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 23.

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 23.

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 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

Véngponttól 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. Mérni kell a metrikákat a kitűnő felhasználói élmény biztosítása érdekében. Ki kell tudni szolgálni termérdekféle végfelhasználói hardvert – például mobiltelefonok, különféle böngészők, AR- és VR-eszközök. A szabványok területén is tájékozottnak kell lennie a mérnököknek.

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 szabványban 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 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 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 fájlméretük 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 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 vég-

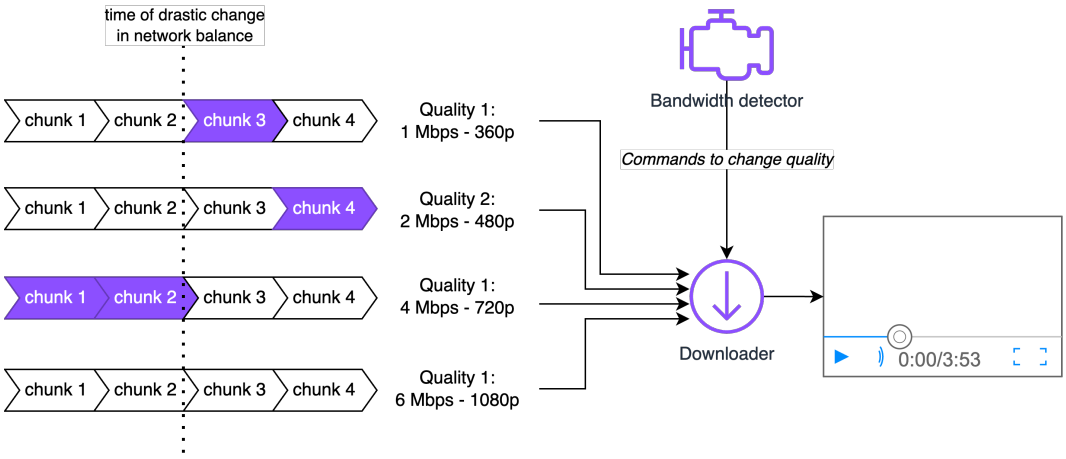
felhasználók felé, hogy a közvetítés valóban élőnek tűnjön, ehhez magas számítási teljesítmény szükséges. A VOD esetében a hálózati sávszélességből adódó problémák leküzdése a központi kihívás, mivel a videót a felhasználók sokkal nagyobb közönsége kívánja elérni, azt kiváló minőségben szeretné megtekinteni, 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 közbeiktatott 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) 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 (2.1. ábra).



2.1. ábra. Az Adaptive Bitrate Streaming működése.

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.

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: még egy keveset ide, hogy lássa az olvasó később, hogy miért nem lesz releváns a mi választásunkban az implementáció során.

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-licenszelésű 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 rendelkezik.

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 videókódolásra 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 interfészén 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 granuláris 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 fájlt leíró metaadatok közösen. A vödör az objektumok tárolója.

3.2.6. Amazon CloudFront

Az előző fejezet egy szekciójában már említetésre 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 kiszolgálni: 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 felkonfiguráljuk 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 továbbítódjanak 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 hozhassunk 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 függvényszerű 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-felhőszolgátatásokon 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 kiélesítésére is.

4. fejezet

Követelmények

TODO: Mire kell képes legyen egy Youtube szintű app, erről rövid duma.

4.1. Funkcionális követelmények

TODO: részleteiben

4.2. Nem funkcionális követelmények

TODO: a rendszerrel szemben támasztott elvárások. Event driven architektúra (kiterjeszthetőség érdekében), skálázhatóság, biztonság, megbízhatóság.

5. fejezet

A tervezett felhőarchitektúra

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

5.1. Logikai felépítés

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

5.2. Video-on-Demand kiszolgálás folyamata

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

5.3. Live streaming folyamata

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

5.4. Konfigurációmenedzsment

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

6. fejezet

Kliens közeli komponensek implementációja

6.1. 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.

6.2. A statikus weboldal

Az S3 bucketee. A statikus website kiszolgálásának módja.

6.2.1. 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.

6.2.2. A weboldal telepítésének CI/CD folyamata

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

6.3. Média erőforrások objektumtárolói

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

6.4. 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.

7. fejezet

Szerver oldali folyamatok implementációi

7.1. A virtuális privát felhő komponensei

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

7.2. 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.

7.3. 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).

7.3.1. 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).

7.3.2. A szerveralkalmazás CI/CD folyamatai

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

7.4. 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.

8. fejezet

Tesztelés és mérés

8.1. Az infrastruktúra terhelése

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

8.2. Népszerű szolgáltatók metrikái

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

9. fejezet

Összegzés

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

9.1. Továbbfejlesztés lehetőségei

Min lehetne javítani, mikroszolgáltatásos architektúra stb.

9.1.1. 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.

9.2. 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