felhasználást

MapReduce-szal szembeni teljesítménynövekedés

- Eltávolítja a replikált írási akadályt az egymást követő számítások között
- Megszünteti a folyamat indítást a munkafolyamatok fölött
- Minden munkafolyamat-feladatban megszünteti a map olvasás extra szakaszát
- Eltávolítja a munkafolyamat által elszenvedett várakozási és erőforrás-vitát, amely az előző/előzetes munka befejezése után kezdődik

Spark – 44, 46-49, 51

- Az Apache Alapítvány nyílt forráskódú projektje. Nem egy termék.
- Önálló, általános BigData számítási keretrendszer
    - Mind batch, mind streaming mód
- A memóriában lévő compute engine, amely az adatokkal működik → Nem adattároló!
- Lehetővé teszi a nagy mennyiségű skálázott adatok magas ismétlődő elemzését
- Egységes környezet az adatelemzők, fejlesztők és mérnökök számára
- Radikálisan leegyszerűsíti az adatokkal táplált intelligens alkalmazások fejlesztését
- A Hadoop-al kombinálható
    - De Hadoop nélkül is működhet: például Kubernetes

Kulcsfontosságú érvek a Spark mellett

Nagy teljesítmény

- A memóriában lévő architektúra nagymértékben csökkenti a lemez I/O-t
- Bárhol 20-100x gyorsabb a gyakori feladatoknál

Produktív

- A tömör és kifejező szintaxis, különösen az előző megközelítésekhez képest (5x kevesebb kód)
- Egységes programozási modell az egyes felhasználási esetekben és (lépések) az adatok életciklusában
- Integrált gyakori programozási nyelvekkel - Java, Python, Scala
- Az új eszközök folyamatosan csökkentik a hozzáférés képességi akadályokat (például SQL az elemzők számára)

Használja a már meglévő technológiákat

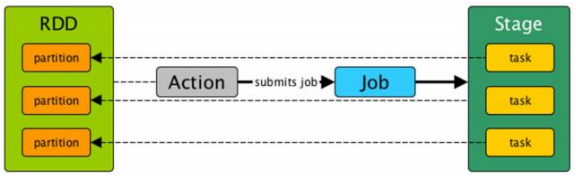
- Jól működik a meglévő Hadoop ecosystem-ben

Idő előre haladásával javul

- A közreműködők nagy és növekvő közössége folyamatosan javítja a teljes elemzési stack-et és bővíti a képességeket

Komponensei

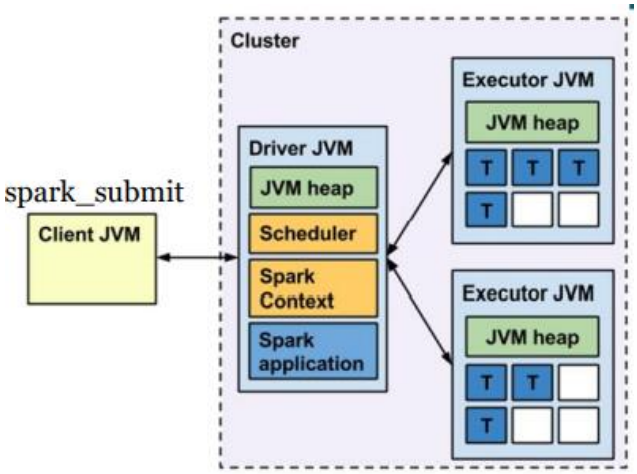
- **Resilient Distributed Dataset (RDD)**
    - Az elsődleges adatabsztrakció és a Spark magja
    - Rugalmas és elosztott rekordgyűjtemény számos partíción
    - Keverés: az adatok újraosztása a partíciókon
- **Stage**
    - A végrehajtás fizikai egysége



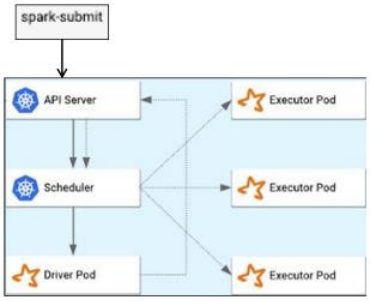
Terminológia

- **Driver:** Folyamat, amely a SparkContext-et tartalmazza
- **Executor:** Folyamat, amely egy vagy több Spark feladatot hajt végre
- **Master**: Folyamat, amely az alkalmazásokat kezeli a klaszteren
- **Worker**: Folyamat, amely egy adott csomópont végrehajtóit kezeli

Klaszter



A Kubernetes az új YARN (a Spark számára)



- Amikor a Spark a YARN-nál van telepítve, a Spark a YARN-t konténerkezelő rendszerként kezeli
- Meghatározott konténereket követel a Spark a YARN-tól származó konténerektől
- Miután megszerzi a konténereket, RPC alapú kommunikációt épít a konténerek között a vezető (driver) és a végrehajtók (executors) futtatásához
- A Spark automatikusan skálázható a konténerek felszabadításával/nyilvánosságra hozásával és megszerzésével
- A Spark 2.3-tól a Spark támogatja a kubernetes-t, mint új klaszter hátteret
- Ez kiegészíti a meglévő listáját a YARN, a Mesos és a standalone backend-nek
- Ez egy natív integráció, nincs szükség a statikus klaszterre ami használat előtt meg van már építve

P2P – Peer-to-Peer

P2P meghatározása 3-6

- egy alkalmazáscsoport mely kihasználja az Internet peremén levő felhasználók erőforrásait:
  - Tárolás – merevlemez kapacitás
  - CPU – számítási kapacitás
  - Tartalom – adatok, informaciók megosztása
  - Bármilyen más megosztható erőforrás, szolgáltatás, funkció
- Egy alkalmazás rétegbeli Internet a fizikai Internet topológia fölött
- Peer-to-peer network” = egyenrangú hálózat
- „Peer” = veled egyenrangú felhasználó
- „Kommunista” rendszer – mindenki egyenlő
- A kliens-szerver architektúra ellentéte

Jellemzők

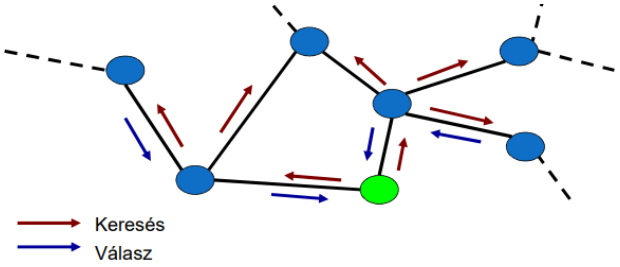
- Minden résztvevő peer egyszerre kliens és szerver
- Nincs központi vezérlés
- Nincs központi adatbázis
- Senkinek nincs globális képe a hálózatról
- A rendszer globális működése a lokális kölcsönhatások eredménye
- Bármilyen megosztott erőforrás elérhető bárki által
- Akkor férhetsz mások erőforrásaihoz, ha megosztod a sajátaidat
- A peer-ek függetlenek egymástól
- A peer-ek és a kapcsolatok alapvetően megbízhatatlanok → Gyakori be- és kilépés

DC (DirectConnect) – 40

- NMDC - NeoModus DirectConnect - visszafejtett szöveges protokoll
- Központosított rendszer
  - Több száz hub
  - Hub ≠ szerver
    - Nem tárolja a fájlok listáját
    - Összeköti a peer-eket, továbbítja a kereséseket
- Korlátozott belépés
  - Megosztott tartalom mérete (több Gb)
  - IP címtartomány
  - Hozzáférési sebesség
- Nincs globális azonosítás
  - Minden hub-on más-más azonosítót használhat a peer
- Kliensek aktív és passzív módban
  - Aktív módban bárkitől tölthet
  - Passzív módban csak az aktív peer-ektől

Gnutella – 47-49, 51-54

- Elosztott rendszerm központi szerver nélkül
- Elárasztás alapú keresés
- Minden peer
  - Megoszt állományokat
  - Kliens és szerver egyidőben – servent
  - Továbbítja a szomszédai felé a kapott Query csomagokat
  - Válaszol a Query üzenetekre, ha rendelkezésére áll a keresett fájl



Előnyök

- Robusztusság, nincs szűk keresztmetszet
- Egyszerűség
- Jogilag nehezen támadható: Nincs perelhető központi entitás

Hátranyok

- Az elárasztás nem skálázható megoldás
  - TTL-t használva (valamilyen szinten) áthidalható
  - Nem minden szomszédnak küldjük tovább az üzeneteket
  - A Message ID alapján, egy üzenetet csak egyszer továbbít egy peer
    - Egy peer többször megkaphat egy üzenetet,
    - Hiányzik a Usenet-ben használt szűrés

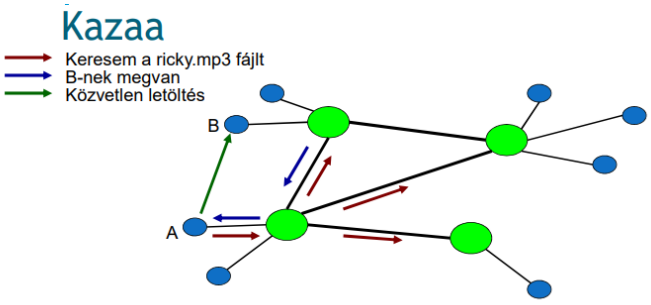
- Nagy hálózati forgalmat generál
- A keresés időtartama nem behatárolható
- A keresés sikerének valószínűsége nem ismert
- A topológia ismeretlen, az algoritmusok nem tudják felhasználni, Csak egy lépésre látok előre
- A peer-ek „hírneve” nincs figyelembe véve
- Kis méretű elérhető hálózat
  - Modemes felhasználók, kis sávszélesség a keresések továbbítására (routing black holes)
- **MEGOLDÁS**:
  - Peer hierarchia kialakítása
  - csatlakozási preferenciák
  - Nagy sávszélességű peer-ek előnyben
  - Nagyméretű megosztott állománnyal rendelkezők előnyben

Freeriding – 59

- Adar, Huberman, Freeriding on Gnutella, 2000 Sept.
  - a kliensek 70%-a nem oszt meg semmit
  - a válaszok 50%-át a peer-ek 1%-a szolgáltatja
  - a freerider-ek egyenlően oszlanak el a hálózatban
  - bizonyos peer-ek olyan fájlokat osztanak meg, melyek senkit sem érdekelnek
- Társadalmi, és nem technikai probléma
- Következmények
  - A rendszer hatékonyságának romlása (skálázhatóság?)
  - A rendszer sebezhetőbb
  - „Központosított" Gnutella – jogi problémák

KaZaA – 72-76, 80-82, 85

Architektúra



- Hierarchikus architektúra
  - A peer-ek (ON – ordinary node) egy supernode-hoz (SN) csatlakoznak
  - supernode = superpeer, ultrapeer, hypernode
  - Az SN-ek egyenrangúak
  - Az SN ismeri a hozzá tartozó ON-ek IP címeit, és a megosztott fájl-ok listáját
  - Mini Napster server
- KaZaA portok
  - Eleinte (első verziók) 1214, 80 (http)
  - ver.2.0 után jellemzően véletlen portszámok
  - Ugyanazon a porton UDP és TCP csomagok
- Titkos forráskód
  - Kód visszafejtés: ON-SN kommunikáció
  - Az SN – SN kommunikáció alig ismert
- Egy SN ismer más SN-eket
  - Gyér konnektivitás
  - Egy SN kb 20-30 más SN-hez kapcsolódik
- Bárki lehet SN
  - Számítási kapacitás, sávszélesség
  - Automatikus kiválasztás
- Több tízezer SN
  - Kb 50-200 ON/SN

SN választás

- Első futtatásnál
  - Lehetséges SN-ek IP címei az alkalmazás letöltésekor
  - Csatlakozás után egy működő SN keresése
  - Aktuális SN lista letöltése
- Későbbi alkalmakkor
  - Korábbi futtatások során 200 SN-ből álló cache lista
- SN választás különböző kritériumok alapján
  - **Terheléselosztás**
    - Nem véletlenszerűen választ, hanem a kevésbé terhelt SN-eket
  - **Lokalitás**
    - Azonos IP prefix (alhálózat) előnyben

        - Ping 5 „véletlenszerűen” választott SN felé
        - **Választás a legkisebb RTT (Round Trip Time) alapján**
        - Az SN-ek is inkább közeli SN-eket választanak szomszédnak
        - A keresési eredmények is viszonylag lokalizáltak lesznek
- Ha az SN „meghal”, új választás
- Terhelés elosztás (új SN választásnál)

Párhuzamos letöltés

- Ha több találat, a felhasználó párhuzamos letöltést választhat
    - A fájl több részre osztva HTTP byte-range header alapján
    - különbözö részek különböző peer-ektől
- Feltöltéskor a fájlhoz meta-adatok tartoznak
    - Fájl neve, mérete
    - Szerzők, média(tartalom) információ
    - ContentHash
- ContentHash – mindent fájlt hash-elnek
    - Ennek alapján hasonló tartalmak (pl. ugyanaz a szerző/szám, de különböző kódolású mp3 fájl) helyett UGYANAZT a tartalmat tölti le párhuzamosan

Hálózati forgalom fenntartása

- Egy átlagos tag érdeke a letöltés
    - A feltöltés ezért egy felesleges teher
- Karban kell tartani a felkínált tartalmat
    - Integrity Rate – 2x többet ‘ér’ az ilyen fájl
- Díjazni kell az aktív tagokat
- Participation level (részvételi szint)
    - PL = (Feltölés/Letöltés) * 100
- Szintek:
    - Low, Medium, High, Guru, Deity, Supreme Being
- PL alapján számítják a prioritásokat a várólistákon, korlátozzák a párhuzamos keresési szálakat

Előnyök

- Skálázható
    - Egy központi szerver helyett több ezer supernode
    - Csak a supernode-ek között történik elárasztás
- Sorbanállás kezelése
    - Előnyt élveznek azok, akiktől sokat töltenek le
    - Hátrány a kis sávszélességgel rendelkezőknek → lassan kerülnek kiszolgálásra
- Jogilag nehezen támadható, de nem lehetetlen

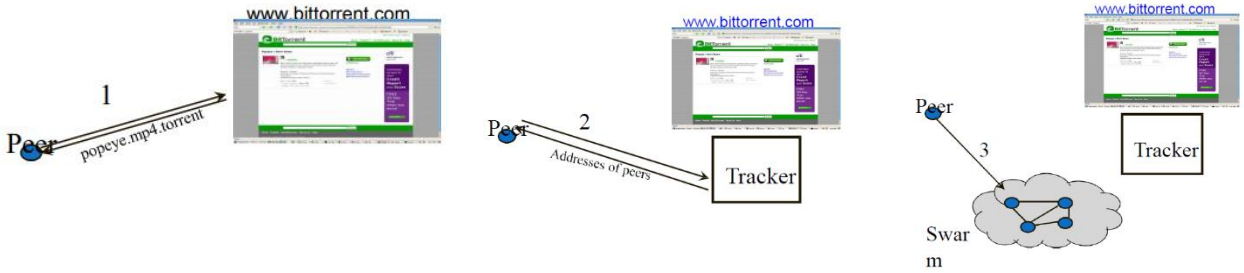
BitTorrent – 93-99, 103-110, 112

Mi a BitTorrent

Hatékony tartalomelosztási rendszer a fájl swarming/elárasztás használatával. Nem hajt végre egy tipikus p2p rendszer összes funkcióját, például a keresést.
Az áteresztőképesség teljesítmény a letöltők számával a hálózati sávszélesség hatékony használatával növekszik.

Letöltés

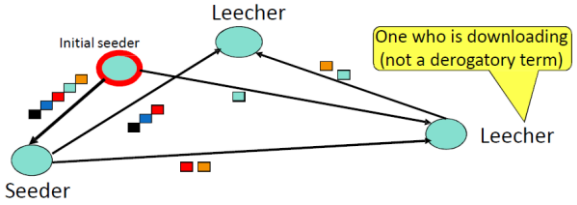
1. Egy „index fájl” popeye.mp4.torrent hosztolva van egy jól-ismert webszerveren
2. A .torrent-nek van tracker/nyomonkövethető címe a fájlhoz
3. A tracker ami a webszerveren fut szintén, nyomon követi az összes peer fájlletöltést



Fájlmegosztás

- Egy fájl vagy fájlcsoport megosztásához, a kezdeményező először létrehoz egy .torrent fájlt, egy kis fájlt, ami tartalmaz:
    - A megosztandó fájlok metaadatai
    - SHA-1 tördeli az összes darabot egy fájlban, a megbízhatóság érdekében
    - „fájlok” - lehetővé teszi több fájl letöltését
    - Információ a trackerről (pl. URL), a fájlelosztást koordináló számítógép.
- A letöltők először egy .torrent fájlt kapnak, majd csatlakoznak a megadott trackerhez, amely megmondja nekik, hogy melyik többi peer töltse le a fájl részeit.
- A swarm azon peer-ek csoportja, akik részt vesznek ugyanazon fájlok terjesztésében
- Egy peer csatlakozik a swarmhoz, megkérdezve a tracker-t, hogy egy peer listát kapjon, és csatlakozzon azokhoz a peerekhez

BitTorrent nameing convention



- **Seeder**: teljes fájlt nyújtó peer
- **Initial seeder**: az eredeti példányt nyújtó peer

Alapötlet

- Ahogy egy leecher letölti a fájlokat, létrejönnek a darabok másolatai. Több letöltés, több elérhető másolatot jelent
- Amint egy leechernek van egy teljes darabja, potenciálisan megoszthatja azt más letöltőkkel. Végül minden leecher seeder-ré válik az összes darab megszerzésével, és összeállítja a fájlt.
- Minden leecher
    - Beszámol a társainak, hogy milyen darabokkal kezdődik
    - Elkezdi cserélni ezeket a darabokat velük
- A webkiszolgáló Torrent fájlja az összes darab SHA1 hash-ját tartalmazza
- A Peers nem számol be arról, hogy van egy darabja, amíg nem ellenőrzik a hash-ét
    - Használhatott törlési kódokat

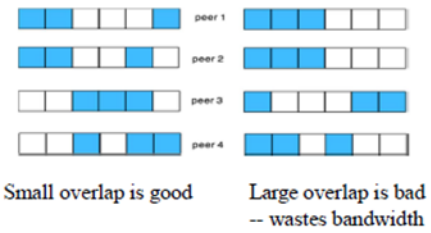
Előnyök

- A részlegesen letöltött fájlok igénybevétele
- Elutasítja a „freeloading” –t
- A leggyorsabb feltöltők jutalmazásával
- Ösztönzi a sokféleséget a „legritkább először”
- Megnöveli a swarm élettartamát
- Jól működik a „hot content” esetében

Hátrányok

- Feltételezi, hogy az összes érdekelt peer egyidejűleg aktív; a teljesítmény romlik, ha a swarm „lehűl”
- Még rosszabb: nincs nyomkövető a népszerűtlen tartalomhoz

Letöltési Stratégiák – Piece Selection – 103, 105-108



- A jó teljesítmény érdekében kritikus fontosságú a különböző peer-ek által kiválasztott darabok sorrendje
- Ha nem hatékony politikát alkalmaznak, akkor a peer-ek olyan helyzetbe kerülhetnek, ahol mindegyiknek megvan az összes azonos, könnyen hozzáférhető darabja és egyetlen hiányzó része sem.
- Ha az eredeti seed-et idő előtt leállították, akkor a fájl nem tölthető le teljesen!

Mik a „jó politikák?"

- **Szigorú prioritás (Strict Priority)**
- **Legritkább a legelső (Rarest First)**
  - Általános szabály (General rule)
  - Meghatározza a legritkább darabokat a peer-ek között, és azokat tölti le először
    - Növeli a letöltött darabok sokféleségét
    - elkerüli azt az esetet, amikor egy csomópont és minden peer pontosan ugyanazokkal a darabokkal rendelkezik; növeli az áteresztőképességet
    - Növeli a valószínűséget, ha az összes darab még rendelkezésre áll még akkor is, ha az eredeti mag elhagyja, mielőtt bármelyik csomópont letöltött egész fájlt
- **Véletlen első darab (Random First Piece)**
  - Különleges eset, az elején (Special case, at the beginning)
  - Kezdetben a peer-nek semmit sem szabad kereskednie
  - Fontos hogy ASAP megkapjuk a teljes darabot
  - Válasszon ki egy véletlen darabot a fájlból, és töltse le
- **Endgame mód (Endgame Mode)**
  - Különleges eset (Special case)
  - A vége felé, hiányzó darabokat kérnek minden őket tartalmazó peer-től
  - Ez biztosítja, hogy a letöltés nem akadályozható meg egy lassú átviteli sebességű egyetlen társa miatt.
  - Kis sávszélesség pazarlás, de a gyakorlatban ez nem túl sok.

Incentives – choking – 109

- Beépített ösztönző mechanizmus (ahol minden varázslat történik):
  - Choking algoritmus
  - Optimista unchoking
- A Choking egy ideiglenes feltöltés megtagadása
  - foglalkozik a freeriderekkel
  - Tit-for-tat stratégia játékelméleti fogalmakon alapul
- A Choking okai:
  - Freeriderek kerülése
  - Hálózati torlódások

Incentives – Optimistic unchoking – 110

- A BT peer-nek egyetlen „optimistic unchoke" van, amellyel az aktuális letöltési sebességtől függetlenül feltöltődik
  - A peerek egyszer használják a fel nem használt kapcsolatokat, hogy megtudják, lehetnek-e jobbak, mint az aktuálisak
  - Ez a peer 30 másodpercenként forog
- Okok:
  - A jelenleg használt, nem használt kapcsolatok felfedezéséhez
  - Minimális szolgáltatás nyújtása az új társaknak

P2P

Strukturált P2P – 3-5

P2P Keresés

- **Strukturálatlan P2P rendszerek:**
  - Gnutella, KaZaa, stb...
  - Véletlenszerű keresés
    - Nincs információ a fájl lehetséges tárolási helyéről
  - Nagy terhelés a rendszeren
  - Minél több helyen tárolva, annál kisebb a keresés által generált terhelés
- **Strukturált P2P rendszerek:**
  - Irányított keresés
    - Kijelölt peer-ek kijelölt fájlokat (vagy azok fele mutató pointereket) kell tároljanak
      - Mint egy információs pult
    - Ha valaki keresi a fájlt, tudja kit kérdezzen
- Elvárt tulajdonságok
  - Elosztottság
    - Felelősség elosztása a résztvevők között
  - Alkalmazkodás
    - A peer-ek ki és bekapcsolódnak a rendszerbe
    - Az új peer-eknek feladatokat osztani
    - A kilépő peer-ek feladatai újraosztani

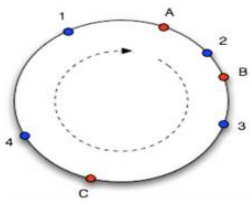
Distributed Hash Table (DHT- elosztott hash táblák)

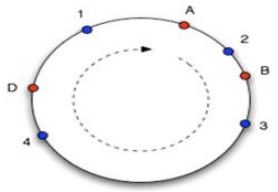
- Egy hash függvény a keresett fájlhoz egy egyéni azonosítót társít
- A hash függvény értékkészletét elosztja a peer-ek között
- Egy peer-nek tudnia kell minden olyan fájlról, melynek a
- hash-elt értéke a saját értékkészletébe tartozik
  - Vagy saját maga tárolja a fájlt....
  - Vagy tudja ki tárolja a fájlt.

Chord – 25-26,28-30, 32, 43, 45-46

Consistent Hashing

- Azonosítók egy azonosító gyűrű mentén elhelyezve modulo 2^m
  - Példa: m = 6
- Minden K kulcs az őt követő legközelebbi N csomópontnál kerül tárolásra
  - N = successor(k)
- A csomópont felelős az 1. és 4. kulcsokért
- Ha C kilép, A lesz felelős a 3. kulcsért is
- Ha D belép, átveszi a felelősséget a 3. és 4. kulcsért
  - A többi megfeleltetés változatlan



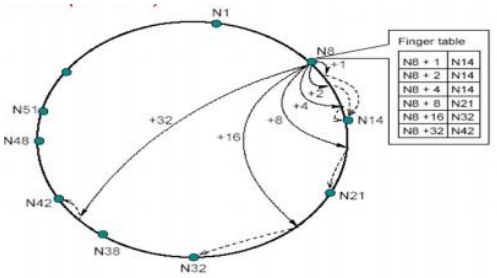


Chord: Alap Keresés

- Minden csomópont ismeri az őt követőt a gyűrűn
  - Az őt megelőzőt is hasznos ismernie
- Keresési idő ~ üzenetek száma: O(N)

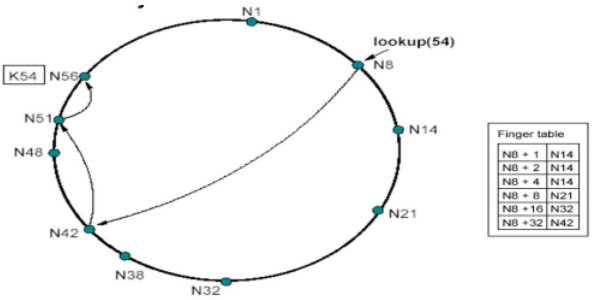
„Mutató táblák” (Finger tables)

- Minden egyes csomópont m számú további csomópontot tart nyilván
- Az előre mutató távolság exponenciálisan növekszik
    - finger[i] = successor $(n + 2)^{i-1}$



Chord: gyors/skálázódó keresés

- A mutató táblák segítségével a keresésnek O(log N) csomópontot kell bejárnia
- Minden csomópont m további bejegyzést tartalmaz
- Minél közelebbi a kulcs, annál részletesebb információval rendelkezik róla a csomópont
- Általában nem biztosítja az azonnali célba jutást



Előnyök

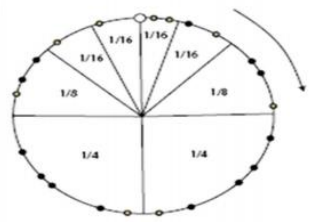
- Hatékony: O(log N) üzenet keresésenként
    - ahol N a kiszolgálók (csomópontok) száma
- Alacsony szórás a keresési időben
- Skálázódik: O(log N) állapot csomópontonként
- Robosztus: megbirkózik jelentős résztvevő változással
- Feltételezés: nincs rosszakaratú résztvevő

Hátrányok

- Merev finger (routing) tábla!
    - Nehezíti a tábla helyreállítását node-ok kiesése után
    - Lehetetlenné teszi a közelségi információ felhasználását
- A bejöv ő és kimenő irányú üzenetek eloszlása éppen ellentétes
    - Nem lehet a bejöv ő forgalmat a routing tábla frissítésére használni

Bidirectionnal Chord

- Kétirányú routing tábla
    - A tábla kétszer akkorára nő
    - Kétszer annyi vezérlő üzenetre van szükség



Klaszterek

Multiprocesszoros környezet architekturális opciók (Message Passing MAchines vs Shared Memory) – 22, 24, 28, 31

Multiprocesszorok

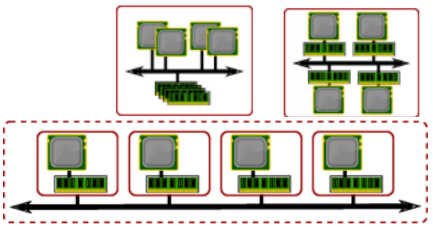
- Az egyprocesszoros sebesség folyamatosan javul, de vannak dolgok, amelyek még nagyobb sebességet igényelnek
- Multiprocesszoros szoftver probléma
    - A legtöbb kód szekvenciális (az egyprocesszorok számára)
        - Sokkal könnyebb írni és hibakeresni
    - A párhuzamos kód helyes, nagyon nehéz írni
        - A hatékony és helyes még nehezebb
        - Még nehezebb hibakeresés (Heisenbugs)

Distributed-Memory Machine

- Két fajta
    - Eloszott megosztott memória (DSM – Distributed Shared-Memory)
        - Minden processzor képes kezelni az összes memóriahelyet
        - Adatmegosztás, mint az SMP-ben
        - Más néven NUMA (nem egységes memória hozzáférés – non-uniform memory access)

        - A különböző memóriahelyek látószögei eltérőek lehetnek (a helyi hozzáférés gyorsabb, mint a távoli hozzáférés)
    - Üzenet átadása
        - A processzor közvetlenül csak a helyi memóriát kezelheti
        - A más processzorokkal való kommunikációhoz kifejezetten üzeneteket kell küldeni / fogadni
        - Multiszámítógépeknek vagy klasztereknek is nevezik
- A legtöbb hozzáférés helyi, így kevesebb memória verseny (akár 1000 processzorra is kiterjedhet)

Message Passing Machines



- Számítógépek klasztere (csoportja)
    - Mindegyik saját processzorral és memóriával rendelkezik
    - Egy összeköttetés az üzenetek között
    - Termelői fogyasztói forgatókönyv:
        - A P1 adatot D ad, SEND-et használ, hogy elküldje azt P2-re
        - A hálózat az üzenetet P2-re továbbítja
        - A P2 ezután felhívja a RECEIVE-t, hogy megkapja az üzenetet
    - Kétféle küldési primitív
        - Szinkron: A P1 leáll, amíg a P2 meg nem erősíti az üzenet fogadását
        - Aszinkron: P1 elküldi az üzenetet, és folytatja
    - Szabványos könyvtárak az üzenet továbbításához:
        - A leggyakoribb az MPI - Message Passing Interface

Message Passing

Előnyök

- Egyszerűbb és olcsóbb hardver

Hátrányok

- Az explicit kommunikációt nehéz programozni
- Kézi optimalizálást igényel
    - Ha azt szeretné, hogy egy változó helyi legyen és hozzáférhető legyen az LD / ST-en keresztül, akkor ezt be kell jelentenie
    - Ha más folyamatoknak ezt a változót olvasni vagy írni kell, meg kell írni az explicit kódot a szükséges küldéshez és fogadáshoz, ami elvégzi

Shared Memory

Előnyök

- A kommunikáció automatikusan történik
- A programozás természetesebb módja
    - Egyszerűbb helyes programok írása és fokozatos optimalizálása
- Nem kell manuálisan terjeszteni az adatokat (de segíthet)

Hátrányok

- Több hardver támogatásra van szükség
- Könnyen írható helyes, de nem hatékony programok (a távoli elérések ugyanazok, mint a helyiek)

Klaszter motiváció (Cycle stealing) – 48-51

- A munkaállomások fejlesztői eszközei érettebbek, mint a párhuzamos számítógépekhez kapcsolódó, egymással szemben álló, sajátos megoldások, elsősorban számos párhuzamos rendszer nem szabványos jellege miatt.
- A munkaállomás klaszterek egy olcsó és könnyen elérhető alternatíva a speciális nagy teljesítményű számítástechnikai (HPC –High Perfomance Computing) platformokra.

- A munkaállomások klasztereinek használata elosztott számítási erőforrásként nagyon költséghatékonyan növekszik a rendszerben!

Cycle Stealing

- Általában egy munkaállomás egy személy, csoport, osztály vagy szervezet tulajdonában lesz, amelyet a tulajdonosok kizárólagos használatra szánnak
- Ez problémákat okoz, amikor a megosztott alkalmazások futtatására szolgáló munkaállomások egy csoportját próbálják létrehozni
- Jellemzően háromféle tulajdonos van, akik a munkaállomásaikat többnyire a következőkre használják:
    1. E-mail küldése és fogadása és dokumentumok készítése.
    2. Szoftverfejlesztés szerkesztése, fordítása, hibakeresés és tesztciklus.
    3. A számítás intenzív alkalmazások futtatása.
- A cluster computing célja, hogy ellopja a tartalék ciklusokat az (1) és (2) pontokból, hogy forrásokat biztosítson a (3) számára.
- Ez azonban megköveteli a tulajdonosi akadályok leküzdését
    - az emberek nagyon védik a munkaállomásaikat.
- Általában szervezeti felhatalmazást igényel, hogy a számítógépeket ily módon használják.
- A szokásos munkaidőkön kívüli (például éjszakai) ciklusok ellopása egyszerű, az üresjárati ciklusok munkaidő alatt történő ellopása anélkül, hogy az interaktív felhasználást (CPU és memória) befolyásolná, sokkal nehezebb.

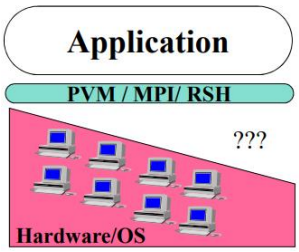
P2P számítás vs Cluster/Grid számítás

- Különbség a célközösségekben
- A Grid rendszer bonyolultabb, erőteljesebb, változatosabb és erősebb összekapcsolt erőforrásokkal foglalkozik, mint a P2P.

Klaszter erőforrás management - 76-81, 83

Cluster Munkaütemező

Tipikus Cluster Computing környezet: kép



A Cluster Computing-nak támogatni kell

- Több felhasználós, időmegosztási környezet
- Különböző CPU-sebességű és memóriaméretű csomópontok (heterogén konfiguráció)
- Sok folyamat, kiszámíthatatlan követelményekkel
- Az SMP(Symmetric Multiprocessors)-vel ellentétben: a csomópontok közötti hiányos kötések
    - Minden számítógép függetlenül működik
    - Az erőforrások nem hatékony felhasználása
- A hiányzó összeköttetést a kalszter middleware/underware biztosítja

SSI Klaszterek – SMP szolgáltatás a Cluster Computing-on

- „Pool Together” a „Cluster Wide” erőforrások
- Adaptív erőforrás-használat a jobb teljesítmény érdekében
- Könnyű használat szinte olyan, mint az SMP
- Méretezhető konfigurációk decentralizált vezérléssel
- Eredmény: HPC / HAC PC / Workstation árakon

Cluster Middleware

- Interfész a használati alkalmazások és a klaszter hardver és az OS platform között.
- A Middleware csomagok támogatják egymást a menedzsment, a programozás és a végrehajtás szintjén.
- Middleware rétegek:
    - SSI réteg

- Elérhetőségi réteg: Lehetővé teszi a klaszterszolgáltatásoka
  - Ellenőrzőpontozás, automatikus meghibásodásmegelőzés, helyreállítás a hibából,
  - az összes klaszter csomópont között működő hibatűrés.

Middleware tervezési célok

- Teljes átláthatóság (kezelhetőség)
  - Lehetővé teszi az egyetlen cluster rendszert
    - Egyetlen belépési pont, ftp, telnet, szoftver betöltése ...
- Skálázható teljesítmény
  - A klaszter egyszerű növekedése
    - nem változik az API és az automatikus terheléselosztás.
- Fokozott elérhetőség
  - Automatikus helyreállítás a hibákból
    - Ellenőrzőpontok és hibatűrő technológiák alkalmazása
  - Kezelje az adatok következetességét, ha ismétlik ...

Erőforrás Manager – Resource Manager (RM)

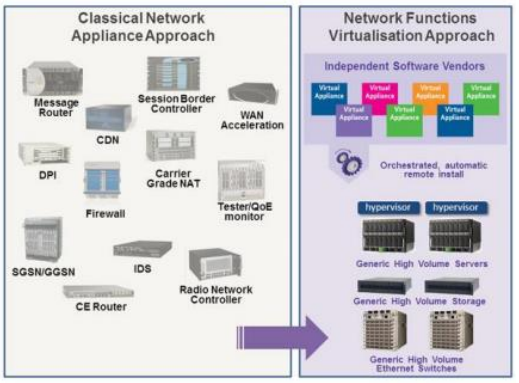
- Míg más rendszereknek szigorúbb értelmezése van az erőforrás-kezelőnek és a felelősségnek, a Moab több erőforrás-kezelő támogatása sokkal liberálisabb értelmezést tesz lehetővé.
- Lényegében minden olyan objektum, amely környezeti információkat és környezeti ellenőrzést nyújt, erőforrás-kezelőként használható.
- A Moab képes a több független forrásból származó információkat összevonni egy nagyobb, teljes körű világképre a klaszterről, amely az összes olyan információt és vezérlést tartalmazza, amely egy szabványos erőforrás-kezelőben, például a TORQUE-ban található:
  - Csomópont (Node)
  - Munka (Job)
  - Sorkezelési szolgáltatások

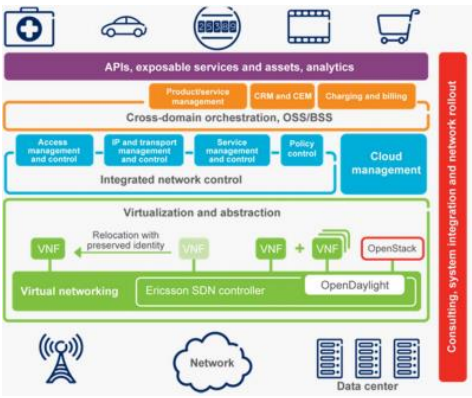
NFV Network Function Virtualization)

NFV koncepció – 9-11

Hálózati funkciók virtualizálása

- hálózati funkció (pl. gyorsító-tárazás, tűzfal) leválasztás a célhardver berendezéstől
- szoftverben megvalósított hálózati funkció
  - tetszőleges általános szerver architektúrán futhat
- Szolgáltatói szempontok
  - CapEx/OpEx költségek csökkentése
  - gyorsabb szolgáltatás létesítés
  - igazodás a változó igényekhez
- Fórumok
  - ETSI NFV
  - Open Platform for NFV (OPNFV)

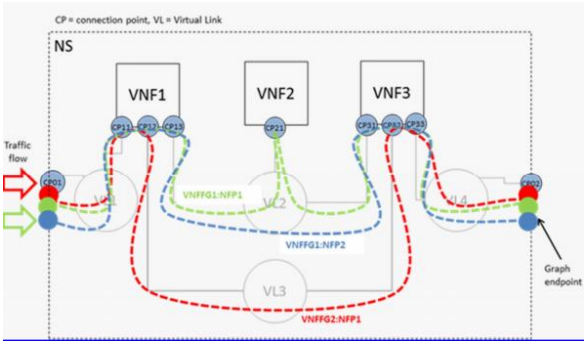




Távközlési felhő

- Virtualizált távközlési funkciók
  - csomagkapcsolt maghálózat (EPC)
  - IMS/VoLTE komponensek (CSCF, HSS, stb.)
  - tartalomszolgáltató hálózat (CDN)
  - csomagtartalom vizsgálat (DPI)

- Teljesítmény
    - terheléskiegyenlítés, skálázhatóság
    - virtuális funkciók közel mozgatása a felhasználási pontokhoz
    - távközlési szintű szolgáltatás
        - létesítés, monitorozás, helyreállítás, számlázás
    - hardveres gyorsítás szükségessége
        - hálózati kártya, virtuális kapcsoló
- Ericsson: valós -idej ű távközlési felhő
    - SDN, NFV és felhő kombinációja



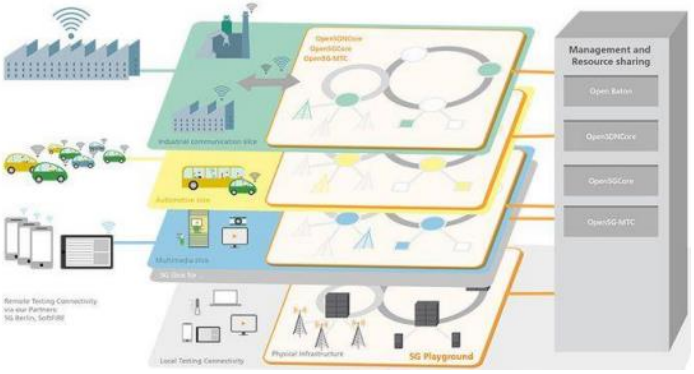
Dinamikus szolgáltatás láncolás

- Egy új szolgáltatás (NS) = VNF-ek összekötése (Virtual Network Function)
- Gráffal lehet leírni

Slicing – 17-18

Network Slicing

A hálózati szeletelés lényegében több virtuális hálózat létrehozását teszi lehetővé a megosztott fizikai infrastruktúrán . Az alapul szolgáló fizikai hardverből származó hálózati erőforrások elvonásához a vezérlési síkot és a felhasználói síkot elválasztjuk. Ez lehetővé teszi, hogy a felhasználó-sík funkcionalitása átálljon a hálózat szélére, és a felügyeleti funkciók maradjanak a magban. Ez a kialakult forgatókönyv lehetővé teszi a hálózati szeletelés végrehajtását

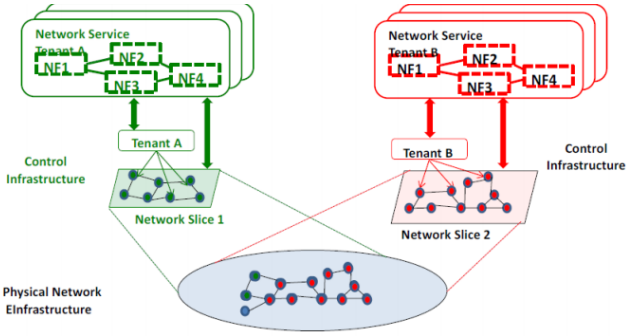


Definíció:

- Funkciókat összekötő gráf
    - Network Functions (VNF, PNF)
- End-to-end hálózati szolgáltatás nyújtás céljából
- Specifikus elvárások és képességek

Végső felhasználók (retail users)

- Slice felhasználója = szolgáltató
- End-user = szolgáltató felhasználója



Slicing Koncepció

- NF – hálózati funkciók
    - Virtualizált eset: NFV
    - Egyazon fizikai hálózat felett

Edge Computing – 26

- A szolgáltatás az UE hozzáférési pontjának közelében található
- Hatékony szolgáltatásszállítást ér el
- Csökkent a végpontok közötti késleltetést és a terhelést a közlekedési hálózatban