FLEETWRIGHT — RELEASE AND OPERATING UPDATE — 15 OCTOBER 2026

Current deployment and public documentation

Frontend/backend release: 20261015-publication-rc1. Immutable runtime source: cd64e6b. Final owner GitHub revision: ae145decde0ec78e58689956dbe39f879ee224e6, authored under Zahidul Islam's established identity. All 80 public Markdown documents, including all 20 READMEs, were reviewed; 450 local links passed. Business runtime is unchanged from the 14 October browser-supervision correction.
Public documentation: https://github.com/Zahidulislam2222/fleetwright/tree/main/docs

Verified behavior and recovery

The complete isolated suite passed 289 tests with six accounted platform/optional skips, zero failures/errors and native exit 0; all 363 runtime/configuration/build-prerequisite hashes were unchanged. Types, lint, builds, security scans and independent reviews passed. Exact native app/web scans each reported zero HIGH/CRITICAL findings; signed image identities, 146 deployed artifact hashes and 224 host source hashes matched.
All original actors drained cleanly with zero active work, backlog and account holds before a validated archive and activation. Actual public owner/TOTP, fresh booking, both approved MCP workflows/receipts/revocation, four-record crawler export, authorization denials, HTTP and all 15 CSP routes passed. A fresh archive restored into two new databases with matching workflow history; scheduler/monitor supervised recovery took 66.74/26.65 seconds. Four fresh monitor probes were healthy, with zero active alerts; an earlier unhealthy metrics sample remains in history.

Security, data, law and operating ownership

The public guides cover typed configuration/private secrets, least privilege, tenant isolation, exact human approval, receipt uncertainty, data retention/deletion/holds, privacy roles and transfers, license/commercial permissions, destination authorization, vulnerability handling, incident ownership, backup and recovery. Customer use requires its jurisdiction, processing and destination assessment; these controls do not establish legal or security certification.
Legal and commercial readiness:
https://github.com/Zahidulislam2222/fleetwright/blob/main/docs/legal/commercial-readiness.md

Future scale and availability roadmap

Future design targets 1M+ simultaneous authorized console sessions, with browser execution budgeted separately. The 99% initial and 99.9% multi-zone objectives require declared SLIs, error budgets and a rolling 30-day observation window. Promotion proceeds through 1k, 10k, 100k and 1M+ sessions with representative hour-long workloads, tenant skew, reconnect/slow-reader/revocation checks and failure/recovery exercises. Multi-zone API/gateway capacity, database failover/read routing, static reserve and isolated cells require measured resource/cost evidence before promotion.
Scale validation: https://github.com/Zahidulislam2222/fleetwright/blob/main/docs/scaling/validation-plan.md

Publication and evidence boundaries

Final revision ae145de passed actual GitHub quality, security and independent public HTTPS checks.
Final checks: quality https://github.com/Zahidulislam2222/fleetwright/actions/runs/38630648071; security https://github.com/Zahidulislam2222/fleetwright/actions/runs/38630648047; independent HTTPS https://github.com/Zahidulislam2222/fleetwright/actions/runs/38630925773. The current deployment remains on one existing server; no new paid provisioning/model verification or off-host export was performed. Installed-model, 50/70-context endurance and independent encrypted restoration retain their 13 October dates. The development dependency advisory, dated passive findings and full-host/key recovery boundary remain disclosed.
Verified release and GitHub run links:
https://github.com/Zahidulislam2222/fleetwright/blob/main/docs/releases/2026-10-15.md

Earlier publication and engineering history — preserved below

All earlier paragraphs, five tables and existing links continue below. Their status and roadmap statements retain their original dates; use this latest update for current release and publication facts.

Document review — 14 October 2026

Reviewed against current repository documentation and recorded release evidence. The latest release record below remains consistent with those sources; its measurements are dated verification, not a new live test. All earlier publications and five tables are retained. Read the newest release first: older roadmap and status statements describe their labelled historical periods.

FLEETWRIGHT — VERIFIED COMPLETION UPDATE — 13 OCTOBER 2026

This current release record comes first. Earlier publication chapters, tables and project history continue after the historical marker at the end of this update. Current measurements, retained limitations and future objectives are distinguished throughout.

Release and acceptance record

Deployed frontend/backend release: 20261013-free-local-rc4.

Immutable runtime source: bee3ebfc13fc9515f94684a4abd02ba237b0811b, authored under Zahidul Islam's established repository identity.

Owner GitHub publication head: d65a62d42002e2c4d51ca82fb1d81e6ebadaee87

Verified GitHub quality/security and independent uptime results: All three workflows succeeded on publication head d65a62d42002e2c4d51ca82fb1d81e6ebadaee87: quality https://github.com/Zahidulislam2222/fleetwright/actions/runs/38439091827; security (Gitleaks, Bandit and Semgrep) https://github.com/Zahidulislam2222/fleetwright/actions/runs/38439091955; independent public HTTPS probes https://github.com/Zahidulislam2222/fleetwright/actions/runs/38439145666. Repository description, homepage and 15 topics were verified through native GitHub readback.

All ten delivery phases are complete for the current bounded feature scope. The 20 engineering acceptance criteria are met, with independent source/artifact review, exact release rehearsal, existing-server deployment and actual public verification. Large-scale commercial capacity, arbitrary third-party destinations, multi-zone operation and measured availability commitments are separate future milestones.

The final full regression collected 280 cases: 274 passed, six expected platform/opt-in skips, zero failures and zero errors in 762.41 seconds. All 435 source/tool hashes remained unchanged during the run. Four POSIX state-privacy controls and one POSIX release-symlink control passed separately; actual installed-model/browser evidence produced three matching receipts. The skips do not substitute for those separate proofs.

Ruff, formatting of 234 files, strict types on 87 source files, nine workspace package builds and repeated affected builds passed. Frontend lint/types, an 18-page static build and contrast checks passed. Mandatory Gitleaks, Bandit and Semgrep hooks passed. The final five-file source scan reported zero findings and zero scanner/parser errors. Fresh-context implementation and artifact reviews approved the reviewed release.

The immutable inventory contains 144 signed primary artifacts, 147 total files and 146 manifest entries. Trusted-key signature verification passed. All 146 deployed artifact hashes, complete manifest bytes and expected file count matched local. All 222 server/host backup-source files matched the committed archive. Eight running application/web service roles matched their respective signed native image identities. Local/server image IDs differ; source equality is not binary equality or reproducible-build proof.

Actual native application/web scans reported zero HIGH/CRITICAL findings at scan time, with identity-bound CycloneDX inventories of 1,094 and 72 components. Actual application-image Chromium isolation/navigation and module imports passed. All 129 served static files matched their manifest. The static runtime had no Node/npm or package tree, and three exact official Alpine security patches were verified.

Platform and authority model

Fleetwright coordinates authorized browser work through a durable control plane. PostgreSQL owns claims, workflow intent, tenant isolation, account pacing, fencing and approvals. Redis transports work and invalidation hints; it does not replace database authority. Playwright workers keep warm sessions within maintained destination contracts. A separate catalogue crawler collects approved fields; a separate local application uses SQLite and an installed model for drafting.

Owners approve the exact tenant, account, inputs and complete versioned workflow contract. MCP observation/execution scopes are separate, short-lived and revocable. Clients cannot grant their own authority or approve actions. Changed, expired, revoked or consumed approvals cannot authorize execution. Lost acknowledgements or process death after action begins leave UNKNOWN until matching destination receipts resolve the outcome. Missing receipts alone are not permission to repeat an action.

Delivered capability areas

Coordinated operations include watcher/actor separation, warm sessions, filters/schedules, account exclusivity, database-clock cooldowns, current fencing and durable raw detections. Tenant handover preserves unread/in-flight detection decisions under current directory authority. Freight and general approved workflows retain separate process/account lifecycles.

Freight booking and maintenance requests use separate payloads and receipt contracts. Six bounded MCP tools describe contracts, submit proposals, list/inspect jobs, cancel safely and export paginated redacted evidence. The console supports owner review, investigation, audit, live updates and evidence download. Owned destinations provide independent outcome records; their verification does not authorize another website.

Catalogue crawling uses maintained same-origin definitions for seeds, JSON/CSS extraction, record identity and field/type/length constraints. A fenced durable frontier preserves committed progress. Requests have pinned address/TLS checks, robots handling, redirect refusal, pacing, finite retries, byte/page budgets and physical process deadlines. Quality guards distinguish complete cycles from partial failures, flag zero-result/drop anomalies and avoid treating unseen partial-cycle records as removals. Authorized operators can download private JSONL; public visitors cannot start/export a crawl. Commercial proxy/challenge services and other delivery formats require separately reviewed integrations/costs.

The standalone agent stores tasks, approval consumption, events and receipts in SQLite. Installed local Ollama produces a typed title/body draft only. Three maintained templates cover maintenance, dispatch handover and inspection records. Templates own destinations, selectors and save/receipt contracts; the model has no approval/action authority. Human edits survive blur/refresh/polling and create a fresh proposal before approval. Interrupted saves reconcile against exact receipts rather than another click.

The agent needs no PostgreSQL, Redis or Docker at runtime. It uses numeric loopback API/model/CDP addresses, session/CSRF safeguards, request/action/time budgets and a dedicated browser profile. Optional CDP isolates its context and preserves owner tabs without inheriting their login. HTTP/WebSocket controls are not an OS sandbox for hostile pages. POSIX state permissions passed separate checks; Windows privacy also depends on directory ACLs.

Monitoring and independent recovery

The deployed collector keeps bounded private SQLite history and a durable local notification sink. Four probes—archive freshness, observation grant, origin health and workflow metrics—were healthy with zero failures in the verification window. Actual collector process death triggered supervised recovery in 28.42 seconds. Observe-only replacement authority and predecessor revocation passed. This measures collector recovery, not whole-service downtime. Notifications remain local JSON/journal output rather than email/SMS paging.

Independent public HTTPS probes run outside the production host. An operator-PC collector previously recorded three successful samples at 60-second intervals. GitHub's public-repository probe is independent of both machines, subject to the actual publication/dispatch result above. Hosted schedules are best effort: delays/dropped runs, default-branch rules and inactivity suspension apply. Runs do not share persistent SQLite history. Brief probe windows do not establish monthly availability or unattended incident response.

The protected backup service is active/enabled on the current release; process directory, unit/environment integrity and all 222 source hashes matched. A fresh post-workflow/monitor archive restored into two new databases; nine jobs, nine approvals, 45 events and six grants matched. One injected backup-process death recovered under supervision in 70.13 seconds and produced a fresh valid archive.

The final post-activity archive was copied independently and protected in a 3,089,454-byte authenticated encrypted bundle. Restoration from that exact ciphertext created two fresh local databases: 24 Fleetwright public tables containing 32,941 rows and eight owned-destination tables containing 83,766 rows. Every restored table matched the source archive's COPY-row count, including nine jobs, nine approvals, 45 events and six grants. No plaintext archive files were written to disk. The independent recovery key was unchanged, prior good encrypted copies were retained and existing databases were not overwritten.

Manifest/filename/size/archive-signature/hash validation rejects invalid archive sets. Wrong keys or altered ciphertext are refused. Preserve application vault/authentication keys separately from recovery keys and archive media. This proves encrypted off-host data restoration, not a second running site, complete host/key/application recovery or measured RPO/RTO.

Configuration, security and release maintenance

Environment-specific settings have one typed boundary with safe environment templates and a generated schema. Maintained product data, model prompts, workflow contracts and patch inputs stay separate from control-flow code. Credentials, private signing keys, browser state and raw recovery artifacts remain private. Obtain public verification keys through an independently trusted channel.

Releases freeze committed source, inventory immutable artifacts, record native application/web identities/SBOMs and sign exact primary files offline. Each role must match its own attested image before startup and afterward. Tags, source hashes and signatures alone do not prove runtime correctness or vulnerability absence. Tested static-only rollback preserves compatible backend/mailbox state; the old backend is not an automatic rollback target after the durable-detection protocol change.

Exact rc4 rehearsal exercised the configured 60-second container grace for all browser/execution workers. Freight drain stopped four slots with zero holds and exit 0/non-OOM in 3.300 seconds, followed by exact-container restoration/four healthy slots. A controlled 12-second workflow turn exited 0 in 12.782 seconds. The crawler exited 0/non-OOM in 0.766 seconds and restarted as the same container. Separate crawler processes respected natural lease expiry, increased fencing and rejected the old fence before completing the four-record export. Static-only rollback preserved a pending mailbox row and backend identities, then restored mounts/control.

The historical upgrade exposed old-release shutdown failures. The old workflow runner lacked a signal handler and was forced after its 60-second grace, exit 137: a failed graceful gate. A separately reviewed idle-only handoff proved seven terminal jobs, seven approvals and 35 receipt/history events unchanged against the pre-stop archive, zero account holds and no legacy backlog before control shutdown. Old freight later passed unchanged strict drain after reviewed recovery; the upstream cause of its earlier disconnected browser driver remains unproven. These incidents are preserved, not relabelled as clean initial stops. New-release gates remain strict.

The rc4 passive baseline completed in 78.450 seconds: zero HIGH, one MEDIUM inline-style CSP warning, one LOW cross-origin isolation warning and six informational alerts; raw exit 2/warnings retained. Production npm and the 52-external-distribution Python audits reported no vulnerabilities. Full npm development audit retains unpatched HIGH braces 3.0.3 / CVE-2026-93687, with no fixed version returned. Static-runtime/package/SBOM checks support a trusted-build exception, not a clean full audit. Scanner results are point-in-time evidence, not universal security, ASVS certification or an independent penetration test.

Build phases and acceptance

1. Foundation completed: repository, interface contracts, design system, typed configuration and security/legal boundaries.
2. Owned destinations completed: authentication, first-booker behavior, maintenance forms and independent receipts.
3. Coordination core completed: tenant transactions, claims, leases, fences and durable message intent.
4. Browser runtime completed: isolated contexts, session vault, account holds and bounded adapters.
5. Delivery pipeline completed: watchers, dispatch, pacing, reconciliation, cell routing and durable handover.
6. Console/observability completed: owner/role workflows, audit, live updates, metrics and verified renewable monitoring.
7. Catalogue crawler completed: durable frontier, network bounds, extraction/quality guards, restart and private export.
8. Local agent completed: installed-model drafting, exact human approval, browser saves and receipt recovery.
9. Hardening/evidence completed: adverse-condition tests, source/image/artifact assurance, fixture endurance, encrypted recovery and offline AWS reference.
10. Release/handoff completed for the bounded scope: exact rehearsal, reviewed compatible replacement, public verification, backup/monitor recovery, public documentation and capability recordings; owner publication evidence is recorded above.

Engineering acceptance: 20/20 criteria met. Public verification covered origin checks/TLS, both approved MCP workflows with receipts/revocation, password/TOTP owner access, four healthy freight slots, crawler completion/private export, public authorization denial, responsive navigation, roles/CSRF/audit/SSE and all 15 hydrated routes over full ten-second observations with zero recorded CSP/page errors. Initial calendar estimates are historical planning records, not new delivery promises. Optional integrations, commercial terms and larger infrastructure are separately scoped.

Scale model and reliability objectives

The long-term target is at least 1,000,000 simultaneous authorized control-plane sessions across tenants, not one million browser actors. Future architecture separates CDN/static delivery, stateless API/authorization, sharded update gateways, pooled authoritative database access, durable fan-out and independently scaled execution cells. It includes tenant fairness, bounded queues/refetch, reconnect control, retention and writer-authority recovery.

Promotion requires representative 1k/10k/100k and then 1M sustained mixed workloads, tenant skew, slow/revoked clients, reconnects and node/zone failures. Browser execution additionally needs authorized account/session/rate/receipt measurements. Separate 50/70-context owned-fixture runs lasted 3,601.986/3,602.632 seconds with zero errors under retained resource limits. These establish fixture endurance, not authenticated freight or 1M-user capacity. API/SSE workloads and three offline AWS provider-substituted plan tests have distinct boundaries; no AWS provisioning/zone-loss exercise is claimed.

The initial operating objective is 99%, with a future multi-zone objective of 99.9%. Over 30 days their time-based error budgets are 7.2 hours and 43.2 minutes; request-based budgets require eligible-event counts. Tests do not establish measured monthly uptime or a commercial SLA. Define SLIs, eligible traffic, missing coverage, windows, burn-rate response, alert ownership and escalation before supporting a commitment. Correctness incidents such as tenant crossover or duplicate action have no acceptable error budget. Timed full-system/key recovery remains necessary before supporting RPO/RTO commitments.

Lawful use, privacy and current legal references

Destination permission, software licensing, account authority and lawful processing are separate requirements. Onboarding records permitted origins/actions/fields, purpose, retention, roles, transfers and incident responsibilities. Do not bypass access controls/challenges/rate limits or create deceptive reviews/unauthorized messages. Customer processing needs an accountable operator/privacy contact, controller/processor assessment, appropriate terms/DPA, subprocessors, hosting/transfer decisions and a tested incident/request process.

The register covers US access/consumer-protection duties, EU GDPR/AI Act/CRA applicability, California privacy rules and Bangladesh's Personal Data Protection Act 2026. Bangladesh Act 63 of 2026 replaced the earlier ordinance; section-specific commencement exceptions require current Gazette checks. EU intended-use/provider/deployer classification and phased AI Act/CRA dates require review for each use. Human approval and public source visibility do not establish legal classification/compliance. These are dated engineering references, not legal advice, universal approval or certification.

Primary references:

Bangladesh official Act 63 of 2026: https://bdlaws.minlaw.gov.bd/act-1692/act-chapter-print-2693.html

EU GDPR: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng

European Commission AI framework: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai

European Commission CRA: https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act

California CPPA: https://cppa.ca.gov/regulations/ccpa_updates.html

Google SRE objectives: https://sre.google/workbook/implementing-slos/

AWS static stability: https://docs.aws.amazon.com/whitepapers/latest/aws-fault-isolation-boundaries/static-stability.html

Cost boundary and next work

Completion used free/local tooling and the existing paid server without adding paid model APIs, monitoring, messaging or cloud infrastructure. Local inference, signing and verification require no new paid service. Public-repository standard CI/probes remain subject to platform terms/quotas/best-effort schedules, not unlimited capacity or availability guarantees. No automatic paid fallback is included.

Next work is periodic operational revalidation, representative authenticated fleet/load evidence, timed full-host/key/application recovery, operational alert delivery, authorized destination contracts, broader assurance and staged multi-zone/control-plane scale. Commercial proxy/challenge providers, new regions, paid integrations and larger hosting require a fresh cost review/explicit owner approval. These future milestones are separate from the delivered bounded feature set.

Public project and owner links

Project: https://fleetwright.zahidul-islam.com

Source: https://github.com/Zahidulislam2222/fleetwright
Documentation: https://github.com/Zahidulislam2222/fleetwright/tree/main/docs
Owner portfolio: https://zahidul-islam.com/
Owner GitHub: https://github.com/Zahidulislam2222

PREVIOUS PUBLICATION AND PROJECT HISTORY — PRESERVED BELOW

The following existing chapters and tables are historical material retained unchanged. Current verified release evidence appears above; older status statements describe their original publication period.

Fleetwright Technical Overview

Reliable Browser Operations — coordination, approval, recovery and evidence

Designed and built by Zahidul Islam

Current release verified 13 October 2026; engineering history retained below

Current release — 13 October 2026

Fleetwright MCP — Reliable Browser Operations is deployed. The public frontend, backend, approved-workflow worker and MCP package are published in the existing GitHub repository under Zahidul Islam's established author identity. Runtime release 20261013-reliable-browser-rc5 uses source revision 446aeb6. This section is the current status; the earlier engineering record follows with its original date and substance retained.

Workflow contracts, authority and recovery

Two owned, versioned workflows are complete: freight booking and maintenance-request creation. Contracts define action identity, account and origin boundaries, validated inputs, preconditions, success receipts and reconciliation. PostgreSQL owns durable jobs, fencing tokens, database-clock lease expiry and account reservations. Stable submission identity rejects changed inputs instead of silently creating another action.

Human owner approval binds the exact tenant, account, inputs and complete contract fingerprint. Changed, expired, revoked or replayed authority is rejected. Six MCP tools support contract discovery, submission, progress inspection, job listing, safe cancellation and evidence export. Observation and execution permissions are separate; every invocation and lookup checks tenant, principal, scope and current revocation. Owner approval remains an authenticated console/API operation.

A worker can die after the destination accepts an action but before confirmation. The durable intent then remains UNKNOWN; reconciliation looks for a matching receipt without another action. Inconclusive receipts retain uncertainty for operator review. Cached destination authentication renews within account ownership and lease bounds. The general worker uses a reserved account pool alongside the existing freight worker.

Verification and deployment evidence

All 14 owner-adjusted implementation criteria are verified. Real Chromium tests cover worker competition, stale fences, interrupted acknowledgement, process death after acceptance, changed approval, tenant denial, revocation, cancellation and bounded evidence. Actual Codex submitted, inspected and exported both workflows using the existing ChatGPT login. Claude configuration is supplied; its actual client verification is deferred while the subscription is inactive.

The final 148-case run recorded 147 passes and one contention failure with TooManyConnectionsError in 461.34 seconds. An overlapping test invocation was observed. The unchanged 200-competitor/10,000-job benchmark then passed separately in 205.19 seconds after the overlap stopped. All final cases have passing evidence across those runs; they did not pass together in one final run. A previous complete 147-case run passed in 546.13 seconds. The overlap's status as the sole cause of the connection failure is unproven.

Lint, formatting, backend types on 66 source files, frontend types/build/contrast, eight Python package builds, configuration regressions, source-security hooks and independent implementation review passed. Release checks found and corrected login readiness, expired-session recovery and shared audit-contract gaps, each with a failing-before/passing-after regression. Clean-checkout CI now generates Next.js route types before standalone TypeScript checks.

The exact artifact passed local MCP, browser and HTTP rehearsal. Public checks then confirmed both approved workflows through real MCP, naturally expired session recovery, metrics authorization and immediate grant revocation. Real browser password/TOTP login, workflow inspection, evidence download, desktop/mobile layout and zero page errors passed. Audit reads after workflow activity, CSRF/roles, live updates and four healthy freight slots passed.

Deployment preserved persistent volumes and rollback releases. The old worker drained four slots with zero held accounts and exit code zero in 3.55 seconds. All 137 deployed artifact hashes, manifest bytes and expected file count matched local; 156 host backup-source files matched the release archive. Twenty-five origin checks, origin TLS and public HTTP passed.

Operations, evidence and recovery

Investigation returns bounded, paginated, redacted state history, fences, safe reason codes and receipt hashes. Export hashes detect content changes; they are not author signatures. Console mutations and audit records commit atomically; external operations record durable intent and outcome. Open update streams recheck revocation within the configured bound, including idle streams.

Authenticated workflow health and Prometheus-compatible metrics expose job states and stalled/unknown alerts. A protected, enabled host service creates verified database archives from the current release. A fresh production archive restored both databases into new targets with matching workflow-job, approval, event and grant counts. After an injected process kill, the backup supervisor recovered in 70.29 seconds and produced a fresh archive. Local archives do not establish off-host disaster recovery; independent encrypted copies and full-host/key restoration remain a required next milestone.

Security, privacy and legal use

Tenant row-level security, separated database authority, encrypted credential/session storage, password/TOTP, CSRF, bounded requests and revocable scoped access form the implemented control boundary. Credentials, private infrastructure details and operational artifacts are excluded from public Git and this overview. Source scans found no real secret in the public source snapshot. Changeable endpoints, operational limits and maintained copy use typed configuration or validated data; wire roles, workflow states and MCP syntax remain fixed protocol definitions.

Python runtime and npm production dependency audits were clean. Full npm audit retains one unpatched development-only braces advisory propagated through five high-severity entries. It is recorded without suppression. The static application still allows inline scripts/styles; stronger CSP, broader dynamic assurance, signed provenance and a release SBOM remain planned work.

Operate only on owned destinations or with documented authorization and permitted accounts. Do not bypass access controls or use the system for unauthorized messaging, fraud or prohibited collection. The public legal register covers platform contracts and computer-access law, privacy, cookies, GDPR/CCPA applicability, controller/processor responsibilities, AI transparency and Cyber Resilience Act applicability. Deployment operators must assess jurisdiction, contracts, notices, rights handling, retention, incident duties, subprocessors and transfers. These controls and documents do not by themselves establish legal compliance or certification.

Future capacity and availability

One million simultaneous users is a future control-plane design target, distinct from one million browser workers. The model assumes one read per session per 30 seconds, roughly 33,333 reads/second before caching; one update per 10 seconds, roughly 100,000 deliveries/second; and a reconnect burst requiring 10,000 handshakes/second. These are disclosed assumptions, not measured capacity. API backpressure, bounded fan-out/refetch, pooled database access, tenant fairness, cell isolation, multi-zone spare capacity and independent browser-density measurements are acceptance requirements.

The validation ladder advances through 1k, 10k, 100k and 1M concurrent sessions with representative reads, writes and updates, slow consumers, reconnect storms, tenant skew and node/zone failure. Browser density is measured separately at 50 and 70 contexts for an hour. Each stage records latency percentiles, errors, resource peaks, recovery behavior and cost before committing capacity.

The minimum future operating objective is 99% availability; the multi-zone target is 99.9%. Over a 30-day month these correspond to about 7.2 hours and 43.2 minutes of error budget. Route-specific SLIs, external probes, rolling measurements, burn-rate alerts and restore drills must validate an objective before it becomes a service commitment. The current single-server deployment has no measured monthly uptime claim.

Roadmap and public documentation

Next: external uptime probes, installed metrics collection and notifications, independent encrypted backups and recovery drills; then additional authorized contracts, repeated client assurance, security assessment, larger browser fleets, multi-zone operation and the staged million-session proof. The separate crawler and standalone local browser-agent interface remain planned. Funding or paid API/infrastructure use requires separate authorization.

Live Fleetwright

Source repository

Release evidence and test boundaries

Public documentation: architecture, API/MCP, operations, security, law/privacy, scaling, SLOs and roadmap

Engineering history — state verified 6 October 2026

The following material preserves the earlier architecture, measurements, decisions and failure-and-fix stories. Status statements describe that date; the current release above supersedes its former gaps in scheduled backups, workflow monitoring, atomic audit, idle-stream revocation and test-database isolation.

Fleetwright runs many logged-in Playwright browser sessions, watches a job feed, and coordinates matching jobs using fenced claims and receipt-based recovery when workers crash, networks drop or two machines race for a job. It is built as one shared core with separate worker roles: watchers that see new jobs, claimers that act on them, and a control process that dispatches, reconciles and measures. A planned crawler and a planned local AI browser agent reuse the same core.

The engine is tested end to end against an owned freight load board built in the same repository. The board has its own login with emailed one-time codes, a captcha, a live feed with ground-truth publish times, first-booker-wins booking and switches that inject failure on purpose. Fleetwright is never pointed at a real third-party site in this project, and this overview explains why that boundary is a legal requirement rather than a convenience.

Since 6 October 2026 the public site runs the complete freight stack. Anyone can open the console and watch the engine claim loads from the owned board in real time; a shared login, shown on the sign-in page, can start a capped run and switch on failure modes, and everything resets automatically.

This overview is written for engineering reviewers, clients and collaborators. It separates what has been built and measured from what is still a design target. Credentials, private infrastructure addresses and personal filesystem paths are deliberately excluded. Source code and public documentation: github.com/Zahidulislam2222/fleetwright. Live site: fleetwright.zahidul-islam.com.

1 The engineering problem

Teams that automate a web workflow with many accounts usually begin with independent scripts. That approach works for one or two browsers and fails at fifty. Two scripts book the same job, or neither does. A crash in the middle of a booking leaves nobody knowing whether it went through. Sessions expire, one-time codes arrive by email, and somebody has to sign in by hand. Nobody can say how quickly the fleet reacts, because nothing is measured.

The hard part is not driving a browser. It is coordination under failure: deciding which worker owns a job, rejecting a worker that woke up late, refusing to retry an action that may already have happened, and showing which jobs completed and which outcomes remain uncertain. Fleetwright moves those decisions out of individual scripts and into one system with a database-enforced state machine, encrypted session storage, a dispatcher with filters and schedules, a reconciler for uncertain outcomes, and per-stage latency measurement.

2 Scope and current status

The project is planned in ten phases, each closed by written acceptance criteria, automated gates, a real end-to-end check and an independent review. Phases one to five are complete: the design system and live site, the owned freight board, the coordination core, the browser runtime and the live engine. Phase six, the live console, is shipped and running on the public site, with one item left: metrics and alerting. The crawler, the local AI agent and the large-scale proof runs are still ahead.

Area	Status
owned freight board with failure switches	Built and tested

Area	Status
Row-level security, claim state machine, fencing tokens, outbox	Built and tested
Worker runtime, session vault, one-time codes, failure evidence	Built and tested
Live engine: watchers, dispatcher, actors, reconciler, cells, latency	Built and tested
Console wired to the live engine	Built, tested and live
Console API, sign-in with password and TOTP, roles, audit log, live updates	Built, tested and live
Public read-only view and capped shared controls with automatic reset	Built, tested and live
Single-server production stack with rehearsed, hash-verified releases	Built and live
Metrics and alerting	Planned (remaining Phase 6 item)
Crawler, local AI agent, scale proof runs, cloud reference	Planned

3 Architecture and responsibilities

Postgres is the source of truth. It holds tenants, target-site accounts, sealed sessions, filters, schedules, jobs, claims with their full history, an outbox of messages to publish and an append-only audit log. Every tenant table is protected by row-level security that is both enabled and forced, so even the table owner sees nothing without a tenant set.

Redis carries fast, short-lived traffic. Each cell has its own Redis with two streams per tenant: detected jobs and claims ready to act on. A core Redis holds control-plane keys such as console sessions. Losing Redis loses speed, never correctness, because every claim is already in Postgres and an outbox relay re-publishes anything left behind.

Worker processes run one Chromium each, with many isolated browser contexts called slots. A slot is either a watcher, which keeps the target's feed page open and listens to the JSON responses it fetches, or a claimer, which keeps a warm, logged-in page ready to book. A control process per cell runs the dispatcher, the outbox relay and the reconciler. Every service is stateless and can be killed at any time.

Component	Responsibility
Watcher slot	Turns each new load in the feed's responses into a detection with a seen time
Dispatcher	Drops fast-path duplicates, applies the tenant's filters and schedule windows, creates the claim and its outbox message in one transaction
Claimer slot	Leases the claim with a fencing token, acts only while the lease is current, records the answer
Reconciler	Applies lease timeouts and resolves uncertain claims by asking the target what the account booked
Outbox relay	Publishes anything the fast path missed, after checking the tenant still belongs to its cell

Component	Responsibility
Console API	Serves the console's read models and owner actions under the caller's role and tenant
Live-update gateway	Holds one server-sent event stream per open console and tells it when something changed
owned freight board	The owned target with a ground-truth log used to audit every test

4 The claim state machine

A claim moves from queued to leased to acting, and then to confirmed, failed or unknown; an unknown claim ends as reconciled. Every transition is a single database statement, and each rule is enforced by the database rather than by worker code.

Exactly one claim exists per job, because a unique constraint on tenant and job key picks one winner however many watchers see the job. Leasing takes a fresh fencing token from a database sequence, and every later step names that token; a worker that paused during garbage collection or a network stall and woke up late is simply rejected. Lease expiry is judged only by the database clock, never by a worker clock.

The most important rule concerns crashes during the action. A claim whose lease expires while it is acting becomes unknown, and an unknown claim is never retried, because the booking may already have happened and a retry could book the job twice. The reconciler opens the account's sealed session, asks the target which loads the account holds, and records the truth. If the original worker later returns with a definite answer and a still-current token, that answer wins. A partial unique index also guarantees that one target account never holds two active leases.

5 Workers, sessions and one-time codes

Browser contexts are much cheaper than separate browsers, which is why each worker packs many of them into one Chromium. Accounts are handed to slots through a holder lease renewed on every heartbeat, so two workers never drive the same account. Sessions are stored as Playwright storage state, sealed per account, and refreshed before they expire. When a target asks for a one-time code, the worker reads it from email through an adapter, or waits for an operator to type it.

Secrets, cookies and one-time codes are never written to logs; a test scans the logs and published events to prove it. When a step fails, the worker records a browser trace and a screenshot with a retention limit, so failures can be investigated without guessing. Every read of a response body is bounded and every browser call has a deadline, because a hostile or broken page must not be able to freeze a slot.

6 Cells and moving tenants

A cell is one Redis plus the workers and control process that serve it. Each tenant belongs to exactly one cell, recorded in the database, so adding capacity means adding a cell and moving tenants, which is a data change rather than a code change. Redis keys carry the tenant as a hash tag, so a cell can later become a Redis Cluster without renaming anything.

Moving a tenant while the fleet is running is tested. The old cell's relay checks ownership inside its own transaction before publishing, the new cell picks up queued claims, and no job is booked twice. Claims already queued are handed over safely; detections in flight at the exact moment of the move may be dropped, which is recorded as a known limitation.

7 The live console and public workspace

The console is a Next.js application exported as static files and served from the same origin as a versioned API, so there is no cross-origin traffic to secure. Every screen reads from the API through one strict contract: the server's response schemas are exported as an OpenAPI document, and a test fails if the server, the contract file or the console's role ranking drift apart.

There are four roles. Owners and operators can create and change filters and type one-time codes for target accounts; viewers can read everything; the shared-access role can only use the shared controls. Schedules are read-only in the console for now. A visitor who is not signed in gets a public, read-only view of the shared workspace tenant with personal data masked, including the email addresses and network addresses in the audit log. Every change made through the console is written to the audit log with the actor, role and address range.

Live updates use server-sent events from a separate gateway. The stream carries only short hints that something changed, never the data itself; the console then refetches through the normal API, so a stream can never show a viewer data that their role should not see. A heartbeat keeps the connection open through proxies, and the console falls back to polling if the stream is unavailable.

The public workspace is designed around abuse rather than politeness. The shared login and its current one-time code are shown on the sign-in page, so many visitors can sign in during the same 30 seconds. Its limits are enforced by the server, not by the screen: one run at a time, claimed atomically, with a maximum length; failure switches capped at safe values; and an automatic return to the calm preset fifteen minutes after the last change. The shared-access role cannot create or change filters or act on target accounts.

8 Production deployment

The public site runs on one shared server as a single container project: the web server, the console API, the live-update gateway, a control process, a browser worker, the owned board, Postgres, two Redis instances and a local mail catcher. Only the web server accepts connections, and only from the server's own reverse proxy behind a CDN. Every other service sits on an internal network with no route to the internet, runs with a read-only file system, no Linux capabilities and memory, CPU and process limits. The whole stack measured about 730 MB of memory in rehearsal.

Releases are immutable. Each one contains the static console, the rendered server configuration, the backend source at a single commit and a manifest of SHA-256 hashes. Before a release reaches the server it is started on a development PC as the exact production stack and checked end to end: pages and security headers, three shared-login visitors at once, capped controls, owner sign-in with TOTP, audit records, live updates and per-visitor limits. On the server, the live files are compared with their local copies before deploying and again afterwards; a deploy only counts as done when every hash matches.

The address of each visitor matters, because it drives rate limits, lockouts and the audit log. The proxy chain trusts only the hop it can see: the visitor's address reported by the CDN is believed only when the connection really comes from the CDN, and any address a client writes into its own headers is ignored. IPv6 visitors are counted per /64 block, the range one subscriber normally receives, so rotating through addresses does not escape the limits. A cap on total live streams sits below what the web server and the gateway can hold.

9 Security design

Security is designed in layers that are each tested. Tenant isolation relies on row-level security rather than on remembered filters in application code, and the test suite fails if forced security is removed from any table. Database roles are separated: an owner role for migrations, an application role, a worker role with column-level grants only, and a separate role for the owned board.

Target-site passwords and sessions use envelope encryption. Each record gets its own AES-256-GCM data key, wrapped by a master key, and the associated data binds the ciphertext to its tenant, account and purpose, so a sealed value copied into another row does not decrypt. In a cloud deployment the wrapping moves to a key management service without changing callers.

Console sign-in requires a password hashed with argon2id and a TOTP code for every account, with lockout per account and per network range and one-time use of each personal code. Password hashing runs outside the request loop with a fixed concurrency limit, and every password attempt from one network range counts towards a window, because the shared password is public and a correct password would otherwise be free work for an attacker. Sessions use secure, HTTP-only cookies and every change needs a CSRF token. The console ships with a content security policy and related headers. Every file edit is scanned for secrets and Python security issues, every commit also runs semgrep, and the same scanners run in GitHub Actions. Findings are fixed, never suppressed. All SQL is static and parameterised.

Threat	Response	Status
Tenant data crossing tenants	Forced row-level security, tested and mutation-checked	In place
Stale worker acting on a lost claim	Fencing tokens and database-clock leases	In place
Database dump exposing sessions	Envelope encryption with row-bound associated data	In place

Threat	Response	Status
Secrets in logs or source control	Log-leak test; secret scanning on edit, commit and push	In place
Operator account takeover	Argon2id, mandatory TOTP, lockout, replay protection, audit log	In place, live
public shared login used to exhaust the server	Bounded hashing, per-range attempt window, per-visitor limits	In place, live
Forged client address to dodge limits or poison the audit log	Trust only the visible hop; CDN header only from the CDN	In place, tested
Address rotation to bypass limits	IPv6 counted per /64; total stream cap	In place, tested
Server-side request forgery from crawl targets	Egress guard after DNS resolution	Planned
Prompt injection into the AI agent	Human approval before publish, send, purchase or delete	Planned

10 Law and acceptable use

The legal boundary shaped the project as much as the technology. In the United States, courts have read the Computer Fraud and Abuse Act narrowly for public pages since Van Buren and hiQ v. LinkedIn, and the fight has moved to contract law. Meta v. Bright Data held that logged-out scraping of public data did not breach Meta's terms, but terms do bind logged-in use, which is exactly what a booking fleet does. Major freight load boards forbid automation outright: DAT's terms, last updated in July 2026, prohibit bots, AI agents and browser extensions used to access the product without written approval and allow immediate termination.

Fleetwright therefore ships with its owned board, contains no code that defeats third-party captchas or access controls, and requires a written onboarding gate before any real target: the client must show that the platform permits automation or that it owns the platform. The live public workspace runs only against that built-in board, on an internal network that cannot reach the internet. For European Union deployments, the legal register covers GDPR minimisation, retention, data-subject rights and deployment-specific controller/processor responsibilities, necessary cookies, potential AI Act transparency duties and Cyber Resilience Act applicability. These require operator assessment and organizational implementation; repository controls alone do not establish compliance. The full register is in the repository's legal documentation and is explicitly not legal advice.

11 Scaling to one million sessions and 99.9 percent availability

Two different scales matter. The control plane, meaning the API, the console and live updates, is designed for one million concurrent connected sessions across many tenants. The worker plane is designed to grow from tens to more than ten thousand browser contexts by adding cells. A million automated browsers would need roughly one hundred terabytes of memory, which is a different problem that the design does not claim to solve.

The availability target for the cloud reference architecture is 99.9 percent per month, about 43 minutes of downtime, which is stricter than the 99 percent originally requested (about 7.2 hours). The public workspace runs on a single shared server with no redundancy and no backup job yet, so it claims no availability figure at all. Correctness is not treated as a service level: a single duplicate booking is an incident, never part of an error budget.

Stage	Shape	Capacity	Availability
Today	One shared server running the live workspace; development PC for load tests	4 live slots on the shared workspace; measured up to 32 contexts on a PC	No claim

Stage	Shape	Capacity	Availability
Single production server	Dedicated server with backups and monitoring	About 75 contexts per 8 GB node (estimate)	About 99 percent with good operations (estimate)
Cloud, one region, multi-AZ	Managed Postgres and Redis, autoscaled containers, load balancer and firewall	Hundreds of contexts, thousands of console sessions	99.9 percent target
Many cells	Tenants spread across cells, gateway fleet	More than 10,000 contexts (target)	99.9 percent per cell
One million sessions	Gateway fleet, CDN for the console, read replicas, partitioned history tables	1,000,000 connected sessions (target)	99.9 percent; 99.95 needs multi-region

The seams that make this possible are already in the code and tested: tenant identifiers leading every key and index, stateless services, cell routing, hash-tagged Redis keys, an outbox with idempotent consumers, bounded queries and one typed configuration boundary. Kubernetes, Kafka, multi-region deployment and database sharding are deliberately not built yet; each has a written trigger that would justify it. The gateway's capacity per process has not been load-tested yet.

12 Verification and measured results

Correctness is checked against the board's own ground-truth log, so the system is never marking its own homework. Ninety-nine automated tests pass against real Postgres, Redis, a mail server, a real Chromium and the real owned board, alongside type checking, linting and security scanning on every commit.

Measurement	Result
Duplicate bookings with 200 concurrent competitors and 10,000 jobs	0; 9,897 claims confirmed equal 9,897 bookings on the board
Crashes injected during the load test	184 after leasing and 213 during the action, all re-queued or reconciled
Chaos run with lost acknowledgements, errors, slow responses and rival bots	Zero duplicates; every lost answer reconciled
Detection to confirmed booking	Median about 365 ms, 95th percentile about 540 ms
Memory per browser context	About 70 to 103 MB; 100 MB used for planning
CPU at 32 contexts	About 1.8 cores
Tenant isolation	Tenant A cannot read or write tenant B rows, even as the table owner
Live site end-to-end check through the CDN	0 failures: three shared-login visitors at once, capped controls, owner sign-in with TOTP, audit, live updates
Live update reaching a client through the CDN	First engine update 5.8 seconds after connecting (one live measurement, with the owned feed at its idle rate)
Per-visitor limits on the live site	A 160-request burst from one visitor: 71 answered, 89 refused with 429

Measurement	Result
Client address tests in rehearsal	Forged headers ignored in every case; IPv6 visitors grouped per /64
Release integrity	Every deployed file matches its recorded hash before and after the deploy
Console data on screen, repeat visit	About 1.4 seconds, measured from Bangladesh

Several checks were broken on purpose to prove they could fail: removing forced row-level security from one table fails the isolation suite, writing a secret into a log line fails the leak test, and undoing the shared login fix makes the three-visitor test fail. The large measurements come from one developer PC against the owned board; the 50 and 70 context one-hour runs are still to be done.

Each phase also had a review by a reviewer that did not write the code. For the live console, the first review said not to ship: the shared login let in only one visitor per 30 seconds, password hashing could stall the API, and there were no per-visitor limits. All three were fixed with tests. The second review found that IPv6 visitors could avoid the limits; that was fixed and tested before the deploy.

13 Failures and the engineering lessons

The reconciler's sweep blocked workers under load. In the 10,000 job test, worker lease updates hit the five-second statement timeout because the sweep moved many claims in one statement and then wrote history rows one by one inside the same transaction, holding locks for seconds. Each sweep rule became a single statement that moves rows and writes history and outbox entries together, and the reconciler stopped holding locks while it called the target. Lesson: never hold a lock across a loop of round trips, and never across a network call.

A green concurrency test proved almost nothing. The first passing run booked only 77 of 10,000 jobs: leases were so short that jobs aged out before anyone acted, and "zero duplicates" was trivially true. Production-like timings and coverage assertions were added, requiring at least 95 percent of jobs booked and at least one crash during the action reconciled. Lesson: a passing test must also prove it exercised the dangerous path.

A browser call with no timeout froze a worker slot. About one run in three, after the target revoked a session, the slot never signed in again and logged nothing. A dump of every suspended task pointed at a page close call, which has no timeout, waiting on a page that was redirecting itself. Sign-in now uses a new page and closes the old one in a bounded background task. Lesson: every call into a browser needs a deadline, especially the ones without a timeout parameter.

An old cell could publish work after its tenant had moved away. The relay's cached tenant list was a few hundred milliseconds stale, so it published one message to the wrong Redis. The relay now checks ownership inside its own transaction with a share lock that a move must wait for. Lesson: a cached ownership list is a hint; the check that guards a side effect belongs in the same transaction.

Click mode froze claimers. A response body read never returned, and the button lookup ran after the claim entered the acting state, so a slow lookup plus the click could outlive the lease. Body reads are now bounded, harmless steps run before acting, and the fleet refuses to start unless the lease outlasts the action timeout; that check immediately caught a wrong setting in the test kit. Lesson: do the slow, harmless steps before taking a lock that has a deadline, and make the deadline relationship a startup check.

The first capacity number was wrong by a factor of two. Summing resident memory across Chromium processes counted shared memory many times, the measurement included the test's own board server, and CPU always read zero because fresh process objects were created for every sample. Unique memory per process and persistent process objects gave 70 to 103 MB per context and about 1.8 cores at 32 contexts, and the same CPU bug was fixed in the worker heartbeat. Lesson: before planning capacity on a number, check what the meter actually counts.

The public shared login let in one visitor per 30 seconds. One-time codes are single use per account, which is right for a personal account and wrong for a shared one whose code is published: the second visitor in the same 30 seconds was rejected, and each retry counted towards a lockout. My own browser check signed in once, so it passed. The shared account now skips the per-account replay check while every personal account keeps it, and a test signs in three visitors with the same code. Lesson: a security control that is right for one user can be a denial of service for a shared account, so test the shared path with more than one user.

My own design for reading the visitor's address could be forged. The first proxy configuration skipped every forwarded hop inside the CDN's published address ranges to find the visitor, but code running on the CDN's own platform sends requests

from those same ranges, so anyone could relay through it and choose the address that rate limits and the audit log would believe. The proxy now reads only the hop it can see and accepts the CDN's visitor header only when that hop is the CDN. Lesson: trusted address ranges are only as trusted as everyone else who can send from them.

A security test passed because it could not see. A check that a backend container could not forge its address reported success, but the audit value it read was "hidden": its owner sign-in had silently reused a one-time code an earlier script had spent, so it read the log as an anonymous visitor with addresses masked. The scripts now require the sign-in to succeed and wait for a fresh code, and the rerun showed the real range. Lesson: assert a security test's preconditions, or a negative check passes for the wrong reason.

The production smoke test was quietly steering the live engine. It proved that an owner can create a filter by creating one, and the API has no delete, so every run would have added another active rule to the engine visitors were watching. The test now reuses one filter and leaves it disabled. Lesson: a smoke test against production is a production user and has to clean up like one.

14 Roadmap and known limitations

The remaining Phase 6 item is metrics and alerting. After that come a crawler with politeness, robots and opt-out handling and server-side request forgery protection; a lightweight local AI browser agent with human approval for risky actions; and a hardening phase with an OWASP ASVS Level 2 checklist, dynamic scanning, software bills of materials and signed images, the 50 and 70 context one-hour fleet runs, a gateway load test, a restore drill and a cloud reference architecture in infrastructure as code.

Known limitations are recorded rather than hidden. Blocked responses mark a claim as failed instead of re-queuing it to another account. Per-account rate pacing and routing by account group are configured but not yet enforced by the dispatcher. Detections in flight during a tenant move may be dropped. Re-planning after a tenant change restarts every slot in that worker. The public site runs on one server with no backup job yet; the workspace data is re-created by a seed job. There are no metrics or alerts yet. The audit entry for a console action is written in its own transaction just after the action. A revoked session keeps an already-open live-update stream until it reconnects. The content security policy still allows inline scripts and styles. The extra load from many open consoles refetching on each update has not been load-tested, and the tests share the local development databases.

15 Source basis

This edition draws on the Fleetwright source code, its automated test suite and test output, the project's measurement runs, the release rehearsal and live-site checks, its architecture decision records and public documentation, and the legal sources cited in the repository's compliance register, checked on 5 and 6 October 2026. Private planning notes, credentials and infrastructure details remain in private project records.

The strongest claims are supported by executable tests against real infrastructure and a ground-truth log kept by the target itself. Design targets for scale and availability are labelled as targets throughout and are not presented as achieved capacity.