# Video Transcoding on AWS Serverless using Lambda

V1.0

**Tom Pflaum**

**TPFL Consulting**

tomp@tpflconsulting.com

**www.tpflconsulting.com**

**20.02.2019**

# Overview

This paper discusses the advantages and disadvantages of different approaches to implement transcoding on AWS. It then details the use of ffmpeg in AWS Lambda functions.

# Approaches

There are many ways to transcode video files on AWS. Here is a list of the most common approaches and some of their advantages and disadvantage:

Transcode on an EC2 Instance

Run one or multiple EC2 instances on AWS and install the transcode software of your choice on the instances. This is sometimes called “lift and shift”.

Your cost is the license of the transcode software and the

Advantage:

- You have complete control over the transcode software you use (versions, patches, etc.)
- You can install other software components (specialized audio tools, etc.)
- Windows and Linux based software is supported.

Disadvantage:

- You have to design and provision the infrastructure on AWS. You have to architect for scalability, high availability, etc. You have to chose the right type of storage, create a virtual network, pick the right EC2 type, maintain the operating system, implement a load balancer, etc.

Using a SaaS service

You can use SaaS service such as Dolby’s Hybrik or Telestream’s Vantage Gateway.

Cost is typically based on the number of output minutes.

Advantage:

- Scalability and redundancy are provided by the SaaS vendor.
- You have access to support from the vendor.

Downside:

- You have to learn a new encoder.
- You have to learn the API to drive the encoder and integrate it in your workflow.
- Since the transcoding is performed in the vendors VPC there are security considerations.

- You also have to engage in a new business relationship with the vendor and manage things such as volume discounts and annual commits.

You can use AWS Elemental Media Convert

AWS Elemental Media Convert is AWS SaaS transcoding service.

There are other ways, like using containers within AWS EKS

You can use SaaS services that charge by the minute of transcoded media. This includes AWS Elemental MediaConvert service, and many 3rd party services from companies such as Telestream, Dolby, etc. These are easy to use, but require a separate contract and usually some form of annual commit.

You can also run a transcoder on an Amazon EC2 instance. You can use most on-prem transcoders

What are the advantages of using Lambda:

- Scaling
  You can have between 0 and 1000 Lambda functions running and you will only be billed for the actually used compute time. There is no need to estimate volume.
- No worries to leave a EC2 instance running while not using it.
  The Lambda function will only run when needed and you will get billed for the time it is actually running.
- Have a watch folder running constantly at no extra cost
  You can have a bucket on S3 to be a watch folder for your Lambda function. There is no cost for this watch operation.

What are the downsides:

- Limit to 15 minutes of transcode time
  This the big one: Currently the maximum run-time for a Lambda function is 15 minutes. AWS will terminate the function after 15 minutes.
  This limits the size of transcode jobs.
- Limit to 6 vCores.
  A Lambda function has access to a maximum of 6 vCores. While this is not a large number if provides “reasonable” transcoding speed. Some benchmarks below.

Benchmarks

Source: MP4, 1920x1080, 30fps, duration 15 minutes

Output: MP4, x264, 1mbit/s, 600x360, 30fps

Lambda Performance (10240Mb memory)

Startup time: 10 seconds

Transcode time: 199 seconds, 135 fps

On prem workstation, 24 Threads (vCore)

Transcode time: 48 seconds, 562 fps

Rough math:

Workstation has 4x the vCores compared to the Lambda environment, and it is 4.5x faster.

Thus, the performance per vCore is similar to what I am seeing on an Intel Workstation (rough estimate.)

Cost:

Using published AWS Elemental MediaConvert pricing:

1 minute of AVC, SD: 0.0075 per minute

Lamba:

The Technical Description

To allow Lambda function to run ffmpeg, I added a Layer. A Layer is a ZIP file that gets extracted to /opt when the Lambda function starts. In this ZIP file contains a statically linked version of ffmpeg.

To start the Lambda function, I am using a watch folder (meaning a Bucket on S3) that will kick off the Lambda function when a new file complets uploading to the Bucket.

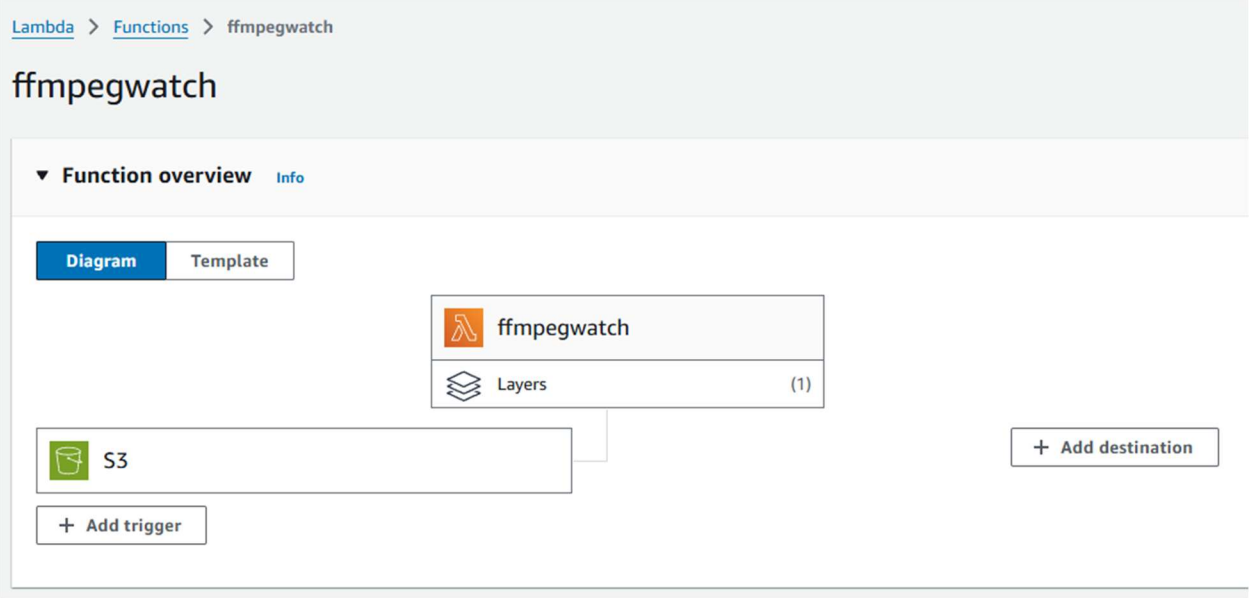
To get good performance on the Lambda function, you have to increase the memory available to the Lambda function. The number of cores is tied to the memory you specify. The maximum is 10240GB of memory, which will also increase the number of vCores to 6. (by default the memory is 512MB and a single core, so you really want to change it.) While the default network speed is not very high (I measured about 0.5Gbit/sec from S3) it is adequate for this use-case. (It is possible to increase the networking, but I did not look further into it).

The Code

I wrote the Lambda function using Python3.

- The function is started, receiving a message indicating which object (file) on S3 started the Lambda function.
- A signed URL for the S3 object (the video file) that started the Lambda function is generated (using Bodo3)

- A path to the output location of the transcoded file is hardcoded to be /temp/foo.mp4. It will be renamed when uploaded to S3.
- ffmpeg is started.
  The source is the signed URL generated earlier. ffmpeg can transcode files directly from S3 without having to localize the file.
  Encoding parameters are specified. In this case a 600x360 poxy size. Any parameter ffmpeg supports can be specified here. In this case a MP4 file with the default bit-rate will be generated.
  The output location is /tmp/foo.mp4
- ffmpeg finishes the transcode is complete, the output file is uploaded to a different S3 Bucket sing Bodo3 (as AWS warns: don't use the watch folder Bucket, as this would result in an infinite loop.)




Code:

```
import json
import os
import boto3
import urllib.parse

def lambda_handler(event, context):
    # boto3 client
    s3 = boto3.client("s3")
    # Get source bucket and key (file)
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
    # Create a signed URL for the source file
    s3_url = s3.generate_presigned_url(
        'get_object',
        Params= {'Bucket': bucket_name, 'Key': key},
```

```
    ExpiresIn=60*15  # URL expiration time in seconds
)

# Hardcoded output file. The storge will be new for every invocation
lambda_output_file_path = "/tmp/foo.mp4"

# transcoding
os.system(
    f"/opt/ffmpeglib/ffmpeg -t 900 -i \"{s3_url}\" -s 600x360 {lambda_output_file_path}"
)

# uploading transcoded file to a different bucket
output_bucket_name = "myoutput_bucket"

s3.upload_file(
    Bucket=output_bucket_name,
    Key=lambda_output_file_path.split("/")[-1],
    Filename=lambda_output_file_path,
)

return {"statusCode": 200, "body": json.dumps("ffmpeg completed.")}
```