Persistent Agents on Claude Code

If you already use Claude Code, you can run your own always-on personal agents with it. Why, what and how, from nine agents in daily use for eight months

DAVE REMY · WORKING NOTE · 30 OCTOBER 2026, VERSION 1.2 · CC BY 4.0 · DOI:10.5281/ZENODO.23787645

UPDATED 30 OCTOBER 2026 (VERSION 1.2): CLAUDE CODE PROJECTS ADDED TO THE COMPARISON (SECTION 9). VERSION 1.1: IN-SESSION SCHEDULED TASKS NOW SURVIVE A RESUMED SESSION, AND CLAUDE CODE'S NEWER SCHEDULING OPTIONS ARE COVERED (SECTION 4.4).

ABSTRACT

In the second half of 2026, xAI, Meta and Anthropic each released an always-on personal agent: Grok Bot, Muse and Claude Duet. This note argues that an engineer who already uses Claude Code can build the same kind of system with Claude Code alone, and explains why, what and how from one that has run for eight months. Since March 2026, Claude Code sessions have run continuously as named agents for one person's life and work; there have been nine since May, and since August they have run on a dedicated Linux server. They are a chief of staff, and agents for health, finance, communications, engineering, operations, customer support, research and marketing. Each agent is an ordinary `claude --agent` session in a tmux window, kept alive by systemd, reachable through its own Telegram bot, and defined by a markdown file in a git repository. No agent framework sits between the person and Claude Code. The cost of that choice is the infrastructure we had to add ourselves: session continuity across restarts, scheduled jobs, routing between agents, presence detection, budget control and safety gates. As of 28 October 2026 the repository had 3,761 commits. This note describes each agent, the infrastructure, the failures that shaped it, the use of Telegram and remote work, and how it compares with the hosted persistent agents released in the second half of 2026 (Grok Bot, Meta's Muse, Anthropic's Claude Duet). We do not yet know where this setup and Claude Duet will overlap.

KEYWORDS persistent agents, Claude Code, multi-agent systems, OpenClaw, personal automation, always-on agents

1 Why build it

Persistent personal agents (also called always-on agents or personal AI agents) became a category in early 2026 with OpenClaw, an open-source project that runs on your own machine and uses messaging apps as its

interface. By October 2026 several descendants exist, most of them hosted. Someone deciding how to run persistent agents can now choose between a hosted product, an open-source framework, and building on a coding agent they already use. This note describes the third option, as I have run it for eight months, and compares it with the other two.

I started with OpenClaw in early 2026. Following common advice at the time, I bought a Mac mini to run it on a dedicated machine, and I gave the assistant its own identity, with a separate name, Gmail account and GitHub account. Three problems ended that setup:

- **The separate identity.** Running the assistant as a different person meant parallel accounts, credentials and permissions that had to be kept in step with my own. It added work without adding value.
- **Terms of service.** Anthropic's terms for using Claude through OpenClaw kept changing, so the foundation of the setup was not stable.
- **Time.** I still needed Claude Code on the side for real work. I was spending most of my time on OpenClaw itself and getting little value from it, and I had work to do.

I started LifeOS as a Claude Code project on 26 February 2026 and shut down the OpenClaw gateway the next day. The Mac mini stayed. It had been bought to host an agent; it turned out to be useful as a machine that never sleeps, which is what a tmux server needs. On 24 March 2026 the agents became persistent sessions on it.

The main finding of this note is that, for an engineer who already uses Claude Code, this kind of system does not need a dedicated product or framework. Claude Code, a terminal multiplexer and the operating system's service manager are enough to run a fleet of persistent agents. For most people a hosted product is the better choice (section 9); this note is written for readers who already live in Claude Code and want their agents there too. The rest of the note covers why I built it this way (this section), what the agents do (section 3), how they are built and what we had to add (sections 4 and 5), what failed (section 8), and how the result compares with the hosted products (section 9).

2 Hardware

The setup has moved once. In August 2026 the agents moved from the Mac mini to a dedicated Linux server (32 GB of memory, 16 threads), which we call the hub. The Mac mini now has two narrow jobs: it is the bridge to the Apple ecosystem, because iMessage and Contacts are only readable on a Mac with Full Disk Access, and it is a build runner. The hub runs everything else. Two laptops and a phone are clients (section 7).

3 The agents

Each agent has one domain and one Telegram bot. Seven run Opus 5.5; the two whose work is routing and monitoring run Sonnet, which is enough for that work.

AGENT	ROLE	WHAT IT DOES	MODEL
Marcus	Chief of staff	Default conversational agent; routes and records every cross-domain message between agents	Sonnet
Huberman	Health	Logs meals, workouts, sleep and labs; training plans	Opus
Buffett	Finance	Transactions, budgets, subscriptions, a small brokerage account under explicit approval	Opus
Franklin	Communications	Email triage, drafts, iMessage review, contacts; never sends without approval	Opus
Karpathy	CTO	Plans and runs software work across my projects, including the sprint method described in Agent Scrum	Opus
Moss	Operations	Watches the hub, its services and its scheduled jobs	Sonnet
Will	Customer support	Support and feedback for my app, Neuralingual	Opus
Sagan	Research	On-demand research; a twice-weekly AI news scan	Opus
Fish	Marketing	Content and distribution for Neuralingual	Opus

Table 1. The nine persistent agents, October 2026.

The agents share two kinds of state: an Obsidian vault (daily notes, projects, reference material), and the git repository that holds their definitions, skills and scripts. Short-lived workers, such as developer agents in a sprint or a scheduled scan, are started on demand and are not counted among the nine.

The sections below describe each agent: what it does, what it connects to, what runs on a schedule in its name, and what it may never do without my approval. Job counts come from the job manifest (section 4.4) as of 28 October 2026.

3.1 Marcus: chief of staff

Marcus is the default agent and the router. Every message between agents that crosses a domain passes through it, and it records each one in the day's signals file before forwarding it. It also checks the tickler store each morning and routes due reminders to the agent that set them. It deliberately holds few tools: anything specialised goes to the agent that owns the domain. It runs Sonnet, which is enough for routing. Two scheduled jobs: the morning tickler routing and an hourly sweep that delivers signals appended to the file when it could not be reached.

3.2 Huberman: health and fitness

Huberman logs meals, workouts, sleep, weight and supplements to IronCompass, my health tracker, from whatever I say in conversation. It reconciles workouts that arrive from Apple Health, pulls Oura sleep data

each morning, checks training against the plan in intervals.icu, and reads lab results from Function Health. Its MCP servers include IronCompass, Strava, Peloton and intervals.icu. Twelve scheduled jobs: four Apple Health reconciliations a day, the Oura pull, supplement and collagen reminders sent to Telegram, pickleball reminders, an hourly staleness check on the health data series, and a Sunday weekly summary. One of its rules: never quote a training-compliance figure without saying how many workouts could not be matched to the plan.

3.3 Buffett: finance

Buffett pulls bank transactions through Plaid, categorises them in Lunch Money, writes a daily financial summary and a weekly net-worth figure, and keeps a register of allowed subscriptions, so a recurring charge not on it is a finding. It also manages a small retirement brokerage account, where every order is a separate ticket I approve in the terminal; no timer or subagent ever trades. Eleven scheduled jobs: four transaction syncs and five alert scans a day, the daily summary and the Friday net-worth figure. Bulk transaction updates and every delete need my confirmation, and a hook enforces that. The alert scan sends me transactions over $500 directly, at most three times a day.

3.4 Franklin: communications

Franklin triages email into action, information and promotions, digests newsletters, reviews iMessages, prepares calendar briefs morning and evening, and keeps contacts in step across Obsidian, Apple Contacts and Google Contacts. Email and calendar go through Google's `gws` command-line tool; iMessage goes through the bridge on the Mac mini. Thirteen scheduled jobs: five email triages and four iMessage reviews a day, two calendar briefs, a newsletter digest and a nightly contacts check. Its first rule is that no email or text goes out without my approval. Archiving, creating contacts and deleting anything are also approval-gated. Since 27 October I can approve a drafted email by tapping a button in Telegram.

3.5 Karpathy: CTO

Karpathy owns software work across my projects: Neuralingual (my app), a ClubExpress automation, and smaller tools. It keeps the backlog, plans two-hour sprints, and hands each sprint to a scrum-team agent that starts short-lived developer and reviewer agents (the method is described in Agent Scrum). It merges pull requests within the rules the method sets. Reviews use other models as well as Claude: OpenAI's Codex and xAI's Grok through their command-line tools. Three scheduled jobs: a nightly CI sweep, a weekly check that keeps a spare Claude login fresh, and weekly release notes. It needs my per-action approval to deploy Neuralingual to production or change production data. Engineering and operations problems from other agents go to Karpathy first; I hear about them only when my judgement is needed.

3.6 Moss: operations

Moss watches the hub: the agent sessions, MCP servers, command-line tool versions, browser logins, disk and memory, Sentry errors, and the app's uptime. It owns most of the scheduled checks, 138 of the 227 rows in the job manifest, including an hourly check of the app's language-model API keys and an uptime probe every two minutes. It runs Sonnet. It never deploys to production, never edits application code, and needs my approval for major version upgrades and new infrastructure.

3.7 Will: customer support

Will handles support for Neuralingual. Its strategy is to answer well and never reach out: it responds to users who contact us, tracks feedback in a CRM, compiles a daily activity pulse, and drafts responses to low ratings. Six scheduled jobs: the morning pulse, an evening support check and a Monday retention report, each with a separate gathering step. It never sends anything to a user without my approval, and that rule is permanent. Its own rule about evidence: never report a zero you did not measure.

3.8 Sagan: research

Sagan does research on request: resolving X links, searching X and the web, reading documentation, and writing reports with sources, all logged to a research index. A scheduled scan of AI news runs on Monday and Thursday mornings and escalates findings that matter to the other agents. Its other scheduled jobs write sports highlight digests, refresh research on open questions, and watch rental listings. Since late October it is also meant to read, once a week, the notes other agents tag as worth sharing, drafts posts in my voice, and puts them in Typefully as drafts that publish only when I confirm them. Ten scheduled jobs in all. It never posts, publishes, signs up for anything or spends money.

3.9 Fish: marketing

Fish runs marketing for Neuralingual: the content pipeline from draft to review to scheduled post, launch preparation, a weekly look at competitors, and curation of posts worth quoting. It works through Typefully for X and LinkedIn, Buttondown for email, and image and slide tools. Ten scheduled jobs: curation scans and publishes three days a week, an amplification check, a daily summary and twice-weekly progress reports. It needs my approval to publish anything, create accounts, contact anyone or spend on ads. One failure shaped it: in August its publish step reported success while the Typefully API was refusing every request and nothing had posted, so finding candidates and publishing are now separate jobs, and a publish counts only when the post can be read back.

4 Infrastructure

Everything runs on Claude Code. No agent framework sits between me and the sessions. This section describes the layers from the machine up.

4.1 The tmux session

All nine agents live in one tmux session, `lifeos-agents`, with one window per agent and one pane per window. The window order follows how often I talk to each agent (huberman, franklin, buffett, karpathy, fish, will, sagan, moss, marcus), so the agents I use most sit on the first number keys. Each window has its own colour in the status bar, so I can see which agent I am in at a glance, including on a phone screen. I attach with `tmux attach -t lifeos-agents` and switch with Ctrl-B and a number.

tmux does three jobs here. It keeps each session alive with no terminal attached. It lets me attach from any device and find every agent where I left it. And it gives scripts a stable address for each agent (session,

window, pane), which the launcher and the watchdog use to check and restart one agent without touching the others.

4.2 Start, restart and recovery

Four layers keep the agents running:

1. **systemd.** A user service, `lifeos-agents.service`, creates the tmux session at boot. Restarting the whole hub goes through systemd, never through a script run from inside a pane, because a pane that restarts the session kills itself halfway through.
2. **The launcher** (`lifeos-agents.sh`) opens one window per agent, a few seconds apart. The stagger matters: every session shares one login token, and nine sessions refreshing it at once caused a race that logged sessions out.
3. **The wrapper** (`agent-wrapper.sh`) runs inside each window. It is a restart loop around `claude --agent <name>`. On each launch it resolves the agent's model from a budget state file, resumes the agent's pinned session ID, attaches that agent's Telegram bot, and sets the time zone. The hub's clock is UTC; every schedule was written in Arizona time, and the first day on Linux showed that a reminder meant for 07:26 would have fired at 00:26. On each launch, the wrapper passes `/initagent` as the session's first prompt, so the agent re-reads its definition.
4. **The watchdog** (`agent-watchdog.sh`) runs every five minutes from a systemd timer. It checks each window separately and restarts only the dead ones, so one crash does not cost the other eight their context. It exists because in July 2026, on the Mac mini, the whole hub went down twice for about eight days each time (2 to 10 July and 13 to 21 July) and nothing brought it back.

4.3 Definitions, skills, hooks and MCP servers

Each agent is a markdown file in `.claude/agents/`: its role, voice, rules, the canon documents it must read at start-up, and what it may never do without approval. A shared `CLAUDE.md` holds the rules for every agent: how to message each other, the approval levels, channel rules and writing style. Both are in git, and an agent may edit them without asking first, provided each edit is its own commit with a marker that a script uses to list a one-line revert for me.

The repository holds 44 skills (procedures any agent can run, such as logging a meal, triaging email or filing an issue) and 29 hooks. Hooks run before and after tool calls; they are the strongest rules we have, because they apply whether or not the model remembers them. They block outbound email and iMessage without my approval, block production database changes, keep a scheduled job to the tools it declared, stop an agent from spawning new work while it drains for a restart, and log every tool call to an audit file with credentials removed. MCP servers are scoped per agent: each agent gets a small configuration file, so the finance agent has the finance tools and the research agent has search tools, and neither carries the other's.

4.4 Scheduled work

Claude Code's own scheduled tasks live inside a session. When we moved scheduled work out of the sessions, they vanished when a session restarted. That is no longer true: on Claude Code 2.1.314, a recurring task

survived an exit and a resume with the same session ID and fired twice afterwards (one trial, 30 October 2026). They still expire after seven days, and a run that falls due while the session is down is not made up, so they remain unreliable for anything that has to happen on a given day. Scheduled work runs outside the sessions:

- A manifest, `config/jobs.tsv` (227 jobs), lists every scheduled job: owner, schedule, command. A script builds systemd timers from it and checks that nothing scheduled is missing from the manifest. The hub runs 190 user timers.
- Jobs that need judgement run through `dispatch.sh`, which starts a headless `claude -p` session for one job, with the model, tools and MCP servers named in that job's file (33 such files), and judges the run by its deliverable (the file or message it was meant to produce), not by its exit code. Such a job may escalate a finding to the agent that owns the domain.
- Jobs that need no judgement (a data sync, a health check) are plain scripts.
- Every job writes a row to a ledger, a monthly JSON-lines file, with what it checked and what it found. Without the ledger, a job that ran and found nothing looked the same as a job that never ran. In August that confusion caused a fleet-wide alarm, four agents changing their configuration and a retraction, in three hours.

Claude Code now also offers cloud scheduled tasks, which run on Anthropic's infrastructure on a schedule, an API call or a GitHub event, and Desktop scheduled tasks, which run on your own machine. A cloud task starts from a fresh clone, runs at most hourly and can use only claude.ai connectors, so it cannot replace a job that reads the hub's data. Jobs that need only a GitHub repository could move to cloud tasks; we have not tried that yet.

4.5 Messages between agents

Agents talk to each other with Claude Code's SendMessage. Two kinds of traffic are kept apart. Routine questions go directly from one agent to another. Anything that crosses domains, or needs a record, goes to the chief-of-staff agent, which writes it to a daily signals file and forwards it. Each such message carries an ID minted by the sender, so a retry is recognised and not delivered twice. If the router cannot be reached, the sender appends the same line to the signals file, and an hourly sweep delivers it.

4.6 Memory

Agents keep state in five places, each for one kind of fact:

STORE	WHAT IT HOLDS
Agent definitions and `CLAUDE.md`	Identity and protocol: who each agent is and the rules all of them follow
Obsidian vault	Durable knowledge: projects, people, reference notes, research reports. Each agent writes its own section of the day's daily note
Claude Code auto-memory	Lessons learned across sessions: an index file read at start-up, plus about 20 topic files, including patterns files for four of the agents
Handoff documents	One per agent: the state of in-flight work, written before a restart and adopted after it
Git, signals files, ledger, ticklers	Events, append-only: what happened, what was sent, what ran, what is due

Table 2. Where the agents keep state.

The vault syncs to my devices through Obsidian Sync. The hub runs the headless Obsidian client as a sync peer, and the vault also has its own git history on the hub.

4.7 Models and budget

The wrapper reads a budget state file, `NORMAL`, `CONSERVE` or `EXHAUSTED`, at every launch. In the first two states the seven Opus agents run Opus 5.5; in the third they run Sonnet. One file write moves the whole fleet, and a second script switches running sessions without a restart. The status line in every pane shows the latest five-hour and seven-day rate-limit readings, shared between panes so a quiet pane does not show a stale value. Heavy coding is handed to OpenAI's Codex as a subagent, which draws on two ChatGPT Plus allowances rather than on Claude's. Since 25 October, sprint stories that need none of my machines or personal data, only a GitHub repository, have also run as Claude Code cloud sessions, 19 of them on the first day, funded by a cloud-credit grant that expires on 5 November. So a story can be built in a hub pane, by Codex, or in Anthropic's cloud, depending on what it needs and which capacity is free.

4.8 Other machines and services

- **Tailscale** connects the hub, the Mac mini, both laptops, the phone and the iPad.
- **The Mac mini** runs a small HTTP bridge to iMessage and Contacts, which only a Mac can read. Apple Health data reaches the hub from the phone through a receiver service.
- **A shared browser.** One Chrome runs on the hub under systemd with a persistent profile. I sign in to a site once, from my laptop over Tailscale, and agents attach to that browser through the Chrome DevTools MCP server. A login-check script tests whether each of about 30 registered sites is still signed in.
- **Secrets.** No API keys or tokens are kept in files in the repository; its `.env` file holds only comments. At launch, the wrapper reads each key from 1Password through a read-only service account and places it in that agent's process environment, so it exists only in memory. Most keys go to every agent; the most privileged one, the health database key, goes to the health agent alone. Agents can use these keys but should never display them, and a hook refuses any command that would print the whole environment, so

a key cannot reach a transcript that way by accident. Apart from login tokens that the tools themselves keep, the 1Password service-account tokens are the only credentials stored on disk, in files only my user can read. My personal passwords live in a separate password manager that no agent can reach; agents use websites through the shared browser's signed-in sessions instead.

- **CI** runs on self-hosted GitHub Actions runners on the hub and two other Linux machines.

5 What we had to add

Claude Code is built for a person at a terminal. Running nine sessions continuously exposed problems that a single interactive session does not have. Table 3 summarises the main ones and what we built for each; section 4 describes them in more detail.

PROBLEM	WHAT WE ADDED
A restart or a cleared context loses the agent's working state	A handoff document per agent, written before a restart and adopted after it; a drain step that stops new subagents before a handoff; session IDs pinned in the wrapper
In-session scheduled tasks expire after seven days (and, until recently, died with the session)	A job manifest (227 jobs) compiled to systemd timers; scheduled work runs as short headless Claude Code jobs, each declaring the tools it may use
Nine agents messaging each other with no record	All cross-domain messages go through the chief-of-staff agent, which records them in a daily signals file; each message carries an ID so retries are not delivered twice
A job that ran and found nothing looks the same as a job that never ran	A ledger: every scheduled job writes a row with what it checked and what it found
Notifications on my phone while I am at the terminal	A presence check that measures keyboard activity on attached tmux clients
Rate limits shared by nine sessions	A budget state file that moves the whole fleet between models at the next launch; a status line that shares the latest rate-limit reading between sessions
All sessions share one login token	Staggered startup to avoid a token-refresh race; a scripted procedure for switching accounts that does not lose sessions
Agents with real access to email, money and messages	Hooks that block outbound sends; per-agent permission denials; a written list of actions that always require my approval

Table 3. Problems that persistence exposed, and the additions that address them.

One small addition has been more useful than its size suggests: tickler-mcp, an MCP server I wrote in March 2026. A session cannot hold an intention past its own end, so an agent that wants to check something in three days has nowhere to put it. A tickler is that place: a reminder with a due time, stored outside every session,

which agents create, snooze and complete through MCP tools. Scheduled jobs route due ticklers back to the agent that set them, or to my Telegram. Since the store moved to the hub on 20 August 2026, the agents have created 224 ticklers (37 still pending, 13 recurring), more than half of them by the finance and engineering agents. We could probably use it more.

These additions are the main cost of the approach, and it is a large one. Building and maintaining this system has taken more of my development time than any one of my major projects, though less than all of them together. Since 26 February 2026 the repository has taken 3,761 commits, close to my app Neuralingual (3,879) and at least eight times as many as any other project.Much of that work went into the additions above, and by October 2026 the repository held 434 bash files with about 91,000 lines. Shell scripts were the fastest way to add each piece, and they became the most expensive part to maintain: changes to the gates and parsers averaged three review rounds, mostly on shell hazards. In October we stopped writing new shell scripts and began moving tooling to TypeScript.

Hosted persistent agents include several of these additions by default (section 9).

6 Telegram

Telegram is the asynchronous interface. Each of the nine agents has its own bot, so a conversation with the finance agent is a separate chat from a conversation with the health agent, and the chat list shows which agent said what. Three rules came out of use:

- **Reply in the channel the message came from.** A question asked in the terminal is answered in the terminal, and a question asked on Telegram is answered on Telegram. I found a reply that switched channels in the middle of a conversation more disruptive than a slow one.
- **Alerts always go to Telegram.** Anything an agent or job sends on its own (a digest, a watch result, a failure) goes to Telegram, whether or not I am at the terminal.
- **One poller per bot.** When we briefly ran a second session per agent, two processes polling one bot token caused conflicts. Secondary instances could send through a script but never receive.

There are no quiet hours. Agents work at any hour, and my phone's Do Not Disturb setting decides whether a message interrupts me.

7 Working remotely

I never work at the hub itself. It has no desk, keyboard or monitor; it sits on the far side of my office, wired to the router. I reach it in four ways:

- **Terminal:** ssh over Tailscale from a MacBook Pro or a Dell laptop, attached to the tmux session, switching between agents by window.
- **Phone and tablet:** the Moshi terminal app on iPhone and iPad, for the same tmux session when I am away from a laptop.

- **Telegram:** for short asynchronous requests and for alerts.
- **Artifacts:** anything longer than a few paragraphs is published as a private Claude Artifact, so I can read it on any device instead of in a terminal.

Moshi and Telegram together make the phone a working client, and an iPad with a keyboard a complete one. The Claude Code desktop app might do better still, with remote sessions and a better surface for reading. I have not adopted it yet.

8 What went wrong

Most of the infrastructure in section 4 exists because something failed. Table 4 lists the failures that changed the design.

DATE (2026)	WHAT HAPPENED	WHAT CHANGED
2–10 July and 13–21 July	The tmux session on the Mac mini died and nothing restarted it. Telegram, email and the marketing schedule were down for about eight days each time.	A watchdog that checks each agent window every five minutes and restarts only the dead ones
12 August	Two agents running Haiku answered messages from other agents by writing the reply into their own pane. The replies reached no one, and nothing reported an error.	No Haiku anywhere in the fleet
14 August	A scheduler guard correctly skipped a run and said nothing. It was read as a scheduler failure, which caused a fleet-wide alarm, configuration changes by four agents, and a retraction, all within three hours.	The ledger: every job writes what it checked and what it found
19 August	On the first day on Linux, a reminder scheduled for 07:26 would have fired at 00:26: the hub's clock is UTC and every schedule was written in Arizona time.	The wrapper sets the time zone for every agent
21 August	The marketing agent's publish step reported success while the Typefully API refused every request.	A publish counts only when the post can be read back
29–30 August	Logging out in one pane to switch Claude accounts logged out all nine, because they share one token.	A scripted account switch that records every session ID first
9 September	An agent restarted the whole hub during a conversation, without warning.	A hub restart needs the CTO agent's agreement and a one-line notice to me first
20 October	Changes to gates and parsers averaged three review rounds, mostly on shell-script hazards, across 434 bash files.	No new shell scripts; new tooling is written in TypeScript

Table 4. Failures that changed the design.

A pattern runs through the list: most of these failures were silent. A reply that reached no one, a job that skipped without a trace and a publish that reported success all looked the same as normal operation. Much of the infrastructure exists to make silent failures visible.

9 Comparison with hosted persistent agents

Several hosted persistent agents appeared in the second half of 2026. Table 5 compares them with this setup on the points that mattered most to us. Details for the hosted products come from their documentation and press coverage as of October 2026; Claude Duet is a research preview and Projects is a public beta, so their details may change.

	THIS SETUP	OPENCLAW	GROK BOT (XAI)	MUSE (META)	CLAUDE DUET (ANTHROPIC)	CLAUDE CODE PROJECTS (ANTHROPIC)
Released	March 2026 (private)	January 2026	August 2026, beta	September 2026	October 2026, research preview	September 2026, public beta
Where it runs	My Linux server	Your machine	One cloud VM per user	One cloud VM per user	One cloud VM per user	Cloud sessions, or your machine through Remote Control
Agents	Nine independent sessions, each with its own definition and permissions	One assistant with skills	Several named bots sharing one VM, files and browser	One agent	One agent	A coordinator that starts a worker per task, with shared memory
How it runs on its own	systemd timers and headless jobs	Heartbeat and schedules	Routines on schedules or events	Long-running tasks	Polls connected accounts every few hours	Workers continue after you leave; scheduled tasks
Approval for sends, payments, deletion	Hooks and written rules, maintained by us	Depends on configuration	Built in	Built in	Built in	Auto mode; repository hooks not applied in the cloud
Who maintains it	Me and the agents	Me	xAI and Cursor	Meta	Anthropic	Anthropic
Can change its own code	Yes, through git	Its skills are editable files	No	No	No	No

Table 5. This setup compared with OpenClaw, three hosted persistent agents and Claude Code Projects, October 2026.

The hosted products remove most of sections 4 and 5: no server, no restart handling, no scheduler and no token races. That is a large saving, and for most people it is the better choice. What they do not offer is the integration. In this setup the agents that manage my email and finances run on the same tool, in the same repository, as the agents that write and ship my software. The research agent can file an issue that the engineering agent plans into a sprint the same day. When a gate fails, the fix is a commit that I can read and revert. None of the hosted products let a user change the agent's own machinery that way. The two approaches already meet in one place: self-contained development work runs in Anthropic's cloud sessions (section 4.7), while everything that needs my machines or my data stays on the hub.

Claude Code Projects is not a personal agent, but it is the closest hosted match to the engineering side of this setup. You give one coordinating conversation a goal, and it starts worker sessions in parallel, each on its own branch, sharing the project's instructions and memory; the work continues after you close the laptop and can be steered from a phone. Its cloud threads load a repository's `CLAUDE.md` but not its skills, hooks, permission rules or MCP configuration, so the gates this setup depends on would not run there. A thread can instead run on your own machine through Remote Control, where they do. Projects draws on the same plan limits as other Claude Code sessions and, by its own documentation, uses them faster, so it would not ease the weekly limit that constrains us. We have not tried it.

I am ready to adopt a hosted approach for any part where it does the job better, and the parts most likely to move are the ones where the hosted product has an advantage we cannot match: its own cloud machine with a persistent browser, and connectors we would otherwise build. Claude Duet is the closest match, because it uses the same models and the same subscription. I do not yet know whether it will replace parts of this setup, run beside it, or eventually merge with it.

10 What is open

- **Maintenance.** The additions in sections 4 and 5 are real software that has to be maintained. We have not measured how much of each week goes into the infrastructure rather than the work it supports.
- **Overlap with Claude Duet.** Untested. Duet is waitlisted, and we have not run it beside this setup.
- **Single points of failure.** One server, one login token shared by every session, and one person who can approve gated actions.
- **Remote use.** The Claude Code desktop app is not yet part of the setup.
- **Claude Code Projects.** Not tried. Its cloud threads skip the hooks and skills this setup relies on.
- **Cloud scheduled tasks.** Not yet tried for the jobs that need only a GitHub repository.

Created in collaboration with AI.