LetsBe Biz — Technical Architecture

System Architecture and Infrastructure

Version:	v1.2
Date:	February 26, 2026
Company:	LetsBe Solutions LLC
Contact:	matt@letsbe.solutions

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

Contents

1. LetsBe Biz — Technical Architecture Document

Date: February 26, 2026 **Authors:** Matt (Founder), Claude (Architecture) **Status:** Living Document — Version 1.2 **Companion to:** Foundation Document v1.0, Product Vision v1.0, Pricing Model v2.2, Financial Projections v1.2

1.1 1. Architecture Philosophy

LetsBe Biz is a privacy-first AI workforce platform for SMBs. Each customer gets their own isolated VPS running an AI-powered team that manages their business tools. The architecture follows four principles:

1. **Secrets never leave the server.** All credential redaction happens locally before any data reaches an LLM. The AI reasons about infrastructure without seeing passwords, keys, or tokens. This is enforced at the transport layer — not by trusting the AI to behave.
2. **The AI acts autonomously within guardrails.** Non-destructive operations execute immediately. Destructive operations require human approval. The boundary is enforced by code the AI cannot modify, with three configurable autonomy tiers that let customers control the trust level.
3. **OpenClaw stays vanilla.** The upstream AI agent runtime is treated as a dependency, not a fork. All LetsBe-specific logic (secrets redaction, command gating, Hub communication, tool adapters) lives in a separate wrapper layer that can pull upstream updates without merge conflicts.
4. **Four layers of defense.** Security is not one wall — it's four independent layers (Sandbox → Tool Policy → Command Gating → Secrets Redaction), each enforced separately, each unable to be bypassed by the layers above it.

1.2 2. System Overview

The platform has two operational domains: the **Central Platform** (Hub + Provisioner) and the **Tenant Server** (OpenClaw + Safety Wrapper + Tool Stacks). Each customer gets their own isolated VPS (Decision #18: one customer = one VPS, permanently).

1.2.1 2.1 Component Summary

Component	Language	Role	Status
Hub	TypeScript / Next.js 16	Central control plane — admin UI, customer portal, billing, provisioning, monitoring	KEEP + RETOOL
Provisioner (was ansible-runner)	Bash	One-shot server provisioning — deploys all components + tool stacks to fresh VPS	KEEP + RENAME
OpenClaw	Node.js 22+ (upstream)	On-server AI agent runtime — manages agent personas, conversa-tions, tool use	NEW (core product, upstream dependency)
Safety Wrapper	Node.js (Decision #11)	Proxy layer — secrets redaction, command classification, execution gating, Hub communication	NEW BUILD (core IP)
~~**MCP Browser**~~	~~Python / FastAPI~~	~~Browser sidecar~~	**DEPRECATED → Decision #14** (replaced by OpenClaw native browser tool)
Orchestrator	Python / FastAPI	Per-tenant task queue + agent management	DEPRECATE — absorbed by OpenClaw + Safety Wrapper
Sysadmin Agent	Python / asyncio	Infrastructure task executor	DEPRECATE — capabilities become OpenClaw tools via Safety Wrapper

1.2.2 2.2 Deprecation Rationale

Orchestrator (letsbe-orchestrator, ~7,500 LOC, ~50 files): Its three responsibilities — task queue (atomic claim via SELECT FOR UPDATE), agent management (registration, heartbeat), and Hub telemetry (windowed SQL aggregates with backoff) — are absorbed by OpenClaw (agent management, task dispatch via native cron/hooks) and the Safety Wrapper (Hub communication, telemetry). Running a separate FastAPI + PostgreSQL service adds two containers per tenant with ~256MB+ RAM overhead. OpenClaw's native agent runtime handles agent lifecycle natively, and its command queue provides lane-aware FIFO with concurrency caps and debouncing — more capable than our custom task queue.

Sysadmin Agent (letsbe-sysadmin-agent, ~7,600 LOC, 30 files): Its 10 executor types (Shell, Docker, File Read/Write, ENV Read/Update, Composite, Nextcloud, Playwright) are ported as tools accessible through the Safety Wrapper. The agent's security patterns (allowlists, path traversal prevention, metacharacter blocking) become pre-execution hooks in the Safety Wrapper. Its heartbeat/circuit breaker logic moves into the Safety Wrapper's Hub client. Its 8 Playwright initial-setup scenarios (Cal.com, Chatwoot, Keycloak, n8n, Nextcloud, Stalwart Mail, Umami, Uptime Kuma) migrate to run via OpenClaw's native browser tool (Playwright + CDP) during provisioning.

Why not keep sysadmin as a separate security boundary? Because process separation doesn't add meaningful security here. The sysadmin executes whatever structured payload it receives — if the AI sends a valid-looking destructive command, the sysadmin runs it. The real security is in *what rules are enforced before execution*, not in which process executes. The Safety Wrapper enforces those rules in a single, auditable layer.

MCP Browser (letsbe-mcp-browser, ~1,246 LOC): Replaced by OpenClaw's native browser tool which is more capable (full CDP + Playwright with accessibility tree navigation, file operations, state management, trace recording) and saves ~256MB RAM per tenant. The MCP Browser was session-based with domain allowlisting and basic actions (navigate, click, type, screenshot, snapshot). OpenClaw's browser does all of this plus drag, select, scroll, fill, press, evaluate JS, cookie CRUD, localStorage, offline mode, device emulation, and more. The sysadmin agent's `MCP_SERVICE_URL` config was declared but never wired to executor code — the integration was incomplete.

1.3 3. Tenant Server Architecture

Each customer VPS runs two processes:

1.3.1 3.1 OpenClaw (Upstream AI Runtime)

Treated as a black box dependency. No modifications to upstream code.

OpenClaw is a single **Gateway process** (Node.js 22+) that owns all state — agents, sessions, tools, channels, and automation. There is no separate database server; session state lives in JSON files on disk (`sessions.json` for metadata, `*.jsonl` for tran-

scripts). The Gateway multiplexes WebSocket and HTTP on a single port (default 18789).

Configuration uses JSON5 format at `~/.openclaw/openclaw.json` with environment variable substitution (`${VAR_NAME}`) and `$include` for splitting files. All fields are optional with safe defaults.

OpenClaw is configured to: - Route all LLM calls through `localhost` (the Safety Wrapper), not directly to OpenRouter - Register available tools from the Safety Wrapper's tool server via the plugin API - Load agent configurations (SOUL.md, TOOLS.md) from the local filesystem - Store conversation history and agent state locally (JSON files on disk) - Use token-based authentication with the Safety Wrapper as the sole authenticated client

OpenClaw manages natively: - Agent personas (IT Admin, Marketing, Secretary, Sales, Custom) - Conversation routing (which agent handles which request) - Multi-step reasoning and planning - Tool call decisions (which tool to invoke, with what parameters) - Prompt caching (SOUL.md as cacheable prompt prefix, `cacheRetention: "long"`) - Session management with compaction (auto-summarizes when context fills) - Scheduled tasks (cron jobs per agent) - Event-driven hooks (message lifecycle, tool results, bootstrap) - Agent-to-agent communication (disabled by default, enabled per agent) - Browser automation (Playwright + CDP, managed Chromium profile) - Command queue (lane-aware FIFO with concurrency caps and debouncing) - Sub-agents (independent session processes with isolated token usage) - Model failover (auth profile rotation before model fallback) - Health checks and logging

Key defaults we configure:

Setting	Default	LetsBe Config	Rationale
`model.primary`	`anthropic/claude-opus-4-6`	Points to Safety Wrapper proxy endpoint	All LLM traffic routes through redaction layer
`timeoutSeconds`	600 (10 min)	Keep default	Reasonable for most tasks
`contextTokens`	200,000	Keep default	Claude context window
`maxConcurrent`	1	Keep default	One agent turn at a time prevents race conditions
`sandbox.mode`	`"off"`	`"off"`	Safety Wrapper handles sandboxing instead
`cacheRetention`	`"short"` (5 min)	`"long"` (1 hour)	Maximize cache savings on SOUL.md
`heartbeat.every`	none	`"55m"`	Keep-warm to reduce repeated cache writes

Setting	Default	LetsBe Config	Rationale
`security.elevated.enable`	`true`	`false` (Decision #22)	Disabled globally — Safety Wrapper handles all elevation
`tools.loopDetection.enabled`	`false`	`true`	Prevent runaway tool calls burning tokens
`logging.redactSensitive`	`none`	`"tools"`	Extra protection — redact tool output in logs

Authentication modes: Token (recommended), password, none, or trusted-proxy. For LetsBe, we use **token auth** with the Safety Wrapper as the sole authenticated client.

Rate limiting is built in: configurable max attempts (default 10), time window (60s), lockout duration (300s), with loopback exemption.

Update strategy: Pin OpenClaw to a tested release tag (not `main`). Test upgrades in a staging VPS before rolling out. The Safety Wrapper adapter layer (Decision #12) and hooks (Decision #13) absorb interface changes. Upgrade cadence: monthly review of upstream changelog, upgrade only when there's a security fix or a feature we need.

1.3.2 3.2 Safety Wrapper (LetsBe Core IP)

This is where all LetsBe-specific logic lives. The competitive moat.

The Safety Wrapper is an **OpenClaw extension** — a TypeScript package installed at `~/.openclaw/extensions/letsbe-safety-wrapper/` that hooks into OpenClaw's typed plugin lifecycle. It runs in-process with the gateway (no separate container, no network hop) and intercepts every tool call, message, and LLM interaction through OpenClaw's `before_tool_call`, `after_tool_call`, `message_sending`, and `llm_output` hooks.

Why an extension, not a proxy? OpenClaw's plugin SDK (v2026.2) provides `before_tool_call` hooks that can block, modify, or audit any tool call before execution — with full context (session key, agent ID, tool name, parameters). This gives us everything the proxy approach provided (secrets redaction, command gating, audit logging) without the overhead of a separate process, separate container, or HTTP round-trips. The extension runs at `priority: 10000` (highest), ensuring it executes before any other plugin hooks. The secrets redaction proxy for outbound LLM traffic remains as a separate lightweight process (see Section 3.2.1).

It has five responsibilities:

3.2.1 Secrets Firewall (Outbound Redaction) All LLM-bound traffic passes through the wrapper before reaching OpenRouter. The wrapper runs four layers of redaction:

Layer 1 — Secrets Registry: During provisioning, all generated credentials (50+ per tenant, from the Provisioner's `env_setup.sh`) are logged in an encrypted local reg-

istry (SQLite) with key names, patterns, and locations. When credentials rotate, the registry updates. The registry maps: secret name → value → which tool → which config file → last rotated.

Layer 2 — Outbound Redaction: Before any text leaves the VPS, the wrapper checks against the registry. Known secrets are replaced with deterministic placeholders: `[REDACTED:postgres_password]`, `[REDACTED:nextcloud_admin_key]`. The AI can reason about which credentials are relevant without seeing values.

Layer 3 — Pattern Safety Net: Regex patterns catch secrets the registry might have missed: - Private key blocks (`-----BEGIN.*PRIVATE KEY-----`) - JWT tokens (`eyJ[A-Za-z0-9_-]+\.ey`
- bcrypt hashes (`\$2[aby]?\$[0-9]+\$`) - Connection strings with credentials (`://[^:]+:[^@]+@`)
- High-entropy base64 blobs (length > 32) - Common env var patterns (`(PASSWORD|SECRET|KEY|TOKEN)=`

Layer 4 — Function-Call Proxy: When the AI needs to *use* a secret (e.g., restart a service that needs a DB password), it calls a proxy function:

```
execute_with_credential("restart_postgres", credential_id="pg_main")
```

The Safety Wrapper injects the real value locally and executes. The AI gets the outcome ("PostgreSQL restarted successfully") but never sees the credential value.

Hook integration (Decision #13, updated): The Safety Wrapper extension registers the following OpenClaw typed plugin hooks:

Hook	Priority	Purpose
`before_tool_call`	10000	Primary interception — classify commands, enforce autonomy levels, block/modify tool calls, inject credentials via SECRET_REF
`after_tool_call`	10000	Audit logging — record every tool call with params, result, duration, agent ID
`message_sending`	10000	Content filtering — scrub PII, redact secrets from outbound messages
`before_message_write`	10000	Transcript redaction — strip secrets before session JSONL persistence
`tool_result_persist`	10000	Output redaction — redact secrets from tool outputs before they're saved
`llm_output`	10000	Token metering — capture per-agent, per-model token counts for billing
`before_prompt_build`	10000	Inject safety instructions and tool registry context into system prompt

Hook	Priority	Purpose
`gateway_start`	10000	Verify Hub connectivity, load credentials from registry, register health endpoint
`subagent_spawning`	10000	Control/limit subagent creation, enforce depth limits

3.2.2 Command Classification (Pre-Execution Gating) Every tool call from OpenClaw passes through a classifier before execution. Commands are classified into five tiers:

Green — Non-destructive (auto-execute at all levels, logged): - `file_read`, `env_read` — Read files, logs, configs (output redacted) - `container_stats`, `container_logs` — List/inspect containers - `query_select` — Database queries (SELECT only) - `check_status`, `dns_lookup`, `cert_check` — Infrastructure health checks - `umami_read`, `uptime_check` — Analytics and monitoring reads

Yellow — Modifying (auto-execute at Level 2+, gated at Level 1): - `container_restart` — Restart services - `file_write`, `env_update` — Modify files and .env configs - `nginx_reload` — Reload web server config - `chatwoot_assign`, `calcom_create` — Internal business operations

Yellow+External — External-Facing (gated by default at all levels until user unlocks per agent/tool — Decision #30): - `ghost_publish` — Publish blog content visible to the public - `listmonk_send` — Send email campaigns to subscribers - `poste_send` — Send emails to external recipients - `chatwoot_reply_external` — Reply to customer conversations - `social_post` — Post to social media accounts - `documenso_send` — Send documents for external signature

External-facing operations are gated by a separate **External Comms Gate** that operates independently of the autonomy level system. Even at Level 3 (Full Autonomy), external comms remain gated until the user explicitly unlocks them per agent or per tool via the app. This is a product principle: a misworded email to a client is worse than a delayed newsletter. See Section 6.6 for implementation details.

Red — Destructive (auto-execute at Level 3, gated at Level 1-2): - `file_delete`, `container_remove`, `volume_delete` — Delete resources - `user_revoke`, `db_drop_table`, `backup_delete` — Revoke access, drop data

Critical Red — Irreversible (always gated at all levels): - `db_drop_database`, `firewall_modify`, `ssh_config_modify` — Infrastructure-critical - `backup_wipe_all`, `user_delete_account`, `ssl_revoke` — Unrecoverable

Each agent's tool permissions specify which operations it can see at all (OpenClaw layer) and what autonomy level governs the gating threshold (Safety Wrapper layer). See Section 5 for the full Four-Layer Access Control Model.

3.2.3 Tool Execution Layer Capabilities ported from the deprecated sysadmin agent, exposed as tools OpenClaw can invoke:

Tool	Source	Description
`shell`	sysadmin executor	Execute allowlisted shell commands with timeout enforcement
`docker`	sysadmin executor	Docker Compose operations (up, down, restart, logs, stats)
`file_read`	sysadmin executor	Read files with path traversal prevention, byte limits
`file_write`	sysadmin executor	Write/append files with path security, size limits
`env_read`	sysadmin executor	Read .env files with optional key filtering
`env_update`	sysadmin executor	Atomic .env updates (temp → chmod → rename)
`browser`	OpenClaw native (Playwright + CDP)	Create sessions, navigate, click, type, screenshot via OpenClaw's built-in browser tool (Decision #14)
`web_search`	OpenClaw native	Web search via Brave/Perplexity/Gemini (Decision #26)
`web_fetch`	OpenClaw native	Fetch and parse web pages (Decision #26)
`nextcloud_api`	NEW — tool adapter	Nextcloud WebDAV + OCS REST operations
`chatwoot_api`	NEW — tool adapter	Chatwoot REST API operations
`ghost_api`	NEW — tool adapter	Ghost Content + Admin API operations
`calcom_api`	NEW — tool adapter	Cal.com REST API operations
...	NEW — tool adapters	22+ additional tool API adapters (see Section 7)

Security patterns preserved from sysadmin agent: - Shell command allowlist (specific binaries + arg validation) - Path traversal prevention on all file operations (root: `/opt/letsbe`) - Shell metacharacter blocking - Timeout enforcement on all operations (shell: 60s default, navigation: 120s) - File size limits on writes (10MB default)

3.2.4 Hub Communication The Safety Wrapper handles all communication with the central Hub:

Function	Replaces	Details
Registration	orchestrator + sysadmin agent registration	On first boot, registers with Hub, receives API key
Heartbeat	orchestrator telemetry + sysadmin Hub heartbeat	Periodic status with agent metrics, task counts, credential sync

Function	Replaces	Details
Command queue	orchestrator remote commands	Poll Hub for pending remote commands from admin
Telemetry	orchestrator Hub telemetry service	Windowed metrics (agent activity, task stats, resource usage)
Token metering	NEW	Report AI token usage per agent per model to Hub for billing
Config sync	NEW	Receive agent config changes (autonomy levels, tool permissions) from Hub
Approval queue	NEW	Push yellow/red commands to Hub for approval, receive approval/denial responses
Backup monitoring	NEW (Decision #27)	Monitor `backup-status.json` from existing backup cron, report status to Hub

Hub communication uses OpenClaw's webhook system (Decision #13): Instead of building a custom communication protocol, the Safety Wrapper exposes webhook endpoints that the Hub calls to push commands, sync config changes, and trigger agent tasks. Webhook authentication (bearer token) maps directly to our Safety Wrapper ↔ Hub auth model.

3.2.5 Token Metering OpenClaw natively tracks token usage per agent with separate counters for input, output, cache-read, and cache-write tokens. The Safety Wrapper hooks into this (via `tool_result_persist` or direct Gateway API) to:

1. Capture per-agent, per-model token counts after each LLM call
2. Aggregate into billing windows (hourly buckets)
3. Report to Hub via heartbeat or dedicated metering endpoint
4. Hub aggregates across tenant → customer → subscription for billing

Sub-agents (independent session processes) have isolated token usage — the metering system tracks these separately to ensure accurate billing.

1.3.3 3.3 ~~MCP Browser~~ (DEPRECATED — Decision #14)

Replaced by OpenClaw's native browser tool (Playwright + CDP). The native browser is more capable, saves ~256MB RAM per tenant, and eliminates a separate process.

OpenClaw's browser provides three profile types: - **openclaw-managed** (our choice): Dedicated Chromium instance with isolated user data directory (CDP ports 18800–18899) - **remote CDP**: External Chromium via `cdpUrl` (for Browserless, remote machines) - **extension relay**: Controls existing Chrome tabs via relay + extension

Browser capabilities include: accessibility tree snapshots with numeric refs, screenshots/PDFs, click/double-click/type/hover/drag/select/scroll/fill/press, JS evaluation, file download/upload, navigation with compound wait conditions, cookie CRUD, localStorage, offline mode, device emulation, HTTP auth, geolocation, trace recording, and visual element highlighting.

SSRF protection is built in with configurable allowlists.

Current Playwright scenarios from the deprecated sysadmin agent (8 initial-setup automations for Cal.com, Chatwoot, Keycloak, n8n, Nextcloud, Stalwart Mail, Umami, Uptime Kuma) migrate to run via OpenClaw's native browser tool during provisioning (Decision #23).

1.3.4 3.4 Local Storage

SQLite for all on-server persistent state: - Safety Wrapper: Secrets registry, audit log, pending approvals, Hub sync state, token usage counters - OpenClaw: Conversation history, agent state, session metadata, cron jobs, memory embeddings (all native — JSON files + SQLite via sqlite-vec extension)

No PostgreSQL container on tenant servers. Saves ~256MB RAM per tenant.

1.3.5 3.5 Per-Tenant Container Layout

Container	Image	Port	Resources
`letsbe-openclaw`	Custom image extending OpenClaw (Safety Wrapper extension + CLI binaries pre-installed)	Internal (18789), host network for tool access via `127.0.0.1:30XX`	~512MB RAM (includes Chromium for native browser tool)
`letsbe-secrets-proxy`	LetsBe custom (lightweight Node.js)	8100 (internal)	~64MB RAM
Tool stacks (28+)	Various	Via nginx (`127.0.0.1:30XX`)	Varies per tool
nginx	nginx:alpine	80, 443	~64MB RAM

Key change from v1.1: The Safety Wrapper is now an OpenClaw extension running inside `letsbe-openclaw`, not a separate container. The only remaining separate process is the `letsbe-secrets-proxy` — a thin HTTP proxy that intercepts outbound LLM

traffic for secrets redaction at the transport layer before it reaches OpenRouter. The secrets proxy is the one component that MUST run out-of-process to enforce the "secrets never leave the server" guarantee, since the redaction must happen at the network boundary.

Network access: The OpenClaw container runs with `--network host` (or uses `host.docker.interna` to reach all tool containers via their localhost ports (`127.0.0.1:3023` for Nextcloud, `127.0.0.1:3037` for NocoDB, etc.). This avoids creating a shared Docker network across all 30 tools — each tool keeps its isolated network while the AI accesses them through the host's loopback interface.

Total LetsBe overhead: ~576MB RAM (down from ~1.5GB+ with the original orchestrator + Postgres + sysadmin agent + MCP Browser stack).

1.4 4. Central Platform Architecture

1.4.1 4.1 Hub (letsbe-hub)

The Hub is the most mature component (~15,000 LOC, 244 source files, 80+ API endpoints, 20+ Prisma data models). It remains the central control plane with significant retooling needed.

4.1.1 Keep As-Is These systems are production-ready and require no architectural changes:

- **Staff admin dashboard** — RBAC with 4 roles (OWNER/ADMIN/MANAGER/SUPPORT), 20 permissions, 2FA via TOTP with backup codes, staff invitations via email
- **Customer management** — CRUD, subscriptions (TRIAL/STARTER/PRO/ENTERPRISE), Stripe integration
- **Order lifecycle** — 8-state automation state machine (PAYMENT_CONFIRMED → AWAITING_SERVER → SERVER_READY → DNS_PENDING → DNS_READY → PROVISIONING → FULFILLED/FAILED)
- **Netcup SCP API integration** — Full OAuth2 Device Flow, server list/detail, power actions, metrics, snapshots, rescue mode, hostname, reinstall (~1,150 lines)
- **Portainer integration** — JWT auth with self-signed cert support via undici, container list/inspect/logs/stats/start/stop/restart/remove (~707 lines)
- **DNS verification workflow** — A-record verification for all tool subdomains, wildcard detection
- **Docker-based provisioning** — Job spawning with retry logic (1min/5min/15min backoff), SSE log streaming, SELECT FOR UPDATE SKIP LOCKED claiming
- **Stripe webhook integration** — `checkout.session.completed` → User + Subscription + Order creation
- **Enterprise client management** — Per-server monitoring, error detection rules, container health (crash/OOM/restart)
- **Email notifications** — Nodemailer SMTP with cooldown, welcome/test/notification emails

- **Credential encryption** — AES-256-CBC with scrypt key derivation, legacy key migration support
- **System settings** — 50+ settings across 9 categories, encrypted storage
- **Security verification codes** — 8-digit codes for destructive operations (WIPE/REINSTALL)
- **Server stats collection** — CPU/memory/disk/network from Netcup + container counts from Portainer, 90-day retention
- **S3/MinIO storage** — Staff profile photos

4.1.2 Retool / Add Customer-Facing Portal (`/customer` route group): Self-service dashboard for customers to view server status, manage agents, configure tools, see credentials (via Safety Proxy side-channel), approve commands, view token usage, and manage billing. This is the web companion to the mobile app.

New API endpoints needed: | Method | Path | Description | |——-|——|———–| | GET | `/api/v1/customer/dashboard` | Customer overview (server status, agent status, recent activity) | | GET | `/api/v1/customer/agents` | List customer's agents with status | | GET/PATCH | `/api/v1/customer/agents/[id]` | Get/update agent config (autonomy level, SOUL.md, tool permissions) | | GET | `/api/v1/customer/agents/[id]/activity` | Agent activity feed | | GET | `/api/v1/customer/agents/[id]/conversations` | Conversation history | | GET | `/api/v1/customer/usage` | Token usage summary (per agent, per model, daily/weekly/monthly) | | GET | `/api/v1/customer/usage/breakdown` | Detailed usage breakdown for billing transparency | | GET | `/api/v1/customer/approvals` | Pending command approval queue | | POST | `/api/v1/customer/approvals/[id]` | Approve or deny a pending command | | GET | `/api/v1/customer/tools` | List deployed tools with status | | GET | `/api/v1/customer/billing` | Current billing period, usage, overages | | GET | `/api/v1/customer/backups` | Backup status and history |

Billing & Token Metering (Decision #24): - Flat token pool per subscription tier with overage billing - Per-customer, per-agent, per-model usage tracking - OpenRouter proxy layer that meters all AI token usage - Sliding markup on token costs (configurable per tier) - Pool tracking: each customer gets a monthly token allotment, usage reported in real-time - Overage billing: when pool is exceeded, charges apply at a per-token rate through Stripe - Web search/fetch tool usage counted in the same token pool model

New API endpoints: | Method | Path | Description | |——-|——|———–| | POST | `/api/v1/admin/billing/usage` | Ingest token usage report from Safety Wrapper | | GET | `/api/v1/admin/billing/[customerId]` | Customer billing summary | | GET | `/api/v1/admin/billing/[cust` | Historical usage data | | POST | `/api/v1/admin/billing/overages` | Trigger overage billing via Stripe |

New Prisma models:

```
model TokenUsageBucket {
  id          String   @id @default(uuid())
  userId      String
  orderId     String
  agentId     String
  model       String
  bucketHour  DateTime
```

```
  tokensInput     Int  @default(0)
  tokensOutput    Int  @default(0)
  tokensCacheRead Int  @default(0)
  tokensCacheWrite Int @default(0)
  webSearchCount  Int  @default(0)
  webFetchCount   Int  @default(0)
  costCents       Int  @default(0)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  user  User  @relation(fields: [userId], references: [id])
  order Order @relation(fields: [orderId], references: [id])

  @@unique([orderId, agentId, model, bucketHour])
  @@index([userId, bucketHour])
  @@map("token_usage_buckets")
}

model BillingPeriod {
  id              String   @id @default(uuid())
  userId          String
  subscriptionId  String
  periodStart     DateTime
  periodEnd       DateTime
  tokenAllotment  Int      // base allotment × founding member multiplier
  tokensUsed      Int      @default(0)
  overageTokens   Int      @default(0)
  overageCostCents Int     @default(0)
  premiumTokensUsed Int    @default(0)  // premium model tokens (separate from pool)
  premiumCostCents  Int    @default(0)  // premium model cost after markup
  stripeInvoiceId String?
  status          String   @default("ACTIVE") // ACTIVE, CLOSED, BILLED
  createdAt       DateTime @default(now())

  user         User         @relation(fields: [userId], references: [id])
  subscription Subscription @relation(fields: [subscriptionId], references: [id])

  @@index([userId, periodStart])
  @@map("billing_periods")
}

model FoundingMember {
  id                String   @id @default(uuid())
  userId            String   @unique
```

```
  tokenMultiplier   Int      @default(2)  // 2× token allotment ("Double the AI")
  startDate         DateTime @default(now())
  expiresAt         DateTime // 12 months from start
  isActive          Boolean  @default(true)
  createdAt         DateTime @default(now())

  user User @relation(fields: [userId], references: [id])

  @@map("founding_members")
}
```

DNS Automation (currently manual): Integrate with Cloudflare or Entri API to automatically create A records (not just verify them). Currently blocks fully automated AUTO mode provisioning because DNS records must be created manually.

New API endpoint: | Method | Path | Description | |———-|——|————-| | POST | `/api/v1/admin/orders/[id]/dns/create` | Auto-create DNS A records via Cloudflare/Entri |

Agent Management API: CRUD for agent configurations (SOUL.md, TOOLS.md, model selection, permissions). Synced to tenant servers via Safety Wrapper heartbeat.

New API endpoints: | Method | Path | Description | |———-|——|————-| | GET | `/api/v1/admin/agents/templates` | List available agent role templates | | POST | `/api/v1/admin/orders/[` | Deploy new agent to tenant server | | GET | `/api/v1/admin/orders/[id]/agents` | List agents on tenant server | | PATCH | `/api/v1/admin/orders/[id]/agents/[agentId]` | Update agent config (SOUL.md, tools, autonomy) | | DELETE | `/api/v1/admin/orders/[id]/agents/[agentI` | Remove agent from tenant |

New Prisma model:

```
model AgentConfig {
  id              String   @id @default(uuid())
  orderId         String
  agentId         String   // matches OpenClaw agent ID
  name            String
  role            String   // it-admin, marketing, secretary, sales, custom
  soulMd          String   @db.Text  // agent personality/instructions
  toolsAllowed    String[] // tool IDs this agent can access
  toolsDenied     String[] // explicitly denied tools
  toolProfile     String   @default("minimal") // OpenClaw tool profile
  autonomyLevel   Int?     // null = use tenant default
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  order Order @relation(fields: [orderId], references: [id])

  @@unique([orderId, agentId])
```

```
  @@map("agent_configs")
}
```

Command Approval Queue: Yellow/red tier commands from tenant servers surface here for admin or customer approval (push notifications via mobile app).

New API endpoints: | Method | Path | Description | |——-|——|———-| | GET | `/api/v1/admin/approvals` | List all pending approvals across tenants | | GET | `/api/v1/admin/approvals/` | Approval detail with command context | | POST | `/api/v1/admin/approvals/[id]` | Approve or deny |

New Prisma model:

```
model CommandApproval {
  id              String   @id @default(uuid())
  orderId         String
  agentId         String
  commandClass    String   // yellow, red, critical_red
  toolName        String
  toolArgs        Json
  humanReadable   String   // "IT Agent wants to delete /nextcloud/data/tmp/* (47 files, 2.3GB)
  status          String   @default("PENDING") // PENDING, APPROVED, DENIED, EXPIRED
  requestedAt     DateTime @default(now())
  respondedAt     DateTime?
  respondedBy     String?  // staff ID or customer ID
  expiresAt       DateTime // auto-expire after 24h
  createdAt       DateTime @default(now())

  order Order @relation(fields: [orderId], references: [id])

  @@index([orderId, status])
  @@index([status, expiresAt])
  @@map("command_approvals")
}
```

Safety Wrapper Communication API (replaces orchestrator phone-home endpoints):

The current orchestrator endpoints (`/api/v1/orchestrator/register`, `/heartbeat`, `/commands`) are retooled for Safety Wrapper communication:

Method	Path	Auth	Description
POST	`/api/v1/tenant/register`	Registration token	Safety Wrapper registers post-deploy, receives Hub API key
POST	`/api/v1/tenant/heartbeat`	Bearer {hubApiKey}	Status heartbeat with metrics, token usage, backup status, receive config updates

Method	Path	Auth	Description
GET	/api/v1/tenant/config	Bearer {hubApiKey}	Pull full agent config (OpenClaw + Safety Wrapper JSON)
POST	/api/v1/tenant/approval-request	Bearer {hubApiKey}	Push command approval request to Hub
GET	/api/v1/tenant/approval-response/[id]	Bearer {hubApiKey}	Poll for approval/denial
POST	/api/v1/tenant/usage	Bearer {hubApiKey}	Report token usage buckets
POST	/api/v1/tenant/backup-status	Bearer {hubApiKey}	Report backup execution status

New Prisma model update:

```
model ServerConnection {
  // Existing fields kept...
  id                String   @id @default(uuid())
  orderId           String   @unique
  registrationToken String   @unique
  hubApiKey         String?
  hubApiKeyHash     String?

  // RENAMED: orchestratorUrl → safetyWrapperUrl
  safetyWrapperUrl  String?

  // NEW fields
  openclawVersion   String?
  safetyWrapperVersion String?
  lastTokenUsageSync DateTime?
  lastBackupReport   DateTime?
  configVersion      Int      @default(0)

  status            String   @default("PENDING") // PENDING, REGISTERED, ONLINE, OFFLINE
  lastHeartbeat     DateTime?

  order Order @relation(fields: [orderId], references: [id])

  @@map("server_connections")
}
```

Token Usage Analytics Dashboard: Per-customer, per-agent, per-model usage dashboards with cost breakdown.

New API endpoints: | Method | Path | Description | |——-|——|————-| | GET | /api/v1/admin/analytics/tokens | Global token usage overview | | GET | /api/v1/admin/analytics/toke

| Per-tenant token usage | | GET | `/api/v1/admin/analytics/costs` | Cost breakdown by customer/model/agent |

4.1.3 Cut

- `ansible/runner.ts` (SSH-based inline provisioning) — the Docker-based provisioner is the canonical path. The SSH inline path adds confusion, attack surface, and maintenance burden. Remove entirely.
- **Legacy patterns** — orchestrator's dual auth (token + secret hash), dashboard auth dependency (defined but unused in orchestrator), plaintext token auth.
- **Orchestrator phone-home endpoints** — `/api/v1/orchestrator/*` routes replaced by `/api/v1/tenant/*` routes for Safety Wrapper communication. Old routes removed after migration.

4.1.4 Current Hub API Inventory For reference, the Hub currently exposes 80+ API endpoints across these groups:

Authentication (8 endpoints): NextAuth handlers, staff invitation accept, 2FA setup/verify/disable/backup-codes, initial setup.

Profile (3 endpoints): Get/update profile, upload photo, change password.

Admin — Customers (4 endpoints): List, create, get, update customers.

Admin — Orders (16 endpoints): Full order lifecycle management including provisioning, DNS verification, automation mode, container management via Portainer.

Admin — Servers (4 endpoints): List, health check, remote commands, Portainer ping.

Admin — Netcup (8 endpoints): OAuth2 auth flow, server list/detail/actions, metrics, snapshots.

Admin — Enterprise Clients (20+ endpoints): Full CRUD, error detection rules, container events, notification settings, server management, security verification.

Admin — Staff (5 endpoints): List, invite, manage invitations, update/delete staff.

Admin — Settings & Analytics (6 endpoints): Settings CRUD, email/storage test, stats, analytics.

Orchestrator Phone-Home (4 endpoints): Register, heartbeat, command poll/report.

Public API (2 endpoints): Create/get order from external source.

Webhooks (1 endpoint): Stripe checkout completed.

Cron (2 endpoints): Stats collection, stats cleanup.

4.1.5 Hub Data Model Summary **Core Business Models (11):** User, Staff, StaffInvitation, Subscription, Order, DnsVerification, DnsRecord, ProvisioningJob, JobLog, ProvisioningLog, TokenUsage

Infrastructure Models (2): RunnerToken, ServerConnection

Enterprise/Monitoring Models (9): EnterpriseClient, EnterpriseServer, ServerStatsSnapshot, ErrorDetectionRule, DetectedError, SecurityVerificationCode, LogScanPosition, ContainerStateSnapshot, ContainerEvent

Notification/Auth Models (3): NotificationSetting, NotificationCooldown, Pending2FASession

System Models (1): SystemSetting

New Models (this redraft): TokenUsageBucket, BillingPeriod, AgentConfig, CommandApproval, plus ServerConnection updates

1.4.2 4.2 Provisioner (was letsbe-ansible-runner)

Renamed to `letsbe-provisioner`. One-shot Bash container (~4,477 LOC) that provisions a fresh VPS via SSH. No architectural changes to the provisioning pipeline — just updates to what it deploys.

4.2.1 Keep

- **10-step server provisioning pipeline** (`setup.sh`, ~832 lines):
  1. System packages
  2. Docker CE installation
  3. Disable conflicting services
  4. nginx + fallback config
  5. UFW firewall (ports 80, 443, 22022)
  6. Optional admin user + SSH key
  7. SSH hardening (port 22022, key-only auth)
  8. Unattended security updates
  9. Deploy tool stacks via docker-compose
  10. Deploy LetsBe agents + bootstrap
- **28+ Docker Compose tool stacks** + 33 nginx configs
- **Template rendering** (`env_setup.sh`, ~678 lines) — 50+ secrets generation, `{{ variable }}` substitution
- **SSH hardening** — port 22022, key-only auth, fail2ban
- **Backup system** (`backups.sh`, ~473 lines) — 18 PostgreSQL, 2 MySQL, 1 MongoDB dumps + rclone remote + rotation (7 daily + 4 weekly) + `backup-status.json` output
- **Restore system** (`restore.sh`, ~512 lines) — per-tool and full restore from local or remote backups
- **Hub integration** — log streaming via `POST /api/v1/jobs/{JOB_ID}/logs`, status updates via `PATCH`

4.2.2 Update

- Deploy **OpenClaw + Safety Wrapper** instead of orchestrator + sysadmin agent (MCP Browser no longer deployed — Decision #14)
- **Clean up config.json** after provisioning (currently contains root password in plaintext — CRITICAL security fix)
- Generate and deploy Safety Wrapper configuration:

  - Secrets registry initial seed (all 50+ generated credentials)
  - Agent configs (SOUL.md defaults, tool permissions per role template)
  - Hub credentials (API key, registration token)
  - Autonomy level defaults (Level 2 — Trusted Assistant)
  - Safety Wrapper JSON config (command classification rules)
- Deploy OpenClaw config:
  - `openclaw.json` with model provider pointing to Safety Wrapper proxy
  - Agent definitions with tool profiles and allow/deny lists
  - Prompt caching settings (`cacheRetention: "long"`, heartbeat keep-warm)
  - Loop detection enabled
  - Elevated mode disabled
  - Browser tool configured with `openclaw-managed` profile
- **Migrate Playwright scenarios** — The 8 initial-setup browser automations (Cal.com, Chatwoot, Keycloak, n8n, Nextcloud, Stalwart Mail, Umami, Uptime Kuma) run via OpenClaw's native browser tool during provisioning step 10 (Decision #23)
- **Add tests** (currently zero — the provisioner has no test suite)
- Remove MCP Browser from `stacks/sysadmin/docker-compose.yml`
- Remove `MCP_BROWSER_API_KEY={{ mcp_browser_api_key }}` template variable (was never generated in `env_setup.sh` — bug identified in repo analysis)

1.5 5. Four-Layer Access Control Model (Decision #22)

Security is enforced through four independent layers, each adding restrictions. No layer can expand access granted by layers above it — they can only restrict further.

1.5.1 5.1 Layer 1 — Sandbox (Where Code Runs)

OpenClaw's native sandbox system controls the execution environment:

Mode	Description	LetsBe Use
`off` (default)	No containerization	Primary mode — Safety Wrapper handles gating
`non-main`	Only non-default agents sandboxed	For untrusted custom agents
`all`	Every agent sandboxed	Maximum isolation (performance cost)

Container configuration per agent: scope (agent/session/shared), workspace access (none/ro/rw), Docker settings (custom image, read-only root, network isolation, UID:GID mapping, capability dropping, memory/CPU/PID limits).

LetsBe default: Sandbox mode `off` for all default agents (IT Admin, Marketing, Secretary, Sales). The Safety Wrapper provides command-level gating which is more

granular than container isolation. For custom user-created agents or untrusted operations, sandbox mode can be enabled per-agent.

1.5.2 5.2 Layer 2 — Tool Policy (What Tools Are Visible)

OpenClaw's native `agents.list[].tools.allow/deny` arrays control which tools each agent can *see at all*. Deny wins over allow. This is configured in the OpenClaw config and synced from Hub.

Cascading restriction model (each level can only restrict further): 1. Tool profiles (`tools.profile` — coding, minimal, messaging, full) 2. Global policies (`tools.allow`/`tools.deny`) 3. Agent-specific policies (`agents.list[].tools.allow/deny`)

Tool group shorthands for convenience: - `group:runtime` → exec, bash, process - `group:fs` → read, write, edit, apply_patch - `group:sessions` → session management tools - `group:messaging` → message tool - `group:ui` → browser, canvas

Example — Marketing Agent can see Ghost/Listmonk/Umami/file_read/browser/Nextcloud but cannot see shell/docker/env_update:

```
{ "id": "marketing", "tools": { "profile": "minimal", "allow": ["ghost_api", "listmonk_api", "umami_api", "file_read", "browser", "nextcloud_api"], "deny": ["shell", "docker", "env_update"] } }
```

1.5.3 5.3 Layer 3 — Command Gating (What Operations Require Approval)

The Safety Wrapper's autonomy level system (Section 6) determines which *operations on visible tools* require human approval before execution. This is LetsBe's layer — OpenClaw doesn't know about it.

Even if an agent can see a tool (Layer 2 allows it), the Safety Wrapper may gate specific operations on that tool. For example, Marketing Agent can use `ghost_api` (Layer 2 allows it), but `ghost_api.delete_post` is classified as Red and requires approval at Level 2 (Layer 3 gates it).

1.5.4 5.4 Layer 4 — Secrets Redaction (Always On, Non-Negotiable)

Regardless of sandbox mode, tool permissions, or autonomy level, **all outbound LLM traffic is redacted**. Secrets are stripped at the transport layer before any data reaches OpenRouter. This layer cannot be disabled. It runs even at Full Autonomy (Level 3).

1.5.5 5.5 Elevated Mode — DISABLED

OpenClaw has a native "Elevated Mode" that requests unrestricted host execution (`security=full`). **This is disabled globally for all LetsBe deployments** (Decision #22). The Safety Wrapper handles all permission elevation through the autonomy level system. There is no path for an agent to bypass the Safety Wrapper by requesting elevated privileges from OpenClaw.

Config: `security.elevated.enable: false` in `openclaw.json`.

1.5.6 5.6 OpenAI-Compatible API — LOCKED DOWN

OpenClaw exposes an OpenAI-compatible API endpoint. **This is locked down and not exposed to customers** (Decision #25). It is only accessible locally for internal Safety Wrapper communication. No external access is allowed.

1.6 6. AI Autonomy Levels (Decision #17)

A customer-facing feature that controls how much the AI can do without human approval. Configured per-tenant and optionally per-agent via the app settings. Synced from Hub to Safety Wrapper.

1.6.1 6.1 Level Definitions

Level	Name	Auto-Execute	Requires Approval	Use Case
1	Training Wheels	Green (read-only)	Yellow + Red + Critical Red	New customers, cautious users, initial onboarding
2	Trusted Assistant (default)	Green + Yellow (reads + modifications)	Red + Critical Red	Established trust, daily operations
3	Full Autonomy	Green + Yellow + Red	Critical Red only	Power users, DevOps-savvy owners

1.6.2 6.2 Per-Agent Override

Each agent can have its own autonomy level independent of the tenant default:

Agent	Tenant Default	Agent Override	Effective Level
IT Admin	Level 2	Level 3	3 — full autonomy for infrastructure
Marketing	Level 2	(none)	2 — default
Secretary	Level 2	Level 1	1 — read-only, approval for all actions
Sales	Level 2	(none)	2 — default

1.6.3 6.3 Implementation

The Safety Wrapper's command classification (green/yellow/red/critical_red) is unchanged. What changes per autonomy level is the **approval threshold**:

Command Class	Level 1	Level 2	Level 3
Green (read-only)	Auto-execute	Auto-execute	Auto-execute
Yellow (modifying)	**Gate → approval**	Auto-execute	Auto-execute
Red (destructive)	**Gate → approval**	**Gate → approval**	Auto-execute
Critical Red (irreversible)	**Gate → approval**	**Gate → approval**	**Gate → approval**

Invariant across all levels: Secrets are always redacted. Audit trail is always logged. The AI never sees raw credentials regardless of autonomy level.

1.6.4 6.4 OpenClaw Tool Restrictions (Native Layer)

TOOLS.md is a freeform guidance file, not a permissions schema. It contains "environment-specific" information — the agent's cheat sheet for SSH hosts, device locations, voice preferences. It does NOT control tool availability.

Actual tool restrictions are configured in `agents.list[].tools` with `allow` and `deny` arrays. These support tool group shorthands (Section 5.2).

LetsBe mapping: OpenClaw's native tool allow/deny handles *which tools* an agent can see (Layer 2). The Safety Wrapper's autonomy levels handle *what the agent can do* with those tools (Layer 3). Two complementary layers: OpenClaw says "Marketing Agent can use ghost_api but not docker", Safety Wrapper says "and ghost_api publish operations need approval at Level 1 but auto-execute at Level 2+".

1.6.5 6.5 Autonomy Level → Config Mapping (Concrete Schema)

Each autonomy level maps to two config layers: **OpenClaw agent config** (tool visibility) and **Safety Wrapper config** (command gating thresholds). The Hub generates both configs when autonomy level changes and syncs them to the tenant server.

```
   { "agents": { "list": [ { "id": "dispatcher", "name": "Your AI Team", "soul": "./agents/dispatcher/
"tools": { "profile": "messaging", "allow": ["agentToAgent"], "deny": ["shell", "docker",
"file_write", "env_update"] } }, { "id": "it-admin", "name": "IT Admin", "soul": "./agents/it-admin/SOU
"tools": { "profile": "coding", "allow": ["shell", "docker", "file_read", "file_write", "env_read",
"env_update", "browser", "portainer_api", "nextcloud_api", "web_search", "web_fetch"], "deny":
[] } }, { "id": "marketing", "name": "Marketing", "soul": "./agents/marketing/SOUL.md", "tools":
{ "profile": "minimal", "allow": ["ghost_api", "listmonk_api", "umami_api", "file_read", "browser",
"nextcloud_api", "web_search", "web_fetch"], "deny": ["shell", "docker", "env_update"] } },
{ "id": "secretary", "name": "Secretary", "soul": "./agents/secretary/SOUL.md", "tools": {
"profile": "messaging", "allow": ["calcom_api", "chatwoot_api", "poste_api", "file_read",
"nextcloud_api", "web_search"], "deny": ["shell", "docker", "env_update", "file_write"] } },
{ "id": "sales", "name": "Sales", "soul": "./agents/sales/SOUL.md", "tools": { "profile":
"minimal", "allow": ["chatwoot_api", "odoo_api", "calcom_api", "file_read", "nextcloud_api",
"web_search", "web_fetch"], "deny": ["shell", "docker", "env_update"] } } ] } }
```

OpenClaw Agent Config (`agents.list[]`)


```
  { "tenant": { "id": "tenant-abc123", "default_autonomy_level": 2 }, "agents": { "it-admin":
{ "autonomy_level": 3 }, "marketing": {}, "secretary": { "autonomy_level": 1 }, "sales": {}
}, "autonomy_levels": { "1": { "name": "Training Wheels", "auto_execute": ["green"], "gate_for_approva
["yellow", "red", "critical_red"] }, "2": { "name": "Trusted Assistant", "auto_execute": ["green",
"yellow"], "gate_for_approval": ["red", "critical_red"] }, "3": { "name": "Full Autonomy",
"auto_execute": ["green", "yellow", "red"], "gate_for_approval": ["critical_red"] } }, "command_classif
{ "green": ["file_read", "env_read", "container_stats", "container_logs", "query_select", "check_status
"dns_lookup", "cert_check", "umami_read", "uptime_check"], "yellow": ["container_restart",
"file_write", "env_update", "nginx_reload", "chatwoot_assign", "calcom_create"], "yellow_external":
["ghost_publish", "listmonk_send", "poste_send", "chatwoot_reply_external", "social_post",
"documenso_send"], "red": ["file_delete", "container_remove", "volume_delete", "user_revoke",
"db_drop_table", "backup_delete"], "critical_red": ["db_drop_database", "firewall_modify",
"ssh_config_modify", "backup_wipe_all", "user_delete_account", "ssl_revoke"] } }
```


Safety Wrapper Config (`safety-wrapper.json`)

Resolution Logic

1. Look up agent ID → check for agent-specific autonomy_level override
2. If no override → use tenant.default_autonomy_level
3. Classify the command (green/yellow/yellow_external/red/critical_red)
4. If yellow_external → check External Comms Gate (Section 6.6)
5. Otherwise → check if command class is in auto_execute or gate_for_approval for the effective
6. Auto-execute → run immediately, log to audit trail
7. Gate → push approval request to Hub → wait for user response via app

Invariants (all levels): Secrets always redacted. Audit trail always logged. AI never sees raw credentials. External comms gated until explicitly unlocked.

1.6.6 6.6 External Communications Gate (Decision #30)

External-facing operations — anything that sends information to someone outside the business — are governed by a separate gate that operates **independently of the autonomy level system**. This is a product principle, not a gating tier.

Rationale: A misworded email to a client or a prematurely published blog post damages the business's reputation. Internal operations (restarting a container, modifying a config) are recoverable. External communications are not. Users must build trust with their AI team before allowing autonomous external-facing actions.

Default state: All external comms operations are **gated** for every agent. The AI prepares the action (drafts the email, writes the blog post, composes the campaign) and presents it for one-tap approval.

Unlock mechanism: Users can explicitly unlock autonomous external sending per agent and per tool via the app settings:

```
  { "external_comms_gate": { "default": "gated", "unlocks": { "marketing": { "ghost_publish":
"autonomous", "listmonk_send": "gated", "social_post": "gated" }, "secretary": { "poste_send":
"autonomous", "calcom_create": "autonomous" } } } }
```

Resolution: When a Yellow+External command is intercepted: 1. Check `external_comms_gate.unl` 2. If `"autonomous"` → follow normal autonomy level gating (Yellow rules apply) 3. If `"gated"` or not set → always gate, regardless of autonomy level 4. Present to user: "Marketing Agent wants to publish: 'Top 10 Tips for...' to your blog. [Approve] [Edit] [Deny]"

Invariant: External comms are gated by default. This cannot be changed by the AI. Only the user can unlock autonomous external sending through the app UI. The unlock is per-agent, per-tool — not global.

New Prisma field on AgentConfig:

```
model AgentConfig {
  // ... existing fields ...
  externalCommsUnlocks Json?  // { "ghost_publish": "autonomous", "poste_send": "gated" }
}
```

New Hub API endpoint: | Method | Path | Description | |——-|——|———-| | PATCH | `/api/v1/customer/agents/[id]/external-comms` | Update external comms gate per tool |

1.7 7. Tool Integration Strategy (Updated v1.2)

The critical engineering investment that creates the competitive barrier.

Each customer's VPS runs 28+ containerized business tools, each with its own API. The AI agent needs admin-level access to all of them — via REST API where available, via Playwright browser automation where not. Rather than building individual TypeScript adapters for each tool, we use a **tool registry + master skill + cheat sheet** architecture optimized for token efficiency and scalability.

1.7.1 7.1 Architecture: Three Access Patterns

The AI agent interacts with tools through three patterns, chosen per-tool based on API availability:

Pattern 1 — REST API via `exec` tool (primary, preferred): The agent runs `curl` commands against tool REST APIs through OpenClaw's built-in `exec` tool. The Safety Wrapper's `before_tool_call` hook intercepts every `exec` call, audits the command, and can block or modify it. Credentials are injected from the tool registry (the agent uses `SECRET_REF` placeholders, the Safety Wrapper substitutes real values at execution time).

```
Agent → exec("curl http://127.0.0.1:3037/api/v2/tables -H 'xc-token: SECRET_REF(nocodb_api_toke
  → Safety Wrapper intercepts → injects real token → executes → returns result to agent
```

Pattern 2 — CLI binaries via `exec` tool (for external services): For external services not running on the VPS, pre-installed CLI binaries provide structured access. OpenClaw's skill system teaches the agent the CLI syntax.

```
Agent → exec("gog gmail search 'newer_than:1d' --max 10")  ← Google via gog CLI
Agent → exec("himalaya envelope list --folder INBOX")       ← IMAP via himalaya CLI
```

Pattern 3 — Browser automation via `browser` tool (fallback): For tools without APIs, or complex UI workflows that aren't exposed via API, the agent uses OpenClaw's built-in Playwright browser tool to navigate the tool's web UI via its localhost port.

```
Agent → browser(navigate: "http://127.0.0.1:3023")  ← Nextcloud web UI
Agent → browser(snapshot) → browser(act: click ref=42) → browser(act: type "content")
```

Selection priority: API first (fast, reliable, token-efficient). Browser second (slower, more tokens, but works for anything with a UI). The master skill teaches the agent to prefer API access and fall back to browser only when needed.

1.7.2 7.2 Tool Registry

During provisioning, alongside the credentials, we generate a `tool-registry.json` file that describes every installed tool. This file is loaded into the AI's context via the `before_prompt_build` hook — it's the agent's "directory" of what's available.


```
{ "tools": { "nextcloud": { "name": "Nextcloud", "category": "file-storage", "internal_url":
"http://127.0.0.1:3023", "external_url": "https://files.{{domain}}", "api_base": "/ocs/v2.php/apps",
"api_auth_type": "basic", "api_auth_ref": "SECRET_REF(nextcloud_admin_credentials)", "has_api":
true, "has_webui": true, "cheat_sheet": "references/nextcloud.md", "description": "File storage,
sharing, calendar, contacts, office docs" }, "nocodb": { "name": "NocoDB", "category": "database",
"internal_url": "http://127.0.0.1:3037", "api_base": "/api/v2", "api_auth_type": "header",
"api_auth_header": "xc-token", "api_auth_ref": "SECRET_REF(nocodb_api_token)", "has_api":
true, "has_webui": true, "cheat_sheet": "references/nocodb.md", "description": "Spreadsheet-database
hybrid, tables, records, views" } }, "external_services": { "google": { "name": "Google Workspace",
"cli": "gog", "skill": "gog", "services": ["gmail", "calendar", "drive", "contacts", "sheets",
"docs"], "configured": true }, "imap": { "name": "Email (IMAP/SMTP)", "cli": "himalaya", "skill":
"himalaya", "configured": true } } }
```


Token efficiency: The registry itself is compact (~2-3K tokens for 30 tools). It's always in context. The cheat sheets are NOT loaded into context by default — the agent reads them on-demand from the `references/` directory when it needs to interact with a specific tool, keeping the base context window lean.

1.7.3 7.3 Master Skill

A single master skill file (`skills/letsbe-tools/SKILL.md`) teaches the AI how to use the tool registry:

```
   --- name: letsbe-tools description: "Access and manage all business tools on this server"
---
   ## Tool Access
   You have admin-level access to all tools listed in tool-registry.json.
   ### API Access (preferred) For tools with `has_api: true`, use curl via the exec tool: -
Always use the `internal_url` (localhost ports), never external URLs - Use SECRET_REF() for
credentials - the system injects real values - Check the tool's cheat sheet (in references/)
for endpoint documentation - Parse JSON responses and present results clearly to the user
   ### Browser Access (fallback) For tools without APIs, or for complex UI tasks: - Navigate
to the tool's `internal_url` using the browser tool - Use snapshot to read the page, act to
interact - Take screenshots to verify your actions
   ### Rules - Always check tool-registry.json before attempting to access a tool - Read the
cheat sheet for a tool before making your first API call to it - Prefer API over browser -
it's faster and uses fewer tokens - All credentials use SECRET_REF() - never hardcode or guess
passwords
```

1.7.4 7.4 Per-Tool Cheat Sheets

Each tool gets a concise reference document in `references/` that covers its most common API endpoints. These are NOT full API docs — they're curated "top 20 operations" optimized for LLM consumption.

Example (`references/nocodb.md`, ~200 lines):

```
   # NocoDB API Cheat Sheet Base: http://127.0.0.1:3037/api/v2 Auth: xc-token header via SECRET_REF(noc
   ## Tables GET /meta/tables → List all tables GET /meta/tables/{tableId} → Get table schema
POST /meta/tables → Create table
   ## Records GET /tables/{tableId}/records?limit=25 → List records (paginated) POST /tables/{tableId}/
→ Create record (JSON body) PATCH /tables/{tableId}/records → Update records (JSON body) DELETE
/tables/{tableId}/records → Delete records
   ## Common Patterns - Filter: ?where=(field,eq,value) - Sort: ?sort=-field (descending)
- Fields: ?fields=field1,field2
```

Scalability: Adding a new tool = add its entry to the registry template + write a cheat sheet. No TypeScript code, no adapter class, no plugin registration. A cheat sheet for a typical tool takes 30-60 minutes to write.

Auto-generation potential (V2): Cheat sheets can be auto-generated from OpenAPI/Swagger specs that many tools provide, further reducing the effort to add new tools.

1.7.5 7.5 Tool Inventory

Tool	API Type	Access Pattern	Cheat Sheet Priority
Portainer	REST v2	API	P0
Nextcloud	WebDAV + OCS REST	API + Browser (admin settings)	P0

Tool	API Type	Access Pattern	Cheat Sheet Priority
Chatwoot	REST v1/v2	API	P0
Ghost	Content + Admin REST	API	P0
Cal.com	REST v2	API	P0
Stalwart Mail	REST	API	P0
Odoo	XML-RPC + JSON-RPC	API	P1
Listmonk	REST	API	P1
NocoDB	REST v2	API	P1
n8n	REST	API	P1
Umami	REST	API	P1
Keycloak	Admin REST	API	P1
Gitea	REST v1	API	P2
Uptime Kuma	REST + Push	API	P2
MinIO	S3 (via `aws` CLI or curl)	CLI/API	P2
Documenso	REST	API	P2
VaultWarden	Bitwarden REST	API	P2
WordPress	REST v2	API + Browser (plugins/themes)	P2
Activepieces	REST	API	P3
Windmill	REST	API	P3
Redash	REST	API	P3
Penpot	REST	API	P3
Squidex	REST + GraphQL	API	P3
Typebot	REST	API	P3

External services (via CLI binaries pre-installed in Docker image):

Service	CLI	Skill	Priority
Google Workspace	`gog`	`skills/gog/SKILL.md`	P0
IMAP/SMTP Email	`himalaya`	`skills/himalaya/SKILL.md`	P0
Web Search	Built-in (`web_search` tool)	N/A	P0
Web Browsing	Built-in (`browser` tool)	N/A	P0

1.7.6 7.6 Credential Flow for Tool Access

```
Provisioning (one-time):
  env_setup.sh generates 50+ credentials
    → writes to /opt/letsbe/env/credentials.env
    → writes per-tool .env files
    → Safety Wrapper reads credentials.env at startup
```

```
    → populates encrypted secrets registry (SQLite)
    → generates tool-registry.json with SECRET_REF() placeholders

Runtime (every tool call):
  Agent decides to call NocoDB API
    → exec("curl ... -H 'xc-token: SECRET_REF(nocodb_api_token)' ...")
    → Safety Wrapper before_tool_call hook intercepts
    → classifies command (green/yellow/red)
    → if allowed: substitutes SECRET_REF with real token from registry
    → executes curl command
    → after_tool_call hook logs the action (with token redacted in logs)
    → returns result to agent
```

1.7.7 7.3 Dynamic Tool Installation (Decision #32)

The AI team can deploy new open-source tools on the user's server — on request.

This is one of the most powerful capabilities in the platform. A user says "I need a wiki" and the IT Agent deploys BookStack or WikiJS, configures it behind nginx with SSL, seeds credentials in the secrets registry, and reports back — all gated behind user approval.

7.3.1 How It Works

1. User requests a tool: "Can you set up a wiki for my team?"
2. Dispatcher routes to IT Agent
3. IT Agent consults the **Tool Catalog** — a curated registry of pre-tested open-source tools with Docker Compose templates, nginx configs, resource requirements, and optional API adapters
4. IT Agent presents the recommendation: "I recommend BookStack — it's a simple wiki that fits your server. It needs ~256MB RAM and you have 4GB free. Want me to install it?"
5. **User approves** (this is a Red-tier operation — always gated)
6. IT Agent executes: pulls Docker images, deploys compose stack, configures nginx reverse proxy, generates credentials, stores in secrets registry, runs health check, creates subdomain DNS record if needed
7. IT Agent reports: "BookStack is live at wiki.yourdomain.com. Here's your admin login." (credentials via secure SECRET_CARD)
8. If an API adapter exists for the new tool, it's automatically registered — other agents can now use it

7.3.2 Tool Catalog A curated, version-pinned catalog of open-source tools that the IT Agent can deploy. Each entry contains:

```
  { "id": "bookstack", "name": "BookStack", "description": "Simple, self-hosted wiki and
documentation platform", "category": "knowledge-management", "docker_compose": "./catalog/bookstack/doc
"nginx_config": "./catalog/bookstack/nginx.conf", "env_template": "./catalog/bookstack/env.template",
"resource_requirements": { "ram_mb": 256, "cpu_shares": 512, "disk_gb": 2 }, "api_adapter":
"bookstack_api", "has_adapter": true, "setup_steps": ["deploy", "nginx_config", "ssl_cert",
"seed_credentials", "health_check"], "tested_version": "24.10", "subdomain_pattern": "wiki.{{domain}}"
}
```


Curated, not open-ended. The IT Agent cannot deploy arbitrary Docker images from the internet. It can only deploy tools from the catalog. The catalog is maintained by LetsBe and ships with each provisioning. Users can request tools be added to the catalog.

Beyond the core 30: The catalog includes the 30 tools that ship by default, plus an extended library of additional open-source tools that users can request. Examples: BookStack, WikiJS, Plausible Analytics, Mattermost, Rocket.Chat, Trilium Notes, Paperless-ngx, Invoice Ninja, Monica CRM, Firefly III, Grafana, Leantime, Plane, Focalboard, and more.

7.3.3 Resource Gating Before deploying, the IT Agent checks: - **Available RAM** — will the new tool fit without starving existing services? - **Available disk** — enough storage for the tool + its data? - **Server tier limits** — is the user's VPS big enough? If not, suggest an upgrade - **Port conflicts** — no collisions with existing services

If resources are insufficient, the IT Agent reports: "BookStack needs 256MB RAM but you only have 128MB free. I can install it if you upgrade your server tier, or I can suggest a lighter alternative."

7.3.4 Safety Controls

Control	Enforcement
Always gated	Tool installation is Red-tier — requires user approval at all autonomy levels
Catalog-only	IT Agent cannot deploy images outside the curated catalog
Resource check	Pre-deployment resource validation prevents server overload
Credential management	All generated credentials stored in secrets registry, never exposed to AI
Rollback	If deployment fails, all changes are cleaned up (compose down, nginx config reverted)
Audit trail	Full deployment log stored for debugging and accountability

7.3.5 Tool Removal Users can also ask the IT Agent to remove tools: "I don't use the wiki anymore, can you remove it?" This is also Red-tier (data deletion involved).

The IT Agent: 1. Warns about data loss 2. Offers to back up tool data first 3. On approval: stops containers, removes compose stack, removes nginx config, removes credentials from registry 4. Reports: "BookStack removed. Your data backup is at /opt/letsbe/backups/bookstack-20260225.tar.gz"

1.8 8. Skills & Extensibility (Decision #29, updated v1.2)

OpenClaw's skills system is our primary mechanism for teaching AI agents how to use business tools — and the main lever for controlling token cost. Skills are Markdown documents injected into agent context, not executable code. This makes them easy to write, easy to audit, and cheap to update.

1.8.1 8.1 OpenClaw Skills System

Skills are directories loaded from workspace, managed, and bundled locations. Each skill has a `SKILL.md` file containing instructions the AI reads at prompt time. Skills have no handler code — they are pure context injection. The AI reads the instructions and follows them using its available tools.

Skills can be distributed via ClawHub registry (OpenClaw's package manager).

Skill loading precedence: 1. Workspace skills (agent-specific, highest priority) 2. Managed skills (installed via ClawHub) 3. Bundled skills (shipped with OpenClaw, lowest priority)

Critical implication for token cost: Every skill loaded into an agent's context consumes tokens on every LLM call. A 500-word skill costs ~750 tokens per turn. Five verbose skills = ~3,750 tokens per turn = hundreds of thousands of wasted tokens per day. Skill design must be compact.

1.8.2 8.2 Tool Registry & Master Skill (v1.2)

Rather than writing 28+ individual skills (one per tool), LetsBe uses a three-layer architecture designed for token efficiency:

Layer 1 — Tool Registry (`/opt/letsbe/config/tool-registry.json`)

A structured JSON file generated at provisioning time that describes every installed tool. The Safety Wrapper extension injects this into agent context via the `before_prompt_build` hook. Total size: ~2-3K tokens for 30 tools.

Each entry contains: tool name, internal URL, API base path, auth type, credential reference (as `SECRET_REF()` placeholder — resolved at runtime by the `before_tool_call` hook), and path to the tool's cheat sheet.

Layer 2 — Master Skill (`skills/letsbe-tools/SKILL.md`)

One skill file (~500-800 tokens) loaded into every agent's context. It teaches the AI three access patterns:

1. **REST API via `exec` + `curl`** — For containerized tools with HTTP APIs. The AI constructs curl commands targeting `127.0.0.1:30XX` endpoints using the tool registry.

2. **CLI binaries (`gog`, `himalaya`)** — For external services (Google Workspace, IMAP email). Installed on the host, callable via `exec`.
3. **Playwright browser automation** — Fallback for tools without APIs. The AI uses OpenClaw's native `browser` tool (Playwright + CDP) to interact with tool web UIs.

The master skill also teaches the AI how to read the tool registry, when to load a cheat sheet (only when working with an unfamiliar tool), and how to handle credential placeholders.

Layer 3 — Per-Tool Cheat Sheets (`references/<tool-name>.md`)

Short reference documents (200-500 tokens each) covering a specific tool's API endpoints, common operations, and gotchas. These are NOT loaded into context by default — the AI loads them on-demand using `memory_search` or file read when it needs to work with a specific tool for the first time in a session.

This means the base context cost per agent is: master skill (~700 tokens) + tool registry (~2,500 tokens) = **~3,200 tokens** regardless of how many tools are installed. Compare this to 30 individual skills at ~750 tokens each = ~22,500 tokens always in context.

Adding a new tool: Write a registry entry + a cheat sheet. No code needed. Estimated time: 30-60 minutes per tool.

1.8.3 8.3 Agent Role Skills

In addition to the tool access skills above, each agent role gets a compact SOUL.md that encodes domain expertise and behavioral rules. These are not tool-specific — they define the agent's personality, decision-making style, and domain knowledge:

Agent Role	SOUL.md Focus	Estimated Size
Dispatcher	Intent routing rules, agent capability summaries, workflow decomposition patterns	~800 tokens
IT Admin	Server health heuristics, incident response protocols, security best practices	~600 tokens
Marketing	Content strategy patterns, campaign workflows, brand voice rules	~600 tokens
Secretary	Calendar management rules, email triage priorities, communication protocols	~600 tokens
Sales	Lead qualification criteria, pipeline management rules, follow-up cadence	~600 tokens

1.8.4 8.4 User Customization

Users can customize to their heart's content (Decision #29). They can: - Modify SOUL.md to change agent personality, tone, and domain knowledge - Add custom skills by writing skill files in the agent workspace - Enable/disable provided template skills - Create entirely new agent roles beyond the four defaults - Install community skills from ClawHub (if we choose to enable this) - Write their own cheat sheets for tools or workflows specific to their business

Customization is done through the app/customer portal, which syncs changes to the tenant server via the Hub → Safety Wrapper config sync.

1.8.5 8.5 Token Efficiency Strategy

Token cost is the largest variable expense in the platform. Every architectural decision in this section optimizes for fewer tokens per interaction:

Strategy	Savings
Tool registry (structured JSON) vs. verbose skill descriptions	~80% reduction in tool context
On-demand cheat sheets vs. always-loaded skills	Only pay for tools actually used in a session
Compact SOUL.md files (~600 tokens) vs. verbose personas	~50% reduction in agent identity context
`cacheRetention: "long"` for SOUL.md via OpenRouter	80-99% cheaper on repeated calls for system prompt
Context pruning (`cache-ttl`, 1h default)	Stale tool outputs auto-removed
Session compaction (auto-summarize when context fills)	Keeps long conversations from blowing up costs

Monitoring: The Safety Wrapper's `llm_output` hook captures per-agent, per-model token counts on every LLM call. These are reported to the Hub for billing and for identifying agents or skills that are disproportionately expensive.

1.9 9. Memory Architecture (OpenClaw Native)

OpenClaw's memory system is file-based and Markdown-driven. This is critical for inter-agent communication and persistent knowledge.

1.9.1 9.1 Memory Layers

Layer	Location	Purpose	Loaded When
Daily logs	`memory/YYYY-MM-DD.md`	Session context, running notes	Today + yesterday loaded at startup
Long-term memory	`MEMORY.md`	Curated durable knowledge (decisions, facts, preferences)	Loaded in private sessions
Session transcripts	Experimental	Full conversation recall	Via `memory_search` when enabled

Key principle: Files are the source of truth. The AI only "remembers" what gets written to disk.

1.9.2 9.2 Memory Search (Hybrid Retrieval)

OpenClaw uses weighted score fusion combining two retrieval methods:

- **Vector search** (cosine similarity via sqlite-vec extension): Captures paraphrases and semantic meaning. Embeddings stored in per-agent SQLite databases (`~/.openclaw/memory/`
- **BM25 keyword search** (SQLite FTS5): Handles exact tokens, IDs, code symbols.

Additional features: - **MMR re-ranking**: Balances relevance with diversity (lambda 0.7 default) - **Temporal decay**: Exponential boost for recent memories (30-day half-life). `MEMORY.md` and non-dated files never decay. - **Local embeddings**: `ggml-org/embeddinggemma-300` (~0.6GB). No external API calls.

1.9.3 9.3 Context Management & Compaction

OpenClaw requires 64K+ tokens of context. When context window approaches capacity:

1. **Memory flush** triggers — a silent agentic turn (uses `NO_REPLY` convention) writes durable state to `memory/YYYY-MM-DD.md` files
2. **Compaction** summarizes older conversation turns into compressed entries
3. The agent continues with compressed context + full access to memory search

Context pruning (separate from compaction): Mode `cache-ttl` with configurable TTL (default 1h) removes stale tool outputs.

Session maintenance: Auto-prunes stale sessions (30 days default), caps entries (500 default), rotates oversized stores (10MB default). Cron sessions have separate retention (24h default).

1.9.4 9.4 Implications for LetsBe

SOUL.md as cacheable memory: Each agent's SOUL.md is loaded as a prompt prefix. With `cacheRetention: "long"` via OpenRouter, the first call per agent per hour costs full price, but subsequent calls are 80-99% cheaper for the system prompt portion.

Agent isolation model: OpenClaw isolates agents by default — each gets its own workspace, auth profiles, and session store. We use OpenClaw's `extraPaths` memory search config to give agents read access to a shared memory directory (`/opt/letsbe/shared-memory/`) while keeping individual workspaces separate. This enables cross-agent discovery without breaking isolation.

Per-agent files (OpenClaw native):

File	Purpose	LetsBe Usage
`SOUL.md`	Agent personality, instructions	Brand voice, expertise area, behavioral rules
`AGENTS.md`	Agent metadata	Role definition, display name, capabilities summary
`USER.md`	User context	Business owner preferences, timezone, communication style
`skills/`	Agent-specific tools	Master tool skill + role-specific skills (see §8.2-8.3)
`auth-profiles.json`	Per-agent credentials	Tool-specific API keys (managed by Safety Wrapper)

1.10 10. Inter-Agent Communication

1.10.1 10.1 OpenClaw Native Support

OpenClaw has built-in agent-to-agent communication, **disabled by default**, enabled via configuration:

```
tools: {
  agentToAgent: {
    enabled: true,
    allow: ["it_admin", "marketing", "secretary", "sales"]
  }
}
```

LetsBe enables this for all agents in a customer's team, with the Safety Wrapper adding gating rules on top.

1.10.2 10.2 Communication Patterns

Shared Memory (Passive, Asynchronous): Each agent has its own workspace, but all agents are configured with `extraPaths` pointing to `/opt/letsbe/shared-memory/`. When one agent writes to the shared directory, other agents can discover it via `memory_search`.

Agent-to-Agent Dispatch (Active, via OpenClaw Native): One agent can message another:

```
agentToAgent(target: "it_admin", message: "Pull Umami analytics for the blog, last 30 days")
```

The target agent processes the request using its own tools and skills, and returns the result.

Event Bus (Reactive, Event-Driven): Agents subscribe to events from tools they monitor via custom skills: - IT Agent subscribes to container crash events → auto-investigates - Sales Agent subscribes to new Chatwoot conversations → auto-qualifies leads - Secretary Agent subscribes to new Cal.com bookings → sends confirmations

Events flow through the Safety Wrapper, which applies command gating before any action is taken.

1.10.3 10.3 Message Routing — The Dispatcher Agent (Decision #31)

The Dispatcher is a first-class default agent, not an optional component. It is the user's primary point of contact and the embodiment of the "talk to your team" experience.

10.3.1 How It Works The Dispatcher Agent is always present on every tenant. It is the default conversation target when a user opens the app or sends a message via WhatsApp/Telegram. It has three responsibilities:

1. **Intent routing:** Receives user messages, classifies intent, and delegates to the appropriate specialist agent. "Send the monthly newsletter" → Marketing Agent. "Why is the website slow?" → IT Agent. "Schedule a meeting with John" → Secretary Agent.
2. **Workflow decomposition:** When a request spans multiple domains, the Dispatcher breaks it into ordered steps and coordinates across agents. "Follow up with everyone from last week's demo" → Secretary (get attendees) → Sales (look up records) → Marketing (draft messages) → Secretary (send emails) → Sales (log activity).
3. **Morning briefing & proactive updates:** The Dispatcher aggregates reports from all agents and presents a unified daily briefing to the user. It surfaces what each agent did overnight, what needs attention, and what's coming up.

SOUL.md for Dispatcher: Knows the domain descriptions of all other agents (reads their SOUL.md summaries). Understands the user's business type and common

request patterns. Defaults to asking the user when routing is ambiguous: "Should I have your IT Agent or Marketing Agent handle this?"

Configuration:

```
{ "id": "dispatcher", "name": "Your AI Team", "soul": "./agents/dispatcher/SOUL.md", "tools": { "profile": "messaging", "allow": ["agentToAgent"], "deny": ["shell", "docker", "file_write", "env_update"] } }
```

The Dispatcher does not have direct tool access (no shell, docker, file operations). It works exclusively through delegation to specialist agents. This keeps it lightweight and prevents scope creep.

10.3.2 Direct Agent Chat Users can also bypass the Dispatcher and talk directly to a specific agent. The app exposes:

- **Default view:** "Talk to your team" → routes through Dispatcher
- **Agent selector:** Tap on a specific agent avatar to open a direct conversation with that agent

Direct conversations go straight to the selected agent without Dispatcher involvement. This is useful when the user knows exactly which agent they need, or when they want to have an ongoing conversation about a specific domain (e.g., discussing blog strategy with Marketing Agent over multiple messages).

Both modes are available simultaneously. The Dispatcher handles the "I just need something done" experience; direct chat handles the "I want to work with this specific agent" experience.

10.3.3 Subagent-to-Subagent Communication All specialist agents can communicate with each other directly via OpenClaw's native agent-to-agent dispatch. The Dispatcher is the primary orchestrator for multi-step workflows, but agents can also coordinate directly:

- Marketing Agent asks IT Agent: "What's the current Ghost version? I need to know if the new editor features are available."
- Sales Agent asks Secretary Agent: "Is there a meeting already booked with Acme Corp this week?"
- IT Agent asks Marketing Agent: "I'm upgrading Ghost — do you have any scheduled posts in the next hour I should wait for?"

This peer-to-peer communication is logged, rate-limited, and visible in the Agent Activity feed.

1.10.4 10.4 Workflow Decomposition

When a user request spans multiple agent domains, the system decomposes it:

User: "Follow up with everyone who attended last week's demo"

```
1. Secretary Agent → Cal.com API: Get attendees from last week's demo event
2. Sales Agent → Chatwoot/Odoo API: Look up each attendee's record
3. Marketing Agent → AI reasoning: Draft personalized follow-up messages
4. Secretary Agent → Stalwart API: Send each follow-up email
5. Sales Agent → Odoo API: Log follow-up activity on each contact record
```

1.10.5 10.5 Safety Controls

Control	Purpose
Loop prevention	Maximum dispatch depth (5 levels). A→B→A→B→... is killed.
Rate limiting	Maximum inter-agent dispatches per minute per agent
Audit trail	Every dispatch logged with source agent, target, task, result
Tool gating	An agent can only dispatch to agents it has permission to communicate with
User visibility	All activity visible in the app's "Agent Activity" feed; user can interrupt or cancel

---

1.11 11. Billing & Token Metering (Decision #24, updated Decision #33)

1.11.1 11.1 Billing Model

Two-tier model: included pool for basic models + pay-as-you-go for premium models.

11.1.1 Included Models (Base Subscription — Decision #33) Every subscription tier includes a generous monthly token pool covering three curated model presets. These cover 90%+ of daily usage. No credit card needed beyond the subscription.

Preset	Model	Use Case	User-Facing Name
Basic Tasks	Gemini Flash / GPT 5 Nano	Quick lookups, simple scheduling, basic drafts	"Basic Tasks"
Balanced (default)	DeepSeek V3.2	Day-to-day operations, most agent work	"Balanced (Recommended)"

Preset	Model	Use Case	User-Facing Name
Complex Tasks	GLM 5 / MiniMax M2.5	Multi-step reasoning, analysis, complex workflows	“Complex Tasks”

Model selection UX: - **Basic Settings (default):** User sees three preset cards with plain-English descriptions. No model names. “Balanced” is pre-selected. One-tap to change. Applies globally or per-agent. - **Advanced Settings (requires credit card):** Full model catalog with descriptions, strengths, use cases, and pricing. Per-agent model selection. Premium models unlocked here.

Token pool per subscription tier (finalize pre-launch, see Pricing Model v2.2):

Tier	Monthly Token Pool	Covers
Lite (€29/mo)	~8M tokens	Basic Tasks, Balanced, Complex Tasks presets
Build (€45/mo)	~15M tokens	Basic Tasks, Balanced, Complex Tasks presets
Scale (€75/mo)	~25M tokens	Basic Tasks, Balanced, Complex Tasks presets
Enterprise (€109/mo)	~40M tokens	Basic Tasks, Balanced, Complex Tasks presets

Founding members (Decision #34): First 50-100 customers receive **2× token allotment** for 12 months (“Double the AI”). Implemented as a multiplier on the subscription tier pool. See Section 11.4 for data model.

11.1.2 Premium Models (Pay-as-You-Go) Premium models are available to users who add a credit card in Advanced Settings. Usage is metered per-token and billed monthly via Stripe. These never draw from the included pool.

Model	Strengths	Use Case	Markup
Claude Sonnet 4.6	Balanced power + speed	Strong all-rounder, great for business operations	10%
Claude Opus 4.6	Maximum reasoning capability	Complex analysis, high-stakes decisions, nuanced writing	8%
GPT 5.2	Strong general intelligence	Multi-domain tasks, creative content	12%

Model	Strengths	Use Case	Markup
Gemini 3.1 Pro	Large context, multimodal	Document analysis, data-heavy tasks	10%

Each premium model card in the app shows: description, strengths, ideal use cases, and approximate cost per 1K tokens.

11.1.3 Sliding Markup (Threshold-Based — Decision #35) Markup on premium models uses a threshold system: the more expensive the model's base cost, the lower the markup. This encourages adoption of premium models without punishing power users.

Base Cost Threshold (blended cost per 1M tokens)	Markup
< $1	25%
$1 - $5	15%
$5 - $15	10%
> $15	8%

Example: Claude Opus 4.6 at ~$41/M blended → 8% markup. Claude Sonnet 4.6 at ~$8/M blended → 10% markup. Gemini 3.1 Pro at ~$6.3/M blended → 10% markup. Thresholds use blended cost (60% input / 40% output weighted), not input price alone. See Pricing Model v2.2 for exact per-model calculations.

Markup thresholds are configurable in Hub system settings and can be adjusted without code changes.

11.1.4 Overage Billing (Included Models) When the included token pool is exhausted, the user is notified and given two options: 1. **Pause AI usage** until next billing period 2. **Opt into overage billing** — included models continue at a per-token rate with tiered markup: - Cheapest models < $0.50/M (DeepSeek V3.2, GPT 5 Nano): 35% markup - Mid-tier $0.50-1.20/M (GPT 5.2 Mini, MiniMax M2.5): 25% markup - Top included > $1.20/M (GLM 5, Gemini Flash): 20% markup

Web search and web fetch tool usage counts against the same token pool — no separate metering.

1.11.2 11.2 Token Tracking Flow

```
1. Agent makes LLM call → OpenClaw routes through Safety Wrapper proxy → OpenRouter
2. OpenRouter returns response with token counts (input/output/cache-read/cache-write)
3. Safety Wrapper captures token counts per agent, per model
4. Safety Wrapper aggregates into hourly buckets (TokenUsageBucket)
5. Safety Wrapper reports buckets to Hub via heartbeat or dedicated /tenant/usage endpoint
6. Hub aggregates into BillingPeriod for each customer
```

7. Hub checks pool usage → if exceeded, triggers overage billing via Stripe
8. Customer sees real-time usage in app/portal

OpenClaw natively tracks input/output/cache-read/cache-write tokens separately. Sub-agents (independent session processes) have isolated token usage — tracked separately for accurate billing.

1.11.3 11.3 Cost Optimization

Prompt caching is the biggest cost saver. With `cacheRetention: "long"` and heartbeat keep-warm (`heartbeat.every: "55m"`): - SOUL.md (agent personality, ~4,000 tokens) is cached for 1 hour - Subsequent calls pay 80-99% less for the system prompt portion - At 50 calls/day per agent × 5 agents = significant daily savings per tenant

Model selection per agent: Not all agents need the most expensive model. Secretary Agent might use a cheaper model for simple scheduling, while IT Admin uses a more capable model for infrastructure reasoning.

1.11.4 11.4 Founding Member Program (Decision #34)

First 50-100 customers receive 2× token allotment for 12 months ("Double the AI").

Implementation: When a BillingPeriod is created for a founding member, the `tokenAllotment` is calculated as: `baseTierAllotment × foundingMember.tokenMultiplier`. The multiplier defaults to 2 but is configurable per founding member.

Flow: 1. Admin flags user as founding member in Hub → creates FoundingMember record with 12-month expiry 2. On each billing period creation, Hub checks: is user a founding member? Is it still active (not expired)? 3. If yes → `tokenAllotment = baseTierAllotment × tokenMultiplier` 4. If expired → normal allotment 5. Founding member status visible in customer portal and admin dashboard

No other special treatment. Founding members get the same product, same features, same support. Just more tokens.

1.12 12. Security Architecture

1.12.1 12.1 Trust Boundaries

```
UNTRUSTED            TRUSTED (on-VPS)

External LLMs        Safety Wrapper (redacts secrets)    OpenClaw
(OpenRouter,
 Anthropic,              Safety Wrapper (classifies commands)    Tool Execution
 Google, etc.)

                                                          Server filesystem,
                                                          Docker, databases,
                                                          tool APIs
```

Key principle: The LLM is untrusted. Everything sent to it is sanitized. Everything received from it is validated before execution. The Safety Wrapper is the single enforcement point.

1.12.2 12.2 Secrets Never Leave the Server

What	Where Enforced	Bypassable by AI?
Secrets redacted before LLM call	Safety Wrapper HTTP proxy	No — transport-level
Credential injection happens locally	Safety Wrapper function-call proxy	No — AI only sees outcome
Secrets registry encrypted at rest	SQLite + encryption key	No — key not in AI context
Pattern safety net catches unknowns	Regex middleware in Safety Wrapper	No — runs before transmission
Hooks redact tool outputs in transcripts	`tool_result_persist` hook	No — runs before save

1.12.3 12.3 Command Gating

What	Where Enforced	Bypassable by AI?
Destructive commands gated	Safety Wrapper pre-execution hook	No — hook runs before exec
Per-agent tool visibility	OpenClaw `tools.allow/deny` config	No — config loaded at startup
Per-agent autonomy level	Safety Wrapper config, synced from Hub	No — local config
Shell command allowlist	Safety Wrapper tool executor	No — allowlist is code

What	Where Enforced	Bypassable by AI?
Path traversal prevention	Safety Wrapper file tools	No — validation is code
Audit log of all actions	Append-only local log	No — write-only from AI
Container runs as non-root	Docker/OS enforcement	No — OS-level
Elevated mode disabled	OpenClaw config	No — global flag
Loop detection	OpenClaw native (30-call window)	No — hard stop at 30

1.12.4 12.4 Network Security

- **OpenClaw bound to loopback** — not accessible from outside the VPS
- **Safety Wrapper bound to loopback** — only OpenClaw and Hub (via nginx reverse proxy) can reach it
- **UFW firewall** — only ports 80, 443, 22022 open
- **SSH hardened** — port 22022, key-only auth, fail2ban
- **OpenClaw rate limiting** — 10 attempts per 60s, 300s lockout
- **SSRF protection** in browser tool — configurable allowlists

1.12.5 12.5 Security Quick Wins (from Repo Analysis)

#	Fix	Repo	Priority
1	Clean up config.json after provisioning (contains root password)	provisioner	CRITICAL
2	Remove inline SSH provisioning path (`ansible/runner.ts`)	hub	HIGH
3	Add rate limiting to Hub public API and webhook endpoints	hub	MEDIUM

#	Fix	Repo	Priority
4	Remove legacy plaintext token auth	orchestrator (before deprecation)	MEDIUM
5	Run `openclaw security audit --deep` during provisioning	provisioner	MEDIUM

1.13 13. Secrets UX — Safe Credential Exchange

The Secrets Firewall (Section 3.2.1) handles redaction at the system level — the AI never sees raw credential values. But users still need to *work with* credentials through the chat interface. This section defines how the app enables that interaction while keeping the AI completely out of the loop on actual values.

Core principle: Secrets travel on a **side channel** between the app and the Safety Proxy. The AI conversation only ever contains references (`[SECRET_REF:...]`). The actual values are exchanged directly between the app UI and the Safety Proxy's secrets API, never passing through OpenClaw or any LLM.

1.13.1 13.1 Flow 1 — User Provides a Secret (Inbound)

```
1. User: "Create an email account for sarah@mybusiness.com"
2. AI calls tool: create_email_account(address="sarah@mybusiness.com", password=?)
3. Safety Proxy detects missing credential → returns structured response:
   { "status": "awaiting_secret", "secret_id": "sarah_email_password",
     "prompt": "Password for sarah@mybusiness.com",
     "allow_generate": true,
     "constraints": { "min_length": 12, "require_special": true } }
4. AI responds: "I'll set that up. Please provide a password."
   (response includes metadata tag: SECRET_INPUT_REQUIRED)
5. App detects SECRET_INPUT_REQUIRED → renders secure modal
6. User enters password or taps Generate → app sends DIRECTLY to Safety Proxy:
   POST /secrets/provide { secret_id: "sarah_email_password", value: "..." }
7. Safety Proxy stores in registry, notifies AI: "Credential received"
8. AI retries tool call with SECRET_REF("sarah_email_password")
9. Safety Proxy injects real password, calls Stalwart API
10. AI reports: "Email account created for sarah@mybusiness.com  "
```

The password never appears in the chat transcript. The AI never sees it.

1.13.2 13.2 Flow 2 — User Requests a Secret (Outbound)

```
1. User: "What's my Nextcloud admin password?"
2. AI calls tool: get_credential(service="nextcloud", key="admin_password")
3. Safety Proxy returns to AI: { "secret_id": "nextcloud_admin_password", "display": true }
4. AI responds: "Here's your Nextcloud admin password:" (includes SECRET_CARD metadata)
5. App renders secure credential card with [Tap to Reveal] [Copy]
6. User taps Reveal → app fetches from Safety Proxy (authenticated, encrypted)
7. Value displayed temporarily → auto-clears after 30s
8. Audit log entry recorded
```

1.13.3 13.3 Flow 3 — Password Generation

```
1. AI decides a new credential is needed
2. AI calls: generate_credential(service="ghost", key="admin_password", constraints={...})
3. Safety Proxy generates cryptographically secure password locally
4. Safety Proxy stores in registry, returns: { "secret_id": "ghost_admin_password", "status": "
5. AI uses SECRET_REF for subsequent tool calls, includes SECRET_CARD for user
```

1.13.4 13.4 Flow 4 — Password Rotation

```
1. User: "Rotate my Nextcloud admin password"
2. AI calls: rotate_credential(secret_id="nextcloud_admin_password")
3. Safety Proxy: generates new → updates target service → updates registry → logs event
4. AI reports success, includes SECRET_CARD for new password
5. If service update fails → rollback, report failure
```

1.13.5 13.5 Safety Proxy Secrets API

Endpoint	Method	Purpose	Auth
`/secrets/provide`	POST	User submits a new credential value	Hub session token
`/secrets/reveal`	GET	User requests to view a credential	Hub session token
`/secrets/generate`	POST	Generate a new credential	Hub session token
`/secrets/list`	GET	List credentials (names + metadata, no values)	Hub session token
`/secrets/rotate`	POST	Rotate a credential	Hub session token
`/secrets/audit`	GET	View access/rotation history	Hub session token (admin)

Transport: All secret values encrypted in transit (TLS). For the Hub relay path (App → Hub → Safety Proxy), end-to-end encryption between app and Safety Proxy — Hub cannot read values in transit.

1.13.6 13.6 Messaging Channel Fallback

For WhatsApp/Telegram channels where secure modals aren't available:

Outbound: One-time secure link: `"Here's your password: https://app.letsbe.biz/secret/abc123 - expires in 10 minutes, viewable once."`

Inbound: Secure input link: `"Please enter the password here: https://app.letsbe.biz/secret/i`

1.13.7 13.7 Invariants

Rule	Enforcement
AI never sees raw secret values	Safety Proxy returns references only; outbound redaction catches leaks
Values never in chat transcripts	OpenClaw session files contain `SECRET_REF(...)` tokens only
Values never sent to LLM providers	Outbound redaction proxy strips before forwarding
All access is audited	Every provide, reveal, generate, rotate, and use event logged
User must authenticate to view	Hub session token required
Generated passwords are cryptographically secure	`crypto.randomBytes()` (Node.js)
Values encrypted at rest	SQLite secrets registry encrypted with tenant-specific key
Rotation is atomic	New password applied to service first; registry updated only on success

---

1.14 14. Data Flow

1.14.1 14.1 Customer Signs Up → Server Provisioned → AI Ready

```
1. Customer completes Stripe checkout
2. Hub webhook creates User + Subscription + Order (status: PAYMENT_CONFIRMED)
3. Automation state machine: PAYMENT_CONFIRMED → AWAITING_SERVER
4. Hub assigns Netcup server from pre-provisioned pool (region per customer choice: EU or NA)
5. State: AWAITING_SERVER → SERVER_READY
6. Hub creates DNS records via Cloudflare/Entri API (NEW - currently manual)
7. State: SERVER_READY → DNS_PENDING → DNS_READY
8. Hub spawns Provisioner container with job config
```

9. Provisioner: SSH → upload stacks/scripts → generate 50+ secrets → deploy tools → deploy Oper
10. Safety Wrapper registers with Hub, receives API key
11. OpenClaw loads default agent configs (per business type template)
12. State: PROVISIONING → FULFILLED
13. Customer receives welcome email with dashboard URL
14. Customer opens app → talks to their AI workforce

1.14.2 14.2 Agent Executes a Non-Destructive Task

1. User: "Check why Nextcloud is slow"
2. OpenClaw routes to IT Agent (via dispatcher or direct agent selector)
3. IT Agent decides to: check container stats, read logs, inspect config
4. OpenClaw calls Safety Wrapper tool: container_stats("nextcloud")
5. Safety Wrapper classifies: Green → auto-execute
6. Safety Wrapper executes: docker stats nextcloud → returns result
7. IT Agent calls: read_logs("nextcloud", lines=100)
8. Safety Wrapper classifies: Green → auto-execute
9. Safety Wrapper reads logs → redacts any secrets in output → returns
10. IT Agent analyzes: "Nextcloud is using 95% of its memory limit"
11. OpenClaw responds to user with diagnosis and recommendation

1.14.3 14.3 Agent Executes a Destructive Task

1. User: "Clean up old Nextcloud temp files and restart it"
2. IT Agent plans: delete temp files (Red) + restart container (Yellow)
3. OpenClaw calls: delete_files("/opt/letsbe/stacks/nextcloud/data/tmp/*")
4. Safety Wrapper classifies: Red → check autonomy level
   - If Level 3: auto-execute
   - If Level 1 or 2: gate → push to Hub → mobile notification
5. User sees: "IT Agent wants to delete /nextcloud/data/tmp/* (47 files, 2.3GB)"
6. User taps "Approve" → Hub notifies Safety Wrapper → executes
7. OpenClaw calls: restart_container("nextcloud")
8. Safety Wrapper classifies: Yellow → auto-execute at Level 2+
9. IT Agent reports: "Cleaned up 2.3GB and restarted Nextcloud"

1.15 15. Error Handling & Resilience

1.15.1 15.1 Severity-Based Alerting (Decision #19)

Severity	Examples	Auto-Recovery	Alert
Soft	OpenClaw crash, Safety Proxy hang, tool adapter timeout	Auto-restart immediately, log to Hub	None on first occurrence. Push notification after 3 failures in 1 hour.
Medium	Tool API unreachable, OpenRouter timeout, Hub communication failure	Retry with exponential backoff (30s → 1m → 5m). Mark tool as degraded.	Push notification after 3 consecutive failures.
Hard	Auth token rejected by Hub, secrets registry corrupted, disk full, SSL cert expired	Stop affected component. Do NOT auto-restart.	Immediate push notification to customer + Hub alert to LetsBe staff.

1.15.2 15.2 Process Supervision

Each tenant server uses Docker restart policies (`restart: unless-stopped`): - **OpenClaw crash**: Auto-restart. Sessions survive (persisted to disk). - **Safety Proxy crash**: Auto-restart. During downtime (~2-5 seconds), OpenClaw's LLM calls fail — agents pause until proxy is back. - **Both down**: Docker restarts both. Hub detects missed heartbeats after 2 intervals and marks tenant as "degraded".

1.15.3 15.3 Graceful Degradation

Component Down	User Experience
Single tool (e.g., Nextcloud)	Agent says "I can't reach Nextcloud right now. I'll try again in a few minutes."
Safety Proxy	Agents pause (can't make LLM calls). Resume automatically on restart.
OpenClaw	All agents offline. Auto-restart. User sees "Your AI team is restarting" in app.
Hub	Agents continue working locally (cached config). Heartbeats queue up. Approvals delayed. User can still talk via WhatsApp/Telegram.

Component Down	User Experience
OpenRouter	LLM calls fail. Model failover chain tries alternatives. If all fail, agent reports temporary issue.

1.15.4 15.4 Model Failover (OpenClaw Native)

OpenClaw supports model fallback chains:

```
  { "model": { "primary": "anthropic/claude-sonnet-4-5", "fallbacks": ["anthropic/claude-haiku-4-5",
"google/gemini-2.0-flash"] } }
```

Failover includes auth profile rotation before model fallback — if the primary model fails due to an API key issue, OpenClaw rotates through auth profiles before falling back to a different model. Billing-specific cooldowns prevent repeated failures from burning budget.

1.15.5 15.5 Multi-Gateway (Decision #28)

Single gateway per VPS for v1. Each tenant gets one OpenClaw Gateway instance.

For future enterprise tier, we may revisit multi-gateway for: - Higher availability (active-passive failover) - Load distribution for high-volume tenants - Separate gateway for cron jobs vs. interactive use

This is a post-launch concern. The current architecture supports multiple Gateway instances on the same host using different config paths and state directories.

1.16 16. Backup Strategy (Decision #27 — Option C Hybrid)

1.16.1 16.1 Application-Level Backups (Existing — KEEP)

The Provisioner already deploys a robust backup system (`backups.sh`, ~473 lines): - **18 PostgreSQL databases** backed up (Chatwoot, Ghost, Keycloak, n8n, Odoo, etc.) - **2 MySQL databases** (WordPress, Ghost) - **1 MongoDB** (LibreChat) - **Plus:** env files, nginx configs, rclone config, crontab - **Schedule:** Daily 2:00 AM cron job - **Rotation:** 7 daily local + 4 weekly remote (via rclone) - **Output:** `backup-status.json` with execution timestamp and per-database status

This system is proven and stays as-is. It's the safety net.

1.16.2 16.2 OpenClaw Monitoring (NEW)

A cron job in OpenClaw monitors backup health by reading `backup-status.json`: - **Schedule:** Daily at 6:00 AM (after 2:00 AM backup completes) - **Checks:** Was backup-status.json updated today? Are all databases listed? Any failures? - **Reports to**

Hub: Via Safety Wrapper's `/tenant/backup-status` endpoint - **Alerts:** Medium severity if backup failed or stale

This gives us AI-powered backup monitoring without modifying the existing backup system. The IT Agent can also inspect backup status on demand: "When was my last backup? Were there any failures?"

1.16.3 16.3 Netcup VPS Snapshots (Decision #20)

Automated full-VPS snapshots via the Netcup SCP API: - **Frequency:** Daily, triggered by Hub cron job - **Retention:** 3 snapshots per tenant (rolling) - **Timing:** Staggered across tenants to avoid API rate limits - **Type:** Offline recommended (brief server pause, exportable) - **Cost:** Free to create and store (unlimited, stored on Netcup infrastructure, not VPS disk) - **Export:** One free export per server; additional exports have a fee

1.16.4 16.4 What's Backed Up Where

Data	Backup Method	Recovery
Tool databases (Chatwoot, Ghost, etc.)	Application-level DB dumps (existing)	Restore from dump
OpenClaw agent state (sessions, memory)	Included in VPS snapshot	Restore from snapshot
Safety Wrapper state (secrets registry, audit log)	Included in VPS snapshot	Restore from snapshot
Tool stack configs + docker-compose files	Included in VPS snapshot	Restore from snapshot
Customer files (Nextcloud data, MinIO objects)	Included in VPS snapshot	Restore from snapshot
Hub database (PostgreSQL)	Separate Hub backup strategy	Hub infrastructure concern

1.16.5 16.5 Future Considerations

- **Cross-region backup:** Periodic encrypted offsite backup for disaster recovery (post-launch)
- **Backup verification:** Automated restore tests on test VPS (post-launch)

1.17 17. Web Search & Fetch Tools (Decision #26)

OpenClaw natively supports web search and web fetch tools via configurable backends.

1.17.1 17.1 Available Backends

Tool	Backend Options	Description
`web_search`	Brave Search, Perplexity, Google Gemini	Search the web and return structured results
`web_fetch`	Built-in	Fetch and parse web pages, extract content

1.17.2 17.2 LetsBe Configuration

- **Enabled** for all agents with appropriate tool visibility (Layer 2)
- **Usage tracked** in the same token pool as LLM usage — web searches count against the monthly allotment
- **Overage billing** applies when pool is exceeded
- **Rate limiting** enforced by both OpenClaw (native) and Safety Wrapper (additional business logic)

1.17.3 17.3 Use Cases

- Marketing Agent researches competitors, industry trends
- Sales Agent looks up prospect companies, news
- IT Agent checks tool documentation, troubleshooting guides
- Secretary Agent verifies contact information, finds business hours

---

1.18 18. Channel Support (Decision #16)

OpenClaw natively supports multiple messaging channels — this is a major product differentiator. Channels are fallback for when the app isn't sufficient, but provide huge expansion potential.

1.18.1 18.1 Supported Channels

Channel	Key Features	LetsBe Use
WhatsApp	DM policies (pairing/allowlist/open), group support, media handling (50MB), read receipts	High priority — "talk to your AI from WhatsApp"

Channel	Key Features	LetsBe Use
Telegram	Bot token auth, message history, reply-to modes, SOCKS5 proxy, webhook support	High priority — popular for tech-savvy SMBs
Discord	Bot token, media (8MB), thread bindings with TTL, voice with TTS	Medium priority — for team collaboration
Slack	Socket mode, per-channel settings, slash commands, thread history	Medium priority — enterprise expansion
Google Chat	Service account auth, Pub/Sub integration	Low priority — niche

1.18.2 18.2 DM Security Model

All channels use the same security model: - **Pairing** (default): Unknown senders get time-limited pairing codes (1-hour expiry, max 3 pending) - **Allowlist**: Unknown senders blocked entirely - **Open**: Requires explicit `"*"` in allowlist - **Disabled**: Ignores inbound DMs

1.18.3 18.3 Message Queuing

Configurable modes for rapid multi-message input: `collect` (batch), `steer` (route), `followup`, `interrupt`. Debounce window (1-2s default) prevents partial-message processing.

1.18.4 18.4 Configuration

Channel setup is purely configuration — zero custom code per channel:

```
{
  "channels": {
    "whatsapp": {
      "enabled": true,
      "phoneNumberId": "${WHATSAPP_PHONE_ID}",
      "accessToken": "${WHATSAPP_TOKEN}",
      "dm": { "policy": "pairing" }
    },
    "telegram": {
      "enabled": true,
      "botToken": "${TELEGRAM_BOT_TOKEN}",
      "dm": { "policy": "allowlist" }
    }
  }
}
```

1.18.5 18.5 Secrets Limitation

Channels cannot use secure modals. Secrets UX falls back to one-time secure links (Section 13.6).

1.19 19. Testing Strategy

1.19.1 19.1 Priority Order

Priority	What	Why	When
P0	Secrets redaction unit tests	If this fails, customer secrets leak. Catastrophic.	Written FIRST (TDD).
P0	Command classification tests	If `rm -rf /` classifies as green, servers get destroyed.	Written alongside classifier.
P1	Autonomy level mapping tests	Verify Level 1/2/3 correctly gate the right command classes.	Written alongside autonomy config.
P1	Tool adapter integration tests	Verify each adapter authenticates, calls APIs, handles errors.	Per adapter, tested against Docker Compose instances.
P2	Hub ↔ Safety Wrapper protocol tests	Heartbeat, config sync, approval flow.	During Phase 2.
P2	Agent routing tests	Dispatcher routes to correct agent based on intent.	During Phase 3.
P3	End-to-end journey tests	Full user message → agent → tool → Safety Wrapper → execution → response.	Pre-launch.

1.19.2 19.2 Secrets Redaction Test Matrix

Must cover at minimum: - Known secrets from registry (exact match, partial match, substring) - Pattern-based detection (private keys, JWTs, bcrypt hashes, connection strings, high-entropy blobs) - Secrets embedded in JSON, YAML, logs, error messages - Unicode edge cases, base64-encoded, URL-encoded - False positive avoidance (don't redact the word "password", only actual values) - Performance: redaction must add <10ms latency to LLM calls

1.19.3 19.3 What We Don't Test

OpenClaw itself. It's upstream with its own test suite. We test our integration points only: plugin registration, hook behavior, config loading, and the model provider proxy.

1.20 20. OpenClaw Platform Capabilities (Full Reference)

This section documents everything OpenClaw provides natively — organized by capability area for quick reference during development.

1.20.1 20.1 Gateway Architecture

Single Node.js 22+ process. JSON5 config with env var substitution. All state on disk (JSON/JSONL). WebSocket + HTTP multiplexed on port 18789. Token auth recommended. Rate limiting built in. Multi-instance support.

1.20.2 20.2 Session Management

Two-layer persistence: mutable session store + append-only transcripts with tree structure. Scoping: per-sender (default), per-channel, per-peer. Auto-reset daily at 4:00 AM. Idle expiry configurable. Compaction with memory flush. Context pruning (cache-ttl mode). Maintenance: 30-day stale prune, 500 entry cap, 10MB rotation.

1.20.3 20.3 Tool System

Exec tool: Three environments (sandbox/gateway/node). Security levels: deny/allowlist/full. PATH override rejection. Loader variable blocking. 1800s timeout. Background execution.

Browser tool: CDP + Playwright. Three profiles (openclaw-managed/remote-CDP/extension-relay). Full page interaction (click, type, drag, scroll, evaluate JS). File operations. Navigation with compound waits. State management (cookies, localStorage). SSRF protection.

Tool policies: Profile-based (minimal/coding/messaging/full). Per-agent allow/deny. Loop detection (30-call window, hard stop at 30).

Plugins: TypeScript modules loaded at runtime. Register: RPC methods, HTTP handlers, tools, CLI commands, background services, skills, auto-reply. Discovery: config

paths → workspace → global → bundled. Security: path traversal blocking, ownership verification, optional allowlist.

1.20.4 20.4 Automation System

Cron: Built-in scheduler. Main session or isolated. Schedule types: at (one-shot), every (interval), cron (5-field with timezone). Delivery: announce/webhook/none. Exponential backoff on failures. Top-of-hour stagger.

Hooks: Event-driven scripts. Events: command:new/reset/stop, agent:bootstrap, gateway:startup, message:received/sent, tool_result_persist. Bundled: session-memory, bootstrap-extra-files, command-logger, boot-md. Custom: HOOK.md + handler.ts.

Webhooks: POST /hooks/wake (system event), POST /hooks/agent (isolated turn), POST /hooks/ (custom). Bearer token auth. Rate limiting. Payload safety wrapping.

1.20.5 20.5 Channel System

WhatsApp, Telegram, Discord, Slack, Google Chat. DM security (pairing/allowlist/open/disabled). Message queuing (collect/steer/followup/interrupt). Debounce. TTS via ElevenLabs/OpenAI.

1.20.6 20.6 Model Provider System

Supports: OpenAI, Anthropic, Qwen, Mistral, OpenRouter, LiteLLM, Venice AI, Together AI, Amazon Bedrock, Hugging Face, Ollama, vLLM, and more. Fallback chains. Provider-specific settings. Separate image model. Auth profile rotation before model fallback.

1.20.7 20.7 Prompt Caching

`cacheRetention`: none/short(5min)/long(1hr). Supported on Anthropic (direct + Bedrock + OpenRouter). Heartbeat keep-warm. Context pruning to prevent cache bloat.

1.20.8 20.8 Container Sandboxing

Modes: off/non-main/all. Scope: agent/session/shared. Workspace access: none/ro/rw. Docker settings: custom image, read-only root, network isolation, capability dropping, memory/CPU/PID limits. Idle pruning (24h default).

1.20.9 20.9 Streaming & UX

Block streaming with configurable min chars (800 default). Human delay simulation. Typing indicators: never/instant/thinking/message.

1.20.10 20.10 Security

Token auth (auto-generated if not configured). Loopback binding. File permissions (600/700). Security audit tool (`openclaw security audit --deep --fix`). Operator trust model (single-operator per Gateway — matches our 1-customer-1-VPS model).

1.20.11 20.11 Logging & Diagnostics

Levels: debug/info/warn/error. File + console output. Sensitive data redaction in logs. Canvas host for agent-generated HTML/reports.

1.20.12 20.12 Capability Matrix — Build vs. Buy

Capability	Native?	Custom Build?
Multi-agent management	✅ Native	No
Agent-to-agent communication	✅ Native	No — enable + configure
Memory & search	✅ Native	No — configure extraPaths
Session management	✅ Native	No
Scheduled tasks (cron)	✅ Native	No
Event-driven hooks	✅ Native	Minimal — custom hooks
Webhook ingress	✅ Native	No — configure endpoints
Command execution	✅ Native	Layer Safety Wrapper on top
Browser automation	✅ Native	No (Decision #14)
Tool allow/deny	✅ Native	Layer Safety Wrapper on top
Loop detection	✅ Native	No — enable in config
Prompt caching	✅ Native	No — configure
Streaming	✅ Native	No — wire to app WebSocket
TTS	✅ Native	No — configure provider
Multi-channel messaging	✅ Native	No — configure per channel
Model failover	✅ Native	No — configure chains
Container sandboxing	✅ Native	No — configure per agent
Security audit	✅ Native	No — run during provisioning
Plugin system	✅ Native	Integration path for adapters
Token tracking	✅ Native	Capture for billing
Sub-agents	✅ Native	Track for billing
Command queue	✅ Native	No — lane-aware FIFO
Compaction	✅ Native	No — auto-manages context
Web search/fetch	✅ Native	Enable + meter
Secrets redaction	❌	**YES — Safety Wrapper core IP**
Command gating (5-tier)	❌	**YES — Safety Wrapper core IP**
Credential injection proxy	❌	**YES — Safety Wrapper core IP**
Hub communication	❌	**YES — Safety Wrapper**
Token metering for billing	❌	**YES — Safety Wrapper + Hub**
Tool API adapters (24)	❌	**YES — custom adapters**
Customer-facing portal	❌	**YES — Hub retooling**

Capability	Native?	Custom Build?
Mobile app	❌	**YES — new build (React Native)**
Billing system	❌	**YES — Hub + Stripe**

1.21 21. Migration Path

1.21.1 Phase 1: Foundation (Weeks 1-4)

- Build Safety Wrapper skeleton (HTTP proxy, tool server, Hub client)
- Port sysadmin agent executors as Safety Wrapper tools (shell, docker, file, env)
- Implement Secrets Registry + outbound redaction (Layers 1-3)
- Implement command classification + gating (green/yellow/red/critical_red)
- Set up OpenClaw as upstream dependency (pin to release tag)
- Configure OpenClaw to route through Safety Wrapper
- Implement token metering capture
- Write P0 tests (secrets redaction, command classification)

1.21.2 Phase 2: Integration (Weeks 5-8)

- Configure OpenClaw native browser tool with Safety Wrapper gating
- Build adapter framework (BaseToolAdapter)
- Build P0 tool API adapters (Portainer, Nextcloud, Chatwoot, Ghost, Cal.com, Stalwart Mail)
- Implement function-call proxy (Layer 4 — credential injection)
- Hub: Add `/api/v1/tenant/*` endpoints (replace orchestrator phone-home)
- Hub: Add token metering/billing endpoints
- Hub: Remove inline SSH provisioning path
- Hub: Add AgentConfig, CommandApproval, TokenUsageBucket models
- Implement OpenClaw hooks (message:received, tool_result_persist, gateway:startup, agent:bootstrap)
- Configure four-layer access control
- Disable Elevated Mode globally

1.21.3 Phase 3: Customer Experience (Weeks 9-12)

- Hub: Build customer-facing portal (`/customer` routes)
- Hub: DNS automation integration (Cloudflare/Entri)
- Hub: Command approval queue + push notifications
- Build P1 tool API adapters (Odoo, Listmonk, NocoDB, n8n, Umami, Keycloak)
- Mobile app: initial build (React Native)
- Provisioner: Update to deploy new stack (OpenClaw + Safety Wrapper)
- Provisioner: Migrate Playwright initial-setup scenarios to OpenClaw browser
- Deploy LetsBe template skills per agent role

- Enable web search/fetch with usage metering
- Configure channel support (WhatsApp, Telegram)

1.21.4 Phase 4: Polish & Launch (Weeks 13-16)

- Build P2 + P3 tool API adapters
- Integration testing across full stack
- Security audit (secrets redaction, command gating, permissions, `openclaw security audit --deep`)
- Performance optimization (prompt caching, token usage, heartbeat keep-warm)
- Provisioner: Add tests
- Backup monitoring via OpenClaw cron job
- Launch founding member program

1.22 22. Decisions Log

#	Decision	Date	Rationale
1	OpenClaw treated as upstream dependency, not modified fork	2026-02-25	Enables pulling upstream updates without merge conflicts. All LetsBe logic in Safety Wrapper.
2	Safety Wrapper as OpenClaw extension (updated v1.2 — was separate proxy)	2026-02-26	OpenClaw's typed plugin hooks (`before_tool_call`, etc.) provide interception without process overhead. OpenClaw stays vanilla — extension is a separate installable package. Only the secrets redaction proxy remains as a separate thin process for transport-layer security.
3	Deprecate orchestrator — absorb into OpenClaw + Safety Wrapper	2026-02-25	Saves ~256MB+ RAM per tenant. OpenClaw handles agent management natively.
4	Deprecate sysadmin agent — port capabilities as Safety Wrapper tools	2026-02-25	Executor pattern preserved. Security patterns become pre-execution hooks.
5	SQLite for all on-server state (no per-tenant PostgreSQL)	2026-02-25	Embedded, zero-config, no extra container.
6	Per-agent tool permissions configurable	2026-02-25	Same operation gated differently per agent role. Sensible defaults, user-customizable.

#	Decision	Date	Rationale
7	OpenClaw routes LLM calls through Safety Wrapper (not direct to OpenRouter)	2026-02-25	Enables transparent secrets redaction without modifying OpenClaw.
8	~~MCP Browser kept as separate sidecar~~ **SUPERSEDED by #14**	2026-02-25	Replaced after OpenClaw findings.
9	Provisioner renamed from ansible-runner	2026-02-25	Name accuracy.
10	Hub inline SSH provisioning path removed	2026-02-25	Docker-based provisioner is canonical. SSH path adds confusion and attack surface.
11	Safety Wrapper written in Node.js	2026-02-25	Same runtime as OpenClaw. One ecosystem.
12	Hybrid deployment: tool adapters as OpenClaw plugin, secrets proxy as separate process	2026-02-25	Best of both: one process for tools, process isolation for security-critical redaction.
13	Use OpenClaw hooks for Safety Wrapper integration	2026-02-25	`message:received`, `tool_result_persist`, `gateway:startup`, `agent:bootstrap`.
14	Use OpenClaw native browser tool, deprecate MCP Browser sidecar	2026-02-25	More capable, saves ~256MB RAM per tenant, fewer processes.
15	Hub relay for mobile app communication (App → Hub → Tenant Server)	2026-02-25	Simpler, more secure, Hub can show offline messages.
16	Native WhatsApp/Telegram/Discord/Slack channel support at launch	2026-02-25	Zero custom code — configuration only. Fallback and product differentiator.
17	Configurable AI Autonomy Levels (3 tiers)	2026-02-25	Per-agent configurable. Secrets always redacted at every level.
18	One customer = one VPS, permanently	2026-02-25	Simplifies everything. Revisit only if demand proves otherwise.
19	Severity-based error alerting with auto-recovery	2026-02-25	Three-strike escalation for soft failures. Immediate alert for hard failures.
20	Daily Netcup snapshots, retain 3	2026-02-25	Free to create/store. Offline recommended. One free export per server.
21	Secrets UX via side-channel modals	2026-02-25	Secrets never pass through AI. SECRET_REF tokens only.

#	Decision	Date	Rationale
22	Four-layer access control model, Elevated Mode disabled	2026-02-25	Sandbox → Tool Policy → Command Gating → Secrets Redaction. Elevated Mode disabled globally.
23	Playwright scenarios migrate to OpenClaw native browser	2026-02-25	8 initial-setup scenarios run via OpenClaw's browser during provisioning.
24	Flat token pool billing with overage	2026-02-25	Simple. Monthly allotment. Overage via Stripe. Web tools in same pool.
25	OpenAI-compatible API locked down	2026-02-25	Not exposed to customers. Local-only for Safety Wrapper communication.
26	Web search/fetch enabled with usage limits	2026-02-25	Brave/Perplexity/Gemini backends. Counts against token pool. Pay-per-extra.
27	Backup hybrid strategy (Option C)	2026-02-25	Keep existing backups.sh as safety net. OpenClaw cron monitors via backup-status.json. Reports to Hub.
28	Single gateway per VPS for v1	2026-02-25	Multi-gateway revisited for enterprise tier if needed.
29	LetsBe template skills + open-ended user customization	2026-02-25	Pre-built skills per agent role. Users can customize freely.
30	External Communications Gate — independent of autonomy levels	2026-02-25	External-facing operations (publish, send, reply) gated by default for all agents at all levels. User explicitly unlocks per agent/tool. Product principle: protect the user's external relationships.
31	Dispatcher Agent as first-class default component	2026-02-25	Every tenant gets a Dispatcher Agent that routes user messages to specialist agents, decomposes multi-step workflows, and delivers morning briefings. Users can also chat directly with specific agents.

#	Decision	Date	Rationale
32	Dynamic tool installation from curated catalog	2026-02-25	IT Agent can deploy additional open-source tools from a curated, version-pinned catalog. Red-tier gated. Resource checks before deploy. Extends the platform beyond the core 30 tools.
33	Two-tier model strategy: 3 included presets + premium pay-as-you-go	2026-02-25	Basic: “Basic Tasks” (Gemini Flash), “Balanced” (DeepSeek V3.2), “Complex” (GLM 5/MiniMax). Premium: Claude Opus/Sonnet, GPT 5.2, Gemini Pro — credit card required, sliding markup.
34	Founding member program: 2× token allotment for 12 months	2026-02-25	First 50-100 customers. “Double the AI” — all tiers margin-positive. Multiplier on base tier allotment. FoundingMember Prisma model with expiry date. Updated from 3× (margin-negative on higher tiers) to 2× on 2026-02-26.
35	Threshold-based sliding markup on premium models	2026-02-25	More expensive base cost → lower markup (25% under $1/M → 8% over $15/M). Encourages adoption. Configurable in Hub settings.

1.23 23. Open Questions

All original questions resolved. Remaining open items are product/UX decisions to be developed collaboratively:

#	Question	Status
1	Basic/Advanced UX mode implementation details	To be developed — UI/UX design session needed
2	First-hour quick wins per business type	To be developed — needs business type template library
3	Website design and onboarding flow	To be developed — after branding redesign

#	Question	Status
4	Netcup domain reselling API integration	To be developed — integration spec needed
5	Specific token pool sizes per tier	To be finalized pre-launch — see Pricing Model
6	Extended tool catalog contents (beyond core 30)	To be developed — initial catalog for launch

#	Question	Resolution
1	Safety Wrapper language	**Decision #11: Node.js**
2	OpenClaw tool discovery	**Decision #12: Plugin registration + hooks**
3	Mobile app communication path	**Decision #15: Hub relay**
4	OpenClaw upstream breaking changes	**Operational runbook.** Pin to release tag, monthly review, staging test before rollout.
5	Backup monitoring	**Decision #20 + #27: Netcup snapshots + hybrid monitoring**
6	Multi-server per customer	**Decision #18: One VPS permanently**
7	Native browser vs. MCP Browser	**Decision #14: Native browser**
8	Safety Wrapper plugin vs. process	**Decision #2 (updated v1.2): Extension + thin secrets proxy**
9	OpenClaw hooks for integration	**Decision #13: Yes**
10	Native channel support	**Decision #16: Enabled at launch**
11	Netcup snapshot pricing	**Decision #20 updated: Free to create/store**
12	TOOLS.md permission schema	**Section 6.5: Two-layer config with concrete JSON schemas**

1.24 24. Document Lineage

Version	Date	Changes
0.1	2026-02-25	Initial architecture. Component disposition, Safety Wrapper design, tenant server architecture, migration path, security model, tool adapter strategy.
0.2	2026-02-25	Added OpenClaw Platform Capabilities (Section 18). Gateway, sessions, tools, automation, channels, providers, caching, sandboxing, streaming, security.
0.3	2026-02-25	Resolved all 10 open questions (Decisions #11-#20). Added AI Autonomy Levels, Error Handling, Testing Strategy, Backup Strategy.
0.4	2026-02-25	Added Secrets UX (Decision #21). Four credential flows, Safety Proxy API, secure modals, messaging fallback.
0.5	2026-02-25	Fixed section numbering throughout. No content changes.
0.6	2026-02-25	Consistency pass. Removed all stale MCP Browser references. Resolved all 12 open questions. Added concrete autonomy config schemas.
1.0	**2026-02-25**	**Comprehensive redraft.** Reorganized and deepened all sections. Added: Four-Layer Access Control Model (Decision #22), Billing & Token Metering (Decision #24), Skills & Extensibility (Decision #29), Web Search/Fetch (Decision #26), Channel Support detail, Backup Hybrid Strategy (Decision #27). Incorporated all OpenClaw capabilities. Maximum-depth Hub retooling with new API endpoints, Prisma models, and data flow. Added Decisions #22-#29.

Version	Date	Changes
1.1	**2026-02-25**	**Vision alignment update.** Added: External Communications Gate as independent product principle (Decision #30, Section 3.2.2 + 6.6). Dispatcher Agent as first-class default component (Decision #31, Section 10.3). Dynamic Tool Installation from curated catalog (Decision #32, Section 7.3). Two-tier model strategy with 3 included presets + premium pay-as-you-go (Decision #33, Section 11.1). Founding Member program with 3x multiplier (Decision #34, Section 11.4 + Prisma model). Threshold-based sliding markup (Decision #35, Section 11.1.3). Updated OpenClaw agent config to include Dispatcher. Updated open questions to reflect remaining product/UX work.

Version	Date	Changes
1.2	**2026-02-26**	**OpenClaw deep dive integration.** Safety Wrapper changed from separate proxy to OpenClaw extension with typed plugin hooks (updated Decision #2, Section 3.2). Removed `letsbe-safety-proxy` container — replaced by in-process extension + thin `letsbe-secrets-proxy`. OpenClaw container uses `--network host` for tool access via `127.0.0.1:30XX` (Section 3.5). Complete rewrite of Tool Integration Strategy (Section 7) — replaced 30+ individual tool adapters with tool registry + master skill + on-demand cheat sheets. Skills & Extensibility (Section 8) rewritten for token efficiency: structured tool registry (~2-3K tokens) always in context, cheat sheets loaded on-demand only. Added token efficiency strategy table and monitoring via `llm_output` hook. Updated open questions table.

End of Document