Media Processing

Media Processing

- You do not always want to store user uploaded media as-is
  - Users can upload anything
  - What they upload might break your server
  - Never trust your users!
- Even your most trustworthy users will upload large media files
  - High storage cost
  - Slower load times

Media Processing

- Process the media when it’s uploaded
  - Compress the size of the image/video
  - Only store and serve the compressed files
  - Limits storage costs
  - Maintains fast load times even after large files are uploaded

- *Typically also limit the file size on the front end
  - Can always be bypassed

Aspect Ratios

- It’s important to preserve the original aspect ratio of the media
- In our examples, we’ll have a target maximum number of pixels for the height or width

- The Process:
  - Read the dimensions of the media to find the aspect ratio
  - Set the larger dimension equal to your maximum height/ width
  - Compute the other dimension based on the aspect ratio
  - Scale the media to this new height and width

Image Processing

- Recommended to use the Pillow library in Python
  - Pillow has many methods for working with images

- You may use any library you'd like to process images

Video Processing

- Recommended to use ffmpeg
- ffmpeg is *the* answer for video manipulation
- Need to install ffmpeg
  - Include the installation in your Dockerfile

- You may use any library you'd like to process videos

ffmpeg

```
ffmpeg -i inputVideo.avi -f mp4 outputVideo.mp4
```

- Example of command line ffmpeg usage
  - Converts inputVideo.avi into an mp4
- The -i flag indicates the input filename
- The -f flag indicates the output format
- The last argument is always the output filename
  - No flag for the output filename

ffmpeg

```
ffmpeg -i inputVideo.avi -s 640x360 -f mp4 outputVideo.mp4
```

- We can add more arguments for more control
  - Output filename is still the last argument
- The -s flag is sets the resolution of the output file
  - We convert the file to 640x360

ffmpeg

- To run ffmpeg in your code
- Option 1: Make a system call
  - Same as typing a command into the command line
  - Build into every language
- Option 2: Use ffmpeg bindings for your language
  - Simplifies the syntax by calling methods instead of working with command line arguments
  - Makes the system calls for you

Streaming

The Problem

- You host a video on your web app
- You want high quality so you host a large 1080p mp4
  - The entire file is 100's of MB
- Every user visiting your page has to download the entire video before playback can begin
  - Very slow to load
  - Entire file must download even for users who will only watch for a few seconds

Chunking

- Avoid the requirement of downloading the entire video before it plays
- Provided a way to request short segments of the video
- Download one "chunk" of the video at a time
  - Typically ~2-10 seconds of playback

- Advantages:
  - Only a few seconds need to be downloaded before playback starts
  - If the user skips around in the video, only request they chunk they skipped to
  - If the user leaves the page without finishing the video, the entire file doesn't have to be downloaded

HLS vs MPEG-DASH

- Two major protocols support the idea of breaking a video into smaller segments/chunks: HLS and MPEG-DASH
- **HTTP Live Streaming (HLS)**
  - Developed by Apple
  - Only supports the H.264 encoding for video
  - Wide-spread adaptation
  - Spec freely available in RFC8216
- **Dynamic Adaptive Streaming over HTTP (MPEG-DASH)**
  - Developed by Moving Picture Experts Group (MPEG)
  - Supports any video encodings
  - No support on Apple devices
  - Spec published as ISO/IEC 23009-1:2022 - Available for $245 (!)


```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:6.773433,
space0.ts
#EXTINF:8.341667,
space1.ts
#EXTINF:8.341667,
space2.ts
#EXTINF:8.341667,
space3.ts
#EXTINF:8.341667,
space4.ts
#EXTINF:8.341667,
space5.ts
#EXTINF:8.341667,
space6.ts
#EXTINF:8.341667,
space7.ts
#EXTINF:8.341667,
space8.ts
#EXTINF:8.341667,
space9.ts
#EXTINF:8.341667,
space10.ts
#EXTINF:8.341667,
space11.ts
#EXTINF:6.973633,
space12.ts
#EXTINF:2.836167,
space13.ts
#EXT-X-ENDLIST
```

HLS

- Divide the video into multiple .ts files
  - MPEG Transport Stream files
- One .m3u8 index file containing information about each .ts file and how they combine into a single video
- Your server hosts all files
- Set the video source as the index file
- Browser reads the index file to know when to request each ts file

HLS - Transcoding

```
ffmpeg -i space.mp4 -hls_list_size 0 -f hls space.m3u8
```

- Use ffmpeg to convert to HLS
- "-f hls" to specify the output format as HLS
- "-hls_list_size 0" to keep all ts files in the index
  - By deafult, ffmpeg will only keep the last 5 ts files in the index file
  - This is good if you are live-streaming (This is the HTTP Live Streaming protocol after all)
  - Since our use case is hosting Video on Demand (VOD), we want to keep every ts file in the index
  - Setting the list size to 0 means the size is not limited

Adaptive Bit-Rate Streaming

The Problem

- You host a video on your web app
  - You even use HLS or MPEG-DASH to segment the video into 2-10 seconds chunks
- You want high quality so you host chunks in 4K@60Hz
  - Can require ~25Mb/s bandwidth to stream
- And someone visits your site using eduroam on a bad day..
  - The video buffers, stutters, or doesn't play at all

- We need a solution that:
  - Allows users with slow connections to enjoy your content
  - Delivers high quality to users with high-speed Internet

Adaptive Bit-Rate

- Instead of hosting the a single video at a single resolution
  - Host multiple versions of the same video at different resolutions
  - Each resolution requires a different bit rate to stream

- User visits your page
  - Their browser adapts to the current download bandwidth available
  - Stream the highest bit rate video that fits the bandwidth

Adaptive Bit-Rate

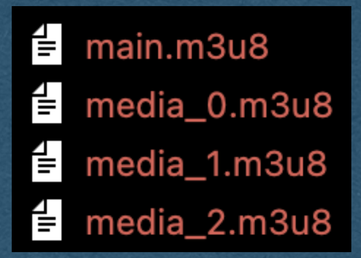
- Using HLS or MPEG-DASH
  - Create chunks at several different resolutions/bit rates
  - Add information about all resolutions in the index file

- With the video segmented into ~2-10 second chucks
  - Easy for the browser to switch between resolutions

Adaptive Bit-Rate

- The browser can adapt the requested bit-rate based on current conditions
- Limited interruption for the user, though quality can change over time

Adaptive Bit-Rate - HLS



- Using HLS, m3u8 index files can be nested
- Convert your video into multiple HLS resolutions
- Combine them into a single index file with references to the others
- This example contains references to 2 video index files at different resolutions, and 1 audio index file

```
#EXTM3U
#EXT-X-VERSION:7
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="group_A1",NAME="audio_1",DEFAULT=YES,URI="media_1.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=131049,RESOLUTION=540x960,CODECS="avc1.64001f,mp4a.40.2",AUDIO="group_A1"
media_0.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=1131049,RESOLUTION=322x572,CODECS="avc1.64001e,mp4a.40.2",AUDIO="group_A1"
media_2.m3u8
```

Video Players

- Most built-in video players do not support HLS or MPEG-DASH
  - You cannot rely on the browser having a player for either of these formats
- We must use a 3rd party video player
  - Several players available (eg. dash.js)
  - Examples in the following slides will use video.js

Video Players

- This example downloads the css and js for video.js from a CDN
- Uses "class" and "data-setup" attributes on a video element to tell the library to do its thing
- You now have a video player that supports both HLS and MPEG-DASH

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <link href="https://vjs.zencdn.net/8.10.0/video-js.css" rel="stylesheet"/>
    <title>CSE312 Video Example</title>
</head>
<body>

<video class="video-js" width="300" controls autoplay data-setup="{}">
    <source src="main.m3u8"/>
    Your browser does not support video playback
</video>

<script src="https://vjs.zencdn.net/8.10.0/video.min.js"></script>
</body>
</html>
```

```html
<video class="video-js" width="300" controls autoplay data-setup="{}">
    <source src="output.mpd"/>
    Your browser does not support video playback
</video>
```

Video Players

- Your video player will now have a consistent look across all browsers
- Don't have to worry about what formats each browser supports
  - You support any format supported by video.js

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <link href="https://vjs.zencdn.net/8.10.0/video-js.css" rel="stylesheet"/>
    <title>CSE312 Video Example</title>
</head>
<body>

<video class="video-js" width="300" controls autoplay data-setup="{}">
    <source src="main.m3u8"/>
    Your browser does not support video playback
</video>

<script src="https://vjs.zencdn.net/8.10.0/video.min.js"></script>
</body>
</html>
```

Live Streaming

Live Streaming

- We've talked about uploading and hosting mp4 videos using a streaming protocol
  - A VOD service

- What about live streaming?

- Most live streaming isn't truly live
  - There will be several seconds of delay in the stream
  - Acceptable loss to gain accuracy

Live Streaming

- Typical setup (eg. Twitch/YouTube Live/etc.)

- User streams their video into an ingest server using the Real-Time Messaging Protocol (RTMP)
  - RTMP is a container for any real-time communication
  - The content of RTMP happens to be a media stream in this case
- The server transcodes the video into a streaming format (eg. HLS/MPEG-DASH) and continually updates/ generates index files

Live Streaming

- When a viewer visits a live stream
  - The browsers asks for the latest index file and starts requesting content
  - When it nears the end of that index file, request a new index file
  - Repeat until the stream ends

- When a viewer visits the VOD of a past live-stream
  - Serve an index file for then entire stream
  - No different than watching the stream live

Live Streaming

- Since the transcoding process of the ingest server takes some time:
  - The stream is not truly live
- The streamed content is downloaded via TCP/HTTP
  - Reliable. You will not miss a second of video

- If the delay is unacceptable (eg. Zoom):
  - Use UDP instead of TCP
  - Do not transcode
  - Accept dropped packets as a part of life