Migrating from Adobe Connect

The Victory of FOSS Over Proprietary Software

Jess Portnoy jess.portnoy@kaltura.com, Kaltura, Inc

Adobe - dictionary definition

> *a kind of clay used as a building material.*

Abstract

Adobe Connect is a proprietary platform for virtual presentations, conferencing sessions and screen recordings.

Recently, one of our customers has requested our assistance in migrating their content from Adobe Connect to
Kaltura.

We've released the project as FOSS [licensed under AGPLv3]. This session will cover the challenges we faced and the FOSS tools we used to overcome them.

Session Overview

During the session, we will review the steps the code takes in order to extract and process the data associated with an Adobe Connect recording [video and audio files, metadata, etc].

We'll walk attendees through our use of FFmpeg, Selenium, Mozilla's Geckodriver and OpenCV and demonstrate how they were harnessed for the purpose of migrating the content.

API

For this project, we used the Adobe Connect API in order to retrieve the metadata for the assets [name, creation date, creator, etc].

Several third-party FOSS clients are available, we chose adobe_connect which is written in Ruby.

Sadly, no API for obtaining one cohesive media file of a given session is available. Which, of course, is the most crucial component.

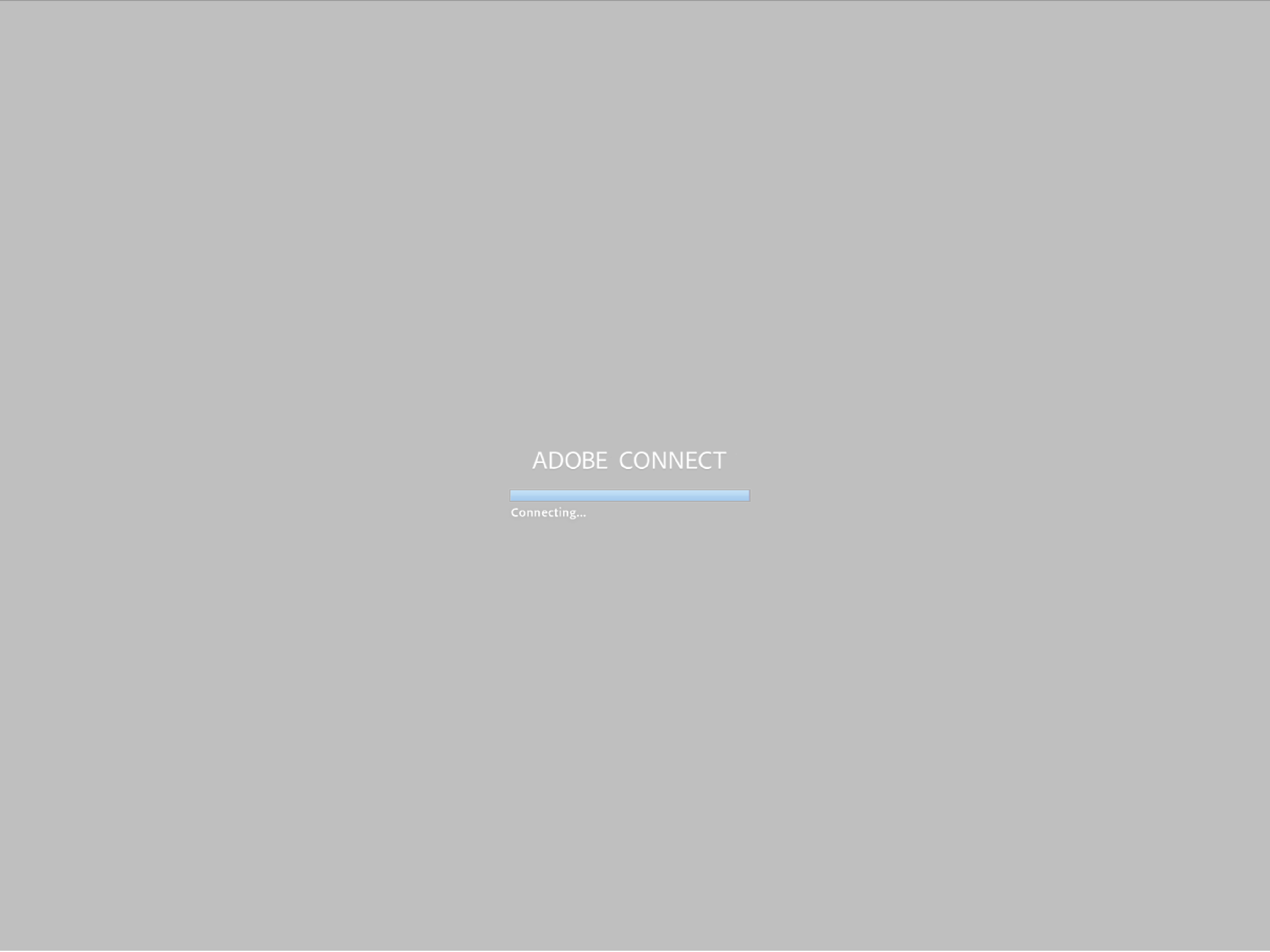
Obtaining the assets

To properly appreciate the challenge, one must first understand how Adobe Connect stores the assets. A typical asset [recording] consists of the following:

- Audio FLV files
- Video FLV files
- Widget [pod] FLV files
- Metadata XML files

General flow

- Download the ZIP archive and concat the audio FLVs into one MP3 using FFmpeg
- Using Selenium and Mozilla's Geckodriver, launch Firefox with Xvfb and play the recording using the Adobe SWF
- Use FFmpeg's x11grab option to capture the screen display
- Once done, use FFmpeg's scene detection feature to determine when the recording had actually started
- Merge the audio and video files and use the Kaltura API to ingest the resulting file



ADOBE CONNECT

Connecting...

Parallel processing

Our customer had roughly 40,000 assets to migrate. Therefore, it was of paramount importance for the code to be able to handle multiple assets concurrently.
To that end, we've written a small wrapper around xvfb-run.
The number of concurrent jobs to run is determined based on the value of the `MAX_CONCUR_PROCS` ENV var and the only real limitation is HW resources [namely: CPU, RAM].

A word about slides...

By following the above method, we were able to produce a standalone MKV file that most common media players can play and our platform can easily ingest and transcode into different flavours.

However, we felt that more could be done...

A word about slides - cont'd

The Kaltura player is capable of displaying slides alongside the video. A slide is represented by an object called a "thumb cue point". This object has several important properties:

- An image representing the actual "slide"
- The title [string]
- The description [a longer string, typically the full textual contents of the slide]
- The start_time which denotes when the slide should be displayed during playback [i.e starting from the N second of the video]

A word about slides - cont'd

While the video we produced by recording the screen display shows the speaker, as well as the presentation widget [pod], there is a lot of metadata here that could be leveraged to provide a better user experience.

Alas, the original presentation files [typically PPT or PPTX] cannot be easily downloaded from Adobe Connect, certainly not by using the API. It was, therefore, necessary to find an alternative way of obtaining the slides.

Enter OpenCV

Using a combination of the FFmpeg scene detection feature and OpenCV we were able to accomplish the following:

- Get the timings for the slide changes
- Determine the dimensions of the slide widget/pod inside the frame and generate images per slide

A careful review of the asset ZIP archive yielded an interesting find: a file called srchdata.xml, which consists of the textual contents of the presentation, per slide. We've used that data to set the title and description members on the cue point objects.

Credit where credit is due

This project would have taken far longer had it not been for the existence of FOSS. In particular, I'd like to thank the fine people responsible for FFmpeg, Geckodriver, Selenium, and OpenCV for all their amazing work.

I would also like to thank my friend and colleague `Hila Karimov`. Hila joined the project immediately after the POC phase and has been instrumental in implementing several important features as well as supporting the customer through the migration process.

And, last but not least, thanks to `Jack Sharon` for his initial discovery work with the customer, his encouragement and commitment to finding an apt solution. Cheers, Jack!

Thank you && Questions

Appendix - Useful Resources

- Migration code
- FFmpeg
- Selenium
- Mozilla Geckodriver
- OpenCV