Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Videó streaming szolgáltatások implementációja

Diplomaterv

Készítette
Piller Trisztán

Konzulens
Kövesdán Gábor

2025. március 11.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Piller Trisztán*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2025. március 11.

Piller Trisztán
hallgató

Kivonat

Az IT szakma meghatározó kihívása a magasan teljesítőképes, könnyen skálázódó és stabil infrastruktúra létesítése különféle üzleti célok megvalósítására. A hírközlés, a média és a szórakoztatatás iparágaiban is kiemelt figyelmet kap ez a kihívás.

Ezek a virágzó és feltörekvő iparágak folyamatosan igénylik az olyan szoftverfejlesztőket, akik ezekre az iparágakra specializáltan is folyamatosan képzik magukat, valamint mélyen ismerik a szakterület technológiáit.

Diplomatervem célja demonstrálni egy az Amazon Web Services platformján futó felhő alapú médiaszolgáltatás-rendszer alapos tervezésének, implementálásának, valamint tesztelésének folyamatát. A rendszer a videó streaming szolgáltatások területén nyújt weben elérhető megoldást, és a felhasználók számára lehetővé teszi, hogy saját időbeosztásuknak megfelelően férjenek hozzá videótartalmakhoz (Video-on-Demand), valami élő adásokat is tudjanak megtekinteni (live streaming).

A megoldás ismertetése során kiemelt fókuszt kap a komponensek közötti laza kapcsolás kialakítása, az IT biztonsági kockázatok kezelése, az konfigurációmenedzsment fenntarthatósága. Felhasználásra kerülnek modern webes technológiák és DevOps technikák, mint a konténerizáció, a serverless függvények, a CI/CD csővezetékek és az Infrastructure as Code.

Abstract

A key challenge for the IT profession is to build highly performant, easily scalable and stable infrastructure to support a variety of business goals. This challenge is also a major focus in Telecommunications, also in the Media & Entertainment industry.

These booming and emerging industries are in constant need of software developers who train themselves continuously for these industries and have a deep knowledge of the technologies in the field.

My thesis project aims to demonstrate the process of thoroughly designing, implementing and testing a cloud-based media delivery system running on the Amazon Web Services platform. The system will provide a web-based solution for video streaming services, allowing users to access Video-on-Demand content and live streams.

The solution will focus on the design of loose coupling between components, the management of IT security risks and the sustainability of configuration management. Modern web technologies and DevOps techniques such as containerisation, serverless functions, CI/CD pipelines and Infrastructure as Code will be used.

1. fejezet

Bevezetés

1.1. A témaválasztás indoklása

Arról írok, hogy miért választottam a témát: Érdemesnek tartom a szakterület vizsgálatát, a média streaming maga elég szűk területe a szoftveres világnak, mégis roppant nagy szaktudást lehet benne felvenni. Médiaszolgáltatások fejlesztésének infrastrukturális és szoftveres igényei jelentős kihívást jelentenek a szakma architektjei számára, és a témában való elmélyülés nagyban hozzájárulhat a szakmai tudásom bővítéséhez.

Sok szemszögből vizsgálható:

1.2. Kihívások a tématerületen

Ez a szekció arról fog szólni, hogy szoftver architekteknek milyen kihívásokkal kell szembenézniük, ha egy olyan kaliberű szórakoztató médiaszolgáltatás-rendszert kell megtervezniük, mint a Netflix vagy a Twitch.

Fel lesz sorolva jó pár dolog (felhasználói élmény szintre emelése, gyorsan fejlődő protokollok, különféle hardverekre/platformokra való optimalizálás, globális forgalmazás), kiemelve hogy én a biztonságra, a skálázhatóságra, az alaposan megtervezett és fenntartott infrastruktúrára helyezem a hangsúlyt.

2. fejezet

Technológiai áttekintés

A videó egy multimédiás eszköz auditív és mozgó vizuális információ tárolására, visszajátszására. Fontos felhasználási területei a média és a szórakoztató ipar. Ebben a fejezetben azon technológiák kerülnek vizsgálatra, amelyek ismerete nélkülözhetetlen a videó streaming megértéséhez, valamint az streaming megvalósításához.

2.1. A videó formátumai

Egy videó tartalmazhat különböző nyelvű hanganyagokat, mozgóképet, és egyéb metaadatokat – például feliratokat és miniatűr állóképeket – mind egy fájlban. Ezek közös tárolására konténerformátumokat alkalmazunk, amelyek megadják, az egyes adatfolyamok hogyan, milyen paraméterekkel, kódolással, tömörítéssel kerüljenek tárolásra, és hogyan kerülhetnek majd lejátszásra.

Szokásos összekeverni, de a konténerformátumoktól függetlenül a mozgóképkódolás és a hangkódolás különálló folyamatok. A kódolás egy algoritmus nyomán az adatot tömöríti, hogy a tárolás és a továbbítás hatékonyabb legyen.

2.1.1. Konténerformátumok

A legelterjedtebb és legszélesebb körben támogatott konténerformátum a Moving Picture Experts Group (MPEG) gondozásában specifikált MP4 – avagy a sztenderdben használt nevén: MPEG-4 Part 14 –, amely az MPEG-4 projekt részeként született 2001-ben.

Ugyancsak az MPEG gondozásában, az MPEG-2 projekt részeként született 1995-ben az MPEG Transport Stream (MPEG-TS) konténerformátum, amely elsősorban a digitális televíziózásban használatos, és így az internetes videó streaming során is. Felbontja a videóadatokat kisebb, fix hosszú adatcsomagokra, ezzel is elő-

készítve a tulajdonképpeni kis késleltetésű azonnali továbbítására a videóanyagnak a hálózaton keresztül.

További elterjedt konténerformátumok közé tartozik a Matroska Video (MKV), amely a hibatűréséről ismert; Apple vállalat által macOS-re és iOS-re optimalizálva fejlesztett QuickTime Movie (MOV) formátuma; valamint a WebM, egy szabad felhasználású webes kiszolgálásra optimalizálódott formátum, amelyet nagyobb jelentőségű webböngészők mind támogatnak. Régebbi, már kevésbé használt vagy kivezetett formátumok közé tartozik az Audio Video Interleave (AVI) és a Flash Video (FLV).

2.1.2. Mozgókép kódolási módszerei

Az MPEG-4 projekt keretében született az Advanced Video Coding (AVC) – avagy H.264 – kódolási szabvány mozgókép kódolására, amely 2004-ban vált elérhetővé. Továbbra is ez az egyik legelterjedtebb szabvány, a videó streamingben használt konténerformátumok is ezt a kódolást alkalmazzák.

Természetesen azóta több új szabvány is megjelent hasonlóan az MPEG projektjei alatt, mint például az 2013-ban megjelent High Efficiency Video Coding (HEVC) – avagy H.265 –, amely az AVC-től jobb tömörítést és jobb minőséget ígér, de a licencdíjak miatt nem vált annyira elterjedtté, mint az elődje.

Ezen modern problémák kiküszöbölésére terjedt el a VP8 és a VP9 – a WebM formátumnak tagjaként –, illetve az AV1 szabad felhasználású kódolási szabványok.

A szabványok konkrét megvalósításával (kodekek) nem foglalkozunk részletesebben, de egy szabad felhasználású és nyílt forráskódú megvalósítása a H.264-nek a x264, amelyet például az FFMpeg szoftvercsomag is használ videó kódolására és dekódolására.

2.1.3. Hang kódolási módszerei

Hanganyag kódolására több elterjedt szabványt is Ilyen az MPEG Audio Layer III – rövid nevén: MP3 – is 1991-ből, ezt 1997-ben az Advanced Audio Coding (AAC) szabvány váltotta le. Ezen szabványok az MPEG által kerültek kifejlesztésre, valamint mindkettő veszteséges tömörítést alkalmaz, azaz a kódoláson és dekódoláson átesett hanganyag minősége nem lesz azonos az eredeti hanganyaggal.

Egy másik, az előbbieknél jobb minőséget ígérő veszteséges tömörítéssel dolgozó kódolási szabvány Dolby AC-3 – ismertebb nevén: Dolby Digital.

2.2. Videó streaming kiszolgálása

A médiastreamelés egy olyan folyamat, amely során az médiaadatokat – mi esetünkben videóadatot – egy adott hálózati protokoll felett, egy adott konténerformátumban, adott kódolásssal továbbítjuk a végfelhasználók számára. Elsősorban az azonnali elérhetőségre összpontosít, azaz a lejátszásnak a lehető legkisebb késleltetéssel kell megtörténnie, kevésbé fontos a streaming során a minőség megtartása, mint ahogy az fontos lenne teljes médiumok egyben való letöltésekor.

A streaming során az adatfolyamba helyezés előtt a videóadatokat újrakódoljuk, majd kisebb adatcsomagokra – úgynevezett „packetekre" – bontjuk, és ezeket a csomagokat a hálózaton keresztül továbbítjuk a végfelhasználók felé. Az adatcsomagok önmagukban is értelmezhetőek, és a végfelhasználók lejátszó alkalmazásai képesek az adatcsomagokat a megfelelő sorrendben és időzítéssel lejátszani. A streaming könnyen reagál a lejátszás során ugrálásokra, előre- és visszatekerésre, mivel a videót nem kell teljes egészében letölteni a végfelhasználói eszközre, hanem a feldarabolt videócsomagot a lejátszás során továbbítjuk abban a pillanatban, amikor arra szükség lesz.

2.2.1. Az élő közvetítés és video-on-demand különbségei

Annak megfelelően, hogy az adat egésze mikor áll rendelkezésünkre, kettő fő streaming típust különböztetünk meg: az élő közvetítést (live streaming) és a video-on-demand (VOD) streaminget. A VOD esetében a videóadatokat előre rögzített formában tároljuk, és a végfelhasználók a videóadatokat a saját időbeosztásuknak megfelelően nézhetik meg. Az élő közvetítés esetében a videóadatokat valós időben továbbítjuk a végfelhasználók felé, és a végfelhasználók a videóadatokat a közvetítés során nézhetik meg.

Különbözik mindkettőnél, hogy milyen fizikai infrastrukturális tervezést kell végrehajtani a legjobb kiszolgálás érdekében. Élő adásoknál a késleltetés a középponti kihívás, mivel a videóadatokat a lehető leggyorsabban kell továbbítani a végfelhasználók felé, hogy a közvetítés valóban élőnek tűnjön, ehhez magas számítási teljesítmény szükséges. A VOD esetében a hálózati sávszélességből adódó problémák leküzdése a központi kihívás, mivel a videót a felhasználók sokkal nagyobb közönsége kívánja elérni, azt kiváló minőségben szeretné megtekinteni, ennek ellenére a globális sávszélesség korlátozott. Itt a caching és a tartalomterjesztés optimalizálása a kulcs, ekkor jönnek képbe a Content Delivery Networkök (CDN-ek), azaz a tartalomterjesztő hálózatok.

Üzleti szempontból a bevétel az élő adások során a közbeiktatott reklámokból származik főleg és a pay-per-view rendszerekből, míg a VOD esetében a közbeik-

tatott reklámokon kívül a felhasználók előfizetési díjából, tranzakcionális egyszeri vásárlásból – amennyiben a reklámokat kerülni szeretnék.

2.2.2. Adaptive Bitrate Streaming

A streamelést erősen befolyásoló tényező a hálózati feltételek változása, amelyek a videólejátszás minőségét és késleltetését is befolyásolják. Az Adaptive Bitrate Streaming (ABR) technológiával a videólejátszó alkalmazás képes a hálózati feltételek változására reagálni.

Az ABR technológiával a videóadatokat több különböző bitrátával kódoljuk a médiaszerveren, és a lejátszó alkalmazás a hálózati feltételeknek megfelelően választja ki a megfelelő bitrátájú videóadatokat a lejátszásra onnan.

TODO: KÉSZíTS EGY ÁBRAJZATOT AZ ABR MŰKÖDÉSÉRŐL!

2.3. Videó streaming hálózati protokolljai

Videó streamelésére – másképp fogalmazva: valamely korábban ismertetett formátumban tárolt videó adatfolyamba való illesztésére különböző hálózati protokollok palettája áll rendelkezésünkre.[1]

Ez a szekció az alkalmazás rétegben alkalmazott protokollok (Layer 7) közül vizsgál meg néhány streaming használatára elterjedt protokollt, megemlítve, milyen szállítási rétegű protokollokkal (Layer 4) tudnak együttműködni.

2.3.1. Real-Time Messaging Protocol

TODO

2.3.2. HTTP Live Streaming

TODO

2.3.3. Dynamic Adaptive Streaming over HTTP

TODO

2.3.4. WebRTC

A források és a streamet figyelők számának kardinalitása szempontjából ez a protokoll a legjobb választás, amennyiben a felhasználási célja az, hogy a felhasználók

közvetlenül egymással kommunikálhassanak, és nem szükséges közbeiktatni egy központi médiaszervert.

TODO: ez bullshitnak hangzik!

2.4. Az FFMpeg szoftvercsomag

TODO

2.5. Amazon Web Services

Bevezető az AWS szolgáltatások világába, röviden bemutatom, hogy épül fel egy AWS account, milyen szolgáltatáscsoportokat ajánl, mivel lehet korlátozni a hozzáférést, az erőforrások akár egymás közötti kommunikációjá irányítani.

2.5.1. AWS Elemental

TODO: MediaConvert

TODO: MediaLive

TODO: MediaPackage

2.5.2. Egyéb megoldások videó streamingre AWS-en

IVS, és saját konténerizált ffmpeg szerver miért nem.

2.5.3. Amazon CloudFront

Korábbi szekcióban már említettük a CDN-eket, mint a video-on-demand alapú streaming szolgáltatások egyik kulcsfontosságú elemeit. Az Amazon CloudFront egy globális CDN, amely lehetővé teszi a felhasználók számára, hogy a tartalmat hozzájuk közelebbi szervereken tárolt cache-ből töltsék le, ezáltal csökkentve a késleltetést, csökkentve a központi szerverek terhelését, és növelve a letöltési sebességet.

2.5.4. Amazon ECS

TODO

2.5.5. Amazon RDS

TODO

2.5.6. Orkesztrációs szolgáltatások

Lambda és EventBridge, IAM role-ok, hálózati infra.

2.6. A webes komponensek technológiái

Ebben a részben a konkrét AWS-beli PaaS és FaaS szolgáltatásokon futó több rétegű appok választott technológiáit mutatom be: Node.js, React, Prisma, Postgres.

2.6.1. TypeScript és JavaScript nyelvek

Miért választottam a TypeScriptet, miért jobb a JavaScriptnél, felhasználása a JavaScriptnek a Lambda és Cloudfront Function-ökben.

2.6.2. Node.js ökoszisztéma

Node.js futtatókörnyezet, NestJS keretrendszer, Prisma ORM, AWS SDK.

2.6.3. React

TODO: Mire jó a React: Statikus oldalak generálása (SSG), egyszerűbb JavaScript egy helyen kezelése, gyors build toolok, erre ott a Vite.js.

TODO: Sok könnyen integrálható könyvtár, amelyek segítségével gyorsan és hatékonyan lehet webes felületeket fejleszteni: React Hook Forms, React Query.

3. fejezet

A tervezett felhőarchitektúra

Gyors bevezető arról, hogy külön AWS accountba dolgoztam és felügyeltem a költségeim.

3.1. Követelmények

Funkcionális és nem-funkcionális követelmények, a rendszerrel szemben támasztott elvárások. Event driven architektúra (kiterjeszthetőség érdekében), skálázhatóság, biztonság, megbízhatóság.

3.2. Logikai felépítés

High-level leírása az AWS accounton belüli/datacenteren belüli hálózati kommunikáció rétegeinek.

3.3. Video-on-Demand kiszolgálás folyamata

High-level leírása a VoD kiszolgálás folyamatának, megemlítve, melyik választott szolgáltatások vesznek részt ebben.

3.4. Live streaming folyamata

High-level leírása a live streaming folyamatának, megemlítve, melyik választott szolgáltatások vesznek részt ebben.

3.5. Konfigurációmenedzsment

Terraform választásának indoklása. Kialakított Terragrunt module rendszer. GitHub actions a Terraform tervek előnézetére, OIDC felállítása az AWS role felvételére.

4. fejezet

Kliens közeli komponensek implementációja

4.1. A CDN és a hozzácsatolt erőforrások

A CDN és az originjeinek tárgyalása, melyik origin mire való. VPC Origin tárgyalása, annak haszna. A WAF szerepe, a felkapcsolt cert és WAF web ACL szerepe.

4.2. A statikus weboldal

Az S3 bucketee. A statikus website kiszolgálásának módja.

4.2.1. A React alkalmazás fejlesztése

Fejlesztés részletei. Nem annyira lényeges most ebben a dolgozatban, de néhány szép kihívást jelentő kidolgozott form és egyebeket be lehet itt mutatni.

4.2.2. A weboldal telepítésének CI/CD folyamata

GitHub Actions a smoke tesztekre, workflowk, a deploymentek. Értsd itt: S3 telepítés.

4.3. Média erőforrások objektumtárolói

Hogy lettek felkonfigurálva és miért az egyes S3 bucket-ok (bucket policy, CORS policy).

4.4. Elemental MediaLive és MediaPackage a live streamingben

Az Elemental stack részeinek felkonfigurálása, a MediaLive channel és a MediaPackage channel felépítése, a MediaPackage endpoint konfigurálása. Miként kerül kiszolgálásra, melyiket mire használom. OBS bekötésének módja.

5. fejezet

Szerver oldali folyamatok implementációi

5.1. A virtuális privát felhő komponensei

A VPC-beli (Virtual Private Cloud) subnetek, a security groupok, route táblák. A biztonság vizsgálata.

5.2. A Node.js alkalmazás fejlesztése

Fejlesztés részletei. Kitérve arra, hogy miképp könnyíti a munkát a Prisma, milyen egyéb szolgáltatások kerültek be, mi a felépítése a reponak, miért választottam ezt a stacket. S3 bucketba való mentése a videónak egy érdekes rész. Itt lehet szó az adatbázisbeli entitásokról is.

5.3. A konténerizált környezet

Leírás, hogy miért választottam a konténerizált környezetet, a konténerizálás előnyeit, hátrányait. Hogy használható ki a legjobban a konténerizáció az Application Load Balancer-rel együtt. Miképp kapcsoltam ezt a kettőt össze (ECS service, ALB).

5.3.1. A Node.js szerveralkalmazás ECS-en

Az ECS orkesztrációs toolsetjének kialakítása, a konténer rétegződés felépítése ECS-ben, a konténer registry (ECR) bekötése. Környezeti változók, portok, ALB-re való kötése. Miből állt a dockerizálás nekem (Dockerfile, registry, image build, push, networking).

Milyen IAM role-okat kellett feltenni rá, mikkel kommunikál kifelé, mi indokolta, hogy publikus subnetbe kerüljön. Hogy hív meg más külső rácsatlakozó erőforrásokat (S3 bucket, RDS instance, Lambda függvény, MediaLive channel).

5.3.2. A szerveralkalmazás CI/CD folyamatai

GitHub Actions a smoke tesztre, a deploymentre: Docker build és ECR-be telepítés.

5.4. Elemental MediaConvert felhasználása

Az Elemental MediaConvert API használata a Lambdából, illetve hogy hogy hívódik meg a Lambda, milyen triggerrel, milyen környezeti változókkal, milyen IAM roleal. Hogy kellett felkonfigurálni a MediaConvert job, hogy kellett magát a Lambdát felkonfigolni, hogy tudja is hívni.

6. fejezet

Tesztelés és mérés

6.1. Az infrastruktúra terhelése

Szimpla load tesztelés összeállításáról fogok itt értekezni, és a mérési eredmények kiértékelése.

6.2. Népszerű szolgáltatók metrikái

Keresni kell cikkeket Netflix blogban vagy valahol arról, a népszerű szolgáltatók milyen SLA-val, milyen statisztikával dolgoznak.

7. fejezet

Összegzés

Tanulságok, a rendszer működésének és fejlesztési élmények értékelése. Média streaming jövője saját meglátások szerint.

7.1. Továbbfejlesztés lehetőségei

Min lehetne javítani, mikroszolgáltatásos architektúra stb.

7.1.1. Vendor lock-in kezelése

Miként befolyásolja egy üzlet működését a vendor lock-in, és hogyan lehet ezt kezelni. Miképp lehetne a jövőben a vendor lock-in-t csökkenteni ebben a rendszerben.

7.2. Kitekintés: egyéb AWS szolgáltatások

Miért nem használtam az Aurorat RDS helyett, miért nem került sor Amplify Hosting alkalmazására a frontend és backend összekapcsolására és egy stackben való deployálására (skálázhatóságra való kiélezés miatt).

Irodalomjegyzék

[1] Ferenc Wettl – Gyula Mayer – Péter Szabó: *LaTeX kézikönyv*. 2004, Panem Könyvkiadó.