EdgePipes | The Alternative to SSR and RSCs

If you've talked with me at any point over the past few years (in person, at conferences, or listened in anywhere someone unwisely chose to give me a platform) you've probably heard me mention "SSD" or "Server Side Data".

The basic idea is that instead of SSR (Server Side Rendering) we would hoist **just the fetch hooks and router** out of an SPA or MPA, and embed them in a SharedWorker in the browser or deploy them as an Edge Function.

Three design requirements are pre-requisites for this idea:

- the router and fetch hooks should be fully isolatable from any rendering requirements
- the fetch hooks should be non-blocking by design (waterfalls / linear requests are acceptable via chaining fetch hooks if required, but ideally some annotation may be used to differentiate those that block due to need vs due to ux considerations)
- the fetch process needs to be managed on the client

Unlike SSR and RSCs where the goal is to output the page and data in a state ready for the consumer to read, our goal with this architecture is purely to tackle optimizations around waterfalls, latency, and network reliability. The output artifact is not a page, but a streamable payload containing a fully replayable response for each request made by the fetch hooks.

This design is premised on a few key ideas:

1. The existence of "The Backbone Effect": Network connections are more stable, throughput is higher and latency is lower between the server running an edge function and the data-center hosting your backend than between the device and your backend.

2. That booting an edge-function with a simple optimized router and running the fetch hooks is significantly faster than the time it would take to await asset load on the client, boot the full application, begin routing and run all fetch-hooks in parallel there.

3. That we are able to manage network requests intelligently-enough in the client so that we can make use of the response from this edge-function.

4. That delivering N request responses through one pipe or via server-push is in aggregate faster and more reliable than sending N separate requests.

5. That the application has intelligent enough state and routing management to take advantage of this edge-function to optimistically prefetch data for pages the users might go to next, and can delegate this

responsibility off-thread.

As I've thought more on this idea I've started rebranding it as "EdgePipes" instead of SSD to better communicate what it does and where it differentiates.

What's different?

Unlike SSR, EdgePipes should be fairly resource-consumption friendly. Like SSR, EdgePipes can be used to progressively enhance the performance or experience of an application.

Unlike SSR, EdgePipes optimize page-loads even after the initial page.

Unlike SSR and RSCs, EdgePipes don't create a double-request or rehydration problem, which should lead to applications being *actually* interactive faster, not just appearing to be.

Unlike RSCs EdgePipes don't change your security model, make you think about where your servers are in relation to your database, or ask you to write your API in JS.

Unlike RSCs, EdgePipes actually make sense on the edge, because they don't need to be located close to your microservices and databases.

I also suspect that EdgePipes would be easy to hyper-optimize. Their restricted nature likely means they can be compiled into executables with the likes of bun and static hermes. They could potentially rewrite responses into more streamable and compressible forms for final delivery (if the plugin for doing so is paired with a plugin in the client that understands how to re-expand).

Since routes are often nested, the client could easily send a hint for which fetch-hooks it already has (and thus can be skipped).

Using it for prefetch doesn't even have to deliver the response all-the-way to the client: it could just prefetch to a cache on the edge so that the cost to the user's device is only paid if they navigate to the associated page, solving elegantly a common problem seen with sites that attempt to use prefetch aggressively for every link and button.

But Why?

I started thinking about this architecture while working at LinkedIn, where quite often we ran our SSR (fastboot) in a "data only" mode somewhat similar to this and for similar reasons. But (for lots of reasons) that implementation always left way more to be desired than it solved problems.

I've never liked SSR. Okay that might be a bit of a lie there was a short period when SSR first burst onto the scene and EmberJS added one of the first out-of-the-box SSR solutions with fastboot that I thought it was awesome. But the rose colored lenses faded to grey quickly: the real world evidence mostly showed that SSR for anything except pre-rendering html for static sites was both cost-prohibitive and net-negative to user experiences and performance. While I'd never considered it a silver bullet, I did at least think at the time it would *improve* user experiences and maybe lead to some interesting pre-render-in-a-worker-like concepts.

Deadends. Useful deadends mind you, but deadends. Useful because I really like the tooling and ideas in many new frameworks like Astro, the built-in SSR mode of vite, and the exploration of minimalism that lit and qwik and htmx are driving (and I don't think we fully get there without the failure of SSR as a performance fix).

But there were ideas in SSR I did like: the idea that we could multiplex all the requests associated to a given page visit. The idea that we could flatten waterfalls. The idea that we could leverage how the internet backbone prioritizes data. The idea that we could shift some compute closer to the user to boost their experience. The idea that we could do this progressively overtop of existing applications as little more than an opt-in enhancement without requiring teams to adopt a whole new paradigm or language.

And what I like most about the EdgePipe approach: it even should work for fairly lightweight minimal apps that do a lot of their work on the server.. because it is fundamentally around optimizing the fetch pipeline wherever one exists.

Anyhew, so those are EdgePipes and if you've been full of WTFs wondering what I meant talking about SSD these past few years now you know 💜