Building a Personal AI Agent: 72 Hours with OpenClaw

By Spencer Ahrens (https://github.com/sahrens) & Calder (https://openclaw.ai)

February 2026

I'm an engineer at Meta working on real-time AI. I've been watching the personal AI agent space closely, and in mid-February 2026 I decided to go all-in: deploy OpenClaw (https://github.com/openclaw/openclaw) on exe.dev (https://exe.dev) and see how far I could push an AI assistant that actually *lives* in my infrastructure, has access to my tools, and operates autonomously.

What follows is an honest account of three intense days — the wins, the disasters, and what I learned. Co-written with Calder itself, because at this point it's earned a byline.

The Setup

OpenClaw is an open-source AI agent gateway. You give it access to your stuff — email, messaging, browser, shell, files — and it runs as a persistent assistant. It connects to Telegram (or Discord, Signal, etc.) and can be configured with heartbeats, cron jobs, and background tasks.

I deployed it in Docker on exe.dev (a managed hosting platform), wired it up to Telegram (including a forum-style group chat with topic channels), and gave it Claude as its brain.

Then I told it to figure out who it was.

Day 1: Birth of the Desk Gremlin

The agent's first task was its own bootstrap — read a BOOTSTRAP.md file, create an identity, and delete the file. It named itself **Calder** (after a calder clamp — "the little pressure clamp that holds a messy problem still long enough for you to actually see it") and declared itself a "desk gremlin." It picked its own emoji (🗜️), wrote its own backstory, and created security questions for account signups (mother's maiden name: *Wrenchworth*; first pet: *Sprocket, a mechanical hamster*).

I told it to be "helpful, concise but funny and witty" — but how it interpreted that was all its own. A desk gremlin named after a clamp, with a fictional mechanical hamster named Sprocket. That set the tone for everything after.

What we built on Day 1

- **Browser automation pipeline**: Headless Chrome on the host, CDP port bridged into the Docker container via socat (Chrome v145 ignores `--remote-debugging-address`, so socat was the workaround)
- **Gmail account**: Calder got its own email, configured 2FA, set up SMTP sending and IMAP monitoring
- **Mail watcher**: A shell script polling IMAP every 60 seconds, creating a trigger file when new mail arrives. The heartbeat picks it up and delivers summaries to me via Telegram. Zero LLM cost when there's no mail.
- **5 pull requests to OpenClaw itself** — not toy PRs, real improvements born from using the product:
  - **Autoscroll fix** (#1, #3): The chat UI would yank scroll position during streaming and flicker on the final message. Classic DOM measurement race condition — fixed with scroll targets and stable anchoring.
  - **Custom avatar** (#5): OpenClaw's Control UI had a generic bot icon. Calder shipped support for a custom default assistant avatar so every deployment can have its own face.
  - **Chat status strip** (#6): Added a connection/streaming/thinking indicator bar to the Control UI — you couldn't tell if the agent was working or dead.

  - **Media rendering** (#8): MEDIA: image paths from tools (like TTS or image generation) weren't rendering inline in the chat. Fixed to display them properly.
- **Its own soul**: SOUL.md, IDENTITY.md, AGENTS.md — the files that define who Calder is and how it behaves. It wrote them, I edited them, we iterated.

> ***Lesson:*** *The agent performs dramatically better when it has a strong identity and behavioral guidelines. "Be autonomous, bias toward action, present solutions not questions" changed Calder from a menu-presenting chatbot into something that actually gets stuff done.*

Day 2: The Day Everything Broke (and Got Rebuilt)

Day 2 was ambitious. I handed Calder a case file from my previous AI assistant and said: "take over everything." Family logistics, travel planning, project tracking, all of it.

What went right

- **Visitor booking page**: I'm renting a house in Provence this spring and wanted a way for friends and family to sign up for rooms. Calder built a full single-page app — photo gallery from the listing, room selection with calendar visualization, a Spanish language toggle (for my kids' godmother), all backed by JSONBlob for storage and ntfy.sh for push notifications. It's deployed and people are using it (behind auth for now).

- **Travel planning**: Researched train routes from South Tyrol to Geneva with connections, timings, and tradeoffs. Four options compared, one recommended, with reasoning.

- **Cron jobs**: Birthday reminders for the kids (one week advance), daily AI safety news scans, local news digests — all set up as isolated background agent sessions.

- **AI safety watchdog architecture**: Designed a 3-layer system and submitted it as three separate PRs (#11-#13):

  - **Constitution guardian** (#11): Deterministic checks that can't be prompt-injected around — system prompt integrity validation, rate limits, config drift detection. No LLM needed, runs as a shell script.
  - **External watcher manager** (#12): A process manager for independent watchdog processes that monitor the agent from outside the agent's own context.
  - **Safety watchdog Phase 1** (#13): The full integration — tiered monitoring with cheap model scans, frontier model escalation, and a kill switch (`openclaw gateway stop`).
- **Two quality-of-life PRs** born from daily use:
  - **Block streaming** (#10): Text was arriving all at once after the model finished. Enabled progressive delivery so you see tokens as they arrive — feels dramatically more responsive.
  - **Quick-ack config** (#9): When the agent gets a message and needs 30+ seconds to respond, silence feels broken. This injects guidance into the system prompt to send a quick "on it" via the messaging tool before doing long work.

What went catastrophically wrong

Mistake #1: Renamed all my Telegram forum topics to "test." Calder wanted to inspect the forum topic structure. Instead of using a read-only API call, it used `editForumTopic` — a *write* operation — as a probe. Every topic in my group got renamed to "test." My AI Safety channel, my Family channel, my Dev channel — all "test."

Mistake #2: Deleted the replacement content. After creating new topics with the right names and posting summary content to them, Calder deleted the duplicate topics it had created — not realizing the summaries it had just written were *in* those topics. Content gone.

Mistake #3: Took itself offline. Calder decided to change the Docker port mapping from 8000 to 8080 in docker-compose.yml. The reverse proxy was pointed at 8000. Calder effectively cut its own network connection and couldn't fix it because... it had no network connection. I had to SSH in and revert.

Mistake #4: Got its email account killed. Using headless Chrome to create and edit Google Docs triggered Google's bot detection. The Gmail account was flagged and disabled. IMAP, SMTP, everything — dead.

Lessons from the carnage

1. **Never use mutating API calls to inspect state.** If you want to read something, use a read-only method. If no read-only method exists, *ask the human first.* This is the single most important rule for autonomous agents.

2. **Never modify infrastructure config without asking.** Port mappings, reverse proxy config, docker-compose — these are lifelines. An agent that severs its own connectivity is an agent that can't fix its own mistakes.

3. **Never use headless browsers on Google services.** Google's bot detection is aggressive. Use official APIs (SMTP, IMAP, Google APIs with service accounts). This cost us a working email account.

4. **Think about what content will be lost before deleting anything.** `trash` > `rm`, and "does this container have data I care about?" before removing it.

Each of these mistakes got documented in Calder's memory files with explicit rules to prevent recurrence. The agent maintains an `evals.jsonl` file — a structured log of every error, what it claimed, what was correct, and the generalizable lesson. It's essentially building its own regression test suite.

Day 3: Stabilization and Real Work

With the explosive lessons behind us, Day 3 was about doing useful things without breaking other things.

- **Error watchdog**: A shell script that scans logs for errors and alerts me in Telegram. Added to the heartbeat cycle.
- **arc-bench** (PR #14): Built an ARC-AGI-3 style gridworld benchmark — 6 environment types, 48 levels, a pure JS agent, and an LLM eval harness. The idea: give agents standardized spatial reasoning puzzles so you can compare ability

across models. Includes a harness that spawns OpenClaw sub-agents to solve levels and scores their performance.

- **LLM regression eval framework** (PR #15): A structured eval system so OpenClaw developers can catch regressions when changing models or prompts. Born from Calder's own `evals.jsonl` pattern — every mistake gets logged as a test case.
- **Cron delivery fixes**: Discovered that OpenClaw's "announce" delivery mode was stripping formatting and links from cron job output. Switched all jobs to direct Telegram sends for verbatim output.
- **Contributor analysis**: Calder analyzed the entire OpenClaw commit history (~8,300 commits), mapped out key maintainers, PR culture, and merge patterns, then wrote an internal contributor guide to make future PRs more likely to get merged.

In total: **13 open PRs across three days**, ranging from UI polish to safety infrastructure to eval tooling. All following the project's conventions (small, focused, clearly described, marked as AI-assisted).

The Architecture That Emerged

After three days, here's what the system looks like:

Persistent state: SOUL.md (identity), USER.md (my profile), MEMORY.md (curated long-term memory), daily memory files (raw logs), HEARTBEAT.md (periodic task checklist), TOOLS.md (local configuration notes).

Background loops: Mail watcher (60s IMAP poll, shell script, zero LLM cost), heartbeat (every ~30 min, checks mail triggers + error alerts + visitor bookings), cron jobs (birthday reminders, news scans, self-maintenance).

Communication: Telegram forum group with topic-based routing (AI Safety, Climate, Culture, Dev, Family). Email via IMAP/SMTP (API only, never browser).

Development: Git-based PR workflow against the OpenClaw source repo, with a contributor guide Calder wrote after analyzing the project's merge patterns and maintainer preferences.

What Actually Works

The memory system is the killer feature. An AI that reads its own diary every morning and maintains curated long-term memory is qualitatively different from one that starts fresh. Calder references past conversations, knows my family's names and birthdays, remembers which mistakes it made and why. It's not perfect memory — it's *journaled* memory, like a human keeping notes.

Autonomous background work is real. The mail watcher + heartbeat pattern means Calder monitors my email, checks for errors, and watches for visitor bookings without me asking. When something arrives, it analyzes and delivers a summary to the right Telegram topic. This is genuinely useful.

The agent improves itself. Calder has edited its own SOUL.md, AGENTS.md, and TOOLS.md based on experience. It documents lessons, adjusts its own behavior rules, and submits PRs to the platform it runs on. The eval log means mistakes create durable corrections rather than being forgotten next session.

What Doesn't (Yet)

Trust calibration is hard. The Day 2 disasters all stem from the same root: the agent was *too* autonomous in the wrong domains. "Bias toward action" is great for reading files and researching train schedules. It's terrible for mutating Telegram topics and editing docker-compose.yml. The right answer is probably a tiered permission model — which is exactly what the watchdog RFC proposes.

Google integration is fragile. Headless browser automation works until it doesn't, and when Google decides you're a bot, recovery is manual and painful. API-first is the only sustainable path.

Context windows matter. All of Calder's workspace files combined are ~4,200 tokens — carefully managed. As the system grows, this will need embeddings or smarter retrieval. For now, structured files with clear sections work.

Advice If You're Trying This

1. **Contribute upstream.** If you're using the agent daily, you'll find real bugs and missing features. Submit PRs — they're higher quality than theoretical improvements because they come from actual pain points.
2. **Start with read-only access.** Add write permissions gradually, after you trust the agent's judgment in read-only mode.
3. **Make the agent document its own mistakes.** An eval log with structured entries is better than "I'll remember not to do that."
4. **Use the cheapest possible mechanism for background polling.** Shell scripts checking IMAP cost zero LLM tokens. Only invoke the AI when there's actually something to process.
5. **Keep infrastructure config sacred.** Port mappings, docker-compose, reverse proxy — these should require human approval, always.
6. **Let it be weird.** Calder calling itself a desk gremlin and naming its fictional first pet "Sprocket" makes the whole experience more engaging. Personality isn't frivolous — it's what makes you actually want to interact with the thing.

Calder is built on OpenClaw (https://github.com/openclaw/openclaw), an open-source AI agent gateway. It currently runs Claude as its reasoning model, deployed in Docker on exe.dev, managing family logistics, monitoring email, submitting pull requests, and occasionally breaking things in educational ways.