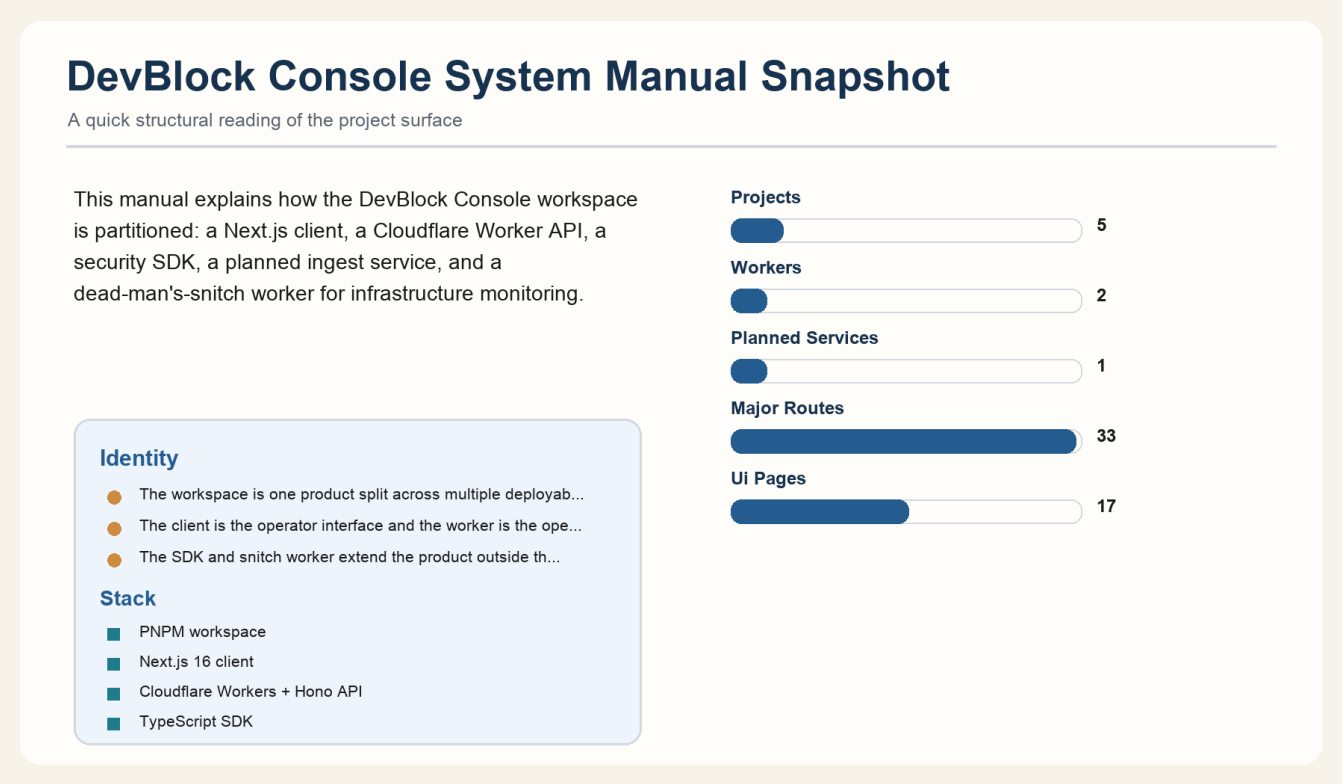
DevBlock Console System Manual

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.



DevBlock Console System Manual visual snapshot

What this project is for

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.

> **Primary audience: Start here when you need the big picture before drilling into a specific package, service, or deployment surface.**

How to identify it quickly

- 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.

Technology stack

- PNPM workspace
- Next.js 16 client
- Cloudflare Workers + Hono API
- TypeScript SDK
- Scheduled monitoring worker

Structural counts

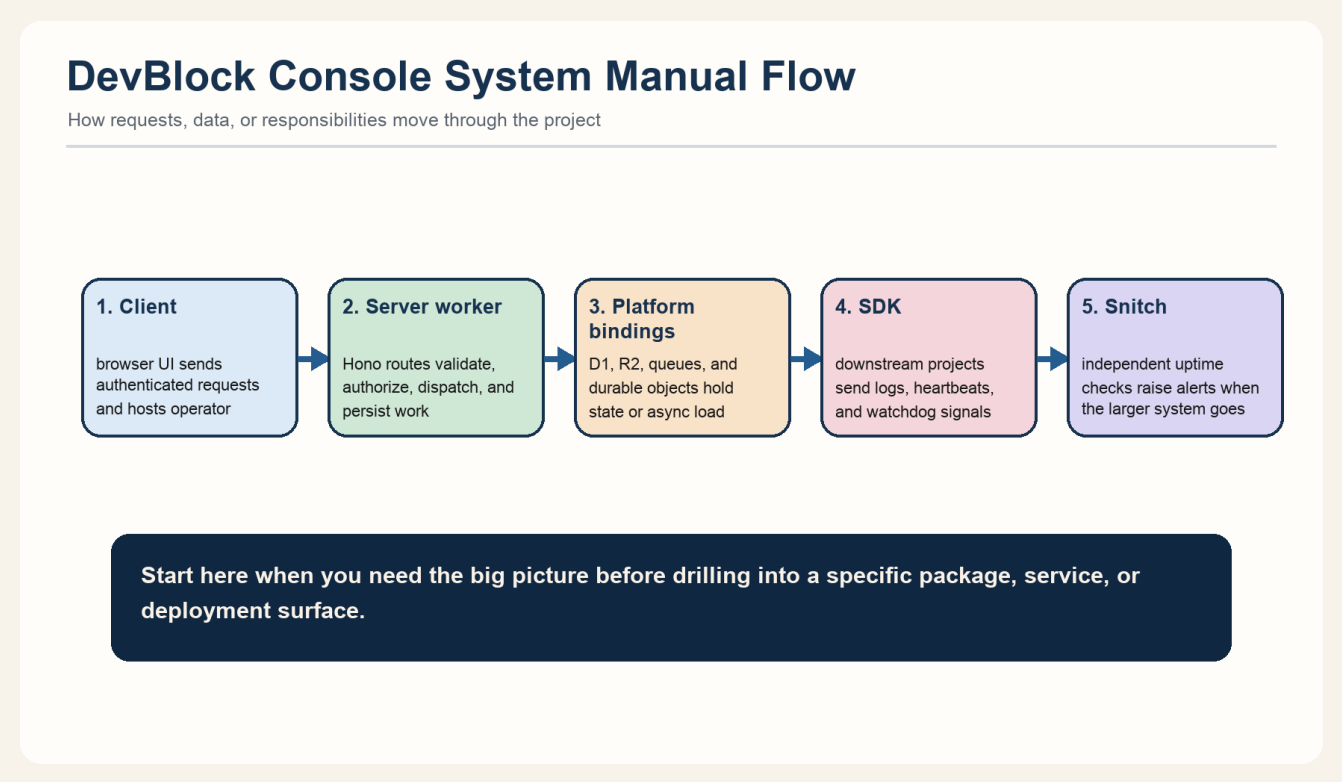
Metric	Count
Projects	5
Workers	2
Planned Services	1
Major Routes	33
Ui Pages	17

Architecture reading

- 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.
- The SDK and snitch worker are smaller in code size but strategically important because they extend trust and observability.

Core flow

This sequence is the shortest useful mental model for how the project behaves at runtime or during operator use.



DevBlock Console System Manual responsibility or data flow

Responsibility matrix

DevBlock Console System Manual Responsibility Matrix

What this project owns versus what it coordinates

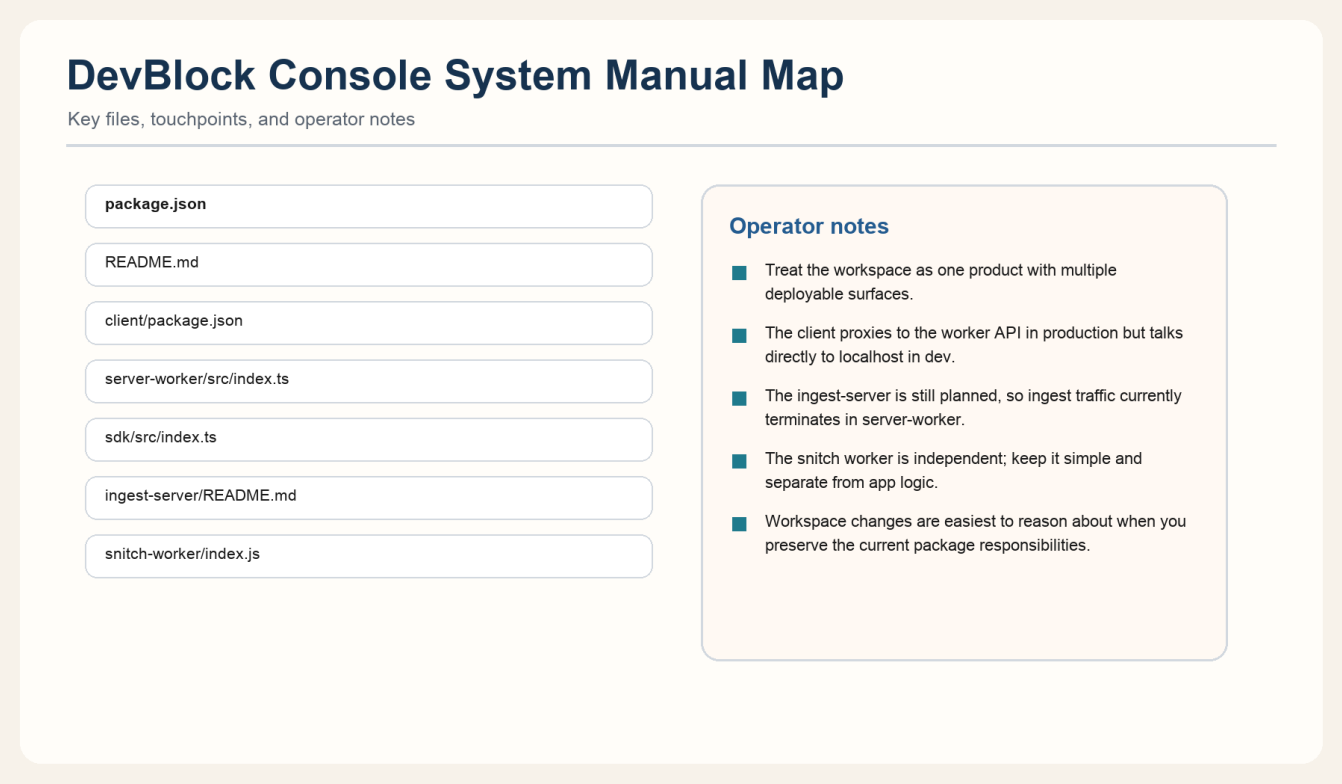
Area	Owns	Coordinates with	Failure mode
Product role	The workspace is one product split across multiple deployable projects.	The client is the operator interface and the worker is the operational backend.	Cross-project changes can silently break auth, routing, or environment wiring.
Main inputs	Client: browser UI sends authenticated requests and hosts operator workflows	Server worker: Hono routes validate, authorize, dispatch, and persist work	The workspace has both active and planned services, so docs and code can drift in different directions.
Change surface	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.	OAuth and cloud integrations live in the worker, which keeps secrets and bindings as an ongoing change
Operator posture	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.	Treat the workspace as one product with multiple deployable surfaces.

DevBlock Console System Manual ownership matrix

Key files to read first

If you are new to this project, read these files in order before making broad edits.

- package.json
- README.md
- client/package.json
- server-worker/src/index.ts
- sdk/src/index.ts
- ingest-server/README.md
- snitch-worker/index.js



DevBlock Console System Manual orientation map

Change workflow

- 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.
- Use targeted validation per surface rather than assuming one monorepo command gives enough confidence.

Operator notes

- Treat the workspace as one product with multiple deployable surfaces.
- The client proxies to the worker API in production but talks directly to localhost in dev.
- The ingest-server is still planned, so ingest traffic currently terminates in server-worker.
- The snitch worker is independent; keep it simple and separate from app logic.
- Workspace changes are easiest to reason about when you preserve the current package responsibilities.

Validation posture

- 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.

Current risks or watchpoints

- 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.

Glossary

Term	Meaning
Control plane	The dashboard and backend system that governs projects and integrations.
Runtime edge	Any separately deployed surface such as the SDK or snitch worker.
Planned boundary	A service that is not implemented yet but already shapes architecture decisions.