I replaced Mailchimp with a Rust Worker and a self-hosted mail server

Emil Lindfors | February 10, 2026

I wanted a newsletter for my blog. The requirements were simple: let people subscribe, send them posts when I publish, let them unsubscribe. That's it.

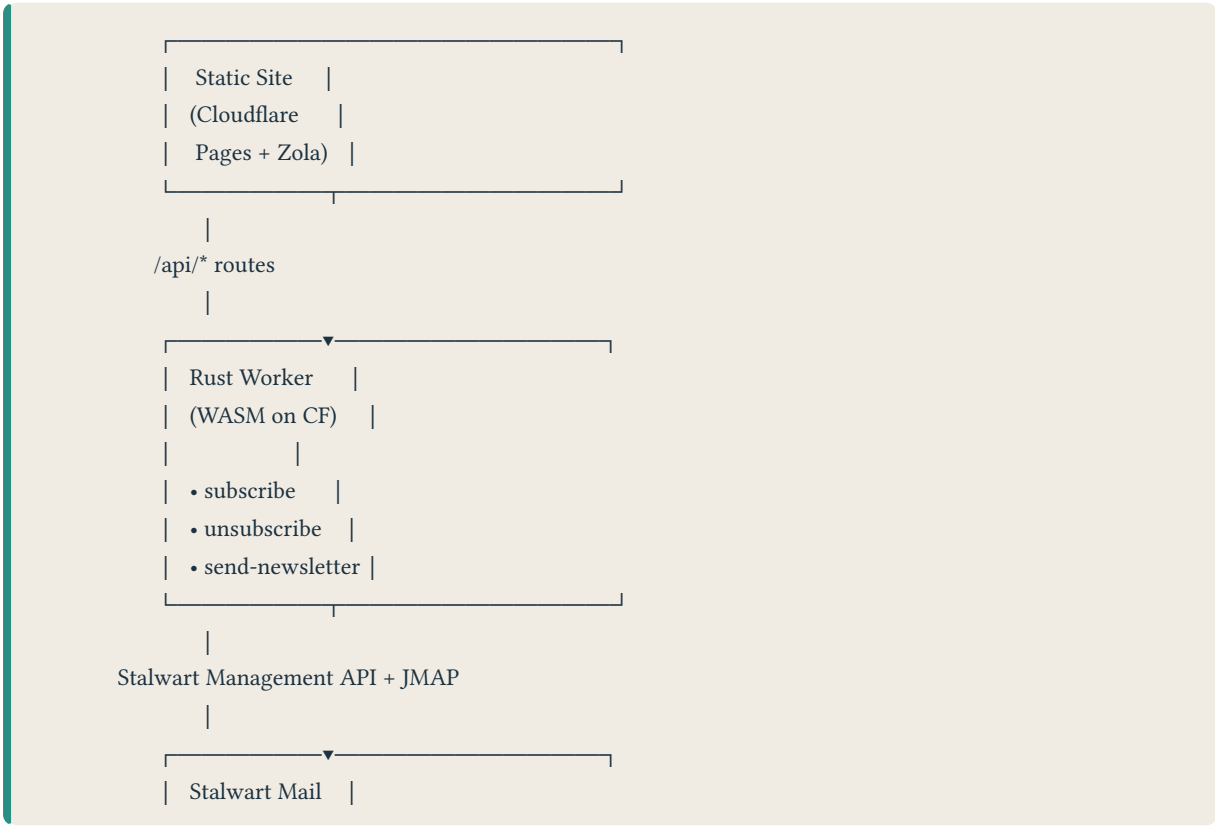
Every newsletter platform I looked at wanted $10-30/month, required me to hand over my subscriber list, injected their branding, and came with dashboards I'd never use. For a personal blog that publishes once or twice a month, this felt absurd.

So I built my own. The entire system is ~700 lines of Rust, runs on Cloudflare's free tier, and uses my existing self-hosted mail server for delivery. Monthly cost: $0 on top of infrastructure I already had.

Here's how it works.

The architecture

The setup has three components:



```
|  Server (VPS)    |
|                  |
|  • Mailing list  |
|    principal     |
|  • JMAP sending  |
|  • Fan-out to    |
|    subscribers   |
└──────────────────┘
```

1. **Zola static site** on Cloudflare Pages – generates the blog, serves HTML
2. **Rust Cloudflare Worker** – handles `/api/*` routes for subscribe, unsubscribe, and sending
3. **Stalwart mail server** – self-hosted on a VPS, acts as both the subscriber store and the delivery engine

The key insight is that **Stalwart's mailing list eliminates the need for a database entirely**. A Stalwart mailing list principal has an `externalMembers` field – an array of email addresses. That array is my subscriber list. When I send an email to `newsletter@lindfors.no`, Stalwart fans it out to every address in that array. No database. No subscriber table. No sync jobs.

The Cloudflare Worker

The worker is built with workers-rs, the Rust SDK for Cloudflare Workers. It compiles to WASM and runs on Cloudflare's edge network.

The `Cargo.toml` is minimal:

```
[dependencies]
worker = "0.7"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
pulldown-cmark = { version = "0.12", default-features = false, features = ["html"] }
```

Four dependencies. `pulldown-cmark` is there because the worker renders newsletter markdown to HTML at send time.

Subscribe

When someone enters their email on my site, the form posts to `/api/subscribe`. The worker validates the email and calls Stalwart's management API:

```
#[derive(Serialize)]
struct StalwartPatchOp {
    action: &'static str,
```

```
    field: &'static str,
    value: String,
}

async fn handle_subscribe(mut req: Request, ctx: RouteContext<()>) -> Result<Response> {
    let body: SubscribeRequest = req.json().await?;
    let email = body.email.trim().to_lowercase();

    // ... validation ...

    let ops = [StalwartPatchOp {
        action: "addItem",
        field: "externalMembers",
        value: email,
    }];

    stalwart_patch(&api_url, &api_key, &list_id, &ops).await?;
    // ...
}
```

That's it. One PATCH request to Stalwart's `/api/principal/{list_id}` endpoint with an `addItem` operation on `externalMembers`. The subscriber is added to the mailing list. No confirmation email, no double opt-in dance (I should probably add that eventually).

Unsubscribe is the same thing but with `removeItem`:

```
let ops = [StalwartPatchOp {
    action: "removeItem",
    field: "externalMembers",
    value: email,
}];
```

Sending newsletters via JMAP

This is where it gets interesting. Instead of SMTP, I use JMAP (JSON Meta Application Protocol) to send emails. JMAP is a modern, stateful, JSON-based protocol designed to replace the IMAP/SMTP combo. Stalwart has full JMAP support.

Sending a newsletter is a single JMAP request with two method calls batched together:

```
let body = serde_json::json!({
    "using": [
        "urn:ietf:params:jmap:core",
        "urn:ietf:params:jmap:mail",
        "urn:ietf:params:jmap:submission"
    ],
    "methodCalls": [
        ["Email/set", {
```

```
        "accountId": account_id,
        "create": {
          "draft": {
            "from": [{ "name": "Emil Lindfors", "email": from }],
            "to": [{ "email": "newsletter@lindfors.no" }],
            "subject": subject,
            "header:List-Unsubscribe:asRaw":
              " <https://lindfors.no/api/unsubscribe>",
            "header:List-Unsubscribe-Post:asRaw":
              " List-Unsubscribe=One-Click",
            "htmlBody": [{ "partId": "html", "type": "text/html" }],
            "bodyValues": {
              "html": { "value": html_body }
            }
          }
        }
      }, "0"],
      ["EmailSubmission/set", {
        "accountId": account_id,
        "create": {
          "send": {
            "identityId": identity_id,
            "emailId": "#draft",
            "envelope": {
              "mailFrom": { "email": from },
              "rcptTo": [{ "email": "newsletter@lindfors.no" }]
            }
          }
        },
        "onSuccessDestroyEmail": ["#send"]
      }, "1"]
    ]
});
```

The first call (`Email/set`) creates the email as a draft. The second (`EmailSubmission/set`) submits it for delivery, referencing the draft with `#draft`. The `onSuccessDestroyEmail` cleans up the draft after sending. All in one HTTP request.

The email goes to `newsletter@lindfors.no` – the mailing list address. Stalwart handles fan-out to all subscribers. I never iterate over subscribers or send individual emails.

Note the `List-Unsubscribe` and `List-Unsubscribe-Post` headers. These are important for deliverability – they tell email clients to show a native "Unsubscribe" button rather than flagging the email as spam.

The Stalwart side

Stalwart is a mail server written in Rust. I run it on a small VPS. It handles SMTP, IMAP, JMAP, and has a management API.

The only Stalwart configuration specific to newsletters is a mailing list principal. In Stalwart, a "principal" is any entity – user, group, or mailing list. A mailing list principal has:

- An email address (`newsletter@lindfors.no`)
- A list of `externalMembers` (the subscriber email addresses)

When Stalwart receives an email addressed to the list, it delivers a copy to each external member. That's the entire delivery mechanism.

I created the list principal through Stalwart's admin interface. No special configuration files. The worker manages members through the management API.

The send workflow

I write blog posts in markdown with Zola. When I want to send a post as a newsletter:

1. Generate the newsletter markdown:

```
./scripts/generate-newsletter.sh content/blog/my-post/index.md
```

This script extracts the post body, strips Zola-specific shortcodes (KaTeX math blocks, figure shortcodes), and writes a clean markdown file to `static/newsletter/my-post.md` with YAML frontmatter:

```
---
title: "My Post Title"
date: "2026-02-10"
description: "Post description"
url: "https://lindfors.no/blog/my-post/"
---
```

2. Deploy the site so the `.md` file is accessible at `https://lindfors.no/newsletter/my-post.md` .

3. Send:

```
./scripts/send-newsletter.sh my-post
```

This script reads the admin key from `.env` and calls the worker:

```
curl -s -X POST "https://lindfors.no/api/send-newsletter?key=$ADMIN_KEY" \
  -H 'Content-Type: application/json' \
  -d '{"slug":"my-post"}'
```

The worker fetches the markdown from my site, parses the frontmatter, renders the body to HTML with `pulldown-cmark` , wraps it in an email template, and sends it via JMAP.

The email template is inline in the worker – hardcoded HTML with inline styles (because email clients). No template engine, no CSS framework. It looks clean and renders consistently across Gmail, Apple Mail, and Outlook.

What this costs

Component	Cost
Cloudflare Pages	Free
Cloudflare Worker	Free (100k requests/day)
Stalwart on VPS	~$5/month (shared with other services)
Domain	~$10/year

Compare this to Mailchimp ($13/month for 500 subscribers), $ConvertKit$ (29/month), or Substack (10% of paid subscriptions). For a personal blog newsletter, the economics aren't even close.

What's missing (honestly)

This is not a replacement for Mailchimp if you need marketing features. Things I don't have:

- **Double opt-in** – I should add a confirmation email flow. Right now subscribing is single opt-in.
- **Analytics** – No open tracking, no click tracking. I genuinely don't care about this, but you might.
- **Bounce handling** – Stalwart handles bounces at the SMTP level, but I don't automatically remove bouncing addresses from the list.
- **Pretty email editor** – I write markdown. The template is hardcoded Rust. This is a feature, not a bug.
- **Scheduling** – I run a shell script when I want to send. No scheduled sends.

For a personal blog with a handful of subscribers who actually want to read what I write, none of these are real problems.

Workers-rs tips

A few things I learned building this:

Route order matters. In workers-rs, register GET routes before POST routes for the same path. I had the unsubscribe GET page and POST handler on the same `/api/unsubscribe` path and the order of registration determined which matched.

Headers aren't mutable the way you'd expect. `Headers::new()` methods take `&self`, not `&mut self`. You don't need `let mut headers`.

No `.url()` on RouteContext. If you need the request URL (for query parameters), use `req.url()?` rather than trying to get it from the route context.

Release profile matters for WASM size. I use aggressive optimization:

```
[profile.release]
lto = true
strip = true
codegen-units = 1
```

Why not just use Substack?

Substack is free for free newsletters. Here's why I didn't:

1. **I already have a blog.** My posts are markdown in a git repo, rendered by Zola, deployed to Cloudflare. I don't want a second copy of my content on someone else's platform.
2. **I already run a mail server.** Stalwart was set up for personal email. The newsletter is just one mailing list principal.
3. **Ownership.** My subscriber list is a JSON array in my mail server. I can export it with one API call. No platform lock-in, no "download your data" request, no worrying about a service shutting down or changing terms.
4. **It's a fun project.** The whole thing took an afternoon. It's 700 lines of Rust that I fully understand and can modify. That has value.

The newsletter stack

- **API**: workers-rs (Rust, compiled to WASM)
- **Mail server**: Stalwart (Rust) on a VPS
- **Protocol**: JMAP for sending, Management API for subscriber management
- **Markdown rendering**: pulldown-cmark (in the worker, at send time)
- **DNS**: Cloudflare (SPF, DKIM, DMARC records)

The code for the worker is on GitHub. If you're running Stalwart and want to try this, the setup is straightforward – a worker, a mailing list principal, and a few environment variables.

This post is part of a series on the infrastructure behind this blog. See also: Site overview, Citations, Typst PDF generation, Images.