Watcher24

Case study by Naveen Raj — Full Stack Developer

Go | Python | Next.js | TypeScript | BetterAuth | PostgreSQL | ClickHouse | Redis | WebSocket | MinIO

All six services (gateway, analytics worker, realtime, notifier, console, IAM) run locally via Docker Compose with per-service dev commands, backed by JavaScript and Python SDKs and full test suites for each service.

Problem

Teams that want audit logging and observability for their own product usually end up gluing together an ingestion API, a queue, a time-series store, a realtime layer, and an auth/IAM system from scratch, per project.

Approach

Same process as FileNest — I'm the brain behind it: I designed the system before any code was written — the six-service split (a Go ingestion gateway, a Python analytics worker, a Go WebSocket realtime fan-out service, a Go notifier, a Next.js console, and a separate Next.js + better-auth IAM), the Redis Streams ’ ClickHouse event pipeline, and the pub/sub-driven realtime fan-out. The implementation itself was AI-assisted — "vibe coded": I gave the architecture, data flow, and specs as input, and an AI coding agent built the Go services, the Python worker, the Next.js console and IAM, and the JS/Python SDKs against that design.

Result

All six services (gateway, analytics worker, realtime, notifier, console, IAM) run locally via Docker Compose with per-service dev commands, backed by JavaScript and Python SDKs and full test suites for each service.