We Built a Custom Workflow with the Buffer API — and Tripled Our X Impressions

Until March of this year, about 42% of people clicking through from our social posts were landing on expired or closed job listings.

I run QualityAssuranceJobs.com, a site that aggregates software testing roles and promotes them on social media. And for someone actively looking for work in a tough market, that's a frustrating dead end. For us — a community built around getting QA jobs in front of the right people as fast as possible — it meant our social distribution was working against us.

The whole point is speed: get a fresh listing in front of someone before the role closes. But we'd outgrown the workflow that was supposed to make that happen, and the gap between "job goes live" and "job hits social" had grown wide enough to have a measurable impact on our community.

So we built a workflow to fix it: here's what happened to our traffic when we did.

We needed a change to our old setup

For a long time, our social distribution ran through Zapier, and it did the job. Zapier makes a fantastic product, and I have no idea how they stay on top of the sheer number of integrations they support.

But our particular workflow had three moving parts, each introducing its own delay. It started with our publishing platform’s RSS feed, which was never designed to be a real-time tap into a live job board. Updates came when they came. Zapier polled that feed, which added more lag. And the Zapier-to-Buffer connection, our only link to social, added more on top of that.

Sometimes, more than an hour would pass between a job going live and hitting social. In the job market, an hour can mean the difference between a listing still being open and someone else already having the role. On top of that, I was spending close to 10 hours a week checking, rechecking, and chasing down misses. Things kept slipping through, and that’s how we got to 42% of our social clicks landing on expired listings

and the very real people who needed them hitting dead ends.

Something had to change.

If anyone, especially a seasoned software engineer, tells you to reach for code as your first tool, don't trust them. It makes things unnecessarily complicated when there are enough ways to get over the initial hurdle of bringing your ideas to life.

I didn't start this project because I wanted to build a custom automation platform. I started it because we kept hitting the same wall: *why can't I just get this to behave slightly differently?* When you're asking that question every other day, that's usually the sign that you need something you own.

We needed three things: speed, reliability, and the ability to see exactly what was happening and why. We also needed to know what was going out, when, and whether it actually worked. Off-the-shelf tools had gotten us far, but this particular problem needed something more custom.

I chose Clojure because it's the programming language that best reflects how I think: small, composable parts moving data through clear pipelines. And before anyone assumes a small community of users means a janky ecosystem, it doesn't. Every library I pull in has been built by people who think carefully about software, and the whole thing runs on the JVM, one of the most battle-tested platforms around.

But the language is secondary. The principle is what matters: poll for new content, deduplicate against what you've already posted, generate a payload, and send it to an API. Any language can do that.

How the new pipeline works

We call the system we built **Daedalus** internally, and the way it works is straightforward.

1. Job listings are automatically imported into our publishing platform as drafts. A human reviews each one and publishes it; an editorial step stays manual on purpose. We want a person deciding what’s worth promoting.
2. From there, Daedalus polls for newly published jobs every ten minutes, from 8 AM to 10 PM Eastern. We’ve tried different timing windows, and this is the sweet spot: it catches everything during active hours without overengineering posting.
3. When it finds something new, it checks against a SQLite table before doing anything else. The schema is about as simple as it gets — a job ID and a timestamp. If we’ve already posted it, we skip it. That’s what keeps the

system stateful and prevents duplicate posts even if the same job gets picked up twice.

The post payload is intentionally simple: job title, company name, location, and a link. Something like “Quality Assurance/Tester (REMOTE) – Koniag Government Services – Heart Butte, Montana, United States” followed by the URL.

We keep it stripped down because, when people are putting together which jobs are worth applying to, they often need speed first before doubling down on applications. With the right information immediately presented, they can decide faster what’s worth their time.

That payload goes to Buffer through the GraphQL API using the createPost mutation with addToQueue mode and automatic scheduling. Buffer handles the distribution timing and sends it out to X and Bluesky.

One note: X has been generating great link cards from our URLs automatically, which gives posts a visual element without any extra work on our end. Bluesky has been giving us some trouble with generated attachments lately, so that’s on my to-do list.

What changed after the switch

The switchover happened on March 19, 2026. That’s when Daedalus took over social distribution through the Buffer API on a consistent schedule, and we started measuring.

The numbers tell a more interesting story than a clean before-and-after anyway.

The clearest win was on X. Impressions increased by 200.4% in the weeks after the switch compared to the same window before it, and likes went up by 70%. On the site side, job-page views increased by 18.8%, overall sessions rose by 11.4%, and active users grew by 13.6%. For a niche job board, that’s meaningful traffic going to actual listings instead of dead links.

Some metrics moved in the other direction. Overall engagements dropped by 15.2%, new followers fell by 41.7%, and profile visits went down by 21.1%. That makes sense to me! We’re a job board, not a personality brand. The goal was never to goose engagement numbers — it was to get fresh listings in front of people faster. Reach and freshness improved. That’s what we

were optimizing for.

Posting volume went from about 7.8 posts per day to about nine, a modest but intentional increase.

The operational win is harder to put in a chart, but just as important. Before the switch, I was spending close to ten hours a week babysitting the pipeline. After that, it dropped to somewhere between three and five hours saved weekly, and that's time I can put back into the community.

What we're building next

The Buffer API has been pretty useful for the core job posting pipeline that we've started extending the same architecture to a content marketing pipeline. We can run blog posts and newsletter content through the same detect-deduplicate-distribute pattern, with Buffer handling the social layer. We plan to send this automation live sometime in early May.

I've been in the software industry for a decade. I know what it feels like to be between work — it's the opposite of what most people would call fun. The market is competitive, and the job market is rough right now. I know that firsthand.

This automation matters to my business because it matters to my community. I want to see smart, conscientious people put to work. If QualityAssuranceJobs.com can be a better part of making that happen by getting fresh jobs to the right people faster, then I'm happy.

Ready to build?

If you're taking Buffer's API for a spin, we've got resources to get you moving. Our developer docs cover the GraphQL schema, auth flow, and quick-start examples. The Buffer MCP server docs walk through plugging it into Claude or any MCP-compatible AI agent.

If you need hands-on help, our support team is around, or you can join our Discord server and chat to other people building with the API.

We’d love to hear about what you make. Find us in Discord, or @buffer on all major social channels.

https://buffer.com/resources/we-built-a-custom-workflow-with-the-buffer-api-and-tripled-our-x-impressions/