DevBlock Console Master Handbook

An umbrella guide that maps the entire workspace and points into each project manual

This handbook links the system overview, client, server-worker, SDK, ingest-server, and snitch-worker manuals into one operator-facing reference path.

Table of contents

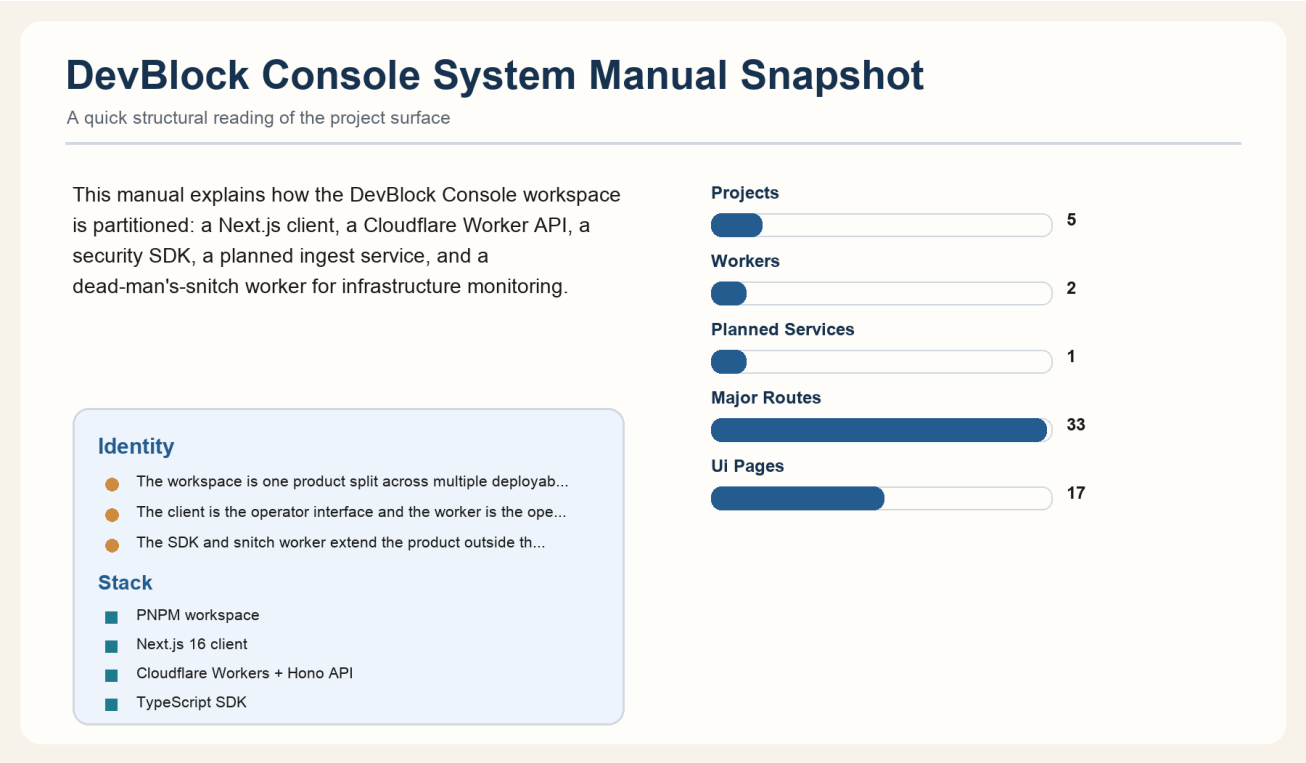
Recommended reading order: system manual first, then client and server-worker, then SDK, then ingest-server and snitch-worker as specialized appendices.

DevBlock Console System Manual

Workspace-level map of the frontend, backend, SDK, and monitoring surfaces

This manual explains how the DevBlock Console workspace is partitioned: a Next.js client, a Cloudflare Worker API, a security SDK, a planned ingest service, and a dead-man's-snitch worker for infrastructure monitoring.

Who should read this chapter: Start here when you need the big picture before drilling into a specific package, service, or deployment surface.



DevBlock Console System Manual chapter snapshot

What it owns

- The workspace is one product split across multiple deployable projects.
- The client is the operator interface and the worker is the operational backend.
- The SDK and snitch worker extend the product outside the core dashboard runtime.
- The ingest-server is a future boundary that is already important architecturally.

Why it matters

- Think of the workspace as a control-plane product with several runtime edges.
- The server-worker is not just an API; it is the integration hub and the strongest architectural gravity well.
- The client reflects the backend domain structure closely, which makes route naming a useful orientation tool.

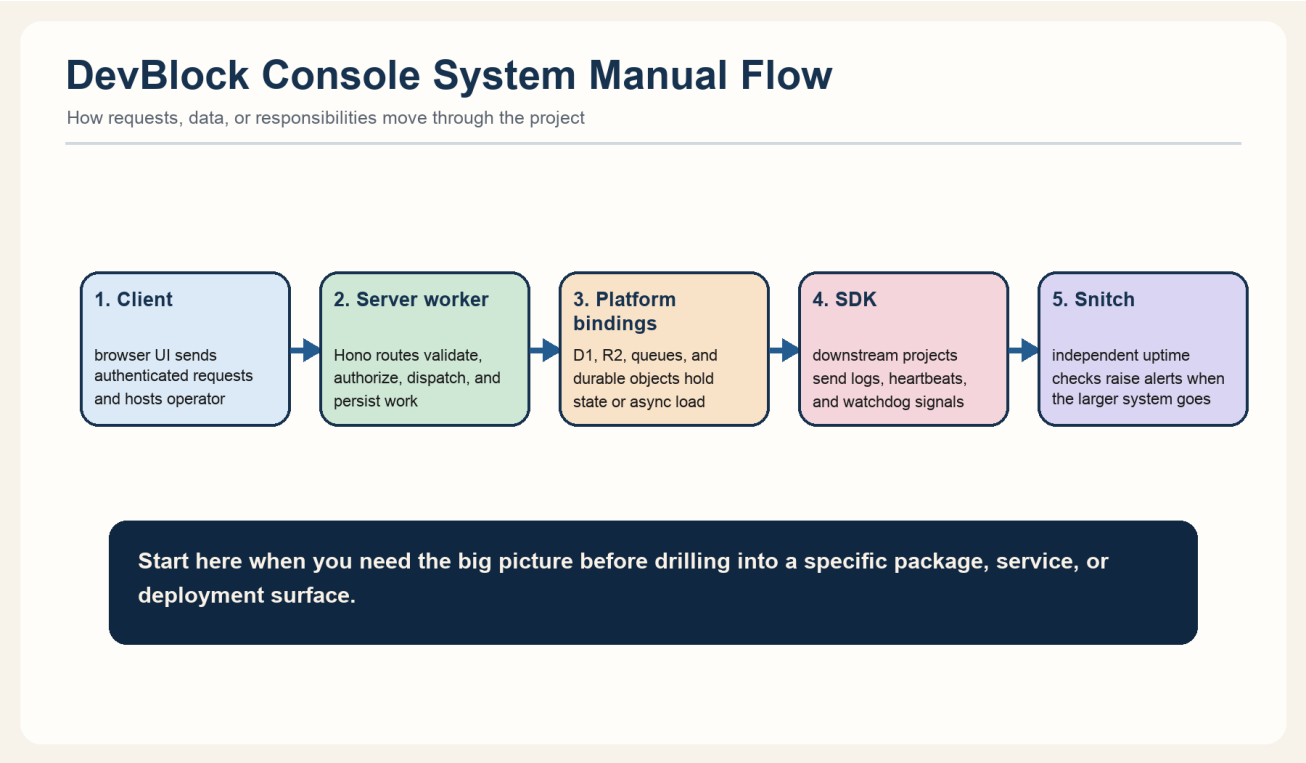
How to approach changes

- Start with the system manual whenever a change crosses more than one package boundary.
- Map the change to a primary owner first: client, server-worker, sdk, ingest-server, or snitch-worker.
- Confirm whether the change affects secrets, auth, queue flow, or browser runtime assumptions before editing.

Metric	Count
Projects	5
Workers	2
Planned Services	1
Major Routes	33
Ui Pages	17

Back to contents

DevBlock Console System Manual runtime flow



DevBlock Console System Manual runtime flow

Top risks

- Cross-project changes can silently break auth, routing, or environment wiring.
- The workspace has both active and planned services, so docs and code can drift in different directions.
- OAuth and cloud integrations live in the worker, which keeps secrets and bindings as an ongoing change hotspot.

Validation notes

- Validate per project surface instead of relying on one global command.
- For UI-only changes, targeted lint and type checks are more useful than noisy repo-wide validation.
- For backend changes, re-check bindings and local bootstrap assumptions, not just TypeScript.

Full manual filename: system-manual.pdf

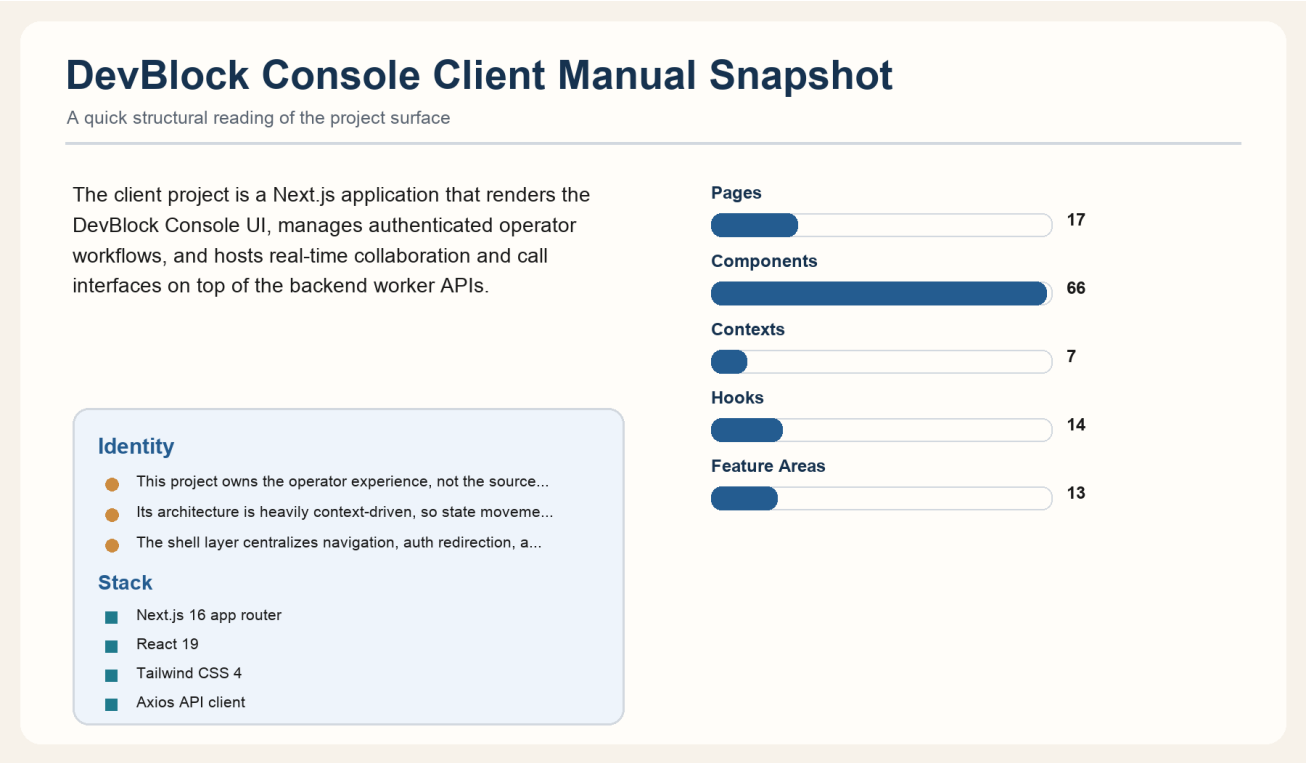
Back to contents

DevBlock Console Client Manual

Frontend operations, layout, contexts, route surfaces, and browser behavior

The client project is a Next.js application that renders the DevBlock Console UI, manages authenticated operator workflows, and hosts real-time collaboration and call interfaces on top of the backend worker APIs.

Who should read this chapter: Use this when changing the UI shell, route behavior, contexts, browser-side API integration, or operator-facing feature pages.



DevBlock Console Client Manual chapter snapshot

What it owns

- This project owns the operator experience, not the source of truth for business state.
- Its architecture is heavily context-driven, so state movement matters as much as JSX structure.
- The shell layer centralizes navigation, auth redirection, and global call overlays.
- Feature pages mirror backend domains closely: dashboard, projects, files, logs, clients, chat, settings.

Why it matters

- The client is best understood as a shell-plus-context system rather than a set of isolated pages.
- Global UX features such as auth-expiry handling and active calls are mounted above page level.
- The folder layout under `components/` reveals feature ownership more clearly than the generic README does.

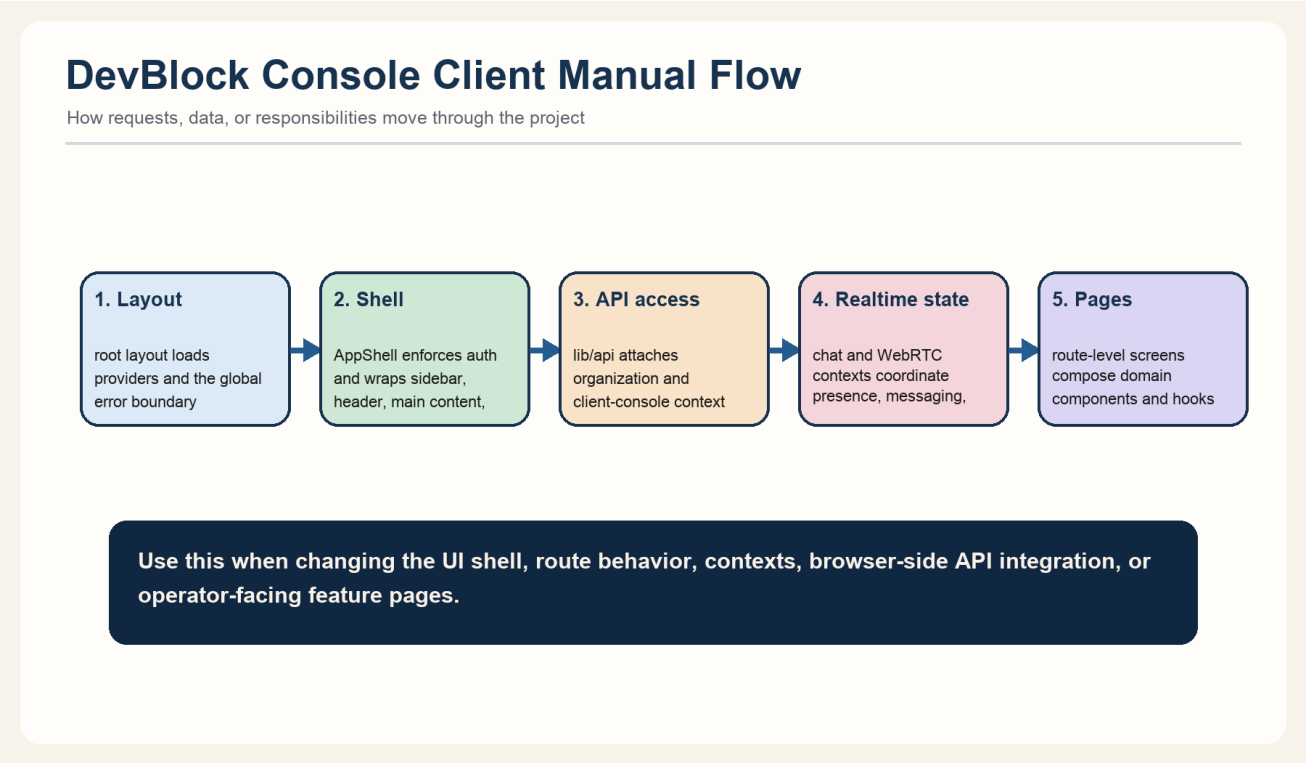
How to approach changes

- Start from the visible feature area, then trace upward into the owning context and shell if the behavior feels global.
- Inspect `lib/api.ts` before assuming a frontend bug is local; request shaping and redirect behavior live there.
- Use route files to identify screen entrypoints, then inspect hooks and context providers for state movement.

Metric	Count
Pages	17
Components	66
Contexts	7
Hooks	14
Feature Areas	13

Back to contents

DevBlock Console Client Manual runtime flow



DevBlock Console Client Manual runtime flow

Top risks

- Context coupling can create hidden side effects across screens.
- The default client README is generic and understates the actual product complexity.
- Browser-only assumptions in auth and localStorage flows must stay guarded for SSR and hydration safety.

Validation notes

- Prefer targeted lint and type checks for touched files because the broader frontend baseline can be noisy.
- Any auth or redirect change should be tested through the shell, not just with unit-level reasoning.
- Visual changes in shared layout surfaces should be checked on at least one representative feature page.

Full manual filename: client-manual.pdf

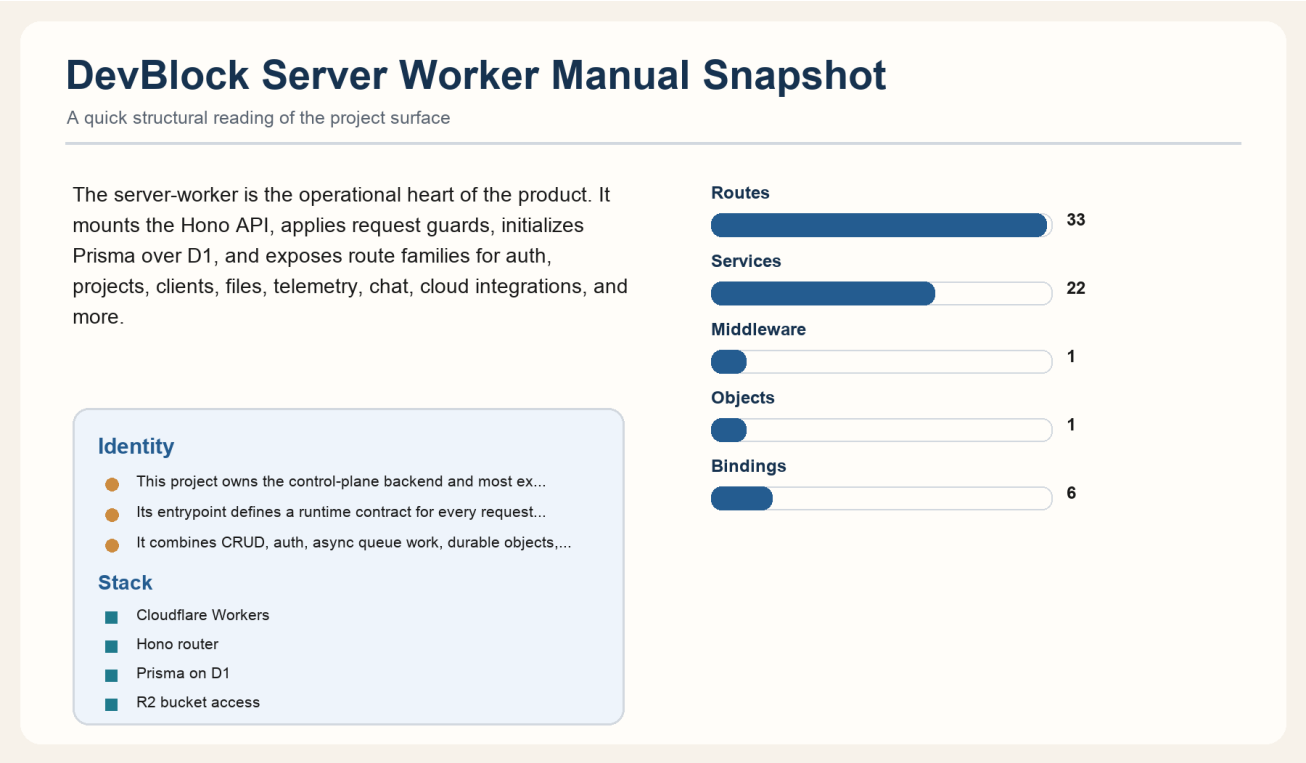
Back to contents

DevBlock Server Worker Manual

API entrypoint, middleware, route families, services, and Cloudflare bindings

The server-worker is the operational heart of the product. It mounts the Hono API, applies request guards, initializes Prisma over D1, and exposes route families for auth, projects, clients, files, telemetry, chat, cloud integrations, and more.

> **Who should read this chapter: Use this when changing backend behavior, bindings, middleware, route ownership, or cloud integration logic.**



DevBlock Server Worker Manual chapter snapshot

What it owns

- This project owns the control-plane backend and most external integrations.
- Its entrypoint defines a runtime contract for every request before route code even starts.
- It combines CRUD, auth, async queue work, durable objects, and provider integrations in one deployment surface.
- This is the package most likely to accumulate architectural pressure over time.

Why it matters

- The order of middleware in `src/index.ts` matters because it encodes security and runtime assumptions.
- Bindings in `wrangler.toml` are part of the architecture, not just deployment configuration.
- Services provide a softer abstraction boundary than routes, but the project still has a large responsibility surface.

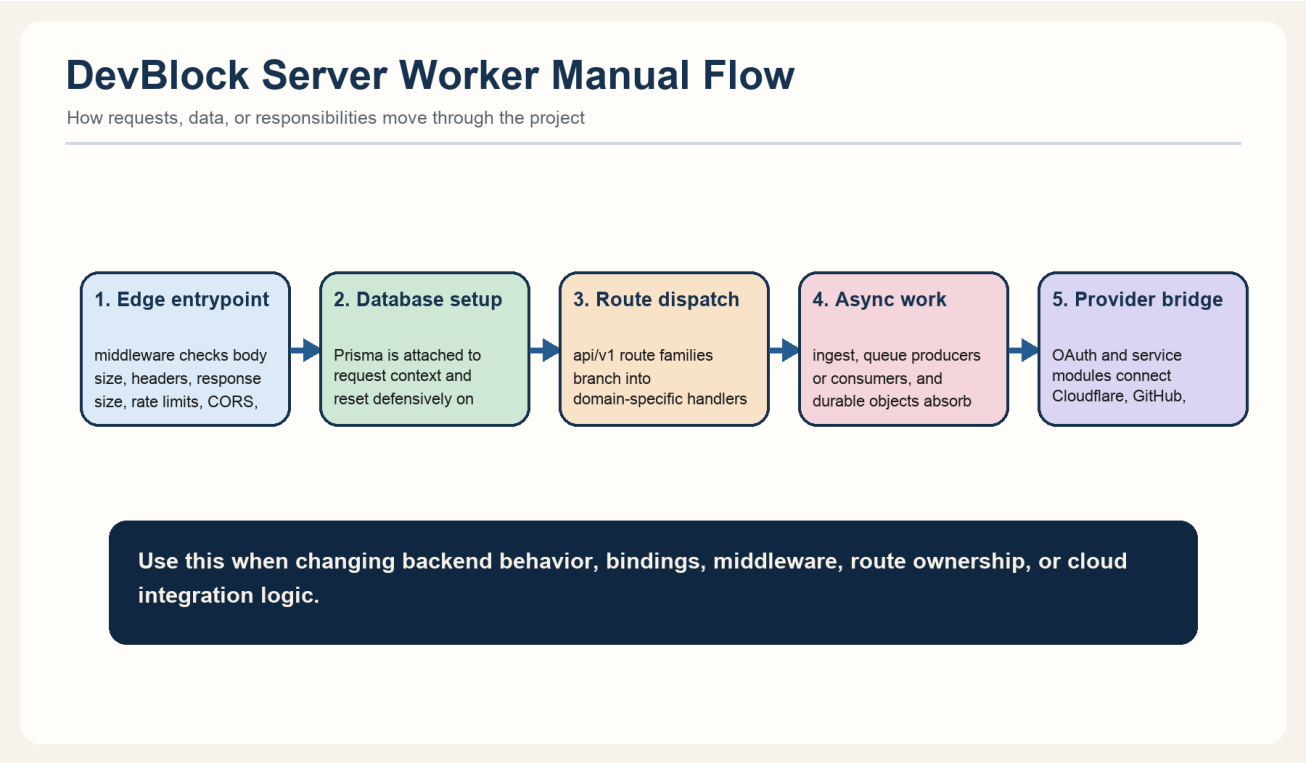
How to approach changes

- Start at `src/index.ts` for any runtime, header, or binding question before jumping into individual routes.
- Map the change to a route family first, then inspect its service module and any shared lib helpers.
- If a change touches integrations, re-check wrangler bindings and local secret assumptions before coding too far.

Metric	Count
Routes	33
Services	22
Middleware	1
Objects	1
Bindings	6

Back to contents

DevBlock Server Worker Manual runtime flow



DevBlock Server Worker Manual runtime flow

Top risks

- This worker has a broad responsibility surface, so cohesion can erode over time.
- Integration routes are sensitive to secret drift, environment mismatch, and provider API changes.
- In-memory rate limiting is per-worker-instance and should be treated as best-effort rather than global enforcement.

Validation notes

- Validate backend changes with route-aware smoke testing rather than TypeScript alone.
- Any binding or wrangler change should be treated as architecture-sensitive and re-checked locally.
- Ingest and realtime changes need extra attention because their failure modes are often delayed or indirect.

Full manual filename: server-worker-manual.pdf

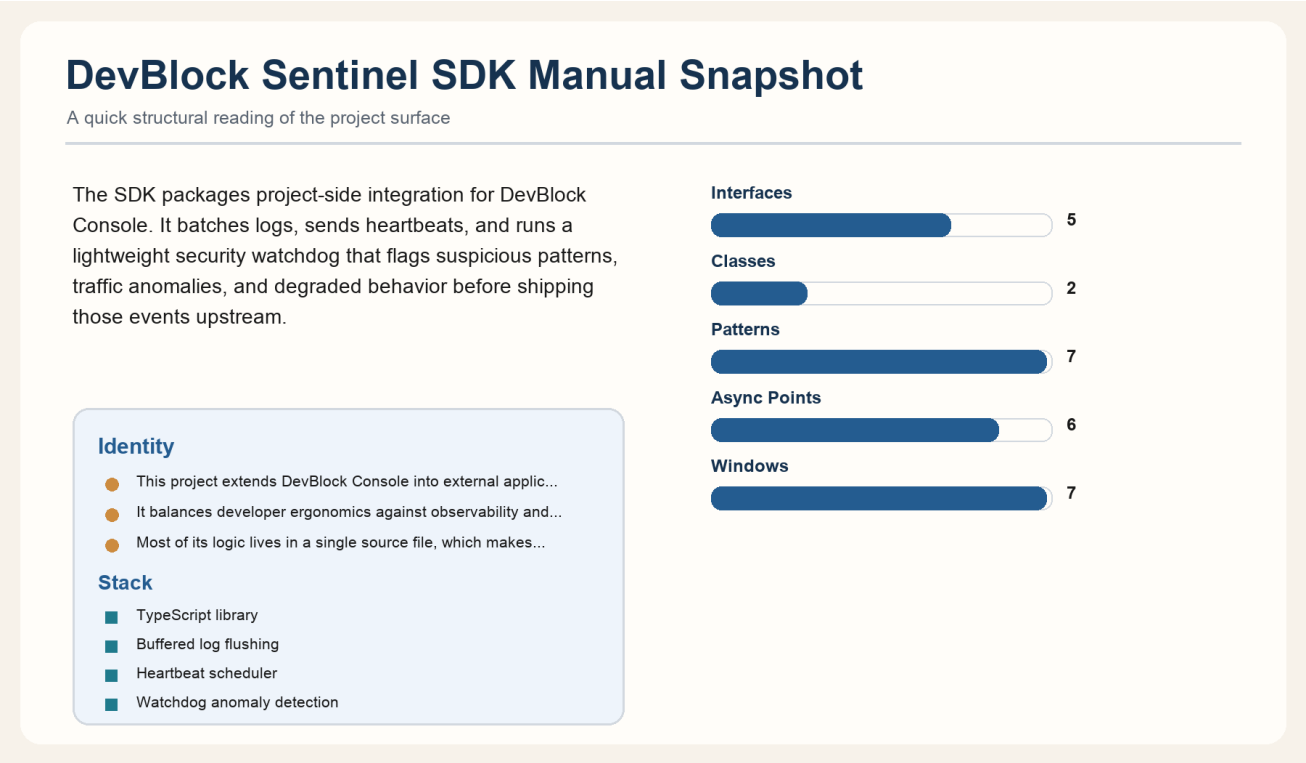
Back to contents

DevBlock Sentinel SDK Manual

Telemetry, batching, heartbeat, and security-watchdog behavior

The SDK packages project-side integration for DevBlock Console. It batches logs, sends heartbeats, and runs a lightweight security watchdog that flags suspicious patterns, traffic anomalies, and degraded behavior before shipping those events upstream.

Who should read this chapter: Use this when integrating customer projects into DevBlock Console or changing the telemetry, buffering, and watchdog contract.



DevBlock Sentinel SDK Manual chapter snapshot

What it owns

- This project extends DevBlock Console into external applications rather than rendering product UI.
- It balances developer ergonomics against observability and false-positive risk.
- Most of its logic lives in a single source file, which makes reading easy and maintenance dense.
- Its behavior affects perceived trust in the broader platform because it determines what gets reported upstream.

Why it matters

- The SDK acts like a small runtime engine rather than a thin HTTP wrapper.
- Watchdog defaults are part of the product contract because they shape alert quality and noise.
- Sliding windows are core to the anomaly model, so threshold edits are architectural changes, not cosmetic tweaks.

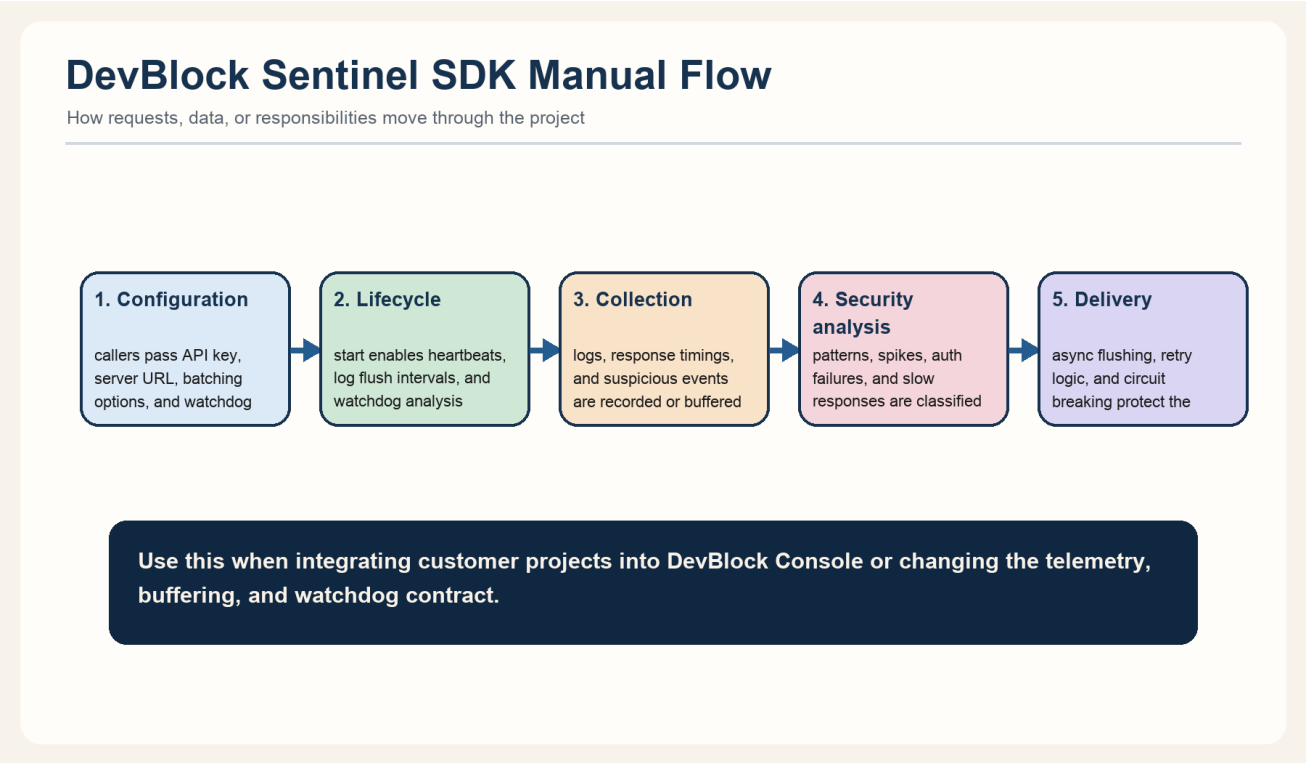
How to approach changes

- Read the public config interfaces first because they define how host projects experience the SDK.
- Trace `start`, buffering, and watchdog analysis together before changing intervals or flush semantics.
- Treat suspicious-pattern lists as encoded threat-model decisions rather than random regex collections.

Metric	Count
Interfaces	5
Classes	2
Patterns	7
Async Points	6
Windows	7

Back to contents

DevBlock Sentinel SDK Manual runtime flow



DevBlock Sentinel SDK Manual runtime flow

Top risks

- A single-file implementation makes local reasoning easy but can become dense fast.
- False positives in watchdog heuristics can damage trust if thresholds drift.
- Any transport failure policy change affects both telemetry completeness and backend load.

Validation notes

- Validate behavior with scenario-based reasoning, not only compile success.
- Threshold changes should be reviewed against realistic traffic patterns to avoid noisy alerts.
- Changes to buffering or stop behavior should consider shutdown and crash boundaries explicitly.

Full manual filename: sdk-manual.pdf

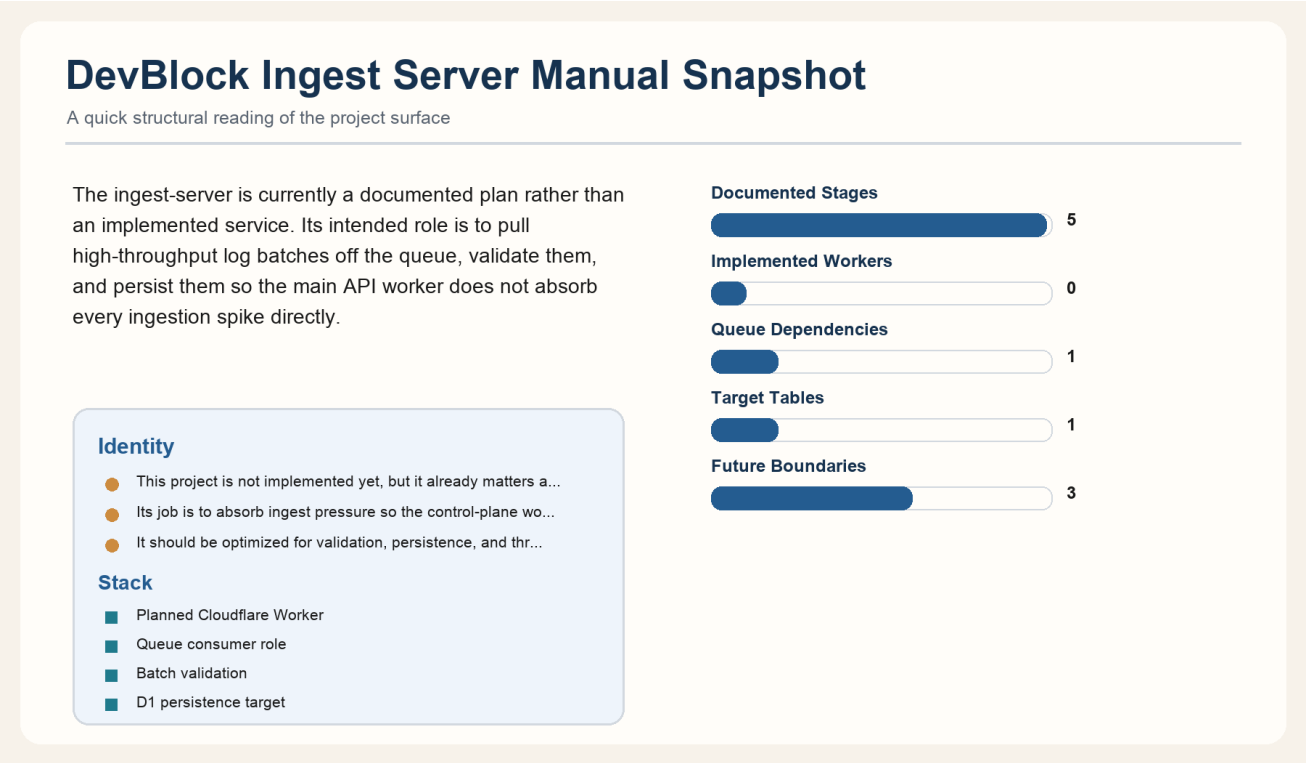
Back to contents

DevBlock Ingest Server Manual

Planned queue-consumer split for high-throughput telemetry ingestion

The ingest-server is currently a documented plan rather than an implemented service. Its intended role is to pull high-throughput log batches off the queue, validate them, and persist them so the main API worker does not absorb every ingestion spike directly.

Who should read this chapter: Use this when planning the queue-consumer split or deciding what should leave the main API worker over time.



DevBlock Ingest Server Manual chapter snapshot

What it owns

- This project is not implemented yet, but it already matters as an architectural boundary.
- Its job is to absorb ingest pressure so the control-plane worker does not carry every burst directly.
- It should be optimized for validation, persistence, and throughput, not broad product logic.
- A good design here reduces coupling in the main worker rather than recreating it elsewhere.

Why it matters

- This service exists to preserve backend focus by separating ingestion throughput concerns from control-plane concerns.
- Its success depends on clean responsibility boundaries more than on raw queue mechanics alone.
- It should only own the ingest lifecycle, not opportunistically absorb unrelated backend tasks.

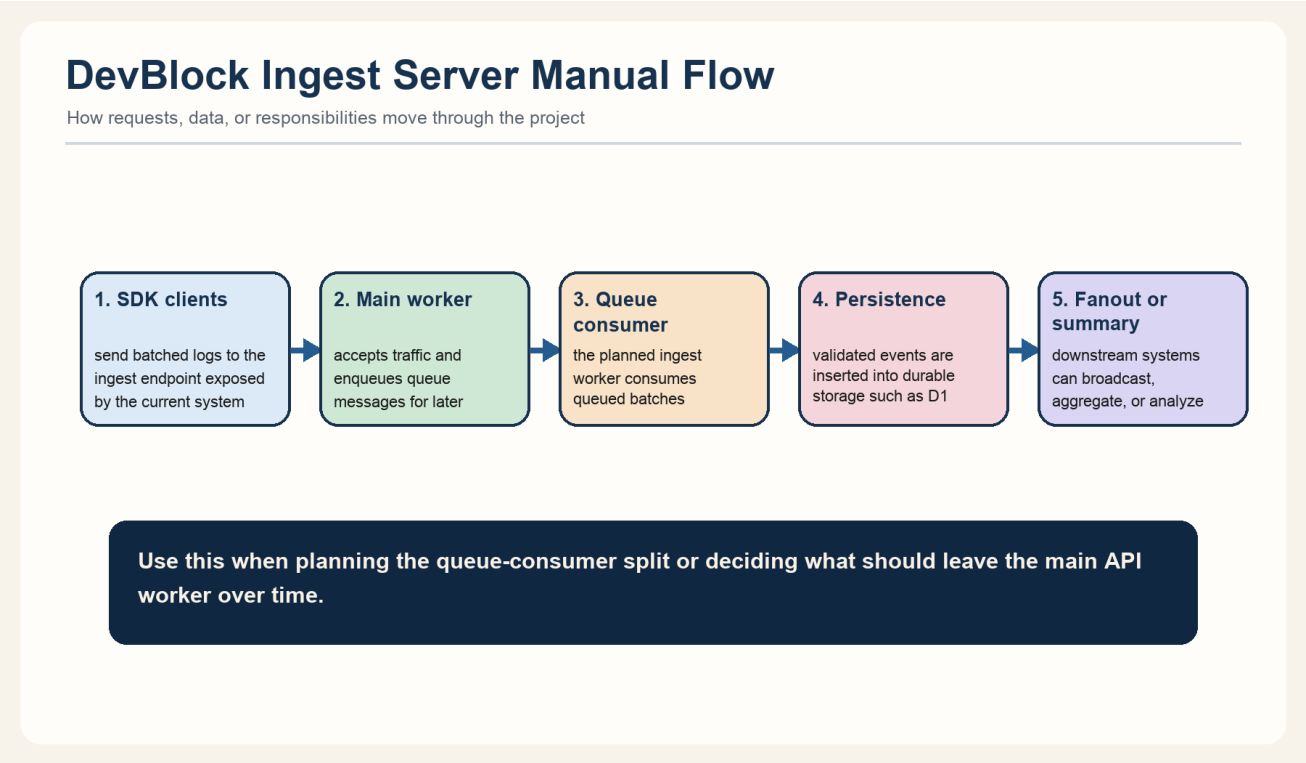
How to approach changes

- Read the ingest README and current worker route together before proposing a split.
- Clarify queue semantics, idempotency, and failure handling before writing any implementation code.
- Use the future service to remove pressure from the main worker, not to duplicate control-plane concerns.

Metric	Count
Documented Stages	5
Implemented Workers	0
Queue Dependencies	1
Target Tables	1
Future Boundaries	3

Back to contents

DevBlock Ingest Server Manual runtime flow



DevBlock Ingest Server Manual runtime flow

Top risks

- Because the service is planned, docs can drift from eventual implementation choices.
- Queue semantics, retry policy, and idempotency strategy still need explicit design.
- The split only helps if ownership boundaries are maintained after implementation.

Validation notes

- Design reviews matter more than code-level checks until the service exists.
- Validate proposed boundaries against actual ingest load and current server-worker responsibilities.
- Insist on explicit retry and idempotency notes before implementation begins.

Full manual filename: ingest-server-manual.pdf

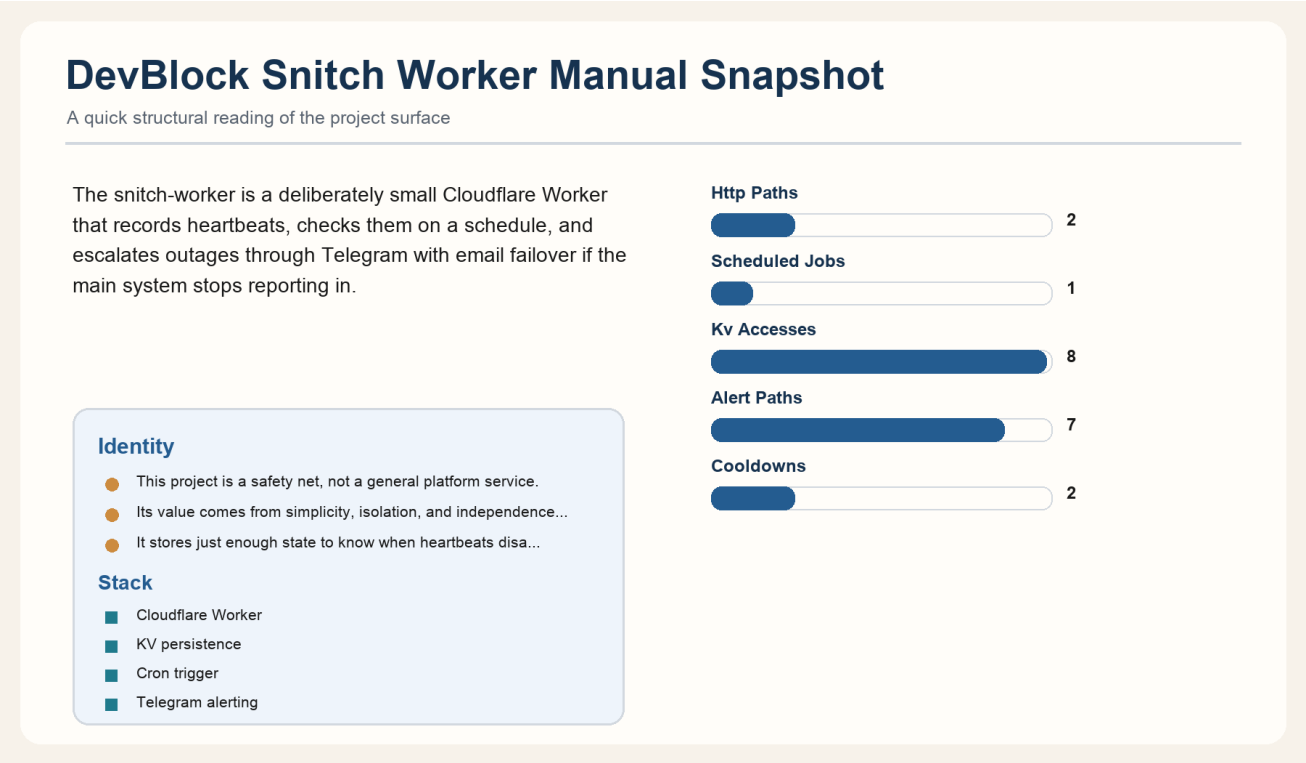
Back to contents

DevBlock Snitch Worker Manual

Heartbeat-loss monitoring, cooldown logic, and escalation failover

The snitch-worker is a deliberately small Cloudflare Worker that records heartbeats, checks them on a schedule, and escalates outages through Telegram with email failover if the main system stops reporting in.

Who should read this chapter: Use this when changing uptime monitoring, alerting thresholds, cooldowns, or heartbeat semantics.



DevBlock Snitch Worker Manual chapter snapshot

What it owns

- This project is a safety net, not a general platform service.
- Its value comes from simplicity, isolation, and independence from the main application stack.
- It stores just enough state to know when heartbeats disappear and alerts need to fire.
- The worker's logic should remain easy to audit because it exists for failure scenarios.

Why it matters

- The worker is intentionally tiny so it stays trustworthy when the larger system fails.
- Its cooldown logic is as important as the alert path because paging loops destroy signal quality.
- Independence from the main application stack is part of its architecture, not just a deployment convenience.

How to approach changes

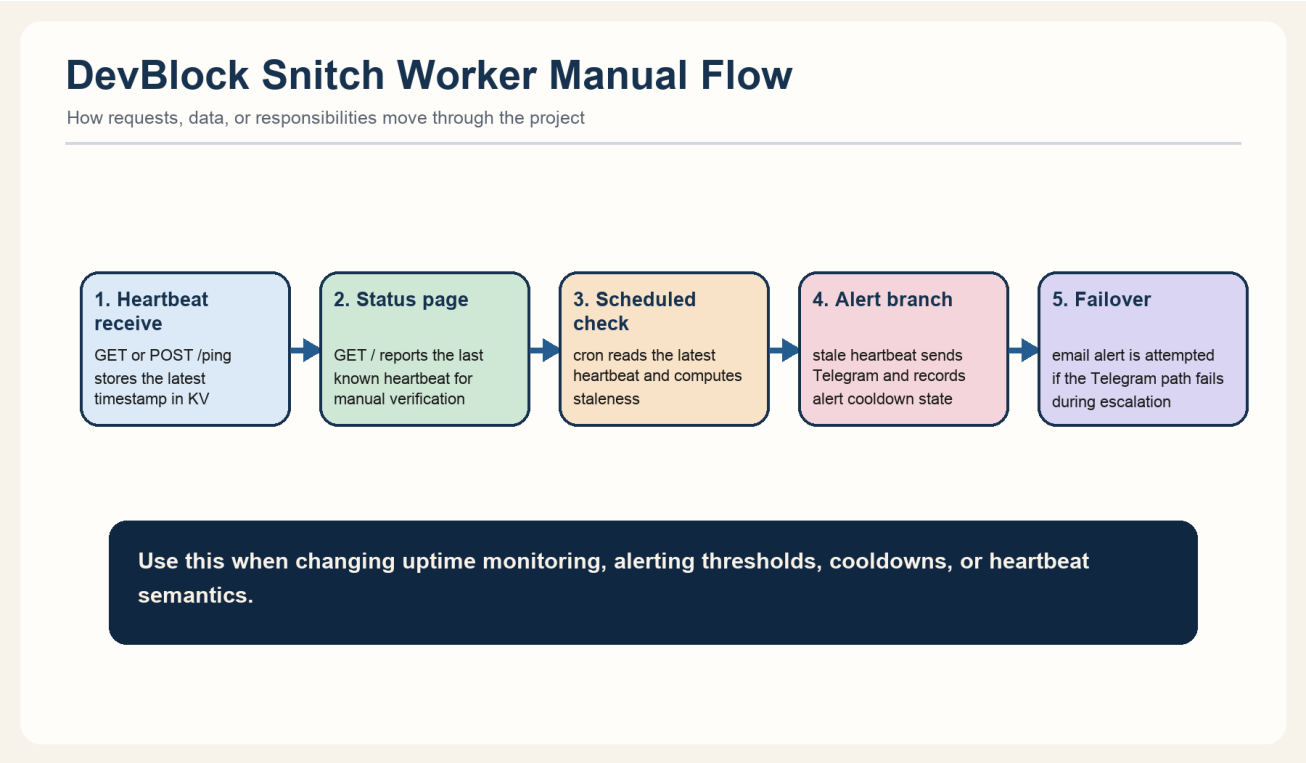
- Read the full worker file before editing because its behavior is compact and tightly coupled.

- Treat heartbeat timing, stale thresholds, and cooldown values as operational policy changes.
- If you change alert providers or secrets, preserve the primary-versus-failover simplicity.

Metric	Count
Http Paths	2
Scheduled Jobs	1
Kv Accesses	8
Alert Paths	7
Cooldowns	2

Back to contents

DevBlock Snitch Worker Manual runtime flow



DevBlock Snitch Worker Manual runtime flow

Top risks

- Alerting depends on external provider credentials and network reachability.
- If heartbeat semantics change in the main system, the snitch threshold must move with them.
- Minimal code is a strength here; turning it into a general monitoring service would weaken it.

Validation notes

- Review stale-threshold and cooldown changes as incident-policy changes, not just code edits.
- Test both the normal heartbeat path and the degraded failover path whenever alert logic changes.
- Preserve manual verification output because operators need a quick truth source during incidents.

Full manual filename: snitch-worker-manual.pdf

Back to contents

Library directory

The generated manuals are intended to live in /Users/odunayojibona/Documents/Books. This chapter exists so the master handbook can point operators to the individual files even outside Codex.

- system-manual.pdf
- client-manual.pdf
- server-worker-manual.pdf
- sdk-manual.pdf
- ingest-server-manual.pdf
- snitch-worker-manual.pdf
- devblock-console-master-handbook.pdf