Photo by Mohamed Osama

Starting from scratch

thibault 11 Aug 2022 372 **6,791** views

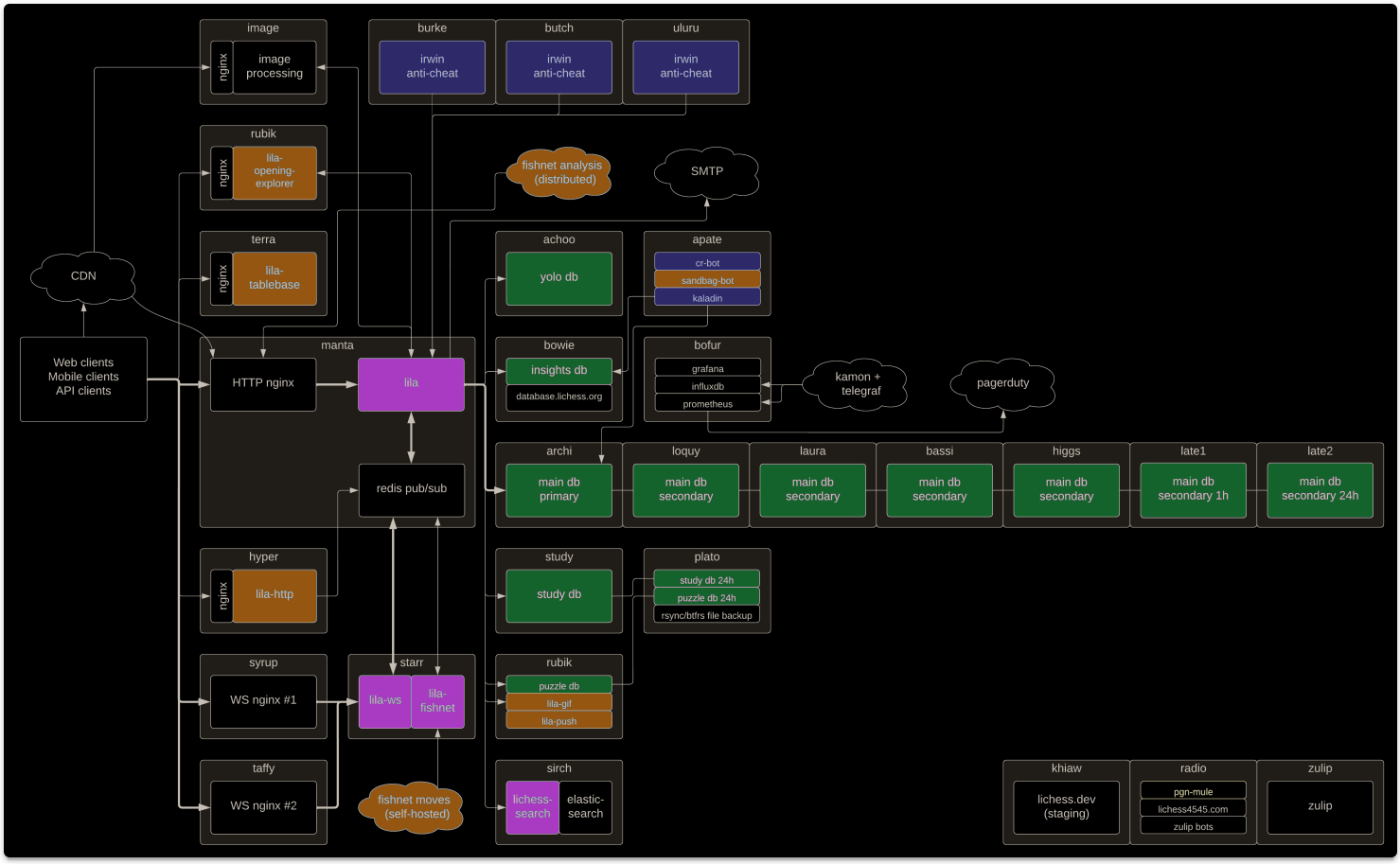
Lichess Software Development

If you had to start Lichess from scratch, what would you change?

This question was asked after the legendary talk given by @arex at Big Techday 22. It was suggested that I answer it in a tweet, but I don't think 280 chars will do.

Here are the most defining technical aspects of Lichess:

- Scala is the main programming language
- MongoDB is the main database storing our data
- Web frontend uses snabbdom and sass
- Lichess looks like a monolith (lila) with a bunch of satellite services
- All the code is open sourced as free software with the GPL license



These choices were made a while ago, at a time when Lichess was just a hobby project, and I didn't intend for it to become the home of millions of chess players around the world.

So, how relevant are these choices today, and if we were to go back in time while keeping the experience we accumulated, what would we do differently?

Scala as the main language

First, Lichess is not only made of scala. Since there are a bunch of different services communicating through Redis or HTTP, we can choose the best language for each service - and it's usually either

scala or rust - but it could be anything, really, depending on what the service does, and who works on it.

But lila, the core of Lichess, is a scala program. And I still think it's the correct choice, and I would still go for it. Lila is a big collection of relatively simple features in a cohesive package. For that, I want a programming language that:

- has **strong static typing**, so I can refactor to my heart content, without fear of breaking everything. And without having to write and maintain a bazillion tests.
- can do **functional programming**, so I can work with functions that compose
- is **expressive**, so I can succinctly describe what things are intended to be, without boilerplate and low level details
- **manages memory automatically**, so I can focus more on the problems to solve, and less on how the computer solves them
- **has a great ecosystem** so I can focus on the chess aspects, and reuse as much open source libraries as possible for everything else

Runtime performance is nice to have, but not of utmost importance for lila. Indeed, lila does A LOT of things, and most of them are not performance-sensitive. Ease of programming and maintenance is therefore more important than runtime performance.

For all these reasons I think scala is an excellent choice. There are only two other languages I would seriously consider for a lila rewrite:

Haskell

It does type safety and functional programming better than scala, which is a big deal. Back in the days, the JVM had a better ecosystem for building a chess web server, which is why Scala won. I would reconsider Haskell very seriously now.

Rust

It's great at static typing, and what it lacks in functional programming features, it makes up with other ways of making concurrent programming safe. It also runs ridiculously fast, but at the cost of putting the burden of memory management onto the developer. Even if rust provides excellent

compile-time tools to help with that, I just don't want to manage memory when building something as big as lila, because it can only take more effort than just not having to deal with it whatsoever.

Oh, and lila is built on the Play Framework. It made sense originally to speed up development with a cohesive and opinionated set of library that work together - a framework. But as years went and lila got bigger, it outgrew the framework, and we would now be better off with smaller independent libraries that we can swap as needed. So instead of a framework, I would now be looking at one library for HTTP, another one for routing, another one for HTTP forms, JSON, and so on.

MongoDB as the main database

Another choice that was made 10 years ago. MongoDB is serving us well, and I'm happy with it.
It's currently juggling with 5.5 billion games and about 15 billion other documents, and serving tens of thousands of queries per second. It's fast, compresses the data, and replicates seamlessly for redundancy and data safety. Its aggregation framework allows the complex data queries Lichess needs.

If we had to remake that choice, we would probably go for PostgreSQL, because it can also do all that, and it's released under an open-source license.

TypeScript/Snabbdom and Sass on the frontend

These are pretty great, but to be honest I haven't looked at the alternatives for a while. For the frontend I value speed and lightness, and snabbdom gives us that.
I would also go with TypeScript again, because it fixes JavaScript's main weakness, the absence of static typing, without adding a runtime that would make Lichess heavier and slower.
Sass is annoying but that's just because CSS is annoying.

The lila monolith

The core of lichess is a single monolith program that's deployed on a single server. It's both convenient and terrifying.

It's convenient because all the state is in one place, and can be cached in-heap for all modules of the site to use. Which makes it very efficient and quick at runtime. Everything compiles as a single unit, which ensures the entire site is coherent and free of incompatibilities.

It's terrifying because there's a big single point of failure. When lila goes down, everything goes down. Also it requires restarting the entire thing to update just one module.

We have a bunch of distinct services to handle specific parts of the website, like websockets, the opening explorer, the search engine, and more. While it's nice to be able to deploy them separately, without bringing the entire website down, the benefit is mitigated by the necessary networking between services. The network is, by definition, slow and unreliable, and each new service adds complexity and performance compromises, when compared to a simple, cohesive and in-memory monolith.

The current state of things is the result of years of evolution. There was no master plan, we just looked at monitoring to figure out what needed scaling. I like where it's at, but to be honest I'm biased by the fact it's the only large codebase I've been working on for years, so I lack experience with different ways of doing things, that I could compare it to.

Maybe being able to deploy lila over multiple servers, for resilience, would be nice? But then we'd lose the ability to cache things in-heap, because proper cache invalidation across several instances would be impossible. The performance and complexity implications have deterred me from trying it out so far.

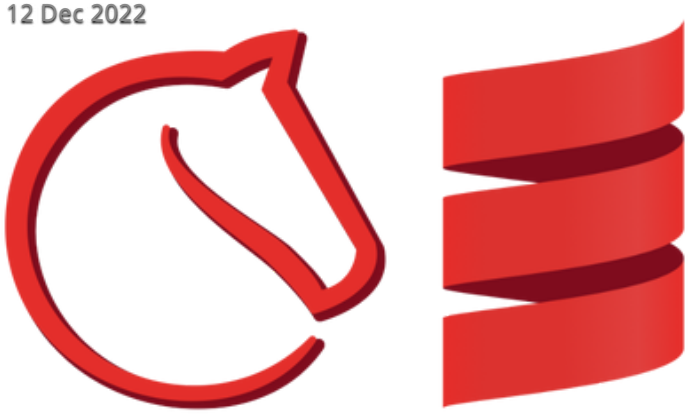
Open source all the things

The one best decision we've ever made, and I wouldn't have it any other way. Not only do we get contributions from the chess community, we also empower it to build new products with the help of our source code and open database.

DISCUSS THIS BLOG POST IN THE FORUM

More blog posts by thibault

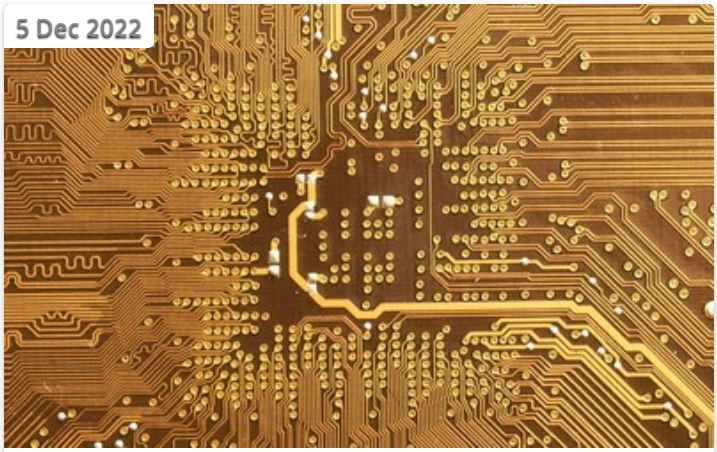
12 Dec 2022



Lichess & Scala 3

Lichess gets a big upgrade. It doesn't go as planned.

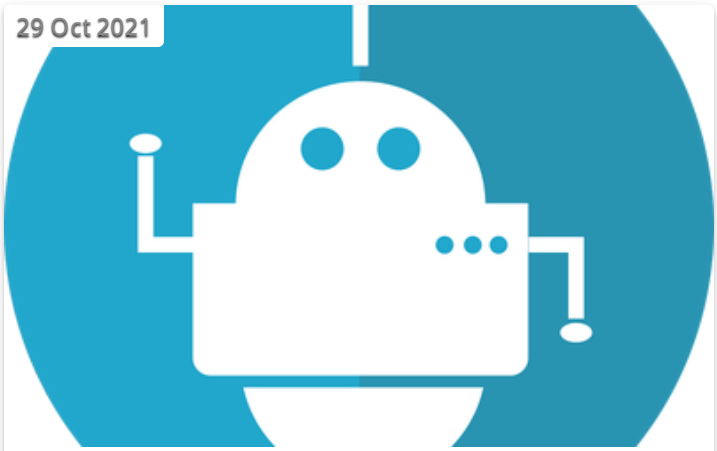
5 Dec 2022



Lichess on scala3 - help needed

A week after deploying Lichess rewritten with scala 3, I had to revert to the scala 2 version. I nee…

29 Oct 2021



How to create a Lichess bot

Well, it depends. On you.

1 Oct 2021

How to ask technical questions

Simple guidelines to get quick and actionable answers