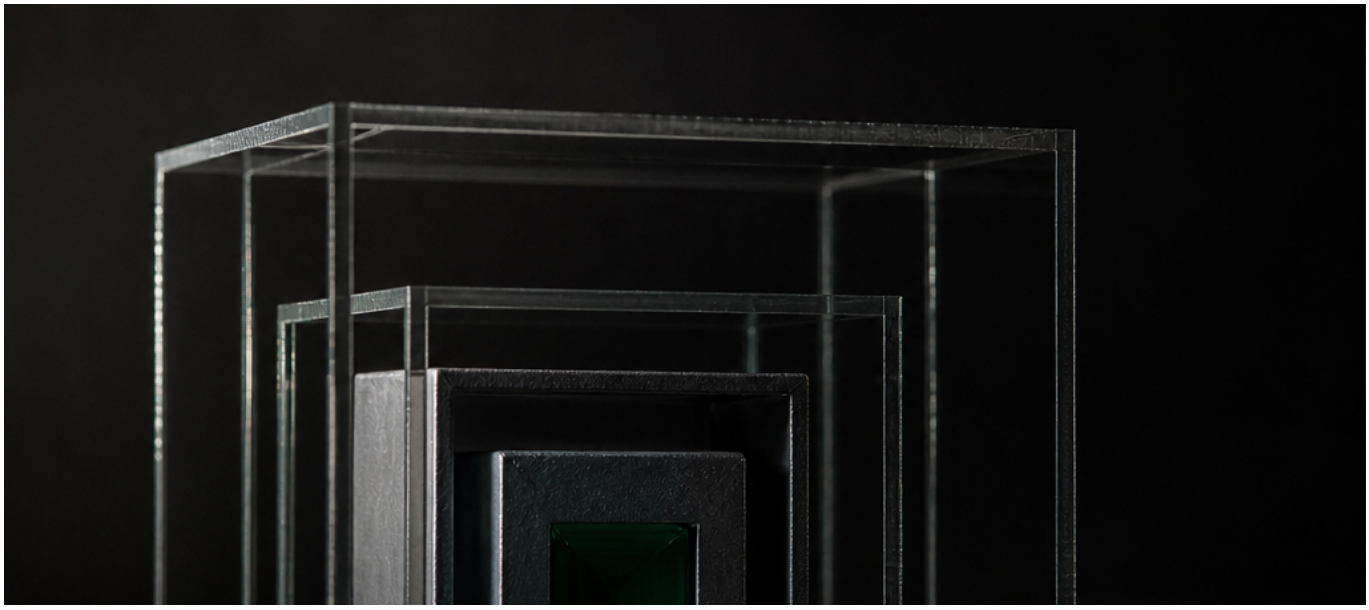


sandbase-harness v0.3.8 in Practice: What a Local-First Agent Runtime With an MCP Bridge Does, and What It Does Not

The most useful thing in this repository is not the sandbox. It is the compatibility matrix that tells you, entry by entry, where the sandbox stops being a wall.

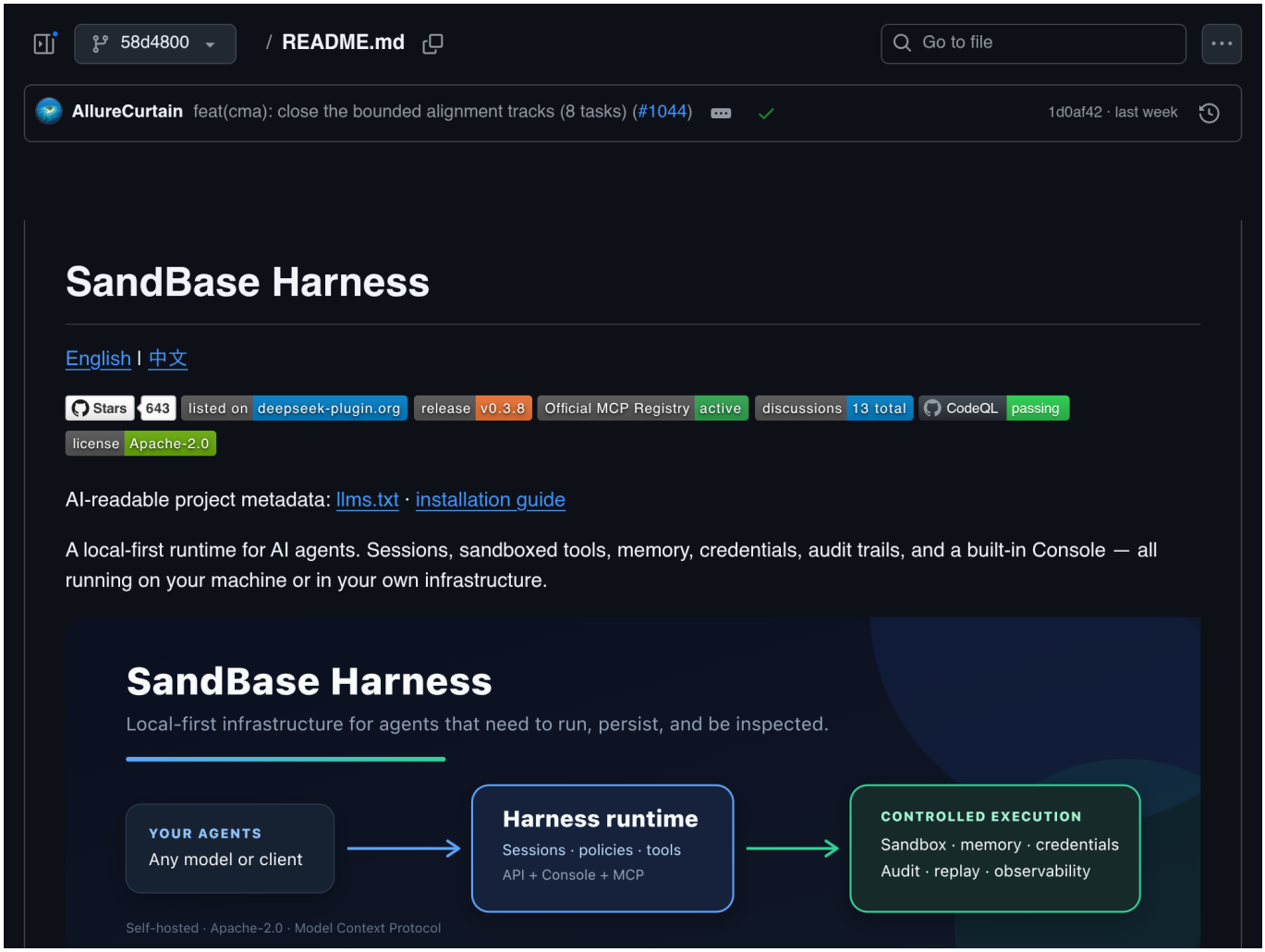
sandbase-harness is a local-first, self-hosted agent runtime in TypeScript that bundles sessions, sandboxes, memory, credentials, an MCP bridge and audit/replay into one process. It sits at v0.3.8 (published 30 August 2026) with 643 stars, and the project states plainly that only the Docker and Kubernetes backends isolate the workload from the runtime host. Local and self-hosted worker are best effort by construction.

The repository description calls it a “local-first, self-hosted AI agent runtime and MCP bridge with sandboxed sessions, memory, credentials, audit/replay, and a local Console.” That is accurate and unusually literal. The README frames the problem the way I would: agent SDKs handle the model loop, and everything else (persistent sessions, tool

governance, sandbox boundaries, credential handling, memory, auditability, a UI for humans) is the part teams currently assemble from four or five separate projects. This repo collapses that into one Node process with SQLite metadata and local file storage. It was created 11 July 2026, last pushed 22 November 2026, and carried 77 forks and 13 open issues and pull requests at fetch time.

Install is a pinned source checkout, and the README is explicit about why: `git clone --branch v0.3.8 --depth 1`, then `npm ci`, `npm run build`, then `init` and `start` from a separate workspace directory. The unscoped `managed-agents` name on npm is somebody else's package, and the README tells you not to run `npx managed-agents` or `npm install managed-agents`. Requirements are Node 22+ and npm 10+. Docker is optional and only needed for Docker-backed sandboxes.

> ! **The caveat that defines the review**
> The v0.3.8 release notes end with one sentence: "The local sandbox remains intended for trusted development only; use Docker or Kubernetes for stronger isolation." The sandbox backend table in the usage docs says the same thing in columns: local process is not isolated from the runtime host, has the host workspace, and has no resource limits.



The sandbaseai/sandbase-harness README on GitHub.

Only two of the four sandbox backends isolate anything

Docker runs per-session containers over the `docker` CLI with resource limits and no host workspace. Kubernetes uses `kubectl exec/cp` with the same properties. Local process runs a child process on your machine with the host workspace mounted and no limits. Self-hosted worker is a queue external workers claim.

The network story follows the same split, and the compatibility matrix spells it out rather than rounding up. Under a limited network policy, Docker attaches the session container to an `--internal` network whose only permitted peer is a relay sidecar forwarding to a per-session egress proxy, which the matrix describes as “enforced, not advisory.” Kubernetes

pins the Pod's egress with a per-session NetworkPolicy, "enforced by the CNI." Local and self-hosted inject proxy environment variables, and the entry says exactly what that is worth: "a process that ignores proxy variables egresses freely." The Environment read surfaces this as a `networking_enforcement` field with values `enforced`, `best_effort`, `unsupported`, or `not_applicable`.

That field is the design decision I would point to if someone asked whether this project is serious. A runtime that reports its own enforcement level as data, per environment, is a runtime you can write a policy against. One that says "sandboxed" in the README and stops is not.

WHAT WORKS

- Honest capability reporting: 110 official SDK method/path pairs declared entry by entry in `src/core/capabilities/matrix.ts`, 90 mounted, 20 refused by name, none deferred. A contract-honesty test fails when the generated table drifts from the matrix.
- Tool approval is a real gate. `permission_policy.type` takes `always_allow`, `always_ask`, `never_allow` and an opt-in `auto` that parks the call on any evaluation failure rather than allowing it.
- An external veto hook: set `MANAGED_AGENTS_EXTERNAL_AUTHZ_ENDPOINT` and every governed call the local policy admits is POSTed to your control plane first. Timeouts (2000 ms default), unreachable endpoints and malformed responses all refuse the call.
- Refusals are persisted as `agent.external_authorization` audit events carrying reason code and SHA-256 digests, with no raw arguments and no secrets.
- Secrets round-trip safely in settings: a stored literal key reads back as `********`, and that sentinel means "keep the stored value" on write.
- The official Anthropic TypeScript SDK drives the runtime unchanged, and that quickstart runs on every pull request via `tests/conformance/official-sdk-quickstart.test.ts`.

ROUGH EDGES

- Pre-1.0 at v0.3.8, and the release line is young: v0.3.6, v0.3.7 and v0.3.8 all landed within ten days in late August 2026.
- No verified one-line install. One git clone path, zero verified Docker, pip or npm commands in the README, even though the v0.3.6 and v0.3.7 notes publish GHCR images with digests.
- Every Settings V2 field requires a full runtime restart to take effect. The API exposes `saved_config` and `effective_config` separately so you can see the gap, but it is still a restart.
- Threads, workflows, advisors and inline agents are refused by name. Delegation executes as nested turns, and the five published thread routes answer `unsupported_capability`.
- Session budgets price from an operator-supplied cost profile rather than official list prices, and a session whose model the profile cannot price is refused a budget outright.
- Audit/replay is named everywhere, but the retention policy, storage format and redaction behaviour are not documented in the returned repo excerpts. Resumable SSE for replay is listed as supported; the governance around the log is not described.
- Planned adapters (S3, mem0, MemU, Harness, Codex, Claude) are listed as unavailable until their implementations exist.

Where the Awesome Copilot rejection got it wrong, and where it did not

An Awesome Copilot submission was rejected as a "purely paid service." The maintainers dispute this in discussion #116, pointing at the Apache-2.0 licence and the self-hosted design, and GitHub's own licence detection on the repository reads Apache-2.0. No paid tier, pricing page or commercial plan appears anywhere in the repository. The maintainers also state no SandBase hosted SaaS account is required: plugins use Docker stdio and the operator's own `MANAGED_AGENTS_URL` endpoint.

My read: the rejection was wrong on the facts and understandable on the signal. A project named after a company, with a `SandBase` prefix on every surface and an environment variable pointing at a managed endpoint, reads commercial at a glance. The reviewer did not read past the glance. That is a packaging problem the maintainers can fix in a week, and the licensing question is already settled.

The ecosystem section of the v0.3.8 notes is the part I would treat with the most caution, and to the maintainers' credit they treat it the same way. They list submissions to the Hugging Face agent-harness registry, Agent Switchboard and MeshKore as "pending review and are not endorsements or security certifications," and they say of an auto-indexed MCPVault listing that "its automated quality signals are not maintainer certification." Directory presence is distribution. It is not assurance, and most projects at this stage blur that line.

Run it with Docker or Kubernetes, or do not call it isolation

If you are wiring MCP servers into something that touches production data, the part of this repo that earns attention is the combination of a named enforcement level per environment, a fail-closed external authorization hook, and a tool approval policy that parks on evaluation failure. That is a governance spine. Set against the state of MCP server authentication, having any of this in one self-hosted binary is above the median by a distance.

What I would not do is run the local sandbox against untrusted model output and call it isolation. The project tells you not to. If your deployment cannot give each session a Docker container or a Kubernetes Pod, you have a runtime with good audit and a child process, and the audit will record exactly what that child process did to your home directory.

WHERE I LAND

My take, unconfirmed and a reading rather than a documented fact: the restart-on-every-settings-change constraint plus the pre-1.0 tag suggests the storage and effective-config layer is still moving. I would pin the tag, treat upgrades as migrations, and hold off building a control plane on the `/v1/x` extension routes until 1.0, since the canonical `/v1` surface is the one under contract test.

WHAT TO DO WITH THIS

Try it if you need one self-hosted process for MCP tooling with real approval gates and an auditable refusal trail, and you can run Docker or Kubernetes sandboxes. Skip it if you need a hosted control plane, thread-level orchestration, or a documented retention and redaction policy for the audit log, because none of those exist yet. If you are deciding where agent isolation sits in your architecture before you commit to a runtime, that is a conversation worth having early. Book a call →



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 →