NJE GAMF · FELHŐALAPÚ SZOLGÁLTATÁSOK · 2. ÓRA

Felhő ökoszisztéma

Szolgáltatási modellek, az AWS teljes szolgáltatás-térképe, a globális infrastruktúra felépítése, és hogy mit kapunk automatikusan — anélkül, hogy kérnénk.

Neumann János Egyetem · GAMF Műszaki és Informatikai Kar

2026/2027 őszi félév

ISMÉTLÉS & MAI TERV

Honnan indultunk — és hova tartunk ma

📌 Múlt óra röviden

- Bérelt tárhelytől a VPS-en át a felhőig
- AWS regisztráció, free tier, biztonságos bankkártya-választás
- A felhő definíciója: megosztott, elasztikus erőforrások

🎯 Mai óra — 4 blokk

- **1.** Szolgáltatási modellek (IaaS/PaaS/SaaS/DBaaS/IaC) + felelősségmegosztás
- **2.** Az AWS teljes szolgáltatás-térképe — mire való melyik
- **3.** Globális infrastruktúra: régiók, zónák, és mi automatikus
- **4.** Vendor lock-in — a felhő ára

Miért fontos ez?

A projektfeladatban ASG-t, LB-t, RDS-t, VPC-t, S3-at fogtok használni. Ma minden egyes ilyen szolgáltatásról lesz szó — mire való, mikor válasszuk, és a nagy kép, ahova illeszkedik.

MOTIVÁCIÓ

Miért ilyen bonyolult ez az egész?

A méretgazdaságosság (skálázás) két, egymásnak feszülő elvárást támaszt egy felhőszolgáltatóval szemben:

⚡ Rugalmasság

A rendszernek pillanatok alatt kell tudnia reagálnia egy hirtelen megnövekedett terhelésre — akár tízszeres forgalomra is percek alatt.

🌱 Energiahatékonyság

Ugyanakkor amikor nincs terhelés, nem fizethetjük (és a szolgáltató nem égetheti el) a felesleges energiát — ez költség és fenntarthatósági kérdés is.

🎭 Elrejtett komplexitás

Ezt a két, feszültségben álló elvárást egy egyszerű, néhány kattintásos felület mögé kell rejteni — ez indokolja a mai óra témáit.

Ez a mai óra vezérfonala

Minden, amit ma látunk — szolgáltatási szintek, régiók, zónák, automatikus redundancia — végső soron ennek a két elvárásnak (rugalmasság + hatékonyság) az egyidejű kiszolgálására született.

SZOLGÁLTATÁSI SZINTEK

Egy spektrum: mennyit intézünk mi, mennyit a szolgáltató

Minél feljebb megyünk, annál kevesebb az üzemeltetési terhünk — de annál kevesebb a kontrollunk is. Négy megállót nézünk végig ezen a skálán, párszáz szóban és néhány példával mindegyikhez — plusz egy ötödik témát (IaC), ami nem maga a skála, hanem az eszköz, amivel bármelyik megállót leírjuk.



← Teljes kontroll, teljes üzemeltetési teher

Nulla üzemeltetés, kész termék →

Hol a helye az IaC-nek?

Az **Infrastructure as Code** nem egy hatodik szint ezen a spektrumon — hanem egy módszer, amivel bármelyik szintet (jellemzően IaaS-t) leírhatjuk és automatizálhatjuk kattintgatás helyett. Külön beszélünk róla néhány dia múlva.

SZOLGÁLTATÁSI SZINTEK

IaaS Infrastructure as a Service

Tárhelyet, hálózatot és alapvető számítási erőforrást kapunk — a belső infrastruktúrához nem férünk hozzá, de a virtuális gép erőforrásait (CPU, memória, tárhely, OS, terheléselosztás) szabadon testreszabhatjuk. Ez áll legközelebb a korábban tárgyalt VPS-hez.

Mikor jó választás?

Induló vállalkozásnak, aki számítási kapacitást akar saját szerverüzemeltetés nélkül — vagy gyorsan növekvő szervezetnek, ahol a rugalmasság a fő szempont.

PÉLDÁK A PIACON

AWS EC2 | Google Compute Engine | Azure Virtual Machines | DigitalOcean Droplets

SZOLGÁLTATÁSI SZINTEK

PaaS Platform as a Service

A szolgáltató adja az operációs rendszert, a futtatókörnyezetet, a hálózatot — nekünk csak a saját programunkra és annak konfigurációjára kell koncentrálnunk, feltéve hogy a szolgáltató támogatja az általunk használt programkészletet.

Mikor NEM jó választás?

Ha az alkalmazásnak hordozhatónak kell lennie, ha szabadalmaztatott/egyedi nyelvi környezetet használunk, vagy ha az erőforrásokat finoman konfigurálni akarjuk. Jellemzően több fejlesztő közös, automatizált fejlesztés/teszt munkafolyamatához illik jól.

PÉLDÁK A PIACON

- AWS Elastic Beanstalk
- Heroku
- Google App Engine
- Azure App Service

SZOLGÁLTATÁSI SZINTEK

SaaS Software as a Service

Sem az alkalmazás kódját, sem az alatta futó infrastruktúrát nem ismerjük (és nem is kell). Használjuk, esetleg testreszabjuk a felületét (logó, név, kinézet) — de a technológiai részletek teljesen rejtve maradnak.

Egy tervezési szempont, amit érdemes tudni

Sok "klasszikus" SaaS (pl. levelezés, számlázás, backoffice rendszerek) **nem garantál valós idejű feldolgozást** — ez nem minden SaaS-ra igaz (pl. egy Slack vagy Google Docs igen is valós idejű), de érdemes rákérdezni, mielőtt egy SaaS-ra időkritikus folyamatot építünk.

PÉLDÁK A PIACON

- Gmail / Google Workspace
- Microsoft 365
- Salesforce
- Dropbox
- Slack

SZOLGÁLTATÁSI SZINTEK

DBaaS Database as a Service

Egy külön kiemelt kategória: azonnali skálázhatóságot és teljesítményt garantál, a legújabb adatbázis-technológiákra épül, és beépített hibakezeléssel rendelkezik.

Self-service — konfigurálás és telepítési díj nélkül azonnal indítható, nem csökkentett teljesítménnyel

Eszköz- és helyfüggetlen erőforrás — bárhonnan, bármilyen kliensről elérhető

Csak a használatért fizetünk — nincs előre lekötött kapacitás

Zökkenőmentes verzióváltás — a technológia/verzió cseréje nem a mi gondunk

Rétegei: UI réteg (elérhetővé tétel) → alkalmazás réteg (szolgáltatáslogika) → adatbázis réteg (lekérdezés-gyorsítótárazás) → adattároló réteg (titkosítás, mentés, lemezfigyelés).

PÉLDÁK A PIACON

AWS RDS | AWS DynamoDB | Azure SQL Database | Google Cloud SQL | MongoDB Atlas

SZOLGÁLTATÁSI SZINTEK

IaC Infrastructure as Code

Az infrastruktúra leírása magas szintű, deklaratív konfigurációs fájlokkal — kézi kattintgatás helyett. Megadjuk, milyen erőforrásokat (EC2, VPC, RDS…) szeretnénk és milyen kapcsolatban álljanak egymással; az eszköz kitalálja, hogyan érje ezt el.

Gyakori tévhit

Az IaC **nem** követeli meg sem a mikroszolgáltatás-architektúrát, sem a konténereket. Ugyanígy leírható vele egy egyszerű, monolitikus alkalmazáshoz tartozó hagyományos virtuális gép is.

Miért éri meg?

A leírt állapot **verziókezelhető** (git), **újrafelhasználható** (több környezet: teszt/éles), és **automatizáltan alkalmazható** — pont ez lesz a témája a féléves IaC gyakorlatnak is (Terraform + Ansible + Jenkins).

PÉLDÁK A PIACON

Terraform **Ansible** **AWS CloudFormation** **Pulumi**

SZOLGÁLTATÁSI SZINTEK

Megosztott felelősség — ki véd meg mit?

A leggyakoribb felhős biztonsági incidens nem az AWS hibája — hanem a miénk, mert nem tudtuk, hogy egy adott réteg a mi felelősségünk volt.

RÉTEG	SAAS	PAAS	IAAS
Felhasználói hozzáférés, adatok	mindig az ügyfél felelőssége		
Alkalmazás	**AWS**	ügyfél	ügyfél
Operációs rendszer	**AWS**	**AWS**	ügyfél
Hálózati konfiguráció	**AWS**	**AWS**	ügyfél
Hypervisor, fizikai infrastruktúra	**AWS**	**AWS**	**AWS**

A mai óra hátralévő része pontosan erről a legalsó rétegről szól

Mielőtt eldöntjük, mi a mi felelősségünk, értenünk kell, mit épít fel helyettünk az AWS a fizikai infrastruktúra szintjén.

SZOLGÁLTATÁSI SZINTEK

Privát, publikus, hibrid felhő

🔒 Privát felhő

Saját (vagy elkülönítetten bérelt) infrastruktúra. Nem korlátozódik a saját telephelyre — VPS/VDS/bérelt szerver is lehet, jellemzően VPN-alagúton át.

pl. saját OpenStack

🌐 Publikus felhő

Megosztott infrastruktúra egy szolgáltatónál. Nincs üzemeltetési többletköltség, a CPU-idő alapú árazás rugalmas. **Ezzel dolgozunk a félévben.**

pl. AWS, Azure, GCP

🔗 Hibrid felhő

A kettő kombinációja: érzékeny adat marad helyben, a többi kimehet a publikus felhőbe, kihasználva annak minden előnyét.

pl. AWS Outposts, Azure Arc

A mai óra hátralévő részében kizárólag a **publikus felhővel** foglalkozunk — előbb áttekintjük az AWS teljes szolgáltatás-térképét, majd megnézzük, fizikailag hogyan épül fel mindez.

2. BLOKK

Az AWS szolgáltatás-térképe

Az AWS induló, IaaS-szintű ajánlata mára tucatnyi, egymásra épülő szolgáltatássá nőtte ki magát. Vegyük végig a legfontosabbakat — ezekkel fogtok dolgozni a gyakorlaton és a projektben is.

EC2 Elastic Compute Cloud

Szerverpéldányok futtatása a felhőben, egyszerű felületen keresztül. Ez a "munkaló" szolgáltatás — erre épül a legtöbb más AWS-komponens.

Operációs rendszer választék: Linux disztribúciók, OpenSolaris, FreeBSD, NetBSD, Windows Server

Régió + zóna kötelező megadása indításkor, valamint a teljesítmény szerinti típus

Dinamikus IP alapból — leállításkor elveszik, statikushoz Elastic IP kell (2024 óta mindig fizetős)

AMI-ból indul (Amazon Machine Image) — előre elkészített vagy saját mentett képfájlból

S3 Simple Storage Service

Objektumtárolás — írás, olvasás, törlés. Korlátlan mennyiségű objektum, egyenként 1 bájttól 5 terabájtig.

Bucket-ekben tárol, egyedi azonosítóval, kiválasztott régióban

ACL-lel (access control list) szabályozható a hozzáférés

REST API, HTTP(S) alapon (a régebbi SOAP felületet az AWS 2013 óta nem támogatja)

MD5 hash (ETag) minden fájlhoz — letöltés után ellenőrizhető a sértetlenség

Előreutalás

Az S3 az egyik szolgáltatás, ami **automatikusan** több zóna között replikálódik — erről bővebben a 3. blokkban.

EBS Elastic Block Store

Tartós blokktár, amit egy EC2-höz csatolva használunk — mintha egy fizikai lemez lenne. Kötettípustól függően akár 16 tebibájtig terjedhet.

Egy EC2 több EBS-t is használhat, de egy EBS jellemzően csak egy EC2-höz aktív (kivétel: io1/io2 Multi-Attach)

Pillanatkép (snapshot) készíthető belőle, amiből új EC2 indítható

Előreutalás

Az EBS **csak a saját zónáján belül** replikálódik — ez az egyik legfontosabb tervezési csapda, amiről a 3. blokkban lesz szó.

AWS SZOLGÁLTATÁSAI · ADATBÁZIS

DynamoDB

natív NoSQL adatbázis

Kulcs-érték és dokumentum modellű adatbázis, web szolgáltatásokon keresztül elérve.

Globális táblák: földrajzilag távoli helyeken is képes másolatot tartani

Automatikus skálázás a használat alapján — erőforrás, index, replikáció mind kezelve

Történeti megjegyzés

Az AWS korábbi hasonló szolgáltatása a **SimpleDB** volt — évek óta nem ajánlott új projekthez, helyette mindig DynamoDB-t válasszunk.

AWS SZOLGÁLTATÁSAI · KOMMUNIKÁCIÓ

SQS Simple Queue Service

Üzenetsor szolgáltatások szinkronizálására — bejelentkezés nélkül lehet adatot küldeni bele.

Zárolás feldolgozás alatt — így egy üzenetet nem dolgoz fel véletlenül kétszer

Sikertelen feldolgozás esetén a zárolás feloldódik, az üzenet újra kivehető

Megosztható különböző regisztrációk (fiókok) között

IP és idő alapú hozzáférés-tiltás lehetséges

Mire jó igazán?

Komponensek szétválasztására (decoupling): a küldő nem várja meg, amíg a fogadó feldolgozza az üzenetet — csökkenti az egymástól való közvetlen függést.

AWS SZOLGÁLTATÁSAI · MONITOROZÁS

CloudWatch

monitorozó rendszer

Telemetria adatokat gyűjt, amivel az alkalmazás optimalizálható — anélkül, hogy bármit is telepítenünk kellene.

Alap monitorozás ingyenes — 5 percenkénti mintavétellel

Részletes (1 perces) mintavételezés is kérhető, felárral

Tucatnyi metrika közül választhatunk kattintgatás nélkül

Figyeli pl. az EBS elérési idejét, adatbázis-hozzáférést, SQS-ben tárolt üzenetszámot

Miért fontos ez a projektnek?

A CloudWatch metrikái (pl. CPU-kihasználtság) indítják az **Auto Scaling** triggereket — enélkül nincs automatikus skálázás.

VPC Virtual Private Cloud

Saját, elkülönített, logikailag izolált virtuális hálózat az AWS-en belül — modellezhetjük egy routerként: alhálózatok, route tábla, internet kilépési pont (Internet Gateway).

Security Group + Network ACL — tűzfalként szabályozzák a forgalmat a VPC-n belül

Nem kötelező egy meglévő céges hálózathoz kapcsolódnia

Site-to-Site VPN — ha mégis kell, VPN alapon köti össze a helyi hálózattal

Direct Connect — dedikált vonalas alternatíva a VPN helyett

Auto Scaling

Auto Scaling Group (ASG)

Szabályok alapján dinamikusan változtatja az erőforrásaink darabszámát — csoportba szervezett EC2 példányokat monitoroz és triggerek alapján indít/állít le közülük.

Leggyakoribb trigger: CPU — magas kihasználtságnál új példány indul, alacsonynál leáll egy

Min/max/desired — létrehozáskor megadjuk, milyen határok között mozoghat a darabszám

Egészség-teszt (health check) táplálja a Load Balancert, hogy csak elérhető gépek kapjanak forgalmat

CloudFormation

AWS natív Infrastructure as Code

JSON (vagy YAML) fájlban írjuk le az alkalmazás futtatásához szükséges infrastruktúrát — az erőforrások paramétereivel együtt (pl. instance típusa, adatbázis portja/mérete).

Hogyan viszonyul a korábban látott IaC eszközökhöz?

A CloudFormation az AWS **saját, natív** megoldása — csak AWS-en működik, cserébe mélyen integrálódik. A Terraform ezzel szemben szolgáltató-független (multi-cloud). A féléves IaC gyakorlaton a Terraform+Ansible párost fogjuk használni, pont emiatt a hordozhatóság miatt.

Elastic Beanstalk

PaaS-réteg az AWS-en belül

Automatikusan skálázza az erőforrásokat az alkalmazás igényeihez igazítva — úgy, hogy a háttérben más AWS-szolgáltatásokat (EC2, S3, SNS, Load Balancer, Auto Scaling Group) automatikusan paraméterez és hangol össze helyettünk.

A gyakorlatban

Ez a legjobb példa arra, hogy a "szolgáltatási szint" nem éles határvonal: az Elastic Beanstalk **IaaS-építőelemekből (EC2, ELB, ASG) rak össze nekünk egy PaaS-élményt.**

Lambda

esemény alapú számítási erőforrás

Egy-egy függvényt írunk meg, amit egy erősen izolált, könnyűsúlyú mikro-virtuális gépben (AWS Firecracker) futtat le a szolgáltatás — kizárólag esemény hatására.

Serverless: nem kell folyamatosan futó szerver — a függvény elindul, lefut, majd leáll

2020 óta Docker (OCI) image formájában is csomagolható — de a futtatókörnyezet ekkor is Firecracker, nem hagyományos Docker daemon

Idő alapú díjazás: csak a tényleges futásért fizetünk

Kombinálható más szolgáltatásokkal (SQS, SNS, DynamoDB) — egy alkalmazás jellemzően több Lambdából épül fel

Gyorsreferencia: melyik szolgáltatás mire való

SZOLGÁLTATÁS	KATEGÓRIA	EGY MONDATBAN
EC2	Feldolgozás	Virtuális szerver
S3	Tárolás	Objektumtárolás, automatikusan multi-AZ
EBS	Tárolás	Blokktár egy EC2-höz, egy zónához kötve
DynamoDB	Adatbázis	NoSQL, automatikusan skálázódó
SQS	Kommunikáció	Üzenetsor, komponensek szétválasztására
CloudWatch	Monitorozás	Metrikák, ez táplálja az Auto Scalinget
VPC	Hálózat	Saját, izolált virtuális hálózat
Auto Scaling	Skálázás	Automatikus EC2 darabszám-szabályozás
CloudFormation	IaC	AWS-natív infrastruktúra-leírás
Elastic Beanstalk	PaaS	Automatikusan összehangolt EC2+ELB+ASG
Lambda	Serverless	Esemény alapú, szerver nélküli függvény

3. BLOKK

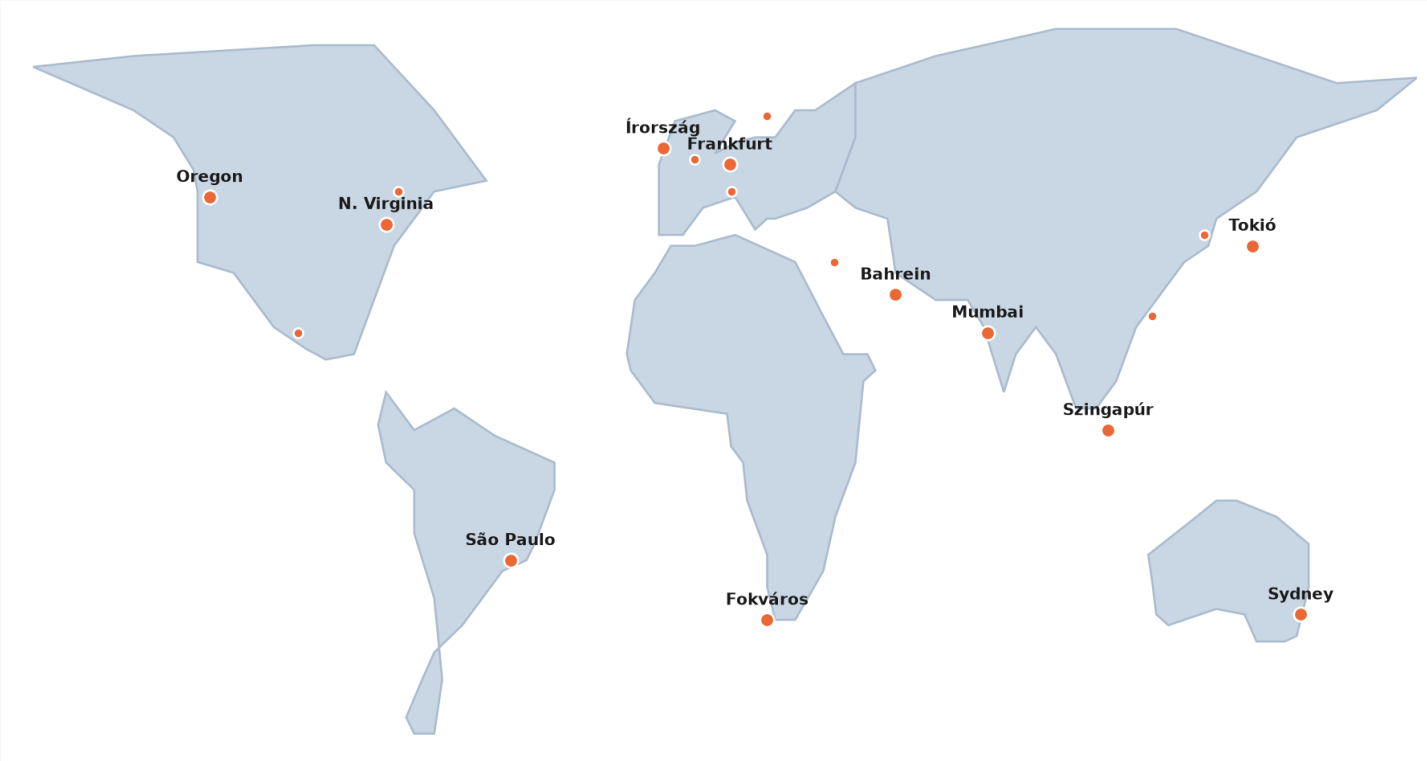
Az AWS globális infrastruktúrája

Régiók, rendelkezésre állási zónák, adatközpontok — és hogy ez a felépítés miért nem csak mérnöki érdekesség, hanem a napi architektúrai döntéseitek alapja.

GLOBÁLIS INFRASTRUKTÚRA

Mi az a régió?

Egy **régió** egy önálló földrajzi terület (pl. Írország, Frankfurt, Tokió), ahol az AWS saját adatközpontokat üzemeltet. Minden régió **teljesen független** a többitől — külön fizetés, külön elérhető szolgáltatások, és alapból **semmi erőforrás nem másolódik át** egyik régióból a másikba.



GLOBÁLIS INFRASTRUKTÚRA

Léptékben

39+
régió világszerte

124+
rendelkezésre állási zóna

750+
CloudFront edge helyszín

Ez a szám folyamatosan nő

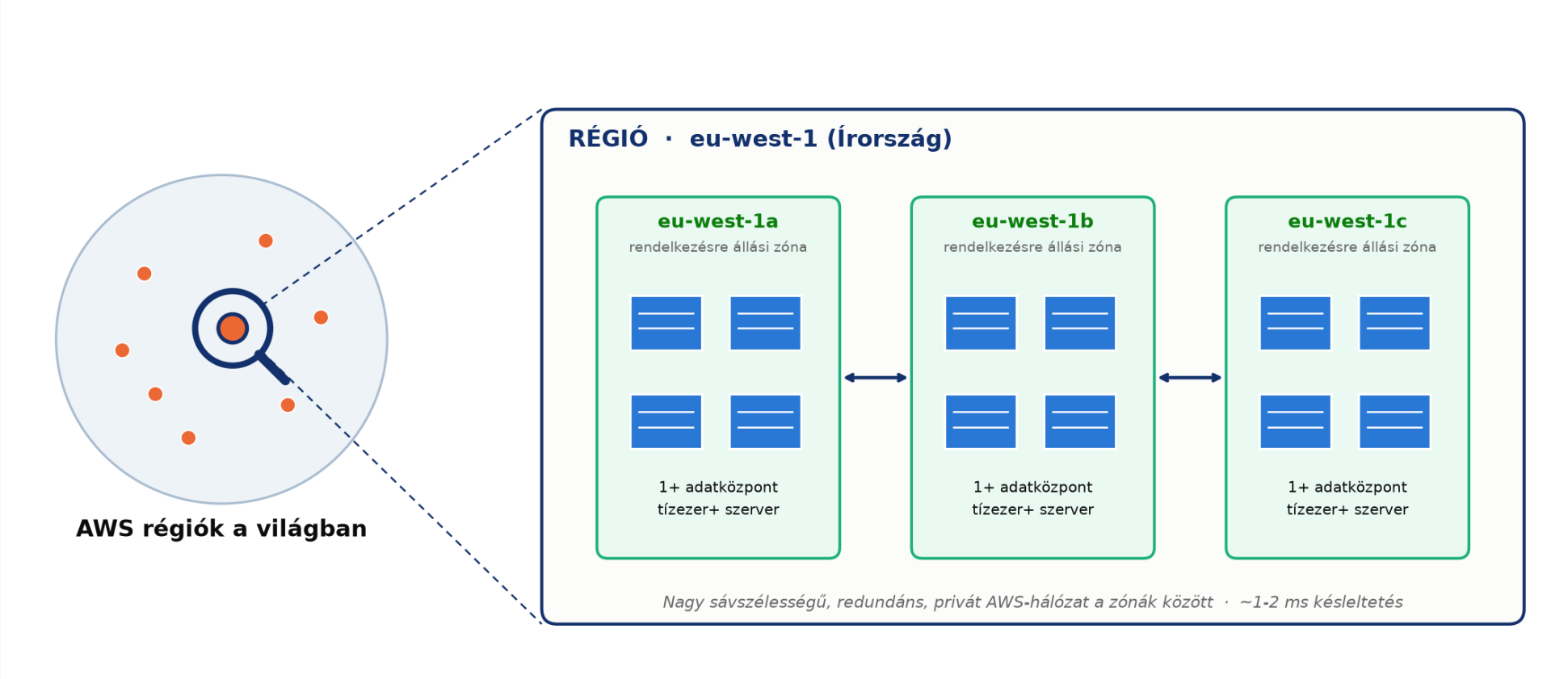
Az AWS rendszeresen jelent be új régiókat (jelenleg pl. Szaúd-Arábia és Chile van tervben) — a pontos, aktuális számot mindig az `aws.amazon.com/about-aws/global-infrastructure` oldalon érdemes ellenőrizni, ne memorizáljuk a mai számot vésett kőtáblaként.

Hogyan válasszunk régiót egy projekthez?

Három szempont dönt tipikusan: **késleltetés** (melyik van közel a felhasználóinkhoz), **ár** (régiónként eltér), és **jogi megfelelőség** (pl. EU-s személyes adat maradjon EU-s régióban — GDPR).

GLOBÁLIS INFRASTRUKTÚRA

Egy régió belülről: rendelkezésre állási zónák



Minden AWS régió legalább 3 rendelkezésre állási zónából (Availability Zone, AZ) épül fel.

GLOBÁLIS INFRASTRUKTÚRA

Miért nem elég egyetlen (nagy) adatközpont?

Mert egyetlen adatközpont egyetlen hibaforrás. A zónák tudatosan úgy vannak szétválasztva, hogy **egy helyi probléma ne tudjon átterjedni** a szomszéd zónára:

⚡ Független áramellátás

Külön tápvonal, külön generátorok — egy áramkimaradás nem viszi le a szomszéd zónát is.

❄️ Független hűtés

Egy hűtési meghibásodás csak a saját zónáját érinti.

🌊 Földrajzi távolság

Elég messze vannak egymástól ahhoz, hogy tűz, árvíz ne érintsen kettőt egyszerre — de elég közel a gyors hálózathoz.

🔌 Független hálózati bekötés

Külön internetkilépési pontok — egy szolgáltatói hiba nem izolálja az egész régiót.

A lényeg

Két zóna egyszerre történő kiesése statisztikailag elenyésző eséllyel fordul elő — **ezért ér annyit**, ha egy alkalmazást két zónában futtatunk egy helyett.

GLOBÁLIS INFRASTRUKTÚRA

Zónák egymás közt: gyors. Régiók egymás közt: csak a "cső".

✅ Egy régión belül (zóna ↔ zóna)

- Dedikált, redundáns, privát AWS-fényvezeték köti össze őket
- ~1-2 ms késleltetés, nagyságrendileg tíz Tbps sávszélesség
- Gyakorlatilag úgy viselkedik, mintha egy hálózaton lennének

⚠️ Két régió között

- A régiók a háttérben **redundáns hálózaton keresztül össze vannak kötve** (ez maga az AWS gerinchálózata, ill. Direct Connect ügyfeleknek privát link)
- De ez csak az **elérési utat** adja — **erőforrás, adat nem másolódik át rajta automatikusan**
- Minden régióközi adatmozgatás **explicit, tudatos döntés** marad — erről még lesz szó a mai órán

Gyakori kezdő tévhit

„A felhőben van, tehát biztosan mindenhol elérhető." **Nem.** Egy erőforrás pontosan ott létezik, ahova létrehoztuk — egy adott zónában, egy adott régióban. A régiók között van "út", de rajta semmi nem indul el magától.

GLOBÁLIS INFRASTRUKTÚRA

A teljes kép: régiókon túl

A régió/zóna páros a **számítási és tárolási** réteg. Emellett az AWS-nek van egy sokkal sűrűbb, világszerte szétszórt hálózata is — ez viszont más célt szolgál:

750+ helyszín

CloudFront Edge Location

Statikus tartalmat (kép, videó, cache) visz nagyon közel a felhasználóhoz — nem EC2/RDS fut itt, hanem egy CDN gyorsítótár.

helyi kiterjesztés

Local Zone

Egy régió "kihelyezett" mini-verziója, nagyvárosokhoz közel, extra alacsony késleltetéshez (pl. Los Angeles).

Ami a mai órán számít

A projektfeladatotokhoz elsősorban a **régió/zóna** réteg lesz releváns (EC2, RDS, VPC). Az Edge/Local Zone réteget csak azért érdemes ismerni, hogy tudjátok elhelyezni a térképen, ha később találkoztok vele.

A MAI ÓRA FŐ KÉRDÉSE

Mi replikálódik automatikusan — és mi nem?

Ugyanabban a régióban futtatva a szolgáltatásaitokat, néhány dolog "magától" túlél egy zóna-kiesést.
A többségük viszont nem — hacsak ti nem terveztek rá tudatosan.

AUTOMATIKUS VS. MANUÁLIS REDUNDANCIA

Amit NEKÜNK kell megoldanunk

1 EC2 = 1 zóna

EC2

Egy EC2 példány **pontosan egy** rendelkezésre állási zónában létezik. Ha az a zóna kiesik, az a példány elérhetetlenné válik — az AWS nem indítja újra automatikusan egy másik zónában.

Megoldás: Auto Scaling Group, amit több zónára állítunk be, + Load Balancer, ami tudja, melyik példány érhető el.

AZ-hez kötött

EBS

Egy EBS kötet **csak a saját zónáján belül** replikálódik (több másolat, de mind ugyanabban a zónában). Ha a zóna elérhetetlen, a kötet is az — átmenetileg vagy tartósan.

Megoldás: rendszeres snapshot (ami már régió-szinten, S3 mögött tárolódik), amiből másik zónában is új kötet indítható.

AUTOMATIKUS VS. MANUÁLIS REDUNDANCIA

Amit AZ AWS automatikusan megold

alapból multi-AZ

S3

Minden feltöltött objektum **automatikusan, kérés nélkül** több zóna között replikálódik ugyanazon a régión belül. Innen az emlegetett „11 kilences" (99,999999999%) tartósági garancia.

Mi a teendőnk? Semmi extra — ez a szolgáltatás alapviselkedése.

alapból multi-AZ

DynamoDB

Szintén automatikusan több zóna között tartja szinkronban az adatot a régión belül, teljesítményveszteség nélkül.

Mi a teendőnk? Semmi extra — ez is beépített garancia, nem kapcsoló, amit be kell kattintani.

Észrevehető minta

A **tárolási** szolgáltatások (S3, DynamoDB) jellemzően automatikusan multi-AZ-osak. A **számítási** szolgáltatás (EC2) soha nem az — mert az AWS nem tudja, hogy a bennük futó alkalmazásotoknak van-e állapota, amit nem szabad csak úgy máshol újraindítani.

AUTOMATIKUS VS. MANUÁLIS REDUNDANCIA

A köztes eset: RDS Multi-AZ

Az RDS egy külön kategória: alapból **ugyanolyan sebezhető, mint egy EC2** (egyetlen zónában fut) — de egyetlen kapcsolóval "félig-automatikussá" tehető.

alapértelmezett

Single-AZ RDS

Egy adatbázis-példány, egy zónában. Zóna-kiesés esetén az adatbázis elérhetetlen, amíg a zóna vissza nem áll.

bekapcsolható

Multi-AZ RDS

Az AWS **automatikusan** létrehoz és szinkronban tart egy készenléti (standby) másolatot egy másik zónában. Kiesés esetén az AWS **automatikusan átvált** a standby-ra — nekünk csak a bekapcsolás a dolgunk.

Fontos árnyalat

A Multi-AZ **nem** ugyanaz, mint egy Read Replica (ami teljesítmény-skálázásra való, külön kell kérni, és nem automatikus failoverhez készült). A kettőt gyakran összekeverik vizsgán is.

Összefoglalva: mit intéz az AWS, mit intézünk mi

EC2	MANUÁLIS	1 zóna. Multi-AZ redundanciához saját ASG + Load Balancer kell.
EBS	MANUÁLIS	Csak a saját zónáján belül replikálódik. Más zónába: snapshot kell.
RDS	FÉLIG-AUTOMATIKUS	Alapból 1 zóna — Multi-AZ opció bekapcsolásával AWS kezeli a failovert.
S3	AUTOMATIKUS	Minden objektum alapból több zóna között replikálódik.
DynamoDB	AUTOMATIKUS	Alapból több zóna között szinkronban tartott adat.
Régiók között	MINDIG MANUÁLIS	Egyetlen szolgáltatás sem lép át régióhatáron a ti kifejezett kérésetek nélkül.

GYAKORLATBAN

Forgatókönyv: kiesik az egyik zóna

Képzeljük el, hogy futtatjuk a féléves projektünket, és a régiónk egyik zónája (pl. `eu-west-1a`) átmenetileg elérhetetlenné válik. Mi történik?

EC2 (1 példány, nincs ASG)
Leáll. Amíg a zóna vissza nem tér, nem érhető el — nincs automatikus áthelyezés.

EC2 (ASG, 2+ zóna)
Túléli. A Load Balancer a másik zónában futó példányok felé irányítja a forgalmat, az ASG pótolja a hiányzó kapacitást.

EBS kötet abban a zónában
Elérhetetlen a zóna helyreállásáig, hacsak nincs friss snapshot másik zónában visszaállítva.

RDS (Multi-AZ)
Automatikus failover a standby-ra, jellemzően ~1-2 perces kieséssel.

S3 bucket
Észre sem vesszük. A többi zónában lévő másolatokból szolgál ki.

A másik régió (pl. Frankfurt)
Teljesen érintetlen — a régiók pontosan emiatt vannak elkülönítve egymástól.

GYAKORLATBAN

Hogyan építsünk erre tudatosan?

- **Állapotmentes komponensek (UI, API) → Auto Scaling Group, legalább 2 zónában**, Load Balancer mögött — ezt fogjuk a gyakorlaton is felépíteni
- **Adatbázis → RDS Multi-AZ**, ha valódi rendelkezésre állás a cél, nem csak teljesítmény
- **Fájlok, statikus tartalom → S3**, ahol a multi-AZ redundancia "ingyen" jár
- **EBS-en tárolt adat → rendszeres snapshot**, különösen, ha az adat nem tartható újra elő máshonnan
- **Katasztrófa-terv (Disaster Recovery) → tudatos, régiók közötti** döntés — csak akkor, ha a kockázat ezt indokolja

Ökölszabály

Zóna-szintű redundancia szinte mindig megéri (olcsó, gyors, sok helyen "ingyen" jár). Régió-szintű redundancia drága és bonyolult — csak akkor vezessük be, ha a projekt tényleg megköveteli.

GYAKORLATBAN

Ha mégis régiók között kell mozognunk

Mivel régiók között **semmi nem replikálódik automatikusan**, minden ilyen megoldás explicit beállítást igényel:

S3 Cross-Region Replication

Bekapcsolható funkció: minden új objektum automatikusan átmásolódik egy másik régióba is — de csak, ha külön beállítjuk.

AMI másolása

Egy EC2 image (AMI) manuálisan átmásolható másik régióba, hogy ott is tudjunk belőle szervert indítani.

RDS cross-region read replica

Az adatbázisról aszinkron másolat tartható fenn egy másik régióban — tipikusan olvasási terhelés elosztására vagy DR-célra.

Ne felejtsük

Ezekért mind **ti fizettek külön** (adatátvitel + tárolás a másik régióban is) — pont ezért nem alapértelmezett egyik sem.

4. BLOKK — A FELHŐ ÁRA

Vendor lock-in: a szabadság korlátai

Minden felhőszolgáltatónak vannak erősségei és gyengeségei. Ha egy cégnek a korábban jól bevált szolgáltatója hirtelen alulmarad egy területen, a váltás — vagy akár csak egy új funkció máshová telepítése — sokkal nehezebb, mint elsőre gondolnánk.

Mit jelent a vendor lock-in?

Azt, hogy a szolgáltatóváltás feloldásához **rengeteg idő és energia** kell — és lehet, hogy az új szolgáltatóhoz költöztetés épp úgy fájdalmas lesz a régi funkciókkal, mint amennyire kényelmes volt annak idején az újításokat az eredeti szolgáltatónál elhelyezni.

Van-e megoldás?

Léteznek próbálkozások az átjárhatóságra (Intercloud exchange koncepció), de a probléma messze nem egyszerű: nincs szabvány az adattárolásra vagy kommunikációra, és a szolgáltatók sem érdekeltek a saját, versenyelőnyt jelentő megoldásaik megosztásában.

Mit tehetünk mi, tervezőként?

Tudatosan mérlegeljük, mikor érdemes AWS-natív szolgáltatást (pl. DynamoDB, Lambda) választani a kényelemért, és mikor inkább szolgáltató-független megoldást (pl. konténerek, Terraform, sima PostgreSQL) a hordozhatóságért — ez már most, a projektfeladatnál is releváns döntés.

KÖVETKEZŐ ÓRA: ADATKÖZPONTOK BELSŐ HÁLÓZATA

Összefoglalás & kérdések

Öt szolgáltatási szint, tizenegy AWS-szolgáltatás, egy globális infrastruktúra és egy fő szabály: **zóna-szinten sok minden ingyen jár, régió-szinten semmi.** Két hét múlva a saját kezetekkel fogjátok ezt felépíteni.