Collabora Online and WASM

Assembling off-line Collabora Online with the Web

Tor Lillqvist

tml@collabora.com

Collabora Online

FOSDEM

The problem

- You are happily editing a document in Collabora Online in your browser.
- Then you internet connection breaks. Your trains enters a tunnel. Your guinea pig gnaws through the cable. Your IT staff pulls out a router plug.
- You are not able to edit any longer and not even to save the document locally.

The solution

- When the connection breaks, the JavaScript bits notice, and an in-browser WASM application is activated that contains the Collabora Online server functionality.
- The JavaScript functionality of Collabora Online now talks to the in-browser WASM code instead.
- A reasonably current version of the document has automatically been kept downloaded in memory in the browser.
- All editing functionality works normally.

The solution (2)

- When the connection is re-established, the edited document is uploaded and editing continues using the Collabora Online server.
- Until some better name is invented, let's call this concept COWASM, for Collabora Online as WASM.

Non-solution

So what is WASM then?

- Thorsten or Balázs maybe already described that in their talks, but in case not:
- WASM stands for WebAssembly. It is a way to run code compiled from C, C++, Rust and other languages in a restricted environment in a browser, on any platform, in a JavaScript sandbox. The code has no more access to the system than JavaScript has.
- As such there are WASM execution environments that are not browsers, too.
- Supported in most modern browsers.

And what is Emscripten?

- Emscripten is the Clang-based toolchain targeting WASM, and a runtime C and C++ library. Much of normal Linux functionality is present, including pthread and a file system (in-memory) .
- Currently Emscripten SDK version 2.0.31 is used. That is a bit old, but known to work.
- Upgrading to a more current Emscripten is desirable and will probably be done soon. I have tried 3.1.30 and it did seem to work.
- LibreOffice itself has been buildable for WASM for a while, thanks to Allotropia. That effort uses Qt5. COWASM does not.

What all goes into COWASM?

- The COWASM application binary consists of the C++ code of LibreOffice core, the C and C++ code of LibreOffice external dependencies (libraries like boost and libxml2), and the C++ code of Collabora Online.
- The JavaScript code of Collabora Online is what provides the UI. It switches between using a Collabora Online server and the downloaded COWASM binary as needed. The COWASM code does not display anything on the web page itself.

How does COWASM work?

- The structure is quite similar to how the Collabora Office iOS and Android applications work. Also these have code from LibreOffice core, libraries, and Collabora Online linked together as one program that runs as a single process.
- That there is just one process is essential. The “real” Collabora Online server uses multiple processes but that is not possible or desirable for an iOS or Android app. Instead, a multitude of threads is used.
- The same holds for COWASM. One webpage, one WebAssembly program, with multiple threads (that run as “Web Workers”).

Comparison: How does COOL work?

- The “real” Collabora Online (COOL) server uses these processes:
  - The main process, “WSD”.
  - A process that has loaded LibreOffice core and other libraries dynamically, “ForKit”.
  - For each document being edited, a “Kit” process.
- These processes communicate over sockets with messages encapsulated as WebSockets for consistency. WebSockets is also used for the communication between the COOL server and the JavaScript in the browser.

The mobile apps and COWASM

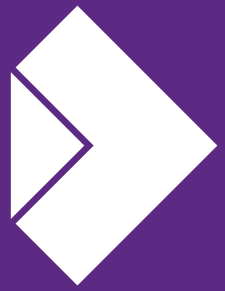
- Instead of multiple processes, the corresponding C++ objects are in the same process, but each runs a separate thread. Messages between them are exchanged over an in-process buffering mechanism called “FakeSocket”. That is designed to mimic IPC socket communication. (WebSockets encapsulation is not used.)
- This was the fastest and easiest way to get Collabora Online working as a mobile app.
- There is also a GTK+ app that is similarly structured. It is intended to offer a way to experiment with the concept on just a Linux machine.

The mobile apps and COWASM (2)

- Ideally, not so many threads should be used. In many cases, instead of sending a message to an object over a FakeSocket, the sender should just directly invoke a method on the receiving object to handle the message. After all, they are all in the same process.
- But for now, at least in the mobile apps the FakeSocket approach has worked reliably. The hope is that it works as well in COWASM, but that is not yet fully clear.

COWASM development details

- See static/README.wasm.md in LibreOffice core.
- See wasm/README in Collabora Online.
- Additional dependencies: Poco and zstd. Both are easy to compile with Emscripten. Instructions in the above README.
- First you build core, then Poco and zstd, then Online.
- Because of pthreads, and thus SharedArrayBuffers, browsers require that the HTML, JS, and WASM is loaded from a web server, not from file: URLs. Additionally, some extra HTTP headers are needed.



Collabora Online

Thank you!

By Tor Lillqvist

@CollaboraOffice
hello@collaboraoffice.com
www.collaboraoffice.com

Join the team: col.la/join