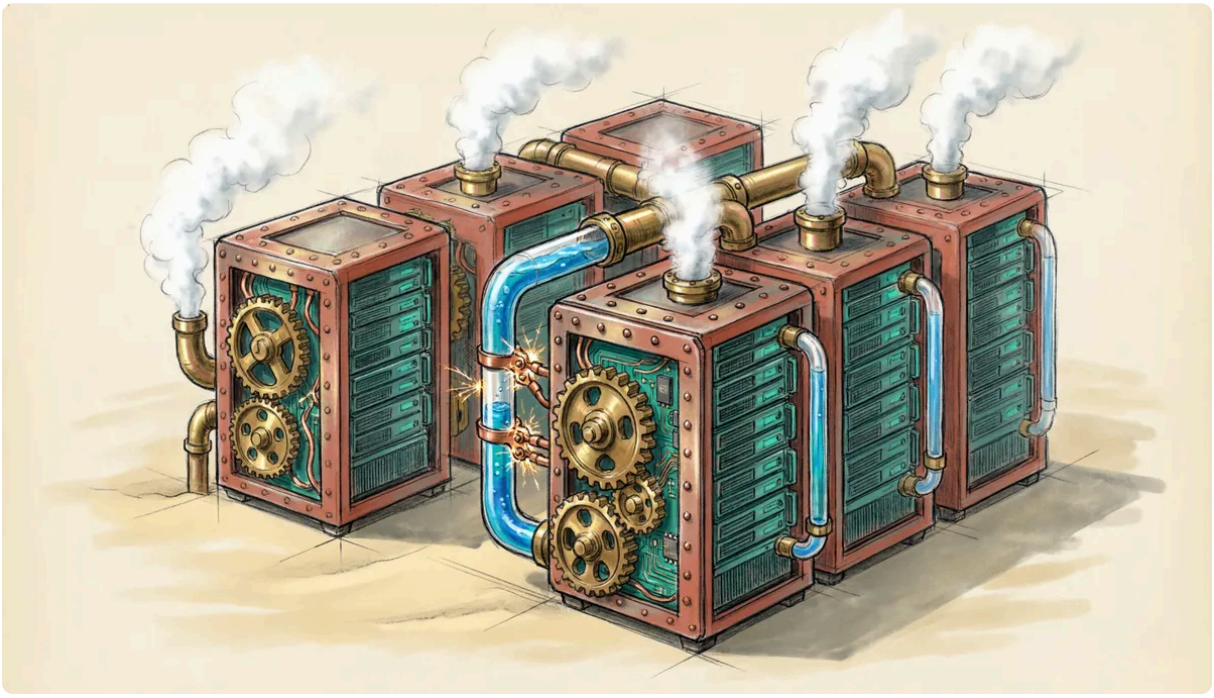
Why I built my blog with Zola and kept everything in one repo

Emil Lindfors | 2025-01-27
2026-02-11



This site is a static blog that does more than most static blogs. Every post has a downloadable PDF. Citations link to formatted references with DOIs. There's a newsletter powered by a Rust Cloudflare Worker talking to a self-hosted mail server. Client-side search works without any external service. Light and dark themes respect your system preference. The fonts are self-hosted. There are no JavaScript frameworks, no build tools beyond what ships with the static site generator, and no third-party services besides Cloudflare for hosting.

Everything lives in one git repo. Push, and it's live.

This post covers the foundation: why Zola, how the site is structured, and the design decisions that make the rest possible. The more interesting pieces – the newsletter, citations, and PDF generation – each have their own deep-dive.

Why Zola

I evaluated Hugo, Astro, Next.js, and Zola. The decision came down to a few things.

Single binary, zero dependencies. Zola is one executable. No Node.js, no Ruby, no Python, no package.json, no `node_modules`. I `cargo install zola` and I'm done. On a new machine, I can build my site in under a minute from a clean checkout.

Hugo also has this property. Astro and Next.js do not – they bring the entire Node ecosystem along, which means version management, lockfiles, and the occasional `npm audit` nightmare.

Batteries included. Zola ships with Sass compilation, syntax highlighting (via syntect), Atom/RSS feed generation, and a search index builder (elasticlunr). These are features I'd otherwise need plugins or build steps for.

The search index is interesting. Zola generates a JavaScript file (`search_index.en.js`) that sets `window.searchIndex` – a pre-built elasticlunr index of all my content. My search page loads this index and queries it client-side. No Algolia, no server, no API calls. For a personal blog with a few dozen posts, this is the right level of complexity.

Tera templates. Zola uses Tera, a Jinja2-like template engine. If you've used Django, Flask, or Ansible, Tera feels familiar. Importantly, it has:

- Macros with arguments (so I can write `{{ macros::format_reference(ref=ref) }}`)
- Template inheritance (`{% extends "base.html" %}`)
- `set_global` for variables that escape loop scope (a Tera-specific feature I use heavily for related posts)

Fast. Full site build is sub-second. Zola is written in Rust, and it shows. I never wait for builds. The dev server (`zola serve`) reloads instantly on save.

TOML frontmatter. Zola uses `+++` delimiters with TOML instead of `---` with YAML. This matters because TOML supports inline arrays and tables natively – my citation data is an array of structured objects in frontmatter, and TOML handles this cleanly where YAML would be painful.

The repo structure

```
lindfors-site/
├── content/
│   └── blog/
│       └── my-post/
│           └── index.md        # Blog post (TOML frontmatter + markdown)
├── templates/
│   ├── base.html               # Layout shell
│   ├── page.html               # Blog post template
│   ├── index.html              # Homepage
│   ├── search.html             # Client-side search
```

```
|   ├── macros.html            # Citation formatting
|   ├── shortcodes/
|   |   └── reference.html     # Inline citation shortcode
|   └── pdf/
|       └── academic.typ       # Typst template for blog PDFs
├── sass/
|   └── style.scss             # All styles (~1500 lines, both themes)
├── static/
|   ├── fonts/                 # Self-hosted Inter + Literata
|   ├── pdf/                   # Generated blog post PDFs
|   └── newsletter/            # Generated newsletter markdown
├── api/
|   └── src/lib.rs             # Rust Cloudflare Worker (newsletter API)
├── scripts/
|   ├── generate-pdf.sh        # Blog → Typst → PDF
|   ├── generate-newsletter.sh # Blog → newsletter markdown
|   └── send-newsletter.sh     # Send via Worker API
├── cv.typ                     # CV in Typst
├── build.sh                   # Full build pipeline
├── deploy.sh                  # Build + push
└── zola.toml                  # Site config
```

One repo. The blog content, templates, styles, Worker API, build scripts, CV source, and font files all live together. `deploy.sh` processes citations, generates PDFs, builds the site, and pushes – one command.

Typography and fonts

I spent more time on font selection than I'd like to admit. The site uses two typefaces:

- **Literata** for body text – a serif face designed for long-form reading, originally created for Google Play Books. It has a generous x-height and comfortable spacing that works well at body sizes.
- **Inter** for headings, navigation, and UI elements – a sans-serif designed for screens. The contrast between serif body and sans-serif headings creates visual hierarchy without needing size differences alone.

Both are self-hosted. No Google Fonts CDN, no third-party requests. The fonts are served from `/fonts/` as variable woff2 files (Inter) and subset woff2 files (Literata).

For Literata, I converted the TTF source files to woff2 with Latin subsetting to reduce file size:

```
uvx --from fonttools --with brotli pyftsubset \
  Literata-Regular.ttf \
  --output-file=Literata-Regular.woff2 \
```

```
  --flavor=woff2 \
  --layout-features='*' \
  --
unicodes='U+0000-00FF,U+0131,U+0152-0153,U+02BB-02BC,U+02C6,U+02DA,U+02DC,U+2000-206F,U+2074,U+20AC,U+2122,U+2191,U+219
```

The font declarations are inlined in `<style>` in the `<head>` rather than in an external stylesheet. This avoids a flash of unstyled text on first load – the browser discovers the font declarations immediately, before any external CSS is fetched.

The dual theme

The site has light and dark themes. Light is "Sandy Shore" – warm beige backgrounds with deep navy text. Dark is "Deep Ocean" – dark blue-green backgrounds with light text. Both use coral and teal as accent colors.

The implementation is pure CSS custom properties with a `data-theme` attribute on `<html>`:

```
:root {
    --color-bg: #F0EAE0;          // Sandy Shore
    --color-text: #1C3240;        // Deep Sea
    --color-link: #D4706A;        // Coral
    --color-accent-secondary: #2A8F82; // Teal
}

[data-theme="dark"] {
    --color-bg: #0E1A20;          // Deep Ocean
    --color-text: #E8F0F0;
    --color-link: #F2A07B;        // Warm Coral
    --color-accent-secondary: #4DD4AC; // Bright Teal
}
```

Every color in the stylesheet references these variables. Theme switching is a single `setAttribute` call plus a `localStorage.setItem` to persist the choice. On load, the script checks `localStorage` first, then falls back to `prefers-color-scheme`. Zero flicker because the script runs synchronously in the `<body>`, before paint.

The blog post template

Each blog post renders with a two-column layout: main content on the left, sidebar on the right. The sidebar is sticky and contains:

- **Table of contents** with scroll-spy highlighting (the current section updates as you scroll)
- **Post metadata** (published date, reading time)
- **Action buttons** – PDF download, cite (with BibTeX/APA modal), share (copies URL)

- **Tags** linking to the taxonomy pages
- **Author card** with photo and links

The reading progress bar at the top of the page is a `<div>` whose width is set as a percentage of scroll progress. Scroll spy and progress bar share a single passive scroll listener:

```
window.addEventListener('scroll', function() {
    updateActiveLink();
    updateProgress();
}, { passive: true });
```

The citation modal lets readers export a BibTeX or APA citation of *my* post – including a generated `@article` key, the post URL, and today's date as the access date. This makes blog posts citable in academic writing, which matters for my audience.

What Zola doesn't do

Zola is deliberately minimal. It doesn't have:

- **Asset processing** – No image optimization, no CSS minification, no JS bundling. I don't need these for a personal blog. If I did, I'd add a build step.
- **Client-side routing** – It generates static HTML. Each page is a full document. This is fine; my pages load in milliseconds.
- **A plugin system** – Unlike Hugo or Gatsby, there's no plugin ecosystem. Extensions are shell scripts that preprocess content before Zola sees it. This is both a limitation and a feature – my build pipeline is transparent and debuggable.
- **Newsletter/email/CMS features** – These are all handled by other components in the repo.

The lack of a plugin system is actually what led me to the architecture I have. Since Zola can't be extended internally, I built external tools that process content *before* the build. Citations are resolved to TOML frontmatter. PDFs are generated from the same markdown. Newsletter content is extracted into a separate file. Zola just sees clean markdown and structured data.

The extended pipeline

The interesting parts of this site aren't in Zola itself – they're in what happens around it:

Newsletter

A Rust Cloudflare Worker handles subscribe, unsubscribe, and sending. It talks to my self-hosted Stalwart mail server, which manages the mailing list as a principal with

`externalMembers`. No database, no Mailchimp, no third-party email service. The Worker renders newsletter markdown to HTML with pulldown-cmark and sends via JMAP.

Deep-dive: **I replaced Mailchimp with a Rust Worker and a self-hosted mail server**

Citations

A Rust CLI tool (zotero-cite) reads from my Zotero library and transforms `@citekey` references into linked citations with structured bibliographic data in TOML frontmatter. Tera macros render the references differently for HTML (with DOI links and semantic markup) and the template supports article, book, conference paper, thesis, and generic formats.

Deep-dive: **Proper citations on a static site: Zotero, Rust, and Typst**

PDF generation

Every blog post gets a downloadable PDF generated with Typst. A shell script extracts the markdown body, preprocesses it (stripping HTML citations, converting shortcodes), and feeds it to Typst via the cmarker package. The academic template uses the same fonts and colors as the website. The CV is also Typst, also in the repo, also auto-compiled on deploy.

Deep-dive: **I use Typst to generate PDFs of my blog posts and my CV**

The build

Everything runs in sequence:

```
# 1. Resolve @citekey references → TOML frontmatter
zotero-cite process ...

# 2. Compile CV
typst compile cv.typ static/cv.pdf

# 3. Generate blog post PDFs
./scripts/generate-pdf.sh ...

# 4. Build static site
zola build
```

Citations must run first (PDFs need resolved references). PDFs must run before Zola (they go in `static/`). Zola runs last and copies everything to `public/`. Total build time: a few seconds.

The cost

Component	Monthly cost
Cloudflare Pages (hosting)	$0
Cloudflare Worker (newsletter API)	$0
Stalwart mail server (shared VPS)	~$5
Domain	~$0.83

Under $6/month for a blog with a newsletter, PDF generation, client-side search, self-hosted fonts, and no external service dependencies. The VPS runs other things too, so the marginal cost of the mail server is close to zero.

What I'd do differently

Start with the typography. I picked colors first and fonts second. It should be the other way around – the font choice constrains everything else. Literata's warm character pushed me toward the sandy beige palette, but I arrived there by accident after trying three other color schemes.

Skip the custom theme toggle initially. System preference detection (`prefers-color-scheme`) is good enough for launch. I spent a day on the toggle button, localStorage persistence, and avoiding flash-of-wrong-theme. Ship with system-only first; add the toggle when someone asks.

Use page bundles from the start. Zola supports both `blog/my-post.md` and `blog/my-post/index.md`. The directory form lets you colocate assets (images, data files) with the post. I started with flat files and had to migrate later.

The stack

Role	Tool
Static site generator	Zola
Hosting	Cloudflare Pages
Newsletter API	Rust Cloudflare Worker (workers-rs)
Mail server	Stalwart
PDF generation	Typst
Citation processing	zotero-cite
Body font	Literata
Heading font	Inter
Search	elasticlunr (client-side, Zola-generated index)
Math rendering	KaTeX (web), MiTeX (PDF)
Styles	Sass (compiled by Zola)

Everything is text files in a git repo. No CMS, no database, no admin panel. I write markdown in a text editor, run a deploy script, and the site updates. The tools are all either Rust binaries or shell scripts. It compiles fast, it's cheap to host, and I understand every line.

The source is on GitHub.