KIRK

AN ENTERPRISE GUIDE

WHITE PAPER · JUNE 2026

The Enterprise Guide to Private AI Infrastructure

A practical reference for CIOs, CTOs, and general counsel at regulated mid-market firms — what private AI is, why it now matters, and what its infrastructure looks like.

By **Skyler Truax**, Kirk Tech Solutions

June 2026

kirktechsolutions.com

Innovation-driven IT consultancy — AI, cloud, digital transformation, eCommerce, MedTech.

WORKED EXAMPLE

flatclaw

CONTENTS

Executive Summary

The category converged in six months

Between January and June 2026, the frontier labs and Microsoft converged on a single product shape. Anthropic's Claude Cowork defined it. Google's Gemini Enterprise Agent matched it. OpenAI's GPT-6 with Atlas unified it into a super-app. Microsoft Scout, announced at Build on June 2, 2026, brought it inside the enterprise envelope. Four products from four vendors in six months, with the same pre-integrated capability set: chat, agents, scheduled tasks, skills, per-agent memory, RBAC, a human approval queue, and document RAG. The category is no longer in dispute. The deployment posture is.

The data-locality gap

Every one of those products is structurally cloud-hosted. Each request, document, memory write, and tool invocation routes through the vendor's servers under the vendor's policy plane. For a regulated mid-market buyer that posture is unreachable. GLBA, HIPAA, attorney-client privilege, NCUA, family-office NDAs, and the GDPR/PIPEDA/NIS2/DORA/ISO 27001 stack each constrain the same wall. The architecture the vendors sell cannot resolve it.

The architectural answer

The private-AI agent harness is the answer the category is converging on: single-tenant, deployed end-to-end into infrastructure the customer owns and bills, running open-weight models on dedicated GPU silicon, with mechanically verifiable zero vendor egress. "Mechanically verifiable" is the load-bearing phrase — a packet capture on the tenant's egress interface, not a contractual representation. The implementation vendor is an operations contractor, not a node in the data path.

LangChain's framing, *Agent = Model + Harness*, is the cleanest articulation of what the category is actually shipping: the model is the intelligence, and the harness is every piece of code, configuration, and execution logic around it that makes the intelligence useful. FlatClaw's claim is the sentence the enterprise needs next: if you're not the model, you're the harness, and if you're the enterprise harness, you're also the policy surface.

flatclaw

Why the open-source runtime is enterprise-ready

The marquee evidence is Microsoft Scout itself. Microsoft's June 2, 2026 announcement credits OpenClaw open-source technology as Scout's runtime, with Microsoft contributing policy conformance upstream. Peter Steinberger, in Alex Heath's *Sources* newsletter, was blunter: Scout *is* the OpenClaw gateway. An independent contractor can pull, audit, and deploy the same harness — roughly 180,000 GitHub stars within three months of its January 2026 launch — into a single-tenant project today.

Who this guide is written for

This guide is for CIOs, CTOs, general counsel, and operations leaders at regulated mid-market firms who need the coworker shape without the cloud-tenant compromise. The chapters that follow define the category against on-prem and BYOC, walk the harness anatomy and where FlatClaw adds the policy surface LangChain leaves out, lay out a vendor-neutral reference architecture across the harness layers, address compliance and audit, and decompose the economics against the May 2026 frontier API rate card. The honest trade-offs are called out in the economics chapter, not buried.

Kirk Tech's worked example

Kirk Tech Solutions publishes this guide. FlatClaw, Kirk's productized take on the pattern, is the worked example named where this document needs one. FlatClaw is an agent harness with RBAC, deployed single-tenant in the customer's own cloud account.

The new wave of enterprise AI coworkers

Between January and June 2026, four of the largest software vendors shipped what is structurally the same product.

Anthropic's Claude Cowork defined the category. Google's Gemini Enterprise Agent was the identical-shaped response. OpenAI's GPT-6 and the Atlas browser fused the coworker and the operating environment into a single super-app. Microsoft closed the convergence window at Build 2026 on June 2, when Omar Shahine introduced Scout — internally Project Lobster — and Satya Nadella branded the category "Autopilot."

The convergence is the story. Four product orgs, working from different model strengths and distribution surfaces, arrived at the same eight-capability envelope: persistent-transcript chat, a fleet of named agents, scheduled tasks, per-agent memory, document RAG, per-user connected services, tool-call-layer RBAC, and a human-approval queue. Voice and image generation sit alongside as optional add-ons. The shape is no longer a hypothesis, it is the settled answer, and the only remaining questions are deployment posture, identity model, and where the data sits.

What "Autopilot" actually means

Nadella's word choice draws a line against the previous generation of copilots. Shahine, quoted in *Computerworld*, described autopilots as agents that stay active in the background and take action without needing to be prompted each time. The 2024 copilot was a chat box that waited for a human turn. The 2026 coworker has its own identity, schedule, approval queue, and memory of what it did last week. The eight capabilities are the minimum substrate required to make an always-on agent useful and governable at the same time.

Scout's architectural posture

The Microsoft 365 Blog post is unusually direct: Scout is powered by OpenClaw, and Microsoft is contributing policy conformance directly upstream. Every Scout instance gets its own Entra identity, with credentials scoped per task and redacted from logs. Purview labels and DLP rules are evaluated before an action executes, not after. Microsoft's engineering site describes the runtime as zero-trust: identity, tokens, and policy live outside the agent's container, and the container itself is treated as hostile. External applications are reached over MCP; browser interaction is a first-class tool. Private preview began June 2 for M365 Frontier customers, with GA reportedly targeting October 2026 as an E3/E5 add-on.

The harness-layer story

The load-bearing beat is that Scout does not merely resemble the open-source harness the rest of the category is being built on — it *is* that harness. Peter Steinberger, quoted in Alex Heath's *Sources* newsletter, put it plainly: not OpenClaw-like, it is the OpenClaw gateway. When a hyperscaler with the resources to build anything in-house picks an open-source agent harness, ships it as its flagship

enterprise agent product, and signs up for upstream contributions, the harness has crossed from interesting project to default substrate. The same layer is what Hostinger, Moltworker, AnythingLLM, and Hermes Agent wrap for the mid-market, and what FlatClaw, Kirk Tech's productized take on the pattern, sits on as well. OpenClaw's ~180,000 GitHub stars in three months is not a research-project curve; it is the shape of infrastructure the ecosystem has decided to standardize on.

Why the shape converged

The product orgs converged because the constraints converged. An always-on agent needs stable identity or its actions cannot be audited; scheduled execution or the "without being prompted" property collapses back to a chat box; per-agent memory or every run starts cold; tool-call-layer RBAC because the UI is no longer where decisions happen; an approval queue because no compliance regime will let an autonomous agent ship binding outbound work unilaterally; RAG and per-user connected services because the agent's operating context is the firm's corpus and the user's actual Gmail, Drive, and calendar, not the model's pretraining. Strip any one out and the product reverts to a 2024-era chatbot.

What differs across the four is deployment posture: which cloud the data passes through, which identity provider gates the agent, and whether the harness layer is open source or closed. The capability set is settled. The interesting decisions have moved one layer down.

The data-locality gap

A cloud-tenant coworker is a network application before it is anything else.

Every chat turn, every tool invocation, every retrieval against a connected document store, every memory write — the buyer's bytes leave the buyer's network and land on the vendor's servers, where they are processed, logged, and retained for some contractual window. This is the default posture of Claude Cowork, Gemini Enterprise Agent, GPT-6 / Atlas, and Microsoft Scout. For a marketing team at a consumer brand, that posture is unremarkable. For a regulated-financial-services compliance officer, a hospital privacy officer, a managing partner at a mid-size law firm, or a family-office general counsel, it is the end of the conversation.

The gap is not about whether the vendor is trustworthy in the abstract. The regimes these buyers operate under do not award credit for vendor trustworthiness. They award credit for control: who holds the data, who can read it, who logged the read, and whether the chain of custody can be reconstructed for a regulator who is not interested in the vendor's marketing claims.

GLBA and the financial-services exam floor

For community banks, credit unions, and adjacent financial-services firms, GLBA sets the baseline for nonpublic personal information and the relevant IT examination program sets the operational floor for proving it. Examiners look for segregation of duties and an audit trail that ties every action to a named identity, timestamp, and sign-off chain. A coworker that fields a front-line query about a customer's balances, or stages a core-system transaction, has already moved PII across the perimeter and is staging a privileged action against the core. The examiner question is not "did the vendor promise not to misuse it," it is "where does the data live, who has standing to read it, and is the staging gated by a logged human-in-the-loop that survives detach of the AI layer." For a hosted coworker, the honest answer is "the vendor's cluster, under the vendor's access policy." That is not a posture compliance counsel can paper over with a DPA.

HIPAA, BAAs, and the healthcare back-office

In healthcare back-office workflows, the question that ends most cloud-AI procurement reviews is whether the vendor will sign a BAA. A BAA from the substrate provider is the floor, not the ceiling. The moment the agent is invoking tools against PHI — pulling claim status, drafting an appeal, reconciling an EOB, summarizing a chart — the surface area expands from "model inference" to every downstream system the agent touches on the patient's behalf. Each touch needs to be inside the covered entity's control plane, with a log that names the user, the tool, the patient identifier, and the result. A cloud-tenant coworker cannot keep PHI inside the practice's perimeter while still answering the question, because the answer is computed on the vendor's silicon.

Attorney-client privilege

Small and mid-size law firms run on a different kind of risk: privilege is the foundation of the engagement, not a regulatory checkbox. Sending privileged documents to a third-party AI provider is a bet that the provider's data handling is secure, that subpoena response will preserve privilege, and that no training pipeline, evaluation harness, or operator-access path will surface client work product outside the firm — and the lawyer remains responsible for confidentiality regardless of which vendor's terms of service the file passed through. That responsibility is much easier to discharge when the file never left the firm's infrastructure.

Family offices and boutique NDAs

Family offices and boutique consultancies operate under a contractual regime that is often stricter than statute. A family-office mandate routinely prohibits transit of beneficiary data, deal documents, or trust structures through any vendor cloud, with no carve-out for "AI features." Boutique strategy and research shops sign client NDAs that name specific clouds — and increasingly specific AI providers — as off-limits for client work product. A hosted coworker that processes a client deck on the vendor's servers is a breach of contract, full stop.

GDPR, PIPEDA, NIS2, DORA, and EU health data

Any mid-market firm with international subsidiaries or EU customers inherits a second layer: GDPR on processing basis, PIPEDA on cross-border disclosure, NIS2/DORA on resilience and ICT-third-party risk, EU health-data rules on residency, ISO 27001 underneath. A cloud-tenant coworker cannot answer the residency question in a way that scales across these regimes simultaneously. The vendor can offer a region, a sub-processor list, and a model card; what it cannot offer is a deployment in which the customer is the data controller and processor of record on infrastructure the customer's own auditor can inspect.

The gap

The buyer who would extract the most value from the coworker product shape — document-heavy law firms, scheduled-work-heavy healthcare back-offices, integration-heavy financial-services firms, family offices that are all three — is, with uncomfortable regularity, the same buyer who cannot put any of that corpus on a hosted vendor's servers. The product shape and the regulatory shape line up; the deployment shape does not.

> The way out is not a better DPA or a tighter retention window. It is to deploy the same product shape — chat, agents, scheduled tasks, per-agent memory, document RAG, connected services, RBAC, human approvals — into infrastructure the buyer controls, where the data never leaves the perimeter, the audit log lives next to the action it describes, and egress is mechanically verifiable rather than contractually promised. That is the architecture the rest of this document specifies.

What private AI really means

Private AI refers to systems — particularly large language models and the agents built on them — that are deployed, operated, and controlled entirely within the customer's own infrastructure rather than via a third-party provider.

That definition is borrowed, almost verbatim, from Qanswer's working description of the category, and it holds up better than most. The qualifier that matters is **entirely**. A workload is not private because the vendor says it is private; it is private when the four things a regulated buyer actually cares about — where the model runs, where the data goes, who controls the version, where the logs live — sit inside a boundary the buyer can draw on a network diagram and prove with a packet capture.

A five-point operational test

A deployment counts as private AI when all five are true. The first four characterize where the workload lives; the fifth is what separates real portability from a more isolated cage.

1. **The model runs on infrastructure the customer owns or leases.** Bare metal, a single-tenant cloud project, or a dedicated GPU billed to the customer — not a shared inference pool the vendor operates.
2. **The customer's data is never sent to an external AI provider.** No prompts, completions, embeddings, or tool-call payloads cross the boundary toward Anthropic, OpenAI, Google AI, or Hugging Face. A hyperscaler as compute substrate is fine; an AI vendor in the request path is not.
3. **The customer controls the model version, update cadence, and access permissions.** Upgrades are deliberate and evaluable, not silent server-side swaps. Access is governed by identity systems the customer already operates.
4. **All audit logs and conversation data remain within the customer's security perimeter.** Per-tool, per-action rows in a database the customer owns, surfaced to the customer's SIEM — not "available on request" from a vendor portal.
5. **The model weights are open, portable, and re-deployable.** If the vendor disappears, the weights stay on disk and the deployment keeps running.

The synonym thicket

Private AI, sovereign AI, on-premise AI, self-hosted AI, single-tenant SaaS, and bring-your-own-cloud are used interchangeably in vendor decks, and they should not be. This paper uses private AI as the umbrella criterion and treats the others as variations underneath it.

On-prem AI — physical hardware in a facility the customer operates. Clears the test by construction, at the cost of a hardware refresh cycle and a GPU ops hiring plan.

Self-hosted AI — software the customer runs on infrastructure they control, on-prem or in a leased cloud project. Most deployments described as on-prem are actually this.

Single-tenant SaaS — vendor-managed and dedicated. Fails points two and four: the vendor still terminates the traffic and holds the logs.

BYOC — the vendor's software in the customer's cloud account. Decisive question: does the vendor retain a control plane into the deployment? If yes, the vendor is in the data path under a different name.

Sovereign AI — private AI plus a jurisdictional overlay, for regulatory rather than security reasons.

Open weights versus private deployment of a closed model

The fifth test quietly disqualifies the largest category of "private" offerings. A hosted closed-weight model deployed into a customer's cloud can clear the first four cleanly — weights in the customer's account, data inside the boundary, logs local — but the customer does not have the weights as an asset they can redeploy elsewhere. If the provider deprecates the version, raises the license, or changes the terms, options collapse to renegotiation. Private deployment of an open-weight model — Gemma 4, for instance — clears the fifth test too: the weights can be copied, archived, audited, and run on a different substrate next quarter.

The criterion this paper adopts

The load-bearing rule is short: **your cloud, your bill, your control plane**. The customer holds the cloud account and is the GPU lessee; the implementation vendor is an operations contractor, never in the data path at runtime, compensated by a fixed engagement fee rather than a per-token overlay. Kirk Tech's FlatClaw, used later as the worked example, implements that rule literally: FlatClaw is deployed into the Azure, AWS, or Google Cloud account the customer already holds, the hyperscaler bills the customer directly, and Kirk Tech is paid a separate fixed retainer. The substrate could be a different cloud and the implementation vendor a different firm; the criterion does not move.

The comparative this frames

The open-source agent harness that powers Microsoft Scout — OpenClaw, sandboxed under Entra identity and Purview policy inside Microsoft's cloud — is the same harness a regulated mid-market firm can deploy on a dedicated H100 in its own cloud project, against an open-weight model the customer controls. The technology is shared; the deployment posture is the variable. Private AI is the name for the posture where the boundary the buyer needs to defend, and the boundary the workload actually lives inside, are the same boundary.

The anatomy of a private AI coworker

By mid-2026 the category had converged on a fixed set of pre-integrated capabilities, and any product that ships fewer reads as incomplete to the regulated mid-market.

Chat, agent fleet, scheduled automation, skills and tools, persistent agent memory, role-based access, approvals, and document RAG — plus voice and image as optional ninth and tenth. The first eight are not optional. The frame LangChain has put on it is the clearest in the field: *Agent = Model + Harness*. The model is the intelligence. The harness is every piece of code, configuration, and execution logic that isn't the model itself — state, tool execution, feedback loops, and enforceable constraints. What follows walks the harness anatomy as a category requirement, and calls out where FlatClaw adds the two surfaces LangChain's enumeration is silent on: a policy surface and a human-approval queue as a peer of the tool layer.

Chat and the loop

Chat is the surface every entrant ships, and precisely because it is the surface, it carries the audit posture. The requirement is not conversation — that is solved — but that conversations are durable, multi-session, replayable, and addressable by a per-action audit log. Each turn has to land somewhere a compliance team can subpoena. Cowork, Gemini Enterprise Agent, GPT-6 + Atlas, and Microsoft Scout all expose transcript history at the chrome layer, but the more useful question is whether the transcript is the audit record or a pretty rendering of one; in Scout's case, the agent container is treated as untrusted and the transcript is a view onto an underlying authoritative log. The loop underneath the chat surface is what LangChain calls the harness proper: the run-tool / observe / decide-next-action cycle that turns a model into an agent. A harness that cannot replay a session for an examiner is not finished.

Agent fleet and orchestration

The agent abstraction supplanted the chatbot abstraction because the unit of delegation changed. A chatbot is a session; an agent is a named worker with its own identity, system prompt, skill roster, and memory. The structural claim of the category is identity, not interface: always-on workers acting under their own principal on the user's behalf. A chatbot has no principal. An agent does. At the harness layer this is the orchestration logic — subagent spawning, handoffs across subagents with isolated context, model routing across the fleet — the place where work gets divided so a long-horizon job does not have to fit inside one model's window.

Scheduled automation

The move from synchronous chat to background work is the move from "ask the agent" to "the agent did it overnight." An accounting practice schedules end-of-month reconciliation: the agent pulls the trial balance at month close, drafts adjusting journals, and parks anything ambiguous in an approvals queue for the partner. Cron is the underlying primitive; the requirement is that scheduling is per-

agent, recoverable across restarts, and produces audit rows identical in shape to interactive runs. This is where long-horizon autonomous execution stops being a model property and starts being a harness property.

The tool layer: skills, tools, MCPs

The tool layer is the executable capability surface the model can call. Skills come in three flavors — built-in, plugin, and connected-MCP — the latter often using progressive disclosure so the model sees only the slice of the tool inventory relevant to the current task. MCP is the load-bearing piece: the open transport contract that makes per-user OAuth tool integrations portable across harnesses. Scout speaks MCP and can drive a browser; FlatClaw uses MCP for per-tenant services. The category has settled, and a tool written once is reachable by any compliant harness. The substantive engineering question moved up a layer: who holds the OAuth token, where is it stored, and how is it injected into the skill process without ever crossing the LLM prompt boundary.

The state surface and per-agent memory

LangChain calls the filesystem the most foundational harness primitive and a natural collaboration surface — the durable storage that lets context survive past the turn-level budget of any context window. Persistent memory is the same idea raised to the agent level. The requirement has three parts: persistent across sessions, search- recallable, and scoped per agent so a fleet serving different roles inside the same tenant cannot contaminate each other. OpenClaw's built-in implementation runs BM25 keyword search over per-agent markdown stores; semantic recall via embeddings is roadmap. Both modes belong in a mature implementation. BM25 is precise and explainable and serves the audit posture; semantic retrieval catches paraphrase. The category will end up with both, layered, with citations. Around memory sits the harness's broader context management — compaction and tool-call offloading, what LangChain calls the fight against Context Rot — that keeps performance from degrading as the session grows.

The policy surface: RBAC at the tool-invocation path

LangChain's enumeration is silent on RBAC, governance, and approval queues. That silence is the wedge enterprise harnesses have to fill. The most common implementation mistake is to gate at the chrome. The UI hides a button, the agent's roster still lists the tool, and a determined prompt elicits a call the operator believed was disallowed. The category requirement is that RBAC is a policy surface on the harness, enforced at the tool-invocation path, inside the loop that decides whether to dispatch a tool call at all. The agent cannot lie about its tools because the tools were never in its roster to begin with. The credential layer rides the same contract: OAuth tokens scoped per `(tenant, user, service)`, redeemed by the skill over loopback, never visible to the LLM prompt or the network boundary.

The approval surface: human-approval queue

The approval queue is the second piece of the policy surface — the cross-service mechanism that gates any tool call, not just shell commands, and routes the decision to either the requester or a different role. Any MCP tool can compose a controlled action and return a pending-approval envelope instead of executing. Routing is per-policy: `self` for actions where the requester signs off, `role` for actions that must travel to a different principal. A staff member drafts a first-touch outbound to a new client and the request lands in the senior reviewer's queue; the staff member's own queue stays empty, because segregation of duties is enforced by routing, not by policy hope. The harness requirement is that the primitive exists as a peer of the tool layer, and that it gates tool calls rather than chat messages.

The observability surface: audit and document RAG

The final required capabilities sit on either side of the observability surface. Cited retrieval over the customer's own corpus is grounded in documents the customer holds, not the vendor's web index; RAGFlow handles ingest and retrieval in FlatClaw's reference architecture, wrapped as an OpenClaw skill, with citations as the load-bearing detail — an answer that cannot be traced back to a specific paragraph in a specific document is not a private-AI answer. The audit log underneath records every tool call, every approval decision, every memory write: debuggability upgraded to enterprise-grade attestation. Voice and image complete the surface as optional ninth and tenth capabilities. Their omission is not a category failure. The eight above are.

A reference architecture for private AI infrastructure

The convergent product shape sits on a stack that looks the same whether the vendor is Anthropic, Google, OpenAI, Microsoft, or a customer-owned deployment. The differences live in deployment posture, not in the layer inventory.

The architecture is most naturally read as a walk through the harness layers — planner / loop / tools / state / memory / policy / observability — with FlatClaw's concrete instantiation alongside. The structural antecedent at the layer below this one is Canonical's Ubuntu AI / Charmed Kubeflow inventory: storage, silicon, networking, OS, container orchestration, data, MLOps, open source as the substrate, vendor as commercial-support wrapper. The architecture below addresses the layers that sit above Canonical's orchestration tier, on the same posture.

The ten layers, top to bottom

1. **Substrate** — the cloud project the whole harness lives in.
2. **Silicon** — the GPU class the model server actually runs on.
3. **Model server / inference** — the open-weight model plus the serving stack.
4. **Agent harness / gateway** — the loop, orchestration, tool execution, scheduling, sandboxing.
5. **State surface / per-agent memory** — persistent, search-recallable, per-agent isolated.
6. **Document RAG** — ingest, retrieval, cited answers, wrapped as a tool.
7. **Policy surface** — RBAC + per-user credentials + capability-token bridge.
8. **Approval surface** — cross-service, role-aware human-approval queue.
9. **Observability surface** — per-action audit rows, tcpdump-provable egress.
10. **Provisioning + teardown** — one project per tenant, scripted.

Substrate

The substrate owns the network boundary every other layer lives inside. The defining design choice is single-tenancy: one project per customer, one bill per customer, customer holds the cloud account and the IAM root. Anything multi-tenant at this layer — shared VPCs, shared API gateways, shared databases keyed by `tenant_id` — inherits the egress and blast-radius profile of a hosted SaaS, regardless of what the rest of the architecture claims. FlatClaw's instantiation is a dedicated project inside the Azure, AWS, or Google Cloud account the customer already holds, with the implementation contractor holding only an operator seat; the GPU plan and egress meter bill directly to the customer, and the SOC 2 / ISO 27001 / HIPAA-eligible posture of the hyperscaler is the compliance umbrella the rest of the harness inherits.

Silicon

The silicon layer is where the choice of which open-weight model the harness can serve is made or foreclosed. The practical floor for a 30B-class dense FP8 model at production context lengths is an 80 GB native-FP8 card. Older cards without native FP8 hardware run quantized models only through a Marlin-style fallback, and that fallback breaks on specific projection widths in newer architectures — which is why the model choice and the card choice are one decision, not two.

Model server / inference

The inference layer is an open-weight model plus a serving stack that does continuous batching, prefix caching, and tool-call parsing. Continuous batching is the difference between a single stream at conversational latency and a regime where one card keeps dozens of streams in flight without collapsing per-stream latency. Prefix caching — RadixAttention in SGLang, automatic-prefix-cache in vLLM — is what makes a long-context system economic when most of the prompt is the agent's prior turns and tool outputs. Tool-call parsers are the model-specific glue that turns the assistant's emitted JSON into a routable tool call.

Agent harness / gateway

The agent harness is where the loop lives: session state, tool dispatch, scheduled tasks firing, sandboxes spinning up to verify work, RBAC enforced at the tool-call path rather than the UI. The category has converged on this layer being separate from both the model server below it and the UI above it — separable because the same harness needs to drive a chat UI, a scheduled task, and a connected MCP service with the same session and tool semantics. In LangChain's vocabulary this is the loop wrapped around the tool layer, the orchestration logic for subagent handoffs, the hooks and middleware around context compaction, and the bundled infrastructure that gives the model a computer to work on.

This is the layer where the marquee comparison lands. Microsoft Scout, launched at Build 2026 as the enterprise Autopilot product, runs the same open-source agent harness: Microsoft's own posts describe Scout as powered by OpenClaw and commit to upstreaming policy conformance, and Peter Steinberger, in Alex Heath's *Sources* newsletter, called it the OpenClaw gateway rather than a lookalike. Scout wraps that harness under Entra identity and a hardened Hyper-V sandbox; FlatClaw wraps it in a single-tenant project on a customer-owned H100 serving the open-weight model the customer chooses on SGLang at that model's full native context window per session, plus in-tenant memory, RAG, and connected services. Same foundation at the harness layer, different deployment posture. The architectural point is that the harness layer of this category is now a shared open-source primitive, and the differentiation has moved up to the substrate, down to the silicon, and sideways into the policy surface.

State surface and per-agent memory

Memory at this layer is per-agent, not per-user and not per-tenant. The unit is the agent — a named persona with a prompt, a tool roster, a transcript history, and a persistent memory store that survives across sessions and is searchable from inside the agent's own tool calls. Two retrieval modes have

settled in: keyword for cheap exact recall, and dense semantic for paraphrased recall. The point of memory living in the harness rather than in the model context is that a long context window is a turn-level budget, not a knowledge store. Memory is what survives compaction. LangChain's framing of durable storage and the filesystem as a natural collaboration surface across subagents lands cleanly here.

Document RAG

The RAG layer is corpus ingest, chunking, embedding, retrieval, and cited-answer assembly, wrapped as a tool the agent invokes rather than a chat product the user talks to directly. The shift from "chat over your documents" to "rag-search skill the agent calls when it needs to" is what makes RAG composable with the rest of the tool roster. FlatClaw uses RAGFlow surfaced to the agent as one more skill, gated by the same RBAC and approval primitives as every other tool.

Policy surface: RBAC, per-user credentials, and the capability-token bridge

This is the load-bearing security layer and the one LangChain's enumeration leaves open. It is also the one that most often gets shortcut into UI-only gating. The right design has three pieces: tool-call gating at the harness rather than at the UI, credentials scoped per user and per service rather than per tenant, and a capability-token bridge that hands the skill a short-lived credential at invocation time rather than letting it read raw secrets from the environment.

Harness-level gating puts policy on the same code path the loop uses for its own decisions, not in a sidecar that has to be kept in sync. Per-user scoping matters because the connected service is the user's — one staff member's Gmail token is not another's. The bridge closes the last gap: the gateway mints a per-(user, service) capability token, injects it into the skill's process environment, and the skill calls back to a loopback-only endpoint to exchange it for a freshly minted, narrowly scoped service credential at use time. The raw service credential is never in chat, never in an LLM prompt, never in a transcript.

Approval surface: human-approval queue

The approval queue is the primitive that lets any tool call be gated by a human signature before it executes, and it has to be cross-service and role-aware, not bolted onto a single skill. The shape that has settled in: any MCP tool can compose the controlled action, return a pending-approval envelope instead of executing, and the queue routes that envelope by an approver policy field — `self` for actions the requester signs off on, `role` for actions that route to a different role's queue. The architectural beat: the approval surface is a peer of the tool layer, sitting above the loop, not buried inside it. Most harnesses' native approval primitives are shell-command-shaped, designed to gate code execution from a coding agent, and cannot gate an arbitrary MCP tool call without re-introducing the kind of before-tool-call hook that proper harness-level RBAC just retired.

Observability surface, provisioning and teardown

The audit log is per-action — every tool call, every approval decision, every credential mint — and it has to survive the detach of demo or pilot modules so the substrate stays the load-bearing record. The egress story is tcpdump-provable: the architecture either sends bytes to a vendor or it does not, and a packet capture on the tenant's egress interface is the only mechanically honest way to assert it. Provisioning and teardown are the scripts that turn the layered architecture into a one-command operation; a reference architecture that requires hand-laying every pod is not a reference architecture, it is a runbook waiting to be automated.

Open source as the foundation

Every layer of the private-AI harness pattern — the harness itself, the model, the serving stack, and the retrieval engine — is now available as production-grade open source under permissive licenses.

That was not true eighteen months ago, when the open stack ended at the model weights and everything above them was a research artifact or a closed framework. It is true today, and the evidence is the shape of the marquee enterprise adoption.

The layered OSS inventory

The harness is OpenClaw, which provides the loop, session state, per-agent memory, tool dispatch, sandboxing, scheduling, and approval workflows. The model is the customer's choice of open weights; the harness is model-agnostic, and the reference configuration runs Gemma 4, Google's open-weight release. The serving layer is SGLang, carrying the FP8 path that native H100-class silicon requires. The retrieval layer is RAGFlow, paired with bge-m3 embeddings. Around that core sit the usual supporting OSS components for the portal surface, the projection store, voice, image, and managed web retrieval. None are placeholders waiting for a commercial replacement; each is what currently ships in production deployments under a permissive license.

Microsoft Scout as the marquee endorsement of the harness layer

The clearest signal that OpenClaw is now the canonical open-source agent harness is that Microsoft picked it. The Microsoft 365 Blog announcement of Scout on June 2, 2026 describes the product as powered by OpenClaw and notes that Microsoft is contributing policy conformance directly upstream; Peter Steinberger's confirmation on X, reproduced in Alex Heath's *Sources* newsletter, removes any ambiguity that it is the OpenClaw gateway itself, not a fork.

This is the structural move that matters. Microsoft wrapped an OSS harness under Entra identity and Purview policy and shipped that as Scout, choosing to standardize on the harness layer and compete on the envelope around it. The agent harness is not a place a hyperscaler wants to differentiate anymore; standardization is what lets identity, policy, and audit live cleanly above it. FlatClaw uses the same harness, with RBAC built in for buyers who need policy at the action layer.

The convergence signal

OpenClaw launched in January 2026 and reached roughly 180,000 GitHub stars in three months. Beneath the Microsoft case sits a mid-market cohort — Hostinger, Moltworker, AnythingLLM, and Hermes Agent — shipping chat-UI products that are structurally wrappers over hosted LLMs sitting on the same harness. When the hyperscaler entrant and the bootstrap-funded wrapper cohort pick the same gateway in the same year, the harness question is answered.

Audit-and-fork posture

The license terms are themselves the trust artifact. The harness, serving stack, and retrieval engine are Apache 2.0 or MIT; the model weights carry their own license, which is one more thing to read when choosing the model — in the reference configuration, Gemma Terms layered over an Apache-style grant. Apache 2.0 carries the explicit patent grant, the language enterprise counsel actually reads. There are no source-available licenses, no Business Source carve-outs, no "free under $X revenue" clauses anywhere in the harness graph. Pull it. Audit it. Run it. Fork it if a future license change makes you uncomfortable.

Appliance versus platform

The category is converging on one packaging decision that open-source availability does not by itself answer: ship the components pre-integrated as one product, or ship them as a kit the customer assembles. The platform posture asks the customer's platform team to wire OpenClaw, SGLang, RAGFlow, and the model together. The appliance posture treats the pre-integrated capability surface — chat, agents, scheduled tasks, skills, per-agent memory, RBAC, approvals, document RAG — as one product with one upgrade cadence. The mid-market regulated buyer this paper is written for does not have a platform team to assign to the kit.

Structural alignment with Canonical Ubuntu AI

The closest existing analogue at scale is Canonical's. Their "Productizing AI: an enterprise guide to private AI infrastructure" describes a layered stack across storage, compute, networking, OS, orchestration, and MLOps — open source as the substrate at every layer, Canonical's commercial offering positioned as integration, support, and packaging. The agent-harness pattern sits one layer above, addressing the loop, serving, retrieval, and the per-user policy surface; the posture is identical: open stack, vendor contract for implementation and operations, customer owns the substrate.

> Kirk Tech's FlatClaw is one such appliance: the same OpenClaw harness that powers Microsoft Scout, shipped pre-integrated with RBAC and the approval surface built in, serving the open-weight model the customer chooses, and deployed in the customer's own cloud account.

Compliance, audit, and segregation of duties

The hyperscaler carries the infrastructure-level certifications, because that is where the hypervisor, network, storage, and physical data center actually live. The per-action controls live in an in-tenant audit log. Together, those are the two facts a regulated-SMB procurement reviewer needs from a private-AI deployment.

Substrate certifications carry the procurement story

In a single-tenant private deployment, the cloud provider is the one party in the data path with infrastructure-level certifications to carry: SOC 2 Type II, ISO 27001, HIPAA with a BAA. The implementation contractor is not in the data path, so the certifications that matter for "where does our data sit?" are the substrate's, not the contractor's.

FlatClaw, for example, runs inside the Azure, AWS, or Google Cloud account the customer already holds — which means the customer's existing hyperscaler agreements, BAAs, and compliance boundary carry over unchanged. The cloud provider bills the customer directly, never Kirk Tech. The hardware, the project, and the bill all sit on the customer's side of the trust boundary. This is the structural answer to a recurring procurement question — *who else has access to our data?* — because the contractor's deploy-and-operate role does not require holding the data.

Zero egress as a test, not a claim

The privacy posture of a private AI deployment is only as strong as its weakest outbound connection. The architecture's claim — that no model request, embedding call, tool result, or document chunk leaves the tenant — has to be mechanically verifiable, or it is marketing.

At runtime, the hosted AI providers, hosted vector databases, and third-party inference endpoints are not talked to. Model server, embedder, agent harness, and document index are all in-tenant, with no remote weights pull at request time and no telemetry shipped to a vendor analytics endpoint. The proof recipe is a `tcpdump` on the tenant egress interface during a representative session; the expected result is zero bytes to any named third party — a claim a security reviewer can run themselves on the live deployment.

One append-only audit log, per-action rows

The audit substrate is a single append-only table that records per-action, per-tool, per-approval rows. Approval decisions land as rows with `approval.approved` or `approval.denied`, a tool-call ID that makes the decision idempotent against double-clicks and retries, and a confirmation reference written on approve. The pending approval queue is derived: items recorded in the transcript, minus items decided in the log. No separate ledger that can fall out of sync with the substrate of record — which matters because the audit log has to survive operational change as demo modules get detached, MCP services get swapped, and RBAC tables get reshaped during pilots.

The cross-service human-approval queue as the policy surface

Segregation of duties in the agent layer is the requirement that no model can execute a controlled action without a human in the loop, and that the human is sometimes a different human from the one who asked. This is what LangChain's harness enumeration is silent on, and where enterprise harnesses have to fill in: a policy surface on the harness, expressed as a single primitive that any MCP tool can mark human-approval-required by returning `PENDING_HUMAN_APPROVAL` plus an `approval` envelope. The agent has composed the exact downstream request — the real wire-shape, the real arguments — but parked it.

Routing is set per policy on the envelope: `self` for the everyday staff-member-signing-off-on-their-own-action pattern, or `role` to route to a named role's queue with the requester labeled. On approve, the queue calls the owning service's execute hook before the audit row is written, so a failed execution leaves the item pending rather than recording a phantom approval. Either way, the decision is durable, idempotent on the tool-call ID, and tied to both requester and approver.

Worked example: the professional-services back-office pilot

FlatClaw's professional-services back-office pilot exercises both routing modes against the everyday outbound surface of a mid-market firm. A staff member on Tier 1 — an associate, an analyst, a bookkeeper, an account executive, whatever the cohort calls the production-level individual contributor — drafts a routine reply on an existing matter to an existing external counterparty; the MCP composes the verbatim outbound message and parks it as `PENDING_HUMAN_APPROVAL` with `approverPolicy: self`, the item lands in their own Approvals tab, the execute hook fires on sign-off, and a confirmation reference is written. A first-touch outbound to a new external recipient, or an external document share, uses the same primitive with `approverPolicy: role` targeting the Tier 2 senior reviewer; the item routes to the reviewer's queue labeled with the originating staff member, and the audit row names both parties. A binding outbound commitment — an engagement letter, a fee proposal, an invoice over the firm's threshold, a signed deliverable — routes one tier further, to the Tier 3 managing principal.

That is the examiner-grade story in one walkthrough: the AI cannot ship a controlled outbound action without a human sign-off at the appropriate tier, the agent composed the exact downstream request and parked it, and the audit log lets the managing principal replay any matter end-to-end with actor, persona tier, matter, timestamp, and the exact payload that would have left the tenant.

Economics: flat rate versus token metering

A two-to-ten-person content shop in 2026 is rarely on a single AI bill.

The working stack is stitched: ChatGPT seats, a Claude API key, Midjourney, ElevenLabs, Jasper, Apify, Zapier, Buffer. Counted honestly — seats plus metered API plus per-render and per-second charges — the all-in lands between $900 and $2,800 per month, with the variance driven almost entirely by how much the team actually uses the metered surfaces. The seats are predictable. The meters are not. The bill that arrives is a function of how productive the team was, which is exactly the wrong shape for a CFO planning headcount against software spend.

The flat-rate alternative inverts that: a single dedicated GPU, a single open-weight model, a single monthly invoice that does not move when the team writes more.

The May 2026 rate card, output-dominant

MODEL	PER MILLION OUTPUT TOKENS
Claude Haiku 4.5	$5
GPT-5.3 Codex	$14
Claude Sonnet 4.6	$15
Claude Opus 4.7	$25
GPT-5.5	$30
Self-hosted H100, FP8, at 70% utilization	**~$1.35**

List prices, May 2026. Input runs four to six times cheaper across the board.

That asymmetry matters because output dominates real bills: agents read a corpus once and emit a long answer, then a longer follow-up, then a longer revision. The headline number to budget against is the output column.

The self-hosted cost curve

A single H100 serving Gemma 4 31B in FP8, the reference configuration, has a measurable cost per million output tokens that depends on one variable: how much of the day the card is working. At 70% utilization — a realistic single-tenant steady state — ~$1.35/M. At 50%, ~$1.90/M. At 30% the cost doubles; at 15% it quadruples. Other models move the figures; utilization is the whole story, and vendor decks gloss it.

Against the rate card, $1.35/M sits roughly 11× under Sonnet 4.6 and 18× to 22× under Opus 4.7 and GPT-5.5. The honest headline is the range, not a single number — a "20× tax" framing is correct as a thesis at sustained volume, but the multiplier on any given workload depends on which tier of the

frontier ladder it would have been routed to.

Why the gap is structural

Four multipliers compound on top of hardware: the underlying GPU cost ($1–$2/M at high utilization), spare-capacity overhead for multi-tenant burst headroom (1.5–2×), the orchestration and billing control plane (1.5–2×), and venture-funded margin (3–5×). Compound those and the 10× to 25× delta falls out of the arithmetic. It is not waiting on a price war to close. The single-tenant flat-rate side avoids the second, third, and fourth multipliers by construction.

Where the breakeven sits

A single H100 sustains roughly 1,260 output tokens per second at concurrency 128, supporting 8 to 12 concurrent streaming chats and a per-tenant ceiling around 2 billion output tokens per month at 60% utilization. The same volume billed against Sonnet 4.6 would arrive as a ~$30,000 monthly invoice. Below the breakeven the API is structurally cheaper; above it, every additional token is paid for under the flat-rate envelope at no marginal cost.

Kirk Tech's FlatClaw is one instance of this shape: a single-tenant H100 deployment in the customer's own cloud account, with the hyperscaler billing the customer directly at no markup, alongside a fixed retainer that scales with engagement complexity rather than usage. The harness is the same code Microsoft Scout uses on the multi-tenant side; the deployment posture is what determines the cost structure. Per-token billing is one consequence of choosing multi-tenant shared compute. It is not the only one — fragmenting the model's context window across a shared user pool is another — but it is the one that shows up on the invoice.

Where self-hosting is the wrong answer

Below approximately 100 million output tokens per month, the per-token API is almost always correct. A team running a few thousand agent turns a day pays for headroom it will not use; the GPU sits at 15% to 30% utilization, the per-token cost doubles or quadruples, and amortization fails.

For the hardest reasoning tasks, the frontier API is still better. Open-weight models in the Gemma 4 class clear the bulk of day-to-day agent work, but Opus 4.7 and GPT-5.5 still pull ahead on the longest-horizon planning and the most adversarial proof-style tasks. The architectural answer is not provider loyalty in either direction — it is route by complexity. The open-weight model handles 70 to 80% of traffic at the flat-rate cost; the frontier API is reserved for the hardest fraction, explicitly escalated.

> **The rule of thumb.** Below ~100M output tokens/month, use the API. Above ~1B/month, evaluate self-hosting. The band in between is where data-locality posture, not the cost curve, usually decides.

Reference deployments

One pilot is useful to walk through, because it runs the FlatClaw appliance on the architectural pattern in the shape every ICP cohort in scope would recognize.

The substrate is invariant. The interesting variation is upstream of the gateway, in how the tenant's systems and roles get mapped into MCP services and tool policies — and the example below is deliberately written so the same architecture lands on a law firm, an accounting practice, a healthcare back office, a family office, a boutique consultancy, or a marketing agency without rewriting any of it.

The professional-services back-office pilot

Picture a representative mid-market professional-services firm of roughly forty to two hundred people — the kind of organization that looks structurally identical whether it is a regional law practice, an accounting and bookkeeping group, a multi-clinic healthcare back office, a single-family or multi-family office, a boutique strategy consultancy, or a content and marketing agency. Work arrives by email and shared drive, is governed by a small number of senior reviewers, and is ultimately signed off by a managing principal. The firm already has a corpus of prior matters, templates, and engagement artifacts sitting in a shared drive, and every staff member has their own mailbox and calendar. The firm wants leverage on inbound volume without giving up the segregation-of-duties controls their profession — or their insurer, or their regulator — requires.

The deployment is a per-user agent that reads inbound client correspondence, drafts replies and artifacts against the firm's own document corpus, and routes every outbound action through a persona-tiered approval queue with a full audit log. Three persona tiers sit on top of one shared MCP service, and RBAC lives inside the MCP rather than in a separate admin path.

> **Tier 1 — Staff.** Associate, analyst, bookkeeper, coordinator, account executive — whatever the cohort calls the production-level individual contributor. Owns day-to-day matter execution. Can ask the agent to draft replies and artifacts on matters explicitly assigned to them. Cannot release anything externally without an approver of the appropriate tier signing off.

> **Tier 2 — Senior reviewer.** Senior associate, manager, lead clinician, senior advisor, account director. Owns a book of matters and supervises Tier 1. Self-approves their own routine outbound mail; approves first-touch outreach and external document shares for Tier 1; can see and act on every matter inside their book but not across the firm.

> **Tier 3 — Managing principal.** Partner, managing partner, practice owner, family-office principal, agency principal. Firm-wide visibility. Sole approver for binding outbound commitments, engagement letters, fee proposals, signed deliverables, and invoices above the firm's threshold. Sees the full audit log.

The agent's capability set is the universal back-office shape: read inbound external correspondence from each user's own mailbox and thread it onto the matter it belongs to; draft outbound replies in the firm's voice, citing prior matters surfaced by RAG over the in-tenant shared-drive corpus; maintain per-agent, per-matter memory so each ongoing engagement has a durable running summary of where it stands, what was promised, and what is owed back; search the firm's document corpus — templates, prior engagements, redlines, working papers, case notes, whatever the cohort's equivalent is — and pull the right precedent into the draft; schedule follow-ups on the user's own calendar; prepare client-facing artifacts staged in the shared drive but never released until an approver signs off; and triage inbound mail by urgency and matter, with everything timestamped to the audit log.

The controlled actions and their default approvers are the segregation-of-duties spine. A routine reply on an existing matter to an existing external counterparty is self-approved by the assigned staff member, logged. A first-touch outbound to a new external recipient or a new matter thread routes to the Tier 2 senior reviewer. Any external document share — sending an attachment, granting external drive access, posting a deliverable — routes to Tier 2, always. Scheduling an external meeting on the user's own calendar is self-approved by the assigned staff member, logged. A binding outbound commitment — engagement letter, fee proposal, signed deliverable, invoice above the firm's threshold — routes to the Tier 3 managing principal. Any modification or deletion of an entry in the firm's matter memory or audit log routes to Tier 3, with the original entry preserved. Same MCP, same data, role determines response shape, no separate admin path, no parallel principal codebase.

Each user's connected services are their own: per-user Gmail or Outlook with that user's own credentials, per-user Google Calendar or Microsoft 365 Calendar with the same, and the firm's shared drive (Drive, OneDrive/SharePoint, or equivalent) accessed under each user's own permissions, so the agent never sees more of the corpus than the user themselves can. The in-tenant pieces sit underneath: the document RAG index over the shared drive, the per-matter memory store, and the cross-service approval queue and audit log.

The audit-and-live-swap path

The pilot ships first against a synthetic sandbox: a seeded mailbox, a seeded calendar, and a sanitized copy of the firm's corpus with names and figures scrubbed. Every drafted reply, every proposed document share, every scheduled follow-up, every approval decision, and every memory write is recorded to the in-tenant audit log with actor, persona tier, matter, timestamp, and the exact payload that would have left the tenant. The managing principal can replay any matter end-to-end from the log alone — this is the audit-grade story their insurer, regulator, or board wants to see before anything goes live.

The live-swap path is then deliberately boring. The persona-tiered RBAC, the approval routing, the memory store, the RAG index, and the audit log are all already wired against the production shape. Going live means swapping the sandbox mailbox credential for the user's real mailbox credential, the sandbox calendar credential for the real one, and pointing the drive connector at the real shared drive — one credential per service, per user, no rewrites to policy, prompts, or controlled-action wiring. The same approval queue that gated synthetic sends now gates real ones.

What the pilot closes on

The deployment converges on three claims. Same MCP, same data, role determines response shape — no separate admin path, no parallel principal build. Every word of analysis was written by the in-tenant open-weight model on that tenant's H100, not a frontier API in the background, not a routed fallback. And the deployment posture is verifiable with a packet capture rather than a marketing claim: a `tcpdump` on the tenant egress while the workload runs shows no traffic to Anthropic, OpenAI, Google AI, Hugging Face, or any third-party inference endpoint. One appliance, one architecture, the same shape across every cohort in the ICP, one mechanical proof.

An adoption path for private AI

Most regulated mid-market buyers do not need a transformation program to adopt a private-AI agent harness. They need a sequence: discovery, scoped pilot, production expansion, ongoing operations. Four stages, sequenced, not parallelized.

Discovery

Discovery answers three questions. First, what is the workload — which roles, against which downstream systems, at what monthly volume? Second, what is the data-locality posture, scored against the five operational tests established earlier in this paper; a buyer who fails three of the five against a hosted incumbent has already made the case. Third, a route-by-complexity split: most routine work clears cleanly on an open-weight 30B-class model, with the hardest reasoning honestly routed to a frontier API. The output is a single page: workload inventory, posture score, routing split.

Scoped pilot

The pilot stands up one project on the substrate, one persona group, two or three MCP services, and a small RAG corpus. The professional-services back-office pilot template is illustrative: three persona tiers, the everyday outbound surface a mid-market firm actually runs on, and the cross-service approval queue exercised end-to-end across self-routed, senior-reviewer-routed, and managing-principal-routed approvals. The pilot is not a demo. It is the buyer's own data — or a sanitized copy of it on the audit-and-live-swap path — the buyer's own role boundaries, and a tool roster narrow enough that the agent's behavior is legible to the compliance officer reading the audit log.

Production

Production widens along the dimensions the pilot tested: the second persona group comes online, and the remaining MCP services get registered and gated through per-user RBAC. The scheduled work the buyer actually bought the appliance for ships — weekly executive brief, end-of-month reconciliation, daily ops digest — and the security review gets its evidence in the form of an egress capture demonstrating no traffic leaves for any third-party inference endpoint, filed in the audit binder next to the SOC 2 report.

Ongoing operations

The commercial structure is two contracts, not one. The cloud provider — Azure, AWS, or Google Cloud, whichever account the customer already holds — bills the customer directly for the dedicated GPU and supporting services. The implementation contractor runs a separate fixed monthly retainer covering tenancy management, model and software upgrade cadence, and on-call. The retainer scales with engagement complexity — personas, services, approval depth — not with the infrastructure bill. The contractor does not mark up hosting and is not in the data path.

About Kirk Tech Solutions

Kirk Tech Solutions is an engineer-led implementation and operations practice for private AI infrastructure. The firm designs, deploys, and operates single-tenant agent-harness stacks built on OpenClaw — with FlatClaw as its productized take on the pattern, an agent harness with RBAC, deployed single-tenant in the customer's own cloud account — and adjacent private-AI work for buyers whose data-locality posture rules out the hosted incumbents. The technical work is the firm's whole surface area: no resale margins, no managed-services overlay, no consulting-deliverable layer between the engineer and the substrate.

The commercial relationship reflects that posture. The customer holds the cloud account directly and pays Azure, AWS, or Google Cloud directly for the dedicated H100 and the supporting services that make up the per-tenant infrastructure bill. Kirk Tech does not mark up hosting and is not in the data path at runtime. Kirk's revenue is a separate fixed monthly retainer covering implementation, ongoing operations, RBAC tenancy management, and the model and software upgrade cadence. The retainer scales with engagement complexity, not with the customer's compute bill.

About the author

Skyler Truax is the engineer-founder of Kirk Tech Solutions and the builder of FlatClaw, the firm's productized private-AI agent harness with RBAC. He designs and operates the OpenClaw-based deployments described in this paper end-to-end: substrate provisioning, model serving, the agent harness, RBAC and approval primitives, MCP integrations, and the upgrade cadence that keeps the open-source stack current in regulated tenants.

NEXT STEP

Private AI, secure, at a fixed cost.

If your data cannot or should not leave your network, the answer is not a stricter contract with an AI vendor — it is a deployment posture where there is no one to send data to. We size, deploy and run that stack inside infrastructure you already operate, at a cost that does not move with how much your team uses it.

Schedule a Private Architecture Review

Whether you want to calculate your self-hosting break-even point with our team or run through the details of your next project, we can help. Bring a workload, a posture, and a volume estimate — we will bring the architecture and the honest read on whether self-hosting is the right answer.

REQUEST EXECUTIVE BRIEFING