Blog post

Postgres WASM by Snaplet and Supabase

2022-10-03
•
12 minute read

Mark BurggrafEngineering

Today we're open sourcing `postgres-wasm` with our friends at Snaplet.
postgres-wasm is a PostgresQL server that runs inside a browser. It provides a full suite of features, including persisting state to browser, restoring from pg_dump, and logical replication from a remote database.
We're not the first to run Postgres in the browser - that title belongs to the team at Crunchy Data who shared their version of this on HN a month ago. It's awesome, and we wanted an open source version, so we teamed up with Snaplet build it. Let's explore how it's built, and some extra features we've added.

> 🐈‍⬛ Who's Snaplet?
> Snaplet gives developers production-accurate data and preview databases that they can code against, so they can focus on shipping.
> Checkout out how you can use Snaplet to clone Supabase environments.

Demo

Before we get into the technical details, try it out yourself (warning: it will download ~30MB):
wasm.supabase.com

To run it locally:

```
# From Snaplet Repo
git clone git@github.com:snaplet/postgres-wasm.git
cd postgres-browser/packages/pg-browser
npx serve

# From Supabase Fork
git clone git@github.com:supabase-community/postgres-wasm.git
cd postgres-wasm
git checkout web
cd packages/supabrowser
npx serve
```

And open a browser at localhost:3000.

Features

Our demo version has a few neat features!

Postgres 14.5, psql, pg_dump, etc.

Save & restore state to/from a file.

Save & restore Postgres state to/from the browser storage (IndexedDB).

Quick start from a state file or fully reboot the emulator.

Memory configuration options from 128MB to 1024MB.

Adjust the font size for the terminal.

Upload files to the emulator (including database dumps and CSVs).

Download files from the emulator.

Outgoing network connectivity from the emulator to the internet.

Incoming network tunnel to postgres port 5432 inside the emulator.

Why?

That's a good question. `postgres-wasm` is currently about 30mb. So at this stage, running Postgres in the browser isn't great for general use-cases. It has a lot of potential though. Some ideas we'll be playing with over the next few months:

Documentation: for tutorials and demos.

Offline data: running it in the browser for an offline cache, similar to sql.js or absurd-sql.

Offline data analysis: using it in a dashboard for offline data analysis and charts.

Testing: testing PostgresSQL functions, triggers, data modeling, logical replication, etc.

Dev environments: use it as a development environment — pull data from production or push new data, functions, triggers, views up to production.

Snapshots: create a test version of your database with sample data, then take a snapshot to send to other developers.

Support: send snapshots of your database to support personnel to demonstrate an issue you're having.

Repository Overview

We've divided in our repository into three folders: a virtual machine, a web application, and a network proxy.

Virtual Machine (VM)

We create an embeddable Virtual Machine (VM) using Buildroot. Our VM is a stripped-down Linux build with Postgres installed.

See source code.

Web application

Next, we need to run the the VM inside our browser. How? WASM. We use v86 to run our VM inside the browser. Our demo application is very simple - plain HTML and some basic styling.

See source code.

Network proxy

Running Postgres in a browser is great, but connecting to it with PgAdmin is even better. Unfortunately browsers block TCP network access to the VM. To circumvent this, we proxy the traffic through websockets. We run a fork of Websockproxy which allows the emulator to talk to the internet by converting data sent over a websocket port into TCP packets. Our fork of Websockproxy adds the ability to tunnel into the Postgres server.

See source code.

Technical deep-dive

There were a lot of hurdles that we discovered throughout the development of postgres-wasm.

WASM

The first thing to point out is that our implementation isn't pure WASM. We attempted to compile Postgres for WASM directly from source, but it was more complicated that we anticipated.

Crunchy's HN post provided some hints about the approach they took, which was to virtualize a machine in the browser. We pursued this strategy too, settling on v86 which emulates an x86-compatible CPU and hardware in the browser.

PostgreSQL 14 segfault errors

We quickly had PostgreSQL 13.3 running using an outdated version of Buildroot. But version PG14+ wouldn't start, giving a segfault during initialization. We tried:

manually copying build files for PG14 into the older version(s) of Buildroot

building with (many) newer copies of Buildroot

adjusting kernel parameters and environment settings such as the amount of memory allocated to the emulator, etc.

Eventually, Fabian, the creator of v86 suggested we turn off JIT compilation for v86 and that solved the issue. He narrowed it down to a bug in v86 and is pushing an update that will fix it. Switching Postgres from sysv to posix memory management also solved the issue for the current release of V86.

Optimizing startup time and image size

With PG14 running in the emulator, we shifted our focus to performance. The image size for the emulator was too big for a browser-based tool. Even with our best efforts, a compressed snapshot was over 30mb - a fairly large payload to download before you can see any interaction.

We solved this by booting only a minimal Linux image and then dynamically loading the rest of the VM over HTTPS after initialization.

We achieve this by mounting a compressed 9P filesystem in the VM. 9P provides a Python script which takes a filesystem folder, renames every file an 8-character name and produces a `filesystem.json` file representing a nested structure with files, original file names, sizes, etc. We then copy this compressed output to the VM. We modified the kernel command line and v86 boot parameters to boot directly from the 9P filesystem, and even put our kernel file into the p9 filesystem. All non-essential files are loaded asyncrhonously over HTTPS in the browser as needed.

The initial state was smaller, but still 15-20mb in size. We discussed this with Fabian who pointed us towards the `page_poison=on` kernel parameter. This parameter allowed us to clear caches before creating the snapshots, by forcing Linux to write arbitrary bytes on freed memory instead of random bytes, so unused memory is compressed much more efficiently.

The end result of all these changes? The compressed initial state file is about 12mb - including a running network state and Postgres 14.4 running with psql loaded.

We experimented without Postgres started too - this produced an even smaller initial state but the time to initialize Postgres was usually longer than the time to download a few extra megabytes. We opted for a faster startup time.

Additionally, without a running state you'd hit a performance penalty every time you refreshed the page, since Postgres would need to be initialized each time. In our current version, after the initial state is downloaded and cached by the browser the VM feels almost instant on refresh.

Networking

Networking was particularly difficult to solve. For security reasons, browsers only support access to a nominal set of ports. In short, browsers speak “80/443” and Postgres speaks “5432” so there was no way for our Postgres to communicate with the outside world.

WASM isn't much help here - a WebAssembly module running in the browser is generally not able to do anything outside itself, except for calling functions exposed from JavaScript.

Browsers provide another connection option for us however: Websockets. The nice thing about websocket connections is that they are persistent, so all we needed was to figure out how to proxy the virtual machine traffic through a websocket connection.

The VM includes an emulated network card (`ne2k-pci`) and a `proxy_url` startup parameter which points to an external server running a websocket proxy (i.e. on the internet). The websocket proxy establishes connections via a websocket port, then accepts raw ethernet packets. It turns those raw packets into TCP/IP packets and routes them between the Internet and the VM.

With our tunnel established, we could now send network traffic to the outside world. For example, try “starting” the network on the VM, "exiting" out of psql (`cmd+D`), and then run `ping 1.1.1.1`.

Most v86 VM's use an open proxy at `wss://relay.widgetry.org/` based on websockproxy. This proxy allows the VM to communicate with the outside world. It can handle all the basic use-cases like loading data from other PostgreSQL databases using `pg_dump` and `psql`, or operating as read-only replica of another database.
However, this proxy doesn't allow incoming traffic to be routed into a VM running locally on a laptop. For instance, you can't connect a local `PgAdmin4` to the PostgreSQL instance running inside your browser. So we forked the proxy and added nginx to provide a reverse proxy.

The reverse proxy listens on a range of ports, mapping each port to the VM at a specific private address you want to reach. The simple solution was to take the last 2 segments of the private IP and map it to a port number. So if your private IP is 10.5.6.123, you connect to the proxy on port 6123 and that points you to PostgreSQL (running on port 5432) on private IP 10.5.6.123. Shortened addresses are padded, so 10.5.6.2 maps to port 6002 and 10.5.6.44 to port 6044.

Sending commands to the emulator from the browser UI

Once the emulator is running, in the words of Fabian, “it runs as a black box process”, and the only way to communicate between the UI and the operating system running in the emulator is through `serial0_send` to send keystrokes over the emulated serial port, or through the `create_file` function which uploads a file to the running emulator.

To restart the network after restoring a state file, we need to run a short script to unbind and re-bind the emulated network card (so it gets a new mac address and thus a unique private IP address at the proxy), and then a standard network restart command. We could run this script by sending keystrokes to the emulator, but that's clumsy. What if the user is at a `psql` prompt? Then we'd send `\! script_command`. But if they're at an OS prompt, that won't work.

We solved the issue by creating a folder called `\inbox` and added a listener to the folder. When files arrive there, if the file ends in `.sh` (a shell script), we `chmod +x` that file, execute it, then delete it. That allows us to run commands “in the background” without upsetting the UI.

Just for fun

If you want to do something very cool, you can connect to another user's Postgres instance that they are running in their browser. Here are the steps:

Find a friend (sometimes the first step is the hardest)

Tell them to go to wasm.supabase.com

Tell them to “Start Network” and get their Network URL

On your own machine, open a terminal and run `psql postgres://postgres:woohoo@wasm.supabase.com:<PORT>`

And you're in! You're now connecting to a remote Postgres instance, inside a VM, inside a browser, from a terminal on your own computer.

Database Replication

Even more mind-blowing - it's possible to replicate data from an online PostgreSQL database (for example, a Supabase project) to PostgreSQL running inside your browser, and vice-versa using Postgres's logical replication. The steps are no different than any other Postgres database - just use the proxy URL as the target.

What's next?

For now, this is very experimental - but it has a lot of potential. If you want to get involved, please reach out to us or the team at Snaplet. The work they're doing over at Snaplet is incredible, and we've had a blast collaborating with them.

Snaplet repo: https://github.com/snaplet/postgres-wasm

Supabase fork: https://github.com/supabase-community/postgres-wasm

Share this article

Next post
Choosing a Postgres Primary Key
8 September 2022

postgres

On this page

Demo
Features
Why?
Repository Overview
Virtual Machine (VM)
Web application
Network proxy
Technical deep-dive
WASM
PostgreSQL 14 segfault errors
Optimizing startup time and image size
Networking
Sending commands to the emulator from the browser UI
Just for fun
What's next?

Related articles

Postgres WASM by Snaplet and Supabase
Choosing a Postgres Primary Key
Implementing "seen by" functionality with Postgres
Postgres Auditing in 150 lines of SQL
Protecting reserved roles with PostgreSQL Hooks

View all posts

Start your project

© Supabase Inc