C202: How To Build Your Own Cloud Encoder With FFmpeg

Jan Ozer, Principal - Doceo Publishing
David Hassoun, Principal - RealEyes
About Your Speakers

● Jan Ozer,
  ○ Contributing Editor, Streaming Media Magazine
  ○ Author, Learn to Produce FFmpeg in 30 Minutes or Less, Doceo Press, 2017
  ○ www.streaminglearningcenter.com

● David Hassoun
  ○ Founder, CEO, Realeyes
  ○ Consultancy, developer for exceptional video experiences to desktop, mobile, and OTT set-top devices
  ○ Clients include NBCS, Oracle, Adobe, MLBAM, Lionsgate
  ○ www.realeyes.com
WHO IS THIS PRESENTATION FOR?

● You have lots of video to transcode
● You distribute via one or more adaptive bitrate technologies
● You’re familiar with concepts like codecs and packaging
● You’re familiar with creating command line executions and JavaScript doesn’t offend you
● You understand some very basics of servers and how to work with them
Intro to FFmpeg

Jan Ozer
@janozer
Book from Last Year’s Class

- Includes H.264/H.265
- Creation of variant playlists with FFmpeg
- Variant/master playlists with Apple tools
- Show special:
  - Buy book
  - Email receipt to janozer@gmail.com
  - get free copy of PDF ($24.95 value)
  - Valid till 11/15
Introduction

- There are always multiple ways; seldom is there a single correct “one”
- We’re showing minimum necessary commands; there are lots more configuration options
- Location of configuration option in string typically doesn’t matter
- If you don’t choose a configuration option, FFmpeg uses the default
- Configurations in command line override defaults
Script 1: Choosing Codec

```
ffmpeg -i TOS_1080p.MOV  -c:v libx264     TOS_s1.mp4
```

- **Program**: `ffmpeg`
- **Input file**: `TOS_1080p.MOV`
- **Video codec**: `libx264`
- **Output file**: `TOS_s1.mp4`

- **Input file**: 1080p file in MOV format
  - YUV video
  - PCM audio
- **Simple script** means that you accept all FFmpeg defaults
- **Generally acceptable** for home movies; not acceptable for streaming, particularly adaptive streaming
### Encoding Output - Simple

- **Codec: x264**
  - Data rate: 15 Mbps
  - Bitrate control: average bitrate
  - Key frame: 250
  - Scene change: Yes
  - Resolution: same (1080p)
  - Frame rate: same (24)
  - Profile: High
  - CABAC: Yes
  - x264 preset: Medium
  - B-frames: preset (3)
  - B-adapt: preset (1)
  - Reference frames preset (3)

- **Audio codec: AAC**
  - Audio channels: 2
  - Audio samples: 48 khz
  - Audio bitrate: 2277 b/s

- **Other Topics**
  - Encoding multiple files
  - Converting to HLS
Bitrate Control

30 seconds talking head/30 seconds ballet
Setting Data Rate-Video

- `b:v 5000k`

bitrate video

- Sets video bitrate to 5 mbps
- No real bitrate control
- Spikes may make file hard to play

Images from Bitrate Viewer
Setting Data Rate - Two-Pass

Line 1:
- `-y` - overwrite existing log file
- `-pass 1` - first pass, no output file
- `-f mp4` - output format second pass
- `NULL` - creates log file cataloguing encoding complexity (can name log file if desired)
- `&&` - run second pass if first successful

Line 2:
- `-pass 2` - find and use log file for encode
- `Test_1080p_2P.mp4` - output file name
- Note - all commands in first pass must be in second file; can add additional commands in second line (more later)
Setting Data Rate-Two-Pass

- **Two-Pass Encode**
  - Improved bitrate control (5007 kbps)
  - Higher peak!

- **Single-Pass Encode**
  - Poor data rate control (5062 kbps)
Setting Data Rate-CBR

```bash
ffmpeg -y -i test_1080p.MOV -c:v libx264 -b:v 5000k -pass 1 -f mp4 NUL && 
ffmpeg -i test_1080p.MOV -c:v libx264 -b:v 5000k -maxrate 5000k -bufsize 5000k 
-pass 2 test_1080p_CBR.mp4
```

**Line 2:**

- `maxrate 5000k` - maximum rate same as target
- `bufsize 5000k` - VBV (Video Buffering Verifying) buffer set to one second of video (limits stream variability)
Setting Data Rate - Two-Pass

- **CBR - not flat line**
  - Peak is 5295
  - Much less variability
  - Lower overall quality (not much)
  - Can show transient quality issues

- **Two-pass ABR**
  - Poor data rate control
  - Better overall quality
CBR Can Show Transient Quality Issues

Setting Data Rate-Constrained VBR

```bash
ffmpeg -y -i Test_1080p.MOV -c:v libx264 -b:v 5000k -pass 1 -f mp4 NUL &&
ffmpeg -i Test_1080p.MOV -c:v libx264 -b:v 5000k -maxrate 10000k -bufsize 10000k
-pass 2 Test_1080p_200p_CVBR.mp4

ffmpeg -i Test_1080p.MOV -c:v libx264 -b:v 5000k -maxrate 5500k -bufsize 5000k
-pass 2 Test_1080p_110p_CVBR.mp4
```

**Line 2: 200% Constrained VBR**

- - maxrate 10000k - 200% of target
- - bufsize 10000k - VBV buffer set to two seconds of video (more variability)

**Line 2: 110% Constrained VBR**

- - maxrate 5500k - 110% of target
- - bufsize 10000k - VBV buffer set to one second of video (less variability)
Setting Data Rate-Constrained VBR

- 200% Constrained VBR - more stream variability
  - slightly higher quality
  - Avoids transient problems
- Too much variability

- Peak is 5295
- Much less variability
- Lower overall quality (not much)
- Can show transient quality issues
Setting Data Rate-Constrained VBR

- 110 Constrained VBR
  - Slightly higher quality than CBR
  - Slightly higher peak
  - Avoids transient frame issues
  - More easily deliverable than 200% constrained

- Peak is 5295
- Much less variability
- Lower overall quality (not much)
- Can show transient quality issues
Bottom Line

● Technique is pretty simple
● My tests
  ○ CBR delivers best QoE (bit.ly/BRcontrol_QoE)
  ○ CBR can introduce transient quality issues (bit.ly/VBR_not_CBR)
  ○ Bottom line: recommend 110% CVR
    ■ Very deliverable
    ■ Avoids transient quality issues
Key Frame/Scene Change - Single File

- **-g 250**
  - GOP Size
- **-keyint_min 25**
  - Minimum Space B/T Keys
- **-sc_threshold 40**
  - Sensitivity to Scene Change

- Default is:
  - Interval of 250
  - Scene change enabled
  - Minimum interval between 25
  - Sensitivity of 40
- Don’t have to do add anything; FFmpeg will deliver these defaults with or without entries
Key Frame/Scene Change - Single File

-g 250
GOP Size

-keyint_min 25
Minimum Space B/T Keys

-sc_threshold 40
Sensitivity to Scene Change

Images from Telestream Switch

Irregular Keyframes
Key Frame/Scene Change - ABR - Alt 1

- **-g 72**  
  GOP Size

- **-keyint_min 72**  
  Minimum Space B/T Keys

- **-sc_threshold 0**  
  Sensitivity to Scene Change

- **ABR**
  - Need smaller GOP so can switch to different streams much faster
  - Need consistent keyframe interval
    - Have to be at the start of all segments

- **GOP 72 (3 seconds)**
  - 72 is about the longest; many use 2-seconds
  - Adjust for frame rate

- **Minimum 72 e.g. no scene changes**
- **-sc_threshold 0 - no scene changes**
- **Need in Pass 1 and Pass 2**
Key Frame/Scene Change - ABR - Alt 1

- \( g \) 72
- \( \text{keyint\_min} \) 72
- \( \text{sc\_threshold} \) 0

GOP Size
Minimum Space
B/T Keys
Sensitivity to Scene Change

Regular Keyframes but none at scene changes
Key Frame/Scene Change - ABR - Alt 2

- `force_key_frames expr:gte(t,n_forced*3)`
  - Force Keyframe every 3 seconds

- `keyint_min 25`
  - Default Minimum

- `sc_threshold 40`
  - Default Sensitivity

- Should deliver
  - Keyframe every 72 frames

- Green are defaults
  - Don’t really need to be there
Key Frame/Scene Change - ABR - Alt 2

- `force_key_frames expr:gte(t,n_forced*3)`
  - Force Keyframe every 3 seconds

- `keyint_min 25`
  - Default Minimum

- `sc_threshold 40`
  - Default Sensitivity

Regular Keyframes, and keyframes at scene changes
Which Alternative is Better?

Static (no scene change)
PSNR - 41.22207

Scene Change Detection
PSNR - 41.25565
.08% better
Resolution

- **Simple**
  - Default is same as original; if not changing resolution can leave out
  - Set size directly
  - Simple and easy
  - Will distort if aspect ratio changes

- **More Complex**
  - More flexible approach
  - Preserves aspect ratio
  - Makes sure height is multiple of 2 \((\text{mod} \ 2)\)
    - If odd value can cause encoding problems

```
-s 1280x720

Video
Filtergraph

 vf scale=1280:trunc(ow/a/2)*2

Set width
Compute height
Same aspect ratio
Multiple of 2
```
Frame Rate

- \( r \ 12 \)

- Don’t need to include
  - Default is use source frame rate
  - Typically used to cut frame rate on lower quality streams
    - 480x270@12 fps
Profile/Level

- profile:v Baseline, Main or High
  - profile:v Baseline

- level:v number
  - level:v 4.2

- Default is High; need to use baseline for files created for Android and older iOS devices
- Use when encoding for constrained devices (mobile)
- Simply inserts level in file metadata; does not restrict encode to level parameters
x264 Preset/Tuning

- **preset** preset name (slow)
  - preset slow

- **tune** tune name (animation)
  - tune animation

- x264 has collections of encoding parameters called presets
  - Ultrafast to placebo
  - Trade encoding speed against quality (see next page)

- Default is medium - if no entry, medium parameters are applied

- Tune encoding parameters for different footage types
  - Animation, film, still images, PSNR, SSIM, grain

- My experience - animation works pretty well, the rest not so much

- Default is no tuning
## x264 Preset

<table>
<thead>
<tr>
<th>Option</th>
<th>ultrafast</th>
<th>superfast</th>
<th>veryfast</th>
<th>faster</th>
<th>fast</th>
<th>medium</th>
<th>slow</th>
<th>slower</th>
<th>veryslow</th>
<th>placebo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ad-mode</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b-adapt</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>2</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
</tr>
<tr>
<td>bframes</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3*</td>
<td>3</td>
<td>3</td>
<td>8*</td>
<td>16*</td>
</tr>
<tr>
<td>deblock</td>
<td>[0:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
<td>[1:0:0]</td>
</tr>
<tr>
<td>direct</td>
<td>spatial</td>
<td>spatial</td>
<td>spatial</td>
<td>spatial</td>
<td>spatial</td>
<td>spatial</td>
<td>auto*</td>
<td>auto*</td>
<td>auto*</td>
<td>auto*</td>
</tr>
<tr>
<td>me</td>
<td>dia*</td>
<td>dia*</td>
<td>hex</td>
<td>hex</td>
<td>hex</td>
<td>hex*</td>
<td>umh*</td>
<td>umh*</td>
<td>umh*</td>
<td>tesa*</td>
</tr>
<tr>
<td>merange</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16*</td>
<td>24*</td>
<td>24*</td>
</tr>
<tr>
<td>cabac</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>1</td>
</tr>
<tr>
<td>partitions</td>
<td>none*</td>
<td>i8x8,i4x4*</td>
<td>p8x8,p8x8,i8x8,i4x4</td>
<td>p8x8,p8x8,i8x8,i4x4</td>
<td>p8x8,p8x8,i8x8,i4x4</td>
<td>p8x8,i8x8,i4x4,x4</td>
<td>all*</td>
<td>all*</td>
<td>all*</td>
<td>all*</td>
</tr>
<tr>
<td>rc-lookahead</td>
<td>0</td>
<td>0</td>
<td>10*</td>
<td>20*</td>
<td>30*</td>
<td>40</td>
<td>50*</td>
<td>60*</td>
<td>60*</td>
<td>60*</td>
</tr>
<tr>
<td>ref</td>
<td>1*</td>
<td>1*</td>
<td>1*</td>
<td>2*</td>
<td>2*</td>
<td>3*</td>
<td>5*</td>
<td>8*</td>
<td>16*</td>
<td>16*</td>
</tr>
<tr>
<td>scenecut</td>
<td>0</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40*</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40*</td>
</tr>
<tr>
<td>subme</td>
<td>0</td>
<td>1</td>
<td>2*</td>
<td>4*</td>
<td>6*</td>
<td>7*</td>
<td>8*</td>
<td>9*</td>
<td>10*</td>
<td>11*</td>
</tr>
<tr>
<td>trellis</td>
<td>0</td>
<td>0</td>
<td>0*</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>1</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
</tr>
<tr>
<td>weightp</td>
<td>0</td>
<td>1</td>
<td>1*</td>
<td>1*</td>
<td>1*</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Yellow - default
- Green - ones that you may adjust with

* - are differing values from medium.

excerpted from http://dev.beandog.org/x264_preset_reference.html
x264 Preset

- Medium is default; works well in most cases
- If capacity becomes an issue, consider switching to Faster
  - Slightly lower quality
  - 58% of encoding time
Audio

- \texttt{c:a aac}  \hspace{1cm}  -b:a 64k  \hspace{1cm}  -ac 1  \hspace{1cm}  -ar 44100

Audio codec  \hspace{1cm}  Bitrate  \hspace{1cm}  Channels  \hspace{1cm}  Sample Rate

- Default:
  - AAC for MP4
  - Channels: source
  - Sample rate: source
  - Data rate: inconsistent

- HE, HE2 are different codecs
- Channels
  - 1 = mono
  - 2 - stereo
Multipass Encoding ABR Streams

- Can run first pass once, and apply to multiple encodes

- Which config options must be in first pass?
  - Frame settings (B-frame/Key frame)
  - Target data rate
  - Some say audio settings
    - My tests haven’t shown this is true
Which Config in First Pass?

**Pass 1 (1080 config):** ffmpeg -y -i Test_1080p.mov -c:v libx264 -preset medium -g 72 -keyint_min 72 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 3000k -c:a aac -b:a 64k -ac 1 -ar 44100 -pass 1 -f mp4 NUL && 

**Pass 2:** ffmpeg -i Test_1080p.mov -c:v libx264 -preset medium -g 72 -keyint_min 72 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 3000k -maxrate 3300k -bufsize 3000k -c:a aac -b:a 64k -ac 1 -ar 44100 -pass 2 Test_1080p.mp4

**Pass 2:** ffmpeg -i Test_1080p.mov -c:v libx264 -s 1280x720 -preset medium -g 72 -keyint_min 72 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 1500k -maxrate 1650k -bufsize 1500k -c:a aac -b:a 64k -ac 1 -ar 44100 -pass 2 Test_720p.mp4

**Pass 2:** ffmpeg -i Test_1080p.mov -c:v libx264 -s 640x360 -preset medium -g 72 -keyint_min 72 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 1000k -maxrate 1100k -bufsize 1000k -c:a aac -b:a 64k -ac 1 -ar 44100 -pass 2 Test_360p.mp4
Which Config in First Pass? Three Tests

- **Pass 1:** 1080p params
  - Pass 2: 1080p
  - Pass 2: 720p
  - Pass 2: 360p

- **Pass 1:** 720p params
  - Pass 2: 1080p
  - Pass 2: 720p
  - Pass 2: 360p

- **Pass 1:** 360p params
  - Pass 2: 1080p
  - Pass 2: 720p
  - Pass 2: 360p

- Most resources say use file in the middle 720p
- 360p produced highest results in my tests
- Not a huge difference

<table>
<thead>
<tr>
<th>TOS</th>
<th>1080p First Pass</th>
<th>720p First Pass</th>
<th>360p First Pass</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1080p</td>
<td>34.99</td>
<td>35.14</td>
<td>35.09</td>
<td>0.41%</td>
</tr>
<tr>
<td>720p</td>
<td>33.36</td>
<td>33.24</td>
<td>33.46</td>
<td>0.65%</td>
</tr>
<tr>
<td>360p</td>
<td>32.93</td>
<td>33.00</td>
<td>32.97</td>
<td>0.20%</td>
</tr>
<tr>
<td>Average</td>
<td>33.76</td>
<td>33.79</td>
<td>33.84</td>
<td>0.42%</td>
</tr>
</tbody>
</table>
**HLS Packaging**

- **Format:** Must be in first and second pass
- **Segment length**
  - Keyframe interval must divide evenly into segment size
  - Shorter improves responsiveness
- **-HLS_list_size**
  - Typically set to 0 which means all

**HLS Flags**
- When single_file, one TS file with byte-range requests
- When left out, individual .ts segments

**-f hls -hls_time 6 -hls_list_size 0 -hls_flags single_file**

- Format: HLS
- Segment Length
- Max segments in playlist.
- One file (byte-range)
In practice you should expect that all devices will support HLS version 4 ○ can use single file

You should also expect that all devices will be able to play content encoded using High Profile Level 4.1.

<table>
<thead>
<tr>
<th>HDR (HEVC)</th>
<th>HEVC/H.265</th>
<th>H.264/AVC</th>
<th>Resolution 16:9 aspect ratio</th>
<th>Frame rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 fps</td>
<td>30 fps</td>
<td>30 fps</td>
<td>416 x 234</td>
<td>≤ 30 fps</td>
</tr>
<tr>
<td>160</td>
<td>145</td>
<td>145</td>
<td>416 x 234</td>
<td>≤ 30 fps</td>
</tr>
<tr>
<td>360</td>
<td>300</td>
<td>365</td>
<td>480 x 270</td>
<td>≤ 30 fps</td>
</tr>
<tr>
<td>800</td>
<td>660</td>
<td>730</td>
<td>640 x 360</td>
<td>≤ 30 fps</td>
</tr>
<tr>
<td>1200</td>
<td>990</td>
<td>1100</td>
<td>768 x 432</td>
<td>≤ 30 fps</td>
</tr>
<tr>
<td>2050</td>
<td>1700</td>
<td>2000</td>
<td>960 x 540</td>
<td>same as source</td>
</tr>
<tr>
<td>2900</td>
<td>2400</td>
<td>3000</td>
<td>1280 x 720</td>
<td>same as source</td>
</tr>
<tr>
<td>3850</td>
<td>3200</td>
<td>4500</td>
<td>1280 x 720</td>
<td>same as source</td>
</tr>
<tr>
<td>5400</td>
<td>4500</td>
<td>6000</td>
<td>1920 x 1080</td>
<td>same as source</td>
</tr>
<tr>
<td>7000</td>
<td>5800</td>
<td>7800</td>
<td>1920 x 1080</td>
<td>same as source</td>
</tr>
<tr>
<td>9700</td>
<td>8100</td>
<td>n/a</td>
<td>2560 x 1440</td>
<td>same as source</td>
</tr>
<tr>
<td>13900</td>
<td>11600</td>
<td>n/a</td>
<td>3840 x 2160</td>
<td>same as source</td>
</tr>
<tr>
<td>20000</td>
<td>16800</td>
<td>n/a</td>
<td>3840 x 2160</td>
<td>same as source</td>
</tr>
</tbody>
</table>

Source frame rate may be as high as 60 fps
Audio sample rate: 48 khz
Keyframe: Every 2 seconds (i.e., frame rate x 2)
Segment Size: 6 seconds
Bit Rate Variability - Should not exceed 10% of target bit rate
HLS Command Line for First Three Files

Pass 1: ffmpeg -y -i Test_1080p.mov -c:v libx264 -s 1280x720 -preset medium -g 48 -keyint_min 48 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 3000k -c:a aac -b:a 128k -ac 2 -ar 48000 -pass 1 -f HLS -hls_time 6 -hls_list_size 0 -hls_flags single_file NUL &&

Pass 2: ffmpeg -i Test_1080p.mov -c:v libx264 -preset medium -g 48 -keyint_min 48 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 7800k -maxrate 8600k -bufsize 7800k -c:a aac -b:a 128k -ac 2 -ar 48000 -pass 2 -f hls -hls_time 6 -hls_list_size 0 -hls_flags single_file Test_1080p.m3u8

Pass 2: ffmpeg -i Test_1080p.mov -c:v libx264 -s 1280x720 -preset medium -g 48 -keyint_min 48 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 6000k -maxrate 6500k -bufsize 6000k -c:a aac -b:a 128k -ac 2 -ar 48000 -pass 2 -f hls -hls_time 6 -hls_list_size 0 -hls_flags single_file Test_720p_H.m3u8

Pass 2: ffmpeg -i Test_1080p.mov -c:v libx264 -s 1280x720 -preset medium -g 48 -keyint_min 48 -sc_threshold 0 -bf 3 -b_strategy 2 -b:v 4500k -maxrate 5000k -bufsize 4500k -c:a aac -b:a 128k -ac 2 -ar 48000 -pass 2 -f hls -hls_time 6 -hls_list_size 0 -hls_flags single_file Test_720p_M.m3u8
HEVC Encoding


- Integrate x265 commands into FFmpeg
  -x265-params – start of x265 commands, in x265 syntax
  -http://x265.readthedocs.io/en/default/
  -One string of commands, separated by colon, no spaces until finished

- Preset, an (audio no), pass, format, and Null outside of this structure
- Scaling commands outside of –x265-params structure

What can I do with Bento4?

https://www.bento4.com/

- HLS generation, including master manifests, stream level manifests, mpeg-2 ts files, and fMP4 (fragmented MP4)
- MP4 to fMP4 conversion
- DASH generation
- Parsing and multiplexing of H.264 and AAC streams
- Support for DRM (Marlin, PlayReady, Widevine and FairPlay).
- Support for H.264, H.265, AAC, AC3, eAC3, DTS, ALAC, and other codec types.
- Dual generation of HLS and DASH from fragmented MP4
- Atom/box editing, and stream/codec information
- A lot more… https://www.bento4.com/
Bento4 vs FFMPEG

- Bento4 focuses on MP4 based content: Packaging & Transmuxing
- FFMPEG is a broad spectrum tool for media conversion, encoding & packaging
HLS options

- Master playlists
- Single file output with byte range requests
- I-Frame only playlists
- AES encryption
- DRM
- Audio stream sidecar
- Subtitle sidecar
- fMP4
Create Multiple Bitrate Assets

```bash
mp4hls --hls-version 4 input_7000kb.mp4 input_5000kb.mp4 input_3500kb.mp4
```

**Outputs:**

- Master.m3u8
- Stream.m3u8 for each bitrate
- Iframe.m3u8 for each bitrate
- ts fragments for each bitrate
Multiple Audio Streams

mp4hls video.mp4 spanish_audio.m4a (different audio file)

mp4hls video.mp4 [+language=es]audio.m4a (multiplexed audio file, getting the spanish stream)

Outputs:

Master.m3u8
Stream.m3u8 for video and audio
Iframe.m3u8 for video and audio
ts fragments
Audio.m3u8 and aac fragments
WebVTT Subtitles

mp4hls video.mp4 [+format=webvtt,+language=en]english.vtt

Outputs

Master.m3u8
Stream.m3u8
Webvtt manifest and .vtt file
Encryption and Single Segment

mp4hls --hls-version 4 --output-single-file --segment-duration 6
--encryption-mode AES-128 --encryption-key abaa09cd8c75abba54ac12dbcc65acd7
--encryption-url http://getmyKey?token=token video.mp4

Outputs

All HLS assets (master, stream with byterange requests, iframe, single ts file)

Assets are encrypted with AES-128, and encryption URL is added to the stream manifests

Segment duration will be set to 6 seconds, but will only segment at the closest i-frame
Dual HLS and DASH From fMP4

mp4fragment input.mp4 output.mp4 (converts mp4 to fmp4)

mp4dash --force --hls --no-split --use-segment-timeline output.mp4 (without --no-split it will output .m4s segments)

 Outputs

Master.m3u8
Audio.m3u8
Video.m3u8
Stream.mpd (DASH manifest)
Example master playlist for single bitrate

#EXTM3U

#EXT-X-VERSION:6

# Media Playlists

# Audio

#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="audio/mp4a",LANGUAGE="en",NAME="English",AUTOSELECT=YES,DEFAULT=YES,URI="audio-en-mp4a.m3u8"

# Video

#EXT-X-STREAM-INF:AUDIO="audio/mp4a",AVERAGE-BANDWIDTH=3454711,BANDWIDTH=4209761,CODECS="avc1.640020,mp4a.40.2",RESOLUTION=1280x720 video-avc1.m3u8
Other Info

- Bento will only segment at an i-frame
- Creates HLS assets faster than ffmpeg or shaka packager
- Gathers its metadata while segmenting, so codecs, average bandwidth, bandwidth, and resolution are automatically added to the manifests
- A full set of DASH and metadata options

List of all Bento4 binaries: https://www.bento4.com/
Cloud Encoding (The Server)

TIME FOR SYSADMIN
OVERVIEW

• Choose your Cloud:
  ○ AWS
  ○ Azure
  ○ RackSpace
  ○ IBM SoftLayer

• Or don’t (On-prem)

• Or a hybrid (e.g. - On-prem and S3)
SIZING YOUR SERVER

● General
  ○ What general bitrates are you dealing with?

● Live
  ○ How many concurrent live streams?
  ○ Are you also transcoding optional renditions for ABR?

● VOD
  ○ How many concurrent videos being processed?
  ○ Is it transcoding or just transmuxing?
  ○ Do you need to create sidecar assets?
In AWS we’ve found m3.large to be a pretty cost effective, decently performant and reliable instance size.

We made our decision in Azure based on AWS and went with as similar a match we could find, DS2_V2.

We use Linux as our base since it’s friendlier with our software stack. Mostly RHEL.
STARTING POINT

- Get started with ec2 instances:

- Get started with Azure VMs:
Offload processing from CPU to dedicated hardware

- FFmpeg has some support for GPU Acceleration
- You need to have specific supported hardware
  - Example: AWS EC2 g2.2xlarge + CUDA + FFmpeg with -hwaccel option specified
HEVC Live – Intel Scalable Processor Family

- **x265 Boost from Intel Xeon Scalable Processor Family**
- x265 show a 67% average per-core gain for encoding using HEVC Main profile
- 50% average gain with Main10 profile across different presets
You’ll need to download and install software

- Our preferred toolset:
  - FFmpeg (Video processing and Static Builds are easy install)
  - Bento4 (Video packaging and MP4 manipulations)
  - ImageMagick (spritesheets, thumbnails and image manipulation)
  - Node.js (You need an application server wrapper)
  - MongoDB (You need some data persistence)
  - Cloud Provider SDK (e.g. AWS SDK for JavaScript in Node.js)
DIRECT LOADING

Getting started with FFmpeg

1. Select your static build: [https://ffmpeg.org/download.html](https://ffmpeg.org/download.html)
2. Download, extract, and verify:

```bash
jheider@manage:~$ wget https://johnvansickle.com/ffmpeg/releases/ffmpeg-release-64bit-static.tar.xz
jheider@manage:~$ tar xf ffmpeg-release-64bit-static.tar.xz
jheider@manage:~$ cd ffmpeg-3.1.5-64bit-static/
jheider@manage:~$ ./ffmpeg
```

ffmpeg version 3.1.5-static http://johnvansickle.com/ffmpeg/ Copyright (c) 2000-2016 the FFmpeg developers
  built with gcc 5.4.1 (Debian 5.4.1-2) 20160904
You need a good workflow architecture

- Similar to AWS Simple Workflow Service for logical and atomic chunks:
  - Workflow (End to End Execution)
  - Steps (Ingestion, Processing, Transfer)
  - Tasks (Create alternate bitrate rendition, Thumbnails)
  - Adaptors (We added this to be agnostic. E.g. AWS S3 vs. Azure Blob vs. On-prem)
Try to leverage any performance enhancements available

- **Day to Day Ingestion**
  - AWS Multipart Upload
  - Azure Streaming Put a BlockBlob

- **Initial Content Migration**
  - AWS Import/Export Snowball
  - Azure Import/Export Service
Gracefully handle all your users

- Processing takes time. You need to line up requests.
- Queuing w/persistence also lets you keep track of job status and what’s pending in case of restart.
Check out the demo: https://github.com/realeyes-media/demo-encoder

- Here's a snippet

```javascript
input.inputOptions = options.inputOptions;
output.outputOptions = ['-hls_time 8', '-hls_list_size 0', '-bsf:v h264_mp4toannexb', '-threads 0'];
input.inputURI = path.join(__dirname, '../../' + options.inputURI);
output.outputURI = directory + '/' + options.fileName + options.timestamp + '_' + bitrate + '.' + options.outputType;
options.outputURI = output.outputURI;
output.outputOptions.push('-b:v ' + bitrate + 'k', '-r ' + options.fps);

// Use options to call ffmpeg executions in parallel
executeFfmpeg(input, output)
```
Scaling

TIME TO GROW
How high can we go?

- FFmpeg will not error when the CPU is busy, just takes longer to process.
- First - Determine the Scenario:
  - The volume of files you need to simultaneously process
  - The average size of the files you need to process
  - The processing time that’s acceptable for you org
  - The kinds of operations that need to occur (e.g. Just transmux? Transcode to 4 renditions?)
- Second - Run Performance Tests
Bigger instance or more instances?

- **Bigger Instance**
  - PRO: Handles more concurrency
  - CONS: Can be more costly

- **More Instances**
  - PRO: Cheaper - Can be scaled up and down to only pay when needed
  - CONS: More complicated to manage
MULTI INSTANCE BALANCING

Scale Horizontally Transparently

- Clients hit a load balancer
- You can add more instances as needs grow in a transparent and simple way
- If your architecture is sound there’s no need for session stickiness between the clients and the transcoding system
Leverage Auto Scaling Features

- Automate the spin up/down of instances based on a number of criteria:
  - Instance Load
  - Periodic Need for Faster Processing
  - Time of Day
  - Specific Events
- AWS Auto Scaling: [https://aws.amazon.com/autoscaling](https://aws.amazon.com/autoscaling)
Docker is all the rage. Swarms and Service Discovery

- Create a swarm of Docker containers for a highly repeatable processing server snapshot that utilizes system resources efficiently
- Further increase automation through service discovery
- Implement “auto scaling” on steroids
- AWS Elastic Container Service
Encoding and Review Demos

- Demo Encoder Demo
- Manifest Viewer Demo
Conclusion

THINGS TO TAKE AWAY
THANK YOU!

- Jan Ozer
  - Principal - Doceo Publishing
  - jozer@mindspring.com
  - @janozer

- David Hassoun
  - Principal - RealEyes Media
  - david@realeyess.com
  - @hotkeys