OpenClaw Meetup Lisbon • March 31, 2026

MOLTIS

One binary. Sandboxed. Yours.

A secure persistent agent server for real-world AI workflows.

Fabien Penso · @fabienpenso

https://www.moltis.org

Why Moltis exists

Same direction

I loved what OpenClaw unlocked. Moltis exists because I wanted to run my own version of that future.

Different tradeoffs

I wanted Rust, stronger safety boundaries, and a more defense-in-depth architecture once agents get real permissions.

Less friction

I wanted installation and setup to take minutes, not a side quest, so more people could actually try it.

This project started from admiration, not rejection. Moltis is my opinionated Rust-native take on the same future.

Another strong opinion: installation should not be the boss fight

- At the time, people were struggling so much with installing OpenClaw that someone in San Francisco built a business around setting it up for other people on Mac minis
- That was a real signal to me that the appetite was huge, but the setup still had too much friction
- I wanted Moltis to be something you can get running quickly, without turning setup into a side quest
- That meant one-binary installs, Docker and Docker Compose paths, Homebrew on macOS, and binaries for Linux architectures people actually use
- Installing Moltis should take a few minutes at most, and on DigitalOcean it can take about a minute

For me, that was the cue to take installation seriously as part of the product, not just as an afterthought.

The moment agents get real permissions, the architecture changes

Code

If an agent can edit code and run shell commands, isolation stops being optional.

Secrets

If an agent can touch credentials, auth and secret handling become product features.

Continuity

If the agent is persistent, memory, sessions, and recovery matter as much as model quality.

That is the design center for Moltis: not just "can it do something cool?", but "can I safely let it keep running?"

What Moltis actually is

Local-first persistent agent server

- One Rust binary between you and multiple LLM providers
- One agent that can meet you across web UI, API, Telegram, Discord, WhatsApp, Teams, voice
- Durable sessions, long-term memory, tool use, MCP, scheduling, browser automation

1,112	~295K	~204K
files	lines of code	lines of Rust

```
channels + UI
      |
      v
  gateway server
      |
  agent loop + tools + providers
      |
  sessions + memory + sandbox
```

Important features

Core

- single binary
- local LLM support
- streaming-first responses

Security

- passwords, passkeys, tokens
- encrypted vault
- filesystem or sandbox isolation

Memory

- long-term memory
- hybrid vector + full-text search
- session recall and branching

Channels

- web UI
- Telegram, Discord, WhatsApp, Teams
- voice input and output

Also built in: skills, hooks, MCP, browser automation, OpenClaw import, GraphQL, scheduling, and observability.

Design principles

Security

- sandboxed execution
- password + passkeys
- encrypted vault for secrets

Persistence

- cross-session recall
- long-term memory
- automatic checkpoints

Ownership

- your hardware
- one binary
- local-first by default

The goal is not just to make agents more capable. It is to make them survivable.

Security is part of the product

Safer foundation

- Rust for memory safety and stronger boundaries
- one binary, smaller runtime surface
- different trust model from the start

Defense in depth

- passkeys and passwords in the web UI
- sandboxed tool execution
- encrypted vault and network protections

Safer access

- designed to keep strangers out
- Tailscale support for private access paths
- local-first deployment instead of exposing everything by default

One of my motivations for Moltis was simple: if agents are going to get real permissions, security cannot be a bolt-on.

What people actually use Moltis for

Engineering copilot

- code editing
- sandboxed shell
- session recall

Persistent personal agent

- long-term memory
- cross-session recall
- scheduled jobs

Multi-channel assistant

- web UI
- Telegram or Discord
- voice access

Platform for builders

- MCP servers
- custom providers
- GraphQL and API

The pattern is clear: people are wiring Moltis into real workflows, not treating it like a toy chatbot.

Momentum

2,411
⭐ GitHub stars

117
PRs merged in March

92
issues created in March

102
issues closed in March

This is moving fast because people are actually using it, reporting breakage, and pushing it forward.

For the Rust engineers

Architecture

- one Cargo workspace, split into many focused sub-crates
- shared dependencies live in `[workspace.dependencies]`
- clean service boundaries via traits and crate separation

Feature-gated builds

- `web-ui`, `voice`, `vault`
- `graphql`, `local-llm`, `tailscale`
- `lightweight` for smaller builds

Why Rust helps here

- types carry config, protocol, and service boundaries
- compile-time gating beats runtime roulette
- `secrecy::Secret` for credentials, not vibes

House rules

- `unsafe` denied workspace-wide in core
- `unwrap` and `expect` denied by lint policy
- tracing and metrics are first-class, crate by crate

The pitch for Rust people is simple: Moltis uses Rust as an architecture tool, not just as an implementation language.

Why try Moltis if you already know OpenClaw?

Keep your history

- read-only OpenClaw import
- sessions, skills, memory, providers, channels
- run side by side if you want

Change the tradeoffs

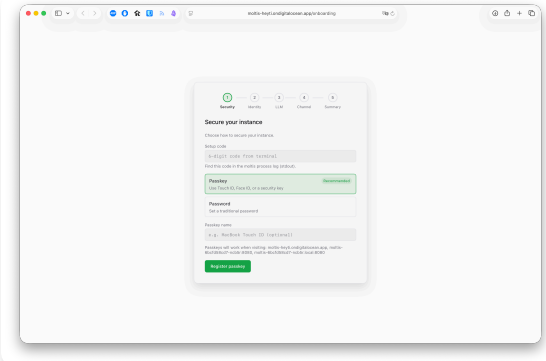
- Rust-native architecture
- stronger local-first and security posture
- different assumptions around sandboxing and persistence

Get started faster

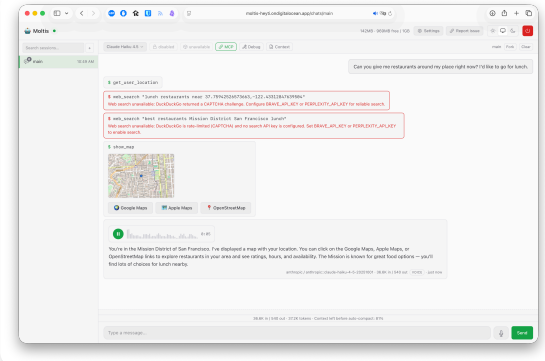
- simple install paths
- one-binary mindset
- low migration risk, low setup friction

The ask is small: keep what already works for you, and try a different trust model without starting over.

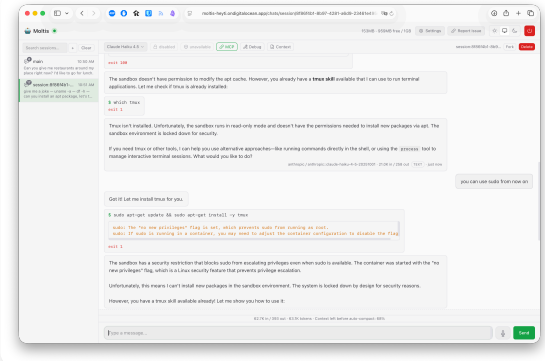
If I skip the live demo, this is the product arc



Secure onboarding, password or passkey



Tool-using chat, voice, maps, live interaction



Shell execution inside the sandbox, not on the host

That is the story: secure setup, persistent interaction, sandboxed action.

One core, more places to use it

Mobile

- native iPhone access to the same agent
- better mobile presence and notifications
- same sessions when you are away from your desk

Desktop

- native macOS app experience
- same Moltis core reused through the Rust Swift bridge
- native UI without splitting the product into separate engines

The point is not "more apps." The point is the same local-first agent becoming available everywhere you actually are.

Takeaways

1. Agents are becoming infrastructure, not toys
2. The trust model is part of the product
3. Moltis is one answer: local, persistent, auditable

If you want an agent server you can actually leave running, come find me after.

🔗 **https://www.moltis.org**

🐙 https://github.com/moltis-org/moltis • 📚 https://docs.moltis.org

Appendix, if questions go technical

Short answers

- Why Rust: memory safety, one binary, auditable boundaries
- Why local-first: keys, memory, and sessions stay under your control
- Why not just plugins everywhere: supply-chain risk and harder auditing
- Why persistence matters: useful agents need memory, checkpoints, and continuity
- Why the OpenClaw import matters: migration without cliff-edge risk