Erőforrás virtualizálás

Hogyan lesz egyetlen fizikai szerverből sok, egymástól elszigetelt bérlő — hypervisorok, virtualizációs technikák és a privát felhő platformok.

Honnan indulunk — és hova jutunk el

1. blokk — A „software stack" és a hypervisor: hogyan lesz egy fizikai gépből sok, elszigetelt virtuális gép

2. blokk — Konkrét környezetek és virtualizációs technikák: QEMU, XEN, paravirtualizáció, HVM, KVM

3. blokk — Privát felhő platformok: Eucalyptus, Open-Nebula, OpenStack

4. blokk — Konkrét termékek, klaszterek, platform-összehasonlítás, és hogyan kapcsolódik mindez az AWS-hez

Eddig azt néztük, *hogyan kommunikálnak* a felhő szerverei egymással (hálózat). Most azt nézzük meg, *mi fut* valójában azokon a szervereken — és hogyan osztozik rajtuk sok egymástól független bérlő.

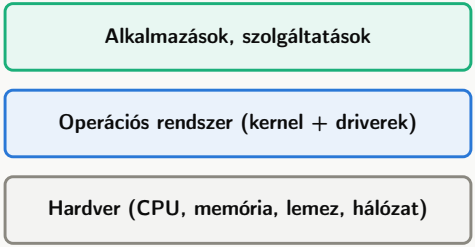
Miért van erre egyáltalán szükség?

Egy felhőszolgáltató (pl. AWS) adatközpontjában a fizikai szerverek száma véges — a bérlők (ügyfelek) száma viszont sokszorosa ennek. Ha minden ügyfél egy teljes fizikai szervert kapna, az szörnyen pazarló és drága lenne.

⚠ A kérdés tehát: hogyan osszunk meg **egyetlen fizikai gépet** sok, egymástól **semmit nem tudó, egymást nem zavaró** bérlő között — úgy, hogy mindegyik azt hihesse, az egész gép az övé?

A hagyományos „software stack”

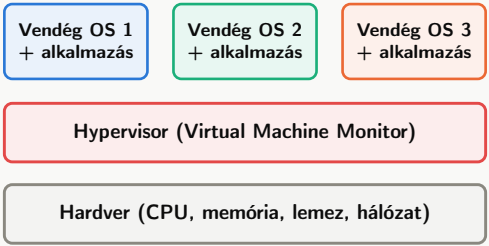
Egy szerver hagyományosan rétegekből épül fel — legalul a fizikai hardver, legfelül az, amiért a gépet valójában üzemeltetjük.



Ez a felépítés egyetlen bérlőnek tökéletes — de egyetlen operációs rendszer fut rajta, amit csakis egyvalaki birtokolhat teljes egészében.

A virtualizált „software stack”

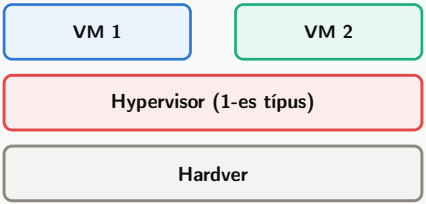
Virtualizáció esetén egy új réteg kerül a hardver fölé: a **hypervisor** (Virtual Machine Monitor, VMM) — ő osztja szét és szigeteli el a fizikai erőforrásokat a rajta futó virtuális gépek (VM-ek) között.



Mindhárom vendég operációs rendszer azt hiszi, saját, önálló gépen fut — valójában a hypervisor osztja szét köztük ugyanazt a fizikai hardvert.

Hypervisor-típusok: 1-es (bare-metal)

Közvetlenül a hardveren fut, nincs alatta külön „host” operációs rendszer.

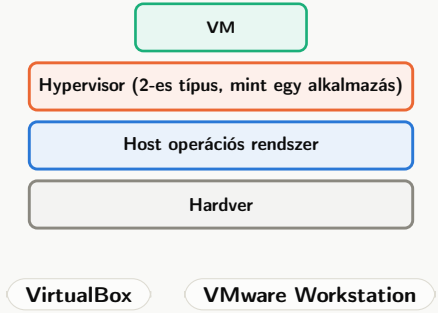


VMware ESXi **Microsoft Hyper-V** **Xen** **KVM***

Ez a jellemző megoldás szerverkörnyezetben, felhőszolgáltatóknál. ***A KVM különleges eset**.

Hypervisor-típusok: 2-es (hosted)

Egy már futó operációs rendszeren belül, **alkalmazásként** fut.



⚠ Kényelmesebb fejlesztői/tesztelési célra (gyorsan telepíthető egy már meglévő asztali gépre), de a plusz réteg miatt nagyobb overhead-del jár — szerverkörnyezetben ritkán használt.

2. BLOKK

Környezetek és virtualizációs technikák

QEMU, XEN, és a három alapvető módszer: paravirtualizáció, HVM, KVM.

QEMU

Nyílt forráskódú **gépemulátor** és virtualizáló szoftver.

Önmagában, hardveres gyorsítás *nélkül* is képes teljes rendszereket emulálni — akár **eltérő processzorarchitektúrát** is (pl. ARM emulálása x86-os gépen). Ennek ára: minden processzorutasítást szoftveresen kell értelmeznie, ami jelentős teljesítményveszteséggel jár.

Linux alatt tipikusan **KVM-mel párosítva** használják: a KVM végzi a CPU/memória virtualizációját hardveres gyorsítással, a QEMU pedig az eszközök (hálózati kártya, lemez, grafikus kártya) emulálásáért felel — így közel natív teljesítmény érhető el, teljes eszközemulációval kombinálva.

XEN

A Cambridge-i Egyetemen indult, ma nyílt forráskódú, **bare-metal (1-es típusú)** hypervisor.

Az egyik úttörője volt a **paravirtualizációnak**: a korai x86-os processzorok még nem támogatták hardveresen a virtualizációt, ezért a Xen a vendég operációs rendszer forráskódjának (kernel) módosításával érte el a hatékony működést.

⚠ **Történelmi jelentősége van az AWS szempontjából is:** az Amazon EC2 hosszú éveken át Xen alapokon futott, mielőtt áttértek volna a saját fejlesztésű, KVM-alapokra épülő **Nitro** rendszerre.

Paravirtualizáció

A vendég operációs rendszer **„tudatában van”** annak, hogy virtualizált környezetben fut.

A kritikus, hardverközeli műveleteket (pl. memóriakezelés, megszakítások) nem a valódi hardver felé, hanem közvetlenül a hypervisor felé, egy erre kialakított interfészen (**hypercall**) keresztül kéri.

Ez elkerüli a drága hardveremulációt — cserébe a vendég operációs rendszer **kernelét módosítani kell**, ami nem minden esetben lehetséges (pl. zárt forráskódú, nem támogatott rendszerek esetén).

Hardware Virtual Machine (HVM)

A modern processzorok **hardveres virtualizációs kiegészítéseit** (Intel VT-x, AMD-V) kihasználó megoldás.

Ezekkel a kiegészítésekkel a processzor **hardveresen** ismeri fel és kezeli a privilegizált utasításokat, így a hypervisornak nem kell szoftveresen emulálnia vagy a vendég kernelt módosítania.

Teljesen **változtatás nélküli** („unmodified"), akár zárt forráskódú operációs rendszer (pl. Windows) is virtualizálható vele — közel natív teljesítménnyel.

Kernel-based Virtual Machine (KVM)

A **Linux kernelbe épített modul**, ami magát a Linux kernelt alakítja hypervisorrá.

A HVM hardveres kiegészítéseket (VT-x/AMD-V) használja ki, és a Linuxot 1-es típusú hypervisorként viselkedteti. Önmagában csak a CPU és memória virtualizációját végzi — az eszközök emulálásához jellemzően a korábban tárgyalt **QEMU-val** párosítják.

Nyílt forráskódú, ingyenes, és mivel a Linux kernel részét képezi, rendkívül elterjedt — ezt használja többek között a **Proxmox**, az **OpenStack** alapértelmezett hypervisoraként, és ez az alapja az **AWS jelenlegi Nitro rendszerének** is.

Dilemma: a KVM 1-es típus — vagy hibrid?

Épp emiatt a KVM nem illeszkedik tisztán a korábban tanult 1-es/2-es tipológiába — két, egyaránt védhető szempont létezik.

1. szempont — privilégium-szint (a klasszikus tankönyvi kritérium): közvetlen, hardveres virtualizációs joggal fut-e a hypervisor, vagy egy másik OS-en keresztül kell azt kérnie? A KVM-modul kernel-szinten, közvetlenül fut ⇒ **1-es típus**.

2. szempont — egyidejű jelenlét a CPU-n: fut-e mellette teljes, általános célú host OS, ami a VM-ekkel versenyezve kap CPU-időt ugyanattól az ütemezőtől? A host Linux és a VM-ek (QEMU-folyamatokként) a KVM esetén egyenrangú entitások ⇒ **hibrid**.

Nincs egyetlen, egyöntetűen „helyes" válasz — attól függ, melyik szempontot tartjuk fontosabbnak: a valóság gyakran nem illeszkedik tökéletesen a tankönyvi kategóriákba.

Paravirt. vs. HVM vs. KVM — összefoglalva

Szempont	Paravirt.	HVM	KVM
Vendég kernel módosítása	Igen	Nem	Nem
Hardveres gyorsítás szükséges	Nem	Igen (VT-x/AMD-V)	Igen (VT-x/AMD-V)
Zárt forráskódú vendég OS	Nem	Igen	Igen
Jellemző megvalósítás	korai Xen	modern Xen, VMware	Linux kernel modul

A fejlődés iránya jól látszik: a kezdeti, kernel-módosítást igénylő paravirtualizációtól eljutottunk odáig, hogy a hardver maga segít a virtualizációban — ezt használja ki mind a HVM, mind a KVM.

3. BLOKK

Privát felhő platformok

A puszta hypervisor még nem felhő — kell hozzá menedzsment, API, önkiszolgálás is.

Miért nem elég önmagában a hypervisor?

Egy hypervisor (pl. KVM) önmagában csak virtuális gépeket tud **egyetlen fizikai gépen** futtatni.

⚠ Egy valódi felhő (akár privát, akár publikus — emlékezzünk a korábban tárgyalt **felhő-típusokra**) ennél sokkal több: önkiszolgáló felület, API, erőforrás-nyilvántartás **sok szerveren át**, hálózat- és tárolás-menedzsment, jogosultságkezelés.

ℹ Ezt a hiányzó réteget adják a **privát felhő platformok** — ők szervezik egységes, API-n keresztül elérhető felhővé a sok, önmagában „buta” hypervisort.

Eucalyptus

Az egyik **legkorábbi (2008-as)** nyílt forráskódú privát felhő platform.

Kifejezetten az **AWS API-jaival** (EC2, S3) kompatibilis felületet biztosított — az AWS-re írt eszközök/scriptek egy saját, helyben üzemeltetett felhőn is futtathatók voltak.

A HP felvásárlása (2014) után a fejlesztése lelassult — mára inkább csak **történeti jelentősége** van, éles környezetben aktívan nem igazán terjedt el.

Open-Nebula

Nyílt forráskódú, viszonylag **könnyűsúlyú** felhő-menedzsment platform privát és hibrid felhők kialakítására.

Az OpenStacknál egyszerűbb architektúrájú, **gyorsabban üzembe helyezhető** — cserébe kevésbé moduláris/rugalmas nagy, komplex, sok komponensből álló környezetekhez.

OpenStack

A **legszélesebb körben elterjedt** nyílt forráskódú IaaS felhőplatform.

Saját, AWS-hez hasonló szolgáltatáskínálatú privát vagy hibrid felhő építhető ki vele — **moduláris felépítésű**, önálló, de egymással együttműködő szolgáltatásokból áll.

Nagyvállalati és telekommunikációs környezetekben (pl. saját adatközponttal rendelkező szolgáltatóknál) különösen elterjedt, ahol fontos szempont, hogy ne legyenek egy adott nyilvános felhőszolgáltatóhoz kötve (**vendor lock-in** elkerülése).

OpenStack — a legfontosabb komponensek

Nova

Számítási erőforrások (VM-ek) kezelése

Neutron

Hálózatkezelés

Cinder

Blokk tárolás

Swift

Objektum tárolás

Glance

Image-ek (virtuálisgép-sablonok) kezelése

Keystone

Azonosítás- és jogosultságkezelés

Horizon — webes admin felület

4. BLOKK

Konkrét termékek — és az AWS

Milyen valódi, telepíthető szoftverek valósítják meg mindezt — és hol fut ez a ma használt AWS mögött?

Virtualizáló szoftverek a gyakorlatban

A fentebb tárgyalt virtualizációs típusokat (paravirtualizáció, HVM) a gyakorlatban konkrét, telepíthető hypervisor-termékek valósítják meg.

Red Hat Virtualization (RHV) — KVM alapú

Hyper-V — Windows **z/VM — IBM**

VMware ESXi **Oracle VM Server**

Elsősorban vállalati/szerver környezetben használt, kereskedelmi vagy vállalati támogatással is elérhető megoldások.

Hypervisor-klaszterek

Eddig egyetlen fizikai gépen futó hypervisort néztünk — éles környezetben viszont jellemzően **több fizikai hostot** kapcsolnak össze egy **klaszterbe**, központi menedzsmenttel.

Élő migráció

Egy futó VM leállás nélkül átköltöztethető egyik hostról a másikra (pl. karbantartás előtt)

Magas rendelkezésreállás

Ha egy host meghibásodik, a VM-ei automatikusan újraindulnak egy másik hoston

Erőforrás-kiegyenlítés

A klaszter dinamikusan a kevésbé terhelt hostok felé mozgatja a VM-eket

⚠ Ehhez a hostoknak közös, **megosztott tárolót** kell használniuk — erről részletesen a következő órán (Adattárolás a felhőben) lesz szó.

Platformok a gyakorlatban

A leggyakrabban előkerülő megoldások — honnan jöttek, és mennyire nyílt/zárt a licencük.

VMware vSphere / ESXi

saját VMkernel — iparági etalon
zárt; 2023 (Broadcom) óta csak drága előfizetés

Proxmox VE

KVM + LXC, beépített klaszterezés
nyílt forráskódú, teljes funkcióval ingyenes

OpenStack

KVM + moduláris szolgáltatások
nyílt forráskódú, magas szaktudás-igény

Hyper-V

Windows Serverbe épülő hypervisor
Windows Server része, van ingyenes verzió

Vissza az AWS-hez: a Nitro rendszer

Amivel az órát elkezdtük (XEN, EC2 múltja) és amivel ma dolgozunk, most összeér.

⚠ Az Amazon EC2 kezdetben **Xen** alapokon futott (paravirtualizáció, majd HVM) — ez volt a felhő egyik első, nagy léptékű, nyilvánosan elérhető virtualizációs megvalósítása.

✔ Mára az AWS a saját fejlesztésű **Nitro** rendszerére épül, aminek az alapja a korábban tárgyalt **KVM** — a hálózati és tárolási virtualizáció egy részét pedig dedikált Nitro-hardver (kártyák) veszi át a fő CPU-tól, hogy a bérlők szinte a teljes gép teljesítményét megkapják.

ÖSSZEFOGLALÁS

Mit vittünk haza?

- A **hypervisor** egy új réteget told a hardver és a vendég operációs rendszerek közé, hogy egyetlen gépen sok, egymástól elszigetelt VM fusson
- **1-es típus (bare-metal)** a szerverek, **2-es típus (hosted)** a fejlesztői/asztali gépek világa
- A **paravirtualizáció** (vendég kernel módosítása) után a **HVM** és a **KVM** hardveres gyorsítással váltotta ki a drága szoftveres emulációt
- A puszta hypervisor még nem felhő — ehhez **privát felhő platformok** (Eucalyptus, Open-Nebula, **OpenStack**) kellenek, amik menedzsmentet, API-t, önkiszolgálást adnak hozzá
- Az AWS is ezen az úton haladt végig: **Xen** → saját, **KVM-alapú Nitro** rendszer

KÖVETKEZŐ ÓRA: ADATTÁROLÁS A FELHŐBEN

Kérdések?

A fizikai hardvertől eljutottunk a hypervisorokon át a privát felhő platformokig — és visszaértünk az AWS Nitro rendszeréhez.