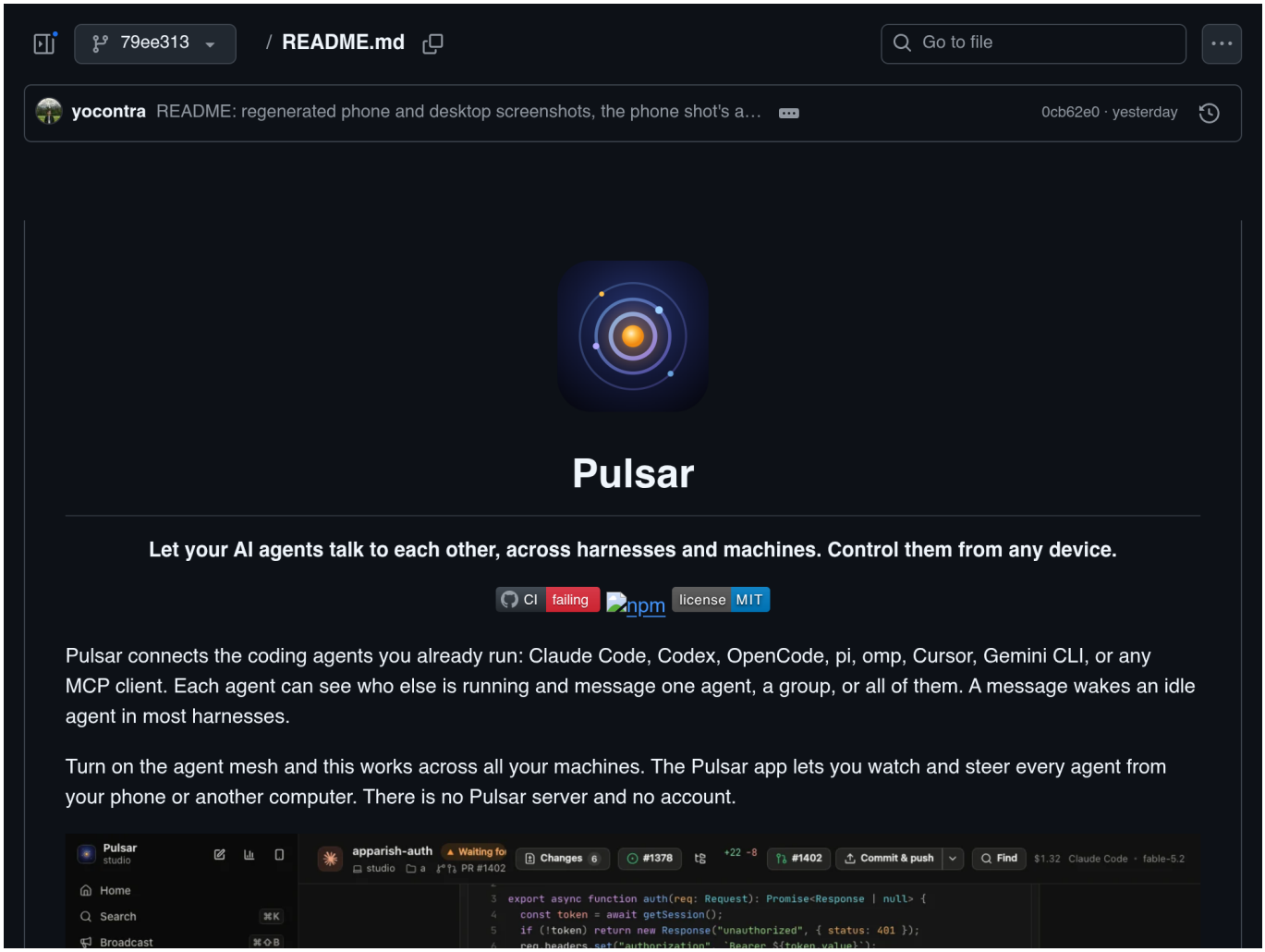


Pulsar in Practice: What a Cross-Harness Agent Message Bus Does, and What It Does Not

The brief I was handed says Pulsar's latest release is 1.4.0 from 17 November 2026. The repository I read on 24 November says 2.0.1, dated 23 November, with 61 stars and zero open issues.

Pulsar is an MCP server plus a small per-machine daemon that puts Claude Code, Codex, OpenCode, Cursor, Gemini CLI, pi and omp on one message bus, so an agent can see who else is running and send a message to one peer, a selector or everyone. It was created on 17 November 2026 and shipped a major version two days later. Anything you build against its tool names today is building against a moving target.



The yocontra/pulsar README on GitHub.

One daemon, no account, six tools

The architecture is deliberately small. There is no Pulsar server and no account: each machine runs a daemon, and each harness talks to it through an MCP server you install with `npx pulsar setup`. Setup finds your agents, shows which files it will change, asks once, and backs up every file it touches. Per-harness installs exist too: `/plugin install pulsar --marketplace yocontra/pulsar` for Claude Code, a two-command sequence for Codex (`codex plugin marketplace add yocontra/pulsar && codex plugin add pulsar@pulsar`), `pi install npm:pulsar`, `omp install pulsar`. The pi package page notes that `pulsar setup` also sets Claude's `crossSessionInbound` and Codex's tool approval, which plugin installs cannot reach. That detail is worth reading twice:

the plugin path leaves two harness settings unconfigured, which is exactly the kind of thing that makes a message bus look broken when it is merely half-installed.

The agent-facing surface is six MCP tools: `list_agents`, `send`, `respond`, `wait`, `inbox`, `set_identity`. The research brief also records an earlier README listing `reply`, `history` and `whoami`. Those names are gone. In roughly a week of development the public tool surface was renamed at least once, which tells you how to treat it.

Delivery has three priorities on `send`: `urgent` arrives mid-turn, `normal` wakes an idle agent for its next turn, `fyi` does not wake the recipient at all. Recipients push back with an inbound policy on `set_identity`: `accept`, `hold` or `refuse`. `respond` carries four statuses (`working`, `reply`, `done`, `declined`). And `wait` blocks on `message`, `reply:` or `idle:` rather than having an agent burn turns in a sleep loop. That last one is the piece I would care about most, because the alternative pattern (agent polls, agent sleeps, agent polls) is where token budgets go to die.

Three harnesses on one machine share nothing but the filesystem

Several coding harnesses on one machine is now an ordinary setup, and they share nothing except the filesystem and git. Each one is a separate process with its own context and its own idea of what is happening in the repo. If the Claude Code session holds `main` frozen for a release tag, the Codex session on `feat/billing` has no way to learn that except through a human relaying it.

Pulsar's answer is a registry plus addressing. Agents tag themselves automatically with harness, model, project, branch, worktree and PR, and can add their own tags (`role:reviewer`, `issue:1378`). You then address a name, a selector (`project=apparish`, `pr=1402 role=reviewer`, `model=fable`) or *. The README's own example is the honest version of the use case: "tell everyone on apparish that main is frozen until the release is tagged", which fans out to three agents and reports the delivery path for each (`sent via claude-socket`, `sent via codex-queue`, `delivered via extension`).

There is a safety property baked into the addressing that I think matters more than the

messaging itself. Messages say they come from a peer, not from your operator, and cannot approve anything. Only after a passkey check ("Prove Human": Face ID, Touch ID, Windows Hello or device PIN) do your messages arrive tagged as the human operator and get followed as your own prompt. No agent or peer can produce that tag. Given that most MCP servers ship without authentication at all, **a local bus that starts from "agents must never be able to approve their own actions" is making the right first decision**.

Delivery is at least once, and the ack lifecycle is not shipped

A message that waits for an answer is delivered at least once: if the recipient's session restarts before answering, Pulsar delivers it again. After 5 attempts or 24 hours it gives up and tells the sender. Each harness receives through its own channel: Claude's inbox socket, `codex queue`, OpenCode's server, the pi/omp extension. For an MCP client with no push path, messages wait in the inbox and every Pulsar tool result reports the unread count.

On Hacker News, a commenter named lajjtis read the design as a small message broker and asked the right question: is "delivered" the end of it, or will there be something like "done", so a consumer acks only after the work completes and a crash halfway triggers redelivery? The author's reply promised a fuller lifecycle: queued, delivered, seen, acked, done, with durable redelivery at any drop point. As of the docs I can read, that lifecycle is not shipped. What exists is at-least-once delivery with a retry cap, which is weaker than acknowledging completion.

WHAT WORKS

- No hosted service, no account; the trust boundary on one machine is your user account, and there is no telemetry.
- Priorities that map to real intent: `urgent` mid-turn, `normal` next turn, `fyi` no wake at all.
- `wait` on `message`, `reply:<id>` or `idle:<agent>` replaces polling loops.
- Peer messages are marked as peer messages and cannot approve anything; operator identity requires a passkey check.
- At-least-once redelivery for messages awaiting an answer, capped at 5 attempts or 24 hours.

ROUGH EDGES

- Codex reports no busy or idle state, so `wait idle:` never resolves for a Codex agent.
- Codex needs explicit tool approval and a longer timeout for blocking calls; plugin installs cannot set it, only `pulsar setup` can.
- Cursor and Gemini CLI cannot have prompts answered from the app and cannot be stopped.
- Claude Code can only be stopped inside a tmux pane; OpenCode not in `--standalone` TUI mode.
- No acked or done lifecycle yet, so a crash after delivery but before completion is silent.

The harness support table is uneven in a way the line “any MCP client” does not convey. Seven named harnesses get messaging. Control from the app is narrower: prompts relay from Claude Code, OpenCode, omp and pi, read-only from Codex, and not at all from Cursor or Gemini CLI. Stop works everywhere except Cursor and Gemini CLI, and for Claude Code only under tmux.

The mesh arrived faster than the Show HN promised

At launch on 17 November the author said cross-machine operation would come in a following release over Tailscale, with no cloud service. The 2.0.0 changelog dated 19 November delivers exactly that. The mesh is off by default, and `pulsar mesh on` turns it on after checking Tailscale is installed, running, signed in, with MagicDNS and HTTPS Certificates enabled. Pulsar serves HTTPS only on each machine’s Tailscale address, with a certificate from `tailscale cert`, and identifies every connection with `tailscale whois`. Machines signed in as the same Tailscale user link on their own, with no pairing codes. Self-hosting on Headscale is documented for networks where traffic must stay internal.

The per-machine identity model holds across the mesh, which I read as the most considered part of the design. Each machine registers its own passkeys. An action against an agent on another machine asks again, because each machine decides for itself. Messages to agents on another machine always arrive unverified there. Proof lasts 12 hours in a browser and 7 days on a phone. First answer wins when a prompt is visible on several nodes.

Two major versions in a week is a signal in both directions

My take: it means the author ships, and it means the surface you integrate against has no stability guarantee yet. The HN criticism that this is heavy setup compared to agents sharing a plain web mailbox (the commenter pointed at chat.bagpiper.dev) undersells the identity work, but it is fair about the install weight: per-harness plugins, a daemon, optionally Tailscale, HTTPS certificates and passkeys. I would not wire a team workflow to the six tool names before the lifecycle the author described on HN actually lands.

Who should install this week, and who should wait

Install it now if you personally run three or more harnesses on one machine and keep losing time to coordination you do the relaying for: branch freezes, who is editing which file, one agent waiting on another's PR. `list_agents` with a `question` ("who is editing src/auth/session.ts?") answers from the registry and session logs without waking anyone, which is cheap compared to asking each agent. It is MIT licensed, TypeScript, zero open issues, and uninstall is a documented single command (`pulsar setup --uninstall`) with file backups taken on the way in. The downside risk of trying it is low.

Wait if you are building tooling on top of it, or if you want durable task handoff rather than notification. The tool names already changed once, the message lifecycle is explicitly described by the author as a future release, and two harnesses in the supported list (Cursor, Gemini CLI) get messaging but no control. Wait also if your Codex sessions are the ones you would coordinate around, because `wait idle:` does not resolve for them at all, and that is the primitive the whole handoff pattern depends on.

One more structural point. This is an agent-to-agent bus, and the thing it does best is wake an idle agent at the right moment. If what you actually want is agents that keep working without being woken, that is a different problem with different tooling. **Pulsar makes agents aware of each other.** It does not make them autonomous, and the README does not claim it does.



WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →