A Server Called Mercury

June 2026

6 min read • 1,346 words

I bought a server this week. Not cloud credits, not a managed platform, not a serverless function bobbing in someone else's abstraction. A single rented box at Hetzner with four cores, eight gigs of RAM, and a name: **mercury**. By the end of the night it was running every site I care about, including this one.

I did it partly to keep costs down. Mostly, if I'm honest, I needed a hobby.

Ex-Herokai, present tense

I used to work at Heroku. That sentence still does a lot of load-bearing work in how I think about computing. Heroku taught a generation of us that deployment is not a chore that follows the real work. Deployment is the real work, or at least the moment the real work becomes real. `git push heroku main` was a complete thought. It was a promise the platform made to you: bring code, we'll handle the rest, and we will not make you think about it.

That promise shaped my whole career. It's why tools like Dokku matter so much to me: the open-source, single-server love letter to the Heroku experience that refused to let the dream die when the dream's parent company moved on to other quarterly priorities. Open source matters double here, because deployment tools are infrastructure for everything else you'll ever make. When they're proprietary, your entire creative output has a landlord.

I'll say the quiet part loudly: **deployment UX is some of the most important UX there is.** For real. The distance between "I made a thing" and "the thing is alive at a URL" is where most side projects go to die. Every additional step in that gap is a tax on the impulse to create: every YAML incantation, every dashboard safari, every "configure your ingress controller". Heroku understood this in 2009. Much of the industry has spent the years since adding the steps back and calling it progress.

Dokploy, and the shape of my brain

The box runs Dokploy, an open-source PaaS that is very much a descendant of that lineage: a dashboard, a git push, a deployed thing with a TLS certificate. Projects contain environments, environments contain apps and databases, Traefik quietly does the routing, and Let's Encrypt does the blessing.

> Mostly quietly. If you migrate DNS while Let's Encrypt can still see your old IPs, you can talk yourself into a rate-limit spiral where every retry extends the timeout. Ask me how I know. The fix was patience, which has never once been my first instinct.

The feature that sold me, though, is that Dokploy speaks **docker-compose natively**. Not as an import shim, not as a second-class citizen. A compose file in your repo is a first-class deployable unit. My photo site is a Django app with a web process, a Celery worker, and a Postgres database, and the entire production topology is one `compose.prod.yml` sitting in the repository next to the code it describes. That fits my brain in a way I find hard to overstate. The compose file is the truth. It's versioned, it's reviewable, it's the same shape locally and in production. Twenty years into this career I have learned that I will always, eventually, forget how my own infrastructure works, so the infrastructure had better be a text file in the repo, where I'll trip over it.

The part where the robot does the DNS

Here's the part that would have sounded like science fiction during my Heroku years: I didn't do most of the migration. Claude (Opus 4.8, running in Claude Code) did.

One app at a time, in conversation: it created the projects and applications through Dokploy's API, generated deploy keys and added them to GitHub, wired up webhooks so pushes auto-deploy, dumped my production Postgres from the old host and restored it on mercury, and (this is the part I keep thinking about) updated my DNS records directly through the DNSimple API. I'd say "let's do kjvstudy.org next" and a few minutes later there would be a verified checklist: DNS flipped, cert issued, health endpoint green, inventory documented in a git repo it was keeping about the server. When the photo site's Celery worker crash-looped because of a psycopg2-vs-psycopg3 mismatch, it read the stack trace, fixed the broker URL, and explained what had been wrong with my compose file. Everything worked really great. Not flawlessly, but recoverably, which is better than flawlessly because it means the process can survive contact with reality.

The whole migration, six sites and a database and object storage and certificates and autodeploys, was an evening. It used to take a team. I know, because I was on the team.

> First thing deployed, naturally, was an instance of httpbin. Some men plant a flag. I return a JSON representation of your request headers.

The box has started accumulating services the way a workshop accumulates jigs. home.kennethreitz.org runs Glance: health checks for everything, CPU and disk, my essay feed, an hourly mirror of my iCloud Drive, bookmarks to the admin panels. A status page for an audience of one.

git.kennethreitz.org runs Gitea, mirroring all three hundred or so of my GitHub repositories every eight hours. GitHub is still origin (for now). The mirror is insurance. Twenty years of work now also lives on a disk I pay for directly, and the repo that documents this server is hosted on the server it documents, which feels correct.

Analytics came home too. This site ran Gauges for years, and Gauges just got sold (yet again). A tracking script you chose has a way of becoming an asset somebody else acquired. It's Umami now: open source, no cookies, every site on the box reporting to one dashboard, the data sitting on my own disk.

Mostly harmless platforms

There is a theory in the modern cloud industry which states that if you can make a platform pleasant enough, developers will arrive in great numbers, and that once they have arrived in great numbers, the pleasantness can be carefully and progressively removed without anyone leaving. There is a second theory which states that this has already happened.

It goes roughly like this. In the beginning, a platform is built for you, the actual human with the half-finished idea at 11pm, because it has to be; delight is the only marketing a small platform can afford. The free tier is generous the way a first date is generous. And then, gradually, imperceptibly, the platform's true customer stops being you and becomes the spreadsheet. The platform begins optimizing for sustainability, which is a word that means almost exactly the opposite of whatever you wanted to do this weekend. The subsidies dry up. The free tier is "sunset," a lovely word for a thing that involves no sun and no setting, merely an email with an effective date. The dynos that once slept harmlessly now bill by the second. And one day you look up and realize the platform built for people like you is now a platform built for people unlike you, who are themselves being optimized for a third group of people who have never deployed anything and never will.

This is not a complaint, exactly. Spreadsheets get hungry; everybody's got to eat. Heroku was a miracle and miracles have operating costs, and I say that with the particular tenderness of someone who helped run the miracle and still loves the people who built it. But it does clarify a choice. You can run your software on platforms that are optimized for their own survival, and ride along as a rounding error in someone else's sustainability story. Or you can run it on a small, slightly absurd machine with a name like a Roman god and a monthly cost lower than lunch, built out of open-source tools made by people who clearly, evidently, want what you want. And when the next subsidy dries up somewhere out in the galaxy, you can read about it over coffee, with your towel within reach, while mercury hums along in a datacenter in Virginia, not optimizing for anything at all except being yours.

Don't panic. Self-host.