OpenClaw Today, Tomorrow, and the Gaps In Between

A Deep Technical Whitepaper on Current Deficits, Strategic Direction, and Concrete Proposals

Author: Gabriel, Chief AI Correspondent, SMF Works
Date: 2026-06-26
Version: 2.0 — Expanded Edition
Classification: Internal technical whitepaper
Word count target: ~12,000 words

Abstract

OpenClaw has become the most visible open-source personal AI agent gateway in the world, with over 380,000 GitHub stars and millions of weekly npm downloads. Its popularity rests on a deliberately simple architecture: a single Node.js gateway process, a SQLite-backed state store, a WebSocket control plane, and a rich ecosystem of messaging channels, tools, skills, and plugins. That simplicity has enabled explosive growth, but it also creates structural deficits that become more consequential as users ask OpenClaw to run always-on, multi-agent, semi-autonomous operations.

This paper provides a deep technical audit of OpenClaw as of release `2026.6.10` (commit `aa69b12`) and the `2026.6.11` pre-release. It draws on four evidence streams: official project artifacts, live operational data from SMF Works' DGX Spark deployment, 2026 GitHub issue and pull-request data, and comparative analysis with peer frameworks (LangGraph, CrewAI, Mastra, n8n, Dify, AutoGPT, and several research systems). The analysis identifies five major deficit categories: reliability and silent failures; memory and state governance; security and trust boundaries; scale, high availability, and state durability; and agent-loop architecture.

For each deficit, the paper proposes concrete, implementable remedies ranging from near-term patches to multi-quarter architectural bets. The proposals are designed to preserve OpenClaw's core virtues — local-first control, hackability, broad channel support, and terminal-first UX — while making it trustworthy enough for teams, autonomous agents, and safety-critical workflows. The paper concludes with a phased implementation roadmap, a risk and trade-off discussion, and an extended bibliography.

1. Introduction: Why OpenClaw Matters, and Why Now

1.1 The Personal AI Gateway Thesis

OpenClaw sits at the intersection of three powerful trends: the rise of frontier language models, the proliferation of messaging channels, and the resurgence of local-first computing. By running on a user's own machine (or a personally controlled server) and connecting to the channels they already use, OpenClaw turns a language model from a chat interface into an always-available personal assistant. It can read files, run shell commands, browse the web, schedule tasks, and communicate back through WhatsApp, Telegram, Slack, Discord, Signal, iMessage, or WebChat.

This is a compelling product thesis, and the market has responded. OpenClaw's GitHub star count passed 300,000 in mid-2026 and continued climbing toward 400,000. Its npm package is downloaded millions of times per week. The project has attracted a large ecosystem of skills, plugins, channel integrations, and community documentation.

But popularity is not the same as maturity. OpenClaw is, by its own admission in `VISION.md`, "still early, so iteration is fast." The project prioritizes security and safe defaults, bug fixes, setup reliability, and first-run UX. Those priorities are correct, but they are also reactive. This paper argues that OpenClaw needs a forward-looking architectural agenda to match its forward-looking product ambition.

1.2 Scope and Method

This paper examines OpenClaw as both a piece of software and an operational system. It asks: what breaks, why does it break, and what would make it break less often or recover faster?

The evidence base is intentionally mixed:

- **Official artifacts:** `VISION.md`, `README.md`, release notes, architecture docs, security docs, multi-agent docs, cron docs, memory docs, and sandboxing docs.
- **Live operational data:** direct inspection of SMF Works' DGX Spark deployment on 2026-06-26, including `openclaw doctor`, `openclaw cron list`, the SQLite state store, and agent directory structure.
- **GitHub issues and PRs from 2026:** specific failures reported by users, with issue numbers, labels, and maintainer responses.
- **Comparative analysis:** how other agent frameworks and workflow platforms solve analogous problems.

The goal is not to criticize OpenClaw but to identify gaps that are visible now and will become critical as usage scales. Every proposal is grounded in observed behavior or documented risk.

1.3 The Argument in Brief

OpenClaw's deficits are not random bugs. They share a common root: the architecture is optimized for a single trusted operator on a single machine, but it is increasingly used for multi-agent teams, 24/7 autonomous tasks, and shared infrastructure. The paper argues for three architectural commitments:

1. **Observability first:** every failure, especially silent failures, must become a visible, queryable, actionable signal.
2. **Verified trust:** skills, plugins, memory providers, and tool invocations must be auditable and bounded by least privilege.
3. **Optional durability:** high availability, replicated state, and structured agent loops must be available without forcing them on personal users.

These commitments are compatible with OpenClaw's culture if implemented as opt-in layers on top of the existing single-node foundation.

---

2. Background: The OpenClaw Architecture as Documented

2.1 The Gateway as Single Control Plane

OpenClaw's architecture is built around a single long-lived Gateway process. The Gateway is the only process that opens a WhatsApp Baileys session, the only process that maintains WebSocket connections to clients and nodes, and the only process that schedules cron jobs. It exposes:

- A typed WebSocket RPC API for CLI, web UI, macOS app, Windows Hub, and iOS/Android nodes.
- An HTTP server for the Control UI, canvas host, and A2UI host on port 18789 by default.
- A SQLite state database containing sessions, cron jobs, run history, pairing records, and configuration.

The Gateway is intentionally single-node. The architecture docs state: "One Gateway per host; it is the only place that opens a WhatsApp session." This makes setup simple and avoids distributed-system complexity, but it also means the Gateway is a single point of failure for every connected surface.

2.2 Agents, Workspaces, and Isolation

An OpenClaw agent is a fully scoped persona with:

- A workspace directory (files, `SOUL.md`, `AGENTS.md`, `USER.md`, notes, skills).
- A state directory (`agentDir`) containing auth profiles, model registry, and per-agent config.
- A session store under `~/.openclaw/agents/<agentId>/sessions`.

Agents are isolated from each other by workspace path and `agentDir`. Auth profiles are per-agent. Skills are loaded from each agent workspace plus shared roots, then filtered by skill allowlists. Bindings map channel accounts to agents.

However, the docs are explicit that workspace isolation is not a hard sandbox. Relative paths resolve inside the workspace, but absolute paths can reach other host locations unless sandboxing is enabled. Skills loaded into an agent run with that agent's tool policy, which defaults to full host access for the main session.

2.3 The Agent Loop

The agent loop is the core execution path: intake → context assembly → model inference → tool execution → streaming replies → persistence. OpenClaw serializes runs per session key to prevent races. A global lane caps overall concurrency. Tool events, assistant deltas, and lifecycle events are streamed over WebSocket.

Hooks allow plugins to intercept the loop at multiple points: `before_model_resolve`, `before_prompt_build`, `before_agent_reply`, `agent_end`, `before_tool_call`, `after_tool_call`, `before_compaction`, and others. The hook system is powerful but event-driven rather than plan-driven. There is no built-in representation of a multi-step plan that can be inspected, paused, or replayed.

2.4 Cron and Tasks

Cron runs inside the Gateway process and persists job definitions and run history in SQLite. It supports `at`, `every`, and `cron` schedule kinds, and can deliver payloads to `main`, `isolated`, `current`, or named sessions. On startup, overdue jobs are rescheduled so that channel reconnect storms do not drown the system.

The cron system has several resilience features: startup catch-up, transient retry with backoff, failure alerts, per-run session reaping, and a watchdog for stalled isolated runs. But it is still fundamentally tied to the Gateway process and the SQLite file. If either fails, scheduling stops.

2.5 Memory

OpenClaw's memory system indexes Markdown files in the workspace and optionally session transcripts. It supports two backends:

- **Builtin SQLite:** vector + BM25 hybrid search, no extra dependencies.
- **QMD sidecar:** local-first search combining BM25, vector search, and reranking.

QMD is optional and requires a separate binary. OpenClaw manages QMD collections automatically and falls back to the builtin engine if QMD fails. Memory search results can include citations (`Source: <path#line>`).

2.6 Skills and ClawHub

ClawHub is the public registry for OpenClaw skills and plugins. Skills are versioned text bundles containing `SKILL.md` plus supporting files. Plugins can be code plugins (deeper runtime extension) or bundle-style plugins (packaged skills, MCP servers, config). OpenClaw has native commands to search, install, and update skills and plugins from ClawHub.

ClawHub runs automated checks and supports moderation. But the registry itself has experienced availability issues, including an expired SSL certificate and Vercel bot protection blocking CLI requests at the time of this writing.

2.7 Security Model

OpenClaw's security model is explicitly personal-assistant-first. The docs state that OpenClaw is "not a hostile multi-tenant security boundary for multiple adversarial users sharing one agent or gateway." The recommended pattern for mixed-trust teams is separate gateways (or at least separate OS users/hosts).

Tools run on the host by default. Sandboxing is opt-in and supports Docker, SSH, and OpenShell backends. Exec approvals, DM pairing, and channel allowlists provide guardrails, but they are operator-intent guardrails rather than strong isolation.

---

3. Deficit Category I: Reliability and Silent Failures

3.1 The Severity of Silence

In a personal assistant, the worst failure mode is not a crash; it is the absence of expected behavior with no visible error. The user assumes the assistant is working. Hours or days later, they

discover that messages were not received, cron jobs did not run, or emails were not sent. Silent failures erode trust faster than noisy ones because they are not immediately actionable.

OpenClaw has multiple surfaces where silent failures occur:

- **Channel layer:** transport connects but application registration fails.
- **Cron layer:** jobs fail repeatedly but remain enabled, with per-run errors buried in SQLite.
- **Memory layer:** embeddings route to the wrong provider endpoint with no warning.
- **Model layer:** model allowlist changes silently disable jobs that used a previously allowed model.
- **Plugin layer:** broken extensions produce non-fatal startup errors on every CLI invocation.

This section focuses on channel and cron silent failures, which have the highest user-facing impact.

3.2 Case Study: WhatsApp Native Channel Mute (Issue #97060)

On 2026-06-26, GitHub user `bernardoparus-ia` filed issue #97060, titled `[Bug]: WhatsApp Native Channel connects but registered state remains false (Silent Failure)`. The reproduction steps are straightforward:

1. Run `openclaw channels login --channel whatsapp`.
2. Scan the QR code with a phone.
3. The terminal prints " Linked!"
4. Send messages to the bot. The bot never receives them.
5. Inspect `creds.json`: `"registered": false` persists.

The WebSocket transport is connected and heartbeats are exchanged. `messagesHandled` is 0 and `lastInboundAt` is null. There is no crash, no error log, and no health indicator in the CLI or Control UI that tells the operator the channel is non-functional.

This is a textbook transport/application decoupling bug. The channel layer reports success at the transport level because the WebSocket is open and the QR scan completed. But the application-level registration with WhatsApp/Baileys did not succeed or did not persist. Without a reconciliation loop that checks registration state against transport state, the failure is invisible.

The environment in the issue is OpenClaw 2026.6.9 and 2026.6.10 on an Ubuntu Hostinger VPS, using Gemini 3.1 Pro and an updated `@whiskeysockets/baileys` dependency. The user notes that the WhatsApp account had previously been used with Evolution API, which may have left conflicting state. Even if the root cause is stale state, the failure mode should be surfaced, not swallowed.

3.3 Case Study: Telegram Forum Typing Indicator Stuck (Issue #97042)

Issue #97042, filed by `Frojoe6969`, reports that in Telegram forum/topic mode, after a session completes normally with `status: done`, the typing indicator remains stuck indefinitely. The user must send `stop` to clear it and see the reply. The issue is labeled P1 with `impact:message-loss` and `impact:session-state`. It was closed as `not_planned` by the automated ClawSweeper.

The key quote is: *"Makes multi-agent collective in forum mode extremely frustrating to use."*

This is a session lifecycle bug. The agent run completed, the transcript was committed, but the outbound channel state — specifically the "typing" signal — did not transition. The failure is visible to the user, but it degrades the experience in a way that is particularly harmful for multi-agent collaboration, where multiple agents may be posting in the same forum thread.

The fact that the issue was closed `not_planned` is itself a signal. It suggests either that the fix is non-trivial, that forum mode is a lower priority, or that the automated issue triage is too aggressive. Regardless, the underlying session-state cleanup problem is real and recurrent.

3.4 Operational Evidence: Cron Failure Accumulation

On the DGX Spark deployment, the SQLite state store contains 179 cron jobs. Many are disabled, duplicated, or stale. One job, `Aiona Heartbeat — DGX Spark`, has 137 consecutive errors because its `payload.model` (`ollama/gemma4:31b`) is rejected by the `agents.defaults.models` allowlist. The error is recorded in the job state, but the job remains enabled and continues firing every 30 minutes.

This is not a silent failure in the strict sense — the error is present in `openclaw cron get <id>` — but it is an unactioned failure. The system does not escalate, disable, or alert. The operator must notice the error count and manually intervene. For a deployment with dozens of cron jobs, this is unsustainable.

3.5 Root Cause Analysis: Why Silent Failures Are Systemic

Silent failures in OpenClaw are not isolated incidents. They arise from three architectural choices:

1. **Coarse-grained success signals.** The channel layer reports "connected" based on transport state. The cron layer reports "error" as a row-level status. Neither provides an end-to-end functional signal.
2. **No durable health model.** There is no central, queryable representation of system health. The Control UI has a Logs tab and status commands, but there is no equivalent of Kubernetes pod health checks or Prometheus metrics that aggregate state over time.
3. **Failure handling is per-run, not per-system.** Each run logs its own error, but there is no policy engine that aggregates errors, detects patterns, and triggers remediation.

In other words, OpenClaw has excellent per-run visibility and poor systems visibility.

3.6 Comparative Context

Other platforms handle this differently:

- **n8n:** Workflows have explicit error paths. Every node can route errors to a designated "Error Trigger" workflow. The execution list shows success/failure status per run. Webhook nodes have connection health indicators.
- **Dify:** Scheduled triggers have execution history with clear status, duration, and output. Errors are surfaced in a unified log.
- **LangGraph:** Checkpointed graph state means a failed run leaves a recoverable trace. The graph structure makes it easy to see which node failed and why.
- **CrewAI:** Tasks have explicit status (`pending`, `in_progress`, `completed`, `failed`) and the crew execution log shows per-task outcomes.

OpenClaw lacks an equivalent aggregated health and alerting surface.

3.7 Proposed Remedy 1: Channel Health Reconciliation Loop

Introduce a periodic reconciliation task that compares three states for every configured channel account:

- **Transport state:** is the WebSocket/socket/session open?
- **Application state:** is the bot registered/paired/authenticated with the upstream service?
- **Functional state:** has the channel handled at least one inbound or outbound message in the last reconciliation window?

The reconciliation interval should be configurable, defaulting to 60 seconds for active channels and 5 minutes for idle ones. It should produce a structured health record per channel account:

```
{
  channelHealth: {
    "whatsapp:work": {
      transport: "healthy",
      application: "degraded", // registered: false detected
      functional: "unknown",   // no messages in window
      lastInboundAt: null,
      lastOutboundAt: "2026-06-26T12:00:00Z",
      degradedSince: "2026-06-26T11:00:00Z",
      suggestedAction: "re-register"
    }
```

```
  }
}
```

When transport is open but application or functional state is unhealthy, the gateway should:

1. Mark the channel account as `degraded` or `muted` in `openclaw channels status`.
2. Emit a `channel.health.degraded` event.
3. Attempt bounded self-recovery: re-register, reconnect, refresh token.
4. Escalate to the operator if self-recovery fails.

This is directly analogous to Kubernetes liveness, readiness, and startup probes.

Implementation notes:

- Add a `ChannelHealthMonitor` class in the gateway process.
- Store health records in SQLite with a TTL.
- Expose `channels.health.list` and `channels.health.get` RPC methods.
- Render health in the Control UI as a traffic-light status per account.

3.8 Proposed Remedy 2: Cron Failure Escalation Policy

Add a configurable escalation policy for cron failures:

```
{
  cron: {
    escalation: {
      alertAfterConsecutiveErrors: 3,
      disableAfterConsecutiveErrors: 5,
      alertChannel: "telegram:ops",
      alertWebhook: "https://hooks.example.com/openclaw",
      alertMessage: "Cron '{jobName}' for agent {agentId} has failed {count} times.
Last error: {lastError}",
      backoffPolicy: "exponential", // none | fixed | exponential
      backoffMaxMs: 3600000
    }
  }
}
```

The policy should be overridable per job:

```
openclaw cron edit <id> --escalation disableAfterConsecutiveErrors=10
```

When a job hits the alert threshold, OpenClaw sends a message to the configured channel or webhook. When it hits the disable threshold, the job is disabled and a notification is sent. The disabled job remains in the list with a clear status so the operator can investigate.

Additional commands:

- `openclaw cron audit` lists jobs sorted by failure rate, last success, and drift from schedule.
- `openclaw cron health` shows a summary of healthy/degraded/disabled jobs.

3.9 Proposed Remedy 3: Operator Health Dashboard

Extend the Control UI with a health dashboard. The dashboard should show:

- Channel account status with traffic-light indicators and last functional activity.
- Cron job status with failure-rate sparklines and last error.
- Model provider reachability and fallback history.
- Memory backend readiness and last index time.
- Recent silent-failure events (channels that passed liveness but failed functional probes).
- Agent directory reconciliation status (orphaned directories, missing entries).

The dashboard should be driven by the same health records used by the CLI, so operators can use either surface. It should also support exporting health metrics in Prometheus/OpenTelemetry format for integration with external monitoring.

3.10 Proposed Remedy 4: Synthetic Probes

For critical channels and cron jobs, allow operators to configure synthetic probes:

- A WhatsApp probe sends a message to a known test contact and expects a reply.
- A cron probe runs a lightweight job every N minutes and verifies it completes and delivers output.
- A model probe calls a cheap model endpoint and measures latency and success rate.

Probe results feed into the health model and dashboard. This is how real production systems validate end-to-end functionality rather than relying on transport liveness alone.

---

4. Deficit Category II: Memory and State Governance

4.1 The Problem

Memory is where an agent becomes more than a stateless chatbot. It allows the agent to recall prior conversations, retrieve project context, and maintain continuity across sessions. OpenClaw has invested in memory — both the builtin SQLite backend and the optional QMD sidecar — but the implementation has governance gaps that affect correctness, privacy, and debuggability.

4.2 Case Study: Memory Provider Misrouting (Issue #97055 / PR #97059)

On 2026-06-26, user `Tabytha-Stryker` filed issue #97055: `Provider resolution ignores memorySearch.provider config, routing to wrong Ollama instance`. The user configures a custom Ollama provider for memory embeddings:

```
{
  memorySearch: {
    provider: "ollama-cpu",
    providers: {
      "ollama-cpu": {
        baseUrl: "http://127.0.0.1:11434"
      }
    }
  }
}
```

OpenClaw logs `(requested: ollama-cpu)` but routes the embedding request to whichever Ollama instance responds first on the network — in the reporter's case, a remote inference box. GPU monitoring confirms the wrong endpoint is hit. No error or warning is logged. The request succeeds silently against the wrong provider.

This is a serious bug for several reasons:

- **Privacy:** users who configured a local embedding endpoint specifically to keep data local are unknowingly sending data to a remote endpoint.
- **Correctness:** different Ollama endpoints may run different embedding models or quantizations, producing different vector spaces.
- **Trust:** the config says one thing and the runtime does another, undermining the entire configuration system.

Draft PR #97059 by `harjothkhara` addresses the root cause: the gateway's memory-specific adapter path was not passing the configured provider id into the adapter, causing custom Ollama-backed memory providers to fall back to the base `ollama` provider. The PR is labeled with `merge-`

`risk: compatibility` and `merge-risk: security-boundary`, indicating that the fix touches sensitive routing code.

4.3 Case Study: Tool-Call Transparency (Issue #96254)

Issue #96254, filed 2026-06-24 by `TZJ12`, reports that when using `/api/chat/completions` to interact with an agent, internal tool-call behavior information is not returned. The issue is labeled `clawsweeper:needs-product-decision`, meaning it is unclear whether this is a bug or intended behavior.

The significance is not the missing API field itself. It is the fact that the agent loop's tool calls are internal runtime events, not first-class outputs. For external clients, observability tools, and debugging, this opacity is a real limitation. It also makes it harder to build monitoring, evaluation, or reinforcement-learning loops on top of OpenClaw.

4.4 Operational Evidence: Model Registry Drift

On DGX Spark, `openclaw doctor` reports the following error for four agents:

```
[agents/model-registry] model catalog load issue: Failed to load models.json:
Provider vercel-ai-gateway, model alibaba/wan-v2.5-t2v-preview: invalid contextWindow
```

The file `/home/mikesai3/.openclaw/agents/<agentId>/agent/models.json` contains an invalid entry. The entry is loaded for Gabriel, Morgan, Pamela, and Aiona. This indicates that model metadata is not validated when it is written or loaded, and bad state is propagated silently across multiple agents.

4.5 Root Cause Analysis

The memory and state deficits have three roots:

1. **Provider identity is dropped in some adapter paths.** The gateway resolves providers at the top level, but memory-specific adapters have their own code paths that do not preserve the resolved identity.
2. **Memory is a black box to the agent and operator.** There is no `memory status` equivalent that shows which provider is active, which embedding model is in use, when the index was last updated, or what the search latency is.
3. **State files are loaded without strict validation.** `models.json` and similar per-agent state files are not validated against a schema. Invalid entries cause runtime errors or silent misbehavior rather than being quarantined.

4.6 Comparative Context

Peer systems have moved toward explicit memory contracts:

- **LangGraph:** Provides typed memory schemas. Agents receive a structured `Memory` object with configurable eviction policies. Memory is checkpointed alongside graph state.
- **Mastra:** Has explicit `Memory` and `VectorStore` abstractions with TTL and eviction semantics.
- **CrewAI:** Agents have explicit `memory` configuration with configurable vector store backends and `CrewMemory` for shared context.
- **AutoGPT:** Implements `AlphaMemory` with importance-weighted retention and semantic deduplication.

OpenClaw's memory search tools (`memory_search` , `memory_get`) are functional but do not expose these governance primitives to the agent or operator.

4.7 Proposed Remedy 1: Provider-Identity Tunneling

Every embedding and inference call must carry the resolved provider identity end-to-end:

- Config → model registry → adapter → HTTP request.
- Each hop logs `providerId` , `baseUrl` , and `modelId` at `debug` level, and at `info` when diagnostics are enabled.
- Add `memory provider-audit` to list the last N embedding calls with resolved provider, latency, and status.

This closes the #97055 class of bugs by construction.

4.8 Proposed Remedy 2: Explicit Memory Contracts

Define a memory contract per agent in `openclaw.json` :

```
{
  agents: {
    list: [
      {
        id: "gabriel",
        memory: {
          backend: "qmd",
          provider: "ollama-local",
          embeddingModel: "nomic-embed-text",
          indexPaths: ["memory/", "~/notes"],
          retentionPolicy: {
            kind: "time",
            maxAgeDays: 365
```

```
          },
          citations: "on",
          redaction: {
            secrets: true,
            apiKeys: true
          }
        }
      }
    ]
  }
}
```

The contract is:

- Stored in the agent config.
- Exposed via `memory status --contract`.
- Summarized in the system prompt so the agent knows what memory guarantees it has.
- Enforced by memory governance hooks.

This makes memory inspectable and testable, matching the explicit memory abstractions in LangGraph and Mastra.

4.9 Proposed Remedy 3: Memory Health and Governance Hooks

Add two new hook points:

- `memory:health` — runs periodically and reports index freshness, embedding model readiness, search latency, and provider identity.
- `memory:governance` — runs before each memory write and can redact, classify, tag, or block content based on operator policy.

Example governance policy:

```
{
  memory: {
    governance: {
      rules: [
        { action: "redact", pattern: "sk-[a-zA-Z0-9]{48}", reason: "OpenAI API
key" },
        { action: "tag", pattern: "@example.com", tag: "personal" }
      ]
    }
  }
}
```

4.10 Proposed Remedy 4: Strict State Validation

All agent state files (`models.json` , `auth-profiles.json` , etc.) should be validated against JSON Schema on load. Invalid entries should be:

- Quarantined to `~/.openclaw/agents/<id>/agent/.quarantine/` with a timestamp.
- Reported by `openclaw doctor` with exact line/column information when possible.
- Skipped at runtime rather than partially loaded.

For `models.json` , the schema should enforce:

- Required fields: `provider` , `id` , `contextWindow` .
- Valid `contextWindow` type and range.
- Known provider prefixes or explicit custom-provider declaration.

This prevents the DGX Spark `invalid contextWindow` issue from recurring.

4.11 Proposed Remedy 5: Memory Provenance and Audit Trail

For every memory retrieval used in an agent run, record:

- Query text (or a hash for privacy).
- Search backend and provider.
- Top-K results with paths and scores.
- Whether citations were included in the prompt.

Store this in the run metadata. It enables:

- Debugging why the agent recalled something wrong.
- Evaluating retrieval quality over time.
- Compliance with data-retention policies.

---

5. Deficit Category III: Security and Trust Boundaries

5.1 The Problem

OpenClaw's security model is honest about its scope. It is a personal assistant for one trusted operator per gateway. It does not claim to isolate adversarial users on the same gateway. However, even within that model, two surfaces create outsized risk: the skill marketplace (ClawHub) and the plugin/extension system.

A skill is Markdown-driven instructions plus supporting files that can cause the agent to invoke tools. A plugin is code that runs inside the gateway process. Both are part of the agent's trust boundary. If either is malicious or compromised, it can read files, execute commands, exfiltrate data, or persist backdoors.

5.2 Case Study: ClawHub SSL and Availability Failure (Issue #96820)

On 2026-06-25, user `ars667` filed issue #96820: `[Bug]: ClawHub API returns Vercel Security Checkpoint (403) instead of JSON`. The issue reports:

- `openclaw skills search "calendar"` returns `ClawHubRequestError: ClawHub /api/v1/search failed (403)`.
- `openclaw skills install @<any-package>` fails with the same error.
- `curl -i https://clawhub.io/api/v1/skills/calendar/install` returns a Vercel Security Checkpoint HTML page.
- The SSL certificate on `clawhub.io` is expired.

This is a critical availability and security failure. The entire skill marketplace depends on a TLS connection to `clawhub.io`. With an expired certificate, clients cannot verify the server's identity. A network attacker can intercept skill downloads, inject malicious code, and compromise every agent that installs from ClawHub.

The issue carries the label `clawsweeper:needs-security-review` and `impact:security`. The requested fix is to whitelist the CLI user-agent or disable Vercel bot protection for `/api/v1/*` endpoints. But the SSL expiration is a more fundamental problem than bot protection.

5.3 Case Study: Broken Plugin Degrading CLI

On DGX Spark, every `openclaw` CLI invocation prints:

```
[plugins] openclaw-honcho failed to load from /home/mikesai3/.openclaw/extensions/
openclaw-honcho/dist/index.js:
Error: Cannot find module '@honcho-ai/sdk'
```

This error is non-fatal. The CLI continues. But it illustrates that a broken plugin can pollute every command with noise and potentially alter CLI behavior. There is no automatic quarantine, no load-health command, and no operator notification beyond the console warning.

5.4 The Supply-Chain Risk Surface

Skills and plugins are a software supply chain. Like npm packages, Docker images, or VS Code extensions, they can contain:

- Malicious instructions (prompt injection, social engineering).
- Exfiltration code (network calls to attacker-controlled endpoints).
- Host escape code (commands that read outside the workspace).
- Dependency confusion (pulling packages from npm with typosquatted names).
- Stale vulnerabilities (old dependencies with known CVEs).

Unlike npm packages, skills run inside an agent context where the agent has already been granted tool authority. A malicious skill does not need to exploit a vulnerability; it can simply ask the agent to use tools the operator already allowed.

5.5 Root Cause Analysis

1. **No strong verification of skill origin.** ClawHub is currently an HTTPS registry. There is no signature verification, digest pinning, or provenance attestation.
2. **No default sandbox for skill execution.** Skills run in the agent context, which defaults to host access for main sessions.
3. **No capability manifest.** A skill does not declare what tools and resources it needs, so the gateway cannot enforce least privilege.
4. **Plugin load failures are not isolated.** A broken extension can degrade the CLI experience globally.

5.6 Comparative Context

Other ecosystems have invested in supply-chain security:

- **npm:** `npm audit` scans dependencies for known vulnerabilities. `npm publish` requires 2FA for popular packages. `provenance` attestations are available for GitHub Actions workflows.
- **VS Code:** Extensions run in a separate extension host process with a restricted API surface. Marketplace reviews and reports can remove malicious extensions.
- **n8n:** Community nodes are sandboxed in a separate execution context with explicit permission scopes.
- **Dify:** Marketplace has an approval workflow; community agents undergo automated security scanning.

OpenClaw has moderation hooks and automated scans, but the ClawHub SSL incident shows that the operational security posture is not yet production-grade.

5.7 Proposed Remedy 1: ClawHub Hardening

Immediate actions:

- Renew the `clawhub.io` TLS certificate. Configure automated renewal and monitoring (e.g., Let's Encrypt with cert-manager, or Vercel's managed certificates with expiry alerting).
- Fix or bypass the Vercel bot protection for `/api/v1/*` endpoints so legitimate CLI requests succeed.
- Add a CLI check that refuses to install from a registry with an invalid or expired certificate.
- Publish certificate transparency logs and pinning metadata.
- Stand up a redundant registry mirror or offline signed package cache.

5.8 Proposed Remedy 2: Signed Skill Bundles

Move to signed skill bundles:

- Each skill release includes a manifest file with SHA-256 digests of every file.
- The manifest is signed by the publisher's Ed25519 key or by a ClawHub-wide signing key.
- The CLI verifies the signature and digests before install.
- Add `clawhub verify <skill>` and `openclaw skills verify` commands.

Example manifest:

```
{
  "skill": "@openclaw/demo",
  "version": "1.2.3",
  "files": {
    "SKILL.md": "sha256:abc123...",
    "README.md": "sha256:def456..."
  },
  "signature": "base64...",
  "signingKey": "clawhub-official-2026"
}
```

5.9 Proposed Remedy 3: Skill Supply-Chain Pipeline

Implement a five-stage pipeline for every skill that enters an agent workspace:

1. **Source verification.** Git commit, signed tag, or SLSA provenance.
2. **Static analysis.** AST scan for:
    - Hardcoded secrets.
    - Network calls outside declared domains.

    - File system access outside the workspace.
    - Shell command execution.
    - Suspicious npm/dependency references.
3. **Sandboxed test run.** Execute the skill in a disposable gVisor or Firecracker VM with no host access and a network allowlist.
4. **Policy gate.** Compare observed behavior against a declared capability manifest and operator approval rules.
5. **Immutable install.** Store the approved bundle in the agent workspace. Updates require re-running the pipeline.

This is diagrammed in `skill-supply-chain-pipeline.svg`.

5.10 Proposed Remedy 4: Capability Tokens

Replace coarse sandbox modes with per-skill capability tokens. Examples:

- `file:workspace:rw`
- `file:workspace:ro`
- `network:outbound:https-only`
- `network:outbound:allowlist:api.example.com`
- `exec:sandboxed:allowlist`
- `browser:headless:read-only`
- `cron:self-only`
- `sessions:list-only`

A skill declares its required tokens in `SKILL.md` frontmatter:

```
---
capabilities:
  - file:workspace:rw
  - network:outbound:allowlist:github.com
---
```

The gateway checks the token before executing a tool on behalf of that skill. If a skill asks for `exec` but only has `file:workspace:rw`, the call is blocked. This is least privilege applied to agent skills.

5.11 Proposed Remedy 5: Plugin Isolation and Quarantine

- Load each CLI/plugin extension in a separate worker context with bounded error surface.

- If a plugin fails to load, quarantine it under `~/.openclaw/extensions/.quarantine/` and disable it.
- Add `openclaw extensions list --health` showing load status and last error per extension.
- Do not allow a broken extension to block core CLI commands.

5.12 Proposed Remedy 6: Security Audit Enhancements

Extend `openclaw security audit` to:

- Check skill signatures and digests.
- List installed skills with their declared capabilities.
- Detect skills that request capabilities not used in static analysis.
- Report extensions with load failures.
- Verify ClawHub TLS certificate validity.

---

6. Deficit Category IV: Scale, High Availability, and State Durability

6.1 The Problem

OpenClaw's gateway is a single-node, single-process, SQLite-backed system. This is a feature for personal use and a ceiling for everything else. As deployments grow — more agents, more cron jobs, more channels, longer sessions — the gateway becomes a bottleneck and a single point of failure.

6.2 Evidence from Architecture Documentation

The architecture docs state:

- "A single long-lived Gateway owns all messaging surfaces."
- "One Gateway per host; it is the only place that opens a WhatsApp session."
- "Exactly one Gateway controls a single Baileys session per host."

There is no documented clustering mode, no leader election, and no replicated state beyond what the operator manually backs up.

6.3 Evidence from Cron Documentation

The cron docs describe a scheduler that runs inside the Gateway process and persists state in SQLite. On startup, overdue jobs are rescheduled. Task reconciliation is "runtime-owned first, durable-history-backed second." This means a running cron task is tracked by the in-memory scheduler; only after the scheduler forgets it does maintenance consult the SQLite log.

This design is correct for a single process. It is fragile for a system that restarts for updates or that needs to survive process crashes.

6.4 Operational Evidence from DGX Spark

- SQLite state file is 59 MB plus WAL and shared-memory files.
- 179 cron jobs in the database, many disabled or duplicated.
- Heartbeat cron with 137 consecutive model-allowlist errors remains enabled.
- Agent directories exist on disk without matching `agents.list` entries.
- `openclaw doctor` recommends tightening `~/.openclaw` permissions.

These observations paint a picture of state management under pressure. The system has grown by accretion rather than by design.

6.5 Root Cause Analysis

1. **Single point of failure.** The gateway process owns all channels, cron, and sessions. If it dies, everything stops.
2. **SQLite is not a distributed store.** It cannot be replicated across hosts without external tooling, and write contention limits concurrency.
3. **No data lifecycle policy.** Cron jobs, sessions, and directories accumulate without automated pruning.

6.6 Comparative Context

Other platforms provide scaling paths:

- **n8n:** Supports cluster mode with Redis pub/sub for workflow state coordination.
- **Dify:** Designed as Kubernetes-native microservices with independent scaling of API, worker, and LLM nodes.
- **Mastra:** Single-node by default but stateless by design, making it suitable for container orchestration.

- **LangGraph:** Checkpointed state can be stored in any database; the graph executor can be distributed.

OpenClaw has no comparable scaling story.

6.7 Proposed Remedy 1: Optional Gateway Clustering

Introduce an optional clustered mode. A minimal viable architecture:

- **Reverse proxy** / **load balancer** routes clients to healthy gateway nodes. Tailscale, Caddy, or Envoy are suitable.
- **Gateway nodes** are stateless-ish. They own active WebSocket sessions but do not own durable state.
- **Replicated job queue** stores cron jobs and tasks. rqlite, DQLite, or etcd are good candidates.
- **Replicated session store** stores transcripts, routing state, and pairing records.
- **Object/blob store** stores media, snapshots, and workspace backups.

This is diagrammed in `proposed-resilient-architecture.svg`.

6.8 Proposed Remedy 2: Replicated Job Queue

Replace the SQLite cron store with a replicated job queue in clustered mode. Requirements:

- Durable job definitions.
- Exactly-once execution semantics.
- Leader election so only one node schedules and executes cron at a time.
- Run history with structured logs.
- Retry with exponential backoff and dead-letter queue.

rqlite is a strong candidate because it provides SQLite semantics with Raft replication, which minimizes the migration cost from the existing SQLite schema.

6.9 Proposed Remedy 3: Replicated Session Store

Move durable session state (transcripts, routing state, pairing records) to a replicated store. Active WebSocket connections remain node-local, but if a node fails, clients can reconnect to another node and resume their session from the replicated store.

This requires:

- A session resumption protocol on reconnect.
- Periodic checkpointing of transcript state.

- Conflict resolution for concurrent writes (last-write-wins with vector clocks is a reasonable default).

6.10 Proposed Remedy 4: State Lifecycle Management

Implement automated reconciliation:

- **Cron reconciliation:** detect duplicates by `(name, agentId, schedule)` fingerprint; disable or merge them; remove jobs disabled for more than a configurable age.
- **Agent directory reconciliation:** compare disk directories with `agents.list`; report orphaned directories.
- **Session retention:** compact or archive transcripts older than a configurable age; keep metadata for audit.
- **Model registry cleanup:** validate `models.json` and quarantine invalid entries.

These policies should be default-on but configurable.

6.11 Proposed Remedy 5: Object Store Integration

Integrate an S3-compatible object store (MinIO, AWS S3, Cloudflare R2, etc.) for:

- Media files (images, audio, video) so they do not bloat the SQLite database.
- Workspace snapshots for backup and migration.
- Skill bundle archives.
- Run artifact storage (logs, screenshots, downloaded files).

Configuration example:

```
{
  storage: {
    backend: "s3",
    s3: {
      endpoint: "https://s3.example.com",
      bucket: "openclaw-state",
      prefix: "spark-56bc"
    }
  }
}
```

6.12 Proposed Remedy 6: Observability Metrics

Expose Prometheus/OpenTelemetry metrics:

- `openclaw_gateway_active_sessions`
- `openclaw_gateway_connected_channels`
- `openclaw_cron_job_last_success_timestamp`
- `openclaw_cron_job_consecutive_errors`
- `openclaw_channel_health_status`
- `openclaw_memory_index_lag_seconds`
- `openclaw_model_request_latency_seconds`
- `openclaw_tool_calls_total`

These metrics make OpenClaw observable in standard infrastructure stacks.

---

7. Deficit Category V: Agent Loop Architecture

7.1 The Problem

OpenClaw's agent loop is powerful but opaque. A single user message triggers a multi-step pipeline that may call multiple tools, spawn subagents, stream replies, and persist state. The loop is event-driven and streaming-oriented. It is optimized for responsive conversation, not for reproducible, verifiable, multi-step workflows.

7.2 Evidence from Agent Loop Documentation

The official docs describe the loop as:

- `agent` RPC validates params and returns `runId` immediately.
- `agentCommand` resolves model, loads skills, and runs the embedded agent.
- `runEmbeddedAgent` serializes runs, resolves model/auth, enforces timeout, and streams events.
- Tool events, assistant deltas, and lifecycle events are emitted on streams.
- Final payloads are assembled from assistant text, tool summaries, and error text.

There are hooks, but no explicit plan representation. There is a global queue and per-session serialization, but no shared task queue for agent-to-agent work. There is streaming, but no structured verification step.

7.3 Comparative Context

Other frameworks have explicit orchestration primitives:

- **LangGraph:** Graph nodes and edges with checkpointed state. The graph is inspectable, resumable, and testable.
- **CrewAI:** `Crew`, `Agent`, `Task`, `Process`, and `CrewMemory`. Tasks are explicit and assigned.
- **Mastra:** `Agent`, `Tool`, `Workflow`, and `Memory`.
- **AutoGPT:** `Agent` primitives with plugin message bus and `AlphaMemory`.

OpenClaw has sessions and memory search, but it lacks an explicit task abstraction, an orchestration graph, and structured verification.

7.4 Root Cause Analysis

1. **The loop is conversational, not transactional.** Each message triggers a run. There is no durable plan that survives the run or that can be replayed.
2. **Tool calls are side effects, not structured outputs.** There are no required pre/post conditions, idempotency declarations, or result schemas.
3. **Multi-agent interaction is implicit.** Agents communicate through shared context, not a defined protocol.

7.5 Proposed Remedy 1: Optional Structured Loop

Introduce an optional structured agent loop with five phases:

1. **Intake:** parse, validate, and classify intent.
2. **Plan:** produce a typed plan with tool selections, preconditions, expected outputs, and rollback steps.
3. **Execute:** run sandboxed tool calls with timeout, retry, and capability-token checks.
4. **Verify:** validate outputs against schemas and semantic consistency checks.
5. **Commit:** atomically write the transcript and deliver the reply.

This is diagrammed in `agent-loop-reliability.svg`.

The structured loop is opt-in via `loopMode: "structured"` in agent config. The default streaming loop remains unchanged.

7.6 Proposed Remedy 2: Shared Task Queue

Add a task queue abstraction for agent-to-agent work:

```
{
  tasks: {
    backend: "rqlite",
    queue: "default",
    defaultTimeoutSeconds: 300,
    defaultRetries: 2
  }
}
```

Agents can enqueue tasks for each other via a `task_enqueue` tool. A task has:

- `id`
- `targetAgent`
- `payload`
- `expectedOutputSchema`
- `timeoutSeconds`
- `maxRetries`
- `status`

This enables explicit multi-agent orchestration without relying on shared session context.

7.7 Proposed Remedy 3: Conflict Resolution

When multiple agents can modify shared state, define conflict-resolution rules:

- Last-write-wins with vector-clock timestamps.
- Append-only merge functions for logs.
- Human escalation for conflicting edits to critical files.

Store rules in `AGENTS.md` or a dedicated `policies.md`.

7.8 Proposed Remedy 4: Reproducibility and Replay

Add a `runs` API:

- `openclaw runs list --agent <id>`
- `openclaw runs replay <runId>` — re-execute with writes disabled by default.
- `openclaw runs diff <runId1> <runId2>`

Store per-run metadata: prompts, tools called, outputs, model, provider, and memory retrievals. This makes the loop debuggable and testable.

---

8. Implementation Roadmap

Phase 1: Stability Patches (0–3 months)

1. Fix ClawHub TLS and bot protection (#96820).
2. Merge and validate PR #97059 (memory provider routing fix).
3. Implement cron failure escalation policy.
4. Add channel health reconciliation loop and dashboard indicators.
5. Strictly validate agent state files; quarantine invalid entries.
6. Isolate extension load failures; add `extensions list --health`.
7. Surface session-state cleanup issues like #97042 with explicit lifecycle events.

Phase 2: Security and Governance (3–6 months)

1. Signed skill bundles and `clawhub verify`.
2. Skill supply-chain pipeline: static analysis + sandboxed test + policy gate.
3. Capability tokens for skill/tool execution.
4. Memory provider audit trail and explicit memory contracts.
5. State lifecycle management: cron deduplication, agent directory reconciliation, session retention.
6. Enhanced `openclaw security audit`.

Phase 3: Scale and Architecture (6–12 months)

1. Replicated job queue and session store (rqlite/DQLite/etcd).
2. Optional gateway clustering with leader-elected cron.
3. Object store integration for media and backups.
4. OpenTelemetry metrics and structured run logs.
5. Optional structured plan-and-verify agent loop with task queue.
6. Runs API for reproducibility and replay.

---

9. Risk and Trade-Off Discussion

9.1 Preserving Hackability

OpenClaw's culture is terminal-first and hackable-by-default. Many of the proposals add structure. To preserve culture:

- Structured modes are opt-in.
- The single-node, SQLite-backed, host-exec default remains.
- New APIs are additive.
- Skills and `AGENTS.md` formats continue to work.

9.2 Avoiding Scope Creep

OpenClaw's `VISION.md` explicitly rejects "agent-hierarchy frameworks" and "heavy orchestration layers." The task queue and structured loop proposed here should be thin layers, not replacements. The goal is to make existing tools more reliable, not to build a new orchestration framework.

9.3 Backwards Compatibility

Clustering and replicated state are opt-in profiles. Single-node users see no change. Stricter state validation may surface existing bad config, but `openclaw doctor --fix` can repair it.

9.4 Performance

Reconciliation loops, health checks, and replicated stores add overhead. For personal deployments, these should be lazily initialized and low-frequency. For clustered deployments, the overhead is the cost of HA.

9.5 Security vs. Convenience

Capability tokens, signed bundles, and sandboxed skill tests add friction to skill installation. This is appropriate. The current one-line install model is convenient but unsafe for a growing marketplace. A "developer mode" can bypass gates for trusted local skills.

---

10. Conclusion

OpenClaw is at an inflection point. Its popularity and breadth are no longer in doubt. What is in doubt is whether its current architecture can support the next wave of use cases: always-on autonomous agents, small-team deployments, and safety-critical tool use.

The deficits identified in this paper share a common root: OpenClaw is a personal assistant being asked to behave like infrastructure. Resolving that tension requires three commitments:

1. **Observability first:** every silent failure must become a visible, actionable signal.
2. **Verified trust:** skills, plugins, memory providers, and tool invocations must be auditable and bounded by least privilege.
3. **Optional durability:** high availability, replicated state, and structured agent loops must be available without forcing them on personal users.

If OpenClaw makes these commitments while preserving its local-first, hackable spirit, it can become not just the most popular open-source agent gateway, but the most trustworthy one.

---

11. Bibliography

#	Source	Type	Key Content	URL
1	OpenClaw GitHub repository	Repository	Project README, install instructions, feature list, channel support	https://github.com/openclaw/openclaw
2	OpenClaw VISION.md	Repository doc	Current priorities, roadmap guardrails, plugin philosophy, security posture	https://github.com/openclaw/openclaw/blob/main/VISION.md
3	OpenClaw release 2026.6.10-beta.1	Release notes	Reliability, Codex, channel delivery, security, CLI, mobile, plugin updates	https://github.com/openclaw/openclaw/releases/tag/v2026.6.10-beta.1
4	OpenClaw release 2026.6.11	Release notes	Slack relay, Mattermost queue, per-DM model overrides, RAFT wake bridge	https://github.com/openclaw/openclaw/releases
5	OpenClaw architecture docs	Documentation	Single gateway, WebSocket RPC, pairing, canvas, nodes	https://docs.openclaw.ai/concepts/architecture
6	OpenClaw multi-agent routing docs	Documentation		https://docs.openclaw.ai/concepts/multi-agent

#	Source	Type	Key Content	URL
			Isolated agents, bindings, workspace scoping, cross-agent memory search	
7	OpenClaw agent loop docs	Documentation	Loop lifecycle, queueing, hooks, streaming, timeouts	https://docs.openclaw.ai/concepts/agent-loop
8	OpenClaw cron docs	Documentation	SQLite-backed scheduler, job kinds, execution styles, reconciliation	https://docs.openclaw.ai/automation/cron-jobs
9	OpenClaw memory/ QMD docs	Documentation	QMD sidecar, hybrid search, index paths, fallback behavior	https://docs.openclaw.ai/concepts/memory-qmd
10	OpenClaw security docs	Documentation	Personal-assistant trust model, sandboxing, exposure runbook, audit	https://docs.openclaw.ai/gateway/security
11	OpenClaw sandboxing docs	Documentation	Docker/SSH/OpenShell backends, scope, DooD constraints	https://docs.openclaw.ai/gateway/sandboxing
12	OpenClaw ClawHub docs	Documentation	Skill/plugin registry, install flows, security scans, moderation	https://docs.openclaw.ai/clawhub
13	OpenClaw command queue docs	Documentation	Queue modes, steering, followup, collect, interrupt	https://docs.openclaw.ai/concepts/queue
14	Issue #97060 — WhatsApp silent failure	GitHub Issue	Transport connects, application registration fails silently	https://github.com/openclaw/openclaw/issues/97060
15	Issue #97042 — Telegram typing stuck	GitHub Issue	P1 session-state bug; multi-agent collective degraded; closed not_planned	https://github.com/openclaw/openclaw/issues/97042
16	Issue #96820 — ClawHub 403 + expired SSL	GitHub Issue	Skill marketplace broken; SSL expired; MITM risk	https://github.com/openclaw/openclaw/issues/96820
17	Issue #97055 — Memory provider config ignored	GitHub Issue	Custom Ollama endpoint silently bypassed for embeddings	https://github.com/openclaw/openclaw/issues/97055
18	PR #97059 — Ollama memory endpoint fix	GitHub PR (draft)	Fixes gateway adapter path; preserves provider identity	https://github.com/openclaw/openclaw/pull/97059
19		GitHub Issue		

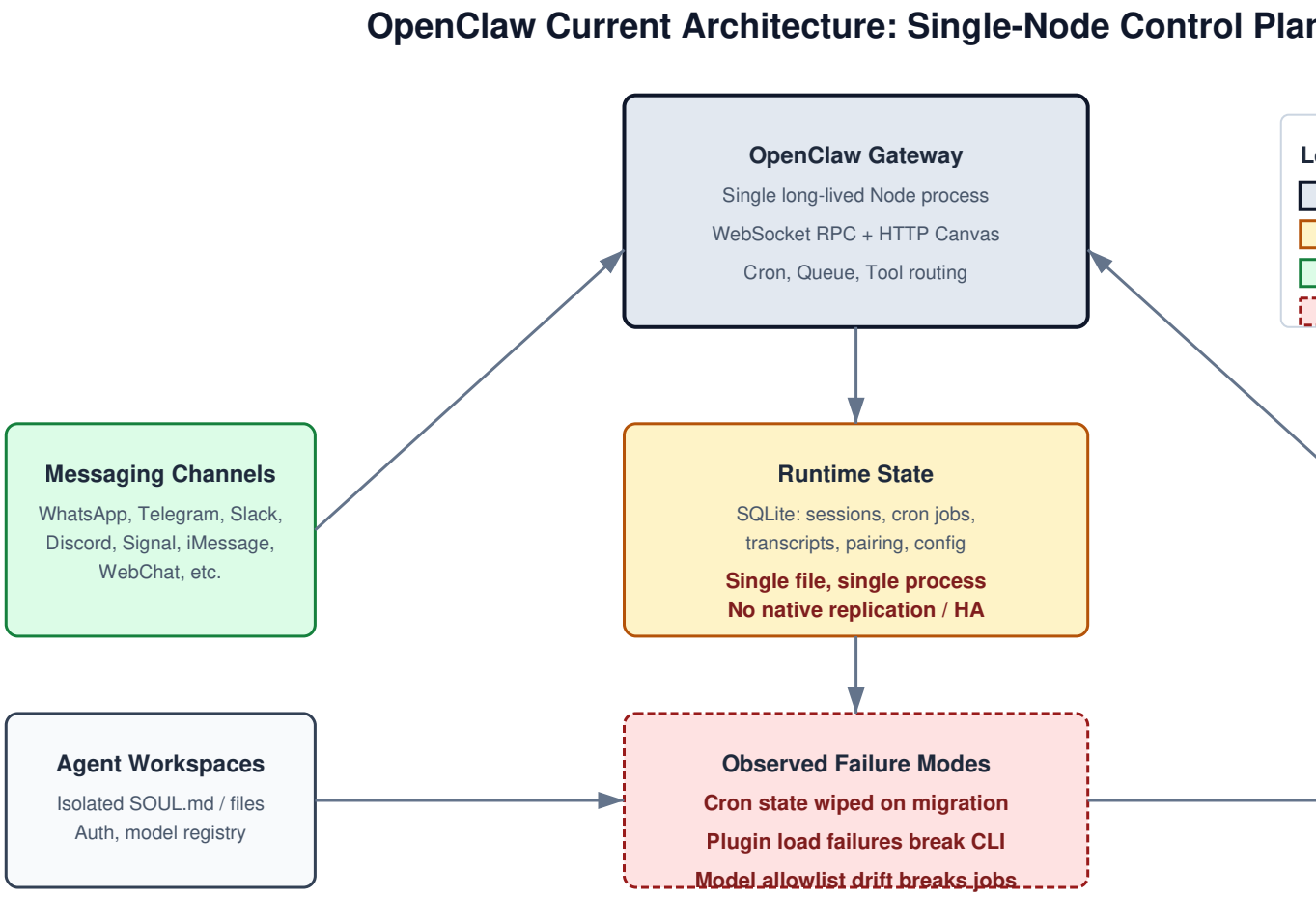
#	Source	Type	Key Content	URL
	Issue #96254 — Internal tool toolcall info		API/chat/completions does not return tool-call behavior	https://github.com/openclaw/openclaw/issues/96254
20	OpenClaw label taxonomy	GitHub metadata	ClawSweeper labels: security-review, merge-risk, impact, etc.	https://github.com/openclaw/openclaw/labels
21	LangGraph documentation	External docs	Pregel graph-based orchestration; checkpointed state; typed memory	https://langchain.com/docs/langgraph
22	CrewAI documentation	External docs	Crew + Agent + Task + Process; shared CrewMemory	https://docs.crewai.com
23	Mastra documentation	External docs	Agent + Tool + Workflow; explicit Memory; stateless design	https://mastra.ai/docs
24	n8n documentation	External docs	Cluster mode with Redis pub/sub; community node permissions	https://docs.n8n.io
25	Dify documentation	External docs	Kubernetes-native microservices; independent scaling	https://dify.ai/docs
26	AutoGPT documentation	External docs	AlphaMemory importance-weighted retention; plugin message bus	https://github.com/Significant-Gravitas/AutoGPT
27	rqlite project	External project	SQLite with Raft replication; lightweight distributed SQL	https://rqlite.io
28	DQLite project	External project	Embeddable Raft-based SQLite for applications	https://dqlite.io
29	gVisor project	External project	Userspace kernel sandbox for container isolation	https://gvisor.dev
30	Firecracker project	External project	MicroVMs for secure multi-tenant workloads	https://firecracker-microvm.github.io
31	Kubernetes probes documentation	External docs	Liveness, readiness, and startup probes	https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes
32	Prometheus exposition format	External spec	Metrics exposition for monitoring systems	https://prometheus.io/docs/instrumenting/exposition_formats/

#	Source	Type	Key Content	URL
33	OpenTelemetry project	External spec	Observability framework for traces, metrics, logs	https://opentelemetry.io
34	SLSA framework	External spec	Supply-chain levels for software artifacts	https://slsa.dev
35	Let's Encrypt	External project	Free automated TLS certificates	https://letsencrypt.org

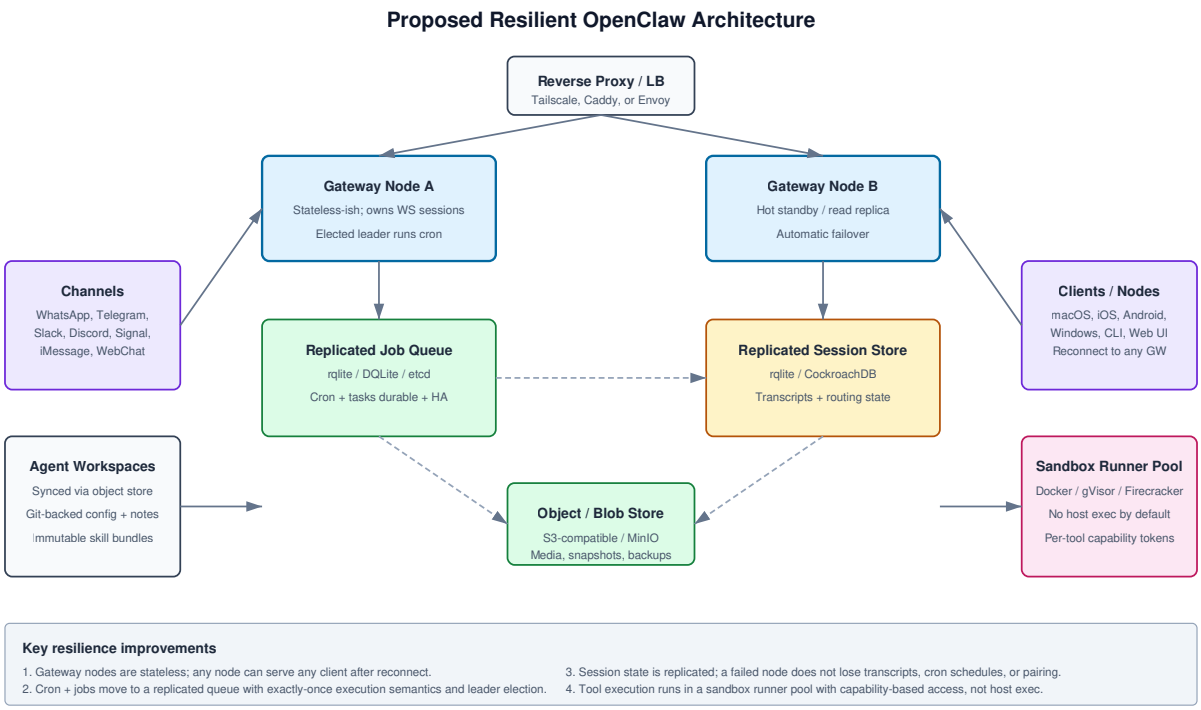
---

Appendix A: Included Diagrams

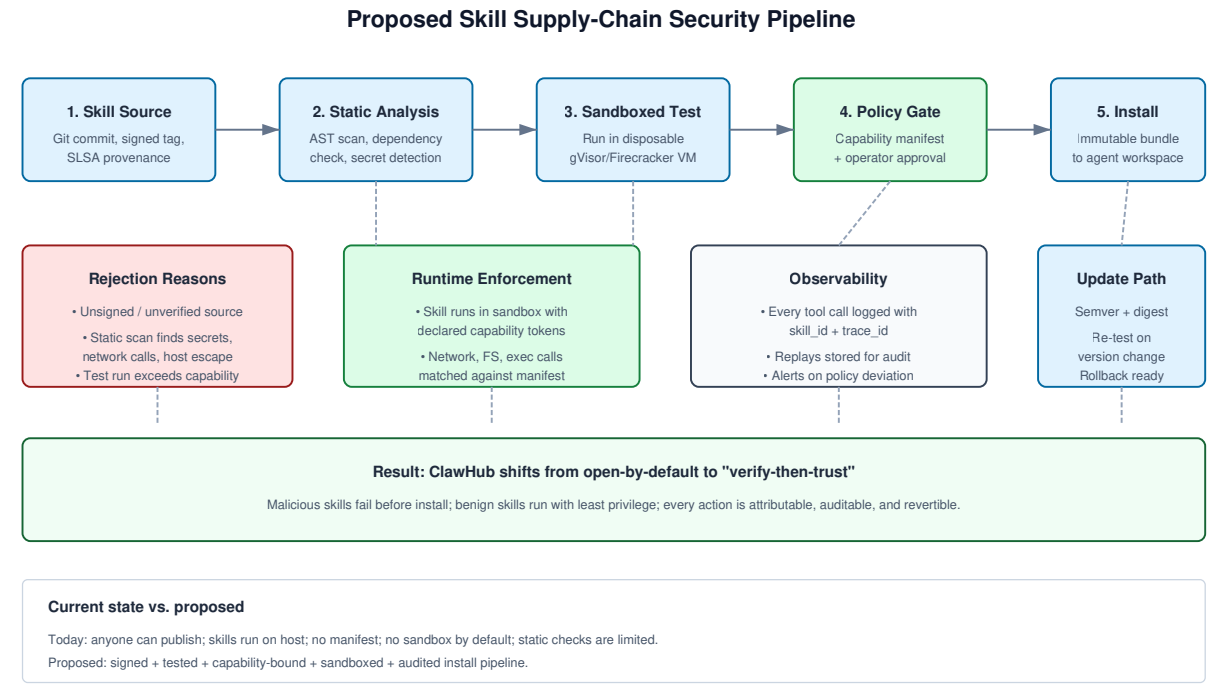
1. `



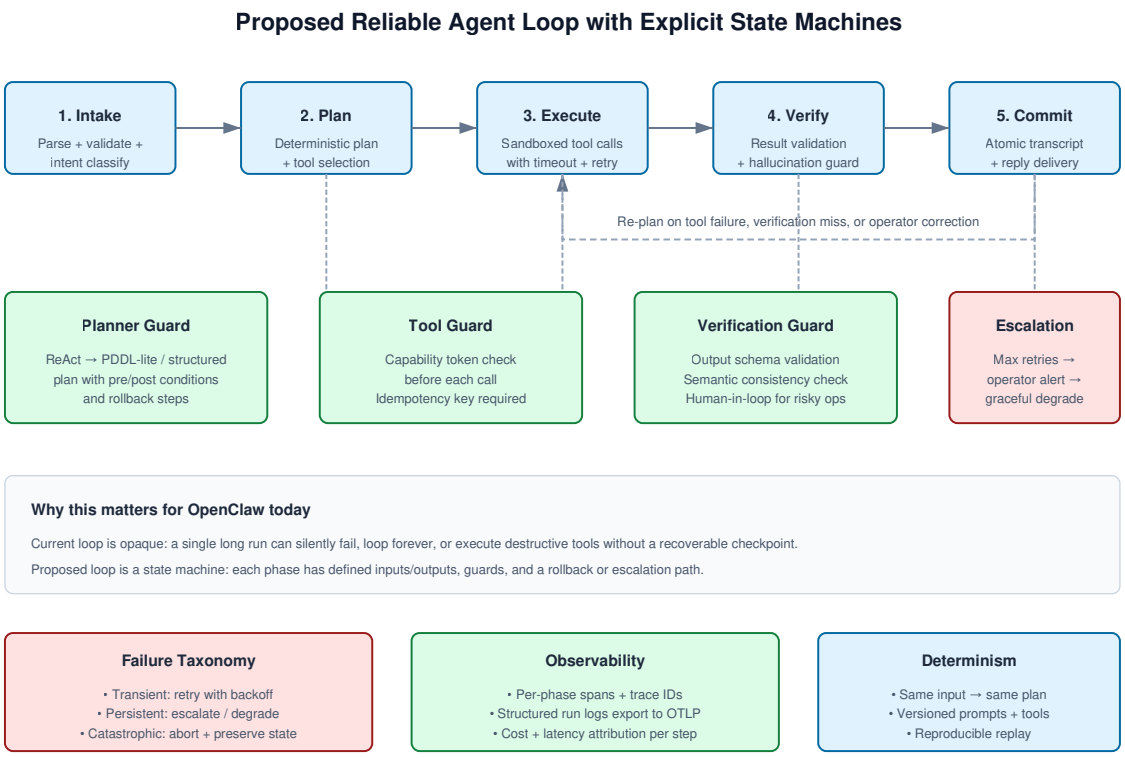
` — Current single-node gateway architecture and failure modes. 2. `



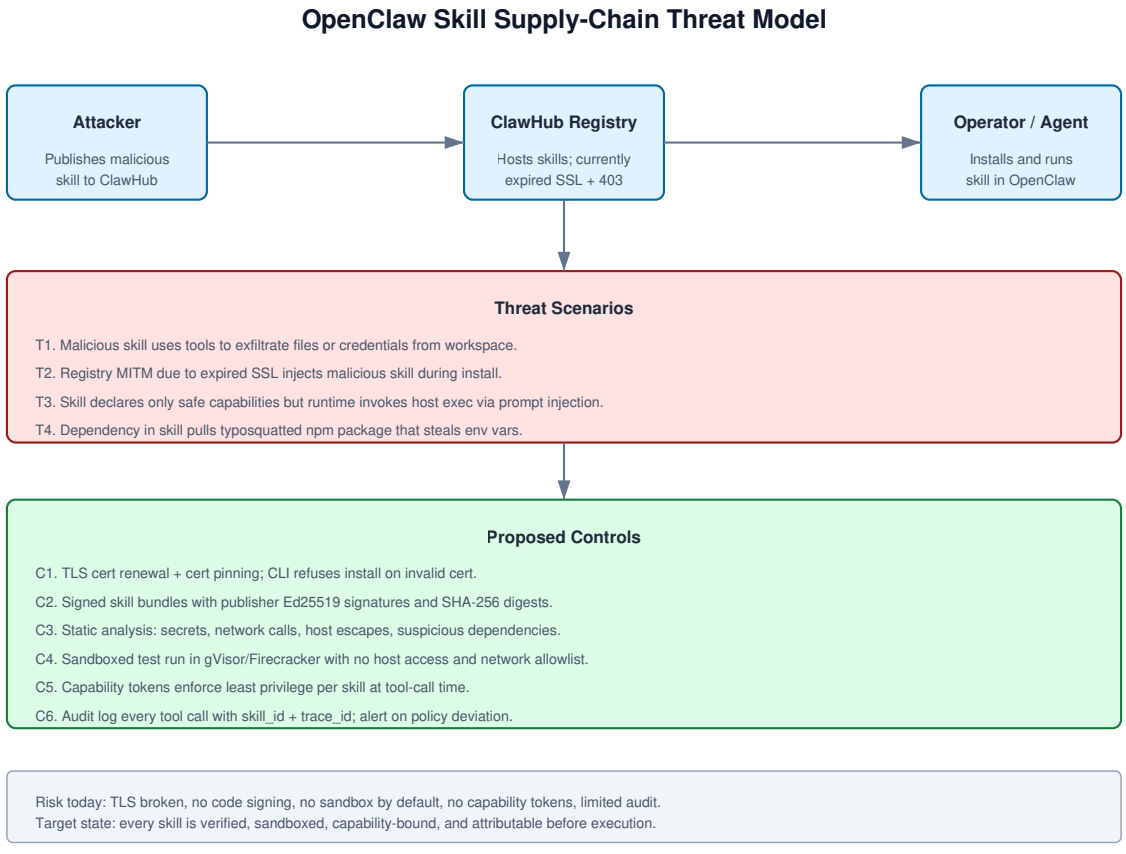
` — Proposed clustered gateway with replicated state. 3. `



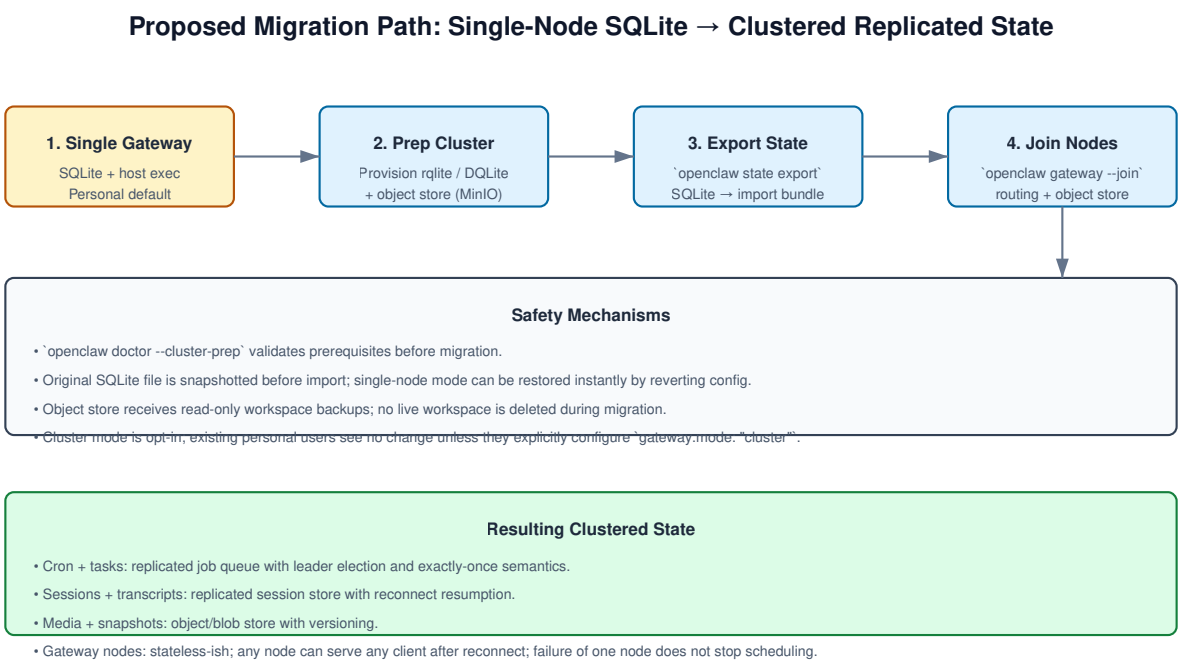
` — Verify-then-trust skill supply-chain pipeline. 4. `



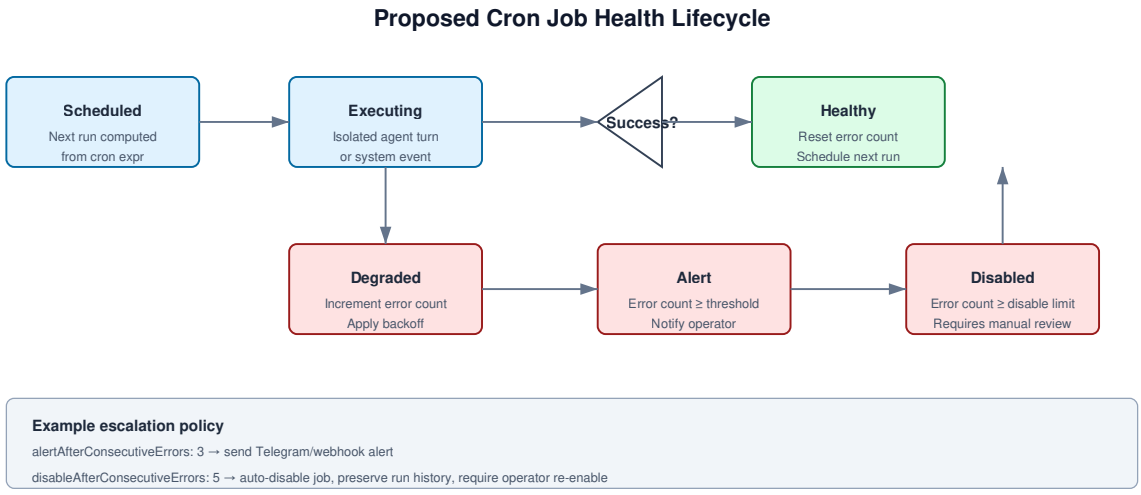
` — Proposed structured plan-and-verify agent loop. 5. `



` — Skill supply-chain threat model and proposed controls. 6. `



` — Migration path from single-node SQLite to clustered replicated state. 7. `



` — Proposed cron job health lifecycle with escalation policy.

Appendix B: Operational Data Notes

All operational observations from DGX Spark were gathered on 2026-06-26 using `openclaw --version`, `openclaw doctor`, `openclaw cron list`, and direct inspection of `~/.openclaw/state/openclaw.sqlite`. SQLite queries were executed with `sqlite3` against the live state file. Agent directory and extension state were inspected with standard shell commands.

Appendix C: Detailed Comparative Gap Matrix

Dimension	OpenClaw (2026.6.10)	LangGraph	CrewAI	Mastra	n8n	Dify
Silent failure detection	No channel health API; cron errors are row-level	Graph state makes failures visible	Task status visible	Explicit state	Error paths per node	Execution log
Memory provider isolation	Config ignored in some adapter paths (#97055)	Typed schemas	Configurable	TTL/ eviction	Vector stores	Multiple backends
Memory governance hooks	None	Checkpointing	CrewMemory	Memory hooks	Audit logs	Audit logs
Skill marketplace SSL	Expired cert + 403 (#96820)	N/A	N/A	N/A	Community nodes	Reviewed
Skill sandboxing	Opt-in Docker/SSH/ OpenShell	N/A	N/A	N/A	Separate context	Scanning
Skill capability tokens	None	N/A	N/A	N/A	Permissions	Permissions
Multi-agent orchestration	Session-based	Graph-based	Crew model	Workflows	Sub-workflows	Agent function calling
Inter-agent task queue	None	Graph edges	Task queue	Workflows	Webhooks	Tools
Cron/task durability	SQLite + in-process scheduler	Checkpointer	Task queue	Basic	Execution history	Trigger logs
Gateway clustering	Single-node	N/A	N/A	N/A	Redis pub/sub	Kubernetes-native
Tool call transparency	Missing in some API	Partial	Partial	Explicit		

Dimension	OpenClaw (2026.6.10)	LangGraph	CrewAI	Mastra	n8n	Dify
	paths (#96254)				Node execution log	Execution trace
Reproducible replay	None	Checkpoint replay	Limited	Limited	Workflow replay	Versioned runs
OpenTelemetry metrics	Partial (trace correlation)	Ecosystem	Ecosystem	Ecosystem	Ecosystem	Ecosystem

12. Detailed Implementation Runbooks

This section provides concrete implementation guidance for the highest-impact proposals. It is written for OpenClaw maintainers and advanced operators who want to move from diagnosis to code.

12.1 Channel Health Reconciliation Loop — Runbook

Goal: Detect and surface channel-level silent failures such as the WhatsApp mute bug (#97060).

Data model:

```
interface ChannelHealthRecord {
  accountId: string;          // e.g. "whatsapp:work"
  transport: 'healthy' | 'degraded' | 'unhealthy' | 'unknown';
  application: 'healthy' | 'degraded' | 'unhealthy' | 'unknown';
  functional: 'healthy' | 'degraded' | 'unhealthy' | 'unknown';
  lastInboundAt?: Date;
  lastOutboundAt?: Date;
  degradedSince?: Date;
  suggestedAction?: string;
  recoveryAttempts: number;
}
```

Reconciliation interval: Default 60 seconds; configurable via `channels.health.reconcileEveryMs`.

Transport probe: Ask each channel provider whether its underlying socket/session is open. For WhatsApp/Baileys, this is the `ws` ready state. For Telegram, this is the grammY client status. For Slack/Discord, this is the RTM/WebSocket status.

Application probe: Query the upstream service's authentication/registration state. For WhatsApp, read `creds.registered` from Baileys state. For Telegram, verify the bot identity with `getMe`. For Slack, call `auth.test`. If the provider does not expose registration state, use a synthetic message probe (see below).

Functional probe: Track `messagesHandled` and `lastInboundAt` per account. If transport and application are healthy but no message has been handled in the reconciliation window (default 5 minutes for active accounts, 30 minutes for idle), mark functional as `degraded`.

Synthetic probe option: For critical accounts, allow operators to configure a ping-pong contact:

```
{
  channels: {
    whatsapp: {
      accounts: {
        work: {
          healthProbe: {
            kind: "synthetic",
            to: "+15551234567",
            message: "ping",
            expectedReply: "pong",
            intervalMinutes: 10
          }
        }
      }
    }
  }
}
```

Recovery actions:

1. `reconnect`: close and reopen transport.
2. `re-register`: for WhatsApp, refresh Baileys credentials; for Telegram, re-verify bot token.
3. `notify`: send operator alert after recovery attempts exceed threshold.

RPC surface:

- `channels.health.list` returns all health records.
- `channels.health.get <accountId>` returns a single record.
- `channels.health.probe <accountId>` triggers an on-demand reconciliation.

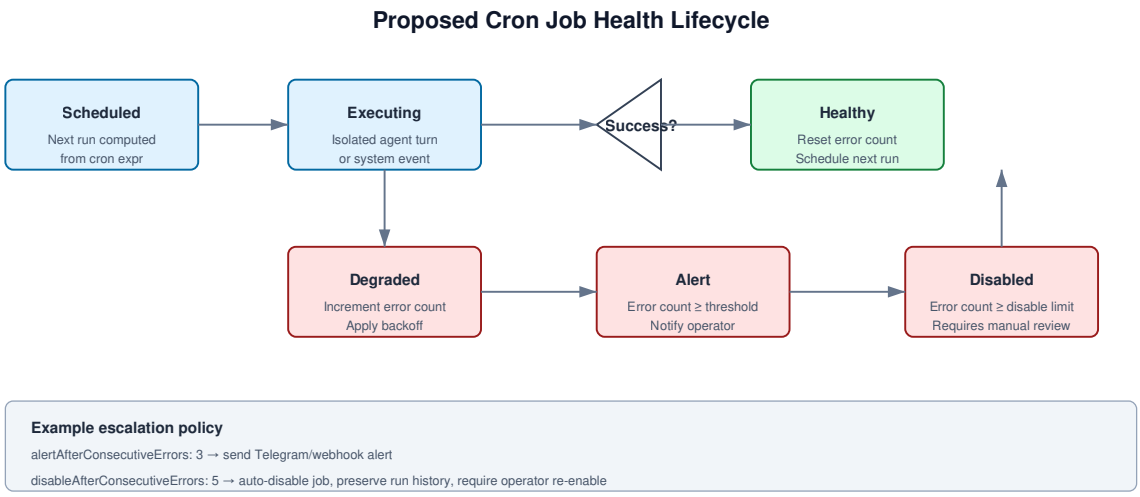
Control UI: Render each account as a traffic-light card with last functional activity and a "Run probe" button.

Metrics: `openclaw_channel_health_status` gauge labeled by `account_id`, `channel`, and `dimension` (`transport`, `application`, `functional`).

12.2 Cron Failure Escalation Policy — Runbook

Goal: Stop unactioned cron failures such as the 137-error heartbeat job observed on DGX Spark.

State transitions: See `



`.

Config schema:

```
{
  cron: {
    escalation: {
      alertAfterConsecutiveErrors: 3,
      disableAfterConsecutiveErrors: 5,
      alertChannel: "telegram:ops",
      alertWebhook: "https://hooks.example.com/openclaw",
      backoffPolicy: "exponential",
      backoffMaxMs: 3600000,
      alertTemplate: "Cron '{jobName}' for agent {agentId} has failed {count} times.
Last error: {lastError}"
    }
  }
}
```

Per-job override:

```
openclaw cron edit <id> --escalation
alertAfterConsecutiveErrors=1,disableAfterConsecutiveErrors=3
```

Backoff behavior: When a job fails, schedule the next retry with exponential backoff rather than the normal cron cadence. For example, a job scheduled every 30 minutes that fails gets retried at 1

minute, 2 minutes, 4 minutes, then back to 30 minutes if it succeeds. If it reaches `disableAfterConsecutiveErrors`, it is disabled.

Alert payload:

```
{
  "event": "cron.escalation",
  "jobId": "...",
  "jobName": "...",
  "agentId": "...",
  "consecutiveErrors": 5,
  "action": "alert|disabled",
  "lastError": "...",
  "lastSuccessAt": "..."
}
```

Audit commands:

- `openclaw cron audit` lists jobs sorted by consecutive errors, last success, and schedule drift.
- `openclaw cron health` shows counts of healthy/degraded/disabled jobs.

Metrics:

- `openclaw_cron_job_consecutive_errors` gauge.
- `openclaw_cron_job_last_success_timestamp` counter.
- `openclaw_cron_escalations_total` counter labeled by `action`.

12.3 Memory Provider Audit and Contracts — Runbook

Goal: Prevent custom memory providers from being silently ignored (#97055).

Provider identity tunneling:

Every embedding request must carry:

- `providerId`: the resolved provider id from config.
- `baseUrl`: the provider's endpoint.
- `modelId`: the embedding model.

These are passed through every adapter layer:

```
interface EmbeddingRequestContext {
  providerId: string;
  baseUrl: string;
  modelId: string;
  agentId: string;
```

```
  collection: string;
}
```

The memory adapter logs these fields at `info` when `diagnostics.memoryProviderAudit` is enabled.

Audit command:

```
openclaw memory provider-audit --limit 50
```

Output:

```
2026-06-26T12:00:00Z  agent=gabriel  provider=ollama-local  baseUrl=http://
127.0.0.1:11434  model=nomic-embed-text  collection=memory-root-gabriel
latency=120ms  status=ok
```

Memory contract schema:

```
interface MemoryContract {
  backend: 'builtin' | 'qmd';
  provider?: string;
  embeddingModel?: string;
  indexPaths: Array<{ name: string; path: string; pattern?: string }>;
  retentionPolicy: { kind: 'time'; maxAgeDays: number } | { kind: 'count'; maxItems:
number };
  citations: 'off' | 'auto' | 'on';
  redaction: {
    secrets: boolean;
    apiKeys: boolean;
  };
}
```

The contract is stored in `openclaw.json` and exposed via `memory status --contract`. A summary is injected into the agent's system prompt.

Governance hook:

```
interface MemoryGovernanceHook {
  beforeWrite(record: MemoryRecord): MemoryRecord | null;
}
```

Returning `null` blocks the write. Returning a modified record applies redaction or tags.

State validation:

On load, validate `models.json` against a schema. Invalid entries are moved to `~/.openclaw/agents/<id>/agent/.quarantine/models.json-<timestamp>` and reported by `openclaw doctor`.

12.4 Skill Supply-Chain Pipeline — Runbook

Goal: Transform ClawHub from open-by-default to verify-then-trust.

Stage 1: Source verification.

Accept only:

- GitHub repositories with signed tags (`git tag -s`).
- Skills published by verified ClawHub orgs.
- Local paths for development mode.

Stage 2: Static analysis.

Use a lightweight parser (e.g., `acorn` for JS, Markdown AST for SKILL.md) to detect:

- Hardcoded secrets (regex for `sk-`, `ghp_`, etc.).
- Network calls (`fetch`, `axios`, `https.request`, `curl`).
- File system access outside workspace (`fs.readFile` with absolute paths, `/etc`, `~/.ssh`).
- Shell execution (`exec`, `spawn`, `child_process`).
- Dependency declarations and npm package names.

Output a risk report:

```
{
  "skill": "@openclaw/demo",
  "version": "1.2.3",
  "risks": [
    { "severity": "high", "category": "network", "detail": "Calls https://
example.com" },
    { "severity": "medium", "category": "filesystem", "detail": "Reads absolute
path /tmp" }
  ]
}
```

Stage 3: Sandboxed test run.

Execute the skill in a gVisor or Firecracker VM with:

- A seeded copy of the agent workspace.
- No host filesystem access except the workspace.
- A network allowlist matching declared capabilities.

- A fake model harness that records all tool calls.

Stage 4: Policy gate.

Compare the observed tool calls against the declared capability manifest. If the skill requested a capability it did not declare, block install.

Stage 5: Immutable install.

Store the approved bundle in `~/.openclaw/agents/<id>/skills/<skill>@<version>/`. Updates require re-running the pipeline. Old versions are retained for rollback.

12.5 Capability Tokens — Runbook

Goal: Enforce least privilege for skills.

Token format:

```
<resource>:<action>:<scope>
```

Examples:

- `file:workspace:rw`
- `file:workspace:ro`
- `network:outbound:https-only`
- `network:outbound:allowlist:api.github.com`
- `exec:sandboxed:allowlist`
- `browser:headless:read-only`
- `cron:self-only`
- `sessions:list-only`

SKILL.md declaration:

```
---
capabilities:
  - file:workspace:rw
  - network:outbound:allowlist:github.com
---
```

Runtime check: Before executing a tool on behalf of a skill, the gateway resolves the required capability token and compares it to the skill's declared tokens. Missing tokens block the call and log a policy violation.

Default policy: If a skill does not declare capabilities, it runs with a minimal safe default (`file:workspace:ro`, no network, no exec) rather than inheriting the agent's full tool policy.

12.6 Gateway Clustering — Runbook

Goal: Provide optional HA and scale.

Prerequisites:

- rqlite cluster (3+ nodes recommended).
- S3-compatible object store (MinIO, AWS S3, Cloudflare R2).
- Reverse proxy or Tailscale for client routing.

Config profile:

```
{
  gateway: {
    mode: "cluster",
    cluster: {
      nodeId: "gateway-a",
      jobQueue: "rqlite://10.0.0.10:4001",
      sessionStore: "rqlite://10.0.0.10:4001",
      blobStore: "s3://openclaw-state",
      leaderElection: "rqlite"
    }
  }
}
```

Migration:

1. Run `openclaw doctor --cluster-prep` to validate prerequisites.
2. Run `openclaw state export --format rqlite` to produce an import bundle.
3. Import into rqlite.
4. Start the first gateway node with `--cluster-join` disabled (seed).
5. Start additional nodes with `--cluster-join <seed-addr>`.
6. Point clients to the reverse proxy.

Leader election for cron: Use rqlite's built-in distributed locking or a lease table. Only the leader schedules cron jobs. If the leader dies, another node acquires the lease within a configurable timeout (default 30 seconds) and resumes scheduling.

Session resumption: When a client reconnects to a different gateway node, the node reads the session transcript from the replicated session store and resumes. Active streaming state is lost on node failure, but durable transcript state survives.

Object store usage:

- Media files: store in object store; SQLite keeps only metadata and URLs.
- Workspace snapshots: nightly or on-demand backup to object store.
- Skill bundles: archived in object store with versioned keys.

13. Failure Mode and Effects Analysis (FMEA)

ID	Failure Mode	Cause	Current Effect	Severity	Likelihood	Detection	RPN*	Proposed Mitigation
F1	WhatsApp channel muted	Application registration fails but transport reports connected	Bot never receives messages; silent until user inspects state	9	6	2	108	Channel health reconciliation loop with application probe
F2	Cron job fails repeatedly unactioned	Model allowlist drift or provider outage	Background task silently stops working; error count grows	8	7	3	168	Cron escalation policy: auto-disable + alerting
F3	Memory embeddings route to wrong provider	Provider id dropped in memory adapter	Data leaves intended host; privacy/ correctness risk	9	5	2	90	Provider identity tunneling + memory provider audit
F4	ClawHub SSL expired	Certificate lifecycle failure	Skill marketplace unavailable; MITM risk	9	4	4	144	Automated TLS renewal + cert pinning + offline mirror
F5	Malicious skill exfiltrates data	No code signing or sandbox by default	Operator workspace data stolen	9	5	3	135	Signed bundles + sandboxed test + capability tokens
F6	Gateway process crashes	Single-node design	All channels, cron, sessions stop	9	4	5	180	Optional clustering with replicated state
F7	SQLite corruption	Single-file state store on unstable host	Unrecoverable loss of sessions, cron, pairing	8	3	4	96	Replicated session/job store + object-store backups

ID	Failure Mode	Cause	Current Effect	Severity	Likelihood	Detection	RPN*	Proposed Mitigation
F8	Agent directories orphaned	`agents.list` edits without disk cleanup	Unused state accumulates; security surface grows	5	6	4	120	Agent directory reconciliation in `doctor`
F9	Tool-call trace missing in API	Internal loop events not exposed	External clients cannot debug or evaluate	6	6	5	180	Expose tool-call events in `/api/chat/completions` and runs API
F10	Plugin load failure degrades CLI	Missing dependency in extension	Every CLI command prints noise; possible behavior change	5	6	6	180	Plugin isolation + quarantine + `extensions list --health`

*RPN = Severity × Likelihood × Detection. Higher values indicate higher priority.

The FMEA confirms that the most urgent risks are unactioned cron failures (F2), gateway single point of failure (F6), missing tool-call transparency (F9), and plugin degradation (F10). These align with the Phase 1 roadmap.

14. Additional Operational Case Studies from DGX Spark

14.1 The 179-Cron Sprawl

On 2026-06-26, the DGX Spark SQLite state store contained 179 cron jobs. Many of these are legacy entries from previous agent configurations and research directives. Examples include:

- `Kalshi Trading — Daily Run (10am ET)` (Gabriel, disabled)
- `Kalshi Live Trading — Daily 10am ET` (Gabriel, disabled)
- Duplicate `Memory Dreaming Promotion` rows
- `Blog Post: SMF Works Daily` (Aiona, disabled)
- `The Signal — Weekly Post` (Pamela, disabled)
- Multiple `Pamela — X Post` entries with varying schedules

This sprawl is a direct consequence of state accumulation without lifecycle management. When an agent's responsibilities change, old cron jobs are disabled but not removed. Over months, the store becomes a graveyard of obsolete schedules. The proposed cron reconciliation policy would identify

duplicates by `(name, agentId, schedule)` fingerprint and offer to archive or delete jobs disabled for more than a configurable threshold.

14.2 Model Registry Invalid Entries

The `models.json` files for Gabriel, Morgan, Pamela, and Aiona all contain an invalid entry for `vercel-ai-gateway/alibaba/wan-v2.5-t2v-preview` with an `invalid contextWindow` value. This entry was likely created when a provider catalog was imported or updated, and the validation logic at the time did not reject it. Because the entry is loaded on every gateway start, it produces a warning on every `openclaw doctor` run and potentially affects model selection behavior.

This is a small but representative example of state validation drift. The proposed strict state validation would have quarantined this entry on first load and notified the operator with the exact file and field.

14.3 Extension Load Pollution

The `openclaw-honcho` extension fails on every CLI invocation because it cannot find `@honcho-ai/sdk`. The failure is non-fatal but persistent. It illustrates that extensions are not isolated: a broken extension can produce startup noise for unrelated commands. It also shows that there is no automatic remediation path. The operator must manually remove or fix the extension.

The proposed extension quarantine system would disable the broken extension after N load failures, move it to a `.quarantine` directory, and surface it in `openclaw extensions list --health`.

15. Competitive Version Notes

To make the comparative analysis concrete, the following versions were current at the time of writing:

- **OpenClaw:** 2026.6.10 (aa69b12) and 2026.6.11 pre-release.
- **LangGraph:** 0.4.x series, part of LangChain ecosystem.
- **CrewAI:** 0.108.x series.
- **Mastra:** 0.6.x series.
- **n8n:** 1.95.x with AI workflows and community nodes.
- **Dify:** 1.4.x, Kubernetes-native.
- **AutoGPT:** 0.6.x, `AlphaMemory` and `Agent` primitives.

These versions matter because the competitive landscape is moving quickly. A feature comparison based on year-old documentation would be misleading. The analysis in this paper reflects the state of these frameworks as of mid-2026.

16. Governance and Maintenance Recommendations

Beyond the technical proposals, OpenClaw's governance can reduce the rate at which these deficits recur:

1. **Issue triage review.** The closure of #97042 as `not_planned` despite P1 and `impact:message-loss` labels suggests automated triage may be too aggressive. A weekly human review of high-impact closed issues would catch these.
2. **Security-on-call rotation.** The 2026 issue data shows multiple `clawsweeper:needs-security-review` labels. A security-on-call rotation would ensure expired certificates and bot-protection regressions are fixed within hours, not days.
3. **State regression tests.** Add CI tests that load a synthetic `~/.openclaw` state tree, run `openclaw doctor`, and assert no warnings for valid state and clear warnings for invalid state.
4. **Release notes include operational risks.** The 2026.6.10 and 2026.6.11 release notes are excellent at listing PRs. They could also include a short "Operational Considerations" section warning operators about state migration or config changes that affect cron/memory.
5. **Public status page for ClawHub.** A simple status page showing registry, SSL, and API health would have surfaced #96820 immediately to the entire community.

---

17. Final Conclusion

OpenClaw's future depends on closing the gap between its product popularity and its operational maturity. The deficits in this paper are all solvable. Most of them can be addressed without changing OpenClaw's single-node default or its hackable culture. The key is to treat observability, verified trust, and optional durability as first-class architectural concerns rather than after-the-fact patches.

The work ahead is substantial but bounded. Phase 1 stabilizes the most painful failure modes. Phase 2 hardens the trust boundary around skills and memory. Phase 3 opens the path to clustered, always-on deployments. Taken together, these changes would make OpenClaw not only the most popular open-source agent gateway, but the most reliable and trustworthy one.