OpenClaw — Architecture Analysis

Safety Wrapper Deep Dive

Version: v1.0
Date: February 26, 2026
Company: LetsBe Solutions LLC
Contact: matt@letsbe.solutions

221 North Broad Street, Suite 3A, Middletown, DE 19709

Contents

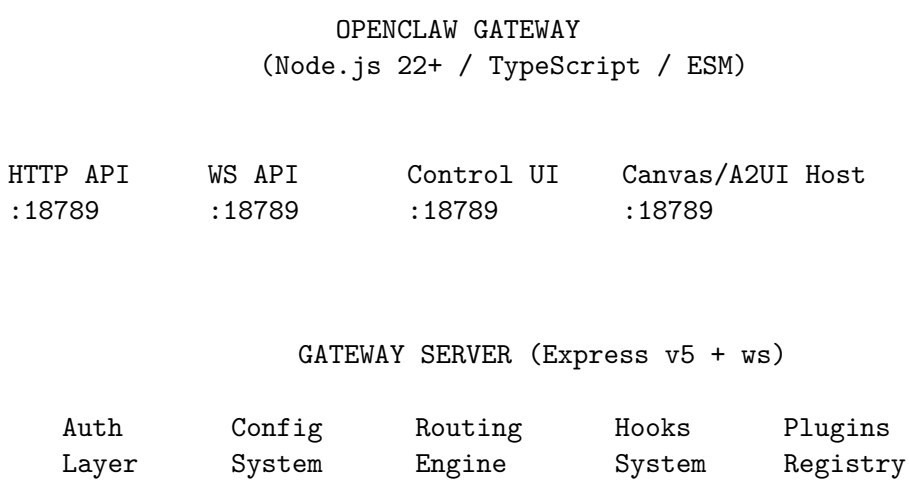
1. OpenClaw Architecture Analysis

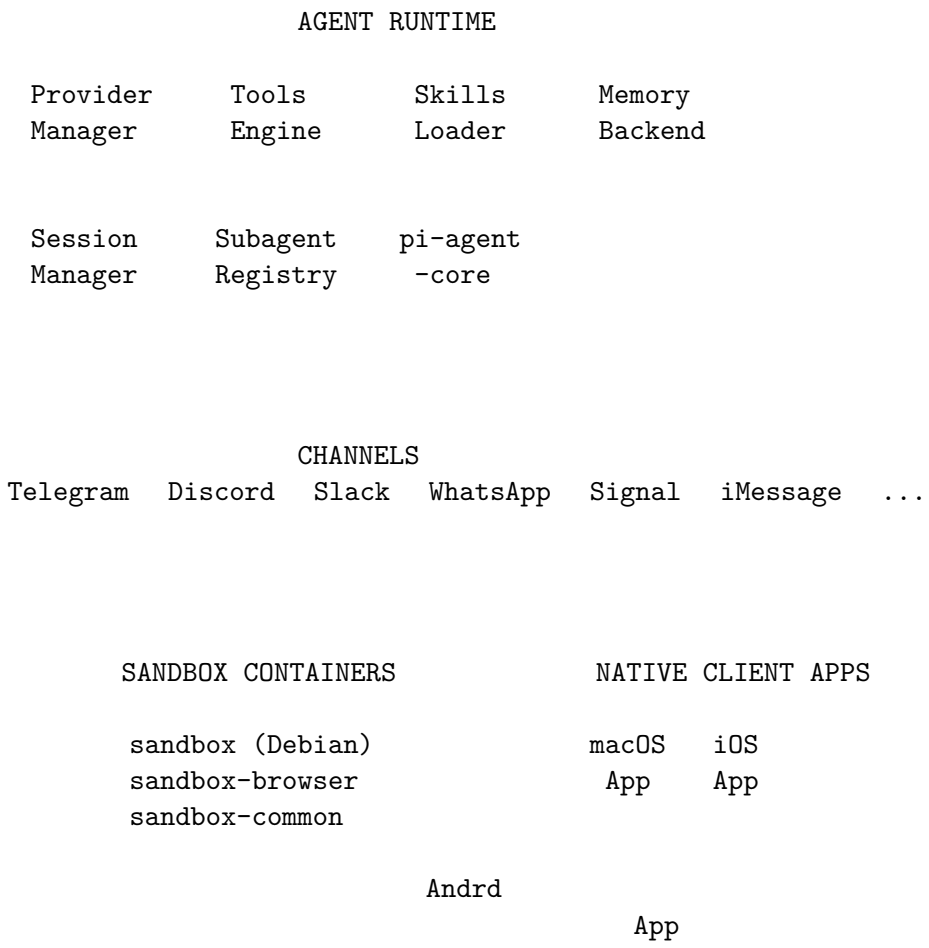
Prepared for: LetsBe Biz Team **Date:** 2026-02-26 **OpenClaw Version:** 2026.2.26 **License:** MIT (Copyright 2025 Peter Steinberger) **Purpose:** Deep architectural analysis to support provisioning, Safety Wrapper integration, tool leveraging, and technical documentation for the LetsBe privacy-first AI workforce platform.

1.1 Table of Contents

1.2 1. Architecture Overview

1.2.1 1.1 High-Level Architecture Diagram





1.2.2 1.2 Core Runtime

Attribute	Value
Language	TypeScript (ESM, strict mode)
Runtime	Node.js 22.12.0+ (required)
Package Manager	pnpm 10.23.0 (primary), Bun supported
HTTP Framework	Express v5
WebSocket	`ws` library
CLI Framework	Commander.js
Schema Validation	Zod v4 (config), TypeBox (tool schemas)
AI Agent Core	`@mariozechner/pi-agent-core`

Attribute	Value
Entry Point	`openclaw.mjs` → `dist/entry.js` → `src/cli/run-main.ts`
Binary Name	`openclaw` (installed via npm)
Default Port	18789 (gateway HTTP + WS multiplexed)
Config Format	JSON5 (`~/.openclaw/openclaw.json`)

1.2.3 1.3 Package/Module Structure

Root-level directories:

Directory	Purpose
`src/`	Core TypeScript source — CLI, gateway, agents, routing, plugins, channels
`extensions/`	Plugin packages — chat channels, memory backends, auth providers, tools
`skills/`	Markdown-based knowledge packages injected into agent context
`apps/`	Native client apps — `apps/shared/OpenClawKit/` (Swift), `apps/ios/`, `apps/macos/`, `apps/android/`
`ui/`	Control UI web frontend (built React app served by gateway)
`docs/`	Mintlify documentation site source
`test/`	Test fixtures, helpers, mocks
`vendor/`	Vendored dependencies (`a2ui`)
`Swabble/`	Swift package for Swabble integration
`scripts/`	Build, test, and release helper scripts
`changelog/`	Changelog fragment system
`assets/`	Static assets (Chrome extension)

`src/` module breakdown:

Module	Purpose	Key Files
`src/entry.ts`	Binary entry point — warning filter, env normalization, respawn	Single file
`src/index.ts`	Library entry point — builds Commander program, exports public API	Single file

Module	Purpose	Key Files
src/cli/	CLI wiring — Commander program builder, command registration, deps injection	run-main.ts, gateway-cli.ts, deps.ts
src/config/	Config loading, validation, schema, paths, migrations	io.ts, zod-schema.ts, paths.ts
src/agents/	Agent runtime — LLM providers, model selection, tools, skills, sessions, subagents	~200+ files
src/gateway/	Gateway server — HTTP/WS, auth, channels, cron, discovery, hooks	server.impl.ts, server-http.ts, auth.ts
src/hooks/	Internal hook system — event bus for gateway/session/command/message events	internal-hooks.ts, loader.ts
src/routing/	Message routing — agent route resolution, session key construction, bindings	resolve-route.ts, session-key.ts
src/plugins/	Plugin loader, registry, service lifecycle, hook runner	loader.ts, registry.ts, types.ts, tools.ts
src/plugin-sdk/	Public SDK for extension authors (re-exports from src/plugins/types.ts)	index.ts
src/channels/	Channel plugin infrastructure — plugin registry, chat type definitions	index.ts
src/providers/	Provider-specific auth helpers (GitHub Copilot, Google, Qwen, Kilocode)	Per-provider files
src/infra/	Infrastructure — dotenv, env, ports, locks, path safety, exec approvals, TLS, SSH, Tailscale, mDNS	Various
src/security/	Security audit system	audit.ts
src/process/	Process supervisor, command queue, child process bridge	supervisor/supervisor.ts
src/memory/	Memory backend — SQLite + FTS5 + sqlite-vec vector search	Various

Module	Purpose	Key Files
src/browser/	Browser automation — Playwright + CDP control server	server.ts, pw-tools-core.*.ts
src/media/	Media pipeline — MIME detection, image ops, audio, file I/O	Various
src/media-understanding/	Multi-provider AI pipeline for audio/video/image understanding	runner.ts, providers/
src/web/	Web provider (Pi/Claude.ai web session)	Various
src/telegram/	Telegram channel (grammy)	Various
src/discord/	Discord channel (discord.js)	Various
src/slack/	Slack channel (@slack/bolt)	Various
src/whatsapp/	WhatsApp channel (Baileys)	Various
src/signal/	Signal channel	Various
src/imessage/	iMessage channel	Various
src/line/	LINE channel	Various
src/acp/	Agent Client Protocol session management	Various
src/canvas-host/	Canvas/A2UI artifact host server	Various
src/auto-reply/	Auto-reply engine — trigger detection, dispatch, templating	Various
src/daemon/	System service installer (launchd, systemd, schtasks)	service.ts
src/tui/	Terminal UI mode	Various
src/tts/	Text-to-speech abstraction	Various
src/commands/	High-level CLI command implementations	~100+ files
src/wizard/	Onboarding wizard	Various
src/logging/	Structured logging (tslog-based)	Various
src/sessions/	Session store, session key types	Various
src/terminal/	Terminal UI utilities (tables, palette, progress)	Various

Module	Purpose	Key Files
src/markdown/	Markdown rendering/transformation	Various
src/link-understanding/	Link preview/unfurl	Various
src/docs/	Docs helpers	Various
src/cron/	Cron scheduling (croner)	Various
src/pairing/	Device pairing protocol	Various
src/shared/	Shared test utilities	Various
src/types/	Shared TypeScript types	Various
src/utils/	General utility functions	Various
src/scripts/	Build/test helper scripts	Various

1.2.4 1.4 Internal Dependency Graph

```
Layer 0 (Foundation):
  src/infra/       ← env, dotenv, ports, paths, fetch, exec-approvals
  src/logging/     ← tslog-based structured logging
  src/types/       ← shared TypeScript types
  src/utils/       ← utility functions

Layer 1 (Config):
  src/config/      ← paths, schema (Zod), io, sessions, migrations
                     depends on: infra, logging

Layer 2 (Agent Core):
  src/agents/      ← model-auth, model-selection, models-config, pi-embedded-runner,
                     skills, subagent-registry, tools, sessions, workspace
                     depends on: config, infra, logging

Layer 3 (Routing & Hooks):
  src/routing/     ← resolve-route, session-key, bindings
  src/hooks/       ← internal-hooks, loader, gmail-watcher
                     depends on: config, agents

Layer 4 (Plugins):
  src/plugins/     ← loader, registry, services, hook runner
  src/plugin-sdk/  ← public API surface for extension authors
                     depends on: config, agents, hooks

Layer 5 (Gateway):
  src/gateway/     ← server.impl, server-http, server-channels, auth, cron, discovery
                     depends on: plugins, hooks, routing, agents, config
```

```
Layer 6 (CLI):
  src/cli/          ← run-main, program, command-registry, deps injection
  src/commands/     ← high-level command implementations
                      depends on: gateway, plugins, agents, config

Layer 7 (Entry):
  src/entry.ts      ← binary entry point
                      depends on: cli
```

1.2.5 1.5 What is OpenClawKit?

Path: `apps/shared/OpenClawKit/`

OpenClawKit is a **Swift Package** (Swift 6.2, iOS 18+, macOS 15+) that serves as the shared native client library for the macOS menu bar app and iOS app. It is NOT part of the server-side runtime.

It consists of three library products:

Product	Purpose
OpenClawProtocol	Swift-side representation of the gateway's JSON-over-WebSocket wire protocol. Auto-generated from TypeScript via `scripts/protocol-gen-swift.ts`
OpenClawKit	Main client library — WebSocket connection management, device auth, TLS pinning, hardware command builders (camera, screen, location, calendar, contacts), tool display metadata
OpenClawChatUI	SwiftUI chat interface components used by both macOS and iOS apps

Relationship to core: OpenClawKit connects to the gateway server over WebSocket using the protocol defined in `src/gateway/protocol/`. The gateway's `server-mobile-nodes.ts` handles mobile node registration and event subscription. The protocol types are kept in sync via code generation.

Relevance to LetsBe: Not directly relevant for server-side provisioning. However, if LetsBe ever wants to offer native mobile apps to SMB customers, OpenClawKit provides the client framework. For our VPS deployment, the gateway is what matters.

1.3 2. Startup & Bootstrap Sequence

1.3.1 2.1 Entry Point Chain

```
openclaw.mjs                          ← shebang binary (#!/usr/bin/env node)
```

```
  module.enableCompileCache()      ← Node compile cache for faster startup
  import dist/warning-filter.js    ← suppress ExperimentalWarning noise
  import dist/entry.js             ← actual entry point

     src/entry.ts
       process.title = "openclaw"
       installProcessWarningFilter()
       normalizeEnv()              ← ZAI_API_KEY alias normalization
       handle --no-color flag
       RESPAWN GUARD:              ← re-spawns with --disable-warning=ExperimentalWarning
        if not already respawned     if not already in NODE_OPTIONS
        (bounded by OPENCLAW_NODE_OPTIONS_READY=1)
       parseCliProfileArgs()       ← extracts --profile <name> from argv
       applyCliProfileEnv()        ← loads ~/.openclaw/profiles/<name>.env
       dynamic import ./cli/run-main.js → runCli(argv)

          src/cli/run-main.ts
            normalizeWindowsArgv(argv)
            loadDotEnv({ quiet: true })  ← loads .env, ~/.openclaw/.env
            normalizeEnv()
            ensureOpenClawCliOnPath()     ← ensures `openclaw` is in PATH
            assertSupportedRuntime()      ← verifies Node >= 22
            tryRouteCli(argv)             ← fast-path subcli routing
            enableConsoleCapture()        ← structured log capture
            buildProgram()                ← builds Commander tree
            installUnhandledRejectionHandler()
            registerCoreCliByName()
            register plugin CLI commands
            program.parseAsync(argv)      ← dispatches to subcommand
```

1.3.2 2.2 Gateway Startup Sequence

When `openclaw gateway run` is invoked, it calls `startGatewayServer()` from `src/gateway/server.impl.ts`

```
startGatewayServer(port, opts)

  Phase 1: Configuration
    readConfigFileSnapshot()         ← read ~/.openclaw/openclaw.json (JSON5)
    auto-migrate legacy config keys
    validate config (Zod schema)     ← throws with "run openclaw doctor" if invalid
    check OPENCLAW_PLUGIN_* env vars ← auto-enable matching plugins

  Phase 2: Auth Bootstrap
    ensureGatewayStartupAuth()       ← generate/validate gateway auth token
```

```
Phase 3: Diagnostics
  startDiagnosticHeartbeat()       ← if diagnostics.enabled

Phase 4: Registry Init
  initSubagentRegistry()           ← initialize subagent tracking
  loadGatewayPlugins()             ← discover and load all extensions

Phase 5: Server Config
  resolveGatewayRuntimeConfig()    ← bind host, TLS, auth mode, control UI
  create auth rate limiters
  resolve control UI asset path
  load TLS if enabled

Phase 6: Server Creation
  create Express + ws HTTP/WS server
  bind to port 18789
  start canvas host (if enabled)

Phase 7: Onboarding Check
  run interactive setup wizard (if fresh install)

Phase 8: Sidecars
  startGatewaySidecars():
      clean stale session lock files
      start browser control server
      start Gmail watcher (if hooks.gmail.account configured)
      load internal hooks from config and discovery dirs
      start channels (Telegram, Discord, Slack, WhatsApp, etc.)
        (skipped if OPENCLAW_SKIP_CHANNELS=1)
      trigger gateway:startup internal hook
      start plugin services (background services registered by extensions)
      reconcile ACP session identities
      start memory backend (builtin SQLite or qmd)
      schedule restart sentinel wake

Phase 9: Discovery & Networking
  startGatewayDiscovery()          ← mDNS and/or wide-area DNS
  configure Tailscale exposure

Phase 10: Watchers & Maintenance
  startGatewayConfigReloader()     ← file watcher for hot config reload
  start channel health monitor
  start maintenance timers (session cleanup, update checks)
```

```
Phase 11: Boot Script
  check for BOOT.md in workspace   ← runs one-shot agent turn if exists

Phase 12: Ready
  emit startup log (bound address, auth mode, channels, skills)
```

1.3.3 2.3 Config File Loading Pipeline

Config file location: `~/.openclaw/openclaw.json` (JSON5 format, supports comments and trailing commas)

Override via: `OPENCLAW_CONFIG_PATH` or `CLAWDBOT_CONFIG_PATH` env var.

Loading pipeline (defined in `src/config/io.ts`):

1. Read raw file (JSON5 parser)
2. Resolve `$include` directives (file includes with circular-include detection)
3. Resolve `${ENV_VAR}` substitutions in string values
4. Apply dotenv fallbacks (shell env import if `env.shellEnv.enabled`)
5. Apply legacy migration if legacy keys detected
6. Validate against Zod schema (`OpenClawSchema` from `src/config/zod-schema.ts`)
7. Apply runtime defaults (model defaults, agent defaults, logging defaults, session defaults)
8. Apply runtime overrides (from `OPENCLAW_RUNTIME_OVERRIDES` env)
9. Apply config env vars (`env.vars`) to `process.env`

Dotenv precedence (highest → lowest): 1. Process env vars 2. `./.env` (project root) 3. `~/.openclaw/.env` 4. `openclaw.json` `env` block

1.3.4 2.4 Environment Variables — Complete Reference

Paths and State

Variable	Description	Default	Required
`OPENCLAW_STATE_DIR`	State/data directory	`~/.openclaw`	No
`OPENCLAW_CONFIG_PATH`	Config file path	`$STATE_DIR/openclaw.json`	No
`OPENCLAW_HOME`	Home directory override	`os.homedir()`	No
`OPENCLAW_AGENT_DIR`	Agent data directory	`$STATE_DIR/agent`	No
`OPENCLAW_OAUTH_DIR`	OAuth credentials directory	`$STATE_DIR/credentials`	No

Gateway Runtime

Variable	Description	Default	Required
`OPENCLAW_GATEWAY_TOKEN`	Auth token for gateway	(generated)	Yes (if auth=token)

Variable	Description	Default	Required
OPENCLAW_GATEWAY_PASSWORD	Auth password	none	Yes (if auth=password)
OPENCLAW_GATEWAY_PORT	Gateway listen port	18789	No
OPENCLAW_GATEWAY_BIND	Bind mode: loopback/lan/tailnet/auto	auto	No

Process Control

Variable	Description	Default
OPENCLAW_NO_RESPAWN	Skip entry-point respawn	unset
OPENCLAW_NODE_OPTIONS_READY	Already respawned guard	unset
OPENCLAW_SKIP_CHANNELS	Skip starting messaging channels	unset
OPENCLAW_SKIP_BROWSER_CONTROL_SERVER	Skip browser control server	unset
OPENCLAW_SKIP_GMAIL_WATCHER	Skip Gmail watcher startup	unset
OPENCLAW_SKIP_CANVAS_HOST	Skip canvas host server	unset
OPENCLAW_SKIP_CRON	Skip cron service	unset
OPENCLAW_DISABLE_CONFIG_CACHE	Bypass config file cache	unset
OPENCLAW_LOAD_SHELL_ENV	Import login shell environment	unset
OPENCLAW_SHELL_ENV_TIMEOUT_MS	Shell env import timeout	15000
OPENCLAW_PROFILE	CLI profile name	unset
OPENCLAW_RAW_STREAM	Enable raw stream logging	unset
OPENCLAW_NIX_MODE	Running under Nix (disables auto-install)	unset

Model Provider API Keys

Variable	Provider
ANTHROPIC_API_KEY	Anthropic Claude
OPENAI_API_KEY	OpenAI
GEMINI_API_KEY / GOOGLE_API_KEY	Google Gemini
OPENROUTER_API_KEY	OpenRouter
GROQ_API_KEY	Groq
XAI_API_KEY	xAI (Grok)
MISTRAL_API_KEY	Mistral
CEREBRAS_API_KEY	Cerebras
TOGETHER_API_KEY	Together AI
MOONSHOT_API_KEY / KIMI_API_KEY	Moonshot/Kimi
NVIDIA_API_KEY	NVIDIA NIM

Variable	Provider
VENICE_API_KEY	Venice AI
LITELLM_API_KEY	LiteLLM
VOYAGE_API_KEY	Voyage (embeddings)
ZAI_API_KEY	ZAI (z.ai)
MINIMAX_API_KEY	MiniMax
OLLAMA_API_KEY	Ollama (local)
VLLM_API_KEY	vLLM (local)
QIANFAN_API_KEY	Baidu Qianfan
AWS_ACCESS_KEY_ID + AWS_SECRET_ACCESS_KEY	AWS Bedrock
COPILOT_GITHUB_TOKEN / GH_TOKEN	GitHub Copilot
HUGGINGFACE_HUB_TOKEN / HF_TOKEN	HuggingFace
OPENAI_API_KEYS / ANTHROPIC_API_KEYS / GEMINI_API_KEYS	Comma-separated key rotation

Channel Tokens

Variable	Channel
TELEGRAM_BOT_TOKEN	Telegram
DISCORD_BOT_TOKEN	Discord
SLACK_BOT_TOKEN / SLACK_APP_TOKEN	Slack
MATTERMOST_BOT_TOKEN / MATTERMOST_URL	Mattermost
ZALO_BOT_TOKEN	Zalo
OPENCLAW_TWITCH_ACCESS_TOKEN	Twitch

Tools and Media

Variable	Purpose
BRAVE_API_KEY	Brave Search API
PERPLEXITY_API_KEY	Perplexity search
FIRECRAWL_API_KEY	Firecrawl web scraping
ELEVENLABS_API_KEY / XI_API_KEY	ElevenLabs TTS
DEEPGRAM_API_KEY	Deepgram speech recognition

Docker-Specific

Variable	Purpose	Default
`OPENCLAW_CONFIG_DIR`	Config dir mount target	`~/.openclaw`
`OPENCLAW_WORKSPACE_DIR`	Workspace mount target	`~/.openclaw/workspace`
`OPENCLAW_BRIDGE_PORT`	Bridge TCP port	`18790`
`OPENCLAW_IMAGE`	Docker image name	`openclaw:local`
`OPENCLAW_EXTRA_MOUNTS`	Extra Docker bind mounts	unset
`OPENCLAW_HOME_VOLUME`	Named Docker volume for /home/node	unset
`OPENCLAW_DOCKER_APT_PACKAGES`	Extra apt packages for image build	unset
`OPENCLAW_INSTALL_BROWSER`	Bake Chromium into main image	unset

1.3.5 2.5 Services/Connections Established at Startup

Service	Connection Type	When
Config file	File read (JSON5)	Phase 1
SQLite memory DB	File-based DB (`node:sqlite` + `sqlite-vec`)	Phase 8 (memory backend)
Browser control server	Local HTTP (Playwright + CDP)	Phase 8
Gmail watcher	Google OAuth → Gmail API	Phase 8 (if configured)
Telegram	HTTPS long-poll (grammy)	Phase 8 (if configured)
Discord	WebSocket (discord.js)	Phase 8 (if configured)
Slack	WebSocket (Socket Mode via @slack/bolt)	Phase 8 (if configured)
WhatsApp	WebSocket (Baileys)	Phase 8 (if configured)
mDNS discovery	UDP multicast	Phase 9 (if configured)
Tailscale	Local Tailscale API	Phase 9 (if configured)
Config file watcher	Chokidar (inotify/FSEvents)	Phase 10

No external databases (Postgres, Redis, etc.) are required. OpenClaw uses flat-file JSON/JSONL + embedded SQLite exclusively.

1.3.6 2.6 Minimum Viable Config

To get OpenClaw running with minimal configuration:

```
// ~/.openclaw/openclaw.json
{
  "models": {
    "providers": {
      "anthropic": {
        "apiKey": "${ANTHROPIC_API_KEY}"
      }
    }
  }
}

# ~/.openclaw/.env
ANTHROPIC_API_KEY=sk-ant-your-key-here
OPENCLAW_GATEWAY_TOKEN=your-64-char-hex-token

# Start command
openclaw gateway run --bind loopback --port 18789
```

This gives you: a gateway server with Anthropic Claude as the LLM, no messaging channels, token auth, loopback binding.

1.4 3. Plugin/Extension System

1.4.1 3.1 Architecture: Extensions vs Skills vs Hooks

OpenClaw has three distinct extension mechanisms:

Mechanism	Type	Runs Code	Location	Purpose
Extensions	TypeScript/JS packages	Yes	`extensions/`	Register tools, channels, providers, services, CLI commands, HTTP routes
Skills	Markdown documents	No	`skills/`	Inject knowledge/instructions into agent context window

Mechanism	Type	Runs Code	Location	Purpose
Hooks	TypeScript/JS modules	Yes	`src/hooks/bundled/`, workspace `hooks/`	React to events (message received, session start, etc.)

1.4.2 3.2 Extension API (Plugin SDK)

Import path: `openclaw/plugin-sdk` (resolved via Jiti alias at runtime)
Source: `src/plugin-sdk/index.ts` → re-exports from `src/plugins/types.ts`

Plugin Definition Interface

```
// An extension must default-export one of these:
type OpenClawPluginModule =
  | OpenClawPluginDefinition
  | ((api: OpenClawPluginApi) => void | Promise<void>);

type OpenClawPluginDefinition = {
  id?: string;
  name?: string;
  description?: string;
  version?: string;
  kind?: PluginKind;              // currently only "memory"
  configSchema?: OpenClawPluginConfigSchema;
  register?: (api: OpenClawPluginApi) => void | Promise<void>;
  activate?: (api: OpenClawPluginApi) => void | Promise<void>;  // alias for register
};
```

Plugin API — What Extensions Can Do

```
type OpenClawPluginApi = {
  // Identity
  id: string;
  name: string;
  version?: string;
  source: string;
  config: OpenClawConfig;
  pluginConfig?: Record<string, unknown>;
  runtime: PluginRuntime;
  logger: PluginLogger;

  // Register an agent tool (direct object or factory function)
```

```
  registerTool(tool: AnyAgentTool | OpenClawPluginToolFactory, opts?: {
    name?: string;
    names?: string[];
    optional?: boolean;     // only included if explicitly allowlisted
  }): void;

  // Register event hooks (two styles)
  registerHook(events: string | string[], handler: InternalHookHandler, opts?: OpenClawPluginH
  on<K extends PluginHookName>(hookName: K, handler: PluginHookHandlerMap[K], opts?: { priorit

  // Register a messaging channel (Telegram, Discord, etc.)
  registerChannel(registration: OpenClawPluginChannelRegistration | ChannelPlugin): void;

  // Register an AI model provider
  registerProvider(provider: ProviderPlugin): void;

  // Register HTTP routes on the gateway
  registerHttpHandler(handler: OpenClawPluginHttpHandler): void;
  registerHttpRoute(params: { path: string; handler: OpenClawPluginHttpRouteHandler }): void;

  // Register gateway WebSocket methods
  registerGatewayMethod(method: string, handler: GatewayRequestHandler): void;

  // Register CLI commands
  registerCli(registrar: OpenClawPluginCliRegistrar, opts?: { commands?: string[] }): void;

  // Register background services
  registerService(service: OpenClawPluginService): void;

  // Register slash-style commands (bypass LLM)
  registerCommand(command: OpenClawPluginCommandDefinition): void;

  // Resolve paths relative to plugin root
  resolvePath(input: string): string;
};
```

Tool Factory Pattern

```
type OpenClawPluginToolFactory = (
  ctx: OpenClawPluginToolContext
) => AnyAgentTool | AnyAgentTool[] | null | undefined;

type OpenClawPluginToolContext = {
  config?: OpenClawConfig;
  workspaceDir?: string;
```

```
  agentDir?: string;
  agentId?: string;
  sessionKey?: string;
  messageChannel?: string;
  agentAccountId?: string;
  sandboxed?: boolean;
};
```

Tool Interface (from `@mariozechner/pi-agent-core`)

```
type AnyAgentTool = AgentTool<any, unknown> & {
  ownerOnly?: boolean;  // OpenClaw extension: restrict to owner senders
};

// AgentTool has:
//   name: string
//   label: string
//   description: string
//   parameters: TSchema (TypeBox schema)
//   execute: (toolCallId: string, args: Input) => Promise<AgentToolResult<OutputDetails>>

type AgentToolResult<T> = {
  content: Array<
    | { type: "text"; text: string }
    | { type: "image"; data: string; mimeType: string }
  >;
  details?: T;
};
```

1.4.3 3.3 Extension Loading & Lifecycle

Loader: `src/plugins/loader.ts` → `loadOpenClawPlugins(options)`

Discovery Sequence

1. **Normalize config** — `normalizePluginsConfig(cfg.plugins)` resolves enable state, allow/deny lists
2. **Discover candidates** — scans 4 locations in priority order:
   - `config` origin: paths from `plugins.loadPaths` in config
   - `workspace` origin: `<workspaceDir>/.openclaw/extensions/`
   - `global` origin: `~/.openclaw/extensions/`
   - `bundled` origin: compiled-in extensions directory
3. **Load manifests** — reads `openclaw.plugin.json` from each candidate

Per-Plugin Load Sequence

1. Check for duplicate IDs (workspace/global wins over bundled)
2. **Resolve enable state** — checks `plugins.enabled`, `plugins.allow`, `plugins.deny`, per-plugin `entries[id].enabled`
3. **Security check** — verifies entry file doesn't escape plugin root; checks file ownership on Unix
4. **Load module** via Jiti (supports `.ts`, `.tsx`, `.js`, `.mjs`)
5. Extract `register` function from default export
6. **Validate config** against `configSchema` (JSON Schema via AJV)
7. **Memory slot gating** — only one `kind: "memory"` plugin activates (`plugins.slots.memory` selects)
8. **Call** `register(api)` — synchronously
9. Push to registry

Enable/Disable Configuration

```
{
  "plugins": {
    "enabled": true,            // master switch
    "allow": ["slack", "memory-core"],  // allowlist (if non-empty, only these load)
    "deny": [],                 // blocklist (always blocked)
    "slots": {
      "memory": "memory-core"   // only one memory plugin active
    },
    "entries": {
      "slack": {
        "enabled": true,
        "config": { /* plugin-specific config */ }
      }
    },
    "loadPaths": ["/path/to/custom/extensions"]
  }
}
```

1.4.4 3.4 Typed Plugin Hooks (Lifecycle Events)

Extensions can register for strongly-typed lifecycle events via `api.on()`:

Hook Name	When Fired	Can Modify?	Return Type
`before_model_resolve`	Before LLM model selection	Yes — override model/provider	`{ modelOverride?, providerOverride? }`
`before_prompt_build`	Before system prompt assembly	Yes — inject context	`{ systemPrompt?, prependContext? }`

Hook Name	When Fired	Can Modify?	Return Type
`before_agent_start`	Before agent run (legacy)	Yes — combines above	`{ prependContext?, systemPrompt? }`
`llm_input`	LLM request payload ready	No (fire-and-forget)	void
`llm_output`	LLM response received	No (fire-and-forget)	void
`agent_end`	Conversation turn complete	No (fire-and-forget)	void
`before_compaction`	Before session compaction	No	void
`after_compaction`	After compaction	No	void
`before_reset`	Before /new or /reset	No	void
`message_received`	Inbound message	No	void
`message_sending`	Outbound message	Yes — modify or cancel	`{ content?, cancel? }`
`message_sent`	After send	No	void
`before_tool_call`	**Before tool execution**	**Yes — modify params or BLOCK**	`{ params?, block?, blockReason? }`
`after_tool_call`	**After tool execution**	**No (fire-and-forget)**	void
`tool_result_persist`	Before JSONL write (SYNC)	Yes — modify message	`{ message? }`
`before_message_write`	Before message JSONL write (SYNC)	Yes — block or modify	void
`session_start`	New session started	No	void
`session_end`	Session ended	No	void
`subagent_spawning`	Subagent about to spawn	Yes — can return error	`{ error? }`
`subagent_spawned`	Subagent spawned	No	void
`subagent_ended`	Subagent ended	No	void
`gateway_start`	Gateway started	No	void
`gateway_stop`	Gateway stopping	No	void

Critical for Safety Wrapper: The `before_tool_call` hook is the primary interception point. It fires before every tool call and can: - Modify the parameters - Block the call entirely with a reason - Observe tool name, params, session context

The `after_tool_call` hook provides audit logging capability after execution.

1.4.5 3.5 Extension File Structure

```
extensions/my-plugin/
```

```
openclaw.plugin.json    ← REQUIRED: manifest
package.json            ← npm package metadata
index.ts                ← entry: default exports OpenClawPluginDefinition
```

Manifest (`openclaw.plugin.json`):

```
{
  "id": "my-plugin",
  "name": "My Plugin",
  "description": "What it does",
  "version": "1.0.0",
  "configSchema": {
    "type": "object",
    "properties": {
      "apiKey": { "type": "string" }
    }
  },
  "uiHints": {
    "apiKey": { "label": "API Key", "sensitive": true }
  }
}
```

`package.json` **entry point:**

```
{
  "openclaw": {
    "extensions": ["./index.ts"]
  }
}
```

1.4.6 3.6 Complete Extension Catalog

Chat Channel Extensions

Extension	Description	Relevance to LetsBe
`discord`	Discord channel plugin	Low — consumer chat
`slack`	Slack channel plugin	Medium — some SMBs use Slack
`telegram`	Telegram channel plugin	Low
`whatsapp`	WhatsApp channel plugin	Medium — business messaging
`signal`	Signal channel plugin	Low
`imessage`	iMessage channel plugin	Low
`msteams`	Microsoft Teams channel plugin	**High** — many SMBs use Teams
`matrix`	Matrix channel plugin	Low

Extension	Description	Relevance to LetsBe
`mattermost`	Mattermost channel plugin	Low
`googlechat`	Google Chat channel plugin	**High** — SMBs on Google Workspace
`irc`	IRC channel plugin	Low
`line`	LINE messaging channel plugin	Low
`feishu`	Feishu/Lark channel plugin	Low
`bluebubbles`	iMessage via BlueBubbles relay	Low
`nostr`	Nostr NIP-04 encrypted DMs	Low
`synology-chat`	Synology Chat channel plugin	Low
`tlon`	Tlon/Urbit channel plugin	Low
`twitch`	Twitch channel plugin	Low
`zalo` / `zalouser`	Zalo channel plugin	Low
`nextcloud-talk`	Nextcloud Talk channel plugin	Low

Memory Extensions

Extension	Description	Relevance to LetsBe
`memory-core`	Built-in file-backed memory search (SQLite + FTS5 + sqlite-vec)	**Critical** — default memory
`memory-lancedb`	LanceDB vector memory with auto-recall/capture	**High** — advanced RAG

Auth Provider Extensions

Extension	Description	Relevance to LetsBe
`copilot-proxy`	GitHub Copilot OAuth provider	Low
`google-gemini-cli-auth`	Gemini CLI OAuth provider	Medium
`minimax-portal-auth`	MiniMax Portal OAuth	Low
`qwen-portal-auth`	Qwen Portal OAuth	Low

Tool & Utility Extensions

Extension	Description	Relevance to LetsBe
`llm-task`	Structured JSON LLM tool for workflow automation	**High** — workflow tasks

Extension	Description	Relevance to LetsBe
`lobster`	Typed workflow tool with resumable approvals	**High** — business workflows
`open-prose`	OpenProse VM skill pack with /prose command	Low
`phone-control`	Arm/disarm high-risk phone node commands	Low
`device-pair`	Setup codes and device pairing approval	Low
`diagnostics-otel`	OpenTelemetry diagnostics exporter	Medium — observability
`talk-voice`	Voice selection management	Low
`voice-call`	Voice call plugin	Low
`thread-ownership`	Prevents multi-agent collisions in threads	Medium — multi-agent safety
`acpx`	ACP runtime backend (pinned CLI)	Low

1.4.7 3.7 Skills System

Skills are **NOT code**. They are Markdown documents injected into the agent's context window at inference time.

Skill Anatomy

```
skills/my-skill/
  SKILL.md          ← REQUIRED: YAML frontmatter + markdown instructions
  scripts/          ← optional executable scripts
  references/       ← optional documentation for context
  assets/           ← optional output files
```

SKILL.md Frontmatter

```
---
name: my-skill
description: "What the skill does and when to use it"
homepage: https://...
metadata:
  openclaw:
    emoji: " "
    requires:
      bins: ["himalaya"]
    install:
```

```
      - id: brew
        kind: brew
        formula: himalaya
        bins: ["himalaya"]
---

(SKILL.md body with agent instructions goes here)
```

Three-Level Progressive Disclosure

1. **Metadata** (name + description) — always in context (~100 words)
2. **SKILL.md body** — injected when skill triggers (< 5k words target)
3. **Bundled resources** — loaded by agent as needed (references/, scripts/, assets/)

Skills are loaded from `skills/` and workspace directories. The agent reads the SKILL.md and uses the instructions as procedural guidance for invoking CLI tools via the `bash` tool.

1.4.8 3.8 Building a Custom Extension (Pseudocode)

Here's what a minimal Safety Wrapper extension would look like:

```
// extensions/letsbe-safety-wrapper/index.ts
import type { OpenClawPluginApi } from "openclaw/plugin-sdk";

const safetyWrapperPlugin = {
  id: "letsbe-safety-wrapper",
  name: "LetsBe Safety Wrapper",
  version: "1.0.0",
  configSchema: {
    type: "object",
    properties: {
      policyEndpoint: { type: "string" },
      strictMode: { type: "boolean" }
    }
  },

  register(api: OpenClawPluginApi) {
    // Intercept every tool call BEFORE execution
    api.on("before_tool_call", async (event, ctx) => {
      const { toolName, params } = event;

      // Check against safety policy
      const decision = await checkSafetyPolicy(toolName, params, api.pluginConfig);

      if (decision.blocked) {
```

```
        return { block: true, blockReason: decision.reason };
      }
      if (decision.modifiedParams) {
        return { params: decision.modifiedParams };
      }
      return {};  // allow
    }, { priority: 1000 });  // high priority = runs first

    // Audit log every tool call AFTER execution
    api.on("after_tool_call", async (event) => {
      await auditLog(event.toolName, event.params, event.result, event.durationMs);
    });

    // Intercept outbound messages
    api.on("message_sending", async (event) => {
      const filtered = await contentFilter(event.content);
      if (filtered.blocked) {
        return { cancel: true };
      }
      return { content: filtered.content };
    });
  }
};

export default safetyWrapperPlugin;
```

1.5 4. AI Agent Runtime

1.5.1 4.1 Core Architecture

The AI agent runtime is the largest subsystem in OpenClaw (~200+ files in `src/agents/`). It is built on top of `@mariozechner/pi-agent-core`, a TypeScript agent SDK that provides the core conversation loop.

Key runtime components:

Component	Location	Purpose
`pi-embedded-runner/`	`src/agents/pi-embedded-runner/`	Core agent run loop wrapping pi-agent-core
`model-auth.ts`	`src/agents/model-auth.ts`	Multi-provider API key resolution
`model-selection.ts`	`src/agents/model-selection.ts`	Model selection and validation

Component	Location	Purpose
`models-config.ts`	`src/agents/models-config.ts`	Provider catalog (implicit + explicit)
`models-config.providers.ts`	`src/agents/models-config.providers.ts`	Built-in provider definitions
`tool-policy.ts`	`src/agents/tool-policy.ts`	Tool allowlist/denylist enforcement
`skills.ts`	`src/agents/skills.ts`	Skill loading and prompt injection
`subagent-registry.ts`	`src/agents/subagent-registry.ts`	Active subagent tracking
`workspace.ts`	`src/agents/workspace.ts`	Workspace directory management
`identity.ts`	`src/agents/identity.ts`	Agent identity resolution
`defaults.ts`	`src/agents/defaults.ts`	Default provider/model

1.5.2 4.2 Supported LLM Providers

Default provider: `anthropic` **Default model:** `claude-opus-4-6`
Defined in `src/agents/defaults.ts` and `src/agents/models-config.providers.ts`:

Provider ID	Models (examples)	Auth Me
`anthropic`	claude-opus-4-6, claude-sonnet-4-6, claude-haiku-4-5	API key
`openai`	gpt-5.1-codex, o3, gpt-4o	API key
`google`	gemini-2.5-pro, gemini-2.5-flash	API key
`openrouter`	Any model via OpenRouter	API key
`groq`	llama, mixtral, gemma	API key
`xai`	grok models	API key
`mistral`	mistral-large, codestral	API key
`cerebras`	cerebras models	API key
`together`	various open models	API key
`ollama`	any local model	Local (n
`vllm`	any local model	Local (n
`amazon-bedrock`	claude, titan, llama via AWS	AWS cre
`google-vertex`	gemini via Vertex AI	Google A
`github-copilot`	copilot models	GitHub C
`minimax` / `minimax-portal`	MiniMax models	API key /
`moonshot` / `kimi-coding`	Kimi models	API key
`qwen-portal`	Qwen models	OAuth
`nvidia`	NVIDIA NIM models	API key
`venice`	Venice AI models	API key

Provider ID	Models (examples)	Auth Me
`litellm`	any model via LiteLLM proxy	API key
`volcengine` / `byteplus`	ByteDance models	API key
`qianfan`	Baidu models	API key
`huggingface`	HF models	Token
`kilocode` / `opencode`	Specialized models	API key
`zai`	z.ai models	API key
`xiaomi`	Mimo models	API key
`chutes`	Chutes models	OAuth /
`vercel-ai-gateway` / `cloudflare-ai-gateway`	Gateway proxy	API key
`synthetic`	Testing only	API key

API key resolution chain (from `src/agents/model-auth.ts`): 1. Auth profile store (`~/.openclaw/agents/<agentId>/auth-profiles.json`) 2. Environment variables (`ANTHROPIC_API_KEY`, etc.) 3. Config file (`models.providers.<id>.apiKey`) 4. AWS SDK (for Bedrock) 5. Key rotation lists (`OPENAI_API_KEYS=sk-1,sk-2`)

1.5.3 4.3 Tool/Function Calling

How Tools Are Registered Tools come from three sources:

1. **Core built-in tools** — defined in `src/agents/tools/` (bash, browser, web_search, etc.)
2. **Plugin tools** — registered via `api.registerTool()` in extensions
3. **Skill-derived tools** — skills inject instructions for using CLI tools via the `bash` tool

Tool Registration Flow

```
Plugin discovery → loadOpenClawPlugins()
  → for each plugin: call register(api)
    → api.registerTool(toolOrFactory, opts)
      → toolOrFactory stored in registry

At agent start → resolvePluginTools()
  → for each registered tool factory:
    → call factory(OpenClawPluginToolContext)
    → returns AnyAgentTool[] or null
  → merge with core tools
  → apply tool policy (allowlist/denylist)
  → pass to pi-agent-core
```

Tool Policy Pipeline Defined in `src/agents/tool-policy.ts` and `src/agents/tool-policy-pipeline.t`

```
// Tools can be controlled via config:
{
  "tools": {
    "allowlist": ["exec", "web_search", "browser"],  // only these tools
    "denylist": ["sessions_spawn"],                    // never these tools
    "groups": {
      "plugins": true    // enable all plugin tools
    }
  }
}
```

Tool schemas use **TypeBox** (not Zod) for LLM-compatible JSON Schema generation. Important constraint from CLAUDE.md: avoid `Type.Union` — use `stringEnum`/`optionalStringEnum` instead.

1.5.4 4.4 Agent Execution Loop

The core execution loop lives in `src/agents/pi-embedded-runner/`:

```
User Message Arrives (via channel, HTTP API, or WS)

    resolveAgentRoute()              ← determine which agent handles this
    construct session key             ← e.g., "agent:main:direct:telegram:12345"

    Load agent config                 ← agents.list[agentId] from config
    resolveAgentIdentity()            ← name, avatar, ack reaction
    Load workspace                    ← MEMORY.md, SYSTEM.md, bootstrap files

    Resolve model                     ← model-auth + model-selection
    Load skills                       ← inject SKILL.md content into system prompt
    Resolve tools                     ← core + plugin tools, apply policy

    Fire "before_model_resolve" hook
    Fire "before_prompt_build" hook
    Fire "before_agent_start" hook

    Build system prompt               ← agent identity + skills + workspace context
    Load session history              ← from JSONL transcript file

    queueEmbeddedPiMessage()          ← enqueue message for processing

        pi-agent-core run loop:
            Send messages to LLM provider
            Fire "llm_input" hook
            Stream response tokens
```

```
            Fire "llm_output" hook

            If tool call requested:
                Fire "before_tool_call" hook  ← CAN BLOCK OR MODIFY
                Execute tool
                Fire "after_tool_call" hook
                Fire "tool_result_persist" hook (SYNC)
                Write tool result to session JSONL
                Loop back to LLM with tool result

            If text response:
                Fire "message_sending" hook  ← CAN MODIFY OR CANCEL
                Fire "before_message_write" hook (SYNC)
                Write to session JSONL
                Deliver to channel
                Fire "message_sent" hook

    Fire "agent_end" hook
    Update session state
```

1.5.5 4.5 Multi-Turn Conversations & Context

Session persistence: Each conversation session is stored as a JSONL file at `~/.openclaw/agents/<ag` Each line is a transcript event (user message, assistant message, tool call, tool result).

Session key format: `agent:<agentId>:<channel>:<accountId>:<peerKind>:<peerId>`

DM scope options (control session isolation): - `"main"` — all DMs share one session - `"per-peer"` — one session per peer across channels - `"per-channel-peer"` — one session per channel+peer - `"per-account-channel-peer"` — maximum isolation

Context window management: - Session history is loaded from JSONL at each turn - When context exceeds the model's window, **compaction** is triggered - `compactEmbeddedPiSessio` summarizes older messages to fit within limits - `before_compaction` and `after_compaction` hooks fire around this

Memory/RAG: - The memory backend (SQLite + FTS5 + sqlite-vec) indexes workspace markdown files and session transcripts - `memory_search` tool performs hybrid BM25 + vector similarity search - Auto-recall (via `memory-lancedb` extension) can inject relevant memories before each turn

1.5.6 4.6 Agent Configuration

Agents are configured in `openclaw.json`:

```
{
  "agents": {
    "defaults": {
      "model": "claude-sonnet-4-6",
```

```
      "provider": "anthropic",
      "sandbox": {
        "mode": "off",          // "off" | "non-main" | "all"
        "scope": "agent"        // "session" | "agent" | "shared"
      }
    },
    "list": [
      {
        "id": "main",
        "default": true,
        "model": "claude-opus-4-6",
        "systemPrompt": "You are a helpful business assistant.",
        "tools": { "allowlist": ["exec", "web_search", "browser", "memory_search"] }
      },
      {
        "id": "researcher",
        "model": "gemini-2.5-pro",
        "systemPrompt": "You are a research specialist."
      }
    ]
  }
}
```

Routing (from `src/routing/resolve-route.ts`) determines which agent handles a message:

Resolution tiers (highest priority first): 1. `binding.peer` — exact peer match 2. `binding.peer.parent` — thread parent match 3. `binding.guild+roles` — Discord guild + role 4. `binding.guild` — Discord guild 5. `binding.team` — Slack team 6. `binding.account` — account-level 7. `binding.channel` — channel wildcard 8. `default` — first agent with `default: true`

1.5.7 4.7 Subagent System

OpenClaw supports spawning child agent sessions:

```
// From src/agents/subagent-registry.ts
initSubagentRegistry()    // initialize at gateway start
// From src/agents/subagent-spawn.ts
spawnSubagent(opts)       // spawn child agent session
```

Subagents are tracked by the registry with depth limits to prevent infinite recursion. Session keys for subagents contain `:subagent:` marker. The `sessions_spawn` tool allows the primary agent to delegate tasks to specialized subagents.

1.6 5. Tool & Integration Catalog

1.6.1 5.1 Core Built-in Tools

These are always available (subject to tool policy):

Tool	File	What It Does	Protocol/API
`exec`	`bash-tools.ts`	Shell command execution	Local shell + optional PTY
`browser`	`browser-tool.ts`	Full browser automation (navigate, click, type, snapshot, screenshot, PDF)	Playwright + CDP
`web_search`	`web-search.ts`	Web search via Brave, Perplexity, Grok, Gemini, or Kimi	REST APIs
`web_fetch`	`web-fetch.ts`	Fetch/scrape URLs (markdown extraction, Firecrawl)	HTTP + Firecrawl API
`image`	`image-tool.ts`	Describe/analyze images using vision models	LLM vision API
`memory_search`	`memory-tool.ts`	Semantic memory search (hybrid BM25 + vector)	Local SQLite
`memory_get`	`memory-tool.ts`	Raw memory file read	Local filesystem
`message`	`message-tool.ts`	Send/read/react/edit messages across all channels	Channel APIs
`canvas`	`canvas-tool.ts`	Present HTML on connected Mac/iOS/Android nodes	WebSocket to nodes
`nodes`	`nodes-tool.ts`	Camera, screen record, location, notifications on connected devices	WebSocket to nodes
`cron`	`cron-tool.ts`	Create/update/remove scheduled jobs	croner library
`tts`	`tts-tool.ts`	Text-to-speech output	ElevenLabs, Deepgram, etc.
`sessions_spawn`	`sessions-spawn-tool.ts`	Spawn sub-agents	Internal
`sessions_send`	`sessions-send-tool.ts`	Send messages to other sessions	Internal
`sessions_list`	`sessions-list-tool.ts`	List active sessions	Internal
`sessions_history`	`sessions-history-tool.ts`	Read session history	Local JSONL
`session_status`	`session-status-tool.ts`	Current session info	Internal
`agents_list`	`agents-list-tool.ts`	List available agents	Internal
`gateway`	`gateway-tool.ts`	Direct gateway method calls	Internal WS

1.6.2 5.2 Google Integration (DETAILED)

Skill: `skills/gog/SKILL.md` **CLI:** `gog` binary (external Golang CLI wrapping Google Workspace APIs)

Supported Google APIs

Service	Operations
Gmail	Search threads, search messages, send (plain/HTML/body-file), create/send drafts, reply, list attachments
Calendar	List events, create events (with color IDs 1-11), update events, list calendars
Drive	Search files/folders
Contacts	List contacts
Sheets	Get ranges, update cells, append rows, clear ranges, sheet metadata
Docs	Export docs (txt/PDF), cat doc content

OAuth Setup

```
# 1. Register credentials (client_secret.json from Google Cloud Console)
gog auth credentials /path/to/client_secret.json

# 2. Add account + select scopes
gog auth add user@gmail.com --services gmail,calendar,drive,contacts,docs,sheets

# 3. Verify
gog auth list
```

The OAuth flow uses a standard Google OAuth2 browser redirect. The `gog` CLI handles token storage locally in its own config directory. An optional convenience env var:

```
export GOG_ACCOUNT=user@gmail.com   # avoid repeating --account
```

Tool Exposure The `gog` skill does NOT register a structured plugin tool. Instead, it teaches the agent to invoke the `gog` CLI via the built-in `exec` (bash) tool. Command patterns:

```
gog gmail search 'newer_than:7d' --max 10
gog gmail send --to recipient@example.com --subject "Subject" --body-file -
gog calendar events <calendarId> --from 2026-02-26T00:00:00Z --to 2026-02-27T00:00:00Z
gog calendar create <calendarId> --summary "Meeting" --from <iso> --to <iso> --event-color 7
gog drive search "quarterly report" --max 10
```

```
gog contacts list --max 20
gog sheets get <sheetId> "Sheet1!A1:D10" --json
gog sheets update <sheetId> "Sheet1!A1:B2" --values-json '[["A","B"]]'
gog docs export <docId> --format txt --out /tmp/doc.txt
```

Configuration for LetsBe For each customer VPS, we would need to: 1. Pre-install the `gog` binary 2. Create a Google Cloud project with OAuth credentials 3. Run `gog auth credentials` with the client_secret.json 4. Run `gog auth add` for the customer's Google account 5. Store tokens in the gog config directory within the container

There is also a `goplaces` skill for Google Places API (New): text search, place details, reviews via a separate `goplaces` binary.

1.6.3 5.3 IMAP/Himalaya Email (DETAILED)

Skill: `skills/himalaya/SKILL.md` **CLI:** `himalaya` binary (external Rust CLI email client)

Configuration Config file: `~/.config/himalaya/config.toml`

```
[accounts.personal]
email = "user@example.com"
display-name = "User Name"
default = true

backend.type = "imap"
backend.host = "imap.example.com"
backend.port = 993
backend.encryption.type = "tls"
backend.login = "user@example.com"
backend.auth.type = "password"
backend.auth.cmd = "pass show email/imap"   # or keyring command

message.send.backend.type = "smtp"
message.send.backend.host = "smtp.example.com"
message.send.backend.port = 587
message.send.backend.encryption.type = "start-tls"
message.send.backend.login = "user@example.com"
message.send.backend.auth.cmd = "pass show email/smtp"
```

Supported backends: IMAP, SMTP, Notmuch, Sendmail. Passwords retrieved via `pass`, keyring, or any shell command.

Operations Exposed to Agent

```
# Listing / searching
himalaya envelope list                              # INBOX
```

```
himalaya envelope list --folder "Sent"
himalaya envelope list --page 1 --page-size 20
himalaya envelope list from john@example.com subject meeting

# Reading
himalaya message read 42
himalaya message export 42 --full               # raw MIME

# Composing / sending
himalaya template send < /dev/stdin
himalaya message write -H "To:r@e.com" -H "Subject:Hi" "body"

# Replying / forwarding
himalaya message reply 42
himalaya message reply 42 --all
himalaya message forward 42

# Moving / organizing
himalaya message move 42 "Archive"
himalaya message copy 42 "Important"
himalaya message delete 42
himalaya flag add 42 --flag seen

# Attachments
himalaya attachment download 42 --dir ~/Downloads

# Multi-account
himalaya --account work envelope list

# JSON output
himalaya envelope list --output json
```

Message composition supports MML (MIME Meta Language) syntax for rich emails with attachments.

Configuration for LetsBe For each customer VPS: 1. Pre-install `himalaya` binary 2. Configure `~/.config/himalaya/config.toml` with customer's IMAP/SMTP settings 3. Store email credentials securely (password command or keyring) 4. Test with `himalaya envelope list`

1.6.4 5.4 Web Search (Brave Search)

Location: `src/agents/tools/web-search.ts`

Built directly into the core `web_search` tool (no separate skill needed).

Supported providers: `brave`, `perplexity`, `grok`, `gemini`, `kimi`

Brave Search config: - Endpoint: `https://api.search.brave.com/res/v1/web/search` - API key from config (`tools.web.search.apiKey`) or `BRAVE_API_KEY` env var - Returns up to 10 results per query - Supports country, language, freshness filters

Tool schema:

```
const WebSearchSchema = Type.Object({
  query: Type.String(),
  count: Type.Optional(Type.Number({ minimum: 1, maximum: 10 })),
  country: Type.Optional(Type.String()),       // "US", "DE"
  search_lang: Type.Optional(Type.String()),   // "en", "de"
  freshness: Type.Optional(Type.String()),     // "pd", "pw", "pm", "py"
});
```

1.6.5 5.5 Browser Automation

Location: `src/browser/`

Full browser automation via Playwright + Chrome DevTools Protocol (CDP).

Browser tool actions:

Action	What It Does
`status`	Check browser state
`start` / `stop`	Start/stop browser
`profiles`	List Chrome profiles
`tabs`	List open tabs
`open`	Open new tab
`focus` / `close`	Focus/close tab
`navigate`	Go to URL
`snapshot`	Accessibility tree (aria or ai format)
`screenshot`	Capture PNG/JPEG
`console`	Get console messages
`pdf`	Save as PDF
`upload`	File upload
`dialog`	Handle browser dialogs
`act`	Perform interaction (click, type, press, hover, drag, select, fill, resize, wait, evaluate)

Security: Navigation guarded by `navigation-guard.ts` (SSRF policy via `src/infra/net/ssrf.ts`). Content wrapped with `wrapExternalContent()`.

1.6.6 5.6 Calendar

No native Cal.com integration exists. Calendar capabilities come from: 1. **Google Calendar** via the `gog` skill 2. **Apple Calendar/Reminders** via `apple-reminders` skill (macOS only) 3. **Cron tool** for scheduled reminders

1.6.7 5.7 Complete Skills Catalog

Communication / Messaging

Skill	What It Does	Requires
`discord`	Send/read/react/edit messages, polls, threads, search	Discord bot token
`slack`	Send/read/edit messages, react, pin, member info	Slack bot token
`bluebubbles`	iMessage via BlueBubbles server	BlueBubbles config
`imsg`	iMessage/SMS via CLI	`imsg` binary (macOS)
`wacli`	WhatsApp send + history search	`wacli` binary
`voice-call`	Start voice calls	voice-call plugin
`xurl`	X/Twitter: post, reply, DM, search, follow	X API key

Google / Productivity

Skill	What It Does	Requires
`gog`	Gmail, Calendar, Drive, Contacts, Sheets, Docs	`gog` binary + OAuth
`goplaces`	Google Places search, details, reviews	`goplaces` binary
`gemini`	One-shot Q&A, summaries via Gemini	`gemini` CLI
`notion`	Notion pages, databases, blocks	Notion API key
`obsidian`	Obsidian vault read/write	`obsidian-cli` binary
`apple-notes`	Apple Notes via `memo`	`memo` binary (macOS)
`apple-reminders`	Apple Reminders management	`remindctl` binary (macOS)
`bear-notes`	Bear notes management	`grizzly` binary (macOS)
`things-mac`	Things 3 todos/projects	`things` binary (macOS)
`trello`	Trello boards, lists, cards	Trello API key/token

Email

Skill	What It Does	Requires
`himalaya`	Full IMAP/SMTP email management	`himalaya` binary + TOML config

Developer / Code

Skill	What It Does	Requires
`github`	Issues, PRs, CI, code review, API queries	`gh` binary
`coding-agent`	Delegate coding tasks to Codex/Claude Code agents	`bash` with PTY
`gh-issues`	Fetch issues, spawn fix agents, open PRs	`gh` binary
`tmux`	Remote-control tmux sessions	`tmux` binary

Media / Audio / Vision

Skill	What It Does	Requires
`openai-whisper`	Local speech-to-text (offline)	`whisper` binary
`openai-whisper-api`	Speech-to-text via OpenAI API	OpenAI API key
`sherpa-onnx-tts`	Local TTS (offline)	sherpa-onnx binary
`sag`	ElevenLabs TTS	`sag` binary
`video-frames`	Extract frames/clips from video	`ffmpeg`
`openai-image-gen`	Image generation via OpenAI	OpenAI API key
`nano-banana-pro`	Image gen/edit via Gemini 3 Pro	Gemini API key
`peekaboo`	macOS UI capture + automation	Peekaboo app
`camsnap`	RTSP/ONVIF camera capture	`camsnap` CLI

Smart Home / Hardware

Skill	What It Does	Requires
`openhue`	Philips Hue lights/scenes	`openhue` CLI
`sonoscli`	Sonos speakers	`sonoscli` binary
`blucli`	BluOS speaker control	`blu` binary
`eightctl`	Eight Sleep pod control	`eightctl` binary
`spotify-player`	Spotify playback/search	Spotify account

Web / Search / Content

Skill	What It Does	Requires
`summarize`	Summarize/transcribe URLs, podcasts, files	`summarize` binary
`blogwatcher`	Monitor blogs and RSS/Atom feeds	`blogwatcher` CLI
`weather`	Weather and forecasts	`curl` (no API key)

Security / System

Skill	What It Does	Requires
`1password`	1Password secrets management	`op` binary
`healthcheck`	Security audit, firewall, SSH checks	various
`session-logs`	Search/analyze session logs	`jq`, `rg`
`model-usage`	Per-model cost/usage summary	`codexbar` CLI

Agent / Platform

Skill	What It Does	Requires
`canvas`	Display HTML on connected nodes	Canvas host
`clawhub`	Search/install/publish skills	`clawhub` npm CLI
`skill-creator`	Create/update agent skills	—
`mcporter`	MCP server bridge	`mcporter` CLI
`nano-pdf`	Edit PDFs with natural language	`nano-pdf` CLI

1.6.8 5.8 Tool Execution & Sandboxing

Tool execution flow:

1. LLM generates tool call request
2. `before_tool_call` plugin hook fires (can block or modify)
3. Tool's `execute()` function runs in the gateway Node.js process
4. For shell commands (`exec` tool): execution happens either locally or in a Docker sandbox container
5. `after_tool_call` plugin hook fires
6. Result returned to LLM

Sandbox modes (from `agents.defaults.sandbox.mode`): - `"off"` — no sandboxing (default) - `"non-main"` — sandbox non-main sessions only - `"all"` — every session sandboxed

When sandboxed, tool calls that execute shell commands run inside Docker containers with hardened defaults (see Section 7 for container details).

1.6.9 5.9 Media Understanding Pipeline

Location: `src/media-understanding/`

Multi-provider AI pipeline for understanding audio, video, and images:

Capability	Providers
Audio transcription	Groq (Whisper), OpenAI (Whisper), Google (Gemini), Deepgram
Image description	OpenAI (GPT-4o vision), Anthropic (Claude vision), Google (Gemini), Mistral, Moonshot
Video description	Google (Gemini — native video support)

Provider auto-selection cascades from the primary LLM provider if it supports the capability, falling back to available alternatives.

1.7 6. Data & Storage

1.7.1 6.1 Storage Architecture

OpenClaw uses **no external databases**. All data is stored as flat files and embedded SQLite:

```
~/.openclaw/                          ← OPENCLAW_STATE_DIR
  openclaw.json                       ← Main config (JSON5)
  credentials/                        ← OAuth tokens, provider auth
  identity/
    device.json                       ← Ed25519 keypair + deviceId
  pairing/                            ← Node pairing state
  agents/
    <agentId>/
      auth-profiles.json              ← Per-agent auth profiles
      sessions/
        <sessionKey>.jsonl            ← Conversation transcripts
      memory.db                       ← SQLite memory index (if builtin)
  workspace/                          ← Agent workspace
    MEMORY.md                         ← Persistent memory (markdown)
    memory/                           ← Additional memory files
  sandboxes/                          ← Sandbox workspaces
  extensions/                         ← User-installed extensions
  hooks/                              ← User hooks
  .env                                ← Environment variable overrides
```

1.7.2 6.2 Data Model

Conversations and Messages Sessions are stored as JSONL files. Each line is a transcript event:

{"type":"user","content":"What's on my calendar today?","timestamp":"2026-02-26T10:00:00Z","cha
{"type":"assistant","content":"Let me check your calendar.","timestamp":"2026-02-26T10:00:01Z"}
{"type":"tool_call","id":"tc_1","name":"exec","params":{"command":"gog calendar events primary
{"type":"tool_result","id":"tc_1","content":[{"type":"text","text":"Meeting at 2pm: Team sync"}
{"type":"assistant","content":"You have a meeting at 2pm: Team sync.","timestamp":"2026-02-26T1

Session Keys Format: `agent-<agentId>/<channel>/<accountId>/<peerKind>/<peerId>`
Special prefixes: - `cron:*` — cron run sessions - `subagent:*` — sub-agent sessions - `acp:*` — ACP sessions

Users/Agents Agents are defined in config (`agents.list[]`). There is no separate user database — OpenClaw's trust model is single-operator (one trusted user per gateway).

1.7.3 6.3 Credentials Storage

What	Where	Format
Gateway auth token	`openclaw.json` → `gateway.auth.token` or env `OPENCLAW_GATEWAY_TOKEN`	Hex string
LLM provider API keys	`openclaw.json` → `models.providers.<id>.apiKey` or env vars	String
OAuth tokens	`~/.openclaw/credentials/` directory	JSON files
Per-agent auth profiles	`~/.openclaw/agents/<agentId>/auth-profiles.json`	JSON
Device identity	`~/.openclaw/identity/device.json`	Ed25519 keypair

Security concern: Config file `openclaw.json` may contain plaintext API keys. The security audit (`src/security/audit.ts`) checks for world-readable permissions.

1.7.4 6.4 Memory/Knowledge Persistence

Built-in Memory Backend (`src/memory/`) Uses Node.js experimental `node:sqlite` module with extensions:

Component	Technology	Purpose
Full-text search	SQLite FTS5 (`chunks_fts` table)	BM25 keyword search
Vector similarity	`sqlite-vec` extension (`chunks_vec` table)	Cosine similarity search
Embedding cache	SQLite table (`embedding_cache`)	Avoid re-embedding

Embedding providers: OpenAI, Gemini, Voyage, Mistral, local (llama.cpp via `node-llama`)

Search strategy: Hybrid BM25 keyword + cosine vector similarity with MMR (Maximal Marginal Relevance) reranking and temporal decay scoring.

Sources indexed: 1. Workspace markdown files (`MEMORY.md`, `memory/*.md`) 2. Session JSONL transcripts

`MemoryIndexManager` manages per-agent SQLite DBs with: - Batch embedding with configurable concurrency - File watching via chokidar for incremental re-indexing - Snippet max: 700 chars per chunk - Max batch failures before lockout: 2

memory-core Extension The `memory-core` extension (default) wraps the built-in memory backend, exposing `memory_search` and `memory_get` tools plus CLI commands.

memory-lancedb Extension Alternative memory backend using LanceDB (vector database) with: - OpenAI embeddings (text-embedding-3-small or text-embedding-3-large) - Auto-capture from user messages - Auto-recall: inject relevant memories before each agent turn - Tools: `memory_recall`, `memory_store`, `memory_forget`

1.7.5 6.5 Temp File Management

Location: `src/infra/tmp-openclaw-dir.ts`

- Preferred: `/tmp/openclaw` (validated: writable, owned by current user, not group/world writable)
- Fallback: `os.tmpdir()/openclaw` or `openclaw-<uid>`
- Used for media handoff between host and sandbox

1.8 7. Deployment & Configuration

1.8.1 7.1 Docker Images

OpenClaw ships four Dockerfiles:

Main Gateway Image (`Dockerfile`)

```
# Base: node:22-bookworm (pinned by digest)
# Build process:
#   1. Install Bun (required for build scripts)
#   2. Enable corepack (pnpm)
#   3. pnpm install --frozen-lockfile (NODE_OPTIONS=--max-old-space-size=2048)
#   4. Optional: OPENCLAW_INSTALL_BROWSER=1 bakes Chromium + Playwright (~300MB extra)
#   5. pnpm build && pnpm ui:build
#   6. Runs as non-root 'node' user (uid 1000)
# CMD: node openclaw.mjs gateway --allow-unconfigured
```

```
# Default bind: 127.0.0.1 (loopback)
# For containers: override CMD with --bind lan
```

Key build args: - `OPENCLAW_DOCKER_APT_PACKAGES` — extra apt packages - `OPENCLAW_INSTALL_BROWSER` — bake Chromium into the image

Sandbox Image (`Dockerfile.sandbox`)

```
# Base: debian:bookworm-slim (pinned by digest)
# Installs: bash, ca-certificates, curl, git, jq, python3, ripgrep
# Creates 'sandbox' user (non-root)
# CMD: sleep infinity (stays alive for exec injection)
# Image name: openclaw-sandbox:bookworm-slim
```

Minimal container for executing agent shell commands in isolation.

Sandbox Browser Image (`Dockerfile.sandbox-browser`)

```
# Base: debian:bookworm-slim (pinned by digest)
# Installs: bash, ca-certificates, chromium, curl, fonts-liberation,
#           fonts-noto-color-emoji, git, jq, novnc, python3, socat,
#           websockify, x11vnc, xvfb
# Exposes:
#   9222 - Chrome DevTools Protocol (CDP)
#   5900 - VNC
#   6080 - noVNC web viewer
# CMD: openclaw-sandbox-browser (entrypoint script)
# Image name: openclaw-sandbox-browser:bookworm-slim
```

Browser-enabled sandbox with Chromium, virtual display (Xvfb), VNC, and noVNC.

Extended Sandbox Image (`Dockerfile.sandbox-common`)

```
# Base: openclaw-sandbox:bookworm-slim (parameterized)
# Adds: nodejs, npm, python3, golang-go, rustc, cargo, pnpm, Bun, Homebrew Linux
# Sets PATH to include Bun (/opt/bun) and Homebrew bins
# Image name: openclaw-sandbox-common:bookworm-slim
```

Full development environment for agents that need to build/run code.

1.8.2 7.2 Docker Compose

File: `docker-compose.yml`

Two services:

```
services:
  openclaw-gateway:
    image: ${OPENCLAW_IMAGE:-openclaw:local}
    environment:
      - HOME
      - TERM
      - OPENCLAW_GATEWAY_TOKEN
      - CLAUDE_AI_SESSION_KEY
      - CLAUDE_WEB_SESSION_KEY
      - CLAUDE_WEB_COOKIE
    volumes:
      - ${OPENCLAW_CONFIG_DIR}:/home/node/.openclaw
      - ${OPENCLAW_WORKSPACE_DIR}:/home/node/.openclaw/workspace
    ports:
      - "${OPENCLAW_GATEWAY_PORT:-18789}:18789"    # Gateway HTTP + WS
      - "${OPENCLAW_BRIDGE_PORT:-18790}:18790"     # Bridge (legacy)
    command: node dist/index.js gateway --bind ${OPENCLAW_GATEWAY_BIND:-lan} --port 18789
    init: true
    restart: unless-stopped

  openclaw-cli:
    # Same image, stdin/tty, no ports
    entrypoint: node dist/index.js
    stdin_open: true
    tty: true
```

1.8.3 7.3 Docker Setup Script (`docker-setup.sh`)

Step-by-step operations:

1. **Token resolution:** Reads `gateway.auth.token` from `~/.openclaw/openclaw.json` (via Python3 or Node), or generates a new 64-char hex token via `openssl rand -hex 32`
2. **Path validation:** Validates mount paths (no whitespace, no control chars), validates named volume names
3. **Directory creation:** Creates `~/.openclaw`, `~/.openclaw/workspace`, `~/.openclaw/identity`
4. **Write `.env`:** Writes all config variables to `.env` file via `upsert_env`:
   - `OPENCLAW_CONFIG_DIR`, `OPENCLAW_WORKSPACE_DIR`
   - `OPENCLAW_GATEWAY_PORT` (default 18789), `OPENCLAW_BRIDGE_PORT` (default 18790)
   - `OPENCLAW_GATEWAY_BIND` (default `lan`)
   - `OPENCLAW_GATEWAY_TOKEN`, `OPENCLAW_IMAGE`
   - `OPENCLAW_EXTRA_MOUNTS`, `OPENCLAW_HOME_VOLUME`, `OPENCLAW_DOCKER_APT_PACKAGES`
5. **Build or pull image:** If `IMAGE_NAME=openclaw:local`: `docker build`. Otherwise: `docker pull`

6. **Onboarding:** `docker compose run --rm openclaw-cli onboard --no-install-daemon`
7. **CORS config:** Configures `gateway.controlUi.allowedOrigins` for non-loopback binds
8. **Start:** `docker compose up -d openclaw-gateway`

1.8.4 7.4 Sandbox Architecture

How sandboxing works (from `src/process/supervisor/`):

The gateway stays on the host. Tool execution (shell commands) can be isolated inside Docker containers.

Configuration:

```
{
  "agents": {
    "defaults": {
      "sandbox": {
        "mode": "off",       // "off" | "non-main" | "all"
        "scope": "agent",    // "session" | "agent" | "shared"
        "docker": {
          "readOnlyRoot": true,
          "tmpfs": ["/tmp", "/var/tmp", "/run"],
          "network": "none",
          "user": "1000:1000",
          "capDrop": ["ALL"],
          "pidsLimit": 256,
          "memory": "1g",
          "memorySwap": "2g",
          "cpus": 1,
          "ulimits": {
            "nofile": { "soft": 1024, "hard": 2048 },
            "nproc": 256
          }
        }
      }
    }
  }
}
```

Scope options: - `"session"` — one container per session - `"agent"` — one container per agent (default) - `"shared"` — one container for all sessions (no cross-session isolation)

Workspace access: - `"none"` (default) — sandbox uses `~/.openclaw/sandboxes`; agent workspace not visible - `"ro"` — agent workspace read-only at `/agent` - `"rw"` — agent workspace read/write at `/workspace`

Networking: `network: "none"` by default. `host` blocked. `container:<id>` blocked.

Container lifecycle: Gateway spawns containers on-demand; reused per scope. Auto-pruned after idle >24h or age >7 days.

Default tool allow/deny in sandbox: - Allow: `exec`, `process`, `read`, `write`, `edit`, `sessions_list`, `sessions_history`, `sessions_send`, `sessions_spawn`, `session_status` - Deny: `browser`, `canvas`, `nodes`, `cron`, `discord`, `gateway`

Process Supervisor (`src/process/supervisor/supervisor.ts`): `createProcessSupervisor()` manages ManagedRun instances with UUID tracking and state machine (`starting` → `exiting`).

1.8.5 7.5 Ports

Port	Service	Default Bind	Notes
18789	Gateway (HTTP + WS multiplexed)	`127.0.0.1`	All API endpoints
18790	Bridge (legacy TCP)	configured	Deprecated
9222	Chromium CDP	sandbox-browser	Chrome DevTools Protocol
5900	VNC	sandbox-browser	x11vnc
6080	noVNC	sandbox-browser	Web-based VNC viewer

Bind modes: - `"loopback"` → `127.0.0.1` (most secure, default) - `"lan"` → `0.0.0.0` (all interfaces) - `"tailnet"` → Tailscale IPv4 address - `"auto"` → prefer loopback, else LAN

1.8.6 7.6 Volumes

Volume	Container Path	Purpose
`${OPENCLAW_CONFIG_DIR}`	`/home/node/.openclaw`	Config, credentials, sessions, memory
`${OPENCLAW_WORKSPACE_DIR}`	`/home/node/.openclaw/workspace`	Agent workspace files
`${OPENCLAW_EXTRA_MOUNTS}`	Various	Additional bind mounts

1.8.7 7.7 Minimum System Requirements

Based on Dockerfile analysis and runtime characteristics:

Resource	Minimum	Recommended
RAM	1 GB	2-4 GB (with browser: 4 GB)
CPU	1 vCPU	2 vCPU
Disk	2 GB (base image)	5-10 GB (with browser + tools)
Node.js	22.12.0+	Latest 22.x LTS
Docker	20.10+	Latest stable

The build process sets `NODE_OPTIONS=--max-old-space-size=2048` to reduce OOM on small VMs. The `OPENCLAW_INSTALL_BROWSER=1` build arg adds ~300MB for Chromium.

1.8.8 7.8 Single-Command VPS Deployment

Based on `docker-setup.sh` and `docs/install/docker.md`:

```
# On a fresh VPS with Docker installed:
git clone https://github.com/openclaw/openclaw.git
cd openclaw
bash docker-setup.sh
```

This will: 1. Generate an auth token 2. Create the config directory 3. Build the Docker image 4. Run interactive onboarding 5. Start the gateway

For non-interactive deployment:

```
# Pre-configure
mkdir -p ~/.openclaw
cat > ~/.openclaw/openclaw.json << 'EOF'
{
  "models": {
    "providers": {
      "anthropic": { "apiKey": "${ANTHROPIC_API_KEY}" }
    }
  }
}
EOF

# Set env vars
export ANTHROPIC_API_KEY=sk-ant-...
export OPENCLAW_GATEWAY_TOKEN=$(openssl rand -hex 32)
export OPENCLAW_GATEWAY_BIND=lan

# Build and start
docker build -t openclaw:local .
docker compose up -d openclaw-gateway
```

1.8.9 7.9 Daemon/Service Mode

For non-Docker deployments, OpenClaw can be installed as a system service:

Platform	Service Type	Install Method
macOS	LaunchAgent	`launchctl` via `src/daemon/launchd.ts`
Linux	systemd user unit	`systemctl --user` via `src/daemon/systemd.ts`
Windows	Scheduled Task	`schtasks.exe` via `src/daemon/schtasks.ts`

CLI: `openclaw daemon install/uninstall/start/stop/restart/status`

1.9 8. API Surface

1.9.1 8.1 HTTP Endpoints

All served on port **18789** (same as WebSocket):

Endpoint	Method	Auth	Purpose
`POST /tools/invoke`	POST	Bearer token	Invoke any agent tool directly
`POST /v1/chat/completions`	POST	Bearer token	OpenAI-compatible chat API
`POST /v1/responses`	POST	Bearer token	OpenAI Responses API compatibl
`GET /__openclaw__/canvas/*`	GET	Bearer or node WS	Canvas host
`GET /__openclaw__/a2ui/*`	GET	Bearer or local	A2UI host
`ALL /api/channels/*`	ALL	Bearer token	Plugin channel HTTP routes
`POST /hooks/*`	POST	Hook token	Webhook receivers
`ALL /slack/*`	ALL	Slack signing secret	Slack HTTP events
`GET /` (Control UI)	GET	None (assets) / Device auth (WS)	Built React web UI

1.9.2 8.2 Tool Invocation API

`POST /tools/invoke` (from `src/gateway/tools-invoke-http.ts`)

```
POST /tools/invoke
Authorization: Bearer <gateway_token>
Content-Type: application/json

{
  "tool": "exec",
  "action": "run",
  "args": { "command": "echo hello" },
  "sessionKey": "agent:main:direct:api:user1",
  "dryRun": false
}
```

Response:

```
{ "ok": true, "result": { "content": [{ "type": "text", "text": "hello\n" }] } }
```

- Max body: 2 MB
- Applies full tool policy pipeline
- Hard default deny list: `sessions_spawn`, `sessions_send`, `gateway`, `whatsapp_login`
- Status codes: 200, 400, 401, 404, 405, 429, 500

1.9.3 8.3 OpenAI-Compatible Chat API

`POST /v1/chat/completions` (from `src/gateway/openai-http.ts`)

```
POST /v1/chat/completions
Authorization: Bearer <gateway_token>
Content-Type: application/json

{
  "model": "openclaw:main",
  "messages": [
    { "role": "user", "content": "What's on my calendar today?" }
  ],
  "stream": true
}
```

- Agent routing via `model` field (`openclaw:<agentId>`) or `x-openclaw-agent-id` header
- Session key via `x-openclaw-session-key` header
- SSE streaming: `Content-Type: text/event-stream`, ends with `data: [DONE]`
- Non-streaming returns standard OpenAI response format

This is the primary API for integrating with OpenClaw programmatically. Any OpenAI-compatible client library can be pointed at the gateway.

1.9.4 8.4 WebSocket API (Gateway Protocol)

All WebSocket methods on port 18789 via the `wss` server. Key method groups:

Core

Method	Purpose
`health`	Health check
`status`	System status
`logs.tail`	Stream gateway logs

Chat & Agent

Method	Purpose
`send`	Send message to agent
`agent`	Run agent task
`agent.wait`	Wait for agent completion
`chat.send`	Send chat message
`chat.history`	Get chat history

Method	Purpose
`chat.abort`	Abort active run

Configuration

Method	Purpose
`config.get`	Read config
`config.set`	Set config key
`config.apply`	Apply full config
`config.patch`	Patch config
`config.schema`	Get config schema

Sessions

Method	Purpose
`sessions.list`	List sessions
`sessions.preview`	Preview session
`sessions.patch`	Update session
`sessions.reset`	Reset session
`sessions.delete`	Delete session
`sessions.compact`	Compact session

Agents

Method	Purpose
`agents.list`	List agents
`agents.create`	Create agent
`agents.update`	Update agent
`agents.delete`	Delete agent
`agents.files.*`	Manage agent files

Management

Method	Purpose
`models.list`	List available models
`tools.catalog`	List available tools

Method	Purpose
`skills.status`	Skill status
`skills.install`	Install skill
`cron.list/add/update/remove/run`	Manage cron jobs
`node.pair.*`	Node pairing
`device.pair.*`	Device pairing
`exec.approvals.*`	Exec approval management
`browser.request`	Browser control
`tts.*`	TTS management
`usage.status/cost`	Usage tracking
`wizard.*`	Onboarding wizard
`update.run`	Trigger update

1.9.5 8.5 Authentication

Auth modes (from `src/gateway/auth.ts`):

Mode	How It Works	Config
`"token"`	Bearer token in `Authorization` header. Constant-time comparison.	`gateway.auth.token` or `OPENCLAW_GATEWAY_TOKEN` env
`"password"`	Password-based.	`gateway.auth.password` or `OPENCLAW_GATEWAY_PASSWORD` env
`"trusted-proxy"`	Delegates to reverse proxy via header (e.g., `x-forwarded-user`). Validates request from `gateway.trustedProxies` IPs.	`gateway.auth.trustedProxy` config
`"none"`	No auth (dangerous)	Explicit config

Tailscale auth: When `allowTailscale: true` and `tailscaleMode: "serve"`, validates via `tailscale whois` API.

Rate limiting (from `src/gateway/auth-rate-limit.ts`): - Configurable: `gateway.auth.rateLimit.{m` `windowMs, lockoutMs}` - Hook auth hard limit: 20 failures per 60s - Returns `429 Too Many Requests` with `Retry-After` header

Device auth: Each gateway host generates an Ed25519 keypair at `~/.openclaw/identity/device.` The Control UI browser must be approved as a known device.

1.9.6 8.6 Gateway Documentation

Additional API docs in the repo: - `docs/gateway/authentication.md` — OAuth/API key setup, key rotation - `docs/gateway/network-model.md` — Network architecture - `docs/gateway/tools-inv` — `/tools/invoke` endpoint - `docs/gateway/openai-http-api.md` — `/v1/chat/completions`

endpoint - `docs/gateway/openresponses-http-api.md` — `/v1/responses` endpoint - `docs/gateway/trusted-` — Reverse proxy auth - `docs/gateway/sandboxing.md` — Sandbox architecture - `docs/gateway/tailscale` — Tailscale integration

---

1.10 9. Security Model

1.10.1 9.1 Trust Model

From `SECURITY.md`: OpenClaw operates as a **"personal assistant"** — one trusted operator per gateway, NOT multi-tenant.

Key implications: - Authenticated gateway callers are treated as **fully trusted operators** - Session identifiers (`sessionKey`) are routing controls, NOT per-user auth boundaries - The gateway does not implement per-user authorization - `agents.defaults.sandbox.mode` defaults to `"off"`

This means: For LetsBe's multi-tenant use case, each customer MUST get their own isolated gateway instance. You cannot serve multiple customers from a single gateway.

1.10.2 9.2 Security Audit System

Location: `src/security/audit.ts`

`runSecurityAudit()` checks for:

Check	Severity	What It Detects
`gateway.bind_no_auth`	CRITICAL	Non-loopback bind without auth token
`gateway.loopback_no_auth`	CRITICAL	Loopback bind without auth (proxy risk)
`gateway.control_ui.allowed_origins_required`	CRITICAL	Non-loopback Control UI without CORS origins
`gateway.token_too_short`	WARN	Token < 24 chars
`gateway.auth_no_rate_limit`	WARN	No rate limiting on non-loopback
`gateway.tailscale_funnel`	CRITICAL	Public Tailscale Funnel exposure
`gateway.tools_invoke_http.dangerous_allow`	CRITICAL/WARN	Dangerous tools re-enabled over HTTP
`gateway.trusted_proxy_auth`	CRITICAL	Trusted-proxy auth mode issues
`fs.state_dir.perms_world_writable`	CRITICAL	`~/.openclaw` world-writable
`fs.config.perms_world_readable`	CRITICAL	`openclaw.json` world-readable
`tools.exec.safe_bins_interpreter_unprofiled`	WARN	Shell interpreters in safeBins
`browser.control_no_auth`	CRITICAL	Browser control without auth
`logging.redact_off`	WARN	Logging redaction disabled
`tools.elevated.allowFrom.*.wildcard`	CRITICAL	Wildcard in elevated exec allowlist
`discovery.mdns_full_mode`	WARN/CRITICAL	mDNS leaking host metadata

Run via: `openclaw security audit`

1.10.3 9.3 Secrets Management

Secret Type	Storage	Protection
Gateway token	Config or env var	File permissions check
LLM API keys	Config, env, or auth profiles	env var interpolation (`${VAR}`)
OAuth tokens	`~/.openclaw/credentials/`	File permissions
Channel tokens	Config or env vars	env var interpolation
Device keypair	`~/.openclaw/identity/device.json`	File permissions

Credential resolution precedence (from `src/gateway/credentials.ts`): - Configurable as `env-first` or `config-first` - Legacy env vars supported: `CLAWDBOT_GATEWAY_TOKEN`, `CLAWDBOT_GATEWAY_PASSWORD`

1.10.4 9.4 Sandboxing for Tool Execution

When `agents.defaults.sandbox.mode` is enabled:

Container hardening defaults:

```
{
  "readOnlyRoot": true,
  "tmpfs": ["/tmp", "/var/tmp", "/run"],
  "network": "none",
  "user": "1000:1000",
  "capDrop": ["ALL"],
  "pidsLimit": 256,
  "memory": "1g",
  "memorySwap": "2g",
  "cpus": 1,
  "ulimits": { "nofile": { "soft": 1024, "hard": 2048 }, "nproc": 256 }
}
```

Network isolation: `network: "none"` by default. `host` and `container:<id>` modes are blocked.

Break-glass: `dangerouslyAllowContainerNamespaceJoin: true` can be set for edge cases.

1.10.5 9.5 Attack Surfaces

For LetsBe deployments, key attack surfaces to be aware of:

Surface	Risk	Mitigation
Gateway HTTP/WS API	Unauthorized access if token leaked	Strong token + rate limiting + bind to loopback behind reverse proxy
LLM prompt injection	Agent executing malicious tool calls	Safety Wrapper (our integration), tool policy, sandbox
Tool execution	Arbitrary command execution	Sandbox mode, tool allowlist/denylist
Config file	API keys in plaintext	File permissions, env var interpolation
Browser automation	SSRF, data exfiltration	Navigation guard, SSRF policy
Channel tokens	Messaging channel compromise	Env vars, not plaintext config
Memory/RAG	Data leakage across sessions	Single-operator model (one gateway per customer)
Webhook endpoints	Unauthorized hook triggers	Hook tokens, rate limiting

1.10.6 9.6 Where to Insert a Proxy Layer

Based on the architecture, there are four insertion points for our Safety Wrapper:

1. **Plugin `before_tool_call` hook** (RECOMMENDED) — intercepts every tool call before execution
2. **HTTP API proxy** — reverse proxy in front of `/tools/invoke` and `/v1/chat/completions`
3. **Custom tool wrappers** — replace built-in tools with wrapped versions
4. **`message_sending` hook** — filter outbound messages before delivery

The `before_tool_call` hook is the cleanest integration point because it: - Runs inside the same process (low latency) - Has access to full context (session, agent, config) - Can block or modify any tool call - Doesn't require forking the codebase - Is the officially supported extension mechanism

---

1.11 10. Integration Points for LetsBe Safety Wrapper

1.11.1 10.1 Primary Interception Point: `before_tool_call` Hook

The `before_tool_call` typed plugin hook is the **single best integration point** for the Safety Wrapper. It fires before every tool call in the agent execution loop.

Hook signature:

```
api.on("before_tool_call", async (event, ctx) => {
  // event contains:
```

```
  //   toolName: string        - e.g., "exec", "web_search", "message"
  //   params: Record<string, unknown>  - the tool call arguments
  //   sessionKey: string      - identifies the conversation
  //   agentId: string         - which agent is running
  //
  // Return options:
  //   { }                     - allow the call
  //   { params: modified }    - allow with modified parameters
  //   { block: true, blockReason: "..." }  - block the call entirely

  return {};
}, { priority: 1000 });  // higher priority = runs first
```

What it can intercept: - Shell command execution (`exec` tool) — inspect the command string - Web searches (`web_search`) — inspect query terms - Web fetches (`web_fetch`) — inspect target URLs - Browser automation (`browser`) — inspect navigation targets and actions - Message sending (`message`) — inspect outbound messages - File operations — inspect paths - Memory operations — inspect search queries - Subagent spawning (`sessions_spawn`) — control delegation

1.11.2 10.2 Secondary Interception Points

Hook	Purpose for Safety Wrapper
`after_tool_call`	Audit logging — record every tool call with result and duration
`message_sending`	Content filtering — modify or block outbound messages
`before_message_write`	PII scrubbing — filter data before it's persisted to JSONL
`tool_result_persist`	Redaction — scrub sensitive data from tool results before persistence
`before_prompt_build`	Inject safety instructions into the system prompt
`subagent_spawning`	Control/limit subagent creation
`llm_input` / `llm_output`	Observe all LLM traffic for monitoring

1.11.3 10.3 Can We Use the Extension System? YES

The extension system is the recommended approach. A Safety Wrapper extension would:

1. Live in `extensions/letsbe-safety-wrapper/` (or be installed globally at `~/.openclaw/extensions/`)
2. Register `before_tool_call` with highest priority (runs first)
3. Register `after_tool_call` for audit logging
4. Register `message_sending` for content filtering
5. Optionally register an HTTP route for external policy API
6. Optionally register a background service for heartbeat/telemetry

1.11.4 10.4 Minimal Safety Wrapper Extension

```
// extensions/letsbe-safety-wrapper/index.ts
import type { OpenClawPluginApi } from "openclaw/plugin-sdk";

interface SafetyPolicy {
  blockedCommands: RegExp[];
  blockedUrls: RegExp[];
  blockedTools: string[];
  maxExecTimeoutMs: number;
  auditEndpoint?: string;
}

function loadPolicy(config: Record<string, unknown>): SafetyPolicy {
  return {
    blockedCommands: (config.blockedCommandPatterns as string[] || []).map(p => new RegExp(p, "
    blockedUrls: (config.blockedUrlPatterns as string[] || []).map(p => new RegExp(p, "i")),
    blockedTools: config.blockedTools as string[] || [],
    maxExecTimeoutMs: (config.maxExecTimeoutMs as number) || 30000,
    auditEndpoint: config.auditEndpoint as string | undefined,
  };
}

const safetyWrapper = {
  id: "letsbe-safety-wrapper",
  name: "LetsBe Safety Wrapper",
  version: "1.0.0",
  configSchema: {
    type: "object",
    properties: {
      blockedCommandPatterns: { type: "array", items: { type: "string" } },
      blockedUrlPatterns: { type: "array", items: { type: "string" } },
      blockedTools: { type: "array", items: { type: "string" } },
      maxExecTimeoutMs: { type: "number" },
      auditEndpoint: { type: "string" },
      piiRedactionEnabled: { type: "boolean" },
    },
  },

  register(api: OpenClawPluginApi) {
    const policy = loadPolicy(api.pluginConfig || {});

    // INTERCEPT: Block dangerous tool calls
    api.on("before_tool_call", async (event) => {
      const { toolName, params } = event;
```

```
  // Block entire tools
  if (policy.blockedTools.includes(toolName)) {
    return { block: true, blockReason: `Tool '${toolName}' is disabled by safety policy` }
  }

  // Block dangerous shell commands
  if (toolName === "exec" && typeof params.command === "string") {
    for (const pattern of policy.blockedCommands) {
      if (pattern.test(params.command)) {
        return { block: true, blockReason: `Command blocked by safety policy: ${pattern}` }
      }
    }
  }

  // Block dangerous URLs
  if ((toolName === "web_fetch" || toolName === "browser") && typeof params.url === "string") {
    for (const pattern of policy.blockedUrls) {
      if (pattern.test(params.url)) {
        return { block: true, blockReason: `URL blocked by safety policy` };
      }
    }
  }

  return {};  // allow
}, { priority: 10000 });  // highest priority - runs before all other hooks

// AUDIT: Log every tool call
api.on("after_tool_call", async (event) => {
  if (policy.auditEndpoint) {
    fetch(policy.auditEndpoint, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        timestamp: new Date().toISOString(),
        toolName: event.toolName,
        params: event.params,
        durationMs: event.durationMs,
        error: event.error,
      }),
    }).catch(() => {}); // fire-and-forget
  }
});
```

```
    // FILTER: Scrub outbound messages
    api.on("message_sending", async (event) => {
      // Add content filtering logic here
      return {};
    });

    api.logger.info("LetsBe Safety Wrapper loaded");
  },
};

export default safetyWrapper;
```

Manifest (`openclaw.plugin.json`):

```
{
  "id": "letsbe-safety-wrapper",
  "name": "LetsBe Safety Wrapper",
  "version": "1.0.0",
  "configSchema": {
    "type": "object",
    "properties": {
      "blockedCommandPatterns": { "type": "array", "items": { "type": "string" } },
      "blockedUrlPatterns": { "type": "array", "items": { "type": "string" } },
      "blockedTools": { "type": "array", "items": { "type": "string" } },
      "maxExecTimeoutMs": { "type": "number", "default": 30000 },
      "auditEndpoint": { "type": "string" },
      "piiRedactionEnabled": { "type": "boolean", "default": true }
    }
  }
}
```

1.11.5 10.5 What CANNOT Be Intercepted

Action	Why Not	Workaround
LLM token streaming	No hook exists for per-token filtering	Use `llm_output` for post-hoc analysis
Config file reads	No hook	File permissions
Plugin loading itself	Plugins load before hooks register	Control via `plugins.allow`/`plugins.deny`
Direct file I/O by extensions	Extensions run in-process	Code review of extensions
Gateway WS method calls	Some methods bypass tool pipeline	Restrict via auth + `gateway.tools_invoke_http`

1.11.6 10.6 Recommendation: Extension Approach

Approach	Pros	Cons
Extension (RECOMMENDED)	Clean API, officially supported, no fork needed, runs in-process (fast), access to full context	Must trust OpenClaw's hook execution order
Reverse proxy	Language-agnostic, external to OpenClaw	Higher latency, loses session context, can't intercept internal tool calls
Fork	Full control	Maintenance burden, merge conflicts on updates

Recommendation: Build the Safety Wrapper as an OpenClaw extension. Install it at `~/.openclaw/extensions/letsbe-safety-wrapper/` in the base Docker image. Configure it per-customer via `plugins.entries.letsbe-safety-wrapper.config` in `openclaw.json`.

1.12 11. Provisioning Blueprint

1.12.1 11.1 Step-by-Step Provisioning Sequence

Step	Action	Est. Time	Pre-bakeable?
1	Provision VPS (2 vCPU, 4GB RAM, 20GB SSD)	30-60s	N/A
2	Apply base image with Docker + OpenClaw pre-built	0s (snapshot)	**YES**
3	Create customer config directory (`~/.openclaw/`)	1s	YES (in image)
4	Write customer-specific `openclaw.json`	1s	Template + inject
5	Write customer-specific `.env`	1s	Template + inject

Step	Action	Est. Time	Pre-bakeable?
6	Install Safety Wrapper extension	0s (in base image)	**YES**
7	Install business tool binaries (`gog`, `himalaya`, etc.)	0s (in base image)	**YES**
8	Configure Google OAuth (customer-specific)	Manual or API	No
9	Configure email (customer IMAP/SMTP)	1s (write config)	Template
10	Generate gateway auth token	1s	At provision time
11	Start gateway container	5-10s	No
12	Run health check	2-3s	No
Total		**~45-75s** (excl. OAuth)	

1.12.2 11.2 What to Pre-bake into Base Image

Build a custom Docker image extending OpenClaw:

```
FROM openclaw:local

# Pre-install business tool binaries
RUN apt-get update && apt-get install -y curl && \
    # Install himalaya
    curl -L https://github.com/pimalaya/himalaya/releases/latest/download/himalaya-linux-x86_64
      -o /usr/local/bin/himalaya && chmod +x /usr/local/bin/himalaya && \
    # Install gog
    curl -L https://github.com/user/gog/releases/latest/download/gog-linux-amd64 \
      -o /usr/local/bin/gog && chmod +x /usr/local/bin/gog

# Pre-install Safety Wrapper extension
COPY extensions/letsbe-safety-wrapper /home/node/.openclaw/extensions/letsbe-safety-wrapper/

# Pre-install skill files (if customized)
# COPY skills/ /app/skills/
```

```
# Set env defaults
ENV OPENCLAW_GATEWAY_BIND=lan
ENV OPENCLAW_SKIP_CHANNELS=1
```

1.12.3 11.3 Per-Customer Config Template

```
// ~/.openclaw/openclaw.json - TEMPLATE
{
  "gateway": {
    "auth": {
      "mode": "token",
      "token": "${OPENCLAW_GATEWAY_TOKEN}"
    },
    "bind": "lan",
    "port": 18789
  },
  "models": {
    "providers": {
      "anthropic": {
        "apiKey": "${ANTHROPIC_API_KEY}"
      }
    }
  },
  "agents": {
    "defaults": {
      "model": "claude-sonnet-4-6",
      "provider": "anthropic",
      "sandbox": {
        "mode": "all",
        "scope": "session"
      }
    },
    "list": [
      {
        "id": "main",
        "default": true,
        "systemPrompt": "You are a business assistant for {{COMPANY_NAME}}. Follow all safety p
      }
    ]
  },
  "tools": {
    "allowlist": ["exec", "web_search", "web_fetch", "memory_search", "memory_get", "browser"]
  },
  "plugins": {
```

```
    "enabled": true,
    "allow": ["memory-core", "letsbe-safety-wrapper", "llm-task"],
    "slots": { "memory": "memory-core" },
    "entries": {
      "letsbe-safety-wrapper": {
        "enabled": true,
        "config": {
          "blockedCommandPatterns": ["rm\\s+-rf", "shutdown", "reboot", "mkfs", "dd\\s+if="],
          "blockedUrlPatterns": [".*\\.onion", "localhost", "127\\.0\\.0\\.1", "169\\.254\\."]
          "blockedTools": ["sessions_spawn", "gateway"],
          "maxExecTimeoutMs": 30000,
          "auditEndpoint": "https://hub.letsbe.ai/api/audit/{{CUSTOMER_ID}}",
          "piiRedactionEnabled": true
        }
      }
    }
  },
  "skills": {
    "bundled": {
      "allowlist": ["gog", "himalaya", "weather", "summarize"]
    }
  },
  "logging": {
    "redactSensitive": true
  }
}
```

Variables to inject per-customer: `OPENCLAW_GATEWAY_TOKEN`, `ANTHROPIC_API_KEY`, `COMPANY_NAME`, `CUSTOMER_ID`.

1.12.4 11.4 Health Check Sequence

```
#!/bin/bash
# health-check.sh - run after provisioning

TOKEN="${OPENCLAW_GATEWAY_TOKEN}"
HOST="localhost:18789"

# 1. Check gateway is listening
curl -sf "http://$HOST/health" -H "Authorization: Bearer $TOKEN" || exit 1

# 2. Check via CLI
openclaw health --token "$TOKEN" || exit 1

# 3. Send test message and verify response
```

```
curl -sf "http://$HOST/v1/chat/completions" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"model":"openclaw:main","messages":[{"role":"user","content":"ping"}]}' \
  | jq -e '.choices[0].message.content' || exit 1

# 4. Run security audit
openclaw security audit || echo "WARN: Security audit findings"

echo "Health check passed"
```

1.12.5 11.5 Minimum Viable OpenClaw Setup

Fewest containers: 1 (gateway only, sandbox mode off) **Simplest config:** Anthropic API key + gateway token **No channels needed** — use HTTP API (`/v1/chat/completions`) exclusively **No browser needed** — skip `OPENCLAW_INSTALL_BROWSER` **No sandbox needed** — if Safety Wrapper provides sufficient controls

```
# Absolute minimum:
docker run -d \
  -e ANTHROPIC_API_KEY=sk-ant-... \
  -e OPENCLAW_GATEWAY_TOKEN=$(openssl rand -hex 32) \
  -p 18789:18789 \
  openclaw:local \
  node openclaw.mjs gateway --bind lan --port 18789 --allow-unconfigured
```

1.13 12. Risks, Limitations & Open Questions

1.13.1 12.1 Maturity Assessment

Component	Maturity	Notes
Core gateway	**High**	Actively maintained, well-structured
Plugin SDK	**Medium-High**	Well-defined API, typed hooks, but limited documentation
Config system	**High**	Comprehensive Zod schema, JSON5 with env var interpolation

Component	Maturity	Notes
Memory backend	**Medium**	Experimental `node:sqlite` module; `sqlite-vec` for vectors
Sandbox system	**Medium**	Docker-based, comprehensive hardening defaults, but complex
Browser automation	**Medium**	Playwright-based, works but adds significant complexity
Channel plugins	**Medium-High**	Many channels, but quality varies by channel
Skills system	**Medium**	Markdown-only, no structured API — agents execute CLI commands via bash

1.13.2 12.2 Scaling Limitations

Limitation	Impact	Mitigation
Single-operator trust model	Cannot serve multiple customers from one gateway	One VPS per customer (our plan)
Flat-file storage	No concurrent write safety for sessions	Session write locks exist but limited
In-process tools	Tools run in gateway Node.js process	Enable sandbox mode for isolation
No horizontal scaling	Single gateway process per instance	One instance per customer (our plan)
Memory backend uses node:sqlite	Experimental Node.js API, may change	Pin Node.js version
No message queue	Direct in-process event dispatch	Acceptable for single-tenant

1.13.3 12.3 Missing Features We'd Need to Build

Feature	Why We Need It	Effort
Multi-tenant auth	Isolate customers on shared infra (future)	High — architectural change
Usage metering/billing	Track per-customer LLM costs	Medium — hook into `after_tool_call` + `llm_output`
Customer onboarding API	Automated provisioning without interactive wizard	Medium — script the config writing
Centralized logging	Aggregate logs from all customer instances	Low — forward logs to central service
Health monitoring	Monitor all customer instances	Low — health endpoint polling
Auto-update mechanism	Update OpenClaw across fleet	Medium — rolling Docker image updates
Backup/restore	Customer data portability	Low — backup `~/.openclaw/` directory

1.13.4 12.4 Licensing

MIT License — fully permissive for commercial use. No concerns for our use case. We can modify, distribute, and sublicense. Copyright notice must be preserved.

1.13.5 12.5 Version Pinning Strategy

- OpenClaw uses **calendar versioning**: `YYYY.M.D` (e.g., `2026.2.26`)
- Release channels: `stable` (tagged), `beta` (prerelease), `dev` (main branch)
- **Recommendation:** Pin to specific stable releases in our Docker image. Test beta releases in staging before rolling out.
- Dependencies with `pnpm.patchedDependencies` must use exact versions (no ^/~)
- Node.js engine requirement: `>=22.12.0`

1.13.6 12.6 Open Questions

1. `node:sqlite` **stability** — OpenClaw uses the experimental `node:sqlite` module. What's the Node.js team's timeline for stabilization? Should we pin a specific Node.js patch version?

2. **Gateway memory usage under load** — With 30 containerized tools + memory indexing + browser automation, what's the actual RAM footprint per customer? Need load testing.

3. **OAuth token refresh** — The `gog` CLI handles its own OAuth token storage. How does token refresh work when running headless in a container? Does it require periodic re-auth?

4. **Himalaya auth in containers** — The `himalaya` config uses `auth.cmd` for password retrieval (e.g., `pass show email/imap`). In a container, how do we securely provide IMAP/SMTP credentials?
5. **Extension stability across updates** — When OpenClaw updates, do plugin SDK interfaces maintain backward compatibility? Is there a versioned plugin API?
6. **Session JSONL file growth** — Sessions are append-only JSONL files. For long-running business agents, these could grow large. What's the compaction behavior? Is there auto-archival?
7. **MCP integration via mcporter** — OpenClaw uses `mcporter` for MCP server integration. Should our 30 containerized tools be exposed as MCP servers via mcporter, or as native OpenClaw skills/extensions?
8. **Browser sandbox networking** — The sandbox has `network: "none"` by default. Business tools (email, calendar, web search) need network access. What's the recommended network policy for business use?
9. **Config hot reload scope** — The gateway watches `openclaw.json` for hot reload. Which config changes take effect without restart vs. requiring restart?
10. **Concurrent agent runs** — If the same customer sends multiple messages rapidly, how does OpenClaw handle concurrent agent runs for the same session? Is there queuing?

1.14 Appendix A: Key File Reference

What	Path
Entry point	`openclaw.mjs` → `src/entry.ts`
CLI main	`src/cli/run-main.ts`
Gateway server	`src/gateway/server.impl.ts`
HTTP endpoints	`src/gateway/server-http.ts`
Auth	`src/gateway/auth.ts`
Config schema	`src/config/zod-schema.ts`
Config loader	`src/config/io.ts`
Agent defaults	`src/agents/defaults.ts`
Agent runner	`src/agents/pi-embedded-runner/`
Model auth	`src/agents/model-auth.ts`
Provider catalog	`src/agents/models-config.providers.ts`
Tool policy	`src/agents/tool-policy.ts`

What	Path
Plugin types	`src/plugins/types.ts`
Plugin loader	`src/plugins/loader.ts`
Plugin SDK	`src/plugin-sdk/index.ts`
Hook system	`src/hooks/internal-hooks.ts`
Routing	`src/routing/resolve-route.ts`
Memory backend	`src/memory/`
Security audit	`src/security/audit.ts`
Browser tools	`src/browser/`
Web search	`src/agents/tools/web-search.ts`
Sandbox supervisor	`src/process/supervisor/supervisor.ts`
Docker setup	`docker-setup.sh`
Main Dockerfile	`Dockerfile`
Sandbox Dockerfile	`Dockerfile.sandbox`
Docker compose	`docker-compose.yml`
Security policy	`SECURITY.md`
Env example	`.env.example`
Google skill	`skills/gog/SKILL.md`
Email skill	`skills/himalaya/SKILL.md`
OpenClawKit (Swift)	`apps/shared/OpenClawKit/`

End of OpenClaw Architecture Analysis Document generated: 2026-02-26 OpenClaw version analyzed: 2026.2.26