Transcoded Documentation

Release 0.1

Martijn Braam

Jul 19, 2018

Contents:

CHAPTER 1

Installation instructions

You can install *transcoded* through the pypi repositories:

```
$ sudo pip install python-transcoded
```

This installs the transcoded daemon and its dependencies. It doesn't include any files for the init system, only the *transcoded* command that starts the daemon. An example init script for systemd:

```
[Unit]
Description=Video transcoding daemon

[Service]
Type=simple
ExecStart=/usr/bin/python3 -m transcoded

[Install]
WantedBy=multi-user.target
```

CHAPTER 2

Configuration

The *transcoded* daemon expects an configuration file in */etc/transcoded.ini* by default, you can specify another configuration file by passing the *–config* parameter to the daemon.

This is an example config file:

```
[general]
port=12380
listen=127.0.0.1

[user-mediacenter]
password=verysecret
paths=/mnt/storage/videos
callback=http://127.0.0.1/transcode-callback

[profile-h264]
container=mkv
vcodec=h264
vpolicy=always
acodec=aac
apolicy=always
vbitrate=2M
vbitratemax=3M
abitrate=192k
abitratemax=1M
```

2.1 Creating users

Every application that uses *transcoded* needs an user definition block in the configuration file. This block defines the authentication information for that application and limits the application to certain paths. This is because otherwise it would be possible to request *transcoded* to overwrite some random files that it shouldn't touch.

It also defines a callback url that is used to inform the requesting application of the transcoding progress. This is also "hardcoded" to make sure *transcoded* isn't used to make random web requests.

The user blocks section always start with *user-* and then the username for that user. The *paths* option can contain multiple paths by specifying a json list instead of a single value. The *paths* are used for both the input and output paths for the files to transcode. To disable the path checking you can just specify / as the root path.

2.2 Creating profiles

The profiles are an abstraction layer between your application and the transcoding backend, since ffmpeg/avconf aren't very consistent between the various packaged versions in distro's. Since the profile definitions are the most complex part of the configuration it is further defined in the next page

CHAPTER 3

Transcoding profiles

The profiles define the output format(s) that are acceptable for usage. With the various settings you can control how strict it is with transcoding or just copying the streams. An example profile:

```
[profile-webm]
container=webm
vcodec=vp9,vp8
vpolicy=only-mismatch
acodec=opus,vorbis
apolicy=only-mismatch
vbitrate=8M
vbitratemax=8M
abitrate=192k
abitratemax=1M
```

This profile will create files that are compatible with the webm specifications.

3.1 container

The container will define the output container format for the files, options are:

- mkv
- webm

3.2 vcodec / acodec

This defines what codec(s) are acceptable as output format. You can specify multiple codecs seperated by a comma.

If multiple codecs are specified it will always use the first codec in the list when transcoding is needed, the other codecs in the list are used for checking if the output is acceptable.

In the example above the video codec is defined as *vp9,vp8*, in that case if the input is an H.264 stream then it will transcode it to *vp9* since H.264 is not an acceptable format and *vp9* is the first specified format. If the input is an *vp8* file it won't transcode the file but just copy the stream to the new container (unless the vpolicy is changed)

Supported video codecs:

- vp8
- vp9
- h264

Supported audio codecs:

- mp3
- aac
- ac3
- vorbis
- opus

3.3 vpolicy / apolicy

The codec policy settings define what should happen if the input codec already matches the output codec options. The default setting is *only-mismatch*, in that case it will will copy the input stream directly into the output container without modification if the inputs match the output requirements.

The other option is *always-transcode*. In that mode it will always put the input stream through the encoder and decoder again, even if the codec matched. This might be necessary if the input files might contain weird things that aren't supported on your player but still use the same codec. This option is not the default since transcoding the stream is way slower than just copying it.

3.4 vbitrate / abitrate

This specifies the output bitrate for the video and audio stream. This is only used when the file actually gets transcoded according the the policy specified. If this isn't specified the defaults will be used for the specific codec.

3.5 vbitratemax / abitratemax

This is the maximum bitrate that the input file can have without it being forced to transcode. This can be used to override the policy defined when the input has a higher bitrate than can be played.

3.6 twopass

This can be set to enable twopass encoding. The default is *false* since this usually doesn't help with the encoding quality but is way slower.

3.7 Codec specific options

Some codecs also have options of their own, prefixed with the codec name.

CHAPTER 4

Codec specific options

This is the documentation for the codec specific options in the profile definitions. Most of the info here is from the ffmpeg wiki.

4.1 H264 specific options

4.1.1 h264_preset

This sets the encoding preset for *libx264*. The default option is *medium*.

Valid options are:

Preset	Speed relative to medium
ultrafast	0.45
superfast	?
veryfast	?
faster	0.75
fast	0.9
medium	1
slow	1.4
slower	2
veryslow	2.8

This controls the transcoding speed/quality tradeoff. The slower options create nicer output but cost more cpu time. Usually *medium* is the right choise.

4.1.2 h264_tune

This sets the tune parameter for the x264 encoder. Options are:

Tune	Usage
film	Use for high quality movie content; lowers deblocking
animation	Good for cartoons; uses higher deblocking and more reference frames
grain	Preserves the grain structure in old, grainy film material
stillimage	Good for slideshow-like content
fastdecode	Allows faster decoding by disabling certain filters

4.1.3 h264_profile

This sets the H.264 profile that is used for encoding, it defaults to *main*, The valid options are:

- baseline
- main
- high
- high10
- high422
- high444

4.1.4 h264_crf

This sets the constant rate factor. The CRF controls the output quality. The default is 23, higher numbers give higher quality and lower numbers give worse quality. The range is 0-51

4.1.5 h264_params

This can be used to pass extra settings to x264

4.1.6 h264_faststart

This moves the H.264 metadata to the start of the file instead of the end, this means that browsers can start playing the files quicker but it costs slightly more time to encode. It defaults to false

4.1.7 h264_yuv420p

This forces it to translate the video to YUV420 before encoding, this is required for most video players so it is *true* by default.

4.2 VP9 specific options

4.2.1 vp9_hdr

not implemented

4.2.2 vp9_deadline

This sets the *deadline* parameter for libvpx. The default value is *good*. The options are:

Deadline	Quality
realtime	Fastest encoding, but less quality
good	Default encoding
best	Very slow encoding but better quality

4.2.3 vp9_yuv420p

This forces it to translate the video to YUV420 before encoding, this is required for most video players so it is *true* by default.

4.2.4 vp9_crf

This sets the constant rate factor. The CRF controls the output quality. The range is 0-63

Recommended values

Resolution	CRF
240p	37
360p	36
480p	33
720p	32
1080p	31
1440p	24
2160p	15

4.3 VP8 specific options

4.3.1 vp8_quality

This sets the *quality* parameter for libvpx. The default value is *good*. The options are:

Quality	Description
realtime	Fastest encoding, but less quality
good	Default encoding
best	Very slow encoding but better quality

4.3.2 vp8_yuv420p

This forces it to translate the video to YUV420 before encoding, this is required for most video players so it is *true* by default.

4.3.3 vp8_crf

This sets the constant rate factor. The CRF controls the output quality. The range is 4-63.

The default value is 10

CHAPTER 5

HTTP API

This defines the API used to control *transcoded*

5.1 Authentication

For authentication HTTP basic auth is used with the username and password specified in the configuration file.

5.2 Submitting a job

```
POST /jobs HTTP/1.1
Authorization: Basic somethingsomething
Content-Type: application/json

{
    "source": "/home/video/input.mov",
    "destination" :"/home/video/output.webm",
    "profile": "webm",
    "id": "21"
}
```

You submit a job by POST'ing a json blob to the */jobs* endpoint. The fields in the json blob:

Field	Description
source	Full path to the source file
destination	Full path to the destination file
profile	The profile to use for transcoding, defined in the configuration file
id	(optional) Id for the transcoding job for reference of the sending application. This is sent back to the application with the callbacks

The possible responses:

Status	Body
200	Job is queued
400	No valid json data received
400	Missing required field xxx
403	Source path not allowed for x
403	Destination path not allowed for x
404	Profile “x” is not defined
404	Source file does not exist

5.3 Receiving callbacks

For every jobs the requesting application will receive various callbacks at the callback url defined in the config file. The callback is always a *POST* request.

5.3.1 Job started

```
{
    "status": "started",
    "source": "/home/video/input.mov",
    "destination" :"/home/video/output.webm",
    "id": "21"
}
```

5.3.2 Job progress

```
{
    "status": "running",
    "progress" 22.4
    "source": "/home/video/input.mov",
    "destination" :"/home/video/output.webm",
    "id": "21"
}
```

5.3.3 Job completed

```
{
    "status": "done",
    "source": "/home/video/input.mov",
    "destination" :"/home/video/output.webm",
    "id": "21"
}
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search