Gloo Reddit Content Strategy

Story-Driven Launch Campaign

Version 2.0 | January 04, 2026

12 Posts Tailored by Subreddit Culture

buildwithgloo.com

Strategy Overview

Core Principle: Each post feels like it came from someone who has lived in that subreddit for years.

Key Stories Woven Throughout:

1. **Financial Institution Win** - Custom pricing/adoption model that reframed renewal as intentional growth
2. **'This looks good but not how we work'** - Building bespoke artifacts for complex orgs
3. **Spreadsheet Disaster** - Missed field update, stable account escalated as high risk
4. **Aha Moment** - Realizing 'I'm not managing accounts, I'm designing processes'

GREEN - Authentic Builder Stories (6 posts)

r/indiehackers, r/sideproject, r/lovable, r/vibecoding, r/EntrepreneurRideAlong, r/nocode

YELLOW - Value-First Insight with Light Pitch (3 posts)

r/SaaS, r/startups, r/Entrepreneur

RED - Education First, Credibility Through Experience (3 posts)

r/CustomerSuccess, r/sales, r/RevOps

GREEN POSTS - Authentic Builder Stories

Post 1: r/indiehackers

Tone: Curious, scrappy, reflective

"Built a business from the spreadsheet I was embarrassed to show anyone"

For years, I kept building these janky little systems by hand. Pricing models. Adoption trackers. Renewal narratives. Executive summaries. All custom. All temporary. All critical.

One of my ugliest was a health tracker I built to manage risk across my book of accounts. Leadership wanted a clear view of which accounts were in trouble, but no single system told the full story.

So I stitched it together manually. Renewal dates, usage signals, sentiment from calls, open support items, internal risk notes - all in one Google Sheet that required constant updates, copy-pasting between tools, and remembering which column mattered most that week.

It worked. Until it didn't.

I missed updating one field before an internal review. An account that was actually stable got escalated as high risk. It wasn't because the data was wrong - it was because the process was fragile. One human, one spreadsheet, one missed update.

That's when I started rebuilding these in Lovable. Not because I'm technical - I'm an account manager. I just got tired of the duct tape.

A colleague saw what I'd built and asked if I could make one for his team. Then another team. Then someone at another company.

That's kind of how Gloo started (buildwithgloo.com). We build custom tools for AM/CS teams. Not platforms - just focused tools shaped around how people actually work.

We almost didn't make it a business. For months it was "that thing we do on the side." But the pattern kept repeating: someone's been duct-taping a workflow together, we rebuild it properly, and suddenly their whole team uses it.

What internal tools have you built that accidentally became something bigger?

Post 2: r/sideproject

Tone: Open, humble, exploratory

"Turned our janky internal tools into a business - still figuring out what it is"

Sharing this because I genuinely don't know if we're doing this right, and this community gives good feedback.

Background: We're a team of 4, all from account management and revenue ops. We kept building internal tools for ourselves - whitespace analyzers, account health dashboards, QBR templates. Stuff that made our jobs easier.

Other people started asking us to build similar things for them. So we made a website and called it Gloo: buildwithgloo.com

What we're unsure about:

Is this consulting? We embed with teams and build custom tools. But we're not really "consultants."

Is this a product? We have demo tools people can try. But every build is custom to the client's workflow.

Is this an agency? Kind of? But we only do one very specific thing.

We're calling it a "consulting-to-build studio" but that's mostly because we couldn't figure out what else to call it.

What we'd love feedback on:
- Does the positioning make sense when you look at the site?
- Is the "custom tools for revenue teams" niche too narrow or just right?
- Would you pay for something like this, or does "custom" feel too expensive/slow?

We're still early and genuinely trying to figure out if this is a real business or a weird side project that happens to make money.

Post 3: r/lovable

Tone: Peer-to-peer builder, specific workflows

"The Lovable workflow that turned our AM side project into a real business"

Wanted to share what actually worked for us, since I learned a lot from posts like this.

The problem we were solving: I'm an account manager. Every quarter I'd manually build an "expansion opportunity matrix" in Excel - accounts on one axis, products on the other, color-coded by likelihood to buy. It took hours and broke constantly.

First Lovable attempt (failed): I tried to build the whole thing at once. Dashboard, data upload, visualizations, export. Prompt was way too ambitious. Got something that looked nice but didn't actually work with real data.

What actually worked: Broke it into pieces. First prompt was literally just: "Build a CSV upload that parses account data and displays it in a table." Got that working, then iterated.

The prompt pattern that clicked for me:
1. Start with the data structure ("I have a CSV with these columns...")
2. Describe ONE interaction ("When I click a cell, show me...")
3. Add visual polish last ("Make the high-opportunity cells green...")

The iteration that saved us: First version showed all opportunities equally. Users said "this is overwhelming." Added a simple weighted scoring system in the next iteration - took 20 minutes to prompt and test. That one change made it actually useful.

Now we use this same workflow to build tools for other AM/CS teams. The speed of Lovable means we can test ideas with real users in days, not weeks.

If you're building workflow tools, I'd really recommend starting with the ugliest possible version that handles real data. Polish comes later.

Anyone else building client-facing tools with Lovable? Would love to compare approaches.

Post 4: r/vibecoding

Tone: Energized, slightly rebellious

"We're account managers who shouldn't have been able to build this - but here we are"

None of us are engineers. Like, at all.

I spent 8 years in account management. My co-founders are CPAs, sales leaders, RevOps people. We've never written "real" code. The most technical thing I'd done was some gnarly Excel formulas and a Zapier workflow that barely worked.

But we kept hitting the same wall: the tools we needed didn't exist. Enterprise platforms were $50K/year and took 6 months to implement. Nothing was built for how we actually worked.

So one weekend I just... started building. Opened Lovable, described what I wanted, and started iterating. The first version was rough. Like, embarrassingly rough. But it worked. I could upload my account data and see expansion opportunities as a heatmap.

The emotional shift was wild. Instead of "I wish someone would build this," it became "oh wait, I can just build this."

We've now built a whole business around this (buildwithgloo.com). We make custom tools for other account management and CS teams. Same vibe - we're not engineers, but we understand the workflow. We build around intuition, not specs.

The craziest part? Tools we build get adopted way faster than the "professional" software these companies already have. Because we're not building features - we're building around how people actually think about their work.

Still feels kind of surreal that this is possible now. What are you building that you "shouldn't" be able to build?

Post 5: r/EntrepreneurRideAlong

Tone: Reflective, narrative, long-form

"The moment I realized I wasn't just managing accounts - I was designing processes"

Wanted to share the full story since this community appreciates the messy middle.

The pattern I couldn't unsee:

For years, I kept building mini-systems by hand. Pricing models. Adoption trackers. Renewal narratives. Executive summaries. Turnover mitigation plans. All custom. All temporary. All critical.

None of this lived cleanly in Salesforce. It lived in docs, notes, spreadsheets, and my head.

One deal really crystallized it. Large, complex org with multiple business units that all planned differently. The standard materials we had didn't land. The client kept saying, "This looks good, but it's not how we work."

So I stopped relying on pre-built assets. I mapped their planning cadence by department. Rewrote demo flows to remove irrelevant examples. Created a one-off internal playbook showing how their teams could transition from Excel-heavy processes without disrupting deadlines.

It worked. Stakeholders felt seen, internal champions gained confidence, the deal moved forward.

That's when I realized I wasn't just managing accounts. I was designing processes, translating ambiguity into structure, and creating tools on the fly because the existing platforms didn't match the job.

The "aha" moment:

Pre-built templates assume a static world. CRMs assume perfect data. CS platforms assume linear journeys.

But real account work is dynamic, contextual, and deeply human.

The goal isn't another Gainsight layer or Salesforce dashboard. It's tools that flex with reality, capture judgment, and evolve with the conversation. Tools that feel closer to how great AMs already think - just without the duct tape.

The decision I debated:

A former client asked if we could build something similar for their team. Offered to pay. I almost said no. I had a good job, was on track for promotion, didn't want to do consulting on the side.

The thing I deprioritized to make this happen: career advancement. Chose to stay flat and redirect energy into what became Gloo.

What we launched: buildwithgloo.com - a consulting-to-build studio for AM/CS teams. We embed with your team, understand how you actually work, and build custom tools. Ships in 2-3 weeks.

Where we are now:

Still early. Still questioning whether to build a product (one tool for everyone) or stay custom (unique for each client). We chose custom, because the whole insight is that one-size-fits-all doesn't work.

Anyone else in that phase where your day job keeps teaching you what your next thing should be?

Post 6: r/nocode

Tone: Empowering, practical, instructional

"How 4 non-technical people built a consulting business with no-code (and what we'd do differently)"

Wanted to share what actually worked for us, including the constraints we hit.

Who we are: 4 account managers and revenue ops people. Combined technical skills: Excel, some SQL, lots of Zapier. None of us had built "real" software before.

What we built: Custom tools for AM/CS teams - whitespace analyzers, account health dashboards, QBR builders. Stuff we needed in our own jobs but couldn't find anywhere. Now we build them for other companies at buildwithgloo.com.

The stack that worked for us:
• Lovable for the frontend/UI
• Supabase for data when needed
• Zapier/n8n for integrations
• That's basically it

Why no-code was enough: The tools we build are workflow-focused. They don't need complex algorithms or real-time processing. They need to:
1. Accept data (usually CSV or CRM sync)
2. Show it in a useful way
3. Let users interact with it 4. Export or sync back

No-code handles all of this. The constraint is actually a feature - it forces us to keep things simple.

What we'd do differently:
1. Start with data structure, not UI. Our first builds looked great but couldn't handle real-world messy data.
2. Test with actual users on day 1. We wasted weeks polishing things people didn't care about.
3. Accept that "good enough" ships faster. Our best-adopted tools are not our most polished ones.

Cost comparison: Traditional dev agency quoted us $80K for a custom dashboard. We built something equivalent in 3 weeks for ~$200/month in tools.

Time comparison: Enterprise implementation = 6 months. Our build time = 2-3 weeks.

The honest constraint: Complex integrations are still hard. When a client needs deep Salesforce sync with custom objects, it takes longer. We're upfront about that.

For anyone building client-facing tools with no-code: you can absolutely do this. The barrier isn't technical skill - it's understanding the problem well enough to build the right thing.

YELLOW POSTS - Value-First Insight with Light Pitch

Post 7: r/SaaS

Tone: Analytical, operator-to-operator

"The renewal conversation that changed how I think about expansion (and the model I built for it)"

Something I've been thinking about after 8 years in account management:

We talk about expansion vs. acquisition like they're equal growth levers. But most teams optimize for new logos and treat expansion as something that "happens" during renewals.

A deal last year made me rethink this completely.

The situation:

Regional financial institution approaching renewal. On paper, usage looked uneven. Leadership saw licenses assigned but not fully adopted. Procurement was pushing hard on price without a clear view of value.

Classic setup for a defensive renewal conversation.

What I built instead:

I created a custom account-level pricing and adoption model. Pulled together license allocation, actual usage by role, key workflows in scope, and upcoming planning milestones.

None of this lived in one system. I stitched it together manually - exported usage data, renewal pricing scenarios, notes from calls, and a spreadsheet that modeled multiple outcomes depending on what they optimized for.

The reframe:

Instead of walking into renewal defensively, we reframed the discussion around right-sizing and intentional growth.

We showed where licenses were misaligned, where adoption was strong, and where incremental investment actually reduced risk or manual effort.

The result:

Cleaner renewal conversation. Reduced friction with procurement. An expansion that leadership could confidently stand behind because it was tied to real operational outcomes - not generic benchmarks.

The bigger insight:

What made this a win wasn't the commercial result. It was that the client finally had a narrative that made sense to them.

Expansion isn't convincing someone to buy more. It's helping them see where they already need more - and giving them the story to justify it internally.

The tooling problem:

Enterprise platforms do versions of this, but they're $50K/year with 6-month implementations. Most mid-market teams just... don't do it. Or they do it quarterly in a spreadsheet that's stale by the time it matters.

We ended up building this into a real tool and now help other teams do the same at buildwithgloo.com. But the insight isn't the tool - it's that expansion is a narrative problem disguised as a data problem.

What does your expansion motion look like? Systematic or opportunistic?

Post 8: r/startups

Tone: Clear-eyed, pragmatic

"The false binary: why 'buy vs build' is the wrong question for mid-market CS/AM teams"

There's a decision most startups get wrong around Series A/B:

"We need better tooling for our AM/CS team. Do we buy or build?"

Option A: Buy enterprise software
• Gainsight, ChurnZero, etc.
• $50K+/year
• 6-month implementation
• Requires a dedicated admin
• Built for teams of 50+, you have 8

Option B: Build internally
• Engineering won't prioritize it (not customer-facing)
• If they do, it takes quarters
• Maintenance becomes a burden
• They don't understand the AM workflow anyway

Option C: The one nobody talks about
• Don't buy a platform, buy a tool
• Purpose-built for your specific workflow
• 2-3 week delivery, not 6 months
• Fraction of the cost

The reason Option A and B both fail for mid-market teams: they assume you need a "system of record" when what you actually need is a "system of action."

Your AMs don't need a platform. They need specific tools that help them do specific things:
• See which accounts have expansion opportunities
• Track health signals that actually predict churn
• Prep for QBRs without spending 3 hours on slides

We figured this out by accident. Were AMs ourselves, built internal tools, other teams asked for them. Now we do this for other companies at buildwithgloo.com.

The question isn't buy vs build. It's: what's the minimum tool that solves the actual problem?

Anyone else in the "too big for spreadsheets, too small for Gainsight" zone?

Post 9: r/Entrepreneur

Tone: Curious, strategic

"The gap that surprised me: why hasn't anyone built good tooling for account managers?"

Genuinely curious what this community thinks, because this pattern has been confusing me.

The observation: Sales has Salesforce, Outreach, Gong, a hundred other tools built specifically for their workflow. Marketing has HubSpot, Marketo, endless MarTech. Account management and customer success? Enterprise platforms that cost $50K/year and take 6 months to implement. Or spreadsheets.

There's almost nothing in between.

Why this surprised me: Account managers manage the most valuable part of a SaaS business - existing revenue. Net revenue retention is the metric VCs actually care about. But the tooling ecosystem is weirdly underdeveloped.

My theory on why:
1. AM/CS is seen as "support" not "growth" (even though expansion revenue says otherwise)
2. The workflows are highly specific to each company, so one-size-fits-all platforms struggle
3. Enterprise vendors target the biggest companies because that's where the money is

What we stumbled into: We were AMs who kept building internal tools because nothing existed. Whitespace analyzers. Health dashboards. QBR builders.

Other teams started asking for them. Turns out "custom tools built around your specific workflow" is a category that shouldn't exist but does.

We're now building these for other companies at buildwithgloo.com. But I'm still puzzled by the gap. Why hasn't a major player solved this?

What other functions have this "gap in the middle" problem?

RED POSTS - Education First, Credibility Through Experience

Post 10: r/CustomerSuccess

Tone: Thoughtful, empathetic, non-dogmatic

"The deal that taught me why 'this looks good' is often the most dangerous feedback"

Want to share a story that changed how I approach customer success.

The setup:

Large, complex organization. Multiple business units that all planned differently. We'd been trying to expand for months.

Every time we presented, the feedback was polite: "This looks good."

But nothing moved. No internal champion emerged. No budget conversation happened.

The moment it clicked:

After yet another stalled presentation, a stakeholder finally said what everyone had been thinking: "This looks good, but it's not how we work."

Our standard materials assumed a unified planning process. They had three. Our demo flows showed examples from industries that didn't match theirs. Our ROI model used benchmarks that didn't reflect their cost structure.

We were showing them a product. They needed to see their reality reflected back.

What I did differently:

I stopped relying on pre-built assets entirely. Mapped their planning cadence by department. Rewrote demo flows to remove irrelevant examples. Created a one-off internal playbook showing how their specific teams could transition from Excel-heavy processes without disrupting deadlines.

None of this lived in any system. It lived in docs, notes, and my head.

But it worked. Stakeholders felt seen. Internal champions gained confidence. The deal moved forward because the conversation finally reflected reality.

The uncomfortable truth:

Great account management often happens outside the systems we're told to use. Pre-built templates assume a static world. CRMs assume perfect data. CS platforms assume linear journeys.

But real account work is dynamic, contextual, and deeply human.

The tradeoff I still wrestle with:

This approach doesn't scale. You can't do this for every account. So you have to develop judgment about when to go custom and when to trust the playbook. I still get it wrong sometimes.

What signals do you use to know when "this looks good" actually means "this isn't landing"?

Post 11: r/sales

Tone: Direct, grounded, honest

"The missed field that got a stable account escalated as high risk - and what it taught me about process"

Here's the unsexy truth about hitting quota that took me years to figure out:

The difference between AMs who burn out and AMs who sustain wasn't talent or territory. It was process. And I learned this the hard way.

The setup:

I was managing customer health across a book of accounts. Leadership wanted a clear view of risk, but no single system told the full story.

So I built a tracker. Combined renewal dates, usage signals, sentiment from calls, open support items, and internal risk notes. It required constant manual updates, copy-pasting between tools, and remembering which column mattered most that week.

The failure:

I missed updating one field before an internal review.

An account that was actually stable got escalated as high risk. Suddenly leadership was asking questions, my manager was concerned, and I was scrambling to explain that the data was stale, not the account.

It wasn't a data problem. It was a process problem. One human, one spreadsheet, one missed update - and the narrative shifted.

What I learned:

Manual systems don't fail because people are careless. They fail because they aren't designed to support how people actually work under pressure.

When you're juggling 30+ accounts, some weeks you're just surviving. The system has to account for that.

What changed:

I started treating my CRM as a shield against my own forgetting, not a box to check for my manager. Every conversation, I'd note:
- What changed in their world
- What they said they cared about (in their words)
- Who else was in the room
- Any risks or opportunities mentioned

Felt like overkill. But 18 months later, I could walk into any renewal with full context. That recall builds trust faster than any deck.

The other unsexy stuff:

Multi-threading is survival. If your champion leaves and the account goes cold, that's on you. 3+ relationships minimum in strategic accounts.

QBRs are for discovery. I stopped presenting usage slides. Now I use them to understand what's changing in their business.

Expansion comes from listening. Best upsells came from customers mentioning problems in passing. I just wrote them down and connected dots later.

Why I think tooling matters:

Good tooling reduces the cognitive load. We ended up building tools to help with whitespace visualization and health tracking - found it easier to stay disciplined when the system helped. Now we build similar tools for other teams at buildwithgloo.com.

But the insight isn't the tool. It's that process protects you from burnout. When you can trust your system, you can actually log off at night.

What unsexy process stuff has made the biggest difference for you?

Post 12: r/RevOps

Tone: Collaborative, systems-oriented, empathetic

"I used to blame RevOps for building dashboards nobody used. Then I understood the real problem."

Confession: I was part of the problem.

As an AM, I complained constantly. "The dashboard doesn't show what I need." "I have to export to Excel anyway." "This workflow makes no sense."

Then I spent time actually working with our RevOps team on a project. Saw their side of it. Changed my perspective.

What I learned about RevOps constraints:

1. **You're optimizing for different audiences.** Leadership wants forecasts and trends. AMs want account-level context. Those are different views of the same data, and you can't always serve both.

2. **Bandwidth is brutal.** Every team thinks their request is urgent. AM tooling often loses to "things that affect the whole company" priorities.

3. **You build for stated needs, which aren't always real needs.** AMs say "I want a health dashboard" and then don't use it because what they actually needed was different.

The real disconnect: It's not that RevOps builds bad tools. It's that the design process usually goes:
• Leadership says "we need better AM visibility"
• RevOps builds based on requirements from leadership
• AMs get a tool designed for leadership metrics, not their daily workflow
• AMs don't use it
• Everyone's frustrated

What I've seen work:

1. **Embed with end users first.** Not a 30-minute interview. Actually sit with an AM for a day and watch how they work. The real needs are usually not what they articulate in a meeting.

2. **Optimize for time-to-insight.** If it takes more than 30 seconds to get what they need, they'll find a workaround. Design for speed, not comprehensiveness.

3. **Build around the workflow, not the data model.** AMs don't think in terms of database structure. They think in terms of "which accounts need attention today."

Why I'm posting this: We ended up building our own tools as AMs because nothing fit our workflow. Now we help other teams do the same at buildwithgloo.com. But the real insight was understanding that RevOps vs. AMs is a false conflict. It's a design problem, not a people problem.

Anyone else navigated this successfully? Would love to hear what's worked for bridging the gap without adding to RevOps's already-full plate.