

Product and ownership

The release combines a static Next.js frontend with a same-origin FastAPI control plane, PostgreSQL ledger, separate durable worker, local Ollama inference and rootless gVisor execution. The web process never executes submitted code. `/control/` uses authenticated persisted records. `/workspace/` maintains separate browser-tab-local interaction state, cleared on reload.

PostgreSQL owns run claims, leases, attempts, artifacts, approvals, audit events, delivery intents and receipts. Model calls occur outside database transactions. Fencing and revision checks prevent stale workers from publishing. Workspace stop, source/member revocation and repository grant changes serialize with publication. An acknowledged change prevents subsequent publication or waits for the bounded in-flight operation.

Git preparation occurs in an isolated clone with installed hooks. Publication compares the expected reference atomically and uses the native index lock. An uncertain post-publication result stays reserved for read-only reconciliation. Dedicated operator-owned repositories must not be edited manually during installation.

Project-specific services, users, storage, private listeners, resource limits and versioned releases isolate the application on shared infrastructure. General shell execution, arbitrary languages, automatic deployment to customer infrastructure, remote providers and external publishing connectors require additional engineering and authorization. Tenant RLS, administrator MFA and multi-region failover remain roadmap gates; application authorization and restricted database roles are the present boundary.

The current release implements the frontend and bounded authenticated drafting/review and Python coding/test/approved-Git workflows. Production evidence is recorded per release. General connectors, autonomous scheduling, administrator MFA, database RLS, customer-data legal readiness and large-scale capacity remain explicit gates. See core workflows and release acceptance.

The authoritative detailed roadmap is BUILD-PLAN.md, which contains ten phases, five parallel workstreams per phase, deploy/rollback gates and 57 specified acceptance scenarios. A specified scenario is not a passing test.

Every phase coordinates frontend, backend, agent/integration, infrastructure and quality work. Every phase ships a scoped increment with rollback; exit criteria are verified in the intended environment before that phase is marked complete.

Frontend and authenticated workspace

The authenticated control plane uses `/api/v1`. Its exact machine-readable contract is `GET /api/v1/openapi.json`. `/api/v1/health` checks database connectivity and migration head; inference and sandbox qualification are separate readiness checks.

Never store the session cookie in JavaScript, URLs or `localStorage`. Cross-origin writes and missing/stale CSRF tokens are refused. Passwords are Argon2 hashed. Email ownership verification, self-service password recovery and administrator MFA are future work; operators create/recover accounts through a private verified process.

Inspect OpenAPI for exact methods, schemas and paths. All writes require session/CSRF validation; object access additionally requires workspace membership and the appropriate role. Repository grants have no self-authorizing HTTP route.

The v1 wire contract limits names to 80 characters, titles to 100, ordinary text to 4,000, artifacts to 20,000 and criteria to 20. Passwords require 12-128 characters. Settings own body, list, task/source/run quotas and request budgets. Responses use stable error codes, including 401 authentication, 403 permission/CSRF, 404 inaccessible objects, 409 stale/conflicting work, 422 invalid input, 429 exhausted budget and 503 unavailable readiness. Lists are bounded, not an unbounded historical export API.

Mutations use task revisions and action request keys where supported. Reusing keys cannot create duplicate consequential actions. Following a timeout, inspect the recorded run/delivery before retrying. Breaking wire-contract changes require a new API version and migration notes.

The authenticated `/control/` workspace uses an essential opaque `HttpOnly` session cookie. Production enables `Secure`, `SameSite=Lax` and a host-prefixed cookie scoped to `/`. The default session duration is 24 hours; login rotates the session and logout revokes it. An anonymous session is used to establish CSRF protection before login. Cookie values are not readable by JavaScript.

The browser receives a CSRF token to authorize same-origin mutations. Account changes clear private UI state; delayed responses cannot install another account's workspace state. Private responses use `no-store`. `/workspace/` interaction state lives only in the current tab and resets on reload.

System architecture and boundaries

The release combines a static Next.js frontend with a same-origin FastAPI control plane, PostgreSQL ledger, separate durable worker, local Ollama inference and rootless gVisor execution. The web process never executes submitted code. `/control/` uses authenticated persisted records. `/workspace/` maintains separate browser-tab-local interaction state, cleared on reload.

PostgreSQL owns run claims, leases, attempts, artifacts, approvals, audit events, delivery intents and receipts. Model calls occur outside database transactions. Fencing and revision checks prevent stale workers from publishing. Workspace stop, source/member revocation and repository grant changes serialize with publication. An acknowledged change prevents subsequent publication or waits for the bounded in-flight operation.

Git preparation occurs in an isolated clone with installed hooks. Publication compares the expected reference atomically and uses the native index lock. An uncertain post-publication result stays reserved for read-only reconciliation. Dedicated operator-owned repositories must not be edited manually during installation.

Project-specific services, users, storage, private listeners, resource limits and versioned releases isolate the application on shared infrastructure. General shell execution, arbitrary languages, automatic deployment to customer infrastructure, remote providers and external publishing connectors require additional engineering and authorization. Tenant RLS, administrator MFA and multi-region failover remain roadmap gates; application authorization and restricted database roles are the present boundary.

Identity, permissions and data

Opaque server-side sessions, Argon2 password hashing, secure production cookies, session rotation/revocation, exact Origin and CSRF checks protect API mutations. Workspace membership and role authorization precede protected object access. Private API responses use Cache-Control: no-store; public caches must never store account or workspace responses.

Task revisions and artifact hashes bind approvals to exact content. Two distinct authorized accounts approve client and developer decisions. Repository execution requires an operator-created workspace/target/profile grant. Revoked members, sources, grants and stopped workspaces cannot authorize later publication. Agent instructions and retrieved documents never grant permissions.

The web service has no code execution route. Workers use pinned local model digests, structured responses, bounded correction rounds and separate review contexts. Protected test answers remain outside the model and guest. Rootless gVisor uses a read-only minimal rootfs, no guest network, no host credentials, fresh case processes and CPU/memory/process/output/time limits. Candidate output is untrusted data.

Use least-privilege service accounts and database roles, pinned SSH host keys, private database/inference listeners, maintained TLS, bounded logs, verified encrypted backups and tested restoration. Do not expose Ollama, PostgreSQL, Docker sockets or runner state to the Internet. Keep runtime secrets outside source, browser bundles, logs and command arguments. Restrict operator access using MFA at the access provider where available; application administrator MFA remains a roadmap item.

Application-level authorization is implemented; PostgreSQL tenant RLS is a planned additional defense. Disk/volume encryption, independent penetration testing, supply-chain attestation, automated privacy erasure and continuous capacity testing require deployment-specific qualification. No scanner result establishes blanket security certification.

Use the private reporting route in SECURITY.md. Do not post credentials, personal information or exploit details in public issues. Assess impact, contain/revoke affected access, preserve redacted evidence, patch, verify and publish an appropriate advisory. Keep incident chronology and corrective actions.

References: OWASP session management, multi-tenant security, gVisor production guide.

Agents, review and human authority

The frontend and authenticated control plane implement two bounded core workflows:

These workflows do not establish completion of the ten-phase production roadmap. Unrestricted language/build environments, remote deployment connectors, production tenancy qualification, cloud surfaces and capacity/recovery targets remain separate gates.

Install the pinned backend requirements into an isolated Python 3.12 environment. Copy the backend settings from `.env.example` into `backend/.env`, supply a fresh session secret and choose a database URL. Keep the real values in the ignored recovery file. SQLite remains supported for development. PostgreSQL is exercised by the full backend suite and the actual local qualification. Use separate migration and runtime roles: runtime connections reject superuser, `BYPASSRLS`, schema-create and table-owning privileges. The migration URL is used only by the explicit migration command. PostgreSQL RLS policies and production operational qualification remain separate roadmap work; restricted roles do not by themselves provide tenant RLS.

Build the frontend with the configured public origin and set `SH_FRONTEND_DIRECTORY` to the resulting static export. From `backend/`:

Open `/control/` on `SH_PUBLIC_ORIGIN`. Register the developer account, create a workspace and invite a separate client account. Add explicit criteria before starting a task. Read the exact artifact before approving. `/workspace/` remains the browser-local workspace interface.

The control UI checks session identity around private refresh batches and hides private state before rechecking a newly activated tab. This handles a shared browser cookie changing in another tab. Keep the configured session request budget compatible with the configured polling frequency; a regression checks the defaults. Identity changes invalidate delayed responses and clear invitation, artifact, CSRF and workspace state.

Release authorization and acknowledged workspace stops, member/source revocations and host repository-grant changes share a database transaction lock. A control change either completes before a new Git write and blocks it, or acknowledges after the bounded in-flight Git operation ends. External inference still runs outside transactions. Verification errors after an accepted Git commit retain an uncertain receipt and the task reservation until read-only reconciliation proves the outcome.

Fixture practice runs remain available and deliver an internal artifact. They no longer implicitly receive a globally configured repository release target.

Protected code execution and Git delivery

The frontend and authenticated control plane implement two bounded core workflows:

These workflows do not establish completion of the ten-phase production roadmap. Unrestricted language/build environments, remote deployment connectors, production tenancy qualification, cloud surfaces and capacity/recovery targets remain separate gates.

Install the pinned backend requirements into an isolated Python 3.12 environment. Copy the backend settings from `.env.example` into `backend/.env`, supply a fresh session secret and choose a database URL. Keep the real values in the ignored recovery file. SQLite remains supported for development. PostgreSQL is exercised by the full backend suite and the actual local qualification. Use separate migration and runtime roles: runtime connections reject superuser, `BYPASSRLS`, schema-create and table-owning privileges. The migration URL is used only by the explicit migration command. PostgreSQL RLS policies and production operational qualification remain separate roadmap work; restricted roles do not by themselves provide tenant RLS.

Build the frontend with the configured public origin and set `SH_FRONTEND_DIRECTORY` to the resulting static export. From `backend/`:

Open `/control/` on `SH_PUBLIC_ORIGIN`. Register the developer account, create a workspace and invite a separate client account. Add explicit criteria before starting a task. Read the exact artifact before approving. `/workspace/` remains the browser-local workspace interface.

The control UI checks session identity around private refresh batches and hides private state before rechecking a newly activated tab. This handles a shared browser cookie changing in another tab. Keep the configured session request budget compatible with the configured polling frequency; a regression checks the defaults. Identity changes invalidate delayed responses and clear invitation, artifact, CSRF and workspace state.

Release authorization and acknowledged workspace stops, member/source revocations and host repository-grant changes share a database transaction lock. A control change either completes before a new Git write and blocks it, or acknowledges after the bounded in-flight Git operation ends. External inference still runs outside transactions. Verification errors after an accepted Git commit retain an uncertain receipt and the task reservation until read-only reconciliation proves the outcome.

Fixture practice runs remain available and deliver an internal artifact. They no longer implicitly receive a globally configured repository release target.

Security and vulnerability response

Opaque server-side sessions, Argon2 password hashing, secure production cookies, session rotation/revocation, exact Origin and CSRF checks protect API mutations. Workspace membership and role authorization precede protected object access. Private API responses use Cache-Control: no-store; public caches must never store account or workspace responses.

Task revisions and artifact hashes bind approvals to exact content. Two distinct authorized accounts approve client and developer decisions. Repository execution requires an operator-created workspace/target/profile grant. Revoked members, sources, grants and stopped workspaces cannot authorize later publication. Agent instructions and retrieved documents never grant permissions.

The web service has no code execution route. Workers use pinned local model digests, structured responses, bounded correction rounds and separate review contexts. Protected test answers remain outside the model and guest. Rootless gVisor uses a read-only minimal rootfs, no guest network, no host credentials, fresh case processes and CPU/memory/process/output/time limits. Candidate output is untrusted data.

Use least-privilege service accounts and database roles, pinned SSH host keys, private database/inference listeners, maintained TLS, bounded logs, verified encrypted backups and tested restoration. Do not expose Ollama, PostgreSQL, Docker sockets or runner state to the Internet. Keep runtime secrets outside source, browser bundles, logs and command arguments. Restrict operator access using MFA at the access provider where available; application administrator MFA remains a roadmap item.

Application-level authorization is implemented; PostgreSQL tenant RLS is a planned additional defense. Disk/volume encryption, independent penetration testing, supply-chain attestation, automated privacy erasure and continuous capacity testing require deployment-specific qualification. No scanner result establishes blanket security certification.

Use the private reporting route in SECURITY.md. Do not post credentials, personal information or exploit details in public issues. Assess impact, contain/revoke affected access, preserve redacted evidence, patch, verify and publish an appropriate advisory. Keep incident chronology and corrective actions.

References: OWASP session management, multi-tenant security, gVisor production guide.

Privacy and legal readiness

Last reviewed: 2026-10-18.

SynqHarbor AI provides a public website and an operator-managed authenticated workspace. The maintainer is Zahidul Islam. Self-hosting operators determine their own deployment purposes and must publish the applicable operator contact and processing schedule before accepting customer personal data.

Passwords are hashed; plaintext passwords are not stored by the application. Source documents and candidate code can contain personal information, so users must provide only authorized, necessary content. The local model adapter sends no context to a paid/cloud inference endpoint. Model downloads and software updates contact their registries separately from task inference.

The hosted website uses existing Contabo hosting and Cloudflare network delivery/security. GitHub hosts public source and feedback. Following third-party links uses those services under their own policies. Do not submit confidential or personal data in public issues.

Configured sessions/invitations expire after 24h by default. Operators control account/workspace retention, access and backup expiry. `SH_RETENTION_DAYS` expresses policy configuration; automatic deletion of all workspace records is not implemented. Operators must enforce and document deletion/retention operationally, including audit/legal holds and backup expiry. Technical log rotation currently bounds volume; a fixed age-based retention guarantee requires deployment verification.

Arrange a private channel through the maintainer profile for access, correction, export, deletion or privacy concerns. Verify the requester before disclosing data. Applicable rights, lawful basis, retention periods, processing locations and international transfers depend on the actual operator/customer arrangement. Do not enter regulated or sensitive customer data until that schedule and the required agreements/controls have been qualified.

See data processing, legal readiness and cookies. No advertising or behavioral analytics trackers are installed.

This guide identifies applicability and implementation work for an operator; it does not replace legal advice or certify a deployment. Determine the business/operator role, markets, user ages/locations, data categories, sector, vendors, regions and intended AI use before deciding which obligations apply.

Official sources: EDPB, European Commission AI Act guidance, FTC data protection, HHS cloud guidance.

Reliability and the 99% objective

The initial service objective is 99% availability over a rolling 30-day window. A time-based 99% objective permits 7 hours 12 minutes of unavailable time in 30 days. It is an operating target; a contractual SLA requires an explicit agreement and remedy terms.

Measure the public website, authenticated API and asynchronous work separately. Website probes must check expected content over public HTTPS. API probes check readiness and a controlled authenticated read. Track valid-request success, p50/p95/p99 latency, database pool saturation, queue depth/oldest age, worker heartbeat, lease recovery and delivery reconciliation.

Define availability as good eligible events / all eligible events for request-based measurement, or good probe intervals / all scheduled intervals for time-based measurement. Do not interchange these denominators. Include eligible timeouts, 5xx and legitimate in-envelope 429s as failures. Exclude intentionally invalid/unauthorized requests consistently. Count planned maintenance against the initial objective; document any later contract-specific treatment. Report monitoring gaps as unknown coverage, never assumed uptime.

Review the rolling window at least weekly. Alert on sustained outage and fast/slow error-budget burn using independent probes where available. When the budget is exhausted, prioritize reliability fixes, reduce risky releases and require recovery evidence before resuming changes. Record detection, impact, mitigation, restoration and corrective actions for incidents. Publish customer-relevant status without sensitive infrastructure details.

The initial shared-host architecture has single-host/database dependency risk. Process restart and backups reduce recovery time but do not provide multi-zone failover. Local AI job latency has a separate queue/completion objective and must not make an otherwise healthy API appear ready for unlimited submissions.

Initial proposed recovery objectives are RPO $\leq 24h$ and RTO $\leq 8h$. Production timings and actual retained backup intervals must be recorded before claiming achieved recovery. A future multi-zone 99.9% objective allows 43m12s in 30 days and requires tested failover, dependency redundancy and funded operations.

Reference: Google SRE: implementing SLOs, error-budget policy.

Scaling toward 1M+ simultaneous users

One million simultaneous users is the future design envelope. Connected users, actively interacting users, API requests and running AI jobs consume different resources. The present single-host release has no measured million-user capacity.

These are planning assumptions. Validate read/write mix, tenant skew, payload distribution, data size, cache hit ratio, event fan-out, geographic latency and burst duration before sizing. The current control UI polls every three seconds and rechecks identity around private batches; at one million tabs this is unsuitable. Replace polling with authorized push/invalidation, cursor-based deltas, jitter and backoff before that stage. Preserve account-switch privacy guarantees.

A stage is entered only after its previous workload envelope meets latency, error, correctness and recovery criteria. Add partitioning or read replicas only when measured query and write patterns justify them. Avoid giving every browser a database connection. PostgreSQL pools need a host-wide connection budget across all replicas and workers.

Use durable admission queues, per-tenant concurrency/token budgets, queue deadlines, cancellation and backpressure. Size local inference from measured tokens/second, prompt/context length, output length, model memory and review multiplier. Size sandbox workers from measured case runtime and resource envelopes. Approximate required busy inference slots as admitted jobs/second measured service seconds/job, then add bounded utilization and redundancy headroom. No useful GPU count or monthly price can be asserted without that workload and hardware benchmark.

The current local-only model adapter does not become a remote GPU fleet by configuration. Secure distributed inference, workload identity, provider qualification and data-processing agreements need a separately reviewed adapter. Preserve model-digest checks and separate review/approval boundaries while scaling.

Serve immutable public assets through a CDN; reserve private API paths from public caching. Apply quotas, request bounds, tenant fairness and queue admission before expensive work. Autoscaling must have an approved maximum, dependency limits and rollback/backpressure behavior. Database saturation, GPU queues, logging/egress costs and third-party quotas are practical blockers. Budget alerts are not spending caps.

References: PostgreSQL connection resources, FastAPI deployment concepts.

Operations and recovery

Confirm the target host and available capacity using native inventories. Capture the current live files/configuration, compare with the prior local manifest, and reconcile drift before editing. Build from pinned dependencies with the intended public origin. Run tests, types, lint, dependency/security scans and a fresh-context source review. Render deployment settings centrally; secrets remain in private environment files with restricted permissions.

Create a new versioned release and manifest. Provision only the project-specific service account, database identities/storage, local inference and protected runner within approved existing capacity. Migrate using the dedicated migration role; web/worker environments receive only runtime credentials. Validate reverse-proxy/systemd/container settings before activation. Keep database/inference listeners private and public routes on the single HTTPS origin.

Start database/inference, apply migrations, then start the API and worker. Check API schema readiness and actual model/sandbox qualification separately. Validate secure cookies, CSRF/Origin denial, outsider access denial, job completion, protected tests, distinct approvals and Git receipts through HTTPS. Switch the static release and proxy configuration only after preparation succeeds. Compare every deployed source/static/config byte with its local manifest. Preserve unrelated host services.

Monitor HTTPS, eligible errors/latency, pool saturation, queue age, worker liveness, disk/memory, model latency and lease/reconciliation failures. Limit log size, redact credentials/content and define retention. Inspect error-budget burn before risky changes. Record incidents and remediation; public status must not disclose private topology or credentials.

Keep the previous release, proxy/service configuration and encrypted database backup. Stop new admissions and let bounded work finish or cancel/reconcile it. Restore the previous frontend/API configuration and restart only project services. Database migrations are additive but schema compatibility must be checked; never blindly downgrade a database with accepted writes. If restoration is required, document lost-write scope against RPO, restore to an isolated database, verify rows/authorization, then reopen. Record actual RTO and final parity.

Roadmap, licensing and qualification

The current release implements the frontend and bounded authenticated drafting/review and Python coding/test/approved-Git workflows. Production evidence is recorded per release. General connectors, autonomous scheduling, administrator MFA, database RLS, customer-data legal readiness and large-scale capacity remain explicit gates. See core workflows and release acceptance.

The authoritative detailed roadmap is BUILD-PLAN.md, which contains ten phases, five parallel workstreams per phase, deploy/rollback gates and 57 specified acceptance scenarios. A specified scenario is not a passing test.

Every phase coordinates frontend, backend, agent/integration, infrastructure and quality work. Every phase ships a scoped increment with rollback; exit criteria are verified in the intended environment before that phase is marked complete.

Read the complete public documentation for exact settings, commands and deployment responsibilities. Acceptance criteria and remaining qualification gates guide each release. Future capacity requires workload assumptions, measured hardware performance, operational controls and approved funding. Source is Apache-2.0; dependencies, model weights and fonts retain their respective licenses. Zahidul Islam maintains this project and its public repository.

Future cloud delivery prioritizes suitable AWS, GCP and Azure components within verified free allowances. The existing authorized host supplies heavy local inference and protected execution. Secure hybrid adapters, account eligibility, dependency costs and separately approved paid growth remain release gates.