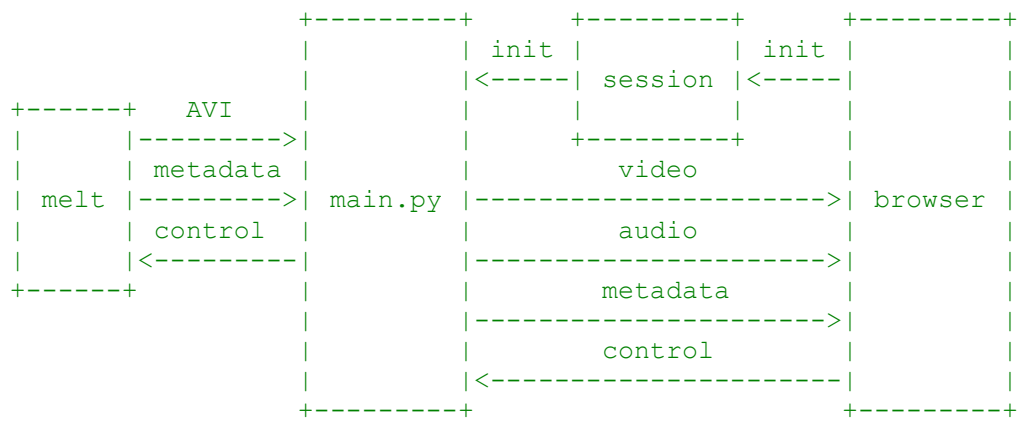
jit-webrtc

Live Transcoder with WebRTC output

Design

The system consists of four "moving parts": three on the backend (`melt`, `main.py` and `session-service`)
and one on the frontend (website with a JavaScript/WebRTC client).



`melt` is used to decode input files and perform rendering.
It is the command line interface (CLI) for the *[MLT framework]* (https://mltframework.org/),
a popular framework for many free software non-linear editors (NLEs).

Control commands are sent from the browser to `main.py` via a WebRTC data channel,
and then from `main.py` to `melt` via a stdout/stdin.
`melt` in turn sends an AVI stream with rendered video and audio to `main.py` via a named pipe,
and status messages and metadata is sent to `main.py` via another named pipe.

`main.py` will extract audio and video from the incoming AVI stream and encode them for sending to the browser via WebRTC.
It has the ability to extract specific channels from the rendered audio, to be rendered to an stereo stream for WebRTC.
It does not appear to be possible to encode surround sound for WebRTC, despite WebRTC using the Opus codec which is surround capable.
This may be a limitation in `aiortc`, in WebRTC or in the browser, or all three.

The JavaScript client receives metadata from `main.py` and sends commands and ping replies in return over the control channel.
It also receives and plays audio and video.

`main.py` and `melt` is started once for each JIT session, and media is presented to it on startup.
`session` is a Java-based service that presents a REST api that can be used to start a new JIT session given a description of
the media to use. It also has support for managing a full ASG cluster on AWS, and distribute new JIT sessions on the instances in
the ASG.

For more information on each component see:
* *[`melt`](https://github.com/sirf/mlt.git)*
* *[`main.py`](jit/README.md)*
* *[`session`](session/README.md)*

Development

For development and quick testing without the session service, on can either start JIT as a docker, or run JIT directly on your computer.
The latter has some problems if you're on an Apple ARM.

Run the docker:

`docker/main/main.sh --threads 16 --port 8080 $VIDEOFILE`

Where `$VIDEOFILE` is either a local path or a URL. `threads` and `port` are both optional, with current default values set to 72 threads and port 8080.

To run JIT directly, see [`jit/README.md`](jit/README.md).

Open Docker IPs

Your Docker container will get an IP in the style of 172.17.0.*. Using e.g. Docker CE on Linux this IP will be reachable from outside the container.
However, on Docker Desktop for Mac, this IP will not be opened.
This will cause problems. There is a service that can fix this. You **install and activate it once** by doing this:

```
# Install via Homebrew
brew install chipmk/tap/docker-mac-net-connect

# Run the service and register it to launch at boot
sudo brew services start chipmk/tap/docker-mac-net-connect
```

Deployment

Currently the only way supported for deployment is by using EC2 instances on AWS. An AMI is created by CI on release, containing a
local instance of the `session-service`, together with `main.py`, `melt` and anything else needed to start JIT processes on the EC2
instance. For more information about what the AMI contains, and how configuration can be fed to it see [`ami/README.md`](ami/README.md).

Terraform

Terraform modules are supplied that simplify setup of either single EC2 instances or a full ASG cluster of instances, including
`session-service` running as an orchestrator on ECS. See [`terraform/README.md`](terraform/README.md) for more information on
available modules, their parameters and outputs.