Design Decision

This project follows production-grade architectural choices to ensure scalability, performance, security, and maintainability.

Database: PostgreSQL with Timescale DB

Why Timescale DB?

Timescale DB is purpose-built for time-series workloads, making it ideal for high-volume API traffic logs.

- Time-series optimization
  Automatic partitioning using hyper tables for efficient time-based queries.
- SQL compatibility
  Fully compatible with PostgreSQL — no new query language required.
- Compression
  Built-in compression reduces storage usage by up to 90%.
- High performance
  Optimized for both high write throughput (log ingestion) and fast analytics queries.
- Production maturity
  Battle-tested with strong community support and documentation.

Schema Design

- Hypertable partitioned on the timestamp column
- Composite indexes:
  - (timestamp, status_code)
  - (timestamp, service_name)
- Additional indexes for efficient filtering of:
  - 2xx, 4xx, and 5xx HTTP status code ranges

Backend: Node.js (Express) with TypeScript

Why This Stack?

The backend is designed for high concurrency, type safety, and data integrity.

- High throughput
  Node.js event-driven architecture handles concurrent API requests efficiently.
- Type safety
  TypeScript reduces runtime errors and improves long-term maintainability.
- Input validation
  Zod schemas enforce strict validation at API boundaries.
- Connection pooling
  pg-pool efficiently manages PostgreSQL connections under load.

Containerization: Docker Compose

Architecture Decisions

Docker Compose is used to orchestrate all services in a consistent, reproducible environment.

- Multi-stage builds
  Produces optimized and lightweight production images.
- Named volumes
  Ensures database data persists across container restarts.
- Health checks
  Services wait for dependencies before starting.
- Network isolation
  Internal service communication is isolated; only Nginx is exposed externally.

Security Implementation

Feature	Description
API Authentication	API key validation via x-api-key header (bcrypt-hashed)
Rate Limiting	express-rate-limit + Nginx rate limiting
Input Validation	Zod schemas for all request payloads
SQL Injection Protection	Parameterized SQL queries
CORS	Restricted to configured origins
Request Size Limits	1MB (backend), 2MB (Nginx)
Security Headers	X-Frame-Options, X-Content-Type-Options, X-XSS-Protection

---

Performance Optimizations

Optimization	Details
Database Indexing	Composite indexes on frequently queried columns
Connection Pooling	Maximum 20 connections, 30s idle timeout
Batch Ingestion	Single INSERT for up to 1000 log entries
Query Optimization	Time-based aggregations using time_bucket()
Static Asset Caching	Enabled via Nginx
Gzip Compression	Enabled for all text-based responses