

Mauricio Zaffari

Lead / Staff Software Engineer | Rails & AI Systems

Brazil · UTC-3 · Remote, with daily overlap across UK and US business hours

linkedin.com/in/mauriciozaffari · github.com/mauriciozaffari · rubygems.org/profiles/mauriciozaffari · mauriciozaffari@gmail.com

Eighteen years building Ruby on Rails systems, and still choosing Rails on purpose. Until September 2026 I led engineering for a SaaS analytics platform used by Disney, Warner Bros, Sony and NBCUniversal, where I finished as the top all-time contributor to the core codebase. I am now open to lead and staff engineering roles.

Most of my work looks like one of three things: making a slow system fast enough that people stop working around it, breaking a monolith apart without stopping the business, or being the person who decides what the architecture is going to be and then living with that decision. I have taken Rails applications from version 2 through version 8, which means I have migrated most of the mistakes the framework has since fixed.

I also build and run my own product, on my own. Inscripto is a multi-tenant sports event platform that handles real signups and real payments for karting and tennis leagues, which keeps me honest about the difference between a system that demos well and a system that survives a Saturday morning.

01 IMPACT

31%

TOP ALL-TIME CONTRIBUTOR

of every commit to Looper Insights' core analytics codebase across the company's history, and 81.7% of its 2025 commits

720,000+

GEM DOWNLOADS

for `mongoid_search`, the full-text search gem I created and maintained

R\$ 482,800+

PAYMENTS PROCESSED

across 109 sports events and 3,603 charges on Inscripto, the platform I built and still operate

18

YEARS SHIPPING RAILS

in production, across every major version from Rails 2 to Rails 8

99.1%

TEST COVERAGE

final coverage on the Looper Insights core analytics codebase when I left in September 2026, roughly 244,000 lines of Ruby

02
EXPERIENCE

Looper Insights

Lead Software Engineer

July 2018 – September 2026 · London, UK (remote)

Looper Insights is a global SaaS analytics platform for studios and streaming services. I was there across three stints: full-stack developer in 2018, senior developer from 2020, and tech lead from March 2022 until September 2026. I finished as the top all-time contributor to the core codebase, which serves Disney, Warner Bros, Sony and NBCUniversal and accounts for 55% of company revenue. I led the Webstores 2.0 replatforming and the company-wide authentication migration to AWS Cognito.

Inscripto

Solo Founder, Product Architect and Lead Engineer

2018 – present · Brazil

Inscripto is a multi-tenant sports event management and signup platform I built and still run as its only person — product, architecture, engineering and operations end to end — replacing paper signups and manual payments for karting and tennis leagues. It has handled 109 events, 3,603 charges and more than R\$ 482,800 in online payments. The stack is Rails 8, PostgreSQL, Hotwire, Tailwind and a PWA frontend, with Stripe, MercadoPago and PIX for payments, and it uses the Gemini API to parse unstructured WhatsApp match results into live leaderboards.

Develoz

Founder & Principal Engineer

October 2013 – present · Brazil

I founded Develoz in 2013. For thirteen years it has built software, integrated systems and automated processes for operations in Brazil and abroad. It now works on applied AI — finding where AI is worth using in a process, building the integration pilot, and defining the governance and human review around it. Earlier clients include ArtistSignal, New York Water Taxi and Explica Fácil.

EARLIER ROLES

Kinscape

Lead Software Developer

June 2018 – March 2020 · Remote

I led the technical modernisation of a private family social network, decoupling a legacy

Rails and AngularJS monolith into a Vue.js frontend running against a Rails JSON API.

StudioBridge

Lead Full-Stack Engineer

January 2018 – March 2018 · Alpharetta, GA, USA (remote)

I migrated the core subscription billing engine from Stripe to CardConnect across more than 100 commits, with zero downtime for active dance and music studios.

Atlantic Coast Conference

Senior Software Consultant

September 2017 – January 2018 · Charlotte, NC, USA (remote)

I built a dynamic multi-sport evaluation template engine in Node.js, AngularJS and PostgreSQL, used for NCAA conference officials and facility evaluations.

Habit

Senior Full-Stack Engineer

November 2016 – September 2017 · San Francisco Bay Area, CA, USA (remote)

I built kitchen operations and logistics software in Rails, React and PostgreSQL for a personalised nutrition startup, turning DNA and blood test results into daily meal prep and delivery workflows. The company was later acquired by Viome.

Decisiv

Senior Software Engineer

2016

I built a maintenance recommendation engine in Rails and MySQL for a commercial fleet service relationship management platform handling over 4 million annual events across 74,000 fleets.

Alldayeveryday

Senior Full-Stack Engineer

2015 – 2016

I cut digital magazine page load times from 5-10 seconds to under one second through query tuning and caching. I also integrated Chargebee billing and introduced the first RSpec test suite.

Everwise

Senior Full-Stack Engineer

January 2013 – November 2016 · San Francisco, CA, USA (remote)

Across two stints I co-built the third generation of the mentor-mentee matching engine in Rails and PostgreSQL, serving more than 60,000 relationships, and developed a reusable React UI component library. The company later merged with Torch.

Toptal

Senior Software Engineer

2012 – 2013

I passed the five-stage selection process, then co-built Toptal's own matching and project management platform.

Stack Builders, Bit Zesty and BlockShopper

Ruby on Rails Developer

2008 – 2012

Where my professional Rails career started, in 2008. I led MiMo, a PayPal-style payment application for Nigeria; built Feesability, a time tracking product on Rails 3, Mongoid and Resque; and built a real estate news CMS spanning more than 32 sites.

03 SELECTED WORK

Shipping the first AI features into a client platform

Looper Insights · 2026

PROBLEM

The company wanted AI features in the client-facing product, and had six engineers it trusted to figure out where AI actually pays for itself. Inside engineering, AI coding assistants were already changing how people worked, with no shared standards: every engineer invented their own workflow.

APPROACH

I was one of the six in the company's AI pilot, and I took the two halves of the problem — how we build with AI, and what we ship with AI.

For the first half, I authored the team's AGENTS.md workflow standard and shared it across the organisation, so assistant-driven development went from personal habits to a documented, reviewable way of working. I published model evaluations that informed the organisation's tool choice.

For the second half, I was delivery lead for the AI initiative on the core product: client-facing AI pages including one authenticated through a studio's own single sign-on, and the multi-tenant data-security architecture that lets an LLM chat feature answer from one client's data without another client's data ever being in reach. That isolation was designed before the feature was green-lit, not retrofitted after.

OUTCOME

Standards adopted organisation-wide; AI pages live in front of clients whose names everyone recognises; a chat architecture whose security model was reviewed before it shipped.

TECHNOLOGIES

Ruby on Rails · LLM integrations · ClickHouse

Cutting a 46-minute import to 16 minutes

Looper Insights · 2026

PROBLEM

Amazon US merchandising imports took around 46 minutes. Client deliveries queued behind them, so a single slow import delayed the reports people were actually waiting for. The volume was the hard part: roughly 273,000 title spots per scan, against 12.5 to 13 million titles processed weekly.

APPROACH

The bottleneck was not the fetching, it was the write path — every row was paying for durability and index maintenance it did not need while still in flight. I moved the intake into UNLOGGED PostgreSQL staging tables, so in-progress rows skip write-ahead logging entirely, then promoted them into the real tables once complete. Alongside that I parallelised the fetch stage and streamed results through Redis instead of accumulating them in process memory.

The staging tables are safe here specifically because the data is reconstructible: an UNLOGGED table does not survive a crash, and that is an acceptable trade when the recovery action is to re-run the import.

OUTCOME

46 minutes down to about 16 — a 65% reduction, confirmed independently by a colleague measuring the same pipeline. Client deliveries stopped queueing behind the import window.

TECHNOLOGIES

Ruby on Rails · PostgreSQL · Redis

A page that took four minutes, and the five reasons why

Looper Insights · 2026

PROBLEM

The main scan page took over four minutes to load. For one client it was worse: ten to fifteen minutes, which is long enough that people stop using the feature and ask someone else to send them a spreadsheet instead.

APPROACH

There was no single cause, which is the interesting part. Five separate things compounded: a `COUNT DISTINCT` fanning out over a join, a scope that scanned the whole table rather than the current request's slice, a sequential scan over 16.7 million rows, a join against a table of roughly 1.5 billion rows, and a missing cache on an expensive grouped aggregate.

I fixed them one at a time and measured after each, because compounding causes mask each other — fix the second-worst first and the improvement looks negligible, which tempts you into concluding it was not the problem. Two of the five needed denormalisation rather than query tuning: a boolean carried onto the row that needed it, so the billion-row join disappeared, and a cached JSONB aggregate refreshed by a dedicated worker instead of computed per request.

For the worst-affected client the fix was narrower and shipped the same day: an existence check that had been doing linear work became an $O(1)$ lookup.

OUTCOME

Four minutes to under a second. The client-specific path went from ten to fifteen minutes to effectively instant, delivered as a same-day hotfix. A related filter on the same page went from 322 seconds cold to 25.5 seconds by anchoring an unscoped subquery so it could use an index that already existed.

I later made the whole class of regression fail loudly, enabling `strict_loading` across the test suite so an accidental N+1 breaks CI instead of quietly reaching production.

TECHNOLOGIES

Ruby on Rails · PostgreSQL · ActiveRecord

Replatforming the core product without stopping it

Looper Insights · 2024 - 2025

PROBLEM

The core product had accumulated the usual decade of decisions: a React frontend embedded inside the Rails monolith, so the two could not be deployed or tested independently, and a large share of the codebase predating the patterns the team had since settled on. Rewriting it wholesale was not an option — it serves Disney, Warner Bros, Sony and NBCUniversal, and it earns 55% of company revenue.

APPROACH

I led the replatforming as a sequence of shippable steps rather than a branch that lived for six months. Over 261 commits across 844 files I rewrote more than 20% of a roughly 244,000-line codebase, raising test coverage as I went because the tests were what made the next step safe.

Separately I extracted the frontend — about 138,759 lines of React — out of the monolith into its own repository, which let the frontend and backend be built, tested and deployed on their own cadences.

OUTCOME

Shipped incrementally, with test coverage moving from 72% to 80% across the effort and continuing to climb afterwards. The product manager described the result as a transformational release. The frontend extraction is what made subsequent frontend work independent of backend release cycles.

TECHNOLOGIES

Ruby on Rails · React · PostgreSQL · RSpec

Sixty releases and one release button

Looper Insights · 2022 - 2026

PROBLEM

Releases were manual and tribal: whoever knew the steps ran them, and the steps lived in someone's head. The product serves Disney, Warner Bros, Sony and

NBCUniversal, so a bad release is not an internal inconvenience — it is a client-visible incident. Two repositories became three as the frontend was extracted, which tripled the ceremony.

APPROACH

I ran the release train end to end across the three repositories: a biweekly code freeze I instituted, QA gating, production deploys, hotfixes, and a rollback plan for every release. Then I turned the parts a machine can do into a machine doing them: an automated release workflow that builds, runs the quality gates, deploys and health-checks. The gate is nine linters plus the Cypress end-to-end suite wired into deploys and pull-request merges, so the checks that used to depend on a person remembering are now the pipeline refusing to continue. The test-plan and rollback-checklist templates I wrote became the ones PM and QA reproduced for every release after.

For one stretch I was also release manager for a frontend codebase I did not otherwise develop, because running the train is a skill apart from writing the code it ships.

OUTCOME

Sixty-plus releases and hotfixes shipped across four product naming eras without the release process itself as the bottleneck. Releases paused while I was on holiday — the strongest evidence I can offer both that the process worked and that I had not yet finished making it replace me.

TECHNOLOGIES

Ruby on Rails · RSpec · Cypress · GitHub Actions

Moving every user onto managed authentication

Looper Insights · 2021 - 2022

PROBLEM

Authentication was homegrown, and enterprise clients were starting to ask for things homegrown auth is bad at: federated identity, their own single sign-on, and token handling we could point at during a security review.

APPROACH

I led the company-wide migration to AWS Cognito over roughly two months: OIDC integration, bearer-token verification, refresh handling, and sessions kept in Redis with database persistence so a restart did not sign everyone out.

The part that takes the care is not the new system, it is the transition. Every existing user had to keep working throughout, which meant running both paths against real traffic before the legacy code could be deleted — and then actually deleting it, rather than leaving a dead branch to confuse the next reader.

OUTCOME

One managed identity provider for the whole platform, with client SSO becoming a configuration exercise rather than an engineering project. I later built client-facing pages authenticated through a studio's own single sign-on on top of it.

I led a small engineering team and never stopped writing the code. I remained the top contributor to the codebase I led, and that combination was deliberate rather than a failure to delegate: I reviewed code I could have written myself, and I did not hand down an architecture I was not willing to build. The leverage ran both ways — through the decisions other people inherit, and through the work I shipped.

I find the cause before I write the fix. A slow page usually has more than one reason for being slow, and the first one you find is rarely the expensive one. I profile, I read the query plan, and I fix the actual thing. One page I inherited took over four minutes to load for five compounding reasons; fixing them one at a time, with a measurement after each, got it under a second. Guessing would have fixed one of the five.

A regression test that passes before the fix is worthless. I have rejected my own tests for this. One seeded a blank value through an ActiveRecord write path that silently coerced it to `nil`, so the test never exercised the bug it claimed to cover. That is false confidence, and it is worse than no test, because it stops anyone from looking again.

Correctness belongs in the database when it can. Application-level uniqueness checks lose races. If a duplicate must never exist, I would rather deduplicate once and add the unique index, and let the system fail closed afterwards.

Code review is where standards actually propagate. I comment on test rigour and architectural shape, not formatting — a linter handles formatting. A principle I keep returning to: a controller with no matching model is a smell, because a state change on a record is almost always that record's update, not a new controller.

Releases need a rollback plan you have actually thought about. I owned release checklists: migration reversibility, queue drain, a recorded last-known-good reference. When our release tooling hit a merge pattern that made rebasing unsafe, I fixed the tool and wrote down why, so the next person meets a documented failure mode instead of a mystery.

I write things down. Architecture decisions, domain notes on the subsystems that surprise people, and the AI agent standards the organisations I have led work to. A decision that lives only in a Slack thread has to be re-litigated every time someone new asks.

I teach the system I lead. I built an internal training curriculum and ran the “Webstores Unlocked” sessions that walk the team through how each part of the platform works. Those recorded sessions became the onboarding material for new engineers — the ones I hired, onboarded and assessed.

mongoid_search

Creator and maintainer · 720,962 downloads

Full-text search for Mongoid, and the gem most Rails applications on MongoDB reached for before Atlas Search existed. It has passed 720,000 downloads.

I eventually transferred it to the official Mongoid organisation, which is the outcome I wanted: the gem outlived my own need for it and now sits where people looking for it will actually find it.

mongoid_activity

Creator · 5,216 downloads

Activity tracking for Mongoid documents, extracted from an application that needed an audit trail and a change feed without a second datastore.

Ruby and Rails ecosystem

Contributor

Patches and maintenance across gems I depended on in production, including Rails itself, `active_admin`, `active_model_serializers`, `cancan`, `carrierwave`, `administrate`, `blueprinter`, `cardconnect` and `cognito_token_verifier`.

Almost none of this was planned contribution. It is what happens when you hit a bug at work, find the cause in a dependency, and decide to fix it upstream instead of monkey-patching around it.

AI AND LLM SYSTEMS

Production LLM integrations · OpenAI · Anthropic Claude · Google Gemini · Async LLM pipelines · Agent workflow standards · Multi-tenant data isolation for LLM features

BACKEND AND ARCHITECTURE

Ruby on Rails (v2 to v8) · Ruby · REST APIs · GraphQL · Microservices · Sidekiq · ActiveJob · State machines · Domain modelling

DATABASES AND PERFORMANCE

PostgreSQL query tuning · UNLOGGED staging tables · Zero-downtime data migrations · MySQL · Redis · OpenSearch · Elasticsearch

DELIVERY AND TESTING

Docker · AWS (Cognito, ECS, S3) · GitHub Actions · RSpec · Capybara · Cypress · Multi-linter quality gates · Linux administration

ENGINEERING LEADERSHIP

Technical team leadership · Architecture governance · Code review · Hiring and interviewing · Release management · Mentoring · Agile and Shape Up

FRONTEND ENGINEERING

Hotwire (Turbo and Stimulus) · React · Vue.js · TypeScript · JavaScript · Tailwind CSS · Propshaft · esbuild · Progressive web apps

EDUCATION

Technologist Degree in Internet Systems	2011
Faculdade Dom Bosco	

Electronics Technician	2005
Instituto Anglicano Barão do Rio Branco	

Content last reviewed 5 October 2026

The current version of this document is at mauricio.zaffari.casa