

# PRAIRIE DEV CON

WEB | DEV | CLOUD | AI



EXCELLENCE IN RECRUITMENT



# DIY Video Streaming



NORTHFIELD

Mike Menzies, Senior DevOps Engineer  
Paul Giles, Principal Engineer

September 2024

Hello everyone, welcome to the Do-It-Yourself Video Streaming presentation here at PraireDevCon

My name is Paul Giles and I work with Northfield IT, a consultancy here in Winnipeg. I met Northfield while working for one of their US-based clients and while working with them, grew determined to join them. Luckily for me, in 2018, they let me!

Mike?

Hi everyone, my name is Mike Menzies, I've been with Northfield now for... jeeze, almost 11 years. I've been in a variety of different roles with the company over those years, but currently I'm a Senior DevOps Engineer.

*<Next slide>*



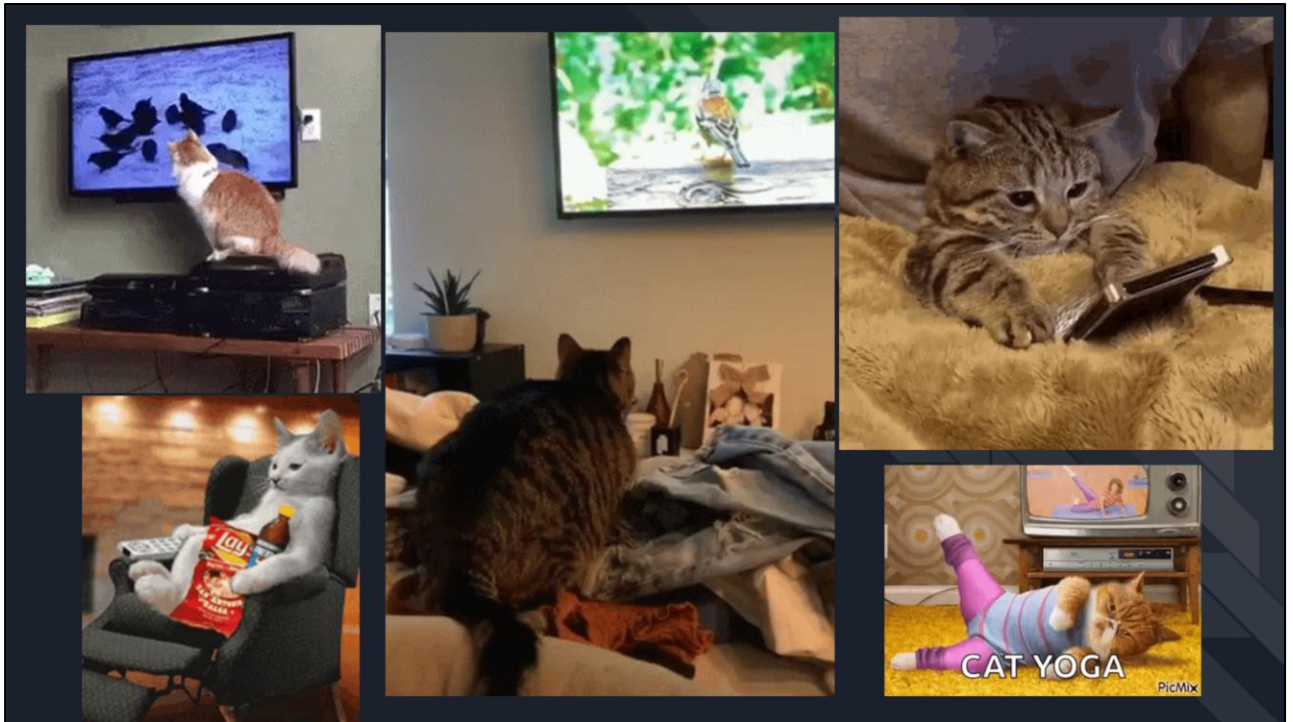
# Agenda

What do we have in store today?

Most IT professionals don't know about the ins and outs of video streaming. We'll get you in-the-know today. It will be 2 parts educational, 1 part demonstration and 1 part success story. If you have ever endeavoured to present video to your users, followers or fans, we're aiming to help you find the signal in the noise.

It's going to be the best one-hour investments in video ever!

*<Next Slide - Shows a cat video in the background - speaker does not acknowledge it. Hopefully audience giggles>*



And it doesn't even come with cat videos!

<Next Slide>

# Learning Objectives

1. What is adaptive bitrate and why is it important?
2. How are videos stored and delivered?
3. Should you use HLS? And what the hell is HLS anyways?
4. Where does the video playback magic happen?

But it does come with some goals! I am going to sow some seeds here by teasing some stuff for you to listen for. If you take the answers to these 4 questions away today, you'll be in good shape

I'll gloss over them here with the hope it gives you something to listen for throughout the presentation.

First...<READ SLIDE 1-4>

Now, let me stop here and say if you don't know what any of these terms are, you're not alone and you're in the right place.

<Next Slide>

# Backstory

As I said earlier we're here to educate, but also to share our success story. Here's the lowdown.

*<Next Slide>*

# Connected Ship



## Efficient Content Delivery

Our team ensures that shore-based content is delivered to ships quickly and efficiently

Centralized caching strategies improve HTTP traffic performance by reducing latency and bandwidth usage

Video files are no exception

We work with Royal Caribbean, as you may know, they operate a global fleet of cruise ships. Our team's responsibility is to ensure that shore-based content is delivered to ships quickly and efficiently.

We're a consultancy, but we're also a systems integrator. Our team is known for providing a highly available, highly performant and easy-to-use centralized caching solution for a ship's services and systems. Our solution handles diverse media formats, ensuring smooth delivery across web and mobile platforms. Our expertise in the content delivery arena is about to be called to action once again.

*<Next Slide>*

## A project is born, a legend is not



Let's add videos!

Videos enhance user engagement by 70%.



Enlisted a legend... in their own mind

Use the media titan solution

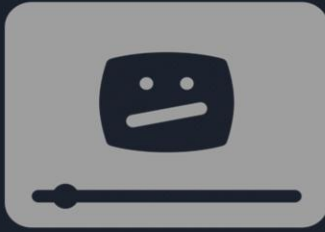
When our client initiated a project to supe-up their mobile apps with videos, understandably they didn't think our team was capable of doing it alone. I didn't blame them—sure, we were caching experts, but we were not experts in video delivery. But, knowing that video content, like other content, would arrive to the ships from the shore, we were invited to join the project team so that we could prepare to serve “streaming video” that would be arriving from shore.

Then, they went to the marketplace and enlisted well-known legends-of-media titans that specializes in media at scale.

<Next Slide>



# Delivery Challenges



## Streaming setback

So-called media titan lacks pre-production testing capability.



## Tight deadlines

Failed acceptance test leaves little time to for a new solution



## Progressive downloads fail user testing

Progressive downloads take too long to play, proving the need for speed- stream

As that engagement underwent the requisite sales, budgetary and provisioning process, the delivery team encountered some challenges.

It turned out, the streaming service partner, lacked lower environment testing capability. I guess this could be its own lesson: If you offer a product, teams that want to use it might want to try it in a non-prod environment! But I digress. The inability to integrate with the media titan's streaming solution put the delivery at risk and with a looming deadline, the team's engineers did what they knew how to do: serve an MP4 as a progressive download. Again, let me say again, if you don't know what progressive downloads are, you're not alone. We'll learn about this technique and in just a bit.

So, now we have a team lacking in know-how, with little time to deliver, falling back on what they knew could be done in time: push an MP4 file to a webserver and let the client player deal with it, err I mean gracefully play it.

In about six weeks, this solution makes it way to user acceptance testing where it quickly fails to meet video experience requirements. i.e. when requesting to play these long, high-quality videos, in **a realistic shipboard environment**, it would take too long to start and buffer during playback. In the live environment, it simply took too long to download enough of the MP4 ensure smooth playback without interruptions. Not to mention, not being able to to interact with the video, like forwarding or rewinding.

<Next Slide>

# Renewed Ambition for Streaming



Testing results prove the need for speed, er streaming



An opportunity emerges for some scrappy insiders

This result made it clear that efficiency was a requirement. While the delivery team looked for answers, a member of our team took this opportunity to formulate an alternative to the media titan's offering that could be integrated with little to no effort from the other teams, tested, delivered quickly and all at a fraction (cough free) of the cost. More on that to come.

This is the end of the backstory.

<Next Slide>

## Lingo Gumbo



You've heard me toss around some terminology, like "progressive", "MP4", "buffering", and heck, even "streaming". What do all these terms even mean exactly...technically?

We had a lot to learn. In terms of serving large MP4s, we may have guessed what not to, but we were still pretty green when it came to knowing what to do. What we learned is that we only needed to master a few concepts and technologies to be successful.

So before I share the rest of our story with you, we want to give you a compact, introductory level course of fundamentals that we wished we'd had.

*<Fade in Next Slide>*



Lingo Bingo

**HLS**  
transcoding  
**VOD**

**ABR**  
**Codec**

This is the first step toward finding the signal in the noise.

*<Next Slide>*

# Video Streaming 101

Paul: Now to educate us, I'm passing the mic to Mike.

Mike?

# Containers & Codecs

## Containers

MP4

MOV

AVI

MKV

## Codecs

H.264

H.265

VP9

AV1

MP4

H.264

AAC

Okay, so we're going to start by getting some low level terminology out of the way. When we think of digital video files, we typically think of the file format. These are referred to as containers. MP4 is by far the most common. You also have Apple's MOV. For the pirates out there, you'll definitely recognize AVI and MKV.

All containers then support a set of codecs. This is the technology that is responsible for compression and decompression, also known as encoding and decoding. I'm sure most you have dealt with image compression at some point, same thing.. In fact, the notion of containers and codecs is universal across all digital media. Audio, video, and images.

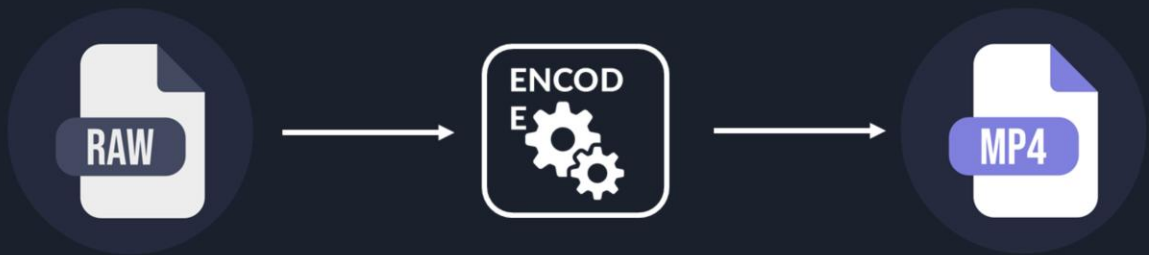
So at its simplest, a video file is a container wrapping a codec, wrapping two codecs actually, one for video and one for audio.

# Transcoding vs. Encoding

Let's also quickly talk about the distinction between transcoding and encoding.

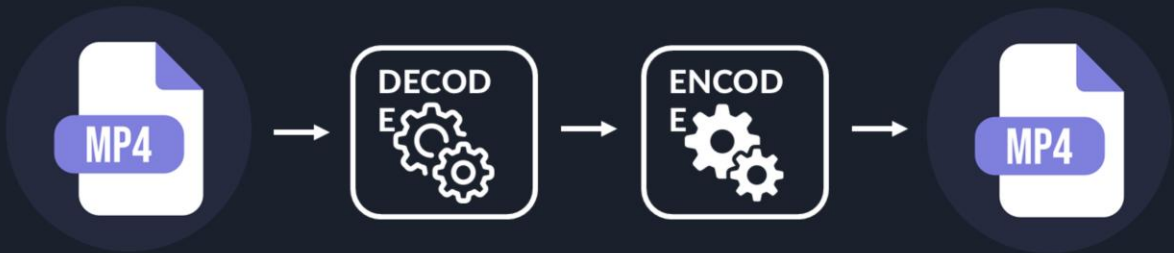


## Encoding



So encoding is the act of taking a raw video file and compressing, or encoding, it to a more manageable format.

## Transcoding

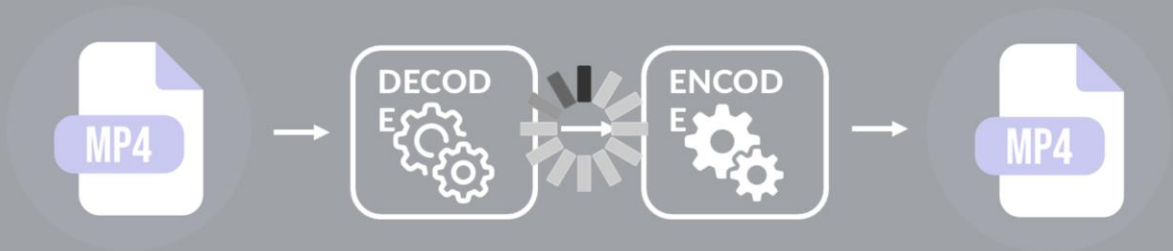


Whereas transcoding, refers to taking an already encoded video and re-encoding it to a different format. So it involves both a decoding and encoding step. In the world of video streaming, we're almost always talking about transcoding. You have to really go out of your way to produce raw video files, these things are beasts. The cameras on your phones, for example, even if crank up the quality, is still encoding the video on the fly for you.

So ya, the terms are somewhat interchangeable, both involve encoding, but if you want to sound like you know your talking about, just remember to always say transcoding.

Okay, so before we get to the good stuff, the how of video streaming, let's first talk about the why.

## Transcoding



Whereas transcoding, refers to taking an already encoded video and re-encoding it to a different format. So it involves both a decoding and encoding step. In the world of video streaming, we're almost always talking about transcoding. You have to really go out of your way to produce raw video files, these things are beasts. The cameras on your phones, for example, even if crank up the quality, is still encoding the video on the fly for you.

So ya, the terms are somewhat interchangeable, both involve encoding, but if you want to sound like you know your talking about, just remember to always say transcoding.

Okay, so before we get to the good stuff, the how of video streaming, let's first talk about the why.

# Buffering

So everyone here understands the pain of buffering. That frustrating little spinning wheel. This is especially true for those of us who lived through the early days of the internet. Nowadays, it's a relatively rare occurrence.. when was the last time you remember a Netflix show stopping to buffer? Obviously part of the reason for this is the exponential increase in available bandwidth over the years. But as bandwidth increased, so too did the quality of video that was being served. Most, if not all, of the major streaming platforms are now serving some level of 4k content.

So yes, internet connections have improved, but so too has the underpinning technologies and architectures.

# HTTP Progressive Download

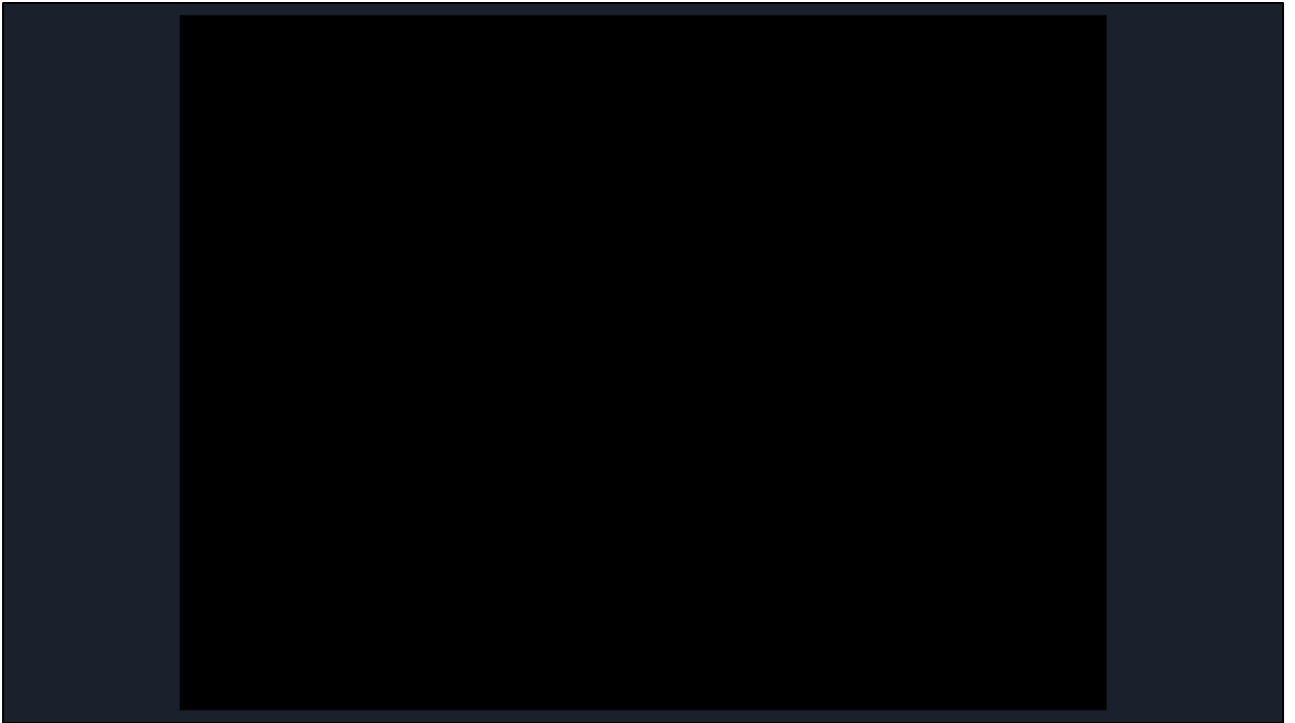
```
<html>
  <head>
    <title>Progressive MP4 Demo</title>
  </head>
  <video muted width="1280" height="720" controls>
    <source src="/videos/content/dam/excalibur/roy
      Your browser does not support the video tag
-Everyone-P2-Dining.mp4" type="video/mp4"
  </html>
```

In the early days of the internet, video streaming was done with HTTP progressive download streaming, sometimes called progressive MP4.

Nowadays, it's considered the naive approach, but it's important to acknowledge that this was cutting edge at one point in time. This spec birthed YouTube, and they spent years delivering video this way.

So, going back to this idea of media. Images, audio, and video. We all know how to serve an image on a website, we use the `img` tag and we set the source to an image that's hosted on a web server. We can do the same thing with videos using the `video` tag.

So this is HTML behind our first demo. Pretty straightforward. We have the `video` tag with some self explanatory attributes, inside we have a `source` tag pointing at an MP4 file. I have a little web server setup, that on top of serving the HTML here, is also doing some proxy magic to a Royal Caribbean domain to serve their actual videos. Something something CORS headers.



The next part of the demo we will roll the dice and do live. For this first one I just have a recording. So just a bit of setup.. I have the page you just saw loaded up in Chrome. I have the network pane open so we can see behind the curtain a bit. And I'm also using the network throttling feature of Chrome dev tools, starting with the Slow 4G setting.



## HTTP Progressive Download

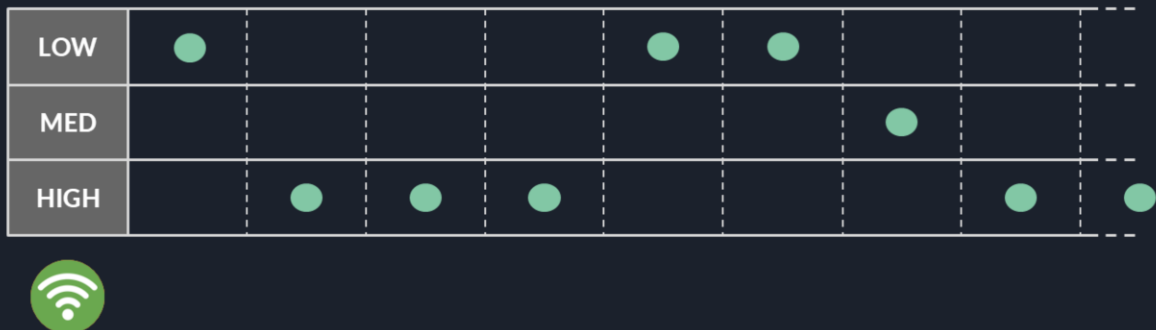


So, ya, this is the OG approach. I called it the naive approach. But maybe that's unfair. It's still actually a thing, by show of hands, who here uses Plex.

Ya, so Plex actually serves your content using progressive download. They have their own set of optimizations around it, both on the client and server side, but that's the underlying technology at play.

So if progressive download is the wrong way, or the less correct way, what's the right way?

## Adaptive Bitrate (ABR) Streaming



Enter adaptive bitrate streaming. Even if you've never heard this term before, you've seen it in action. You often see it when you start streaming a video. Initially, the quality of the video appears low, then after a second or two it switches to HD and looks great.

The source video is transcoded into a series of quality profiles. For illustration purposes, let's say low, medium, and high. This is also sometimes called an encoding ladder. The client side player will generally start with a low quality, this ensures that initial playback happens fast. As the player is downloading the video, it's internally keeping track for the download rate—the bitrate, and if the current estimated bitrate is high enough, it will seamlessly swap out the stream for a higher quality version.

This process is at play for the entire duration of the stream. So if your connection is having a momentary issue, it will detect that and downshift to a lower bitrate. The goal of the player is to play the highest quality possible without ever having to stop the video and show that dreaded buffering wheel.

When the connection improves, it will start to climb the ladder accordingly.





# ABR Streaming Protocols

## DASH

Dynamic Adaptive Streaming over HTTP

- AKA MPEG-DASH
- Developed by MPEG
- Initially released in 2012
- International Standard

## HLS

HTTP Live Streaming

- Developed by Apple
- Initially released in 2009

So we've decided to use a modern video streaming protocol. Now we need to pick one. The two main players are..

### Dynamic Adaptive Streaming over HTTP (DASH)

- AKA MPEG-DASH
- Developed by MPEG, initially released in 2012
- Considered the international standard

### HTTP Live Streaming (HLS)

- Developed by Apple
- Initially released in 2009

So, this was the first gotcha moment for us. Looking at these two protocols, it seems like DASH would be the obvious choice, right? It appears to be more of an open standard. We're trying to implement video on demand, not live streaming. The choice here should be clear, right?

Well, no, like many other specifications in our industry, where something starts and

where something ends up are two very different things.

Apple was a pioneer in this space, and so they had the benefit of being first to market. At one point it was a proprietary technology, but it's been an official standard for years now. Both HLS and DASH can do live streaming and VOD, so HTTP Live Streaming is a bit of a misnomer. And finally, Apple being Apple, doesn't support DASH. Whereas, at this point in time, all major platforms natively support HLS.

So in terms of choosing a protocol based on compatibility, HLS is the clear winner.

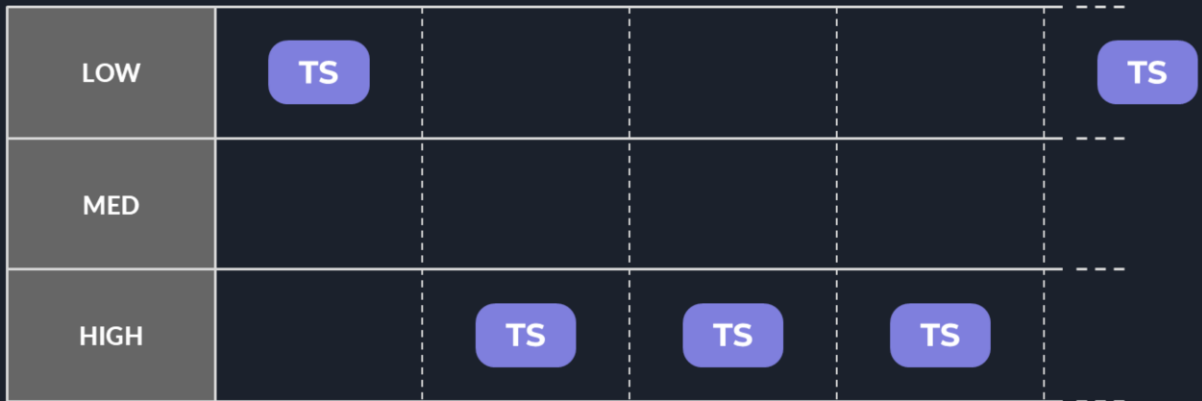


Okay, finally on to the good stuff. An HLS stream is a collection of two different files types. M3u8 and ts files.

An m3u8 is a plain ol' text file that contains metadata. It's also referred to as the manifest or playlist. And a ts file is just another MPEG container. Not to be confused with. TypeScript file.

When you go to serve an HLS stream, instead of pointing to a .MP4 file, you point to the primary M3U8 file. Often called the primary manifest. Let's take a look at one.

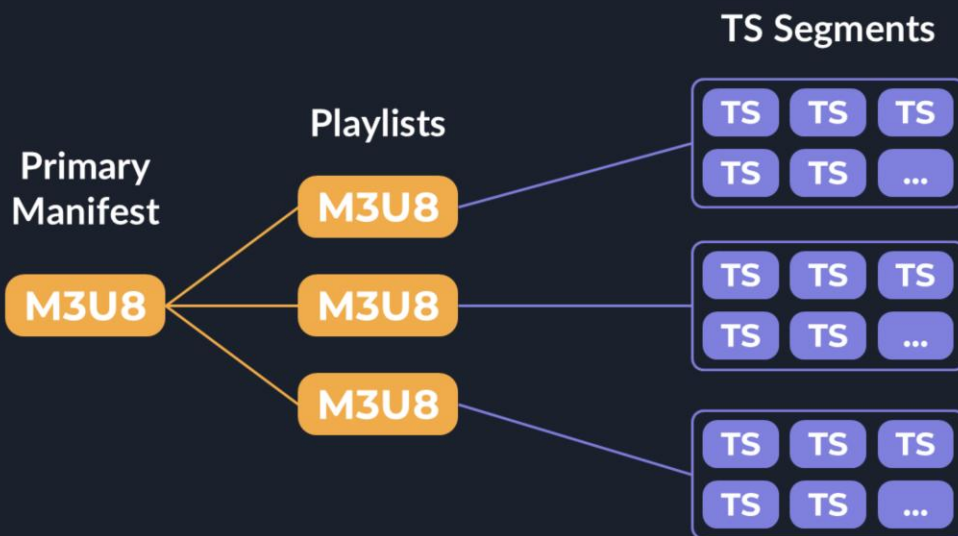
## ABR Revisit



Adaptive bitrate playback is technically possible with progressive mp4, but the available solutions are clunky.. to switch to a different quality MP4, you either need a webserver that support streaming an MP4 file at an arbitrary starting point.. or the client needs to do HTTP byte range calls. On top of that, the MP4 needs to be encoded in such a way to allow playback at those arbitrary starting points. It's a whole thing.

Segmenting the video into a collection of fixed duration files greatly simplifies the process. It provides a natural switching set.

# HLS Recap



# HLS Recap

```
/var/www/html/videos
├── 360p.m3u8
├── 360p_00001.ts
├── 360p_00002.ts
├── ...
├── 540p.m3u8
├── 540p_00001.ts
├── 540p_00002.ts
├── ...
├── 720p.m3u8
├── 720p_00001.ts
├── 720p_00002.ts
├── ...
└── master.m3u8
```



An important concept that I want to drive home is, from the backend perspective, delivering streaming video is a web server serving static files from a directory. That's it. Sure, you still have to transcode a source video to produce these files, but after that, again, you are simply serving static files over HTTP.


All the magic of adaptive bitrate streaming is done on the client side by the player, and by software and hardware decoders on the end device.

One of the benefits of having these streaming protocols built on top of HTTP, is that we can leverage existing HTTP caching solutions. So if you want to cache your videos, and your company is already using a CDN to cache web content.. well, I mean, you're basically done. It's just a bit of config. From a CDN's perspective, it's just another file to cache. I mentioned how the segmentation of the video simplifies the adaptive bitrate process, but it also naturally makes caching, especially in the context of a distributed cache like a CDN. Without getting too deep into caching, larger files don't always play nice in the sandbox.

<transition>

Alright, so I feel like we covered a lot of ground here. So now that you have a grasp of the fundamentals, I'm going to throw it back to Paul to go over our solution and

conclude our story.



Solution

Thanks Mike. When I last spoke, I was telling you how one of our teammates was the inspiration behind a new streaming solution. One that could be delivered quickly and at a fraction of the cost.

*<Next Slide>*



# Green Light: Full Stream Ahead

## Proof of Concept

Developed and demonstrated a proof of concept to showcase the feasibility of the approach.

Proof of Concept Document  
Demo Video  
Technical Feasibility Report

## Team Pitch

Presented the proof of concept to the delivery team leaders, gaining their buy-in for implementation.

Pitch Presentation  
Stakeholder Feedback  
Approval for Implementation

## Development Phase

Executed the development of the video streaming feature components.

Software Codebase  
Development Logs  
Progress Reports

## User Acceptance

Conducted thorough testing to ensure the product met quality standards.

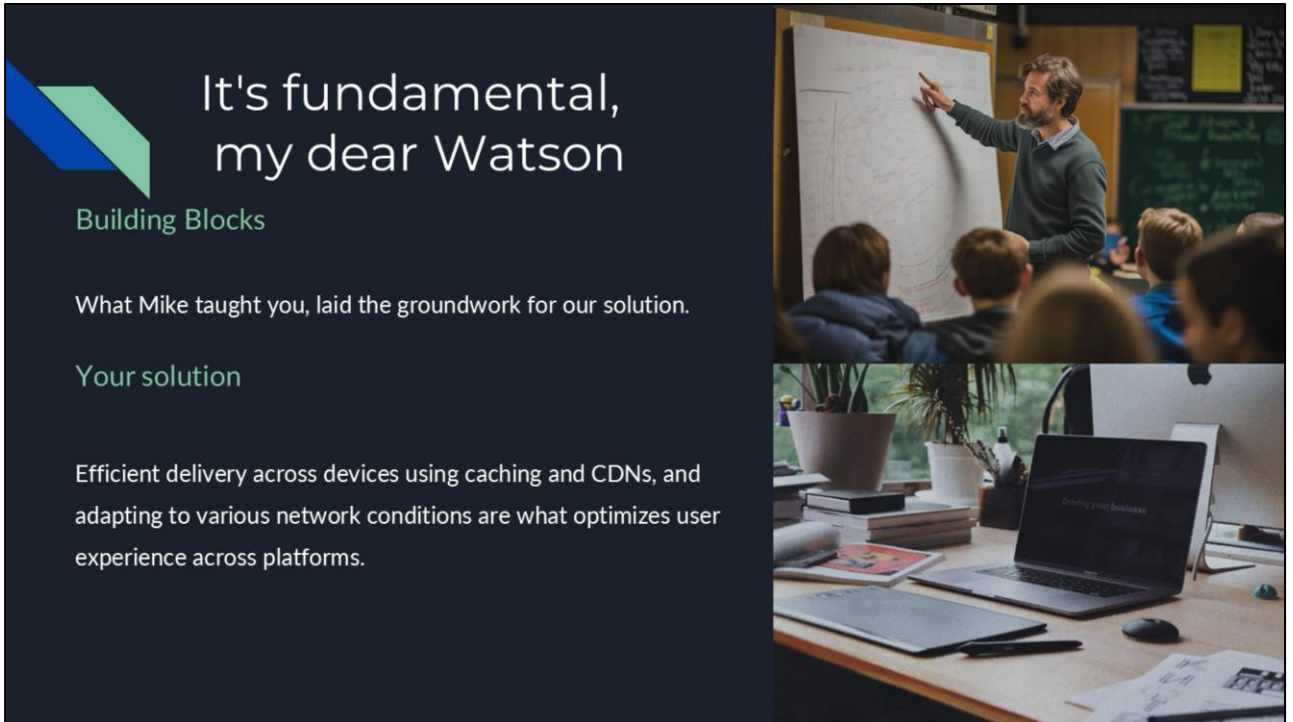
User Acceptance Test Results  
Bug Fix Reports  
Final Product

He wrote up a proof of concept and demonstrated it to me. We wagered this approach could be delivered in time to meet the deadline and was worth asking our client to give us a shot at implementing it. The subsequent pitch to the delivery team leaders went well and we began developing it. A few weeks later, we'd delivered a product that had guests happily streaming in-app videos and plans with the so-called media titan were abandoned. Go team!

Next, we'll show you what it looks like under the hood of a large enterprise and we're betting you could...<pause for effect> do-it-yourself!

Because...

<Next Slide>



# It's fundamental, my dear Watson

## Building Blocks

What Mike taught you, laid the groundwork for our solution.

## Your solution

Efficient delivery across devices using caching and CDNs, and adapting to various network conditions are what optimizes user experience across platforms.

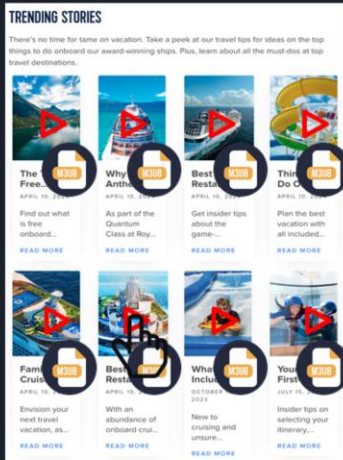
The fundamentals that Mike has taught us were the building blocks of our solution.

It is these streaming fundamentals that underpin our ultimate solution. Sure, we had to account for enterprise-level concerns and non-functional requirements such as hooks and metrics for support, managing metadata complexities and scaling, but it's the fundamentals that deliver video content efficiency to users.

Let's take deeper look.

*<Next Slide>*

# Playing a video stream



The images on this page are links to the .m3u8 playlist



Starting from the perspective of the video consumer, or in this case, a mobile app user, we'll present a list of video selections. To draw this page, the mobile app makes an API call to a Content Service that delivers a payload...

<Next>

containing a list of master M3U8 files along with metadata, such as thumbnails and titles then displays the list. Each image you see here is a link to a master M3U8 file similarly coded to what Mike showed you earlier.

<Next>

When a video is selected...

<Next>

A streaming video player is invoked and plays the HLS stream by loading the playlist file which then loads the .ts files in that manifest as the video is played.

Actually, just as Mike said, the player loads a number of manifests and once the player optimizes for network speed, downloads the .ts files from the suitable quality profile.

<Next Slide>

# Producing a video stream

## Create

Publish video using the CMS

## Transcoding Service Listener

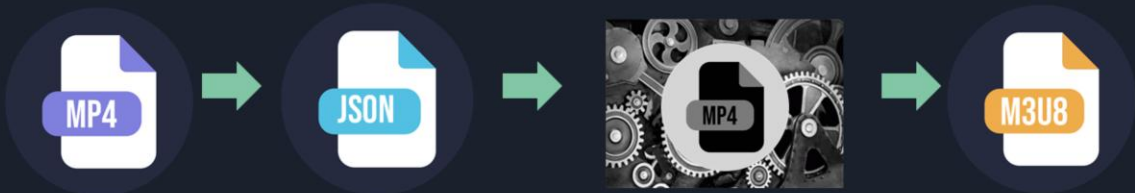
A microservice we wrote

## Transcode

Cloud Transcoder transcodes .mp4 to .m3u8 and .ts

## Ready

Content Service payload updated with HLS files



Of course, before all that can happen the backend needs to prepare the HLS files. So let's go over the production workflow and its primary components.

<Next>

Beginning with the assumption that raw videos have been encoded into MP4 files and stored in a library, content producers select videos from this library, add some metadata like thumbnails and titles, then publish them. ~~so the Content Service can deliver it to the app.~~

<Next>

Once published the video content is serialized and is now available as JSON payloads. Of course, the JSON contains paths to the MP4 files. ~~Here's where it starts to get a bit more interesting because around this step we're thinking about how and where to transcode these MP4s.~~ This is when our solution begins to take shape by and where we introduce a new microservice to listen for publish events and consume the published JSON.

<Next>

In here that we'll extract and upload the MP4 files to a Cloud Transcoder. The Transcoder executes a pipeline of transcoding jobs and notifies us when they're

complete.

<Next>

When the transcoding is complete, we have HLS files and a Content Service can use them instead of the original MP4s.

But...how does it do that? What's the Content Service? Will it have to be modified? What about the mobile apps, did they need to be updated to play HLS?

Let's dive into this workflow to find out.

<Next Slide>

# Transcoding Service Listener

Create

Publish video  
using the CMS



One of the neat things about our solution is that we made it easy for the other team's we integrate with.

First, the mobile team had to do nothing. And in fact, because of the time it takes to release an update to the app stores and nearing deadline, we could not ask the mobile team to make ANY changes to mobile app. Luckily, the current version of the app was capable of playing streaming video. But, it also meant we couldn't ask the mobile app to make any new service calls either. So, we had to make due with the API contract and JSON schemas the app was programmed for.

<Next>

Ok, we can do that. But how then could we get these HLS files to the app instead of the MP4 files it was expecting?

<Next Slide>

# Transcoding Service Listener

## Transcoding Service Listener

A microservice we wrote



On to the backend team's middleware. In the software that was first released—you know the one that delivered progressive MP4s—the Content Service used the JSON payloads published by the CMS.

<Next>

In our solution though we're going to stream video. This is where that microservice we built comes in. You'll see it here as the Golang microservice. To review, this is the microservice that uploads the MP4 files it finds in the JSON originally Published by CMS. But it does more.

Ok, so if the app couldn't call this new microservice, and we weren't the authors to the Content Service, then what could we do instead? Here's what we did.

<Next>

We fetched the same JSON payload from the CMS as the Content Service. But then we extracted and uploaded the MP4 files to a Cloud Transcoder. Once the transcoding jobs completed, we mutated the original JSON payload's video paths and had the Content Service use the mutated payload from our microservice. For that, the Content Service required only a service address config change because the payload schemas were identical to what they had been getting from the CMS.



Now when the mobile app made its call to the Content Service API, the response includes the paths to HLS files! Nice!

*<Next Slide>*

```

"data": {
  "categories": [
    {
      "name": "Home",
      "videos": [
        {
          "title": "Royal Top 5 Family Vacation Destinations",
          "media": [
            {
              "link": "/content/dam/excalibur/royaltv/videos/2023/10/CEI_Alaska.png",
              "type": "image",
            }
          ]
        }
      ]
    }
  ]
}

```



3/10/Royal-Top-5-Family-Vacation-Destinations/master.m3u8"

```

{
  "title": "Silversea - Galapagos",
  "media": [
    {
      "link": "/content/dam/excalibur/royaltv/videos/2023/10/SS_Galapagos.png",
      "type": "image",
      "video": "/content/dam/excalibur/royaltv/videos/2023/10/Silversea_NewIt2025_Galapagos_15sec_LANDSCAPE/master.m3u8"
    }
  ]
}

```

/content/dam/excalibur/royaltv/videos/2023/10/Royal-Top-5-Family-Vacation-Destinations.mp4  
 /content/dam/excalibur/royaltv/videos/2023/10/Royal-Top-5-Family-Vacation-Destinations/master.m3u8

Even more detail still! You see here the original JSON delivered by the CMS. Notice that it contains video paths to the MP4 files.

<Next>

The little Golang microservice modifies just the video path, careful not to change the schema that the Content Service expects to consume.

Essentially, this is modifying the path to the .mp4 with one to the master.m3u8.


<Next>

and now, literally zooming in... <PAUSE>

Later, we'll make another a final configuration change to the Content Service and swap out the hostname that gets prepended to these video paths for the CDN hostname and viola! Integrated!


Clearly our solution leverages a Cloud-hosted Transcoding Service to do some heavy lifting for us. Let's take a look at that component next.

<Next Slide>



# Transcode Pipeline

Job to converts an MP4 to .ts and .m3u8 files



## Create New Pipeline

A pipeline is a queue for your transcoding jobs. You can have more than one pipeline per AWS account. You can use multiple pipelines to organize your transcoding workflow, for example, by having one pipeline for standard-priority jobs and one for high-priority jobs.

Pipeline Name

Input Bucket

IAM Role

Elastic Transcoder previously created a default IAM role for this AWS account. [View the policy.](#)

### Configuration for Amazon S3 Bucket for Transcoded Files and Playlists

Bucket

Storage Class

[+ Add Permission](#)

### Configuration for Amazon S3 Bucket for Thumbnails

Bucket

Storage Class

[+ Add Permission](#)

Notifications (Optional)

Encryption (Optional)

[Cancel](#) [Create Pipeline](#)

To review, this is how the MP4 files get transcoded to HLS files. And, we use a service provider to do this work for us. For this project, because our client uses AWS, we used Elastic Transcoder.

Here, we configured a generic pipeline that defines the name, input locations and output locations. This pipeline gets invoked after our microservice uploads MP4s to the input bucket. And we'll configure our CDN's to serve the HLS files from the output bucket that we've defined here.

<Next Slide>

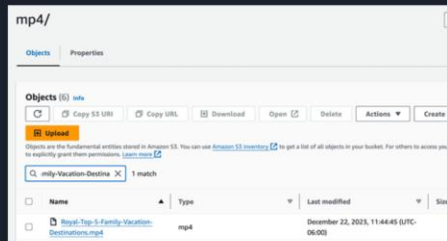
# Transcode

## Transcode

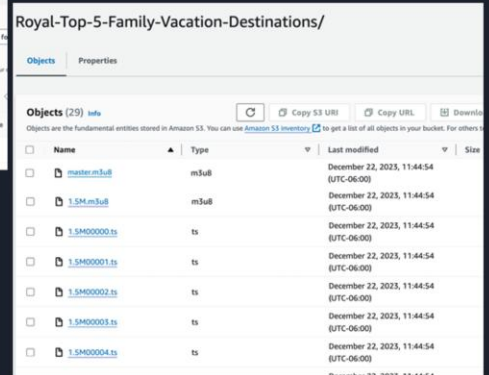
Job to convert an MP4 to .ts and .m3u8 files



## Input



## Output

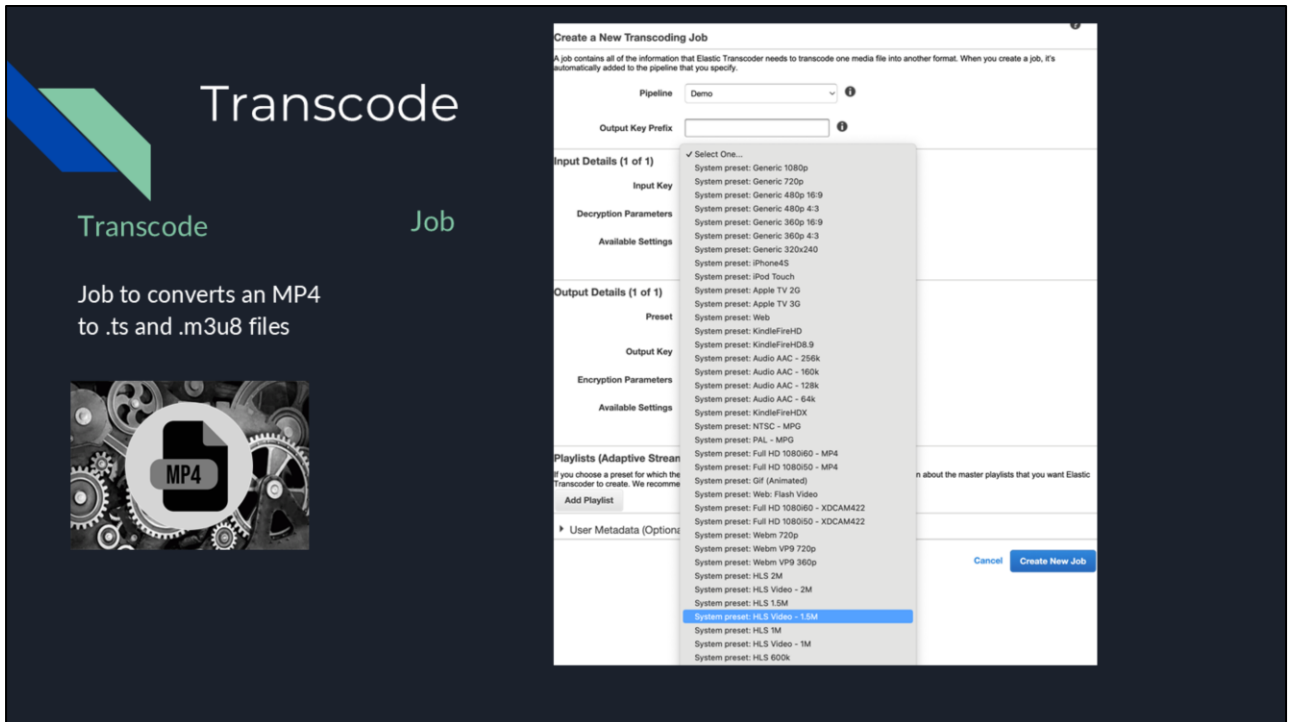


So, again, we send the MP4 files to the input bucket.

<Next>

And when the transcoding job completes, it writes the results in the output bucket

<Next Slide>



As it turned out we were able to take a shortcut here because with just one type of Job, like the one you see being created here, we got good results with a single configuration that could be used for all our transcoding work. Notice how many presets there are to select from. Also notice this stellar UI—I did not crop this screenshot.

Each time we drop an MP4 into the input bucket the pipeline executes this job. **Mike, are we notified when the job completes so that we can update the video path or do we do that eagerly?**

**Gotcha.**

In this architecture, you see that we leveraged a Cloud-hosted Transcoder, to create a scalable and efficient video processing and delivery system. It separates the concerns of content management, transcoding, and storage and delivery, allowing for optimization at each stage of the process.

**Thanks Mike. Am missing anything important to share about our solution? Or, should we get into some of the tidbits we haven't shared yet?**

**On to Appendix**



# Appendix

Our primary goal for this presentation was to cut through the noise, and distill video streaming down to the key concepts and components. We also wanted to provide an opinionated, one size fits all, starting point in terms of technology choices and architecture.

This approach didn't leave much room for detours, which meant topics we wanted to cover needed a home.

So to cap things off, fill in some gaps, we're to fire through a mishmash of additional concepts and considerations we thought you might find valuable or interesting.

# Device Optimization



So speaking of noise, something you'll constantly see when reading about this stuff is device optimization. It's so prevalent that one would assume it's a necessity. We preached that HLS is the only streaming protocol you need to worry about. Both Android and iOS natively support it. All the browsers on all the operating systems support it too. So is device optimization really needed? The short answer is, it only matters if you plan to distribute your content to third party devices where you don't have control over the player. So think, gaming console, smart TVs, Chromecast, or really any other connected device. Many of these platforms have strict requirements around the protocols and codecs that they support. So if you just plan to distribute to web and mobile, stick with HLS and ignore device optimization completely. It's extra complexity that isn't needed up front.



## Transmuxing



The longer answer as to whether you should concern yourself with device optimization involves something called transmuxing.

So we looked briefly at videojs. There's a few other libraries out there too. So, I'll be honest, I don't know a ton about how this works, but basically, these are polyfill libraries. They fill in the gaps and deal with browser specific interfaces to support as many protocols and codecs as possible.

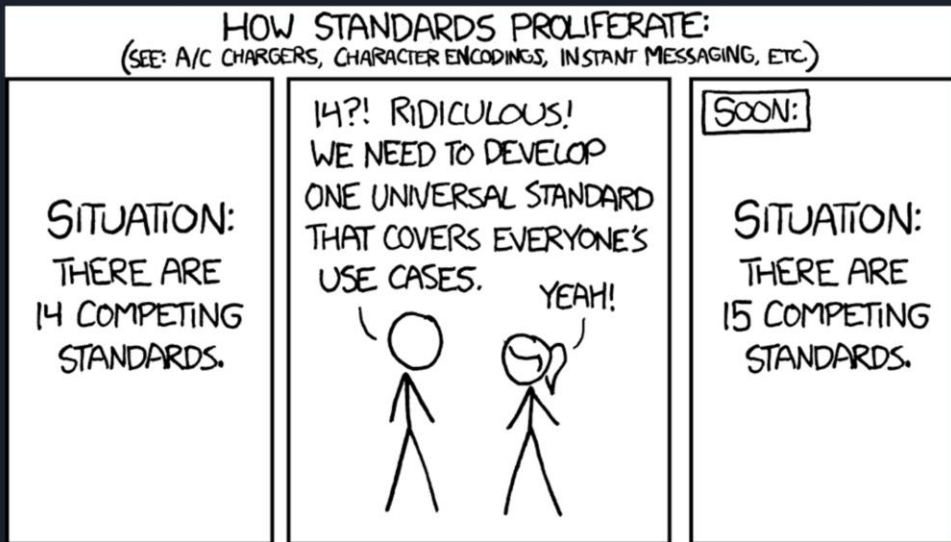
To support certain combos, they transmux the video files on the fly. Which is the process on changing the video container. So there's no decoding or encoding involved. Really it's just wrapping the video in a different container and tweaking the metadata. Repackaging the video. The codecs remain untouched.

So, it's not a terribly expensive process, but it's still a process.. so in practice, you are having some of your userbase pay for your reduced complexity with their CPU cycles and battery life.

A large part of device optimization is compatibility, being able to support multiple devices and platforms. But another part is, you know, actual optimization.

This isn't meant to discourage you. Choosing HLS and relying on libraries like videojs to achieve your compatibility goals is definitely a viable option. HLS.js, a stand alone library itself, but is also incorporated into videojs, has a list of companies that use it in production. Here's a few notable ones.

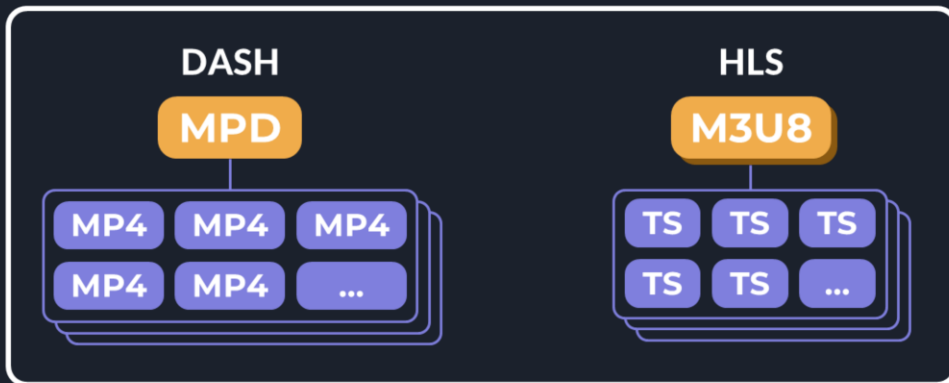
# CMAF



If supporting multiple streaming protocols is important to you, there's a new-ish protocol that aims to make this easier.

# CMAF

## Without CMAF



On the other hand, if optimization, supporting multiple streaming protocols, is important to you, there's a relatively new protocol that aims to make this easier. Common Media Application Format.

This classic XKCD seemed relevant.

We didn't really go into DASH, but in essence, it's the same idea as HLS. Instead of an m3u8 manifest, DASH uses a single XML .MPD manifest. And instead of TS segments, it has fragmented MP4 segment files.

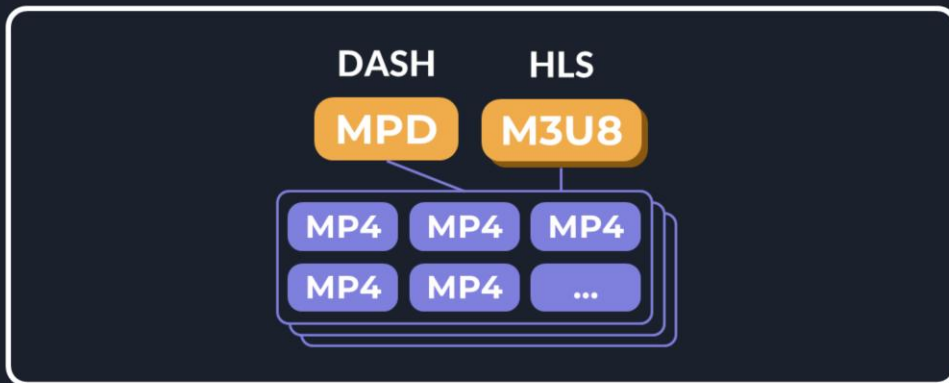
Before CMAF, supporting both protocols meant transcoding the source video twice into both formats. At its core, CMAF is a push forward to standardize containers. They somehow convinced Apple to make concession, and HLS officially supports fragmented MP4 containers now instead of TS.

So instead of having to run two full transcoding.. one to produce fragmented MP4s for DASH, and one to produce TS segments for HLS.. you can just transcode once into fragmented MP4s. You still have your format specific manifests, but they use the same set of video containers. This provides considerable cost savings in terms of compute, storage, and distribution.

In addition to a common container format, the protocol is also trying to standardize some common patterns around playback. It really feels like this.. neutral territory protocol, that the big players are using as a negotiation room to facilitate standardization and universal compatibility.

# CMAF

## With CMAF



On the other hand, if optimization, supporting multiple streaming protocols, is important to you, there's a relatively new protocol that aims to make this easier. Common Media Application Format.

This classic XKCD seemed relevant.

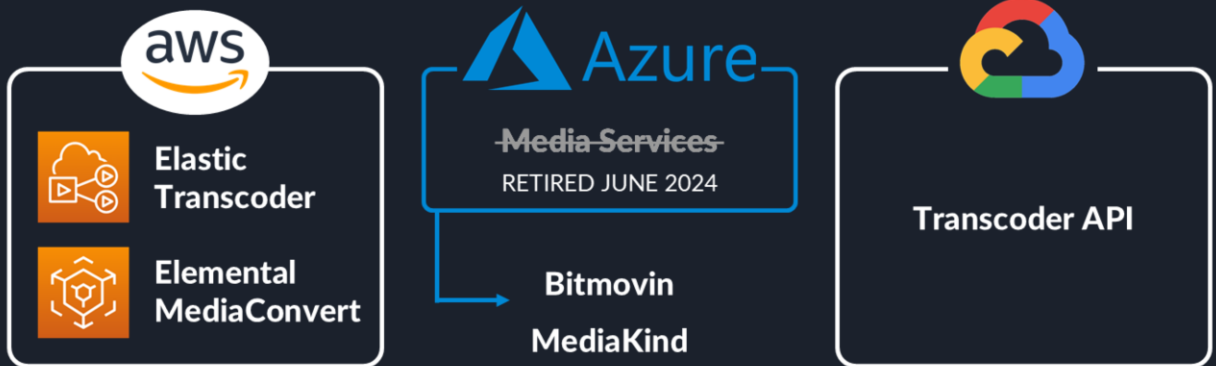
We didn't really go into DASH, but in essence, it's the same idea as HLS. Instead of an m3u8 manifest, DASH uses a single XML .MPD manifest. And instead of TS segments, it has fragmented MP4 segment files.

Before CMAF, supporting both protocols meant transcoding the source video twice into both formats. At its core, CMAF is a push forward to standardize containers. They somehow convinced Apple to make concession, and HLS officially supports fragmented MP4 containers now instead of TS.

So instead of having to run two full transcoding.. one to produce fragmented MP4s for DASH, and one to produce TS segments for HLS.. you can just transcode once into fragmented MP4s. You still have your format specific manifests, but they use the same set of video containers. This provides considerable cost savings in terms of compute, storage, and distribution.

In addition to a common container format, the protocol is also trying to standardize some common patterns around playback. It really feels like this.. neutral territory protocol, that the big players are using as a negotiation room to facilitate standardization and universal compatibility.

# Cloud Transcoding Services



So we didn't really dig too deep into about how to actually transcode your videos. We wanted to avoid getting into the implementation weeds. But ya, there's a variety of services out there with different features and pricing models. We used AWS Elastic Transcoder for this particular project, and more recently have been using AWS Mediaconvert. Azure used to have Media Services, but it was actually retired earlier this year. They have a list of third party services they suggest in their migration guide. A couple of the notable ones being Bitmovin and MediaKind. And finally, Google Cloud, GCP, has their Transcoder API offering.

But what if you're interested in playing around with transcoding without needing to leverage a cloud service, you know, *really* do it yourself... well, you can take a look into FFMPEG.





# FFMPEG

```
#!/bin/bash
set -e

mkdir -p data/

MP4="data/video.mp4"
if ! [[ -f $MP4 ]]; then
  curl -o $MP4 https://ia600209.us.archive.org/20/items/ElephantsDream/ed_hd_512kb.mp4
fi
# -hls_base_url /path/to/output/directory/
# -hls_segment_filename /path/to/output/directory/segment%d.ts
ffmpeg -i $MP4 -codec: copy -start_number 0 -hls_time 10 -hls_list_size 0 -f hls data/video.m3u8
ffmpeg -i $MP4 -codec: copy -start_number 0 -hls_time 10 -hls_list_size 0 -f hls data/nocachevideo.m3u8

echo "Successfully created HLS stream"
ls data/
```

FFMPEG is a free, open source suite of tools for video encoding, decoding, and transcoding. It's the swiss army knife of video manipulation. It's so ubiquitous in this space that I don't think it's even possible to find a transcoding service or tool that doesn't leverage it, at least in part. It's really the only game in town for universal codec decoding and encoding. Any transcoding service you might end up using is basically an FFMPEG wrapper.

So, Paul mentioned that a member of our team created a little proof of concept application as part of our pitch. Well, the actual transcoder for this POC was a bash script running an FFMPEG command. This was our transcoder. So ya, you don't need to use a cloud service.. you can start here. FFMPEG is able to transcode strictly using the CPU, but for more serious workloads, running jobs in parallel using multiple CPUs, or ideally GPUs.. might be needed.

# Conclusion

Alrighty...let's wrap up.

*<Next Slide>*



## Learning Objectives

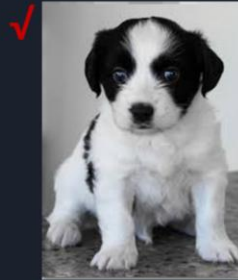
1. What is adaptive bitrate and why is it important?
2. How are videos stored and delivered?
3. Should you use HLS? And what the hell is HLS anyways?
4. Where does the video playback magic happen?

Let's recap to see if we've accomplished our goal of answering the four big questions we wanted answers to.

*<Next Slide>*

# Learning Objectives

1. What is adaptive bitrate and why is it important?



One: What is adaptive bitrate and why is it important?

ABR streaming dynamically adjusts the quality of video streams in real-time to provide the best possible viewing experience.

<Next>

See? \_wink, wink\_

<Next Slide>

# Learning Objectives

## 2. How are videos stored and delivered?



Two:  
As segments by playlist

<Next>

Yum!

<Next Slide>

# Learning Objectives

3. Should you use HLS?

Yes

~~Maybe so~~



Three:

Do you want Apple user to watch your videos?

<Next>

Remember, DASH isn't as widely supported as HLS.

<Next Slide>

# Learning Objectives

## 4. Where does the video playback magic happen?



# HLS



Four: Where does the video playback magic happen?

Anyone?

<Next>

Correct, the magic happens on the front end where the player does the hard work of determining what to do about your device's network capabilities.

<Pause>

The quick win we achieved opened the door to another phase work where we generalized our solution and accommodating more use cases. But, we kept it simple enough, didn't fall into the over-engineering trap, and think you could have similar success by knowing these fundamentals of video streaming.

Let's open it up for questions, thank you.

<Next Slide>

Questions

