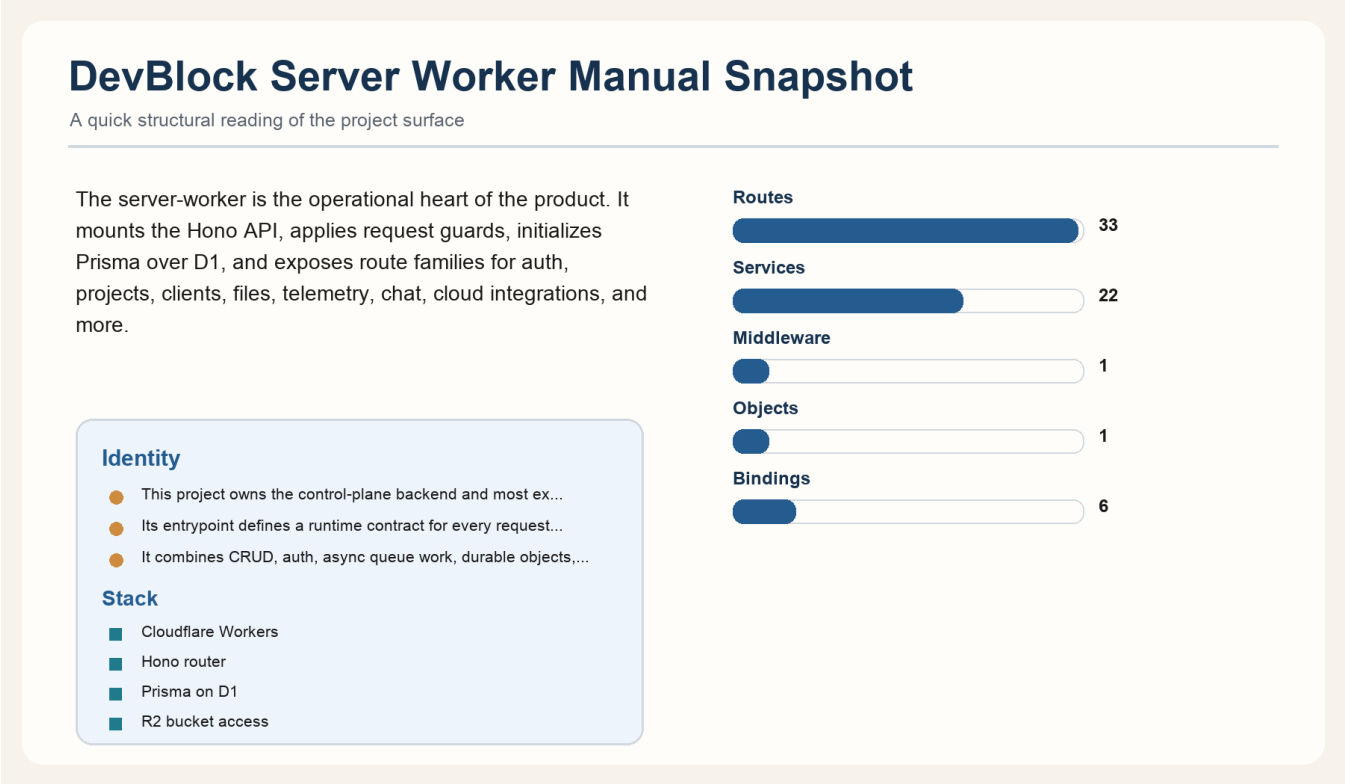
DevBlock Server Worker Manual

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.



DevBlock Server Worker Manual visual snapshot

What this project is for

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.

> **Primary audience: Use this when changing backend behavior, bindings, middleware, route ownership, or cloud integration logic.**

How to identify it quickly

- 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.

Technology stack

- Cloudflare Workers
- Hono router
- Prisma on D1
- R2 bucket access
- Queues + durable objects
- OAuth integrations

Structural counts

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

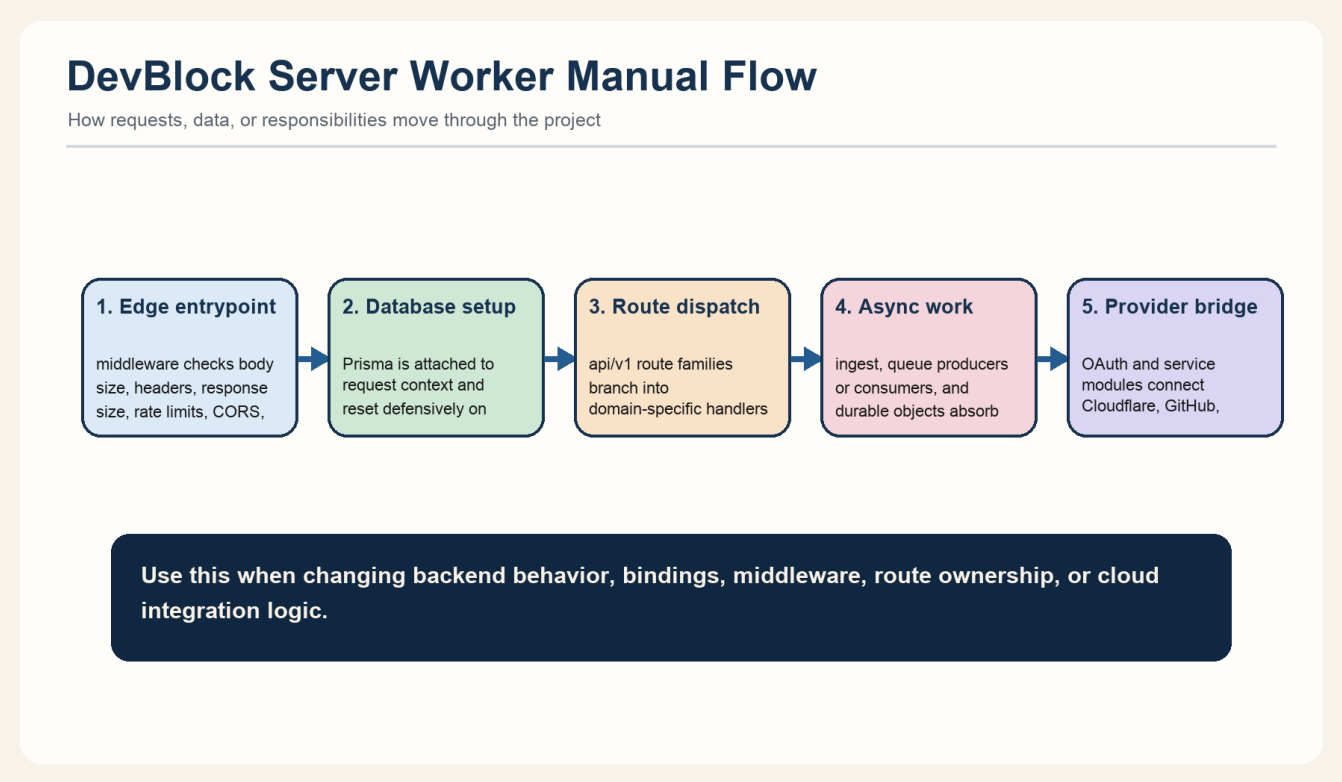
Architecture reading

- 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.

- Read-heavy caching and auth-sensitive exclusions are a key piece of the worker's performance behavior.

Core flow

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



DevBlock Server Worker Manual responsibility or data flow

Responsibility matrix

DevBlock Server Worker Manual Responsibility Matrix

What this project owns versus what it coordinates

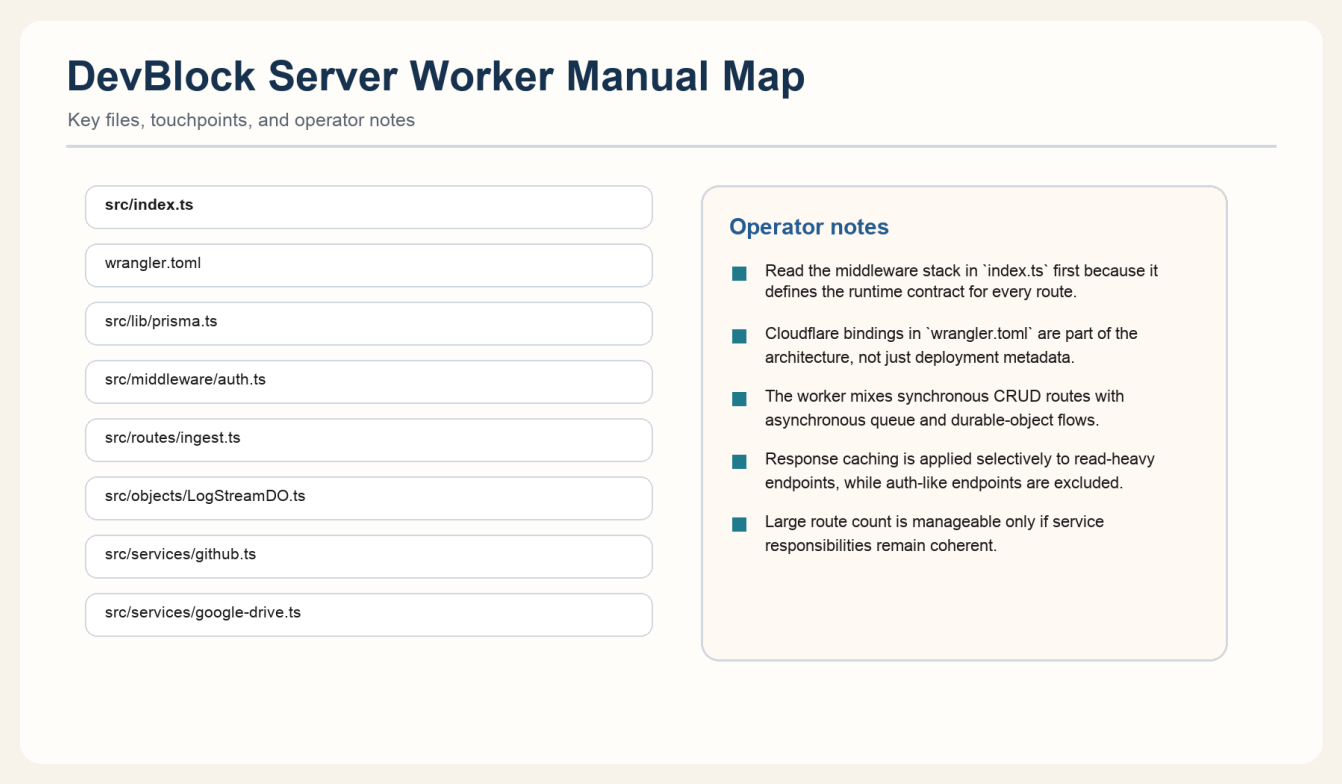
Area	Owns	Coordinates with	Failure mode
Product role	This project owns the control-plane backend and most external integrations.	Its entrypoint defines a runtime contract for every request before route code even starts.	This worker has a broad responsibility surface, so cohesion can erode over time.
Main inputs	Edge entrypoint: middleware checks body size, headers, response size, rate limits, CORS,	Database setup: Prisma is attached to request context and reset defensively on initialization failure	Integration routes are sensitive to secret drift, environment mismatch, and provider API changes.
Change surface	Start at `src/index.ts` for any runtime, header, or binding question before jumping into	Map the change to a route family first, then inspect its service module and any shared lib helpers.	In-memory rate limiting is per-worker-instance and should be treated as best-effort rather than
Operator posture	Validate backend changes with route-aware smoke testing rather than TypeScript alone.	Any binding or wrangler change should be treated as architecture-sensitive and	Read the middleware stack in `index.ts` first because it defines the runtime contract for every

DevBlock Server Worker Manual ownership matrix

Key files to read first

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

- src/index.ts
- wrangler.toml
- src/lib/prisma.ts
- src/middleware/auth.ts
- src/routes/ingest.ts
- src/objects/LogStreamDO.ts
- src/services/github.ts
- src/services/google-drive.ts



DevBlock Server Worker Manual orientation map

Change workflow

- 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.
- Treat ingest, chat, and durable-object behavior as async systems even when entrypoints look synchronous.

Operator notes

- Read the middleware stack in `index.ts` first because it defines the runtime contract for every route.
- Cloudflare bindings in `wrangler.toml` are part of the architecture, not just deployment metadata.
- The worker mixes synchronous CRUD routes with asynchronous queue and durable-object flows.
- Response caching is applied selectively to read-heavy endpoints, while auth-like endpoints are excluded.
- Large route count is manageable only if service responsibilities remain coherent.

Validation posture

- 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.

Current risks or watchpoints

- 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.

Glossary

Term	Meaning
Binding	A Cloudflare-provided runtime resource such as D1, R2, queues, or durable objects.
Route family	A grouped set of related HTTP handlers mounted under a common prefix.
Durable object	A Cloudflare stateful runtime primitive used here for log streaming or realtime coordination.