



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 24.

Tartalomjegyzék

Kivonat	i
Abstract	ii
1. Bevezetés	1
1.1. A téma ismertetése	1
1.2. A témaválasztás indoklása	1
1.3. Az első lépések	2
2. Elméleti háttér és irodalomkutatás	3
2.1. A videó formátumai	3
2.1.1. Konténerformátumok	3
2.1.2. Mozgóképek kódolási módszerei	4
2.1.3. Hang kódolási módszerei	4
2.2. Videó streaming kiszolgálása	5
2.2.1. Az élő közvetítés és video-on-demand különbségei	5
2.2.2. Adaptive Bitrate Streaming	7
2.3. Videó streaming hálózati protokolljai	7
2.3.1. Real-Time Messaging Protocol	8
2.3.2. HTTP Live Streaming	8
2.3.3. Dynamic Adaptive Streaming over HTTP	9
2.3.4. WebRTC	10
3. Felhasznált technológiák	11
3.1. Az FFMpeg szoftvercsomag	11
3.2. Amazon Web Services	12
3.2.1. AWS Elemental	12
3.2.2. Amazon IVS	12
3.2.3. Amazon VPC	13
3.2.4. Amazon ALB	13
3.2.5. Amazon S3	13

3.2.6.	Amazon CloudFront	13
3.2.7.	Amazon ECS	14
3.2.8.	Amazon RDS	14
3.2.9.	Kiegészítő AWS szolgáltatások	14
3.3.	A webes komponensek technológiái	15
3.3.1.	TypeScript és JavaScript nyelvek	15
3.3.2.	Node.js ökoszisztéma	15
3.3.3.	React	15
3.3.4.	Docker	15
3.3.5.	GitHub	15
4.	Követelmények	17
4.1.	Funkcionális követelmények	17
4.2.	Nem funkcionális követelmények	18
5.	A tervezett architektúra	19
5.1.	Logikai felépítés	19
5.1.1.	Összehasonlítás egy hasonló rendszerrel	19
5.2.	Video-on-Demand kiszolgálás folyamata	20
5.3.	Live streaming folyamata	20
5.4.	Konfigurációmenedzsment	20
6.	Kliensközeli komponensek implementációja	21
6.1.	A CDN és a hozzácsatolt erőforrások	21
6.2.	A statikus weboldal	21
6.2.1.	A React alkalmazás fejlesztése	21
6.2.2.	A weboldal telepítésének CI/CD folyamata	21
6.3.	Média erőforrások objektumtárolói	22
6.4.	Elemental MediaLive és MediaPackage a live streamingben	22
7.	Szerver oldali folyamatok implementációi	23
7.1.	A virtuális privát felhő komponensei	23
7.2.	A Node.js alkalmazás fejlesztése	23
7.3.	A konténerizált környezet	23
7.3.1.	A Node.js szerveralkalmazás ECS-en	24
7.3.2.	A szerveralkalmazás CI/CD folyamatai	24
7.4.	Elemental MediaConvert felhasználása	24
8.	Tesztelés és mérés	25
8.1.	Az infrastruktúra terhelése	25

8.2. Népszerű szolgáltatók metrikái	25
9. Összegzés	26
9.1. Továbbfejlesztés lehetőségei	26
9.1.1. Vendor lock-in jelensége	26
Irodalomjegyzék	27

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 24.

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 szórakoztatás iparágaiban is kiemelt figyelmet kap ez a kihívás.

Ezek a virágzó és feltörekvő iparágak folyamatosan igénylik az olyan szoftver-fejlesztőket, akik ezekre specializáltan is folyamatosan képzik magukat szakmailag, 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 (élő közvetítés).

A megoldás ismertetése során kiemelt fókuszot kap a komponensek közötti laza kapcsolat 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 téma ismertetése

A média és szórakoztatás egy óriási szeletét tölti ki az internet teljes forgalmának. A videó streaming szolgáltatók, mint például a Netflix, a Twitch, vagy a YouTube, több százmilliós – vagy akár milliárdos – aktív felhasználói bázissal rendelkeznek, az globálisan kiterjedt és folyamatos forgalom kiszolgálására a világ legnagyobb szerverparkjait üzemeltetik. Kiterjed a felhasználásuk az élet minden területére: az oktatásra, a szórakozásra, a mindennapi és munkahelyi kommunikációra, a hírek és információk terjesztésére, a kulturális és művészeti élmények megosztására, össze tud kötni embereket a világ minden tájáról.

A média streaming szolgáltatások fejlesztése és fenntartása komoly kihívások elé állítja a tervezőmérnököket, és a szakma legjobbjai a világ minden tájáról dolgoznak azon, hogy a felhasználói élményt folyamatosan javítsák, és a szolgáltatásokat a lehető legnagyobb számú felhasználó számára elérhetővé tegyék.

1.2. A témaválasztás indoklása

Végponttól végpontig tartó média streaming szolgáltatásoknak a fejlesztése és üzemeltetése számos komplex, informatikai területeket áthidaló kihívásokat hordoz magában. Ki kell tudni alakítani egy fenntartható, globális kiszolgálásra optimalizált és biztonságos hálózati infrastruktúrát. Folyamatosan kell tervezni a skálázhatóság biztosításával növekvő felhasználói bázissal. Ki kell tudni használni a legfrissebb hálózati protokollok adta lehetőségeket. Mélni kell a metrikákat a kitűnő felhasználói élmény biztosítása érdekében. Ki kell tudni szolgálni termédekféle végfelhasználói hardvert – például mobiltelefonok, különféle böngészők, AR- és VR-eszközök. A szabványok területén is tájékozottnak kell lennie a mérnöknek.

Már csak a kísérletezéssel felszedhető ismeretek is a piacon óriási előnyt jelentenek az ilyen irányba elköteleződő szakembereknek. Ezen indokok nyomán választottam magam is ezt a témát további vizsgálatra, eredményeim osztom meg jelen diplomamunkában.

1.3. Az első lépések

A téma bejárásának megkezdéseképp az Önálló laboratórium 2 című tárgy keretében egy olyan full-stack webes rendszer tervezését és implementációját vállaltam, amely lokálisan is futtatható szabad szoftvereket alkalmaz egy média streaming szolgáltatás alapjainak lefektetésére. Ennek köszönhetően megismerkedtem az FFMpeg szoftvercsomag videókonvertálási lehetőségeivel, videók kliens oldali lejátszásával HLS-protokollon továbbítva, valamint az NGINX webszerver RTMP-moduljának beállításával, amely lehetővé teszi a videók élő közvetítését.

Az elkészült rendszerből a diplomatervezés során alakítottam ki egy natív AWS-felhő alapú szoftverrendszert, némely komponens újrafelhasználásával előzőből – mint például a React alapú kliensoldali weboldal, a Docker-konfiguráció, illetve a szerveroldali kód CRUD-funkciói.

2. fejezet

Elméleti háttér és irodalomkutatás

Ebben a fejezetben kerül bemutatásra minden jelentősebb alapfogalom, valamint az azokhoz szorosan kapcsolódó technikák és folyamatok, amelyek ismerete nélkülözhetetlen a videó streaming megértéséhez, illetve annak szoftveres megvalósításához.

2.1. A videó formátumai

A videó egy multimédiás eszköz auditív és mozgó vizuális információ tárolására, visszajátszására. Fontos felhasználási területei a bevezetésben is ismertetett média- és a szórakoztató ipar.

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 előkészítve a tulajdonképpen kis késleltetésű azonnali továbbítására a videóanyagnak a hálózaton keresztül.

További elterjedt konténerformátumok közé tartozik a Matroska Video (MKV), amely a hibátűréséről 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 részletebben, 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 is több szabványt tudunk megvizsgálni. Ilyen az MPEG Audio Layer III – rövid nevén: MP3 – 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.

Főleg a mozivilágban használt kódolás a Dolby AC-3 – ismertebb nevén: Dolby Digital. Azonos bitrátánál is az előbbieknél jobb minőséget ígér. 2017-ben már lejárt a szabadalmi védelme, így azóta szabadon felhasználható. Egy másik, születése (2012) óta szabad felhasználású kódolás az Opus, amelyet a Skype és a Discord is

használ VoIP alapú kommunikációra, és az összes eddig említett kódolásnál jobb minőséget produkál.

Természetesen találkozhattunk tömörítetlen (pl.: WAV), illetve veszteségmentes kódolásokkal is (pl.: FLAC) is, azonban a streaming világában ezek nem használatosak, mivel a nagyobb fájl méretük meghaladná a sávszélesség és a tárhely korlátait. Az AC-3 és AAC szabványok közül szokás választani a videó streaming során, széles körben kompatibilisek mindenféle lejátszó eszközzel, míg az Opus még nem eléggé támogatott.

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 kódolással 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

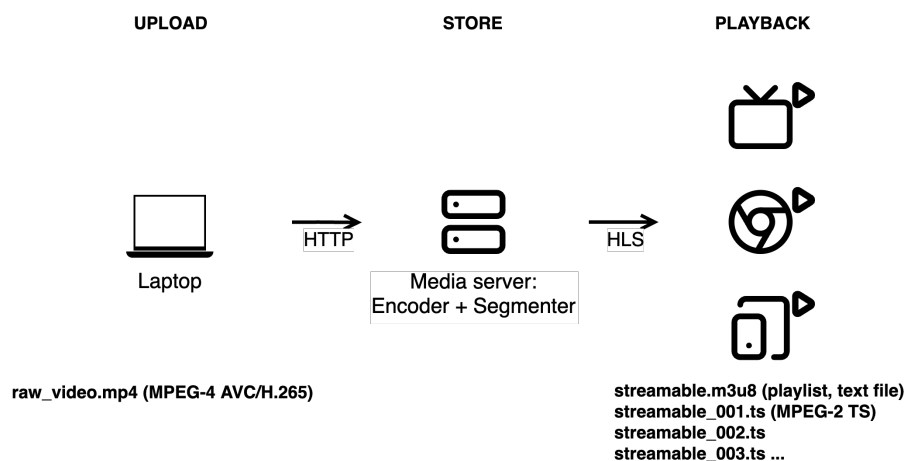
Annak megfelelően, hogy az adat egésze mikor áll rendelkezésünkre, kettő fő streaming típust különböztetünk meg: az élő közvetítést (live streaming) és a video-on-demand (VOD) streaminget. A VOD esetében a videóadatokat előre rögzített formában tároljuk, és a végfelhasználók a videóadatokat a saját időbeosztásuknak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

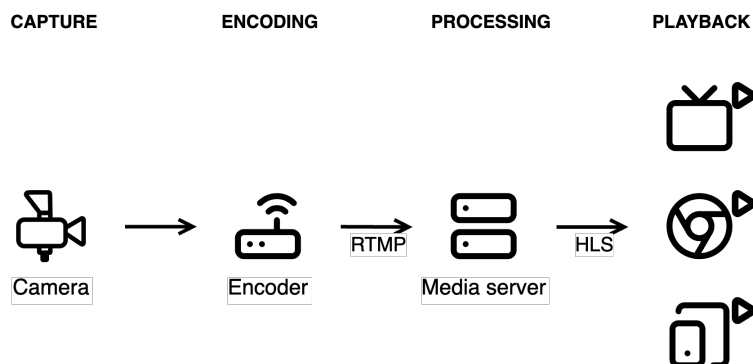
felhasználók felé, hogy a közvetítés valóban élőknek tűnjön, ehhez magas számítási teljesítmény szükséges. A VOD esetében a hálózati sávszélességből adódó problémák leküzdése a központi kihívás, mivel a videót a felhasználók sokkal nagyobb közönsége kívánja elérni, azt kiváló minőségben szeretné megtekinteni, 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 közbeiktatott reklámokon kívül a felhasználók előfizetési díjából, tranzakcionális egyszeri vásárlásból – amennyiben a reklámokat kerülni szeretnék.

Lentebb két ábrát (2.1. ábra és 2.2. ábra) láthatunk, amelyek absztrakt példákat mutatnak egy VOD-kiszolgálás és egy élő közvetítés résztvevő médiaeszközeire. A rajta feltüntetett protokollok és fájlnevek a későbbi alfejezetekben olvasása során érthetőek lesznek.



2.1. ábra. Példa egy VOD-kiszolgálás résztvevő eszközeire.

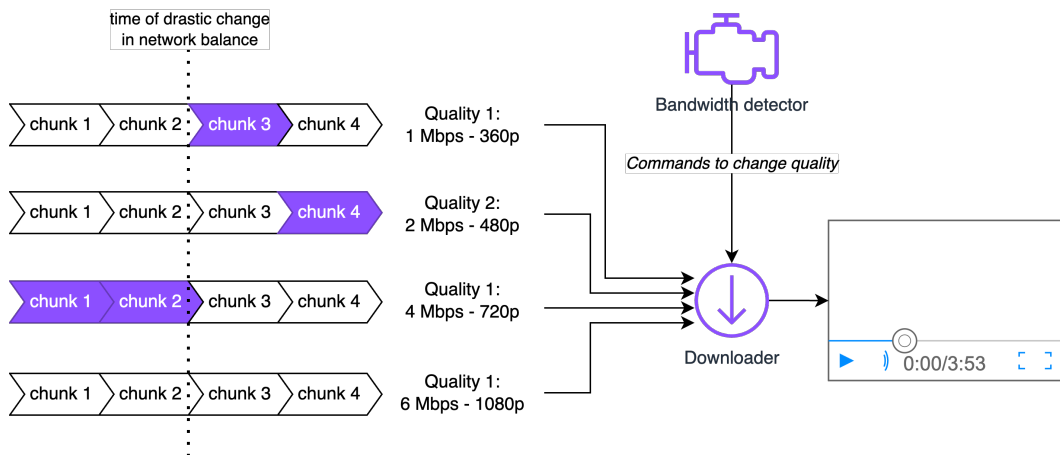


2.2. ábra. Példa egy élő közvetítés résztvevő eszközeire.

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) egy a hálózati kiszolgálás során alkalmazott technika, amely megoldást jelenthet erre a problémára.

Az ABR során a forrásvideót több különböző bitrátával dolgozó kodekekkel és különböző felbontással kódoljuk a médiaszerveren, majd lejátszáskor a lejátszó alkalmazás a hálózati feltételek változására reagálva, valamint a fogadó fél számítási kapacitásától függően valós időben választja ki a megfelelő videóstreamet ezek közül, onnan válogatja a packeteket (2.3. ábra).



2.3. ábra. Az Adaptive Bitrate Streaming működése.

Tiszta, hogy az ABR-t megvalósító rendszer előnyös mind a VOD, mind az élő közvetítés esetében, mivel a hálózati feltételek változása mindkét esetben előfordulhat, az automatikus streamváltogatás beavatkozás nélkül sokkal nagyobb Quality of Experience (röviden QoE, magyarul *az élmény minősége*) lehetőségét tudja biztosítani [2]. Az ABR-t alkalmazó rendszerek a videóadatokat több különböző bitrátával kódolják, ez természetesen a feldolgozási idejét a rendszereknek megnöveli, sőt élő közvetítés során egységnyi idő alatt jóval több számítási terhelést kell kibírnia a rendszernek.

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.

Néhány ismertebb streaming használatára kitalált protokoll létrejöttük időrendjében [6]:

- Real-Time Messaging Protocol (RTMP, 1996)
- Microsoft Smooth Streaming (MSS, 2008)
- Adobe’s HTTP Dynamic Streaming (HDS, 2009)
- HTTP Live Streaming (HLS, 2009)
- WebRTC (2011)
- Dynamic Adaptive Streaming over HTTP (DASH, 2012)

Jelen alfejezet ezen alkalmazásrétegben alkalmazott protokollok (Layer 7) közül vizsgálja meg a legelterjedtebb protokollokat, megemlítve, milyen szállítási rétegű protokollokkal (Layer 4) tudnak együttműködni.

2.3.1. Real-Time Messaging Protocol

A Real-Time Messaging Protocolt (RTMP) nevű protokollt még a Macromedia nevű cég fejlesztette ki, amely vállalatot később az Adobe felvásárolta. Az RTMP egy teljes szervertől kliensig tartó protokoll, amely a Flash Player és a Flash Media Server közötti kommunikációra lett kifejlesztve eredendően. Közvetlen dolgozik a TCP felett, ennek nincs köze a HTTP-hez. [6] Az RTMP-t még támogatja sok-sok platform, mert egész alacsony késleltetést lehet vele elérni, egészen alacsony „költségű”, a Twitch és a YouTube is támogatja, hogy RTMP pull üzemmódjában tudjunk ezeken a platformokon előadást fogadni, tehát szerver oldalon még használt, persze a Flash Player támogatásának 2020-as kivezetése miatt a protokoll használata kliens oldalon pedig visszaszorult.

2.3.2. HTTP Live Streaming

A HTTP-alapú streaming protokollok közül a legelterjedtebb a HTTP Live Streaming (HLS), amelyet az Apple fejlesztett ki 2009-ben. A HLS eredetileg az Apple jogvédett kereskedelmi protokolljaként indult, azóta viszont már szabad felhasználásúvá vált. Az Apple eszközök – macOS, iOS – alapértelmezetten támogatják ezt a protokollt, és a modern böngészők is támogatják a Media Source Extensions (MSE) API-n keresztül. Az HLS a HTTP felett dolgozik, egymástól független packetekre szedi a teljes kiszolgálandó videót, és az RTMP-hez képest ez így állapotmentes adatforgalmazást tud megvalósítani. [6]

HLS használata során H.264 formátumban kell a videóadatokat kódolni, a hangot AAC, MP3 vagy Dolby szabványokkal lehet kódolni. A konténerformátumot tekintve is kötött, MPEG-2 Transport Stream (MPEG-TS) formátumot használhatjuk, vagy pedig az MP4-et – fMP4 technikával, azaz *fragmented MP4-gyel*, ehhez pedig a Common Media Application Format (CMAF) konténerformátumra kell

átalakítani. [5] A darabokra szedést követően a packeteket egy lejátszási listában (.m3u8 kiterjesztésű szöveges indexfájl) tartja számon a szerver, amelyet a lejátszó alkalmazások letöltenek, és a lejátszás során a megfelelő sorrendben és időzítéssel lejátszák. [1] Lásd a példát egy 720p-s streamet leíró indexfájltra a 2.1. kódrészletben.

A HLS egy olyan protokoll, amely magában hordozza az ABR implementációját is, kliensoldalon modern Media Source Extensions funkcionalitást támogató böngészőkben elterjedt használni a *hls.js*¹ nevű JavaScript-könyvtárat, amely implementálja a HLS protokollt. A .m3u8 indexfájl definiálhat több másik ilyen indexfájlt is, amelyek a különböző bitrátájú videóstreameket képviselik (pl. egy *1080p.m3u8* fájl és egy *720p.m3u8* fájl írja le a playlistjét egy-egy bitrátájú streamnek).

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:4
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:4.000000,
skate_phantom_flex_4k_2112_720p1.ts
#EXTINF:4.000000,
skate_phantom_flex_4k_2112_720p2.ts
#EXTINF:4.000000,
skate_phantom_flex_4k_2112_720p3.ts
#EXTINF:4.000000,
skate_phantom_flex_4k_2112_720p4.ts
#EXTINF:4.000000,
skate_phantom_flex_4k_2112_720p5.ts
```

2.1. kódrészlet. Részlet egy .m3u8 indexfájlból.

2.3.3. Dynamic Adaptive Streaming over HTTP

A Dynamic Adaptive Streaming over HTTP – DASH vagy MPEG-DASH, tekintve a fejlesztője ennek is az MPEG csoport volt – egy 2012-ben szabványosított protokoll. Hasonlít a HLS-hez, HTTP felett forgalmaz, sorozatba illeszt egymástól független packeteket. Ezeket a szegmentált packeteket egy manifestfájlban tartja számon a szerver (Media Presentation Description, MPD-fájl). [7]

A HLS-hez képest ez a protokoll kodekagnosztikus, ami annyit jelent, hogy nem kötődik videókodekhez, használható H.264, H.265, akár VP9 is. [4] Igyekeztek ezzel a protokollal egyezményesíteni a tartalomvédelmet is, Common Encryptiont (CENC) használni titkosításra. Digitális jogkezelésre (Digital Rights Management,

¹<https://github.com/video-dev/hls.js>

DRM) is agnosztikus. Amióta a HLS támogatja az CMAF konténerformátumban való szállítást, azóta könnyen át lehet arról állni akár DASH protokollra is, ugyanis a DASH is CMAF-alapú. A DASH a HTTP/2 protokollt is támogatja, sőt HTTP/3-at is már UDP felett.

A HLS-hez hasonló elvekkel dolgozik a DASH is, illetve ez is egy ABR technológiájú protokoll. A DASH implementációjára JavaScript motorral rendelkező kliensekben – böngészők, mobiltelefonok stb. – a *dash.js*² könyvtárat használják legtöbb esetben.

2.3.4. WebRTC

A WebRTC (Web Real Time Communications) egy nyílt forráskódú projekt részeként alapult – támogatják kódbázisának fenntartását mind a Google, a Mozilla és az Opera csapatai is –, egy protokoll böngésző alapú valós idejű kommunikáció kialakítására. Ezt használja a Discord, a Google Chat és egyéb webes videóchat-alkalmazások. [6]

A források és a streamet figyelők számának kardinalitása szempontjából ez eltér az előzőekben ismertetettektől – amelyek egy forrás és több befogadóra optimalizált –, ez viszont egy-egy, vagyis peer-to-peer kardinalításra van kitalálva, emiatt a WebRTC a legjobb választás, amennyiben a felhasználási célja az, hogy a felhasználók közvetlenül egymással kommunikálhassanak, és nem szükséges közbeiktatni egy központi médiaszervert. Ezenkívül a WebRTC-t egyre szélesebb körben próbálják alkalmazni a videó streaming területén is one-to-many közvetítésre is. Szabad felhasználású kodekeket alkalmaz videó- és hangkódolás terén (pl.: Opus, VP9). [3]

²<https://dashjs.org/>

3. fejezet

Felhasznált technológiák

A fejezet célja, hogy bemutassa a videó streaming szolgáltatások implementációjához felhasznált konkrét szoftvereket, webes és felhőszolgáltatásokat, és az azok közötti kapcsolatokat.

3.1. Az FFMpeg szoftvercsomag

Az FFMpeg egy nyílt forráskódú és GPL-licenzelésű szoftvercsomag, amely képes videók és hangok kódolására, dekódolására, átalakítására (konvertálás), valamint streamelésre. Az FFMpeg a legtöbb operációs rendszeren elérhető, és számos különböző formátumot, modern kodekeket támogat. Az FFMpeg a videók és hangok kódolására és dekódolására szolgáló kodekeket tartalmazza, valamint számos különböző formátumot támogat, beleértve az MPEG-4, H.264, H.265, VP8, VP9, AV1, AAC, AC-3, Opus, és sok más formátumot. Folyamatosan frissen tartják a kodekeket, aktív fejlesztőbázissal rendelkeznek.

A lokális videókonvertálásra és streamelésre az FFMpeg-csomagból az azonos nevű ffmpeg parancssori interfészt használtam. Beépítettem a webszerver alkalmazásba egy külön szolgáltatásrészt, amely kihív a webszerver folyamatából és egy megfelelően felparaméterezett ffmpeg parancsot futtat a videókonvertálásra és streamelés megkezdésére aszinkron módon, a kimeneti fájlokat a megfelelő helyre menti.

A felhő alapú megoldásban már nem került felhasználásra az FFMpeg, mivel az AWS Elemental szolgáltatásokat használtam a videó kódolásra és streamelésre, viszont a szoftvercsomag közvetlen megismerése a videókonvertálás folyamatának megértését és a konvertálási folyamatok felparaméterezési lehetőségeinek mélyebb átlátását segítette.

3.2. Amazon Web Services

Az Amazon Web Services (AWS) a világ egyik legjobban elterjedt, legnagyobb szerverfarmjait fenntartó, nagy hírű vállalatok által is megbízható felhőszolgáltatója. Felhasználói számára számítási, hálózati, adattárolási célokat megvalósító szolgáltatások széles palettáját kínálja. Felhasználják az AWS-t a mesterséges intelligencia területén; valamint kiterjedt adatbázisok, adatfeldolgozó rendszerek építésére; megbízható és könnyen skálázható webes szoftverrendszerek kialakítására.

A felhasználók a szolgáltatásokhoz az AWS Management Console webes felületen keresztül, vagy az AWS Command Line Interface (AWS CLI) parancssori interfészen keresztül férhetnek hozzá. Az egyes felhasználók fel tudnak állítani maguknak egy vagy több AWS-fiókot, amelyek a számlázás és a jogosultságkezelés szempontjából elkülönülhetnek egymástól.

A fiókon belül lehetőséget kapunk granulás jogosultságkezelésre, azaz az egyes felhasználók, szolgáltatások, vagy szolgáltatásrészek számára különböző alacsony szintű jogosultságokat adhatunk meg.

Az AWS regionális adatközpontokat üzemeltet a világ számos pontján, amelyek közül a felhasználók választhatnak, hogy melyik adatközpontban szeretnének szolgáltatásokat futtatni.

A költségeket „pay-as-you-go” alapelv alapján számolják fel, azaz a felhasználók csak az általuk használt szolgáltatások számítási kapacitásáért, a tárhelyért, az adatközpontból kifelé történő hálózati forgalomért fizetnek.

3.2.1. AWS Elemental

TODO: MediaConvert

TODO: MediaLive

TODO: MediaPackage

3.2.2. Amazon IVS

TODO: Az Amazon Interactive Video Service (Amazon IVS)

TODO: kép innen <https://aws.amazon.com/blogs/media/awse-choosing-aws-live-streaming-solution-for-use-case/>

Ezen szolgáltatás ismertetése a megértést és a választás indoklását szolgálja későbbi fejezetekben, az Amazon IVS nem került felhasználásra a konkrét lefejlesztett rendszerben.

3.2.3. Amazon VPC

Az Amazon Virtual Private Cloud (Amazon VPC) egy virtuális hálózati környezet. Benne megvalósítható, hogy az AWS publikus felhőjén belül is privát és publikus saját hálózatokat hozzunk létre. Az Amazon VPC segítségével a felhasználók teljes kontrollt gyakorolhatnak a virtuális hálózati környezetük felett, beleértve a VPC-k alatti alhálózatok IP-tartományainak konfigurálását, útvonaltáblák kitöltését, a hálózati interfészek/portok korlátozásait, egyéb hálózati eszközök beillesztését (NAT Gateway-ek, Internet Gateway-ek, valamint Transit Gateway-ek), felülvizsgálható benne a hálózati teljesítmény.

3.2.4. Amazon ALB

Az Application Load Balancer (ALB) az Amazon Elastic Load Balancer (ELB) egy fajtája, ISO–OSI Layer 7 szinten, azaz alkalmazásrétegek szintjén működő terheléselosztó hálózati eszköz. Automatikusan skálázódik, lehetővé teszi a felhasználók számára, hogy egy vagy több szerver példány között egyenletesen eloszthassák a beérkező HTTP- és HTTPS-kéréseket. Az ALB képes a kéréseket a kérések fejlécei alapján vagy a kérések útvonala alapján elkülöníteni a forgalmat.

3.2.5. Amazon S3

Az Amazon Simple Storage Service (Amazon S3) egy objektumtároló szolgáltatás, amely lehetővé teszi a felhasználók számára, hogy nagy mennyiségű adatot tároljanak az AWS-felhőben „vödrökben” (bucketokban). Az objektum egy fájl és a fájl leíró metaadatok közösen. A vödör az objektumok tárolója.

3.2.6. Amazon CloudFront

Az előző fejezet egy szekciójában már említésre kerültek a CDN mint a video-on-demand alapú streaming szolgáltatások egyik kulcsfontosságú eleme. 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. Tartalmaink csoportosítására CloudFront „disztribúciókat” használunk.

A disztribúciók különböző URI-útvonalaikon akár különböző CDN-forrásokból – úgynevezett „originekből” – tudnak tartalmat kiszolgáltatni: ilyen origin lehet egy S3 vödör, AWS Elemental MediaPackage alapú élőadás-csatorna, Amazon Application Load Balancer (ALB) példány, vagy akár egy egyéni HTTP-szerver is saját doménnévvel.

3.2.7. Amazon ECS

Az Amazon Elastic Container Service (ECS) arra szolgál, hogy konténer alapú alkalmazásokat, szoftvercsomagokat futtathassunk a felhőszolgáltatónál. Az ECS segítségével a felhasználók könnyen futtathatnak és skálázhatnak konténereket anélkül, hogy a konténerek futtatásához szükséges infrastruktúra mélyén futó szervergépeket, valamint azok életciklusát, operációs rendszerének patchelését kellene kezelniük – ezek menedzselését az AWS Fargate motor veszi át, mi csupán a környezeti paramétereket kell felkonfigurálnunk az igényeinknek megfelelően.

Ilyen paraméterek a konténerek képei, a konténerek alapvető számítási erőforrásai (CPU-magok száma, memória mérete), a konténerek hálózati beállításai (porttovábbítások, alkalmazott Security Group), a konténerek naplózása (hova továbbítódik a futtatás során a naplók), és a konténerek hozzáférési jogosultságai az AWS felhőn belüli más szolgáltatásokhoz. Könnyedén kapcsolható össze Amazon ALB példánnyal.

Tipikusan alkalmazott az ECS párban az Amazon Elastic Container Registry (ECR) szolgáltatással, amely egy konténerképek tárolására szolgáló privát Docker Registry, amely lehetővé teszi a felhasználók számára, hogy a konténerek képeit biztonságosan tárolják és kezeljék az AWS felhőben.

3.2.8. Amazon RDS

Az Amazon Relational Database Service (RDS) egy relációs adatbázis szolgáltatás, amely segít, hogy könnyen és hatékonyan hozzassunk létre, üzemeltessünk és skálázzunk relációs adatbázisokat az AWS-felhőben. Az RDS támogatja a legnépszerűbb relációs adatbázis motorokat, mint például a PostgreSQL, MySQL, MariaDB, Oracle, és SQL Server. Képes automatikusan kezelni az adatbázisok frissítéseinek telepítését és a folyamatos biztonsági mentéseket.

3.2.9. Kiegészítő AWS szolgáltatások

A konténerek orkesztrációjának kiegészítésére számos könnyen élesíthető és ECS-hez integrálható szolgáltatás áll rendelkezésre az AWS-felhőben, amelyek közül a legelterjedtebbek az Amazon CloudWatch Logs, az AWS Lambda és a Amazon EventBridge.

Az Amazon CloudWatch Logs egy naplózó és monitorozó szolgáltatás, amely lehetővé teszi a felhasználók számára, hogy a konténerek futtatása során keletkező naplókat gyűjtsék, tárolják, és vizsgálják az ezekből származó metrikákat is akár.

Az AWS Lambda egy serverless Function-as-a-Service (FaaS) szolgáltatás, amely lehetővé teszi kód függvényyszerű futtatását anélkül, hogy szükség lenne a

szerverek vagy a futtatási környezet menedzselésére. A Lambda-függvény eseményekre reagálva kerül meghívásra, például HTTP kérésekre, adatbázis eseményekre, vagy más AWS-szolgáltatások eseményeire.

Ezzel kapcsolatban kerül a képbe az EventBridge, az AWS központi eseménykezelő szolgáltatása, amely lehetővé teszi az egyes AWS-felhőszolgáltatásokon futó alrendszerek közötti kommunikációt. Segítségével szűrhetünk eseményekre, azokat könnyen továbbíthatjuk az egyes AWS-szolgáltatások között, a célpontja egy EventBridge által elkapott eseménynek ennek megfelelően egy Lambda-függvény is lehet.

3.3. A webes komponensek technológiái

A különböző felhőszolgáltatásokon futó kódbázisokat elterjedt webes technológiák segítségével fejlesztettem. Ezen technológiák kerülnek bemutatásra a következő szekciókban.

3.3.1. TypeScript és JavaScript nyelvek

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

3.3.2. Node.js ökoszisztéma

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

3.3.3. 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.

3.3.4. Docker

TODO

3.3.5. GitHub

Szoftverrendszerek, alkalmazások fejlesztése során szinte elengedhetetlen a verziókezelés, amelynek segítségével a fejlesztők nyomonkövethetik a kódbázis változásait, visszaállíthatják az előző verziókat, és könnyen együtt tudnak dolgozni a kódon.

Erre a munkafolyamatra az egyik legelterjedtebb Source Code Management (SCM) eszköz a Git verziókezelő. A Git egy elosztott verziókezelő rendszer. Minden fejlesztő saját gépén tárolja a teljes kódbázisát, majd a módosításokat a felhőben lévő tárolóval szinkronizálhatja.

Az elterjedt Git felhőszolgáltatók közül a GitHub platformot választottam, és a GitHub Actions CI/CD szolgáltatását használtam a kódminőség ellenőrzésére, a kódépítés automatizálására, és a kód AWS-re való kiépítésére is.

4. fejezet

Követelmények

Egy nagyszabású, YouTube- vagy Netflix-szintű webes videóstreaming-szolgáltatás megvalósítása rendkívül összetett feladat, amely köré így kiterjedt technikai és üzleti követelményrendszert tudunk kialakítani. Egy ilyen rendszernek kezdetben biztosítania kell az alapvető funkciókat, amelyet hétköznapi webalkalmazások is megvalósítanak, mint például a videók lejátszása során felmerülő interakciók kezelése. Azonban ahogy a platformunk népszerűsége növekedhet, újabb és újabb infrastrukturális igények merülhetnek fel: ezek teszik ki a nem funkcionális követelményeket, amik kiterjednek például a tartalomterjesztés minőségére, a biztonsági felkészültségére a webalkalmazásnak.

A következőkben részletezem azokat a követelményeket, elvárásokat, amelyeket szem előtt tartottam a streaming szolgáltatás megtervezésekor és megvalósításakor.

4.1. Funkcionális követelmények

A streaming szolgáltatást alapvetően egy központosított kliensoldali webes felületen keresztül lehet elérni, azon keresztül tudják adminisztrátorok kezelni a tartalmat. Ennek a közvetlen frontoldali webes felületnek és a szolgáltatásainak a követelményeiket érdemesnek tartottam csoportokba szedni:

- **Felhasználókezelésre vonatkozó funkciócsoport:** TODO folyószöveg
- **Videóprojektek kezelésére vonatkozó funkciócsoport:** TODO folyószöveg
- **Élő közvetítés kezelésére vonatkozó funkciócsoport:** TODO folyószöveg

4.2. Nem funkcionális követelmények

A rendszerrel szemben támasztott nem funkcionális követelmények megállapításakor igyekeztem olyan dolgokra a hangsúlyt tenni, amelyek inkább számomra jelentenek kihívást, mivel nem terveztem a webalkalmazást úgy elkészíteni, hogy az valódi használatra készüljön – azaz valódi haszna legyen és legyenek élő felhasználói a nagyvilágból, csupán a kísérletnek volt része. Ezeket az elvárásokat a következőkben állapítottam meg:

- **Biztonság:** TODO folyószöveg, VPC, WAF, HTTPS, JWT
- **Hordozhatóság és könnyű karbantarthatóság:** TODO folyószöveg, konténerizáció, CI/CD, frissítések, kódba zárt konfiguráció, eseményvezérelt architektúra a webalkalmazás köré
- **Elaszticitás:** TODO folyószöveg, konténerizáció, auto-scaling, CDN

5. fejezet

A tervezett architektúra

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

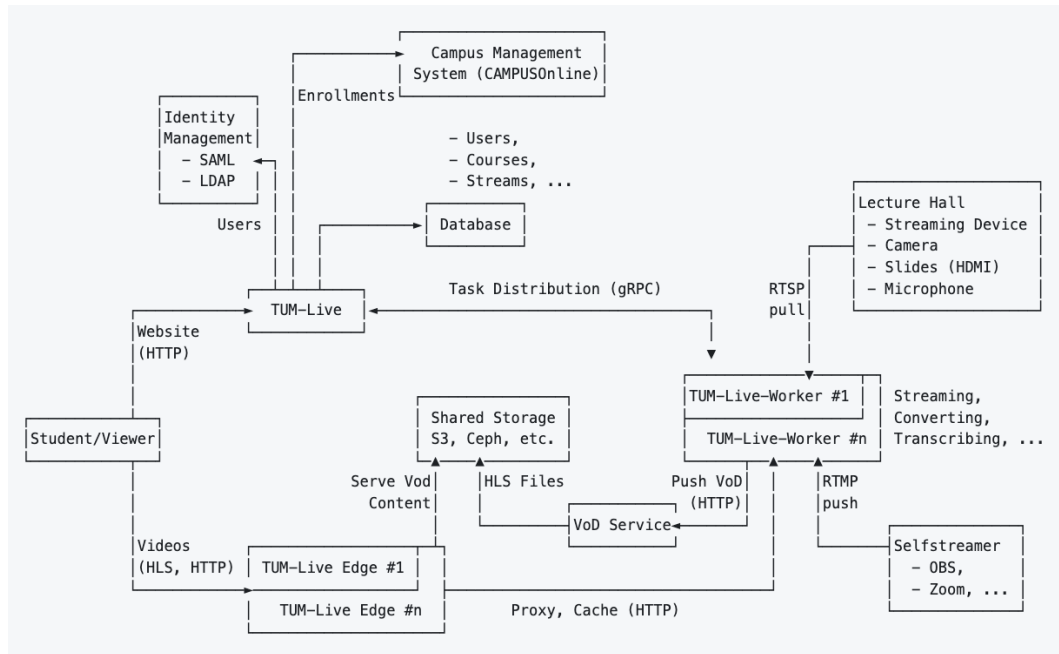
5.1. Logikai felépítés

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

5.1.1. Összehasonlítás egy hasonló rendszerrel

A tervek igazolásához segítségül kerestem az interneten nyílt forráskódú hasonló megoldásokat is. Megtaláltam a Technische Universität München (TUM) egy hallgatói csoportja, a TUM-Dev által fejlesztett az egyetemen is használt VoD és live streaming szolgáltatását, a GoCastot¹. Ez a rendszer önállóan hosztolható szoftvereket komponál össze, nem felhőnatív. A rendszerben hasonló absztrakt terveket lehet megfigyelni az én megoldásaimhoz (5.1. ábra), ugyanígy HLS-sel szolgálják ki a tartalmakat (lásd *TUM-Live Edge* példányok), viszont a live streamingre saját több portos workereket alkalmaznak (lásd *TUM-Live-Worker* példányok), lehetőség ad viszont saját streamerből RTMP-n keresztül feltölteni élő közvetítést, ahogy én is megvalósítom a saját megoldásomban. Külön mikroszolgáltatás biztosítja a live és VOD streamingen kívüli funkciókat, a TUM-Live. Ez a megoldás nem teszi lehetővé, hogy a rendszeren kívül készült videót lehessen feltölteni és VOD-ként elérhetővé tenni rajta keresztül.

¹<https://github.com/TUM-Dev/gocast>



5.1. ábra. A GoCast architektúrája a dokumentációból.

5.2. Video-on-Demand kiszolgálás folyamata

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

5.3. Live streaming folyamata

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

5.4. Konfigurációmenedzsment

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

6. fejezet

Kliensközeli komponensek implementációja

A következőkben részletezem a klienseket kiszolgáló infrastrukturális komponensek konfigurációját, valamint a legfelső megjelenítési réteg szoftveres komponenseinek implementációját. A forgalom először a CDN-nel ütközik, amelyen keresztül lesznek elérhetőek a statikus weboldal erőforrásai, valamint a média-erőforrások csatornái.

6.1. A CDN és a hozzácsatolt erőforrások

TODO: 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.

6.2. A statikus weboldal

TODO: Az S3 bucket. A statikus website kiszolgálásának módja.

6.2.1. A React alkalmazás fejlesztése

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

6.2.2. A weboldal telepítésének CI/CD folyamata

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

6.3. Média erőforrások objektumtárolói

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

6.4. Elemental MediaLive és MediaPackage a live streamingben

TODO: 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.

7. fejezet

Szerver oldali folyamatok implementációi

A megjelenítési réteg alatt található szerveroldali folyamatok implementációja során a kiszolgáló infrastruktúra kialakítására, a Node.js alkalmazás fejlesztésére, a konténerizált környezet kialakítására, valamint a videófeldolgozásra fókuszálunk ebben a fejezetben.

7.1. A virtuális privát felhő komponensei

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

7.2. A Node.js alkalmazás fejlesztése

TODO: 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.

7.3. A konténerizált környezet

TODO: 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).

7.3.1. A Node.js szerveralkalmazás ECS-en

TODO: 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).

TODO: 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).

7.3.2. A szerveralkalmazás CI/CD folyamatai

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

7.4. Elemental MediaConvert felhasználása

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

8. fejezet

Tesztelés és mérés

A média streaming szolgáltatások fejlesztése és üzemeltetése során a mérési eredmények alapján lehet a legjobban optimalizálni a rendszert, kiszámolni a költségeket, és a felhasználói élményt folyamatosan javítani. Jelen fejezetben megvizsgáljuk közelebbről egyszerűbb tesztesetek segítségével, milyen eredményekre lehet számítani egy AWS-felhő alapú streaming szolgáltatás fejlesztése után.

8.1. Az infrastruktúra terhelése

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

8.2. Népszerű szolgáltatók metrikái

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

9. fejezet

Összegzés

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

9.1. Továbbfejlesztés lehetőségei

TODO: Min lehetne javítani, mikroszolgáltatásos architektúra stb.

9.1.1. Vendor lock-in jelensége

TODO: 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. S3 helyett Ceph, konténert kiemelni, akár K8s-re felkészíteni, hogy ne ECS-től függjön.

Irodalomjegyzék

- [1] Apple Inc.: HTTP Live Streaming | Apple Developer Documentation. <https://developer.apple.com/documentation/http-live-streaming>. [Hozzáférés dátuma: 2025-03-24].
- [2] Abdelhak Bentaleb – Bayan Taani – Ali C. Begen – Christian Timmerer – Roger Zimmermann: A survey on bitrate adaptation schemes for streaming media over http. *IEEE Communications Surveys & Tutorials*, 21. évf. (2019) 1. sz., 562–585. p. <https://ieeexplore.ieee.org/document/8424813>.
- [3] Frequent Questions | WebRTC. <https://webrtc.github.io/webrtc-org/faq/>. [Hozzáférés dátuma: 2025-03-24].
- [4] ISO Central Secretary: Information technology – Dynamic adaptive streaming over HTTP (DASH) – part 1: Media presentation description and segment formats. ISO/IEC 23009-1:2022. Standard, Geneva, CH, 2022, International Organization for Standardization / International Electrotechnical Commission. URL <https://www.iso.org/standard/83314.html>.
- [5] Sydney Roy: What is CMAF? Jelentés, 2022, Wowza Media Systems, LLC. <https://www.wowza.com/blog/what-is-cmaf>.
- [6] Servers.com: The history of streaming told through protocols, 2023. https://www.servers.com/docs/whitepaper/history_of_streaming_told_through_protocols.pdf.
- [7] Thomas Stockhammer: Dynamic adaptive streaming over http: Standards and design principles. 2011. 02, 133–144. p. <https://dl.acm.org/doi/10.1145/1943552.1943572>.