Secure Hybrid AI Development Hub — Comprehensive Technical Overview

Fail-Closed Local Broker for Hybrid AI Software Delivery

A client-readable and engineer-ready reference for governed planning, isolated implementation, model routing, research, credentials, quality gates, recovery, audit evidence, and controlled release delivery.

Python | Codex | Claude Code | VS Code | Ollama | SQLite | Typed Artifacts | Deterministic Policy | Evidence-Driven Verification

Open-source public technical edition · Aligned with GitHub main a9f45d6

Latest Status and Reading Guide — 14 October 2026

Evidence date: 24 September 2026. This October review compares the existing overview with the latest recorded project evidence; it adds no new engineering test, implementation, release or production-readiness claim.

Current boundary: experimental, pre-production, open-source and self-hostable. The published baseline provides the Python broker/CLI and project-local integrations. Phase 1 closure remains incomplete. Real-client operation, provider use, credentials and deployment require their separate gates and explicit authorization.

Latest recorded verification: 257 tests passed with one expected private-dossier skip in the clean public snapshot. Python 3.11–3.13 CI and the configured Gitleaks, Bandit and Semgrep checks passed for the September publication. See section 27 for exact revisions, scope and build evidence; section 23 retains the earlier 208-test checkpoint.

Open work: complete authority enforcement and test-data end-to-end acceptance, tighten the packaging build minimum, and finish configuration and supported-host validation. The browser/API, distributed deployment, million-connection capacity and uptime/recovery objectives remain planned or unproven.

Reading order: use this checkpoint and section 27 for evidence, section 32 for delivery gates, sections 28–30 for future interface/capacity/reliability plans, and sections 31–33 for operator responsibilities. All previous sections and appendices are retained below.

1. Executive Overview

The Secure Hybrid AI Development Hub is a fail-closed local control plane for software work performed with Codex, Claude Code, VS Code, terminal workflows, local Ollama models, and separately approved provider-backed workers. It replaces the idea of one unrestricted coding agent with a deterministic broker that owns scope, policy, task state, isolation, artifacts, verification evidence, and release decisions.

The intended product is designed to support single repositories, monorepos, polyrepo microservices, and mixed systems containing applications, services, infrastructure, shared libraries, and deployment units. A developer should ultimately be able to submit one outcome-oriented request while the hub discovers approved context, records an explicit task-specific worker selection, creates bounded workspaces, validates changes, runs quality gates, repairs within configured limits, records evidence, and ends in an explicit verified, paused, blocked, cancelled, or infrastructure-failure state. This end-to-end operating path is still under test-data verification.

The central security rule is separation of authority. Models may reason and propose; they do not grant themselves filesystem scope, network access, credentials, providers, production authority, or completion status. Every privileged transition is mediated by deterministic code and a typed record.

This document describes the intended completed operating state and the currently verified boundary. Phase 1 remains incomplete; the current public-source verification uses test data only. The earlier July checkpoint is preserved in section 23; the published baseline and newer evidence are in section 27. Live client onboarding, real provider transmission, real credentials, regulated data, staging, production, and deployment remain prohibited until their later evidence and exact per-project authorization exist.

Current implementation snapshot — scoped evidence, not completion

- Package baseline: secure-hybrid-hub 0.11.0; backend and public documentation are published on GitHub main at a9f45d6. This publication is not a new package release.
- Runtime: Python 3.11 or newer, standard library only.
- Target host environments: Linux/WSL2 is the reference for Linux isolation controls. macOS requires separate isolation validation; native Windows is unsupported.

- Source inventory: evolving during controlled development; use the reviewed repository revision rather than historical line or symbol counts.
- Test inventory: evolving; only deterministic results for the exact reviewed revision are authoritative.
- Historical authority-foundation checkpoint: scoped test gates passed before integration; the July Phase 1 closure task was cancelled before complete quality, review, or integration. The subsequent public-source test and publication evidence appears in section 27; Phase 1 closure is still unproven.
- Standing security guarantees: previously passed in scoped integrated checkpoints; they do not establish current Phase 1 closure.
- Evidence inventory: maintained through broker checkpoints and the approved dossier; current task state and validated evidence supersede historical counts.
- License: Apache-2.0.

2. Product Outcome

The product turns a request such as “add appointment cancellation and fully verify it” into a governed task lifecycle. The user may start from a terminal, Codex, Claude Code, or VS Code. The broker resolves the registered system, applies the strictest effective policy, prepares isolated workspaces, selects the approved implementation route, and gathers deterministic evidence before reporting an outcome.

The hub is designed for real engineering systems rather than isolated code snippets. It understands repository ownership, service dependencies, contracts, quality commands, environment boundaries, release ordering, and the difference between implementation, verification, staging validation, and production promotion.

Primary user outcomes

- Start work from a familiar developer surface without bypassing central policy.
- Avoid repeatedly explaining a large system to every new model session.
- Keep unrelated clients, repositories, caches, artifacts, and task state separated.
- Use local models for private high-volume implementation when appropriate.
- Use provider-backed workers only when the project profile and human approvals permit it.
- Isolate public internet research from private repository access.
- Use credentials through controlled capabilities without placing their values in model context.
- Produce reproducible evidence for tests, reviews, releases, deployment decisions, and recovery.
- Fail visibly when information, authority, quality, isolation, or infrastructure is insufficient.

3. Design Principles

3.1 Deterministic authority

The policy broker is authoritative. A model response is an input to validation, never an authorization record. Scope, state transitions, artifact types, command execution, egress, provider selection, approvals, and completion are decided by deterministic code.

3.2 Credentials are capabilities

Credentials are referenced by approved capability names. A confined runner may use a credential for one named operation; the model receives neither the secret value nor ambient access to the secret store.

3.3 Private coding and internet research are separate

A worker may receive approved repository context or public internet access, never both at once. Research workers cannot read repositories. Coding workers cannot browse arbitrary public pages. Research output returns as labeled evidence rather than raw authority.

3.4 Strictest policy wins

Managed, client, system, repository, component, environment, and task rules compose by intersection. A lower layer may add restrictions or gates, but it cannot weaken a higher layer.

3.5 Evidence establishes completion

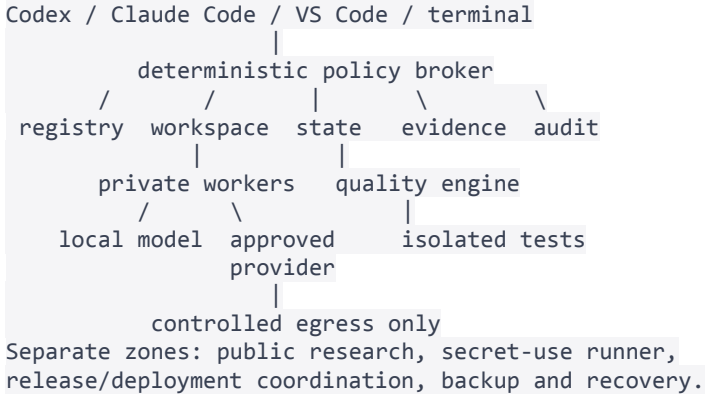
A task reaches a verified state only when required quality commands, scanners, contract checks, review gates, and evidence checks pass. Model confidence, a clean-looking diff, or an optimistic explanation is not verification.

3.6 Recovery is part of correctness

External actions, leases, checkpoints, approvals, provider calls, releases, and deployments are recorded so interrupted work can resume idempotently. A recovery instruction is considered part of the feature and must itself be testable.

4. Architecture

The hub is arranged as a collection of deterministic trust zones around a transactional core.



4.1 Broker core

The broker provides typed schemas, SQLite-backed transactional state, content-addressed artifacts, atomic writes, safe path handling, task transitions, leases, approvals, audit events, and dossier checkpoints. Its public entry point is the `hub` CLI.

4.2 Registry

The registry stores explicitly approved systems, repositories, components, environments, topology relationships, policy profiles, quality commands, provider profiles, research domains, secret capability names, and operational restrictions.

4.3 Worker boundaries

Private coding workers receive only the broker-selected task context and deliverable. Public research workers receive approved research questions and domains but no repository. Secret-use runners receive a capability and bounded input but no general model authority. Provider-backed workers receive only policy-approved outbound context.

4.4 Evidence plane

Every important transition produces structured evidence: discovery, planning, workspace preparation, implementation attempts, tests, repairs, review, release, deployment, rollback, cancellation, and closure. Evidence is sanitized before persistence and linked to task and dossier state.

5. Supported Project Topologies

5.1 Single repository

One repository contains one application or service. The task receives one bounded workspace, one ownership scope, and the quality commands approved for that repository.

5.2 Monorepo

A monorepo may contain multiple packages, applications, or services. Component paths, dependency relationships, ownership, commands, data classifications, and deployment units are registered. Affected-component analysis selects focused gates, while final verification can require broader regression checks.

5.3 Polyrepo microservices

A system may span independent repositories. The registry models provider-consumer APIs, events, schemas, shared libraries, database ownership, infrastructure dependencies, and deployment order. A cross-service task receives one workspace per affected repository and produces a release manifest that ties the independent commits and artifacts together.

5.4 Hybrid systems

Hybrid topology connects monorepos, standalone services, mobile/web clients, infrastructure, and shared contracts in one system graph without granting blanket access to every repository.

5.5 Client separation

Unrelated clients do not share repository roots, task state, caches, research indexes, workspaces, provider bundles, secret capabilities, or dossier projections. Overlapping roots and ambiguous parent-drive registration are rejected.

6. Explicit Registration and Opt-In

The hub never silently intercepts every project. A system is registered through an explicit initialization workflow, followed by deterministic discovery and human approval.

```
hub --runtime RUNTIME system init --id example-system --client example-owner --name "Example system"
--root PROJECT_ROOT --profile standard --purpose "Validation run"
hub --runtime RUNTIME system discover example-system
hub --runtime RUNTIME system approve example-system --approver OWNER
```

Registration records the system boundary, repository roots, components, languages, safe commands, service dependencies, environments, policy profiles, provider permissions, research domains, capability names, quality gates, deployment order, rollback strategy, retention, and approval roles. Secret values are not part of registration.

Discovery output is a proposal, not authority. A responsible human approves the first system profile before any live provider, credential, or deployment route can become eligible.

Project-local integration

The integration installer can add repository-local Codex skills, Claude skills, VS Code tasks, and a project marker. It does not rewrite global model configuration or enroll sibling projects.

7. Policy Model

The policy system begins with managed rules and composes the profiles selected for the registered system.

```
| Profile | Typical control emphasis |
| Standard | Approved source scope and ordinary quality gates |
| Confidential | Narrow egress, stronger approval, shorter retention |
| Healthcare | Local-first handling for health and regulated data |
| Legal | Matter separation, privilege protection, citation review |
| GDPR | Data minimization, transfer records, retention and deletion |
| Financial | Transaction integrity, reconciliation, segregation of duties |
| High secret | Abstract requirements or no provider transmission |
| Production critical | Staging, canary, rollback, protected promotion |
| Public open source | Broader public-code review with credentials still excluded |
```

Policy composition follows deny-overrides-allow, narrowest-scope-wins, highest-classification-wins, and all-required-gates-remain rules. Exceptions are explicit, time-bounded, attributable, and cannot be created by a model or repository file.

Classification-based egress

The live transmitting worker boundary checks the registered system profiles before any provider process starts, any key reference is opened, or any spend is incurred. Restricted profiles fail closed. Permissive profiles still require an approved provider configuration and task-level eligibility.

8. Task Lifecycle and State Machine

The broker represents a task as a validated state machine rather than a free-form conversation.

```
submitted
  -> discovery / context resolution
  -> planning
  -> workspace ready
  -> local or approved worker implementation
  -> targeted quality
  -> bounded repair when needed
  -> full verification
  -> release evidence
  -> optional staging / controlled promotion
  -> verified or another explicit terminal state
```

Completion states

- `VERIFIED` — all required implementation, quality, and review evidence passed.
- `VERIFIED_STAGING` — verified and validated in staging; production was not promoted.
- `PAUSED_INPUT` — required business or technical information is missing.
- `PAUSED_AUTH` — an approved login or credential provisioning action is required.
- `PAUSED_APPROVAL` — a high-impact or new-boundary action awaits human approval.
- `BLOCKED_QUALITY` — bounded repair could not satisfy required evidence.
- `BLOCKED_POLICY` — classification, scanner, provider, contract, or permission policy refused the action.

- `FAILED_INFRA` — infrastructure failed independently of code correctness and the task can be resumed through the defined recovery path.
- `CANCELLED` — the user stopped work and recoverable evidence was retained.

Illegal transitions are rejected. Terminal reporting releases owned workspaces and records the outcome. Resumable failure states return through approved recovery targets rather than rewriting history.

9. Planning and Guided Orchestration

Planning turns the user outcome into typed, bounded deliverables. A plan identifies affected repositories, exact target files, contracts, quality gates, research needs, provider eligibility, and release implications.

High-capability planning may decompose a large outcome into repository-specific packets. Each packet names one broker-selected file and expected behavior. The implementation worker cannot choose arbitrary paths or broaden the task.

Guided orchestration applies accepted file operations through the broker, checks source hashes, rejects stale or unauthorized edits, runs targeted gates, returns sanitized diagnostics, and allows a bounded number of repair attempts. Duplicate output and no-progress loops end safely rather than consuming unlimited time or provider budget.

Example task flow

1. Resolve the registered system and effective policy.
2. Produce a deterministic plan and affected-component set.
3. Create or re-acquire task workspaces and leases.
4. Select the pinned permitted worker.
5. Generate one deliverable at a time.
6. Parse and validate the typed result.
7. Apply through the broker with stale-hash and path checks.
8. Run packet-specific quality gates.
9. Repair from sanitized diagnostics within limits.
10. Run final cross-component verification.
11. Record release and dossier evidence.
12. Report one explicit completion state.

10. Model Selection and Routing

Model routing is project-scoped and provider-neutral. A project can pin a local Ollama model, a subscription CLI worker, or a metered HTTPS API worker from an approved catalog. Selection includes a capability probe, policy compatibility, context and output limits, quality requirements, and availability.

The hub does not silently escalate from local to paid or cloud execution. If the selected worker is unavailable or incapable, the task blocks or pauses and asks the human. This preserves cost, classification, and provider decisions.

Worker categories

- `codex-local` — Codex-shaped local adapter using the approved local Ollama transport.
- `claude-local` — Claude-shaped local adapter using the approved local Ollama transport.
- Subscription CLI — a logged-in provider CLI executed with tools disabled or a read-only sandbox, a minimal environment, and audited prompt metadata.
- Vendor API — an approved HTTPS adapter with explicit endpoint family, pricing, token limits, spend ceiling, key-file reference, and live-enable approval.

Bounded fallback

Fallback is deterministic, ordered, and limited to pre-approved alternatives. A stronger model never receives broader permissions merely because the first model failed.

11. Local Coding Workers

Local coding workers are intended for private, high-volume implementation. They receive the exact deliverable path, bounded context, relevant requirements, approved research guidance, and a structured output contract.

They cannot select arbitrary repository files, execute arbitrary commands, browse the internet, read secret storage, access unrelated projects, or promote their own output. The broker parses the response and applies accepted operations.

Local transport uses bounded loopback communication with explicit timeouts, output-token limits, and stop conditions. Malformed, truncated, repeated, or out-of-contract output is rejected and may enter bounded repair.

12. Subscription and API Workers

Provider-backed workers operate only after classification policy and provider approval both permit transmission. Outbound context is minimized and scanned. Processes run in an empty scratch context with an environment allowlist and no ambient provider credentials.

API workers read an approved key reference only at call time. The key value is never stored in task state, audit events, prompts, dossier output, or exported documentation.

Cost controls

- Explicit input and output token prices.
- Hard per-task spend ceiling.
- Worst-case pre-egress cost check.
- Framing-token allowance.
- Metering of every billable successful response before interpreting content.
- Worst-case accounting when usage is missing or impossible.
- Post-call ceiling backstop.
- Fail-closed behavior when metering cannot be recorded.
- No automatic paid fallback.

Transport and content failures are distinguished. A request that did not reach a provider is not charged as if it did; a response that was generated but could not be interpreted is still accounted for before the task fails or retries.

13. Isolated Research

Research is a separate system with its own policy, cache, provenance, and worker boundary. It can use approved official domains, offline evidence, a local cache/index, direct bounded HTTPS retrieval, or an optional self-hosted discovery service.

Research workers have no repository, credential store, browser profile, clipboard, container socket, or private task workspace. They receive a question and an allowed domain set. Retrieved content is labeled untrusted, size-limited, MIME-checked, redirect-controlled, SSRF-protected, and evaluated for prompt-injection indicators.

Research results return as provenance records and concise findings. Raw fetched pages do not become hidden instructions to the coding worker.

14. Secret Capabilities

The hub separates a secret value from the operation that needs it. Registration and planning use capability names; the confined runner resolves the value only inside the approved operation.

Secret-use guarantees

- Models do not receive secret values.
- General coding workers cannot browse the secret store.
- Values are not inherited through ambient environment variables.
- Exact and encoded forms are redacted from evidence.
- Generated code containing credential-like literals is refused.
- Approved placeholder and reference conventions are documented to the worker.
- A failure to start the isolation boundary is an infrastructure failure, not a false security verdict.

Test fixtures test these controls without using real client credentials.

15. Egress and Data Loss Prevention

When approved content must leave the private zone, the broker constructs an explicit egress bundle. The bundle contains only selected files or safe projections, a manifest, hashes, classification labels, and the approvals required for its intended route.

Scanners inspect credential assignments, private key material, connection strings, regulated-record patterns, personal information indicators, privilege markers, prohibited paths, encoded values, Unicode tricks, and unexpected binary or oversized content.

Unknown or ambiguous content is refused. Passing scanners means the configured controls found no prohibited pattern; it is not proof that arbitrary content is safe. Structural minimization and human approval remain necessary.

16. Workspaces, Paths, and Leases

Each task works in bounded Git worktrees or multi-repository workspaces. Canonical path validation rejects traversal, unexpected roots, sibling-client access, case confusion, and unsafe symbolic links.

One-writer leases prevent two tasks from editing the same registered repository at the same time. Host-global resources such as local inference or subscription CLIs also use named leases. Terminal reporting releases owned resources, while recovery re-acquires required repository leases before implementation resumes.

Atomic writes and content hashes prevent a model result from overwriting a file that changed after planning. A stale proposal is rejected and must be reconciled through a new bounded attempt.

17. Quality Engine

Quality commands are registered as explicit argument arrays and approved by a human. Models do not invent shell commands or weaken the gate list. The runner chooses targeted and full gates from the affected topology and effective policy.

Verification layers

- Syntax and compilation.
- Formatting and linting.
- Static typing.
- Unit and integration tests.
- Contract and schema compatibility.
- Migration safety.
- Secret and regulated-data scanning.
- Test-integrity and anti-weakening checks.
- Security analysis.
- Packaging and release evidence.
- Cross-component regression gates.

Commands execute in disposable snapshots with filesystem, process, and network isolation where supported. Diagnostics are sanitized before returning to a repair worker. The task cannot become verified by deleting tests, skipping required gates, changing an approved command, or editing evidence.

18. Audit, Artifacts, and Dossier Evidence

The hub records sanitized hash-chained audit events. Each event identifies the task, action, result, evidence reference, and time without storing restricted payloads. An external audit anchor can preserve a trusted chain head so a consistently rewritten local chain is detectable.

Artifacts are content-addressed and schema-validated. Important outputs include discovery reports, plans, research provenance, implementation attempts, quality results, egress manifests, release manifests, deployment reports, rollback evidence, and final reports.

The hierarchical dossier stores system, repository, component, environment, and task facts with versions and proposals. Models can propose updates but cannot silently change approved architecture, policy, ownership, compliance, deployment, or exception facts. Every completed implementation phase creates a validated checkpoint.

19. Release and Deployment Coordination

Release coordination ties together verified source revisions, artifacts, contracts, migration order, and deployment units. Polyrepo releases use a manifest because independent repositories cannot share one atomic commit.

Deployment is credential-free from the model perspective. The broker prepares a controlled request for an approved CI/CD adapter, records staging evidence, and requires explicit human approval for production. Approval records are scoped, expiring, and single-use.

Deployment progression

```
verified source
  -> immutable release manifest
  -> staging deployment
  -> staging health and acceptance gates
  -> explicit production approval
  -> canary or progressive rollout
  -> health decision
  -> complete rollout or rollback
```

Rollback is a first-class operation with its own evidence. Partial microservice failure, incompatible contracts, failed health gates, and expired approvals stop progression.

20. Operational Hardening

The operations surface supports offline SBOM inventory, license review status, retention and deletion jobs, access review, exception expiry, backup, backup verification, restore drills, SQLite integrity checks, security evaluation, and incident response runbooks.

Runtime directories, caches, artifacts, and evidence stores have distinct retention and recovery rules. Backups are useful only when verification and restore are exercised, so both are represented as explicit operations.

Self-hosting position

The core package has no third-party runtime dependencies. Local model hosting remains separate from broker authority. Model installation, model downloads, public research services, and live provider configuration are optional operational choices requiring their own approval and capacity planning.

21. CLI and Developer Surfaces

The `hub` CLI provides the authoritative local interface. Repository-local Codex and Claude skills and VS Code tasks call into the same broker rather than re-implementing policy.

Core command groups

```
hub init-runtime
hub system init|discover|approve|disable|show
hub dossier show|export-safe|proposals|propose|decide
hub run|plan|status|cancel|resume
hub local preflight|invoke|invoke-file
hub workspace create|leases
hub research propose|approve|fetch|discover|ingest-offline|search-cache|get|resolve
hub secret propose|approve|show|run
hub egress build|approve|show
hub quality propose|approve|show
hub test|verify|inspect-egress
hub provider propose|approve|show
hub cloud preflight
hub release-manifest|deploy|promote|emergency-stop
hub operations
access-review|security-evaluation|backup|verify-backup|restore-backup|sbom|retention|expire-exceptions
hub integrations install
hub modifier propose|approve|show|list
```

Command output is structured for both human and automation use. A conversational skill may summarize status, but the broker's structured state is authoritative.

22. Security and Privacy Control Map

```
| Risk | Primary controls |
| Cross-client data mixing | Exact root registration, client-separated state/cache/workspaces, safe
projections |
| Repository exfiltration | Research/coding separation, classification egress veto, explicit bundles |
| Credential disclosure | Named capabilities, confined runner, no ambient key environment, redaction |
| Prompt injection | Untrusted research labeling, no repository access, typed evidence |
| Unauthorized provider use | Approved profile, live-enable gate, classification check, task eligibility
|
| Unbounded paid usage | Explicit prices, pre-call ceiling, billable-response metering, no auto fallback
|
| Test weakening | Approved command registry, test-integrity checks, immutable evidence |
| Concurrent writers | Repository and resource leases |
| Unsafe paths | Canonical roots, traversal/symlink/sibling rejection |
| Audit rewriting | Hash chain plus external anchor |
| Unsafe production change | Staging, expiring approval, canary, health gate, rollback |
| Interrupted task | Transactional state, checkpoints, idempotent resume, terminal cleanup |
```

The hub supplies technical controls and evidence. It does not claim automatic legal certification. HIPAA, GDPR, financial, legal-professional, contractual, and client-specific obligations require an accountable operational process around the product.

23. Current Verification Evidence

The historical integrated authority-foundation source passed its scoped test gates before integration. The July Phase 1 closure task was cancelled on 30 July 2026 before complete quality, independent review, integration, release, deployment, or push. Preserve the results below as historical evidence; section 27 records the subsequent GitHub publication and current public-source verification.

Historical integrated suite — not current Phase 1 closure evidence

```
208 passed
0 failed
0 skipped
```

The suite covers policy, topology, workspaces, leases, local routing, provider routes, spend controls, classification egress, guided orchestration, recovery, research, secrets, DLP, quality isolation, release, deployment, operations, and adversarial boundaries.

Standing security guarantees

Four named guarantee groups execute as 14 tests:

1. Codex and Claude worker routes operate without receiving secret values.
2. Restricted classification profiles have no provider-egress loophole, while explicitly permissive profiles remain usable through approved routes.
3. Model-produced code is gated rather than trusted, including credential-like literal refusal.
4. Secret locations and placeholder contracts remain confined and visible to the correct deterministic components.

Evidence interpretation

These historical results establish only the tested implementation state of their reviewed revision. They do not establish current Phase 1 closure, authorize a real provider, prove every operating system configuration, certify every future model, or replace per-client production acceptance.

24. Failure Modes and Recovery

Worker unavailable

The task pauses or fails as infrastructure according to the configured route. It does not silently choose an unapproved paid or cloud provider.

Worker output malformed or incomplete

The typed parser rejects the result. A bounded repair may receive sanitized diagnostics. Repeated failure becomes `BLOCKED_QUALITY` without fabricating artifacts.

Resource conflict

The task ends in a recoverable infrastructure state, releases owned workspace leases through final reporting, identifies the conflicting resource, and can resume after the competing operation finishes.

Authorization missing

The task pauses or blocks before provider start, credential access, spend, or deployment. The operator performs the approved external action and resumes through persisted state.

Scanner or isolation unavailable

The protected operation does not proceed. Missing evidence is reported as an infrastructure or policy failure, never converted into a passing result.

Quality gate fails

The broker returns sanitized, bounded diagnostics to the permitted repair route. If required evidence still fails after configured attempts, the task stops with the failure preserved.

Deployment health fails

Progressive promotion stops and rollback executes according to the approved deployment plan. The final report distinguishes verified source, staging evidence, production state, and rollback outcome.

Process interruption

Transactional state, checkpoints, artifact hashes, and leases allow the operator to inspect and resume safely. External actions are designed to be idempotent or guarded by unique approvals and manifests.

25. Installation and Verification

Requirements

- Linux/WSL2 reference environment. macOS requires separate isolation validation; native Windows is unsupported.
- Python 3.11 or newer.
- SQLite 3.38 or newer through the Python standard library.
- Git for worktree-based tasks.
- Optional local Ollama or separately approved provider tooling.

Local verification

```
cd hybrid-hub
PYTHONPATH=src python3 -m pytest tests -q
PYTHONPATH=src python3 -m pytest tests/test_standing_guarantees.py -v
```

First controlled setup

1. Initialize a dedicated runtime.
2. Register one test system for the current development boundary; real-client onboarding requires the later authorization and acceptance gates.
3. Review discovery output and policy composition.
4. Approve only the necessary repositories and commands.
5. Select and probe one approved worker route.
6. Run a test task through verified status.
7. Exercise cancellation, resume, backup, restore, and emergency stop.
8. Enable research, provider, secret, or deployment routes only after their separate approval records exist.

26. Maintainer Handoff and Release Discipline

Maintainers should begin with current executable evidence rather than historical prose. Verify repository status, run the full suite, run the standing guarantees, inspect the newest dossier checkpoint, and read the authoritative build plan before changing policy or recovery behavior.

Change rules

- Do not weaken policy, scanners, tests, checkpoints, or acceptance criteria to obtain a green result.
- Pair security-boundary changes with both allowed and denied behavioral tests.
- Exercise the recovery instruction a user will actually follow.
- Keep one authoritative definition for policy-sensitive values.
- Treat billing, transmission, credential use, and production promotion as explicit state transitions.
- Preserve client separation and exact root scope.
- Record new evidence without placing secret values or private client material in documentation.
- Require a validated dossier checkpoint for every completed implementation phase.

Release checklist

- Full deterministic suite passes with no unexpected skip.
- Standing security guarantees pass.
- New features have behavioral positive and negative controls.
- No credentials, private keys, private paths, client data, or private infrastructure details appear in public artifacts.
- Version, changelog, README, and operational documentation agree.
- Release manifest binds source, artifacts, contracts, and deployment order.
- Staging acceptance and rollback evidence are current where deployment is in scope.
- Production approval is explicit, attributable, scoped, unexpired, and single-use.

27. Published Baseline and Current Evidence

The public project is an open-source, self-hostable Python broker under Apache-2.0. GitHub main a9f45d6e46307843b37f93f71947410935cf5f11 includes the backend integration and public documentation

publication through merged pull request #1. The publication commit is 0e323919179942c03d866f5ccc024c0010f99ca6. The public README and documentation index are the entry points for current setup, limitations and operational guides.

Verified public-source evidence: the clean Python 3.12 snapshot passed 257 tests with one expected skip because the private master dossier is deliberately absent from public checkouts. The PR and post-merge GitHub checks passed on Python 3.11, 3.12 and 3.13; the Gitleaks, Bandit and Semgrep jobs also passed. Post-merge evidence is available in the test workflow and security workflow. These checks use test data and mocked providers; no million-user load test, live uptime measurement or production deployment was performed.

The publication checks found no secrets in the scanned public snapshot or reachable history; Bandit at the configured medium/high threshold and the repository's Semgrep configuration returned no findings. Scanners establish only their tested scope. Runtime initialization, test registration, inspection, dossier approval, audit verification, backup and restore were exercised. A source distribution and wheel built with setuptools 84.0.0, and the installed CLI passed a smoke test.

Open limitations: the declared setuptools>=68 build requirement is too permissive for the SPDX license metadata; compatible build tooling was used for verification, but tightening the minimum remains an outstanding packaging fix. No standalone type-check or general-lint result is claimed. The publication received an author documentation review, not a new independent implementation review. Phase 1 closure, authenticated authority suitable for untrusted users, real-client operation and production acceptance remain separate gates.

Registration examples in section 6 assume an initialized, private runtime and an installed hub command. RUNTIME and PROJECT_ROOT are operator-supplied paths, and OWNER is an example actor label rather than authenticated identity. Use a test repository, inspect the draft dossier, and follow the backend setup guide for complete commands. Configuration currently uses CLI arguments, registry state and configuration files; a unified environment/settings audit remains planned.

28. Planned Frontend and API

No browser frontend or general web API is implemented in the published baseline. The current user surfaces are the CLI and project-local agent/editor integrations. The following is a proposed extension, with its own implementation, security, usability and release gates; publishing its design does not make the service available.

28.1 User journeys and authority

A future browser interface should cover authenticated onboarding, exact repository registration, discovery review, task submission, bounded progress/evidence views, approval decisions, cancellation/resume, model selection with explicit cost information, and operator recovery. Every request must reach the broker's policy and state machine. Browser state, URL parameters, client-side roles and model text must never become authority.

A versioned API should define request/response schemas, tenant-scoped identifiers, idempotency keys for mutations, revision or ETag conflict checks, pagination, bounded uploads, stable error categories and request correlation. Authentication, membership revocation and authorization must be checked on every object and action. Approval should bind the actor to the exact operation, scope, revision and expiry, with protection against replay.

28.2 Browser safety, accessibility and delivery

Use least-privilege sessions, secure cookie settings where applicable, CSRF protection for cookie-authenticated writes, a restrictive content security policy, safe rendering of untrusted source/model output and bounded streaming. Never send secrets, unrestricted transcripts or private source to analytics. Enforce tenant separation in APIs, queues, caches, artifacts and event subscriptions. Apply upload validation, request quotas, rate limits and backpressure at server boundaries.

Target WCAG 2.2 AA through keyboard navigation, visible focus, clear validation, screen-reader status announcements and accessible approvals. This is an accessibility goal, not a certification. Before release require browser tests, permission-denial tests, session/tenant isolation tests, reconnect/cancellation tests and an independent security review. The frontend/API design describes the proposed contract and rollout boundary.

29. Future Scale: One Million Concurrent Users

One million simultaneously connected users is a future capacity objective, not a measured capability. Connections, active viewers, API requests and running model jobs are different workloads. The current local broker and SQLite runtime must not be presented as a million-user service or placed behind a public load balancer as if that alone solved scale.

29.1 Example workload model

For planning only, assume 1,000,000 connected sessions, 10% polling status once every 10 seconds, and 1% submitting a job every 10 minutes. This implies approximately 10,000 status requests/second and 16.7 new jobs/second. At an average 120-second job duration, Little's Law implies about 2,000 active jobs. These assumptions must be replaced with measured traffic distributions, model latencies and tenant behavior.

A 1 KiB event at 10,000 events/second is about 9.77 MiB/second before protocol overhead. At an assumed 32 KiB of connection memory, one million connections consume about 30.5 GiB before application, TLS, operating-system and redundancy overhead. This arithmetic is not a machine-sizing result or a hosting quote. Model throughput, inference memory and provider quotas can dominate capacity and cost.

29.2 Proposed distributed architecture

Separate stateless API gateways, connection/event gateways, a durable job queue and isolated worker pools. Preserve the deterministic broker as the authorization boundary. Introduce a transactional shared datastore appropriate to measured write patterns, tenant-aware partitions, explicit consistency rules, bounded caches and separately controlled artifact storage. Keep one writer per protected repository and use durable leases with fencing to prevent stale workers from committing.

Expect queue redelivery: make accepted operations idempotent and reconcile external side effects rather than promising exactly-once execution. Enforce per-tenant quotas, fair scheduling, bounded retries, admission control and circuit breakers. Scale from queue age and completion capacity as well as CPU; shed excess work explicitly. Regional placement must respect approved data locations and providers. An outage must never authorize a different model, region or broader data transmission.

29.3 Capacity acceptance gates and economics

Advance through reproducible load tests at 100, 1,000, 10,000, 100,000 and 1,000,000 connections. For each stage publish source revision, environment, traffic mix, latency percentiles, accepted/rejected throughput, queue age, resource use and cost assumptions. Proposed final gates include a 24-hour soak, a 2× burst, worker/database/zone failure drills, tenant isolation and recovery verification. Use results to revise targets before claiming capacity.

Track infrastructure cost per connected session and completed job, storage/egress, retries, provider usage and reserve capacity. Configure budgets and admission limits explicitly; open-source licensing does not make model inference or hosting free. No paid resource or provider is provisioned by this roadmap. See the scalability plan for workload equations and evidence requirements.

30. Proposed Uptime, Recovery and Operations

The repository does not operate a hosted service or offer a hosted SLA. Each self-hosting operator owns availability, staffing and its agreements. Proposed service objectives are 99% for a controlled pilot, 99.9% for a later shared service and 99.95% only after regional resilience and operational evidence justify it. These are future goals; no achieved uptime percentage is claimed.

30.1 Measurement and error budgets

In a 30-day time-based window, 99% permits 7 hours 12 minutes of unavailability, 99.9% permits 43 minutes 12 seconds, and 99.95% permits 21 minutes 36 seconds. A request-based SLI instead measures successful eligible requests divided by all eligible requests; do not convert its budget directly into downtime. Define valid requests, measurement points and windows before rollout. Capacity-related rejections and timeouts count against availability; exclude genuinely invalid input consistently and transparently.

Proposed metadata latency goals are p95 below 500 ms and p99 below 2 seconds, with 99% of status events visible within 5 seconds. Model completion time needs separate workload-specific indicators. Measure API success, queue wait, completion rate, event delay, audit persistence and restore success. Alert on rapid and sustained error-budget burn. When the budget is exhausted, prioritize reliability fixes and pause risky releases under an operator-approved policy.

30.2 Recovery targets and drills

Proposed recovery objectives: no loss of acknowledged transactional state for a tolerated node failure, with recovery within 15 minutes; for an approved regional disaster scenario, at most 15 minutes of data loss and recovery within 60 minutes; for reproducible artifacts, at most 24 hours of loss and recovery within 4 hours. State the failure scope and prerequisites for each target. A regional recovery can exceed a monthly availability budget, so recovery targets and uptime promises must be reconciled.

Encrypt and access-control backups, keep approved independent copies, test key recovery and restore identity, policy, audit anchors and configuration as well as database files. The current built-in backups are integrity-checked but not encrypted. Exercise restore at least quarterly and before major storage changes; validate restored invariants,

prevent duplicate side effects and retain evidence. Proposed disaster objectives remain unproven until measured drills pass.

Prepare monitored canary releases, reversible migrations, rollback criteria, incident ownership, customer-impact communication and post-incident actions. Continuous response requires actual staffed coverage. Test unavailable providers, stale leases, lost workers, queue redelivery, audit failures and revoked access. The reliability plan and operations runbook define the distinction between current local tools and future service responsibilities.

31. Open-Source, Legal and Privacy Responsibilities

Apache-2.0 permits use and redistribution subject to its terms, including applicable license/notice retention and modification notices. It includes a conditional patent grant and does not grant general trademark rights. Preserve third-party licenses and review rights to contributed code, model outputs, model weights and supplied repositories separately. Open source is not permission to publish client materials or use a provider account outside its terms. Refer to the authoritative Apache license.

31.1 Accountability and applicability

Maintainers are responsible for accurate source, license and vulnerability information. Operators are responsible for lawful processing, deployment controls, user notices, agreements and incident procedures. Customers must have authority over the data and repositories supplied. Installing this project does not establish a DPA, BAA, SLA or support contract with its maintainers. A hosted offering needs its own legal identity, privacy notice, terms, contacts and subprocessor disclosures before collecting real user data.

Assess applicable law by actual markets, organizational roles, people served, data and use. For GDPR, establish scope, controller/processor responsibilities, legal basis, rights handling and any transfer mechanism; the European Commission's applicability guidance is a starting point. HIPAA applies to defined covered entities and business associates, not every health-related application; use the HHS applicability guidance and determine required agreements and safeguards before PHI use.

Also evaluate relevant national/state privacy and breach rules, professional confidentiality and privilege, sector outsourcing requirements, employment/consumer obligations and applicable AI regulation. Children, biometric data and payments require distinct review before admission. Profiles named healthcare, legal or gdpr are policy choices, not legal certification. SOC 2 and ISO 27001 assurance are separate from compliance with applicable law; no certification is claimed.

31.2 Data lifecycle and launch evidence

Inventory source, task instructions, identity, credentials, prompts/results, diagnostic evidence, audit records, backups and optional telemetry. For each, record the purpose, authority or legal basis where required, recipients, storage locations, access, retention, deletion and restore behavior. Minimize collection and approved provider context. Never put secret values in model prompts, documentation or analytics.

Retention must account for actual contracts, legal holds and statutory requirements rather than a universal duration. Test deletion across active storage, artifacts, provider retention and backups, while explaining necessary preservation. The current retention command preserves audit/dossier history and referenced evidence; it is not a complete personal-data erasure mechanism. Resolve that gap before accepting real personal data.

Review the exact provider agreement and account settings for training use, retention, subprocessors, support access, location and incident notification. Before real-client launch require a data-flow map, threat review, approved vendors, applicable contracts/notices, access reviews, deletion/restore evidence, incident contacts and a controlled pilot. Legal assessment belongs to the accountable operator with qualified advice for its jurisdiction. The legal/privacy guide records the public readiness checklist.

32. Delivery Roadmap and Exit Criteria

The roadmap is gated by evidence and authorization rather than promised calendar dates. Keep the existing local workflow usable while developing optional shared-service capabilities. Each phase needs reproducible acceptance results, clear ownership, an approved checkpoint and a documented recovery path.

32.1 Foundation and developer experience

Stage A — Close the current foundation: resolve authority and recovery gaps, finish the full test-data end-to-end path, preserve negative security tests and obtain independent implementation review. A passing publication CI run does not close this stage.

Stage B — Reproducible local adoption: tighten packaging requirements, verify install/uninstall and upgrades, document supported operating systems, consolidate configuration ownership and exercise backup/restore. Publish reproducible examples and compatible release evidence.

Stage C — Optional web/API: implement the bounded journeys in section 28, authenticated approval, tenant/object authorization, accessible UI and browser/API security tests. Release only after real-flow and independent review gates pass.

32.2 Real use and shared capacity

Stage D — Authorized real-client pilot: establish exact scope, approved data flows, agreements, provider terms, access control and incident/retention ownership before admitting client material. Require a separate, explicitly authorized production decision.

Stage E — Team/shared service: introduce measured durable scheduling, fair quotas, tenant isolation, shared-state migrations and operator dashboards. Show contention, failure and recovery behavior before expanding access.

Stage F — Regional resilience: validate redundancy, data residency, identity/key recovery, bounded failover and the operator's SLOs with drills and observed service metrics.

Stage G — Million-connection validation: progress through the capacity stages in section 29, publish reproducible load and cost evidence, and expand only when bottlenecks and failure modes are resolved.

Stage H — Mature operations: prove sustained objectives, staffed incident coverage, vulnerability response, upgrade compatibility, supply-chain provenance and periodic access/restore reviews. The public roadmap records remaining work without claiming these stages are implemented.

33. Security Reporting and Public Documentation Maintenance

Use the repository's SECURITY policy for vulnerability reporting; do not post exploit details, credentials or private customer material in public issues. The contribution guide covers scoped changes and validation. Review dependency provenance, pin release inputs, retain required notices and generate/review an SBOM where the release process requires it. Keep a published response commitment within actual maintainer capacity; no guaranteed response time is invented here.

Public documentation should identify the exact source revision, verified behavior, limitations and planned work. Keep the root/backend READMEs, changelog, security policy, operational guides and this overview aligned. The technical source of record is GitHub; this document is a public explanation of that evidence. Retain useful history with clear labels and update this existing document in place.

Before each public release, check links, examples, package/license metadata, secrets and private-path exclusion, dependency/security results, installation and real CLI behavior. Bind claims to the release commit and publish known gaps. A clean scan or successful build does not by itself establish security, legal compliance, million-user capacity or production readiness. The release process and verification record provide the current evidence and reproducibility details.

Appendix A — Status Reference

Status	Meaning	Typical next action
VERIFIED	Required evidence passed	Release or close
VERIFIED_STAGING	Verified and staging passed	Request production approval or close
PAUSED_INPUT	Information missing	Supply bounded input and resume
PAUSED_AUTH	Login or credential action needed	Complete approved action and resume
PAUSED_APPROVAL	High-impact approval needed	Approve or reject explicitly
BLOCKED_QUALITY	Required evidence failed	Review diagnostics or change plan
BLOCKED_POLICY	Policy refused the action	Change scope/policy through authorized process
FAILED_INFRA	Tooling or infrastructure failed	Repair infrastructure and resume
CANCELLED	User stopped work	Inspect evidence, retain or clean workspace

Appendix B — Key Terms

- **Broker** — deterministic authority that validates and coordinates every workflow transition.
- **Capability** — approved named operation that may use a protected resource without exposing it to a model.
- **Checkpoint** — validated immutable record linking task progress to evidence and dossier state.
- **Classification profile** — composable policy describing data and operational restrictions.
- **Egress bundle** — explicit minimized artifact set prepared for an approved external route.
- **Lease** — time-bounded ownership record preventing conflicting writers or resource users.
- **Safe projection** — dossier or source representation that excludes restricted content.
- **Standing guarantee** — executable assertion answering a persistent security question.
- **Strictest-wins composition** — policy rule in which restrictions accumulate and lower layers cannot weaken higher layers.
- **Typed artifact** — schema-validated handoff used instead of unstructured model-to-model trust.

Appendix C — Client and Developer Review Checklist

Client review

- Confirm the registered system boundary contains only authorized repositories and environments.
- Confirm the selected profiles match actual data, jurisdictions, contracts, and operational risk.
- Confirm provider and research routes are acceptable for the intended content.
- Confirm retention, deletion, incident, and approval responsibilities have named owners.
- Confirm the public description does not imply automatic legal certification.

Developer review

- Trace a task from submission through plan, workspace, worker, apply, quality, repair, and final report.
- Verify denied egress before process start, key access, or spend.
- Verify a resumed task re-acquires its repository lease.
- Verify generated code with credential-like literals is rejected.
- Verify missing sandbox isolation cannot appear as passed evidence.
- Verify cross-repository changes produce a consistent release manifest.

Operator review

- Run full and standing-guarantee suites in the supported environment.
- Verify audit chain and external anchor.
- Exercise backup verification and restore.
- Exercise cancellation, emergency stop, resumable infrastructure failure, staging rejection, and rollback.
- Inspect expiring exceptions, access review, SBOM, retention, and provider approvals.

Developed by Zahidul Islam

Secure hybrid AI engineering — policy-governed orchestration, model isolation, evidence-driven verification, controlled egress, operational recovery, and client-safe delivery.