

Bidion MVP System Design Specification

Theory-First Internal Draft & Design Rationale

Jason Ramirez

Draft v0.1 — September 22, 2025

Intended audience: Bidion engineering (backend, frontend, DevOps), product management, and founding leadership.

See §2 for the disclaimer.

Contents

| | | |
|----------|--|-----------|
| 1 | Definitions | 8 |
| 2 | Introduction | 9 |
| 3 | Project Objective & Assumptions | 10 |
| 4 | MVP Scope | 13 |
| 4.1 | In Scope (MVP Deliverables) | 13 |
| 4.2 | Out of Scope (Deferred / Post-MVP Backlog) | 13 |
| 5 | High-Level Architecture | 13 |
| 6 | Data Model (v1) | 14 |
| 6.1 | Table Specifications | 14 |
| 6.2 | Database Schema & Indexes (Supabase Postgres + <code>pgvector</code>) | 14 |
| 6.2.1 | Row-Level Security (PostgreSQL) | 19 |
| 7 | Platform & Runtime | 21 |
| 7.1 | Environment & Configuration | 21 |
| 7.2 | Docker Compose (Redis) | 21 |
| 7.3 | Database Connection & Transactions (<code>lib/db/index.ts</code>) | 22 |
| 8 | Shared Libraries | 23 |
| 8.1 | Text Utilities: Normalization & Chunking (<code>lib/shared/text.ts</code>) | 23 |
| 9 | Matching Engine (v1) | 24 |
| 9.1 | Retrieval pipeline: normalization \rightarrow embeddings \rightarrow vector search | 24 |
| 9.1.1 | Pipeline overview. | 24 |
| 9.1.2 | Implementation Considerations. | 24 |
| 9.2 | Scoring Model | 24 |
| 9.2.1 | Components | 24 |
| 9.2.1.1 | Normalization of component scores to the unit interval $[0, 1]$ | 24 |
| 9.2.1.2 | Conceptual Interpretation of the Scoring Components | 25 |
| 9.2.2 | <i>Semantic Score</i> $S_{\text{sem}} \in [0, 1]$ | 25 |
| 9.2.2.1 | Definition of <i>Semantic Score</i> | 25 |
| 9.2.2.2 | Vector Search (semantic similarity) | 26 |
| 9.2.2.3 | Semantic Score Algorithm | 26 |
| 9.2.2.4 | Scope & Delimitations | 27 |
| 9.2.2.5 | Example | 27 |
| 9.2.2.6 | Tuning | 27 |

| | | |
|---------|--|----|
| 9.2.3 | <i>Keyword score</i> $S_{\text{kw}} \in [0, 1]$ | 28 |
| 9.2.3.1 | Definition of <i>Keyword Score</i> | 28 |
| 9.2.3.2 | Keyword Score Algorithm | 28 |
| 9.2.3.3 | Scope & Delimitations | 29 |
| 9.2.3.4 | Example | 29 |
| 9.2.3.5 | Tuning | 29 |
| 9.2.4 | <i>Rules</i> $S_{\text{rule}} \in [0, 1]$ | 29 |
| 9.2.4.1 | Definition of <i>Rules</i> | 29 |
| 9.2.4.2 | Rule Score Algorithm | 30 |
| 9.2.4.3 | Scope & Delimitations | 31 |
| 9.2.4.4 | Example | 31 |
| 9.2.4.5 | Tuning | 31 |
| 9.2.5 | Final Score | 31 |
| 9.2.6 | Example Calculation of $S_{\text{sem}}, S_{\text{kw}}, S_{\text{rule}}$ and S_{final} | 31 |
| 9.2.6.1 | RFP extract (single chunk). | 31 |
| 9.2.6.2 | Offering A (title + description). | 31 |
| 9.2.6.3 | Offering B (title + description). | 32 |
| 9.2.6.4 | Ranking. | 32 |
| 9.3 | Worker Task: Score Matches (<code>apps/worker/src/scoreMatches.task.ts</code>) | 32 |
| 9.3.1 | Design process of the worker | 32 |

10 Background Jobs & Worker Chain 35

| | | |
|----------|---|----|
| 10.1 | Worker Sequence | 35 |
| 10.2 | Job Infrastructure: Queues & Types (<code>apps/worker/src/queue.ts</code>) | 35 |
| 10.3 | OpenAI Embeddings Adapter (<code>apps/worker/src/embeddings.ts</code>) | 36 |
| 10.4 | Worker Task: Parse Documents (<code>apps/worker/src/parseDoc.task.ts</code>) | 37 |
| 10.5 | Worker Task: Embed Chunks (<code>apps/worker/src/embedChunks.task.ts</code>) | 38 |
| 10.6 | Worker Entrypoint: Concurrency & Startup (<code>apps/worker/src/main.ts</code>) | 39 |
| 10.7 | Operational Resilience | 40 |
| 10.7.1 | Delivery semantics & idempotency | 40 |
| 10.7.1.1 | Reference enqueue + idempotent handler | 40 |
| 10.7.1.2 | Worker wiring (graceful drain compatible) | 41 |
| 10.7.2 | Retries/backoff & budgets | 41 |
| 10.7.2.1 | Retry/backoff (Worker options) + DLQ forwarding | 41 |
| 10.7.2.2 | Per-org inflight and LLM-QPS budgets (Redis gates) | 42 |
| 10.7.2.3 | Using budgets in a worker | 43 |
| 10.7.3 | Kill switches | 43 |
| 10.7.3.1 | Global pause/resume endpoints (API) | 43 |
| 10.7.3.2 | Per-tenant pause (worker-side guard) | 44 |
| 10.7.3.3 | Simple circuit breaker for LLM calls (60s open) | 44 |

| | | |
|-----------|--|-----------|
| 10.7.4 | Worker SLOs & autoscale triggers | 45 |
| 10.7.4.1 | Emit SLO metrics (Prometheus) and backlog gauges | 45 |
| 10.7.4.2 | Autoscale decision (same thresholds as tests) | 46 |
| 10.7.4.3 | KEDA ScaledObject (illustrative) | 46 |
| 11 | API Design | 47 |
| 11.1 | API Server Bootstrap (<code>apps/api/src/server.ts</code>) | 47 |
| 11.2 | RFP Endpoints | 47 |
| 11.2.1 | POST <code>/rfps</code> — Create Manual-Text RFP | 47 |
| 11.2.2 | POST <code>/rfps/upload</code> — Register File-Backed RFP | 49 |
| 11.2.3 | POST <code>/rfps/:rfpId/score</code> — Force Re-Score | 51 |
| 11.2.4 | GET <code>/rfps</code> — List RFPs | 51 |
| 11.2.5 | GET <code>/rfps/:rfpId/matches</code> — Fetch Ranked Matches | 53 |
| 11.2.6 | POST <code>/rfps/:rfpId/match</code> — Recompute Matches Now | 54 |
| 11.3 | Offering Endpoints | 56 |
| 11.3.1 | POST <code>/offerings</code> — Create Offering and Embed, Manual-Text Only | 56 |
| 11.3.2 | POST <code>/offerings/upload</code> — Register File-Backed Offering | 59 |
| 11.4 | KB Endpoints | 62 |
| 11.4.1 | POST <code>/kb/upload</code> — Register File-Backed KB | 62 |
| 11.4.2 | GET <code>/kb</code> — List KB Documents | 63 |
| 11.5 | Assistant Endpoints | 64 |
| 11.5.1 | POST <code>/assistant/draft</code> — RAG Draft | 64 |
| 11.6 | API-Worker Bridge (<code>apps/api/worker-bridge.ts</code>) | 65 |
| 12 | Assistant (RAG) for Drafting Responses | 65 |
| 12.1 | Service Implementation (<code>apps/api/src/svc.assistant.ts</code>) | 66 |
| 13 | Frontend Integration (Next.js) | 68 |
| 13.1 | RFP Creation Page (<code>apps/web/app/rfps/new/page.tsx</code>) | 68 |
| 13.2 | Assistant Page (<code>apps/web/app/assistant/page.tsx</code>) | 70 |
| 14 | Testing & Quality Assurance | 70 |
| 14.1 | Tooling Options | 71 |
| 14.1.1 | Next.js (Frontend) | 71 |
| 14.1.1.1 | Unit & Component Tests. | 71 |
| 14.1.1.2 | E2E & Browser Automation. | 71 |
| 14.1.1.3 | Visual & Accessibility. | 72 |
| 14.1.1.4 | Static Analysis & Type Safety. | 72 |
| 14.1.2 | NestJS/Node (API & Workers) | 72 |
| 14.1.2.1 | Unit & Integration. | 72 |
| 14.1.2.2 | Contract & API Schema. | 72 |

| | | |
|----------|---|-----|
| 14.1.2.3 | Load & Resilience. | 72 |
| 14.1.2.4 | Security & Policy. | 72 |
| 14.1.2.5 | Summary. | 73 |
| 14.2 | Test NextJS | 73 |
| 14.2.1 | Illustrative Login Implementation (for tests to target) | 73 |
| 14.2.2 | Unit & Component Tests. | 74 |
| 14.2.2.1 | Vitest + React Testing Library | 74 |
| 14.2.2.2 | MSW (Mock Service Worker) | 76 |
| 14.2.3 | E2E & Browser Automation. | 77 |
| 14.2.3.1 | Playwright (E2E) | 77 |
| 14.2.3.2 | Cypress (Alternative E2E) | 78 |
| 14.2.4 | Visual & Accessibility. | 78 |
| 14.2.4.1 | Storybook (Visual Catalog) | 78 |
| 14.2.4.2 | Accessibility | 79 |
| 14.2.4.3 | Lighthouse CI (Perf/A11y Budgets) | 79 |
| 14.2.5 | Static Analysis & Type Safety | 80 |
| 14.2.5.1 | TypeScript strict. | 80 |
| 14.2.5.2 | ESLint (Next.js base). | 80 |
| 14.2.5.3 | Dead code scan (knip). | 80 |
| 14.2.5.4 | Import rules and cycles (dependency-cruiser). | 80 |
| 14.3 | Test NestJS/Node (API & Workers) | 81 |
| 14.3.1 | Unit & Integration. | 82 |
| 14.3.1.1 | Shared Test Infrastructure | 82 |
| 14.3.1.2 | API Tests | 84 |
| 14.3.1.3 | Worker Tests | 85 |
| 14.3.2 | Contract & API Schema. | 87 |
| 14.3.2.1 | OpenAPI Schema Checks (jest-openapi). | 87 |
| 14.3.2.2 | Pact (consumer-driven contract) example. | 88 |
| 14.3.3 | Load & Resilience. | 89 |
| 14.3.3.1 | API Stress & Load Testing Plan (k6 SLO thresholds). | 89 |
| 14.3.3.2 | Worker(BullMQ): Throughput Probe + Idempotency Tests | 92 |
| 14.3.4 | Security & Policy. | 100 |
| 14.3.4.1 | DTO invariants with class-validator (unit). | 100 |
| 14.3.4.2 | eslint-plugin-security and audits (CI snippets). | 100 |
| 14.3.5 | Worker Tasks (Parse → Embed → Score) — Example Tests. | 101 |
| 14.3.6 | API-Worker Bridge (enqueue from API). | 102 |
| 14.3.7 | Monitoring Test Results & Regression Handling | 103 |

| | |
|---|------------|
| 15 Infrastructure & Operations Design | 104 |
| 15.1 Deployment Targets & Environments | 104 |
| 15.1.1 Environment Matrix | 105 |
| 15.1.2 Cloud-Native Architecture | 105 |
| 15.1.3 Network & Access Topology | 106 |
| 15.1.4 Secrets & Config | 106 |
| 15.1.5 CI/CD Pipeline (GitHub Actions) | 106 |
| 15.1.5.1 Workflows & triggers. | 106 |
| 15.1.5.2 Core Phases (per workflow). | 107 |
| 15.1.5.3 Environments & approvals. | 110 |
| 15.1.5.4 Artifact promotion (immutability). | 110 |
| 15.1.5.5 Database migrations (Supabase Postgres). | 111 |
| 15.1.5.6 Secrets & configuration. | 111 |
| 15.1.5.7 Concurrency & caching. | 112 |
| 15.1.5.8 Branch protections & required checks. | 112 |
| 15.1.5.9 Rollback & recovery. | 113 |
| 15.1.5.10 Frontend (NextJS) GitHub Actions YAML sketch. | 114 |
| 15.1.5.11 Backend (NestJS/Node) GitHub Actions YAML sketch. | 116 |
| 15.1.5.12 Rationale. | 120 |
| 15.1.6 Scaling & Sizing (Initial) | 120 |
| 15.1.6.1 Frontend (Next.js on Vercel) | 120 |
| 15.1.6.2 API (NestJS on Fly/Render) | 121 |
| 15.1.6.3 Workers (BullMQ on Fly/Render) | 121 |
| 15.1.6.4 DB (PostgreSQL — Supabase Postgres + <code>pgvector</code>) | 122 |
| 15.1.6.5 Postgres storage/IOPS: | 122 |
| 15.1.6.6 Redis (Managed — Upstash/Fly/Render) | 123 |
| 15.1.6.7 Object Storage (Supabase Storage, S3-compatible) | 123 |
| 15.1.6.8 LLM (OpenAI API) | 124 |
| 15.1.7 Observability & Ops | 124 |
| 15.1.7.1 Tooling options. | 124 |
| 15.1.7.2 Logging (Loki stack) | 125 |
| 15.1.7.3 Metrics (Prometheus) | 131 |
| 15.1.7.4 Tracing (OpenTelemetry) | 139 |
| 15.1.7.5 Alerting / On-call | 144 |
| 15.1.8 Upgrade Paths (Post-MVP) | 151 |
| 15.2 Scaling Plan (MVP → v1) | 151 |
| 15.2.1 Horizontally scale stateless tiers first. | 152 |
| 15.2.2 Postgres scaling path (Supabase DB). | 153 |
| 15.2.3 Vector search (<code>pgvector</code>) scale knobs. | 153 |

| | | |
|-----------|--|------------|
| 15.2.4 | Queues, backpressure, and cost controls. | 154 |
| 15.2.5 | Caching layers. | 154 |
| 15.2.6 | Multi-region (v1+). | 155 |
| 15.2.7 | Migrations and deploy safety. | 155 |
| 15.2.8 | Observability & SLO enforcement. | 155 |
| 15.2.9 | Data lifecycle. | 156 |
| 15.3 | Maintenance, Resilience & Recovery | 156 |
| 15.3.1 | Postgres: Backups, PITR, and Maintenance | 156 |
| 15.3.1.1 | Setup & Test PITR (Managed Supabase). | 157 |
| 15.3.2 | Postgres: Scheduling, Vacuum/Analyze, and Tuning | 157 |
| 15.3.2.1 | Pre-requisites (Supabase-managed Postgres) | 158 |
| 15.3.2.2 | Pre-requisites (Self-hosted Postgres: install & preload pg_cron) | 158 |
| 15.3.2.3 | Create jobs (nightly ANALYZE, weekly VACUUM) | 159 |
| 15.3.2.4 | Force-run now (staging/CI validation) | 159 |
| 15.3.2.5 | Test: Observability queries (did it help?) | 159 |
| 15.3.2.6 | Autovacuum tuning (targeted) | 160 |
| 15.3.2.7 | Alternative if pg_cron is unavailable (self-hosted) | 160 |
| 15.3.2.8 | Operational remarks (timeouts & safety) | 161 |
| 15.3.2.9 | Passed-Results Variables (Verification Matrices) | 161 |
| 15.3.3 | Redis: Durability & Recovery (BullMQ backing store) | 162 |
| 15.3.3.1 | Managed Redis (Redis Cloud by Redis) | 162 |
| 15.3.3.2 | Self-hosted on Render (Private Service) | 164 |
| 15.3.4 | Kill Switches & Circuit Breaking | 170 |
| 15.3.5 | SLO Guardrails & Alerting | 170 |
| 15.3.6 | Deploy Safety & Planned Disruption | 170 |
| 16 | Conclusion | 171 |
| 17 | Acknowledgements | 171 |

1 Definitions

Table 1: Definitions of Terms and Acronyms

| Term | Definition |
|------------------------------|--|
| API | Application Programming Interface; our backend HTTP surface that the frontend calls. |
| AuthN / AuthZ | Authentication (verifying identity) and Authorization (verifying permissions). |
| BullMQ (BullMQ) | A Node.js job/queue library backed by Redis for background processing (e.g., parse \rightarrow embed \rightarrow score chains). |
| CDN | Content Delivery Network; serves static assets (JS/CSS/images) close to users. |
| Chunk / Chunking | Splitting long documents into bounded text segments to embed and search efficiently. |
| Cosine Similarity | Measure of similarity between vectors using the cosine of the angle between them; used to compare embeddings. |
| Embedding | Dense vector representation of text produced by an ML model; enables semantic similarity search. |
| HNSW / IVF / IVF-FLAT | Approximate nearest-neighbor index types used by <code>pgvector</code> (<i>Hierarchical Navigable Small World / Inverted File</i> ; IVFFLAT is a specific IVF variant). |
| JWT | JSON Web Token; compact token used for authenticated API requests. |
| KB | Knowledge Base; internal documents (product sheets, past proposals) used for grounding responses. |
| K8s | Kubernetes; container orchestration platform. |
| KNN | k -Nearest Neighbors; returns the k most similar items (e.g., vectors) to a query. |
| LLM | Large Language Model (e.g., OpenAI GPT-class models). |
| MVP | Minimum Viable Product; smallest feature set that delivers user value for validation. |
| NestJS / Express | Node.js web frameworks used to implement the API layer. |
| Next.js | React framework used for the frontend (App Router). |
| OCR | Optical Character Recognition; extracting text from scanned PDFs/images (deferred for MVP). |
| OpenAI API | OpenAI services used for embeddings and chat completions. |
| Offering | A catalog entry describing the business capability/product being matched against RFPs. |

| Term | Definition |
|------------------------------|--|
| pgvector (pgvector) | PostgreSQL extension adding a vector type and similarity operators for embeddings. |
| PII | Personally Identifiable Information; sensitive data requiring careful handling. |
| PostgreSQL (Supabase) | Managed relational database used for metadata and pgvector vector search. |
| RAG | Retrieval-Augmented Generation; generate text using LLMs grounded by retrieved context chunks. |
| RFP | Request for Proposal; procurement document with requirements to which offerings are matched. |
| Redis | In-memory data store; used here as the queue backend for BullMQ. |
| S3 / Supabase Storage | Object storage for uploaded files; S3-compatible API provided by Supabase Storage. |
| Semantic Search | Similarity search over embeddings to find conceptually related text, not just lexical matches. |
| Supabase | Managed platform providing Postgres (with pgvector), Auth, and Storage used in this MVP. |
| Vector Search | Querying a vector index (e.g., pgvector) to find nearest embeddings by cosine (or other) similarity. |

2 Introduction

Purpose. This document is an internal, *theory-first* design plan for the Bidion MVP. It consolidates architectural intent, key concepts, and proposed strategies/procedures to guide the initial implementation and facilitate structured review. It is not a production runbook.

Status / Disclaimer. The contents are non-final and subject to change as we prototype, test, and receive pilot feedback. Configuration values (for example, pool sizes, SLOs, and index settings) are illustrative targets to be validated under load; they do not constitute commitments to specific vendors, capacities, or dates.

Scope. This draft covers the core system shape and interfaces: Next.js frontend, NestJS API, BullMQ workers, and PostgreSQL+**pgvector** with RLS; plus ingestion, matching, retrieval, and observability touchpoints. Detailed production runbooks, security audits, and multi-region data strategies are *out of scope* for this revision.

Audience. Bidion engineering and adjacent stakeholders (product, infrastructure). The goal is to provide sufficient specificity to start implementation spikes, create RFCs, and make early

trade-offs—while avoiding premature lock-in. (Treat statements herein as informational and evolving rather than binding roadmaps.)[?]

How to use this doc.

- Treat numbers as *initial targets*; refine via load tests (k6), prototypes, and early pilot feedback.
- Use sections marked “MVP-first” to prioritize; items labeled v1+ are intentionally deferred.
- Log deviations (trade-offs, vendor changes) as short RFCs; update this spec incrementally.

Validation plan (high level). Prove the design through: (1) unit/integration tests, (2) short k6 PR runs plus nightly soaks, (3) pilot-org trials with telemetry, and (4) cost/throughput reviews. Findings will update thresholds, capacities, and feature flags before GA.

Change policy. This document is versioned; assumptions, limits, and diagrams may change as we learn. When implementation diverges materially, update this doc or attach a brief RFC link.

3 Project Objective & Assumptions

Goal (MVP). Let a business ingest its offerings/capabilities, ingest RFPs from public/commercial sources, and match them with a clear score & rationale. Provide a dashboard to review matches, collaborate, and draft responses using OpenAI grounded on an internal knowledge base.

Delivery & Operational

- Single tenant at first; design for multi-tenant later.
- Team size 2–4 devs; MVP timeline ≥ 6 months (alpha ~ 3 months, beta ~ 4 –5 months, GA ≥ 6 months).
- Public RFP sources via simple scrapers/APIs (phase 1: CSV upload/manual).
- Internal KB = PDFs/Docs uploaded by users.
- Budget-conscious; prefer managed services or light VM/K8s.

Workload Assumptions

General

- Traffic is *not* evenly distributed across time or tenants (orgs). Weekdays 9–6 local time are peak; a few heavy users dominate load.
- Creating or uploading an RFP should be fast ($P95 \leq 500$ ms for metadata create; streaming upload for files).
- Recomputing matches is near real-time for typical docs; queued for very large PDFs or bulk backfills.
- **Active tenants/users (MVP pilot):** 50–200 orgs; 10k–25k monthly active users (MAU).
- **Content volume (monthly, MVP):**
 - 6,000–10,000 new RFPs (200–330/day; end-of-quarter spikes $\times 2$ –3).

- 40,000–80,000 offering updates (catalog edits, new offerings).
- 120,000–250,000 KB document ingests/updates (policies, past proposals, appendices).

Timeline / Reads vs Writes

- Viewing RFPs, matches, and dashboards should be fast (cacheable reads; $P95 \leq 500$ ms).
- The system is **more read-heavy than write-heavy**: reads (match lists, offering lookups, KB previews) dominate UI traffic.
- Ingest paths (RFP upload \rightarrow parse \rightarrow embed) are **write-heavy bursts** and run via workers with backpressure.

Search & Retrieval (RAG)

- Searching should be fast (vector KNN $P95 \leq 200$ ms under nominal load).
- Workload is read-heavy (assistant questions, filters, keyword+vector hybrid).
- Typical assistant flow: shortlist $m=200$ via vector search \rightarrow re-rank with domain scoring \rightarrow return top $k=50$.

Back-of-the-Envelope Usage (MVP)

- **Monthly reads (API GETs)**: *25–45 million/month* (includes CDN edge hits).
Reasoning: $10k\text{--}25k \text{ MAU} \times \sim 40\text{--}60 \text{ reads/day} \times 30 \text{ days} \approx 12M\text{--}45M$.
- **Monthly writes (API POST/PUT/PATCH)**: *1–3 million* (UI edits, comments, status changes).
- **Monthly assistant queries (vector searches)**: *0.5–2 million*.
- **Monthly match recomputes**: $6k\text{--}10k \text{ RFPs} \times (\text{initial compute} + \sim 2 \text{ updates}) \approx 18k\text{--}30k$ recomputes.

Size per Entity (storage planning)

- **RFP (median)**: 40 pages, $\sim 12k$ words $\Rightarrow \sim 50$ chunks (avg 240 words/chunk).
- **Embedding vector (3072-d float32)**: $3072 \times 4 \text{ bytes} = \mathbf{12 \text{ KB}}$ per vector.
- **Embeddings per RFP**: $50 \text{ chunks} \times 12 \text{ KB} \approx \mathbf{600 \text{ KB}}$ (vectors only).
- **Text/PDF storage (RFP)**: median **2–5 MB** per file (depends on scans/figures).
- **Offering record**: $\sim 2 \text{ KB}$ metadata; embedding $\sim 12 \text{ KB}$.
- **KB doc (median)**: 10 pages $\Rightarrow \sim 12$ chunks $\Rightarrow \sim 144 \text{ KB}$ vectors.

Monthly Storage Growth (vectors only, MVP)

- RFP vectors: 6k–10k RFPs \times 600 KB \approx **3.6–6 GB/month**.
- KB vectors: 120k–250k docs \times 144 KB \approx **17–36 GB/month**.
- Offering vectors (updates, net new): 40k–80k \times 12 KB \approx **0.5–1 GB/month**.
- **Total vectors/month \approx 21–43 GB \Rightarrow 0.25–0.5 TB/year.**

Throughput (from monthly to per-second, MVP)

Use the same conversion heuristic; note that bursts can be 5–10 \times the averages and are absorbed via CDN and queues.

- **Read QPS (avg):** *10–20 requests/s* (25–45M reads/month). *Peak:* 5–10 \times .
- **Assistant search QPS (avg):** *0.2–0.8 requests/s* (0.5–2M/month). *Peak:* 5 \times .
- **Writes QPS (avg):** *0.5–1.5 requests/s* (1–3M/month). *Peak:* 5 \times .
- **Match recompute rate:** *12–20/min* at steady state; bursty and handled by workers.

Handy Conversion

- \approx 2.5 million seconds/month.
- 1 request/second \approx 2.5 million requests/month.
- 40 requests/second \approx 100 million requests/month.
- 400 requests/second \approx 1 billion requests/month.

Performance-Critical Operations (Bid/Proposal Context)

- **RFP create/upload:** immediate user feedback; background parsing/embedding starts within < 5 s.
- **Matches fetch:** cached and paginated; $P95 \leq 500$ ms.
- **Assistant ask:** shortlist + re-rank $\Rightarrow P95$ KNN ≤ 200 ms, end-to-end answer target < 2 s.
- **Bulk ops:** throttled and queued (embed, re-score), with progress UI.
- **Viewing RFP matches:** $p95 \leq 500$ ms for GET /rfps/:id/matches (precomputed).
- **Assistant draft (first token):** $TTFT \leq 1.5$ s, streaming thereafter.
- **Full-text search across RFPs/KB:** $p95 \leq 300$ ms for query ≤ 3 terms.
- **Attachment preview (PDF/doc first page):** $p95 \leq 700$ ms with CDN caching.
- **RFP create/update (form submit):** server time ≤ 250 ms; heavy work async via queue.
- **Enqueue parse/embed jobs:** ≤ 150 ms API acknowledgement; visible queue status in ≤ 1 s.
- **Parse \rightarrow matches availability:** $P50 \leq 30$ s, $P95 \leq 90$ s for 10–20 page PDFs.
- **Offerings lookup for mapping to RFPs:** $p95 \leq 400$ ms.

- **Dashboard load (recent RFPs, tasks):** *Largest Contentful Paint* $\leq 2.5\text{ s}$; API backing calls *p95* $\leq 400\text{ ms}$.
- **Export (CSV/Docx) kickoff:** $\leq 200\text{ ms}$ to acknowledge; download ready notification *P95* $\leq 60\text{ s}$.
- **Auth/session resume:** $\leq 150\text{ ms}$ token refresh; cold login $\leq 600\text{ ms}$ server time.

4 MVP Scope

4.1 In Scope (MVP Deliverables)

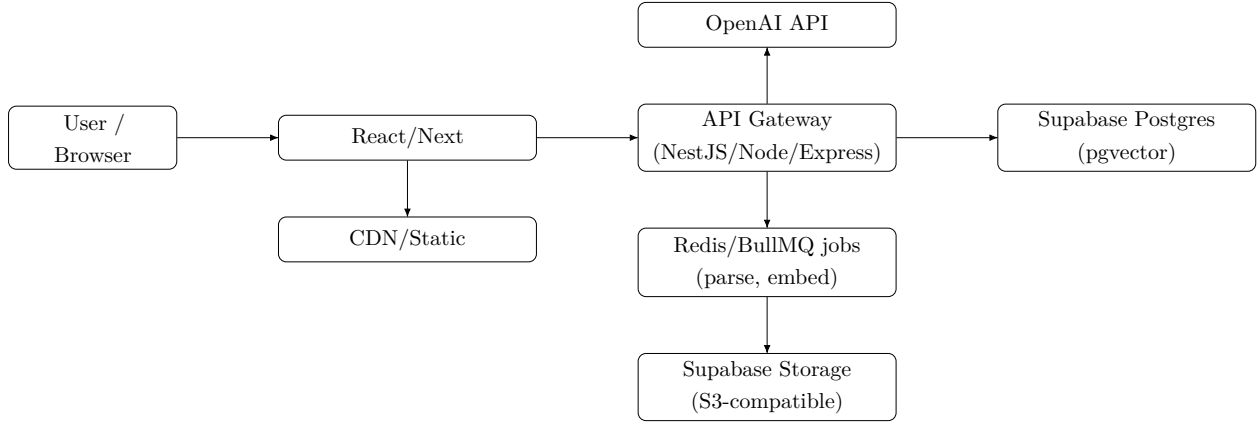
- AuthN/AuthZ, org/user accounts
- Data ingestion: Offerings (forms + CSV), RFPs (URL/PDF/manual), KB uploads
- Matching engine v1: normalization \rightarrow embeddings \rightarrow vector search; hybrid score
- Match review UI: score, snippet highlights, rationale
- Draft response assistant (RAG over KB + RFP)
- Dashboard & basic workflow statuses
- Background jobs (parsing, embeddings, scoring)
- Basic notifications (email/Slack webhook)

4.2 Out of Scope (Deferred / Post-MVP Backlog)

- Fine-grained RBAC + billing
- Advanced procurement integrations; redlining
- Learning-to-rank/ML re-ranker

5 High-Level Architecture

- **Frontend:** Next.js (App Router).
- **Backend:** Node.js (Express/Nest-style).
- **DB:** Supabase PostgreSQL + pgvector.
- **Storage:** S3/Supabase Storage.
- **Jobs:** Redis + BullMQ.
- **Auth:** Hosted (e.g., Supabase/Auth0/Clerk).
- **LLM:** OpenAI embeddings (text-embedding-3-large), GPT-4 class for RAG.



Data flow:

1. Ingest RFPs/offerings/KB \rightarrow *parse* \rightarrow *embed* \rightarrow *score*.
2. Store embeddings in **pgvector**; raw files in S3; metadata in Postgres.
3. UI queries API for matches and drafts; assistant uses RAG over top chunks.

6 Data Model (v1)

6.1 Table Specifications

Table 2: Table: **orgs**

| Column | Type | Key | Notes |
|------------|------------|-----|-----------------------------------|
| id | uuid | PK | gen_random_uuid() |
| name | text | | |
| created_at | timestampz | | default now() (if present) |

Table 3: Table: **users**

| Column | Type | Key | Notes |
|------------|------------|-----|------------------------------|
| id | uuid | PK | gen_random_uuid() |
| email | text | UQ | unique, lowercased |
| name | text | | |
| role | text | | enum: admin member |
| org_id | uuid | FK | \rightarrow orgs.id |
| created_at | timestampz | | default now() |

6.2 Database Schema & Indexes (Supabase Postgres + **pgvector**)

Listing 1: Core tables, vector storage, and indexes

Table 4: Table: documents

| Column | Type | Key | Notes |
|-------------|------------|-----|-------------------------------------|
| id | uuid | PK | gen_random_uuid() |
| org_id | uuid | FK | → orgs.id |
| kind | text | | rfp kb offering (CHECK) |
| ref_id | uuid | | optional link (e.g., offering id) |
| filename | text | | |
| storage_url | text | | Supabase Storage signed URL or path |
| mime | text | | e.g., application/pdf |
| created_at | timestampz | | default now() |
| status | text | | optional: parsed embedded scored |

Table 5: Table: chunks

| Column | Type | Key | Notes |
|-------------|------|-----|-----------------------------------|
| id | uuid | PK | gen_random_uuid() |
| document_id | uuid | FK | → documents.id, ON DELETE CASCADE |
| idx | int | | chunk order |
| section | text | | optional (e.g., “Requirements”) |
| text | text | | normalized chunk content |

Table 6: Table: offerings

| Column | Type | Key | Notes |
|-------------|------------|-----|-----------------------|
| id | uuid | PK | gen_random_uuid() |
| org_id | uuid | FK | → orgs.id |
| title | text | | |
| description | text | | |
| tags | text[] | | GIN index recommended |
| created_at | timestampz | | default now() |
| updated_at | timestampz | | default now() |

```

1  -- Supabase: enable required extensions in our project
2  CREATE EXTENSION IF NOT EXISTS pgcrypto;
3  CREATE EXTENSION IF NOT EXISTS vector;
4
5  -- documents & chunks
6  CREATE TABLE IF NOT EXISTS documents (
7      id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
8      org_id      uuid NOT NULL,
9      kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),

```

Table 7: Table: `embeddings`

| Column | Type | Key | Notes |
|----------|--------------|-----|--|
| id | uuid | PK | <code>gen_random_uuid()</code> |
| org_id | uuid | FK | \rightarrow <code>orgs.id</code> |
| kind | text | | <code>rfp kb offering</code> (CHECK) |
| ref_id | uuid | | document or entity id |
| chunk_id | uuid | | nullable for offering-wide vector |
| vector | vector(3072) | | pgvector (cosine ops); IVF-FLAT/HNSW index |
| text | text | | source text for provenance |
| meta | jsonb | | optional |

Table 8: Table: `matches`

| Column | Type | Key | Notes |
|-------------|------------------|-----|--|
| id | uuid | PK | <code>gen_random_uuid()</code> |
| org_id | uuid | FK | \rightarrow <code>orgs.id</code> |
| rfp_id | uuid | FK | \rightarrow <code>documents.id</code> (kind='rfp') |
| offering_id | uuid | FK | \rightarrow <code>offerings.id</code> |
| score | double precision | | final hybrid score [0, 1] |
| reasons | jsonb | | snippets/keywords/penalties |
| created_at | timestampz | | default <code>now()</code> |

Table 9: Table: `score_config`

| Column | Type | Key | Notes |
|-----------|-------|-------|--|
| org_id | uuid | PK/FK | \rightarrow <code>orgs.id</code> (1:1) |
| boosts | jsonb | | <code>{"soc2":0.25,"24/7":0.15}</code> |
| required | jsonb | | <code>["soc2","24/7"]</code> |
| forbidden | jsonb | | <code>["on-prem only"]</code> |

Table 10: Table: `work_items`

| Column | Type | Key | Notes |
|------------------|------------|-----|--|
| id | uuid | PK | <code>gen_random_uuid()</code> |
| org_id | uuid | FK | \rightarrow <code>orgs.id</code> |
| rfp_id | uuid | FK | \rightarrow <code>documents.id</code> (kind='rfp') |
| title | text | | |
| status | text | | <code>new in_progress review done</code> |
| assignee_user_id | uuid | FK | \rightarrow <code>users.id</code> (nullable) |
| created_at | timestampz | | default <code>now()</code> |

| | | | |
|----|-----------------------|--------------------|---|
| 10 | <code>ref_id</code> | <code>uuid,</code> | <code>-- optional foreign link (e.g., offering id)</code> |
| 11 | <code>filename</code> | <code>text,</code> | |

Table 11: Table: audit_logs

| Column | Type | Key | Notes |
|---------------|------------|-----|-----------------------------------|
| id | uuid | PK | gen_random_uuid() |
| org_id | uuid | FK | → orgs.id |
| actor_user_id | uuid | FK | → users.id |
| action | text | | e.g., create_rfp, match_recompute |
| subject_kind | text | | document offering match ... |
| subject_id | uuid | | referenced entity id |
| meta | jsonb | | optional context |
| created_at | timestampz | | default now() |

```

12  storage_url text,
13  mime         text,
14  created_at   timestampz NOT NULL DEFAULT now()
15 );
16
17 CREATE TABLE IF NOT EXISTS chunks (
18   id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
19   document_id uuid NOT NULL REFERENCES documents(id) ON DELETE CASCADE,
20   idx         int  NOT NULL,          -- chunk order
21   section     text,                  -- optional 'Requirements', etc.
22   text        text NOT NULL
23 );
24
25 -- embeddings (pgvector)
26 CREATE TABLE IF NOT EXISTS embeddings (
27   id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
28   org_id      uuid NOT NULL,
29   kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),
30   ref_id      uuid NOT NULL,          -- document_id (or offering id if we embed inline)
31   chunk_id    uuid,                  -- null for offering-wide vectors if we store one
32   vector      vector(3072) NOT NULL,
33   text        text NOT NULL,
34   meta        jsonb DEFAULT '{}'::jsonb
35 );
36
37 -- matches
38 CREATE TABLE IF NOT EXISTS matches (
39   id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
40   org_id      uuid NOT NULL,
41   rfp_id      uuid NOT NULL,          -- documents.id where kind='rfp'
42   offering_id uuid NOT NULL,          -- documents.id where kind='offering'

```

```

43     score          double precision NOT NULL,
44     reasons        jsonb NOT NULL DEFAULT '{} '::jsonb,
45     created_at     timestampz NOT NULL DEFAULT now()
46 );
47
48 -- keyword/rules (simple MVP)
49 CREATE TABLE IF NOT EXISTS score_config (
50     org_id          uuid PRIMARY KEY,
51     boosts          jsonb NOT NULL DEFAULT '{} '::jsonb, -- {"soc2":0.25, "24/7":0.15}
52     required        jsonb NOT NULL DEFAULT '[] '::jsonb, -- ["soc2", "24/7"]
53     forbidden       jsonb NOT NULL DEFAULT '[] '::jsonb  -- ["on-prem only"]
54 );
55
56 -- offering catalog (you can also model as documents(kind='offering'))
57 CREATE TABLE IF NOT EXISTS offerings (
58     id              uuid PRIMARY KEY DEFAULT gen_random_uuid(),
59     org_id          uuid NOT NULL,
60     title           text NOT NULL,
61     description     text NOT NULL,
62     tags            text[] DEFAULT '{} ',
63     created_at      timestampz NOT NULL DEFAULT now(),
64     updated_at      timestampz NOT NULL DEFAULT now()
65 );
66
67 -- indexes
68 CREATE INDEX IF NOT EXISTS idx_chunks_doc ON chunks(document_id, idx);
69 CREATE INDEX IF NOT EXISTS idx_embeddings_kind_org ON embeddings(kind, org_id);
70 CREATE INDEX IF NOT EXISTS idx_embeddings_vector ON embeddings USING ivfflat (vector
    vector_cosine_ops) WITH (lists=100);
71 CREATE INDEX IF NOT EXISTS idx_matches_rfp ON matches(org_id, rfp_id, score DESC);
72
73
74 -- Row-level security (multitenancy)
75 ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
76 CREATE POLICY org_can_read_rfps ON documents
77     FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid AND kind = 'rfp');
78
79 ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
80 CREATE POLICY org_can_read_matches ON matches
81     FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);
82
83 ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
84 CREATE POLICY org_can_crud_offerings ON offerings
85     USING (org_id = current_setting('app.org_id', true)::uuid)

```

```
WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);
```

6.2.1 Row-Level Security (PostgreSQL)

Rationale.

Enabling Row-Level Security (RLS) with per-request/org scoping is a low-cost, high-impact guardrail for a multi-tenant MVP. The policies below anchor tenant isolation in the *database layer*, not just in API code.

- **True multi-tenancy isolation (defense in depth).** With `SET LOCAL app.org_id=...` derived from the caller's JWT, PostgreSQL itself enforces: a query cannot read or mutate rows outside its org. Even if an ORM filter or controller check is missed, the DB still blocks cross-tenant access.
- **Prevents silent data leaks.** A single missing `WHERE org_id=...` in a handler could expose other tenants' RFPs or matches. RLS makes such bugs fail *closed* instead of leaking.
- **Covers *all* paths (API & Workers).** Background workers (BullMQ) perform direct SQL. The same RLS policies apply when workers run with an explicit org scope, so batch jobs cannot cross-contaminate data.
- **Write integrity via `WITH CHECK`.** For offerings, the policy includes `WITH CHECK` so insert-s/updates cannot spoof another org's `org_id`. The DB rejects any row where `org_id` \neq current session org.
- **Principle of least privilege.** Each table exposes only the minimal actions per tenant:
 - **documents:** tenants can *read* only their own rows and only when `kind='rfp'` (prevents accidental exposure of non-RFP docs).
 - **matches:** tenants can *read* only their computed matches.
 - **offerings:** tenants can *CRUD* only their own offerings (both `USING` & `WITH CHECK` clauses).
- **Auditable and testable.** Policies are declarative SQL; integration tests run with RLS *enabled* and an org scope to catch regressions (e.g., missing policies, incorrect joins).
- **Operational safety for an MVP.** Early-stage code changes (new endpoints, ad-hoc SQL) are common and risky; RLS provides a safety net while velocity is high.
- **Works with `pgvector` and analytics.** Vector KNN queries still respect RLS. Any ANN/IVF-FLAT/HNSW index scans are filtered by the policy, so nearest-neighbor retrieval cannot cross org boundaries.
- **Simple mental model.** App code does not need to thread `org_id` through every repository method; set the org once per request/job and let the DB enforce the contract.

Threat examples prevented.

- *Missing filter*: `SELECT * FROM matches ORDER BY score DESC LIMIT 10`; would otherwise return another tenant's data; RLS prunes rows to the current org.
- *ID guessing*: A user guessing a valid `rfp_id` cannot read it if `org_id` mismatches; the row is invisible.
- *Write spoofing*: `INSERT INTO offerings (org_id, ...)` with a foreign org is rejected by `WITH CHECK`.

Fits request/worker model.

1. API middleware/auth sets: `SET LOCAL app.org_id = $jwt.orgId`; in the transaction.
2. All subsequent `SELECT/INSERT/UPDATE` run under that scope.
3. Workers receive `orgId` in the job payload and set the same `SET LOCAL` before touching Postgres.

Net effect.

These policies make cross-tenant access *impossible by default*, reduce the blast radius of coding mistakes, and align with compliance expectations for customer data separation—while adding near-zero friction to developer workflows.

Intuitively.

Actors: *AcmeGov* (public-sector vendor team) and *BetaBank* (financial vendor team) are separate tenants using Bidion.

1. **AcmeGov uploads an RFP.** The API creates a `documents` row with `org_id=AcmeGov` and `kind='rfp'`, then workers parse \rightarrow chunk \rightarrow embed. Later, the `score-matches` worker writes top- N rows into `matches` with `org_id=AcmeGov`.
2. **BetaBank curates offerings.** Their team creates several `offerings` linked to `org_id=BetaBank`. These embed immediately and will *only* ever be considered inside BetaBank's tenant scope.
3. **AcmeGov views matches.** The UI calls `GET /rfps/{id}/matches`. With **RLS enabled** and `SET LOCAL app.org_id='AcmeGov'` in the request transaction, Postgres returns only rows where `org_id=AcmeGov`. BetaBank's rows are invisible at the storage layer; even a missing controller filter cannot leak them.
4. **Background jobs stay isolated.** When the `score-matches` worker re-scores an AcmeGov RFP, it sets `SET LOCAL app.org_id='AcmeGov'` before reading `documents/chunks/embeddings` and *before* writing `matches`. If a developer accidentally broadens a SQL query (e.g., drops a `WHERE`), RLS still prunes rows to AcmeGov only.

5. **Attack/bug averted.** Suppose a BetaBank user guesses AcmeGov’s `rfp_id` and hits `/rfps/{id}/matches`. The row-level policy evaluates `org_id=current_setting('app.org_id')` and returns *no rows*. Likewise, if someone tries to `INSERT INTO offerings` with `org_id='AcmeGov'` from BetaBank’s session, the `WITH CHECK` clause rejects the write.

Result: Each tenant sees only their RFPs, offerings, and computed matches. RLS makes the safe behavior the default, even during fast-moving MVP development, protecting against one-line mistakes and ID-guessing attempts.

7 Platform & Runtime

7.1 Environment & Configuration

Listing 2: `.env` for Supabase Postgres, Redis, Storage, OpenAI

```
1 # Supabase Postgres (managed)
2 # Use the pooled URL (port 6543) for app traffic. Keep sslmode=require.
3 DATABASE_URL=postgresql://postgres:<password>@db.<project-ref>.supabase.co:6543/postgres?
   sslmode=require
4
5 # Redis
6 REDIS_HOST=localhost
7 REDIS_PORT=6379
8
9 # Storage (S3-compatible; MinIO in compose)
10 S3_ENDPOINT=http://localhost:9000
11 S3_BUCKET=bucket
12 S3_ACCESS_KEY=dev
13 S3_SECRET_KEY=devdevdev
14
15 # OpenAI
16 OPENAI_API_KEY=sk-...
17
18 # App
19 NODE_ENV=development
```

7.2 Docker Compose (Redis)

Listing 3: `docker-compose.yml` for Redis

```
1 version: "3.9"
2
3 networks:
4   backend:
```

```

5     driver: bridge
6     internal: true    # prevents external access
7
8 volumes:
9     redis_data:
10
11 services:
12     redis:
13         image: redis:7
14         command: >
15             redis-server
16             --appendonly yes
17             --requirepass ${REDIS_PASSWORD}
18             --maxmemory-policy allkeys-lru
19             --protected-mode yes
20         healthcheck:
21             test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}", "PING"]
22             interval: 10s
23             timeout: 3s
24             retries: 5
25         volumes:
26             - redis_data:/data
27         restart: always
28         networks: [backend]
29         # NOTE: do NOT publish ports in prod; keep it internal

```

7.3 Database Connection & Transactions (lib/db/index.ts)

Listing 4: PG Pool and transactional helper

```

1 import { Pool } from 'pg';
2 export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
3 // Ensure DATABASE_URL includes sslmode=require for Supabase.
4
5 export async function withTx<T>(fn: (c: import('pg').PoolClient) => Promise<T>) {
6     const client = await pool.connect();
7     try {
8         await client.query('BEGIN');
9         const r = await fn(client);
10        await client.query('COMMIT');
11        return r;
12    } catch (e) {
13        await client.query('ROLLBACK'); throw e;
14    } finally {

```

```
15     client.release();
16   }
17 }
```

8 Shared Libraries

8.1 Text Utilities: Normalization & Chunking (lib/shared/text.ts)

Listing 5: Normalize strings and chunk long text

```
1 export function normalizeForKeywords(raw: string) {
2   return raw
3     .normalize('NFKC')
4     .toLowerCase()
5     .replace(/[\p{L}\p{N}\s.%/-]/gu, ' ')
6     .replace(/\s+/g, ' ')
7     .trim();
8 }
9
10 // ~800-1200 tokens ≈ ~3500-5000 chars (rough)
11 export function chunkByChars(s: string, maxChars = 4000) {
12   const parts: string[] = [];
13   let buf = '';
14   for (const p of s.split(/\n{2,}/)) {
15     if ((buf + '\n\n' + p).length > maxChars) {
16       if (buf) parts.push(buf.trim());
17       if (p.length > maxChars) {
18         for (let i = 0; i < p.length; i += maxChars) {
19           parts.push(p.slice(i, i + maxChars));
20         }
21         buf = '';
22       } else {
23         buf = p;
24       }
25     } else {
26       buf = buf ? `${buf}\n\n${p}` : p;
27     }
28   }
29   if (buf) parts.push(buf.trim());
30   return parts;
31 }
```

9 Matching Engine (v1)

9.1 Retrieval pipeline: normalization \rightarrow embeddings \rightarrow vector search

Purpose. Describe, once, how text from RFPs, offerings, and KB entries is converted into vectors and retrieved for scoring.

9.1.1 Pipeline overview.

1. **Text normalization & chunking:** NFKC, lowercase, preserve `./-`, collapse spaces; chunk by characters with small overlaps.
2. **Embeddings:** OpenAI `text-embedding-3-large` (\mathbb{R}^{3072}); persist to `embeddings(vector, text, org_id, kind, ref_id, chunk_id)` with pgvector index (IVFFLAT/HNSW).
3. **Vector search:** for each RFP chunk, run cosine KNN over offering (and KB) chunks using `<=>` under `vector_cosine_ops`; retrieve top- k candidates per chunk.
4. **Aggregation into scores:** pool per-chunk results into S_{sem} (top- k mean by default), combine with S_{kw} and S_{rule} in §9.2.

9.1.2 Implementation Considerations.

- **Scope.** This pipeline applies uniformly to RFP \leftrightarrow Offering, and to RFP \leftrightarrow KB retrieval used by the Assistant (RAG).
- **Consistency.** Normalization for keyword matching is identical to §9.2.3; embeddings are unaffected by normalization (they operate on raw chunk text).
- **Performance.** Use ANN indexes (IVFFLAT/HNSW) with sensible params; optionally shortlist offerings via per-chunk KNN before full scoring.

9.2 Scoring Model

We combine: *semantic*, *keyword*, and *rules*.

9.2.1 Components

- **Semantic** $S_{\text{sem}} \in [0, 1]$: mean of top- k cosine similarities per offering (default $k = 3$), clamped.
- **Keyword** $S_{\text{kw}} \in [0, 1]$: sum of configured boosts for normalized exact-term hits, capped at 1.
- **Rules** $S_{\text{rule}} \in [0, 1]$: penalty for missing required / present forbidden, then $1 - \text{penalty}$, clamped.

9.2.1.1 Normalization of component scores to the unit interval $[0, 1]$

1. **Interpretability.** A $[0, 1]$ scale reads like a probability/percentage: 0 = none, 1 = maximum.
2. **Negative cosine is not useful for semantics.** Cosine similarity lives in $[-1, 1]$, but for language embeddings negative values rarely indicate “anti-meaning”; we treat them as “unrelated” $\Rightarrow 0$.

3. **Consistency with pgvector distance.** Under `vector_cosine_ops`, `<=>` returns cosine *distance* in $[0, 2]$. We form similarity as $1 - \text{distance}$ and then clamp to $[0, 1]$.
4. **Aggregation stability.** Averaging (e.g., top- k means) is more stable when each contribution lies in $[0, 1]$, making scores comparable across datasets.

Operationalization. We normalize each component as follows:

$$\begin{aligned}
S_{\text{sem}}^{(\text{pair})} &= \max\{0, \text{sim}_{\text{cos}}(\mathbf{a}, \mathbf{b})\}, \\
S_{\text{kw}} &= \min\left(1, \sum_t w_t \mathbf{1}_{\text{substr}(\text{normalize}(t), \text{normalize}(\text{text}))}\right), \\
S_{\text{rule}} &= \max(0, \min(1, 1 - \text{penalty})).
\end{aligned}$$

This enforces $S_{\text{sem}}, S_{\text{kw}}, S_{\text{rule}} \in [0, 1]$ and yields a well-behaved hybrid score.

9.2.1.2 Conceptual Interpretation of the Scoring Components

Semantic (“does it feel right?”) Imagine reading two stories and asking whether they talk about the same idea. We take a few best-matching parts (top- k , usually $k = 3$), average their similarity, and keep the result between 0 and 1 (“clamped”).

Keyword (“does it say the magic words?”) We keep a list of important phrases (e.g., “SOC2”, “24/7”). Each time the text contains one, we add some points, stopping at 1 (“capped at 1”).

Rules (“did it follow must-do / must-not-do?”) There are green-list terms (must have) and red-list terms (must not have). Missing green-list items and seeing red-list items both increase a penalty. The rules meter starts at 1 and goes down by that penalty, and is kept between 0 and 1.

Put simply:

- **Semantic** = “does it talk about the same idea?”
- **Keyword** = “does it say the special words?”
- **Rules** = “did it follow the simple yes/no rules?”

9.2.2 Semantic Score $S_{\text{sem}} \in [0, 1]$

9.2.2.1 Definition of Semantic Score

Semantic Score S_{sem} measures how strongly an offering discusses the same ideas as an RFP. We embed text chunks for each document using OpenAI `text-embedding-3-large` (dimension 3072), store them in Postgres `pgvector`, and compute cosine-based similarity:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \in [-1, 1], \quad d_{\text{cos}}(\mathbf{a}, \mathbf{b}) = 1 - \text{sim}(\mathbf{a}, \mathbf{b}).$$

In pgvector (implementation): under `vector_cosine_ops`, the SQL operator `<=>` returns cosine *distance*:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = 1 - (\mathbf{a} <=> \mathbf{b}).$$

Aggregation uses a *top-k mean* over chunk–chunk matches (default $k=3$): for each RFP chunk, average its best k offering-chunk similarities, then average across RFP chunks.

- **Embeddings:** $\mathbf{v} \in \mathbb{R}^{3072}$ in `embeddings(vector)`.
- **Similarity:** cosine similarity as above; in SQL, `<=>` yields cosine distance.
- **Aggregation:** per-RFP-chunk top- k mean, then mean across RFP chunks.

9.2.2.2 Vector Search (semantic similarity)

Given an RFP *chunk* vector \mathbf{v} , find the most semantically similar *offering* chunks (requires an `ivfflat` or `hnsw` index):

Listing 6: Vector search for a single RFP chunk against offering embeddings (pgvector, cosine)

```

1 -- $1::vector is the RFP chunk vector; $2 is org_id
2 SELECT e.ref_id AS offering_id,
3        1 - (e.vector <=> $1::vector) AS sim
4 FROM embeddings e
5 WHERE e.kind = 'offering' AND e.org_id = $2
6 ORDER BY e.vector <=> $1::vector    -- ascending distance = descending similarity
7 LIMIT 20;

```

Aggregation per RFP: loop over its chunks; for each chunk fetch top- k offering matches. Aggregate to an *offering-level* semantic score using either:

- $S_{\text{sem}}^{\text{max}}(\text{off}) = \max_{\text{chunks}} \text{sim}$ (highlights strongest match),
- $S_{\text{sem}}^{\text{topN-mean}}(\text{off}) = \frac{1}{N} \sum_{\ell=1}^N \text{sim}(\ell)$ (more robust to noise).

9.2.2.3 Semantic Score Algorithm

Let $R = \{r_1, \dots, r_{|R|}\}$ be RFP chunks and $O = \{o_1, \dots, o_{|O|}\}$ offering chunks, with embeddings $\mathbf{r}_i, \mathbf{o}_j \in \mathbb{R}^{3072}$.

Step 1 Compute $\text{sim}_{ij} = \text{sim}(\mathbf{r}_i, \mathbf{o}_j) \in [0, 1]$.

Step 2 Top- k : for each i , take $T_i = \text{TopK}(\{\text{sim}_{ij}\}_{j=1}^{|O|}, k)$ of size $k' = \min(k, |O|)$.

Step 3 Per-chunk mean: $m_i = \frac{1}{k'} \sum_{t \in T_i} t$.

Step 4 Across-chunks mean: $\tilde{S}_{\text{sem}} = \frac{1}{|R|} \sum_{i=1}^{|R|} m_i$.

Step 5 Clamp: $S_{\text{sem}} = \max(0, \min(1, \tilde{S}_{\text{sem}}))$.

Listing 7: pgvector SQL: top-k mean per RFP chunk, then average

```

1 -- Inputs: :rfp_id, :offering_id, :org_id, :k
2 WITH rfp_chunks AS (
3   SELECT e.chunk_id, e.vector

```

```

4  FROM embeddings e
5  WHERE e.org_id = :org_id AND e.kind = 'rfp' AND e.ref_id = :rfp_id
6  ),
7  off_chunks AS (
8    SELECT e.chunk_id, e.vector
9    FROM embeddings e
10   WHERE e.org_id = :org_id AND e.kind = 'offering' AND e.ref_id = :offering_id
11  ),
12  topk_per_rfp AS (
13    SELECT
14      r.chunk_id AS r_chunk_id,
15      AVG(1 - (r.vector <=> o.vector)) AS mean_topk
16    FROM rfp_chunks r
17    CROSS JOIN LATERAL (
18      SELECT o.vector
19      FROM off_chunks o
20      ORDER BY r.vector <=> o.vector
21      LIMIT :k
22    ) o
23    GROUP BY r.chunk_id
24  )
25  SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
26  FROM topk_per_rfp;

```

9.2.2.4 Scope & Delimitations

- **Chunking:** avoid very long chunks (dilution) or very short chunks (noise). Keep stable size and small overlap.
- **Top- k :** small k is outlier-prone; large k over-smooths. Default $k=3$.
- **Indexing:** use IVFFLAT; consider HNSW for better recall/latency where available.
- **Explainability:** persist best snippet pairs (chunk ids + preview) into `matches.reasons`.

9.2.2.5 Example

Two RFP chunks (r_1, r_2), three offering chunks (o_1, o_2, o_3):

$$\begin{aligned} \text{sim}(r_1, o_1) &= 0.88, \text{sim}(r_1, o_2) = 0.42, \text{sim}(r_1, o_3) = 0.66, \\ \text{sim}(r_2, o_1) &= 0.35, \text{sim}(r_2, o_2) = 0.71, \text{sim}(r_2, o_3) = 0.62. \end{aligned}$$

With $k=2$: $m_1 = (0.88 + 0.66)/2 = 0.77$, $m_2 = (0.71 + 0.62)/2 = 0.665$. Then $\tilde{S}_{\text{sem}} = (0.77 + 0.665)/2 = 0.7175$, so $S_{\text{sem}} = 0.7175$.

9.2.2.6 Tuning

- k : sweep $\{1, 3, 5\}$; inspect ranking stability and snippet coherence.
- **Chunk size/overlap:** try $700 \sim 1200$ chars, overlap $50 \sim 150$.

- **Index params:** tune #lists for IVFFLAT; for HNSW, tune M and ef_search .
- **Variants (optional):** section-weighted pooling or symmetric pooling (also top- k from offering→RFP).

9.2.3 Keyword score $S_{kw} \in [0, 1]$

9.2.3.1 Definition of Keyword Score

Keyword Score S_{kw} rewards the presence of configured exact terms (after normalization) in the offering text. Each term t has a nonnegative weight w_t defined per org in `score_config.boosts`. The score is the capped sum of weights for terms found:

$$S_{kw} = \min\left(1, \sum_{t \in \mathcal{T}} w_t \cdot \mathbf{1}\{\text{norm}(\text{text}) \text{ contains } \text{norm}(t)\}\right),$$

where `norm(·)` applies text normalization (NFKC, lowercase, preserve `./-/`, collapse spaces).

Listing 8: Example per-org boosts (stored in ‘score_config.boosts’)

```

1 {
2   "boosts": {
3     "soc2": 0.25,
4     "hipaa": 0.20,
5     "24/7": 0.15,
6     "99.99%": 0.10,
7     "iso 27001": 0.20
8   }
9 }
```

9.2.3.2 Keyword Score Algorithm

Let `text` be the offering text (title + description + relevant fields). Let `Boosts` = $\{(t, w_t)\}$.

Step 1 Normalize both text and each term t : NFKC → lowercase → remove non-alphanumerics except `./-/` → collapse spaces.

Step 2 Accumulate: $s \leftarrow \sum_{(t, w_t) \in \text{Boosts}} w_t \cdot \mathbf{1}\{\text{norm}(\text{text}) \text{ contains } \text{norm}(t)\}$.

Step 3 Cap to $[0, 1]$: $S_{kw} \leftarrow \min(1, s)$.

Listing 9: Keyword score with normalization and capping

```

1 import { normalizeForKeywords } from './text'; // NFKC -> lower -> keep ./-/ -> collapse
   spaces
2
3 export function keywordScore(offTxt: string, boosts: Record<string, number>) {
4   const text = normalizeForKeywords(offTxt);
5   let score = 0;
6   for (const [term, w] of Object.entries(boosts)) {
```

```

7   if (text.includes(normalizeForKeywords(term))) score += w;
8   }
9   return Math.min(1, score); //  $S_{kw}$  in  $[0,1]$ 
10  }

```

Listing 10: Reading boosts for an org (used by API/worker)

```

1 SELECT boosts
2 FROM score_config
3 WHERE org_id = $1;

```

9.2.3.3 Scope & Delimitations

- **Exact substrings after normalization:** no stemming/lemmatization by default; add aliases if needed (e.g., “soc 2”, “soc ii”).
- **Symbols preserved:** -, %, /, . are kept so terms like 24/7, 99.99%, iso 27001 match reliably.
- **Case-insensitive:** enforced by lowercasing in normalization.
- **Explainability:** record matched terms and their weights in `matches.reasons.keyword_hits`.

9.2.3.4 Example

Config: `boosts={"soc2":0.25,"24/7":0.15,"iso 27001":0.2}`. Offering text (normalized) contains `soc2` and `24/7` but not `iso 27001`. Then

$$S_{kw} = \min(1, 0.25 + 0.15) = 0.40.$$

9.2.3.5 Tuning

- **Weights:** calibrate $\{w_t\}$ to reflect business-critical terms; keep $\sum w_t \leq 1$ if we want natural capping.
- **Aliases / n-grams:** add common variants (“soc 2”, “soc ii”) to reduce phrasing brittleness; optionally introduce light lemmatization.
- **Noise control:** prefer a concise list of high-signal terms; revisit weights using offline evals and error analysis.

9.2.4 Rules $S_{rule} \in [0, 1]$

9.2.4.1 Definition of Rules

Rules are non-semantic constraints that adjust the score using simple, configurable checks stored per org in `score_config`:

- **Required terms** (`required[]`): phrases that *must* appear in the offering text. Missing terms increase a penalty.
- **Forbidden terms** (`forbidden[]`): phrases that *must not* appear. Presence increases a penalty.

Listing 11: Rule-related columns in `score_config`

```

1 -- Per-org configuration of rules and keyword boosts
2 -- (Boosts are used by the keyword component, not the rules component.)
3 org_id      uuid PRIMARY KEY,
4 boosts      jsonb NOT NULL DEFAULT '{}'::jsonb, -- e.g., {"soc2":0.25,"24/7":0.15}
5 required    jsonb NOT NULL DEFAULT '[]'::jsonb, -- e.g., ["soc2","99.99% uptime"]
6 forbidden   jsonb NOT NULL DEFAULT '[]'::jsonb  -- e.g., ["on-prem only"]

```

9.2.4.2 Rule Score Algorithm

Let the (normalized) offering text be compared against configured terms:

- **Normalize** both text and terms: NFKC, lowercase, strip non-alphanumerics except `./-/,` collapse spaces.
- **Measure misses/hits:**
 - `miss` = count of required terms not found
 - `forb` = count of forbidden terms found
- **Convert to proportions:**
 - $p_{\text{Req}} = \text{miss} / |\text{required}|$ (or 0 if no required terms)
 - $p_{\text{For}} = \text{forb} / |\text{forbidden}|$ (or 0 if no forbidden terms)
- **Weighted penalty:** $\text{penalty} = 0.7 p_{\text{Req}} + 0.3 p_{\text{For}}$.
- **Rule score:** $S_{\text{rule}} = \max(0, \min(1, 1 - \text{penalty}))$.

Missing required terms is penalized more than finding forbidden terms by design (0.7 vs. 0.3).

Listing 12: JavaScript design for rule score computation

```

1 import { normalizeForKeywords } from './text';
2
3 function ruleCompliance(text: string, required: string[], forbidden: string[]) {
4   const norm = normalizeForKeywords(text);
5   const miss = required.filter(r => !norm.includes(normalizeForKeywords(r))).length;
6   const forb = forbidden.filter(f => norm.includes(normalizeForKeywords(f))).length;
7   const pReq = required.length ? miss / required.length : 0;
8   const pFor = forbidden.length ? forb / forbidden.length : 0;
9   const penalty = 0.7 * pReq + 0.3 * pFor;
10  return Math.max(0, Math.min(1, 1 - penalty));
11 }

```

9.2.4.3 Scope & Delimitations

- **Exact substrings after normalization:** no stemming/lemmatization. Consider synonyms if needed.
- **Symbols preserved:** hyphens, %, and slashes survive normalization, so 24/7, 99.99% still match.
- **Case-insensitive** by design (lowercased).
- **Explainability:** we persist `required_missing` and `forbidden_hit` in `matches.reasons`.

9.2.4.4 Example

Config: `required=["soc2", "99.99% uptime"], forbidden=["on-prem only"]`.

Offering contains `SOC2`, lacks `99.99% uptime`, and does not contain `on-prem only`. Then:

$$p_{\text{Req}} = \frac{1}{2} = 0.5, \quad p_{\text{For}} = 0, \quad \text{penalty} = 0.7 \cdot 0.5 + 0.3 \cdot 0 = 0.35, \quad S_{\text{rule}} = 0.65.$$

9.2.4.5 Tuning

- **Hard gates:** increase the final weight of rules (e.g., 0.6/0.2/0.2) or pre-filter offerings on critical must-haves.
- **Reduce false negatives:** add synonyms in `required[]` or switch to a light fuzzy/regex matcher.

9.2.5 Final Score

$$S_{\text{final}} = 0.7 S_{\text{sem}} + 0.2 S_{\text{kw}} + 0.1 S_{\text{rule}}$$

clamped to $[0, 1]$.

9.2.6 Example Calculation of S_{sem} , S_{kw} , S_{rule} and S_{final}

9.2.6.1 RFP extract (single chunk).

“Looking for **SOC2 Type II** compliant **cloud hosting** with **24/7 support** and **99.99% uptime**.”

9.2.6.2 Offering A (title + description).

“Cloud Hosting (Enterprise). **SOC2 Type II, 24/7 support, SLA 99.99%**.”

Component scores:

- S_{sem} : from pgvector (e.g., max across RFP chunks) \rightarrow **0.84**.
- S_{kw} : term hits: `soc2`(0.25) + `24/7`(0.15) + `99.99%`(0.10) \rightarrow **0.50** (capped ≤ 1).
- S_{rule} : `required`={“soc2”, “24/7”}, `forbidden`={“on-prem only”}.
miss=0, hitForbidden=0 \Rightarrow penalty = 0 \Rightarrow **$S_{\text{rule}} = 1.00$** .

Final score:

$$S_{\text{final}} = 0.7 \cdot 0.84 + 0.2 \cdot 0.50 + 0.1 \cdot 1.00 = 0.588 + 0.10 + 0.10 = \mathbf{0.788}.$$

9.2.6.3 Offering B (title + description).

“Budget VPS. No compliance guarantees.”

Component scores:

- $S_{\text{sem}} \approx \mathbf{0.22}$.
- $S_{\text{kw}} = \mathbf{0.00}$.
- S_{rule} : same **required**; miss= 2/2 $\Rightarrow p_{\text{Req}} = 1$, penalty= 0.7 $\Rightarrow \mathbf{S_{rule} = 0.30}$.

Final score:

$$S_{\text{final}} = 0.7 \cdot 0.22 + 0.2 \cdot 0 + 0.1 \cdot 0.30 = 0.154 + 0 + 0.03 = \mathbf{0.184}.$$

9.2.6.4 Ranking.

Offering A (0.788) \gg Offering B (0.184).

9.3 Worker Task: Score Matches (`apps/worker/src/scoreMatches.task.ts`)

The `scoreMatches` worker is the component that *materializes* a match result from previously embedded data: it computes the hybrid score (semantic similarity + keyword boosts + rule compliance) and persists ranked `matches` for each RFP. In other words, it is the core execution unit of the matching engine’s scoring stage: taking vectorized RFP/Offering representations and turning them into actionable, ranked results that drive the UI and downstream workflow. Background queues and workers are used only as an execution model; the algorithmic logic (*how* we score) is what binds this task to the Matching Engine (v1) rather than to generic job infrastructure.

9.3.1 Design process of the worker

1. **Inputs.** Receives { `rfpId`, `orgId` } from the queue once embeddings for the RFP are ready.
2. **Candidate set.** Pulls all offerings for the organization (`offerings.id`) as matching candidates.
3. **Semantic score.** For each candidate, computes cosine similarity between the offering’s vectors and the RFP’s vectors using `pgvector`. To increase robustness and reduce outlier effects, aggregates as the mean of top- k similarities (default $k=3$).
4. **Lexical features.** Reads `score_config` (boosts, required, forbidden) and computes (a) keyword score via normalized exact-term hits and (b) rule compliance as $1 - \text{penalty}$ for missing required/present forbidden terms.
5. **Hybrid scoring.** Combines features via a simple, transparent formula $S = 0.7 S_{\text{sem}} + 0.2 S_{\text{kw}} + 0.1 S_{\text{rule}}$, clamped to $[0, 1]$.
6. **Explanations.** Captures the top semantic snippet and lists keyword/rule hits into `reasons` (JSON) for “why it matched.”
7. **Persistence.** Deletes prior rows for (`org_id`, `rfp_id`) and inserts the top- N (default 10) matches into `matches`, wrapped in a DB transaction for atomicity.

8. **Idempotency & runtime.** The worker is safe to re-run for the same `rfpId` (previous results are replaced), and concurrency is controlled at the queue level.

Listing 13: Cosine KNN + keyword/rules to rank offerings

```
1 import { pool } from '../.../lib/db';
2 import { keywordScore, ruleCompliance, finalScore } from '../.../lib/shared/scoring';
3
4 async function getScoreConfig(orgId: string) {
5   const { rows } = await pool.query(
6     'SELECT boosts, required, forbidden FROM score_config WHERE org_id=$1', [orgId]
7   );
8   if (!rows[0]) return { boosts:{}, required:[], forbidden:[] };
9   const { boosts, required, forbidden } = rows[0];
10  return { boosts, required, forbidden };
11 }
12
13 async function semanticForOffering(orgId: string, rfpId: string, offeringId: string, k =
14   3): Promise<number> {
15   const { rows } = await pool.query('
16     WITH rfp_chunks AS (
17       SELECT vector
18       FROM embeddings
19       WHERE org_id = $1 AND kind = 'rfp' AND ref_id = $2
20     ),
21     off_chunks AS (
22       SELECT vector
23       FROM embeddings
24       WHERE org_id = $1 AND kind = 'offering' AND ref_id = $3
25     ),
26     topk_per_rfp AS (
27       SELECT AVG(1 - (r.vector <=> o.vector)) AS mean_topk
28       FROM rfp_chunks r
29       CROSS JOIN LATERAL (
30         SELECT o.vector
31         FROM off_chunks o
32         ORDER BY r.vector <=> o.vector
33         LIMIT $4
34       ) o
35       GROUP BY r.vector
36     )
37     SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
38     FROM topk_per_rfp
39     ', [orgId, rfpId, offeringId, k]);
```

```

40     return Number(rows[0]?.s_sem ?? 0);
41 }
42
43 async function offeringText(offeringId: string) {
44     const { rows } = await pool.query(
45         'SELECT title, description FROM offerings WHERE id=$1', [offeringId]
46     );
47     if (!rows[0]) return '';
48     return `${rows[0].title}\n\n${rows[0].description}`;
49 }
50
51 export async function scoreMatchesTask({ rfpId, orgId }: { rfpId:string; orgId:string }) {
52     const offs = await pool.query('SELECT id FROM offerings WHERE org_id=$1', [orgId]);
53     const cfg = await getScoreConfig(orgId);
54
55     const results: Array<{ offering_id:string; score:number; reasons:any }> = [];
56
57     for (const o of offs.rows) {
58         const offId = o.id as string;
59         const sem = await semanticForOffering(orgId, rfpId, offId);
60         const text = await offeringText(offId);
61         const kw = keywordScore(text, cfg.boosts);
62         const rule = ruleCompliance(text, cfg.required, cfg.forbidden);
63         const score = finalScore(sem, kw, rule);
64
65         const topChunk = await pool.query(
66             'SELECT e.text, (1 - (e.vector <=> r.vector)) AS sim
67             FROM embeddings e
68             JOIN embeddings r ON r.ref_id=$1 AND r.kind=\'rfp\' AND r.org_id=$2
69             WHERE e.ref_id=$3 AND e.kind=\'offering\' AND e.org_id=$2
70             ORDER BY sim DESC
71             LIMIT 1
72             ', [rfpId, orgId, offId]);
73
74         results.push({
75             offering_id: offId,
76             score,
77             reasons: {
78                 semantic_top_sim: Number(topChunk.rows[0]?.sim ?? 0),
79                 semantic_snippet: topChunk.rows[0]?.text ?? null,
80                 keyword_hits: Object.keys(cfg.boosts).filter(k => text.toLowerCase().includes(k.
81                     toLowerCase())),
82                 required_missing: cfg.required.filter(r => !text.toLowerCase().includes(r.
83                     toLowerCase())),

```

```

82     forbidden_hit: cfg.forbidden.filter(f => text.toLowerCase().includes(f.
      toLowerCase())),
83   }
84 });
85 }
86
87 results.sort((a,b)=>b.score - a.score);
88 const top = results.slice(0,10);
89 const client = await pool.connect();
90 try {
91   await client.query('BEGIN');
92   await client.query('DELETE FROM matches WHERE org_id=$1 AND rfp_id=$2', [orgId, rfpId
    ]);
93   for (const r of top) {
94     await client.query('
95       INSERT INTO matches(org_id, rfp_id, offering_id, score, reasons)
96       VALUES ($1,$2,$3,$4,$5)
97       ', [orgId, rfpId, r.offering_id, r.score, r.reasons]);
98   }
99   await client.query('COMMIT');
100 } catch (e) { await client.query('ROLLBACK'); throw e; }
101 finally { client.release(); }
102
103 return { count: top.length };
104 }

```

10 Background Jobs & Worker Chain

Queues (BullMQ): parse-doc, embed-chunks, score-matches, optional sync-source.

10.1 Worker Sequence

1. **ParseDocWorker**: fetch file → extract text → chunk → insert chunks.
2. **EmbedChunksWorker**: batch embed → insert embeddings (RLS scope set via `app.org_id`).
3. **ScoreMatchesWorker**: for RFP, compute hybrid scores vs offerings → write matches (RLS scope set via `app.org_id`).

10.2 Job Infrastructure: Queues & Types (`apps/worker/src/queue.ts`)

Listing 14: BullMQ queue setup and job payloads (RLS-ready)

```

1 import { Queue, QueueEvents, JobsOptions } from 'bullmq';
2 import IORedis from 'ioredis';
3

```

```

4 export const redis = new IORedis(process.env.REDIS_URL ?? {
5   host: process.env.REDIS_HOST,
6   port: Number(process.env.REDIS_PORT || 6379),
7   password: process.env.REDIS_PASSWORD,
8   tls: process.env.REDIS_TLS === '1' ? {} : undefined,
9 } as any);
10
11 export function makeQueue(name: string) {
12   const q = new Queue(name, { connection: redis });
13   const events = new QueueEvents(name, { connection: redis });
14   return { q, events };
15 }
16
17 // Include orgId in all payloads that will touch RLS-protected tables
18 export type ParseDocPayload = { documentId: string; orgId: string };
19 export type EmbedChunksPayload = { documentId: string; orgId: string; kind: 'rfp'|'kb'|'
    offering' };
20 export type ScoreMatchesPayload = { rfpId: string; orgId: string };
21
22 export const queues = {
23   parseDoc: makeQueue('parse-doc'),
24   embedChunks: makeQueue('embed-chunks'),
25   scoreMatches: makeQueue('score-matches'),
26 };
27
28 // Align with spec: attempts=5, exponential backoff base 250ms
29 export const defaultJobOpts: JobsOptions = {
30   attempts: 5,
31   backoff: { type: 'exponential', delay: 250 },
32   removeOnComplete: { age: 86400 },
33   removeOnFail: { age: 86400 },
34 };

```

10.3 OpenAI Embeddings Adapter (apps/worker/src/embeddings.ts)

Listing 15: Batch embed texts with OpenAI

```

1 import OpenAI from 'openai';
2 const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });
3
4 export async function embedBatch(texts: string[]) {
5   const res = await client.embeddings.create({
6     model: 'text-embedding-3-large',
7     input: texts

```

```

8   });
9   return res.data.map(d => d.embedding);
10  }

```

10.4 Worker Task: Parse Documents (apps/worker/src/parseDoc.task.ts)

Listing 16: Extract text, chunk, enqueue embeddings (RLS-safe)

```

1  import { pool } from '../.../lib/db';
2  import { chunkByChars } from '../.../lib/shared/text';
3  import { queues, defaultJobOpts } from './queue';
4  import pdf from 'pdf-parse';
5
6  // Node 18+ has global fetch; avoid extra deps
7  async function fetchFile(storageUrl: string): Promise<Buffer> {
8      const r = await fetch(storageUrl);
9      if (!r.ok) throw new Error('fetch failed: ${r.status}');
10     return Buffer.from(await r.arrayBuffer());
11 }
12
13 export async function parseDoc({ documentId, orgId }: { documentId: string; orgId: string
14     }) {
15     // Use a single client/tx so we can SET LOCAL app.org_id per the RLS policy
16     const client = await pool.connect();
17     try {
18         await client.query('BEGIN');
19         await client.query('SET LOCAL app.org_id = $1', [orgId]);
20
21         const { rows } = await client.query(
22             'SELECT id, org_id, kind, storage_url, mime FROM documents WHERE id=$1',
23             [documentId]
24         );
25         if (!rows[0]) throw new Error('doc not found');
26         const { storage_url, mime, kind } = rows[0];
27
28         const buf = await fetchFile(storage_url);
29         let text = '';
30         if ((mime || '').includes('pdf')) {
31             const data = await pdf(buf);
32             text = data.text || '';
33         } else {
34             text = buf.toString('utf8');
35         }

```

```

36     const parts = chunkByChars(text, 4000);
37     for (let i = 0; i < parts.length; i++) {
38         await client.query(
39             'INSERT INTO chunks(document_id, idx, text) VALUES ($1, $2, $3)',
40             [documentId, i, parts[i]]
41         );
42     }
43
44     await client.query('COMMIT');
45
46     // Enqueue embedding with org/kind for downstream RLS + branching
47     await queues.embedChunks.q.add('embed', { documentId, orgId, kind }, defaultJobOpts);
48     return { chunks: parts.length };
49 } catch (e) {
50     await client.query('ROLLBACK');
51     throw e;
52 } finally {
53     client.release();
54 }
55 }

```

10.5 Worker Task: Embed Chunks (apps/worker/src/embedChunks.task.ts)

Listing 17: Embed chunk texts and persist vectors (RLS-safe)

```

1 import { pool } from '../../lib/db';
2 import { embedBatch } from './embeddings';
3 import { queues, defaultJobOpts } from './queue';
4
5 export async function embedChunksTask(
6     { documentId, orgId, kind }: { documentId: string; orgId: string; kind: 'rfp'|'kb'|'
7     offering' }
8 ) {
9     // chunks table is not RLS-protected; embeddings is. We'll set RLS for writes.
10    const { rows: chunkRows } = await pool.query(
11        'SELECT id, text FROM chunks WHERE document_id=$1 ORDER BY idx',
12        [documentId]
13    );
14
15    const batchSize = 64;
16    for (let i = 0; i < chunkRows.length; i += batchSize) {
17        const slice = chunkRows.slice(i, i + batchSize);
18        const vecs = await embedBatch(slice.map(r => r.text));

```

```

18     const textAndVec = slice.map((r, j) => ({ chunkId: r.id, text: r.text, vec: vecs[j]
19     }));
20
21     const client = await pool.connect();
22     try {
23         await client.query('BEGIN');
24         await client.query('SET LOCAL app.org_id = $1', [orgId]); // RLS scope for
25         embeddings
26
27         for (const tv of textAndVec) {
28             await client.query(
29                 'INSERT INTO embeddings(org_id, kind, ref_id, chunk_id, vector, text)
30                 VALUES ($1,$2,$3,$4,$5::vector,$6)
31                 ON CONFLICT (org_id, kind, ref_id, chunk_id)
32                 DO UPDATE SET vector=EXCLUDED.vector, text=EXCLUDED.text',
33                 [orgId, kind, documentId, tv.chunkId, `${tv.vec.join(',')}`, tv.text]
34             );
35         }
36         await client.query('COMMIT');
37     } catch (e) {
38         await client.query('ROLLBACK'); throw e;
39     } finally {
40         client.release();
41     }
42 }
43
44 if (kind === 'rfp') {
45     await queues.scoreMatches.q.add('score', { rfpId: documentId, orgId }, defaultJobOpts
46     );
47 }
48 }
49 }

```

10.6 Worker Entrypoint: Concurrency & Startup (apps/worker/src/main.ts)

Listing 18: Start BullMQ workers with concurrency

```

1 import { Worker } from 'bullmq';
2 import { redis } from './queue';
3 import { parseDoc } from './parseDoc.task';
4 import { embedChunksTask } from './embedChunks.task';
5 import { scoreMatchesTask } from './scoreMatches.task';
6
7 new Worker('parse-doc', async job => parseDoc(job.data), { connection: redis, concurrency
8   : 4 });

```

```

8 new Worker('embed-chunks', async job => embedChunksTask(job.data), { connection: redis,
  concurrency: 8 });
9 new Worker('score-matches', async job => scoreMatchesTask(job.data), { connection: redis,
  concurrency: 4 });
10
11 console.log('Workers up: parse-doc, embed-chunks, score-matches');

```

10.7 Operational Resilience

10.7.1 Delivery semantics & idempotency

10.7.1.1 Reference enqueue + idempotent handler

Listing 19: apps/worker/src/score.enqueue.ts

```

1 import type { JobsOptions } from 'bullmq';
2 import { queues } from './queue';
3 import { Client as Pg } from 'pg';
4
5 /** In-queue de-dup: jobId = orgId:rfpId */
6 export async function enqueueReScore(orgId: string, rfpId: string) {
7   const opts: JobsOptions = {
8     jobId: `${orgId}:${rfpId}`,
9     attempts: 5,
10    backoff: { type: 'exponential', delay: 250 },
11    removeOnComplete: { age: 86400 },
12    removeOnFail: { age: 86400 },
13  };
14  return queues.scoreMatches.q.add('re-score', { orgId, rfpId }, opts);
15 }
16
17 /** Idempotent writer: delete+insert or UPSERT inside a tx */
18 export async function recomputeMatches(pg: Pg, orgId: string, rfpId: string) {
19   await pg.query('BEGIN');
20   try {
21     await pg.query('SET LOCAL app.org_id = $1', [orgId]); // RLS scope
22     await pg.query('DELETE FROM matches WHERE org_id=$1 AND rfp_id=$2', [orgId, rfpId]);
23
24     // ... compute rows
25     const rows = [{ offering_id: 'off-1', score: 0.87 }];
26
27     const sql = `
28       INSERT INTO matches (org_id, rfp_id, offering_id, score, updated_at)
29       VALUES ($1,$2,$3,$4,now())
30       ON CONFLICT (org_id,rfp_id,offering_id)

```



```

31     DO UPDATE SET score=EXCLUDED.score, updated_at=now()
32     ';
33     for (const r of rows) {
34         await pg.query(sql, [orgId, rfpId, r.offering_id, r.score]);
35     }
36     await pg.query('COMMIT');
37 } catch (e) {
38     await pg.query('ROLLBACK');
39     throw e;
40 }
41 }

```

10.7.1.2 Worker wiring (graceful drain compatible)

Listing 20: apps/worker/src/score.worker.ts

```

1 import { Worker, QueueEvents } from 'bullmq';
2 import { redis, queues } from './queue';
3 import { Client as Pg } from 'pg';
4 import { recomputeMatches } from './score.enqueue';
5
6 const events = new QueueEvents('score-matches', { connection: redis });
7
8 const pg = new Pg({ connectionString: process.env.DATABASE_URL! });
9 await pg.connect();
10
11 export const scoreWorker = new Worker('score-matches', async (job) => {
12     const { orgId, rfpId } = job.data as { orgId: string; rfpId: string };
13     await recomputeMatches(pg, orgId, rfpId);
14     // tiny jitter to reduce herds
15     await new Promise(r => setTimeout(r, Math.random() * 50));
16     return { ok: true };
17 }, { connection: redis, concurrency: 4 });
18
19 process.on('SIGTERM', async () => {
20     await queues.scoreMatches.q.pause(true); // global pause
21     await scoreWorker.close();                // finish in-flight
22     await events.close();
23     await pg.end();
24 });

```

10.7.2 Retries/backoff & budgets

10.7.2.1 Retry/backoff (Worker options) + DLQ forwarding

Listing 21: apps/worker/src/retry-dlq.ts

```

1 import { Worker, Queue } from 'bullmq';
2 import { redis } from './queue';
3
4 export const dlq = new Queue('dlq:score-matches', { connection: redis });
5
6 export function withRetry(workerName: string, handler: (job:any)=>Promise<any>) {
7   const w = new Worker(workerName, handler, {
8     connection: redis,
9     concurrency: 8,
10    // attempts/backoff are typically set per-job; this is a safe default if missing
11    settings: { backoffStrategies: {} }
12  });
13
14  // Route exhausted failures to DLQ
15  w.on('failed', async (job, err) => {
16    if (!job) return;
17    const max = job.opts.attempts ?? 1;
18    if (job.attemptsMade >= max) {
19      await dlq.add('dead-${workerName}', { data: job.data, reason: err?.message }, {
20        removeOnComplete: { age: 86400 }, removeOnFail: { age: 86400 }
21      });
22    }
23  });
24  return w;
25 }

```

10.7.2.2 Per-org inflight and LLM-QPS budgets (Redis gates)

Listing 22: apps/worker/src/budgets.ts

```

1 import { redis } from './queue';
2
3 // Inflight budget: allow up to limit outstanding for org
4 export async function beginInflight(orgId: string, limit = 500) {
5   const key = 'inflight:${orgId}';
6   const n = await redis.incr(key);
7   if (n === 1) await redis.expire(key, 60); // safety TTL
8   if (n > limit) { await redis.decr(key); throw new Error('inflight-budget-exceeded'); }
9   return () => redis.decr(key); // endInflight
10 }
11
12 // QPS budget (naive token bucket per org over 1s)
13 export async function checkLlmQps(orgId: string, orgLimit = 5, globalLimit = 100) {
14   const now = Math.floor(Date.now() / 1000);

```

```

15  const orgKey = `qps:${orgId}:${now}`;
16  const gKey   = `qps:global:${now}`;
17  const [o, g] = await redis
18    .multi()
19    .incr(orgKey).expire(orgKey, 2)
20    .incr(gKey).expire(gKey, 2)
21    .exec();
22
23  const orgCount = (o?.[1] as number) ?? 0;
24  const gCount   = (g?.[1] as number) ?? 0;
25  if (orgCount > orgLimit) throw new Error('llm-qps-org-exceeded');
26  if (gCount > globalLimit) throw new Error('llm-qps-global-exceeded');
27  }

```

10.7.2.3 Using budgets in a worker

Listing 23: apps/worker/src/embed.worker.ts

```

1  import { Worker } from 'bullmq';
2  import { redis } from './queue';
3  import { beginInflight, checkLlmQps } from './budgets';
4
5  export const embedWorker = new Worker('embed-chunks', async (job) => {
6    const { orgId } = job.data as { orgId: string };
7    const end = await beginInflight(orgId, 500);
8    try {
9      await checkLlmQps(orgId, 5, 100);
10     // ... call embeddings provider here ...
11     return { ok: true };
12   } finally {
13     await end();
14   }
15 }, { connection: redis, concurrency: 8 });

```

10.7.3 Kill switches

10.7.3.1 Global pause/resume endpoints (API)

Listing 24: apps/api/src/controllers/debug.queue.ts

```

1  import { Controller, Post, Param, NotFoundException } from '@nestjs/common';
2  import { queues } from '@bidion/worker/queue';
3
4  const MAP: Record<string, keyof typeof queues> = {
5    'score-matches': 'scoreMatches',
6    'embed-chunks': 'embedChunks',

```

```

7   'parse-doc': 'parseDoc',
8 };
9
10 @Controller('debug/queues')
11 export class DebugQueueController {
12   @Post('/:name/pause')
13   async pause(@Param('name') name: string) {
14     const key = MAP[name];
15     if (!key) throw new NotFoundException('unknown queue');
16     await queues[key].q.pause(true);
17     return { paused: true, queue: name };
18   }
19
20   @Post('/:name/resume')
21   async resume(@Param('name') name: string) {
22     const key = MAP[name];
23     if (!key) throw new NotFoundException('unknown queue');
24     await queues[key].q.resume();
25     return { resumed: true, queue: name };
26   }
27 }

```

10.7.3.2 Per-tenant pause (worker-side guard)

Listing 25: apps/worker/src/guards.ts

```

1 import { redis } from './queue';
2
3 export async function ensureTenantNotPaused(orgId: string) {
4   if (await redis.get('ks:org:${orgId}')) {
5     throw new Error('tenant-paused');
6   }
7 }

```

10.7.3.3 Simple circuit breaker for LLM calls (60s open)

Listing 26: apps/worker/src/circuit.ts

```

1 type State = 'closed' | 'open' | 'half';
2 let state: State = 'closed';
3 let openedAt = 0;
4
5 export async function withBreaker<T>(fn: () => Promise<T>): Promise<T> {
6   const now = Date.now();
7   if (state === 'open') {
8     if (now - openedAt < 60_000) throw new Error('breaker-open');

```

```

9     state = 'half';
10 }
11 try {
12     const v = await fn();
13     state = 'closed';
14     return v;
15 } catch (e: any) {
16     // open on 429/5xx-like errors
17     if (e?.status && (e.status === 429 || e.status >= 500)) {
18         state = 'open'; openedAt = now;
19     }
20     throw e;
21 }
22 }

```

10.7.4 Worker SLOs & autoscale triggers

10.7.4.1 Emit SLO metrics (Prometheus) and backlog gauges

Listing 27: apps/worker/src/metrics.ts

```

1 import client from 'prom-client';
2 import { Queue } from 'bullmq';
3 import { redis } from './queue';
4
5 export const registry = new client.Registry();
6 export const jobLatency = new client.Histogram({
7     name: 'bidion_job_latency_seconds',
8     help: 'Worker job latency',
9     buckets: [0.5, 1, 2, 5, 10, 30, 60],
10    labelNames: ['queue', 'name']
11 });
12 export const backlogGauge = new client.Gauge({
13     name: 'bidion_queue_backlog',
14     help: 'Queue backlog (waiting jobs)',
15     labelNames: ['queue']
16 });
17 registry.registerMetric(jobLatency);
18 registry.registerMetric(backlogGauge);
19
20 export async function scrapeBacklog(queueName: string) {
21     const q = new Queue(queueName, { connection: redis });
22     const counts = await q.getJobCounts('waiting');
23     backlogGauge.set({ queue: queueName }, counts.waiting || 0);
24 }

```

```

25
26 export function recordLatency(queue: string, name: string, ms: number) {
27   jobLatency.observe({ queue, name }, ms / 1000);
28 }

```

10.7.4.2 Autoscale decision (same thresholds as tests)

Listing 28: apps/worker/src/autoscale.ts

```

1 export function decideScale(
2   s: { backlog: number; minutesOver: number; avgWaitSec: number }
3 ): { action: 'scale-up' | 'hold' | 'scale-down'; reason: string } {
4   if ((s.backlog > 1000 && s.minutesOver >= 5) || s.avgWaitSec > 30) {
5     return { action: 'scale-up', reason: 'backlog/latency over threshold' };
6   }
7   if (s.backlog < 200 && s.avgWaitSec < 5) {
8     return { action: 'scale-down', reason: 'idle' };
9   }
10  return { action: 'hold', reason: 'within band' };
11 }

```

10.7.4.3 KEDA ScaledObject (illustrative)

Listing 29: keda-scaledobject.yaml

```

1 apiVersion: keda.sh/v1alpha1
2 kind: ScaledObject
3 metadata:
4   name: worker-embed-autoscale
5 spec:
6   scaleTargetRef:
7     name: worker-embed-deployment
8   cooldownPeriod: 60
9   minReplicaCount: 2
10  maxReplicaCount: 20
11  triggers:
12    - type: redis
13      metadata:
14        # Prefer a secret for the address; include auth/tls as needed
15        address: REDIS_HOST:6379
16        # BullMQ (lists mode) waiting list key:
17        # note: BullMQ uses "bull:<queue>:wait" for lists mode.
18        listName: bull:embed-chunks:wait
19        listLength: "1000"
20  # For Redis Streams mode, use the redis-streams scaler and the stream key instead.

```

11 API Design

11.1 API Server Bootstrap (apps/api/src/server.ts)

Listing 30: Minimal Express setup with JSON body parsing

```
1 import express from 'express';
2 import bodyParser from 'body-parser';
3 import { pool } from '../lib/db';
4 import { queues, defaultJobOpts } from '../worker-bridge';
5 import { draftAssistant } from './svc.assistant';
6
7 const app = express();
8 app.use(bodyParser.json({ limit: '10mb' }));
9
10 // ... API endpoints here ...
11
12 app.listen(3001, () => console.log('API listening on :3001'));
```

11.2 RFP Endpoints

11.2.1 POST /rfps — Create Manual-Text RFP

Listing 31: Express route: POST /rfps (manual text RFP)

```
1 app.post('/rfps', async (req, res) => {
2   const { orgId, title, bodyText } = req.body;
3   const { rows } = await pool.query(
4     'INSERT INTO documents(org_id, kind, filename, storage_url, mime)
5     VALUES ($1,'rfp',$2,$3,$4) RETURNING id',
6     [orgId, title ?? 'rfp.txt', 'data:text/plain;base64',{Buffer.from(bodyText || '')
7       toString('base64')}}', 'text/plain']
8   );
9   const documentId = rows[0].id;
10  await pool.query('INSERT INTO chunks(document_id, idx, text) VALUES ($1,0,$2)', [
11    documentId, bodyText || '']);
12  await queues.embedChunks.add('embed', { documentId, orgId, kind: 'rfp' },
13    defaultJobOpts);
14  res.json({ documentId });
15 });
```

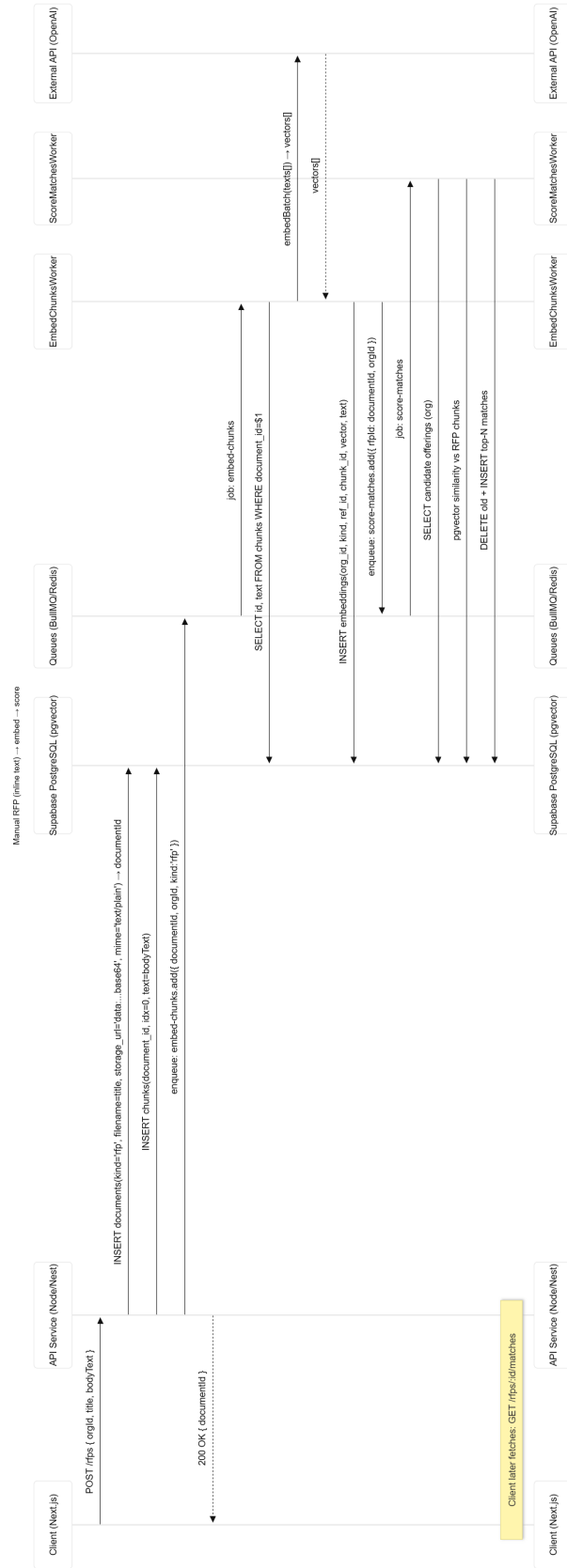


Figure 1: RFP File Upload → Parse → Embed → Score (worker-chained)

11.2.2 POST /rfps/upload — Register File-Backed RFP

Listing 32: Express route: POST /rfps/upload (file-backed RFP)

```
1 app.post('/rfps/upload', async (req,res) => {
2   const { orgId, filename, storageUrl, mime } = req.body;
3   const { rows } = await pool.query(
4     'INSERT INTO documents(org_id, kind, filename, storage_url, mime)
5     VALUES ($1,'rfp',$2,$3,$4) RETURNING id',
6     [orgId, filename, storageUrl, mime]
7   );
8   const documentId = rows[0].id;
9   await queues.parseDoc.add('parse', { documentId }, defaultJobOpts);
10  res.json({ documentId });
11  });
```

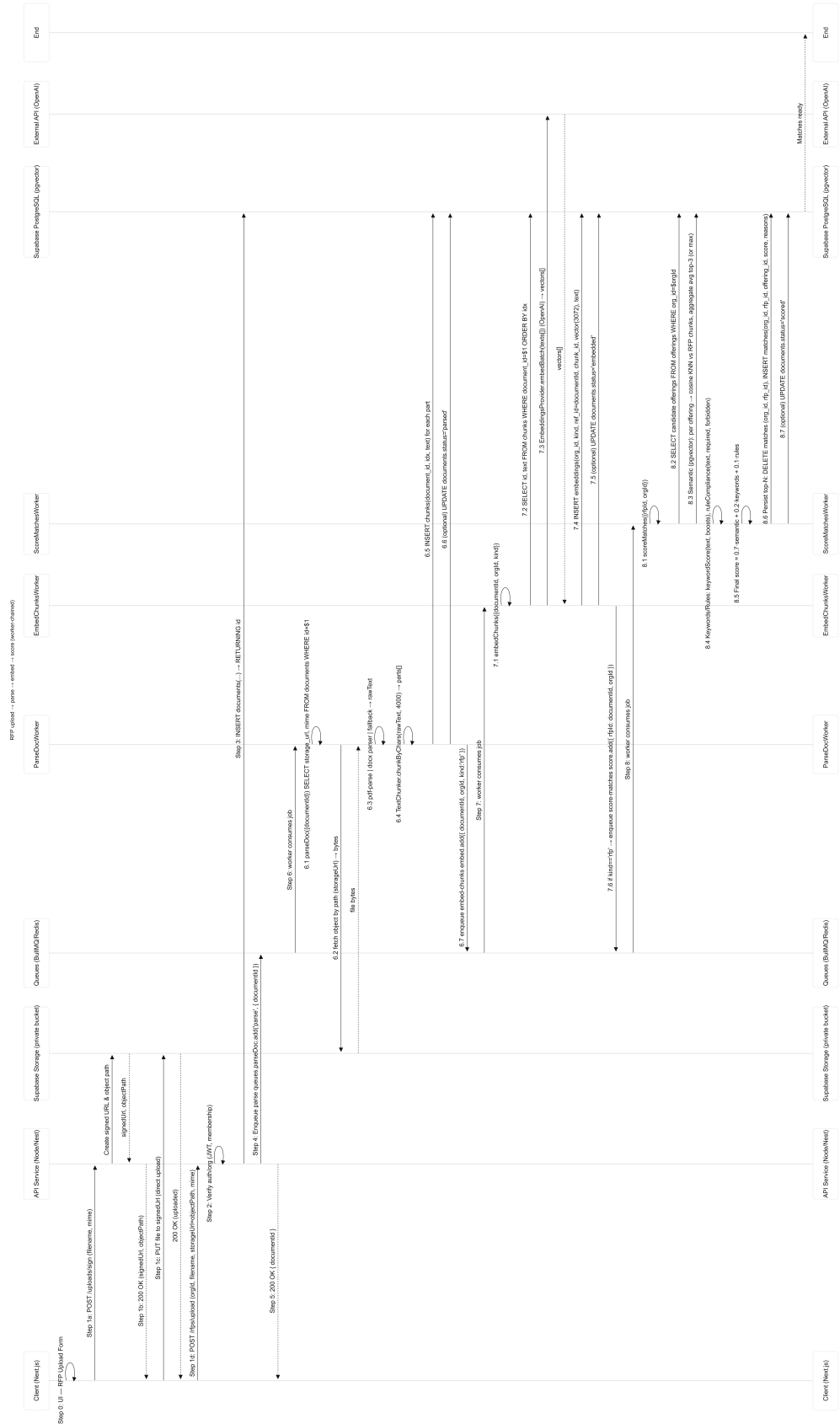


Figure 2: RFP File Upload -> Parse -> Embed -> Score (worker-chained)

11.2.3 POST /rfps/:rfpId/score — Force Re-Score

Listing 33: Express route: POST /rfps/:rfpId/score

```
1 app.post('/rfps/:rfpId/score', async (req,res) => {
2   const { orgId } = req.body;
3   await queues.scoreMatches.add('score', { rfpId: req.params.rfpId, orgId },
4     defaultJobOpts);
5   res.json({ ok: true });
6 });
```

11.2.4 GET /rfps — List RFPs

Returns the caller's RFPs for an organization with cursor pagination and optional search.

Query parameters.

- limit (optional, default 20, max 100)
- cursor (optional; opaque tuple encoded as `created_at|id`)
- q (optional; case-insensitive search over title/filename)

Listing 34: Express route: GET /rfps (cursor pagination + search)

```
1 // GET /rfps?limit=20&cursor=2025-09-01T10:00:00Z/f9b1...&q=soc2
2 app.get('/rfps', async (req, res) => {
3   // NEVER trust orgId from query; derive from auth/JWT or RLS
4   const orgId = req.user.orgId; // e.g., set by auth middleware
5
6   const limit = Math.min(parseInt(String(req.query.limit ?? '20'), 10) || 20, 100);
7   const cursor = typeof req.query.cursor === 'string' ? req.query.cursor : undefined;
8   const q      = typeof req.query.q === 'string' ? req.query.q.trim() : '';
9
10  // Parse cursor "created_at|id"
11  let cursorCreatedAt: string | null = null;
12  let cursorId: string | null = null;
13  if (cursor) {
14    const [ca, id] = cursor.split('|');
15    cursorCreatedAt = ca || null;
16    cursorId = id || null;
17  }
18
19  // Build dynamic WHERE
20  const params: any[] = [orgId];
21  let where = 'd.org_id = $1 AND d.kind = 'rfp'';
22  if (q) {
23    params.push(`%${q}%`);
```

```

24     where += ' AND (d.filename ILIKE ${params.length})';
25 }
26
27 // Cursor predicate: stable tuple (created_at DESC, id DESC)
28 let cursorSql = '';
29 if (cursorCreatedAt && cursorId) {
30     params.push(cursorCreatedAt, cursorId);
31     cursorSql = '
32         AND (d.created_at, d.id) < (${params.length-1}::timestampz, ${params.length}::
33         uuid)
34     ';
35 }
36
37 params.push(limit);
38
39 const { rows } = await pool.query(
40     '
41     SELECT
42         d.id                AS rfp_id,
43         d.filename          AS title,
44         d.created_at,
45         COALESCE(r.last_scored_at, NULL) AS last_scored_at,
46         -- lightweight stats
47         (SELECT COUNT(*)::int FROM matches m WHERE m.org_id = d.org_id AND m.rfp_id = d.id)
48         AS matches_count
49     FROM documents d
50     LEFT JOIN rfp_meta r
51         ON r.org_id = d.org_id AND r.rfp_id = d.id
52     WHERE ${where}
53         ${cursorSql}
54     ORDER BY d.created_at DESC, d.id DESC
55     LIMIT ${params.length}
56     ',
57     params
58 );
59
60 // Next cursor is last row's (created_at/id)
61 const next =
62     rows.length === limit
63     ? `${rows[rows.length - 1].created_at.toISOString()}|${rows[rows.length - 1].rfp_id}`
64     : null;
65
66 res.json({ items: rows, nextCursor: next });

```

```
65 });
```

Indexes & RLS.

Listing 35: Indexes/RLS to support GET /rfps

```
1 -- Speed up listing + cursor ordering
2 CREATE INDEX IF NOT EXISTS documents_rfp_list_idx
3   ON documents (org_id, kind, created_at DESC, id);
4
5 -- Optional metadata for freshness
6 -- rfp_meta(org_id uuid, rfp_id uuid, last_scored_at timestampz, PRIMARY KEY (org_id,
7   rfp_id))
8
9 -- Row-level security (multitenancy)
10 ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
11 CREATE POLICY org_can_read_rfps ON documents
12   FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid AND kind = 'rfp');
```

11.2.5 GET /rfps/:rfpId/matches — Fetch Ranked Matches

Listing 36: Express route: GET /rfps/:rfpId/matches

```
1 // GET /rfps/:rfpId/matches?limit=20&cursor=<offering_id>|null
2 app.get('/rfps/:rfpId/matches', async (req, res) => {
3   const auth = req.user!;           // derive orgId from auth
4   const orgId = auth.orgId;         // NOT from query
5   const limit = Math.min(Number(req.query.limit ?? 20), 100);
6   const cursor = req.query.cursor as string | undefined;
7
8   const params: any[] = [orgId, req.params.rfpId, limit];
9   const whereCursor = cursor ? 'AND (m.score, m.offering_id) < ('
10     SELECT score, offering_id FROM matches
11     WHERE org_id=$1 AND rfp_id=$2 AND offering_id=$4
12   )' : '';
13   if (cursor) params.push(cursor);
14
15   const { rows } = await pool.query(
16     '
17     SELECT m.offering_id,
18           o.title,
19           m.score,
20           m.reasons,
21           r.last_scored_at
22     FROM matches m
```

```

23     JOIN offerings o ON o.id = m.offering_id
24     JOIN rfp_meta r ON r.org_id = m.org_id AND r.rfp_id = m.rfp_id
25     WHERE m.org_id = $1 AND m.rfp_id = $2
26     ${whereCursor}
27     ORDER BY m.score DESC, m.offering_id ASC
28     LIMIT $3
29     ',
30     params
31 );
32
33 // next cursor = last offering_id of this page (stable because of tiebreaker)
34 const nextCursor = rows.length === limit ? rows[rows.length - 1].offering_id : null;
35 res.json({ items: rows, nextCursor });
36 });

```

Indexes & RLS.

Listing 37: Indexes/RLS to support GET /rfps

```

1 CREATE INDEX IF NOT EXISTS matches_rfpid_rank_idx
2   ON matches (org_id, rfp_id, score DESC, offering_id);
3
4 -- Optional: store when the worker last wrote results
5 -- rfp_meta(org_id, rfp_id, last_scored_at timestampz)
6
7 ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
8 CREATE POLICY org_can_read_matches ON matches
9   FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);

```

11.2.6 POST /rfps/:rfpId/match — Recompute Matches Now

Triggers an immediate re-score for a single RFP. Intended for admin/ops or when configuration changes (boosts/required/forbidden) require a fresh ranking.

Listing 38: Express route: POST /rfps/:rfpId/match (idempotent enqueue)

```

1 // Admin-only; orgId must come from auth (e.g., req.user.orgId)
2 app.post('/rfps/:rfpId/match', async (req, res) => {
3   const orgId = req.user.orgId; // derive from JWT/session; do NOT trust client input
4   const rfpId = req.params.rfpId;
5
6   // Optional tuning overrides (validated & bounded)
7   const k      = Math.max(1, Math.min(Number(req.body?.k ?? 3), 10)); // top-k for
   semantic pooling
8   const topN   = Math.max(1, Math.min(Number(req.body?.topN ?? 10), 100)); // number of
   matches to persist

```

```

9   const reason = String(req.body?.reason ?? 'manual');
10
11  // Existence check (and org scoping)
12  const { rows: rfpRows } = await pool.query(
13    'SELECT 1 FROM documents WHERE id=$1 AND org_id=$2 AND kind='rfp'',
14    [rfpId, orgId]
15  );
16  if (rfpRows.length === 0) {
17    return res.status(404).json({ ok:false, error:'rfp_not_found' });
18  }
19
20  // Embeddings readiness (optional but helpful)
21  const { rows: embRows } = await pool.query(
22    'SELECT 1 FROM embeddings WHERE org_id=$1 AND kind='rfp' AND ref_id=$2 LIMIT 1',
23    [orgId, rfpId]
24  );
25  if (embRows.length === 0) {
26    return res.status(409).json({ ok:false, error:'embeddings_not_ready' });
27  }
28
29  // Idempotent enqueue (one in-flight job per (org, rfp))
30  const jobId = `score:${orgId}:${rfpId}`;
31  await queues.scoreMatches.add(
32    'score',
33    { orgId, rfpId, k, topN, reason },
34    {
35      jobId, // dedupe
36      attempts: 3,
37      backoff: { type: 'exponential', delay: 2000 },
38      removeOnComplete: true,
39      removeOnFail: false
40    }
41  );
42
43  return res.status(200).json({ ok:true, enqueued:true, jobId });
44 });

```

- **Auth & RLS.** Derive `orgId` from auth and enforce RLS on `documents/embeddings/matches`.
- **Idempotency.** The stable `jobId` prevents duplicate in-flight recomputes for the same RFP.
- **Safety.** The embeddings-readiness guard avoids enqueueing useless jobs before `embed-chunks` finishes.
- **Overrides.** Optional `k` and `topN` allow controlled rescoring; defaults match the worker's configured values.

11.3 Offering Endpoints

11.3.1 POST /offerings — Create Offering and Embed, Manual-Text Only

Creates an offering (title, description, tags), materializes it as a document of kind 'offering', inserts one chunk (title + description), and enqueues embeddings. The worker will write vectors to embeddings and the offering will participate in scoring.

Request body.

- title (*string, required*)
- description (*string, required*)
- tags[] (*string[], optional*)

Listing 39: Express route: POST /offerings (create + embed enqueue)

```
1 // Admin/user auth middleware must set req.user.orgId (do not trust client-sent orgId)
2 app.post('/offerings', async (req, res) => {
3   const orgId = req.user.orgId;
4   const { title, description, tags } = req.body ?? {};
5
6   // Basic validation
7   if (typeof title !== 'string' || !title.trim()) {
8     return res.status(400).json({ ok:false, error:'invalid_title' });
9   }
10  if (typeof description !== 'string' || !description.trim()) {
11    return res.status(400).json({ ok:false, error:'invalid_description' });
12  }
13  if (tags && !Array.isArray(tags)) {
14    return res.status(400).json({ ok:false, error:'invalid_tags' });
15  }
16
17  // Create offering
18  const off = await pool.query(
19    'INSERT INTO offerings(org_id, title, description, tags)
20     VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
21     RETURNING id',
22    [orgId, title, description, tags ?? []]
23  );
24  const offeringId = off.rows[0].id;
25
26  // Create document(kind='offering') linked to offering
27  const doc = await pool.query(
28    'INSERT INTO documents(org_id, kind, ref_id, filename, mime)
29     VALUES ($1, 'offering', $2, $3, 'text/plain')
30     RETURNING id',
31    [orgId, offeringId, title]
```



```

32  );
33  const documentId = doc.rows[0].id;
34
35  // Insert a single chunk (title + description); we can split further if long
36  await pool.query(
37    'INSERT INTO chunks(document_id, idx, text)
38      VALUES ($1, 0, $2)',
39    [documentId, `${title}\n\n${description}']
40  );
41
42  // Enqueue embeddings for this offering document
43  await queues.embedChunks.add(
44    'embed',
45    { documentId, orgId, kind: 'offering' },
46    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
47    }
48  );
49
50  return res.status(201).json({ ok:true, offeringId, documentId });
51  });

```

- **Auth/RLS.** Derive orgId from auth; enable RLS on documents, chunks, embeddings, offerings.
- **Chunking.** For long descriptions, prefer the same chunker as RFPs; here we keep MVP simple with one chunk.
- **Participation in scoring.** Once embeddings are written, offerings are candidates for score-matches and will appear in GET /rfps/:rfpId/matches.

DDL & Indexing.

Listing 40: Indexes and RLS policy sketches

```

1  -- Fast lookup by org and recency
2  CREATE INDEX IF NOT EXISTS offerings_org_created_idx
3    ON offerings (org_id, created_at DESC);
4
5  -- Documents/chunks for offerings
6  CREATE INDEX IF NOT EXISTS documents_offerings_idx
7    ON documents (org_id, kind, ref_id);
8
9  ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
10 CREATE POLICY org_can_crud_offerings ON offerings
11   USING (org_id = current_setting('app.org_id', true)::uuid)
12   WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);

```

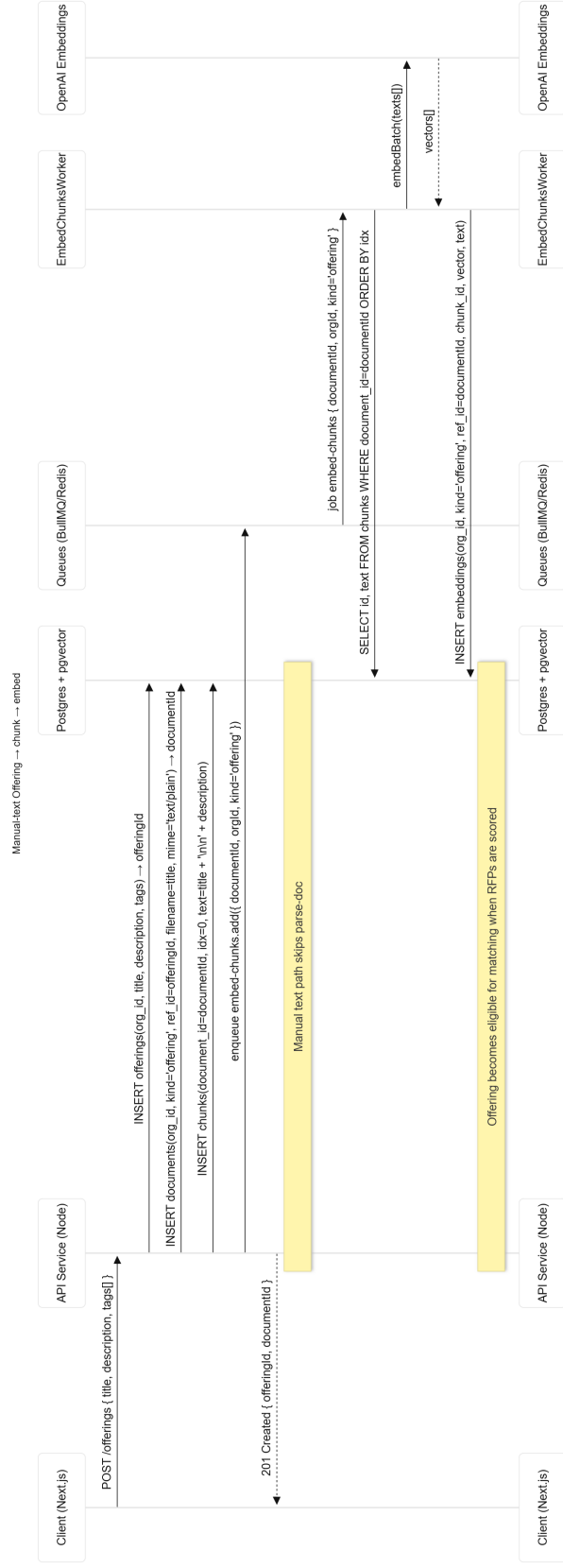


Figure 3: End-to-end sequence for `POST /offerings` (manual-text only). The API creates the offering, materializes a document and a single chunk (`title + description`), then enqueues `embed-chunks`. The worker embeds via OpenAI and persists vectors to `embeddings`. No parsing step is required; the offering becomes eligible for matching when RFPs are scored.

11.3.2 POST /offerings/upload — Register File-Backed Offering

Registers an offering whose source is a file in Storage (PDF/DOCX, etc.). The worker chain will parse → chunk → embed.

Request body.

- title (*string, required*)
- storageUrl (*string, required*) — S3/Supabase URL (client uploads via signed URL)
- mime (*string, required*) — e.g., application/pdf
- tags[] (*string[], optional*)

Listing 41: Express route: POST /offerings/upload (file-backed + parse chain)

```
1 // Auth middleware sets req.user.orgId; never trust client-sent orgId
2 app.post('/offerings/upload', async (req, res) => {
3   const orgId = req.user.orgId;
4   const { title, storageUrl, mime, tags } = req.body ?? {};
5
6   // Basic validation
7   if (typeof title !== 'string' || !title.trim()) {
8     return res.status(400).json({ ok:false, error:'invalid_title' });
9   }
10  if (typeof storageUrl !== 'string' || !storageUrl.trim()) {
11    return res.status(400).json({ ok:false, error:'invalid_storageUrl' });
12  }
13  if (typeof mime !== 'string' || !mime.trim()) {
14    return res.status(400).json({ ok:false, error:'invalid_mime' });
15  }
16  if (tags && !Array.isArray(tags)) {
17    return res.status(400).json({ ok:false, error:'invalid_tags' });
18  }
19
20  // Create offering row
21  const off = await pool.query(
22    'INSERT INTO offerings(org_id, title, description, tags)
23    VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
24    RETURNING id',
25    [orgId, title, /* description */ '', tags ?? []]
26  );
27  const offeringId = off.rows[0].id;
28
29  // Create document(kind='offering') pointing to the file
30  const doc = await pool.query(
31    'INSERT INTO documents(org_id, kind, ref_id, filename, storage_url, mime)
32    VALUES ($1, 'offering', $2, $3, $4, $5)
```

```

33     RETURNING id',
34     [orgId, offeringId, title, storageUrl, mime]
35 );
36 const documentId = doc.rows[0].id;
37
38 // Enqueue parse → (embed) → (score-later when RFPs are scored)
39 await queues.parseDoc.add(
40     'parse',
41     { documentId }, // parseDoc will enqueue embed-chunks
42     { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
43     }
44 );
45
46 return res.status(201).json({ ok:true, offeringId, documentId });
47 });

```

- **Upload flow.** Client performs direct upload to Storage (signed URL), then calls this endpoint with the resulting `storageUrl` and `mime`.
- **Worker chain.** `parseDoc` extracts text → writes `chunks` → enqueues `embed-chunks`. No manual chunk insert here.
- **Scoring.** Offerings become candidates automatically; RFP rescoring can be manual (§11.2.6) or scheduled.

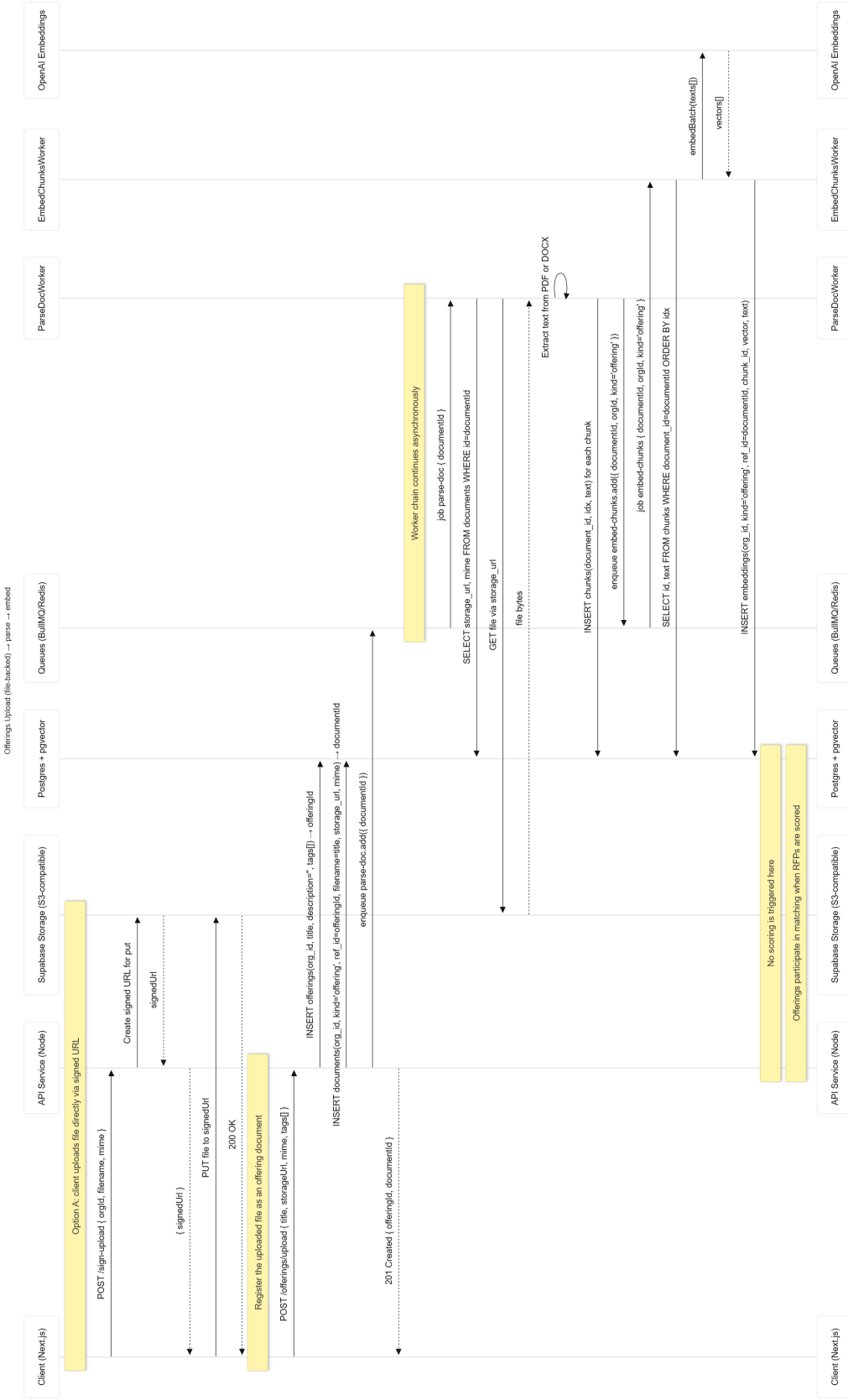


Figure 4: End-to-end sequence for POST /offerings/upload. The client performs a direct file upload via a signed URL to Storage; the API registers the offering and file, then enqueues parse-doc. The parse-doc worker extracts text and writes chunks; embed-chunks generates embeddings and persists them to embeddings. No scoring is triggered at this step; the offering becomes eligible for matching when RFPs are scored.

11.4 KB Endpoints

11.4.1 POST /kb/upload — Register File-Backed KB

Registers a KB document already uploaded to Storage (client-side signed URL flow). Enqueues parse → embed; rows are RLS-scoped to the caller's org.

Request body.

- `title` (*string, required*)
- `storageUrl` (*string, required*) — S3/Supabase URL
- `mime` (*string, required*) — e.g., `application/pdf`

Note: `orgId` is derived from auth and applied via RLS (do not accept from client).

Listing 42: Express route: POST /kb/upload

```
1 app.post('/kb/upload', requireAuth, withPgScope, async (req, res) => {
2   const { title, storageUrl, mime } = req.body ?? {};
3   if (typeof title !== 'string' || !title.trim())      return res.status(400).json({
4     error: 'invalid_title' });
5   if (typeof storageUrl !== 'string' || !storageUrl)    return res.status(400).json({
6     error: 'invalid_storageUrl' });
7   if (typeof mime !== 'string' || !mime.trim())        return res.status(400).json({
8     error: 'invalid_mime' });
9
10  // Materialize a KB document (no orgId in SQL; RLS handles tenancy)
11  const doc = await (req as any).pg.query(`
12    INSERT INTO documents(kind, filename, storage_url, mime)
13    VALUES ('kb', $1, $2, $3)
14    RETURNING id
15    `, [title, storageUrl, mime]);
16
17  const documentId = doc.rows[0].id;
18
19  // Enqueue parse -> (embed) via worker chain
20  await queues.parseDoc.add(
21    'parse',
22    { documentId },
23    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
24    });
25
26  return res.status(201).json({ ok: true, documentId });
27 });
```

Remarks.

- **Upload flow.** Use client direct upload (signed URL), then call this endpoint with the resulting `storageUrl`.
- **Worker chain.** `parse-doc` extracts text → writes `chunks` → enqueues `embed-chunks`.
- **Assistant.** KB embeddings (`kind='kb'`) are included in retrieval for `/assistant/draft`.

11.4.2 GET /kb — List KB Documents

Lists KB documents for the caller's org (RLS-scoped). Supports simple pagination and search.

Query params.

- `q` (*string, optional*) — case-insensitive substring match on `filename`.
- `limit` (*int, optional, default 20, max 100*)
- `offset` (*int, optional, default 0*)

Listing 43: Express route: GET /kb (list)

```
1 app.get('/kb', requireAuth, withPgScope, async (req, res) => {
2   const q = (req.query.q as string | undefined)?.trim();
3   const limit = Math.min(Math.max(parseInt((req.query.limit as string) || '20', 10), 1),
4     100);
5   const offset = Math.max(parseInt((req.query.offset as string) || '0', 10), 0);
6
7   const params: any[] = [];
8   let where = 'kind = 'kb'';
9   if (q) {
10     params.push(`%${q.toLowerCase()}%`);
11     where += ' AND lower(filename) LIKE ${params.length}'
12   }
13   params.push(limit, offset);
14
15   const { rows } = await (req as any).pg.query(
16     '
17     SELECT id AS document_id, filename, mime, created_at
18     FROM documents
19     WHERE ${where}
20     ORDER BY created_at DESC
21     LIMIT ${params.length - 1} OFFSET ${params.length}
22     ',
23     params
24   );
25   res.json({ items: rows, limit, offset });
26 }
```

11.5 Assistant Endpoints

11.5.1 POST /assistant/draft — RAG Draft

Purpose.

Entry point for the *drafting assistant*: a user-facing RAG generator that transforms RFP context into draft prose (answers, proposal sections, emails).

Listing 44: Express route: POST /assistant/draft (orgId derived via RLS)

```
1 // apps/api/src/server.ts
2 import { requireAuth } from './auth';
3 import { withPgScope } from './pg-scope';
4 import { draftAssistant } from './svc.assistant';
5
6 app.post('/assistant/draft', requireAuth, withPgScope, async (req, res) => {
7   // orgId is derived from JWT by requireAuth and applied to the PG session by
8   // withPgScope
9   const { rfpId, question } = req.body || {};
10  if (!rfpId || !question) {
11    return res.status(400).json({ error: 'rfpId_and_question_required' });
12  }
13
14  const out = await draftAssistant({
15    rfpId,
16    question,
17    client: (req as any).pg // request-scoped PG client with SET LOCAL app.org_id
18                          // already applied
19  });
20  res.json(out);
21 }
```

Request body.

- `rfpId` (*UUID, required*) — target RFP to ground retrieval.
- `question` (*string, required*) — drafting instruction or query.

Note: `orgId` is derived from authentication/authorization (RLS); it is not trusted from client input.

Response shape.

- `text` (*string*) — generated draft.
- `sources[]` (*string[] or objects*) — lightweight source hints (e.g., chunk identifiers).

Listing 45: Example: POST /assistant/draft (HTTP)

```
1 POST /assistant/draft
```



```

2 Content-Type: application/json
3 Authorization: Bearer <JWT>
4
5 {
6   "rfpId": "7be3a4fa-5db9-4a19-8b4b-7a7b3f9c1d10",
7   "question": "Draft the Security & Compliance section focusing on SOC 2 Type II and 24/7
8     support."
9 }

```

Listing 46: Example response

```

1 200 OK
2 {
3   "text": "We maintain SOC 2 Type II compliance ...",
4   "sources": ["ctx-0","ctx-1","ctx-2"]
5 }

```

11.6 API-Worker Bridge (apps/api/worker-bridge.ts)

Listing 47: Expose queues to API layer

```

1 import { makeQueue, defaultJobOpts } from '../worker/src/queue';
2 export const queues = {
3   parseDoc: makeQueue('parse-doc').q,
4   embedChunks: makeQueue('embed-chunks').q,
5   scoreMatches: makeQueue('score-matches').q,
6 };
7 export { defaultJobOpts };

```

12 Assistant (RAG) for Drafting Responses

This service generates proposal drafts grounded on retrieved context (RFP & KB chunks). It reuses embeddings produced elsewhere but does *not* compute or persist match scores (see §9).

Overview.

- **Context retrieval.** For the caller’s organization and target RFP, retrieve the most relevant chunks/snippets (optionally including KB).
- **Prompt construction.** Assemble a prompt using the retrieved context and the user’s question (e.g., “Draft Security & Compliance”).
- **Generation.** Call the LLM to produce a grounded draft based on the provided context.
- **Return.** Respond with the draft text and lightweight source hints for traceability.

Usage.

- Draft proposal sections (e.g., “Company Overview”, “SLA/Uptime”, “Compliance”).
- Answer specific RFP questions in the organization’s house style.
- Produce cover letters, executive summaries, or clarifications.

Behavioral remarks.

- **Read-only to scoring.** This endpoint does not alter matches or scores; it only reads context and generates text.
- **Grounding discipline.** The prompt instructs the model to answer strictly from retrieved context and to call out missing information.
- **Latency/limits.** Keep a small top- k context (e.g., $k \in [6, 12]$), deduplicate near-duplicates, and trim long chunks to control latency and token usage.
- **Auth/RLS.** Enforce tenant scoping via authentication-derived `orgId` and Postgres RLS on embeddings/documents.

12.1 Service Implementation (`apps/api/src/svc.assistant.ts`)

Listing 48: Retrieve top chunks (RFP + KB) semantically and call OpenAI

```
1 // apps/api/src/svc.assistant.ts
2 import OpenAI from 'openai';
3 import type { PoolClient } from 'pg';
4
5 const oai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });
6
7 /**
8  * NOTE:
9  * - 'orgId' is NOT accepted from the client. Tenant scoping is enforced by Postgres RLS
10  *
11  * - Use the request-scoped 'client' (middleware sets 'SET LOCAL app.org_id = ...').
12  */
13 export async function draftAssistant({
14   rfpId,
15   question,
16   client
17 }: {
18   rfpId: string;
19   question: string;
20   client: PoolClient; // request-scoped client with RLS org set
21 }) {
22   // Embed the question once
23   const emb = await oai.embeddings.create({
24     model: 'text-embedding-3-large',
```

```

24     input: question
25 });
26 const qvec = `[$${emb.data[0].embedding.join(',')}]`;
27
28 // Semantic KNN over this RFP + org KB (RLS limits rows to caller's org)
29 const { rows: hits } = await client.query(
30   '
31   WITH cand AS (
32     SELECT kind, ref_id, text, vector
33     FROM embeddings
34     WHERE (kind = 'rfp' AND ref_id = $1) OR kind = 'kb'
35   )
36   SELECT kind,
37         ref_id,
38         text,
39         1 - (vector <=> $2::vector) AS sim
40   FROM cand
41   ORDER BY vector <=> $2::vector      -- cosine distance asc
42   LIMIT 12
43   ',
44   [rfpId, qvec]
45 );
46
47 // Light dedupe + clamp to token budget
48 const seen = new Set<string>();
49 const top: Array<{ kind: string; ref_id: string; text: string; sim: number }> = [];
50 for (const h of hits) {
51   const text = (h.text || '') as string;
52   const key = text.slice(0, 160).toLowerCase(); // cheap near-dup heuristic
53   if (seen.has(key)) continue;
54   seen.add(key);
55   top.push({ kind: h.kind, ref_id: h.ref_id, text, sim: Number(h.sim) });
56   if (top.length >= 8) break;
57 }
58 const ctxTexts = top.map(h => h.text.slice(0, 1200)); // trim long chunks
59
60 // Prompt
61 const messages = [
62   {
63     role: 'system' as const,
64     content:
65       'You are a proposal assistant. Answer strictly using the provided CONTEXT. ' +
66       'If information is missing, state what is needed.'
67   },

```

```

68     {
69       role: 'user' as const,
70       content: `CONTEXT:\n${ctxTexts.join('\n--\n')}\n\nQUESTION:\n${question}`
71     }
72   ];
73
74   // Call LLM
75   const resp = await oai.chat.completions.create({
76     model: 'gpt-4o-mini',
77     messages,
78     temperature: 0.2
79   });
80
81   // Return draft + useful citations
82   return {
83     text: resp.choices[0]?.message?.content ?? '',
84     sources: top.map((h, i) => ({
85       rank: i + 1,
86       kind: h.kind,
87       refId: h.ref_id,
88       sim: h.sim
89     })))
90   };
91 }

```

13 Frontend Integration (Next.js)

This section shows minimal client flows that call the API endpoints defined in Section 11. The pages demonstrate:

- Creating a manual-text RFP and polling for matches.
- Asking the assistant to draft content grounded on retrieved chunks.

13.1 RFP Creation Page (apps/web/app/rfps/new/page.tsx)

Listing 49: Create a manual-text RFP and fetch matches

```

1 'use client';
2 import { useState } from 'react';
3
4 export default function NewRfpPage() {
5   const [title, setTitle] = useState('');
6   const [body, setBody] = useState('');
7   const [rfpDocId, setRfpDocId] = useState();

```

```

8   const orgId = '00000000-0000-0000-0000-000000000001'; // from session in real app
9
10  async function createRfp() {
11      const r = await fetch('http://localhost:3001/rfps', {
12          method: 'POST', headers: {'Content-Type': 'application/json'},
13          body: JSON.stringify({ orgId, title, bodyText: body })
14      }).then(r=>r.json());
15
16      setRfpDocId(r.documentId);
17      alert('RFP created. Parsing, embeddings, and scoring are queued automatically.');
```

```

18  }
19
20  return (
21      <div style={{maxWidth:720, margin:'2rem auto'}}>
22          <h1>New RFP</h1>
23          <input value={title} onChange={e=>setTitle(e.target.value)} placeholder="Title"
24          style={{width:'100%', padding:8}}/>
25          <textarea value={body} onChange={e=>setBody(e.target.value)} placeholder="Paste the
26          RFP body..." rows={12} style={{width:'100%', marginTop:8}}/>
27          <button onClick={createRfp} style={{marginTop:12}}>Create & Queue</button>
28          {rfpDocId && <Matches rfpId={rfpDocId} orgId={orgId} />}
29      </div>
30  );
31  }
32
33  function Matches({ rfpId, orgId }) {
34      const [items, setItems] = useState([]);
35      async function refresh() {
36          const r = await fetch('http://localhost:3001/rfps/${rfpId}/matches?orgId=${orgId}').
37              then(r=>r.json());
38          setItems(r);
39      }
40      return (
41          <div style={{marginTop:24}}>
42              <button onClick={refresh}>Refresh Matches</button>
43              <ul>
44                  {items.map(x=>(
45                      <li key={x.offering_id}>
46                          <strong>{x.title}</strong> -- score {x.score.toFixed(3)}
47                      </li>
48                  ))}
49              </ul>
50          </div>
51      );
52  }

```

49 }

13.2 Assistant Page (apps/web/app/assistant/page.tsx)

Listing 50: Ask the assistant to draft grounded content

```
1 'use client';
2 import { useState } from 'react';
3
4 export default function AssistantPage() {
5   const [question, setQ] = useState('Draft the Security & Compliance section.');
```

6 const [answer, setAns] = useState('');

7 const orgId = '00000000-0000-0000-0000-000000000001';

8 const rfpId = '...'; *// pass from navigation/state*

9

10 async function ask() {

11 const r = await fetch('http://localhost:3001/assistant/draft', {

12 method: 'POST', headers: {'Content-Type': 'application/json'},

13 body: JSON.stringify({ orgId, rfpId, question })

14 }).then(r=>r.json());

15 setAns(r.text);

16 }

17

18 return (

19 <div style={{maxWidth:720, margin:'2rem auto'}}>

20 <h1>Assistant</h1>

21 <textarea value={question} onChange={e=>setQ(e.target.value)} rows={5} style={{

width:'100%'}}/>

22 <button onClick={ask} style={{marginTop:8}}>Generate</button>

23 <pre style={{whiteSpace:'pre-wrap', background:'#fafafa', padding:12, marginTop

:12}}>{answer}</pre>

24 </div>

25);

26 }

14 Testing & Quality Assurance

This section defines the QA strategy for the Bidion MVP, covering test types/scope, a realistic test environment, stress & load testing with **k6**, and how we monitor results and handle regressions. CI gates are enforced where practical (fast feedback first, heavier suites on nightly).

Testing Types & Scope

Unit tests (fast, isolated).

- **API/Workers (Node/NestJS):** business logic, DTO validation, keyword normalizer (NFKC), rules scoring, RLS guard utilities.
- **SQL/DB:** SQL functions, views, small pgvector helpers (e.g., distance → similarity transforms).

Integration tests (service contracts).

- API ↔ Postgres (Supabase) with **RLS enabled** and `SET LOCAL app.org_id`.
- Workers ↔ Redis (BullMQ queues) ↔ Postgres (idempotent delete+insert to `matches`).
- OpenAI client stubs (record/replay or strict mocks for error/backoff).

End-to-End (targeted flows).

- RFP upload → parse → embed → score → UI fetch matches.
- Offering create → embed; Assistant draft (RAG) retrieval uses question embedding.

Non-functional.

- **Performance:** API p95, worker throughput, pgvector KNN p95.
- **Reliability:** retries/backoff, poison queue handling.
- **Security:** RLS policies, authz (org from JWT), secrets not exposed.

14.1 Tooling Options

This section lists recommended testing tools for both the Next.js frontend and the NestJS/Node backend.

14.1.1 Next.js (Frontend)

14.1.1.1 Unit & Component Tests.

- **Vitest** (or Jest): fast TS-friendly runner. Use with `@testing-library/react`.
- **React Testing Library:** test by behavior (queries like `getByRole`), not implementation.
- **MSW** (Mock Service Worker): mock HTTP/fetch at the network layer for stable tests.

14.1.1.2 E2E & Browser Automation.

- **Playwright:** cross-browser (Chromium/WebKit/Firefox), robust tracing/videos. Good for CI against Vercel previews.
- *Alternative:* **Cypress** (great DX, single-browser by default).

14.1.1.3 Visual & Accessibility.

- **Storybook**: component catalog; pair with **Chromatic** or **Percy** for visual regression.
- **axe-core** (`@axe-core/playwright` or `jest-axe`): automated a11y assertions.
- **Lighthouse CI**: performance, PWA, a11y budgets on PRs.

14.1.1.4 Static Analysis & Type Safety.

- **TypeScript** strict mode, **ESLint** (Next.js config), **Prettier**.
- **knip** or **ts-prune**: detect unused exports/dead code.
- **dependency-cruiser**: guard against forbidden imports and circular deps.

14.1.2 NestJS/Node (API & Workers)

14.1.2.1 Unit & Integration.

- **Jest** (or **Vitest**): with `@nestjs/testing` to bootstrap modules in-memory.
- **Supertest**: black-box HTTP assertions against the running Nest app.
- **Testcontainers**: real Postgres (with `pgvector`) and Redis in CI; crucial for RLS and queue tests.
- **MSW Node** or simple **nock**: mock outbound HTTP (e.g., OpenAI) deterministically.

14.1.2.2 Contract & API Schema.

- **Pact** (consumer-driven contracts) or **OpenAPI** schema checks (request/response validation in tests).

14.1.2.3 Load & Resilience.

- **k6**: scenario-driven load, thresholds wired to SLOs (p95 and error rate). Use on PR (short) and nightly (long).
- *Alternatives*: **Artillery**, **Locust**, **Gatling**, or quick **autocannon**/**oha** smoke.

14.1.2.4 Security & Policy.

- **Zod**/**class-validator**: enforce DTO invariants.
- **eslint-plugin-security** and **npm audit**/**pnpm audit**: basic hygiene.

14.1.2.5 Summary.

- **Unit:** algorithms (keyword weights, rule penalties), pure utilities (normalization).
- **Integration:** RLS policies, DB transactions, BullMQ pipelines, OpenAI error/backoff.
- **E2E:** critical user journeys (RFP upload → matches; Assistant draft).
- **k6:** capacity/SLO verification under load; pre-release and nightly.
- **Visual/a11y:** guard UX regressions without blocking core CI signal.

14.2 Test NextJS

Assumptions: the login page is at `/login`, renders labels `Email`, `Password`, the submit button `Sign In`, and a link `Forgot password?`. The page posts to `/api/login` on submit.

14.2.1 Illustrative Login Implementation (for tests to target)

`apps/web/(auth)/login/page.tsx` (illustrative).

```
1 'use client';
2 import { useState } from 'react';
3 import { useRouter } from 'next/navigation';
4
5 export default function LoginPage() {
6   const r = useRouter();
7   const [email, setEmail] = useState('jane@acme.com');
8   const [password, setPassword] = useState('');
9   const [msg, setMsg] = useState<string | null>(null);
10  const [loading, setLoading] = useState(false);
11
12  async function onSubmit(e: React.FormEvent) {
13    e.preventDefault();
14    setLoading(true);
15    setMsg('Signing you in...');
16    const res = await fetch('/api/login', {
17      method: 'POST',
18      body: JSON.stringify({ email, password }),
19      headers: { 'Content-Type': 'application/json' }
20    });
21    setLoading(false);
22    if (res.ok) {
23      setMsg('Redirecting to dashboard...');
24      r.push('/dashboard');
25    } else {
26      const { error } = await res.json();
```

```

27     setMsg(error ?? 'Login failed');
28   }
29 }
30
31 return (
32   <section style={{ maxWidth: 420, margin: '80px auto' }}>
33     <h2>Welcome back</h2>
34     <p className="muted">Sign in to continue</p>
35     <form onSubmit={onSubmit}>
36       <label>
37         Email
38         <input
39           aria-label="Email"
40           value={email}
41           onChange={e => setEmail(e.target.value)}
42         />
43       </label>
44       <label>
45         Password
46         <input
47           aria-label="Password"
48           type="password"
49           value={password}
50           onChange={e => setPassword(e.target.value)}
51         />
52       </label>
53       <button type="submit" aria-label="Sign In" disabled={loading}>
54         Sign In
55       </button>
56     </form>
57     {msg && <div role="status">{msg}</div>}
58     <p><a href="#">Forgot password?</a></p>
59   </section>
60 );
61 }

```

14.2.2 Unit & Component Tests.

14.2.2.1 Vitest + React Testing Library

Install.

```

1 npm add -D vitest @testing-library/react @testing-library/jest-dom jsdom @types/jest
  user-event

```

vitest.config.ts.

```
1 import { defineConfig } from 'vitest/config';
2
3 export default defineConfig({
4   test: {
5     environment: 'jsdom',
6     setupFiles: ['./vitest.setup.ts'],
7     globals: true,
8     css: true,
9     coverage: { reporter: ['text', 'lcov'] }
10  }
11 });
```

vitest.setup.ts.

```
1 import '@testing-library/jest-dom';
```

app/(auth)/login/page.spec.tsx.

```
1 import { describe, it, expect } from 'vitest';
2 import { render, screen } from '@testing-library/react';
3 import userEvent from '@testing-library/user-event';
4 import LoginPage from './page';
5
6 describe('Login page', () => {
7   it('renders fields, actions, and defaults', () => {
8     render(<LoginPage />);
9     expect(screen.getByRole('heading', { name: /welcome back/i })).toBeInTheDocument();
10    expect(screen.getByText(/sign in to continue/i)).toBeInTheDocument();
11
12    const email = screen.getByLabelText(/email/i);
13    const password = screen.getByLabelText(/password/i);
14    const submit = screen.getByRole('button', { name: /sign in/i });
15    const forgot = screen.getByRole('link', { name: /forgot password\?/i });
16
17    expect(email).toBeInTheDocument();
18    expect(password).toBeInTheDocument();
19    expect(submit).toBeEnabled();
20    expect(forgot).toHaveAttribute('href', '#');
21  });
22
23  it('accepts input and shows a submitting state', async () => {
24    const user = userEvent.setup();
25    render(<LoginPage />);
26    await user.clear(screen.getByLabelText(/email/i));
```

```

27     await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
28     await user.clear(screen.getByLabelText(/password/i));
29     await user.type(screen.getByLabelText(/password/i), 's3cret!');
30     await user.click(screen.getByRole('button', { name: /sign in/i }));
31     expect(await screen.findByRole('status')).toHaveTextContent(/signing you in/i);
32   });
33 });

```

14.2.2.2 MSW (Mock Service Worker)

tests/msw/handlers.ts.

```

1 import { http, HttpResponse } from 'msw';
2
3 export const handlers = [
4   http.post('/api/login', async ({ request }) => {
5     const body = await request.json() as { email: string; password: string };
6     if (body.email === 'dev@bidion.io' && body.password === 's3cret!') {
7       return HttpResponse.json({ token: 'jwt-123', user: { email: body.email } }, {
8         status: 200 });
9     }
10    return HttpResponse.json({ error: 'Invalid credentials' }, { status: 401 });
11  }],

```

tests/msw/server.ts.

```

1 import { setupServer } from 'msw/node';
2 import { handlers } from './handlers';
3 export const server = setupServer(...handlers);

```

Wire MSW in vitest.setup.ts.

```

1 import '@testing-library/jest-dom';
2 import { server } from './tests/msw/server';
3
4 beforeAll(() => server.listen({ onUnhandledRequest: 'error' }));
5 afterEach(() => server.resetHandlers());
6 afterAll(() => server.close());

```

page.msw.spec.tsx.

```

1 import { describe, it, expect } from 'vitest';
2 import { render, screen } from '@testing-library/react';
3 import userEvent from '@testing-library/user-event';
4 import LoginPage from './page';

```

```

5
6 describe('Login with MSW', () => {
7   it('redirects to dashboard on success', async () => {
8     const user = userEvent.setup();
9     render(<LoginPage />);
10    await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
11    await user.type(screen.getByLabelText(/password/i), 's3cret!');
12    await user.click(screen.getByRole('button', { name: /sign in/i }));
13    expect(await screen.findByText(/redirecting to dashboard/i)).toBeInTheDocument();
14  });
15
16  it('shows error on invalid credentials', async () => {
17    const user = userEvent.setup();
18    render(<LoginPage />);
19    await user.type(screen.getByLabelText(/email/i), 'wrong@user.io');
20    await user.type(screen.getByLabelText(/password/i), 'nope');
21    await user.click(screen.getByRole('button', { name: /sign in/i }));
22    expect(await screen.findByRole('alert')).toHaveTextContent(/invalid credentials/i);
23  });
24 });

```

14.2.3 E2E & Browser Automation.

14.2.3.1 Playwright (E2E)

playwright.config.ts (optional).

```

1 import { defineConfig, devices } from '@playwright/test';
2 export default defineConfig({
3   use: { baseURL: process.env.PLAYWRIGHT_BASE_URL ?? 'http://localhost:3000' },
4   projects: [{ name: 'chromium', use: { ...devices['Desktop Chrome'] } }]
5 });

```

e2e/login.spec.ts.

```

1 import { test, expect } from '@playwright/test';
2
3 test('login success routes to dashboard', async ({ page }) => {
4   await page.goto('/login');
5   await expect(page.getByRole('heading', { name: /welcome back/i })).toBeVisible();
6   await page.getByLabel('Email').fill('dev@bidion.io');
7   await page.getByLabel('Password').fill('s3cret!');
8   await page.getByRole('button', { name: /sign in/i }).click();
9   await expect(page).toHaveURL(/\/dashboard$/);
10  await expect(page.getByRole('heading', { name: /dashboard/i })).toBeVisible();
11 });

```

```

12
13 test('login failure shows error message', async ({ page }) => {
14   await page.goto('/login');
15   await page.getByLabel('Email').fill('wrong@user.io');
16   await page.getByLabel('Password').fill('nope');
17   await page.getByRole('button', { name: /sign in/i }).click();
18   await expect(page.getByRole('alert')).toHaveText(/invalid credentials/i);
19 });

```

14.2.3.2 Cypress (Alternative E2E)

cypress/e2e/login.cy.ts.

```

1 describe('Login', () => {
2   it('logs in successfully', () => {
3     cy.visit('/login');
4     cy.findByRole('heading', { name: /welcome back/i }).should('be.visible');
5     cy.findByLabelText(/email/i).clear().type('dev@bidion.io');
6     cy.findByLabelText(/password/i).clear().type('s3cret!');
7     cy.findByRole('button', { name: /sign in/i }).click();
8     cy.url().should('match', /\s/dashboard$/);
9     cy.findByRole('heading', { name: /dashboard/i }).should('be.visible');
10  });
11
12  it('shows error on bad creds', () => {
13    cy.visit('/login');
14    cy.findByLabelText(/email/i).type('wrong@user.io');
15    cy.findByLabelText(/password/i).type('nope');
16    cy.findByRole('button', { name: /sign in/i }).click();
17    cy.findByRole('alert').should('contain.text', 'Invalid credentials');
18  });
19 });

```

14.2.4 Visual & Accessibility.

14.2.4.1 Storybook (Visual Catalog)

app/(auth)/login/LoginPage.stories.tsx.

```

1 import type { Meta, StoryObj } from '@storybook/react';
2 import LoginPage from './page';
3
4 const meta: Meta<typeof LoginPage> = {
5   title: 'Auth/LoginPage',
6   component: LoginPage,
7   parameters: { layout: 'centered' }

```

```

8 };
9 export default meta;
10
11 export const Default: StoryObj<typeof LoginPage> = {};
12 export const ErrorState: StoryObj<typeof LoginPage> = {
13   args: { initialError: 'Invalid credentials' }
14 };

```

14.2.4.2 Accessibility

Option A: jest-axe (component-level).

```

1 import { render } from '@testing-library/react';
2 import { axe, toHaveNoViolations } from 'jest-axe';
3 import LoginPage from './page';
4
5 expect.extend(toHaveNoViolations);
6
7 it('has no obvious a11y violations', async () => {
8   const { container } = render(<LoginPage />);
9   const results = await axe(container);
10  expect(results).toHaveNoViolations();
11 });

```

Option B: @axe-core/playwright (E2E-level).

```

1 import { test, expect } from '@playwright/test';
2 import AxeBuilder from '@axe-core/playwright';
3
4 test('login page accessibility', async ({ page }) => {
5   await page.goto('/login');
6   const results = await new AxeBuilder({ page }).analyze();
7   expect(results.violations).toEqual([]);
8 });

```

14.2.4.3 Lighthouse CI (Perf/A11y Budgets)

lighthouseci.json.

```

1 {
2   "ci": {
3     "collect": {
4       "numberOfRuns": 2,
5       "url": ["http://localhost:3000/login"]
6     },
7     "assert": {
8       "assertions": {

```

```

9       "categories:performance": ["warn", { "minScore": 0.8 }],
10      "categories:accessibility": ["error", { "minScore": 0.9 }]
11    }
12  },
13  "upload": { "target": "temporary-public-storage" }
14 }
15 }

```

Run.

```

1 npx lhci autorun

```

14.2.5 Static Analysis & Type Safety

14.2.5.1 TypeScript strict.

```

1 {
2   "compilerOptions": {
3     "strict": true,
4     "noUncheckedIndexedAccess": true,
5     "noImplicitOverride": true
6   }
7 }

```

14.2.5.2 ESLint (Next.js base).

```

1 module.exports = {
2   extends: ['next/core-web-vitals'],
3   rules: {
4     '@next/next/no-img-element': 'off'
5   }
6 };

```

14.2.5.3 Dead code scan (knip).

```

1 {
2   "scripts": {
3     "scan:dead": "knip --reporter compact"
4   },
5   "devDependencies": { "knip": "^5.0.0" }
6 }

```

14.2.5.4 Import rules and cycles (dependency-cruiser).


```

1  /** @type {import('dependency-cruiser').IConfiguration} */
2  module.exports = {
3    forbidden: [
4      { name: 'no-cycles', severity: 'warn', from: {}, to: { circular: true } },
5      { name: 'no-test-to-src', from: { path: '^tests?/' }, to: { pathNot: '^tests?/' } }
6    ],
7    options: { doNotFollow: { path: 'node_modules' } }
8  };

```

14.3 Test NestJS/Node (API & Workers)

Assumptions.

- Monorepo layout with `apps/api` (NestJS HTTP) and `apps/worker` (BullMQ workers).
- Postgres (with `pgvector`) and Redis are required for most integration tests.
- OpenAI calls should be mocked in tests (no network).

Environment.

- Production-like defaults with small footprints; tests run against isolated containers/schemas.
- Idempotent setup/teardown: spin up containers for Postgres and Redis per test session; migrate/seed deterministically.
- Enforce tenant isolation in tests: set `SET LOCAL app.org_id = 'org_test'` within DB interactions to exercise RLS (see snippet below).

Local dev smoke: Docker Compose (use locally; CI prefers Testcontainers).

Listing 51: `docker-compose.test.yml`

```

1  # docker-compose.test.yml
2  version: "3.9"
3  services:
4    db:
5      image: supabase/postgres:15
6      environment:
7        POSTGRES_PASSWORD: dev
8      ports: ["5432:5432"]
9    redis:
10     image: redis:7
11     ports: ["6379:6379"]
12    api:
13     build: ./apps/api
14     environment:

```

```

15     DATABASE_URL: postgres://postgres:dev@db:5432/postgres
16     REDIS_URL: redis://redis:6379
17     NODE_ENV: test
18     depends_on: [db, redis]
19     command: ["npm", "run", "test:integration"]

```

Database schema & minimal seed for tests.

Listing 52: test/schema.sql

```

1  -- test/schema.sql (run via migration tool for test)
2  CREATE SCHEMA IF NOT EXISTS test;
3  -- Enable extensions used by MVP:
4  CREATE EXTENSION IF NOT EXISTS vector;
5
6  -- RLS sample toggle (ensure we test with RLS ON):
7  ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
8  -- ... policies reference current_setting('app.org_id', true);
9
10 -- Fast seed (org, 1 RFP, 2 offerings):
11 -- (Use transactions; keep deterministic IDs for tests)

```

Org-scope flag for DB operations.

Listing 53: Set org scope in tests

```

1  # Ensure test runs under an org scope when touching DB:
2  psql "$DATABASE_URL" -c "SET LOCAL app.org_id = 'org_test';"

```

14.3.1 Unit & Integration.

14.3.1.1 Shared Test Infrastructure

Testcontainers (real Postgres+Redis) for RLS/BullMQ.

Listing 54: apps/api/test/tc.setup.ts (Jest globalSetup)

```

1  import { GenericContainer, StartedTestContainer } from 'testcontainers';
2  import { PostgreSQLContainer, StartedPostgreSQLContainer } from '@testcontainers/
   postgresql';
3
4  declare global {
5    // eslint-disable-next-line no-var
6    var __PG__: StartedPostgreSQLContainer;
7    // eslint-disable-next-line no-var
8    var __REDIS__: StartedTestContainer;
9  }

```

```

10
11 export default async function globalSetup() {
12   const pg = await new PostgreSQLContainer('postgres:15')
13     .withEnvironment({ POSTGRES_PASSWORD: 'dev' }) // user/db default to 'postgres'
14     .start();
15
16   // Create pgvector + ensure schema will be applied by migrations later
17   const { Client } = await import('pg');
18   const client = new Client({ connectionString: pg.getConnectionUri() });
19   await client.connect();
20   await client.query('CREATE EXTENSION IF NOT EXISTS vector;');
21   await client.end();
22
23   const redis = await new GenericContainer('redis:7')
24     .withExposedPorts(6379)
25     .start();
26
27   process.env.DATABASE_URL = pg.getConnectionUri();
28   process.env.REDIS_URL = `redis://127.0.0.1:${redis.getMappedPort(6379)}';
29
30   // Expose on global for teardown
31   // @ts-ignore
32   global.__PG__ = pg;
33   // @ts-ignore
34   global.__REDIS__ = redis;
35 }

```

Listing 55: apps/api/test/tc.teardown.ts (Jest globalTeardown)

```

1 export default async function globalTeardown() {
2   // @ts-ignore
3   const pg = global.__PG__;
4   // @ts-ignore
5   const redis = global.__REDIS__;
6   if (pg) await pg.stop();
7   if (redis) await redis.stop();
8 }

```

Listing 56: apps/api/jest.config.ts (hook setup/teardown)

```

1 import type { Config } from 'jest';
2
3 const config: Config = {
4   preset: 'ts-jest',
5   testEnvironment: 'node',
6   globalSetup: '<rootDir>/test/tc.setup.ts',

```

```

7   globalTeardown: '<rootDir>/test/tc.teardown.ts',
8   setupFilesAfterEnv: ['<rootDir>/test/setup.env.ts'],
9   // If rely on migrations, ensure they run before tests,
10  // e.g., in a custom setup file or as part of AppModule bootstrap.
11 };
12
13 export default config;

```

Listing 57: apps/api/test/setup.env.ts

```

1 process.env.NODE_ENV = 'test';
2 process.env.APP_JWT_PUBLIC_KEY = 'TEST_KEY';
3 // Add other environment variables our app reads here

```

14.3.1.2 API Tests

Jest + @nestjs/testing (Unit/Integration).

Listing 58: apps/api/test/rfps.e2e-lite.spec.ts

```

1 import { Test } from '@nestjs/testing';
2 import { INestApplication, ValidationPipe } from '@nestjs/common';
3 import request from 'supertest';
4 import { AppModule } from '../src/app.module';
5
6 describe('RFP Matches (e2e-lite)', () => {
7   let app: INestApplication;
8
9   beforeAll(async () => {
10     const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
11     app = mod.createNestApplication();
12     app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
13     await app.init();
14   });
15
16   afterAll(async () => { await app.close(); });
17
18   it('GET /rfps/:id/matches returns ranked list', async () => {
19     const rfpId = '00000000-0000-0000-0000-000000000001';
20     await request(app.getHttpServer())
21       .get(`/rfps/${rfpId}/matches`)
22       .set('Authorization', 'Bearer TEST_JWT')
23       .expect(200)
24       .expect(res => {
25         expect(Array.isArray(res.body)).toBe(true);
26         for (let i = 1; i < res.body.length; i++) {
27           expect(res.body[i - 1].score).toBeGreaterThanOrEqual(res.body[i].score);

```

```

28     }
29     });
30 });
31 });

```

Supertest (black-box HTTP) + class-validator DTO checks.

Listing 59: apps/api/test/rfps.dto.spec.ts

```

1 import { Test } from '@nestjs/testing';
2 import { INestApplication, ValidationPipe } from '@nestjs/common';
3 import request from 'supertest';
4 import { AppModule } from '../src/app.module';
5
6 describe('RFP Create DTO validation', () => {
7   let app: INestApplication;
8
9   beforeAll(async () => {
10     const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
11     app = mod.createNestApplication();
12     app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true }));
13     await app.init();
14   });
15
16   afterAll(async () => { await app.close(); });
17
18   it('rejects invalid body', async () => {
19     await request(app.getHttpServer())
20       .post('/rfps')
21       .set('Authorization', 'Bearer TEST_JWT')
22       .send({ badField: 'nope' })
23       .expect(400);
24   });
25 });

```

14.3.1.3 Worker Tests

Worker Queue Test (BullMQ Roundtrip) with Redis.

Listing 60: apps/worker/test/queue.roundtrip.spec.ts

```

1 import { Queue, Worker, Job } from 'bullmq';
2 import IORedis from 'ioredis';
3
4 describe('BullMQ roundtrip', () => {
5   const connection = new IORedis(process.env.REDIS_URL as string);

```

```

6   const qName = 'test-q';
7   let queue: Queue;
8   let worker: Worker;
9   let results: any[] = [];
10
11  beforeAll(async () => {
12    queue = new Queue(qName, { connection });
13    // Worker that echoes payload (stub of real work)
14    worker = new Worker(qName, async (job: Job) => {
15      // simulate some work
16      return { received: job.data };
17    }, { connection });
18
19    worker.on('completed', (_, res) => results.push(res));
20  });
21
22  afterAll(async () => {
23    await queue.drain(true);
24    await worker.close();
25    await queue.close();
26    await connection.quit();
27  });
28
29  it('processes embed-chunks job', async () => {
30    await queue.add('embed-chunks', { documentId: 'doc-1', orgId: 'org-1' });
31    // naive wait for worker to run in test, better: event-based gating
32    await new Promise(res => setTimeout(res, 500));
33    expect(results[0]).toEqual({ received: { documentId: 'doc-1', orgId: 'org-1' } });
34  });
35 });

```

Mock outbound HTTP (OpenAI) withnock.

Listing 61: apps/worker/test/openai.mock.spec.ts

```

1  import nock from 'nock';
2  import { embedBatch } from '../src/embeddings'; // adapter that calls OpenAI REST
3
4  describe('OpenAI embeddings adapter', () => {
5    beforeAll(() => {
6      nock.disableNetConnect(); // block real network
7      nock('https://api.openai.com')
8        .post('/v1/embeddings')
9        .reply(200, {
10          data: [{ embedding: Array(3072).fill(0.01), index: 0 }],
11          model: 'text-embedding-3-large',

```

```

12         object: 'list'
13     });
14 });
15
16 afterAll(() => {
17    nock.cleanAll();
18     nock.enableNetConnect();
19 });
20
21 it('returns embeddings from mocked OpenAI', async () => {
22     const vecs = await embedBatch(['hello world']);
23     expect(vecs).toHaveLength(1);
24     expect(vecs[0]).toHaveLength(3072);
25 });
26 });

```

14.3.2 Contract & API Schema.

14.3.2.1 OpenAPI Schema Checks (jest-openapi).

Listing 62: Install jest-openapi

```

1 pnpm add -D jest-openapi supertest

```

Listing 63: apps/api/test/openapi.spec.ts

```

1 import { Test } from '@nestjs/testing';
2 import { INestApplication } from '@nestjs/common';
3 import request from 'supertest';
4 import { AppModule } from '../src/app.module';
5 import 'jest-openapi';
6
7 describe('OpenAPI contract', () => {
8     let app: INestApplication;
9
10    beforeAll(async () => {
11        const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
12        app = mod.createNestApplication();
13        await app.init();
14
15        // Load the generated swagger JSON (exported at build or served in dev)
16        const openapi = require('../openapi.json'); // ensure generated before test
17        expect(openapi).toBeDefined();
18        expect.extend({ toSatisfyApiSpec: (global as any).toSatisfyApiSpec });
19        (global as any).jestOpenAPI(openapi);

```

```

20   });
21
22   afterAll(async () => { await app.close(); });
23
24   it('GET /rfps satisfies OpenAPI spec', async () => {
25     const res = await request(app.getHttpServer())
26       .get('/rfps')
27       .set('Authorization', 'Bearer TEST_JWT')
28       .expect(200);
29
30     expect(res.body).toSatisfyApiSpec();
31   });
32 });

```

14.3.2.2 Pact (consumer-driven contract) example.

Listing 64: contracts/consumer.rfps.pact.spec.ts

```

1  import path from 'path';
2  import { Pact } from '@pact-foundation/pact';
3  import fetch from 'node-fetch';
4
5  describe('Consumer pact for /rfps', () => {
6    const provider = new Pact({
7      consumer: 'BidionWeb',
8      provider: 'BidionAPI',
9      dir: path.resolve(process.cwd(), 'pacts'),
10     logLevel: 'warn',
11   });
12
13   beforeAll(() => provider.setup());
14   afterAll(() => provider.finalize());
15
16   it('lists RFPs', async () => {
17     await provider.addInteraction({
18       state: 'there are RFPs',
19       uponReceiving: 'a request for list of RFPs',
20       withRequest: { method: 'GET', path: '/rfps', headers: { Authorization: 'Bearer TEST_JWT' } },
21       willRespondWith: {
22         status: 200,
23         headers: { 'Content-Type': 'application/json; charset=utf-8' },
24         body: [{ id: 'rfp-1', title: 'Sample RFP' }]
25       }
26     });
27

```



```

28     const res = await fetch(`${provider.mockService.baseUrl}/rfps`, {
29       headers: { Authorization: 'Bearer TEST_JWT' }
30     });
31     const json = await res.json();
32     expect(json[0]).toBeDefined();
33     expect(json[0].id).toBe('rfp-1');
34
35     await provider.verify();
36   });
37 });

```

14.3.3 Load & Resilience.

14.3.3.1 API Stress & Load Testing Plan (k6 SLO thresholds).

This plan is consistent with our SLOs (API p95 \leq 500 ms; KNN p95 \leq 200 ms nominal) and tags requests so thresholds apply only to API traffic.

k6 script (scenarios + thresholds).

```

1  /**
2   * Run:
3   *   k6 run -e BASE_URL=http://localhost:3000 scripts/k6/api-load.js \
4   *     --summary-export=./reports/k6-summary.json
5   *
6   * Note: Do NOT pass --vus/--duration when using scenarios (they will be ignored).
7   */
8  import http from 'k6/http';
9  import { check, sleep } from 'k6';
10 import { Trend, Rate } from 'k6/metrics';
11
12 const BASE_URL = __ENV.BASE_URL || 'http://localhost:3000';
13 const t_api = new Trend('api_latency'); // custom metric for clarity
14 const r_errors = new Rate('api_errors'); // 5xx rate
15
16 export const options = {
17   thresholds: {
18     'http_req_failed': ['rate<0.02'], // <2% request failures
19     'http_req_duration{kind:api}': ['p(95)<500'], // API p95 < 500ms
20     'api_latency': ['p(95)<500'],
21     'api_errors': ['rate<0.02'],
22   },
23   scenarios: {
24     readMatches: {
25       executor: 'ramping-vus',
26       startVUs: 1,

```

```

27     stages: [
28         { duration: '1m', target: 20 },
29         { duration: '3m', target: 50 },
30         { duration: '1m', target: 0 },
31     ],
32     exec: 'getMatches',
33     tags: { kind: 'api' }, // tag all req metrics in this scenario
34 },
35 draftAssistant: {
36     executor: 'constant-arrival-rate',
37     rate: 20,
38     timeUnit: '1s',
39     duration: '3m',
40     preAllocatedVUs: 50,
41     maxVUs: 100,
42     exec: 'postDraft',
43     tags: { kind: 'api' },
44 },
45 },
46 };
47
48 export function getMatches() {
49     const r = http.get(`${BASE_URL}/rfps/00000000-0000-0000-0000-000000000001/matches`, {
50         headers: { Authorization: 'Bearer TEST_JWT' },
51     });
52     t_api.add(r.timings.duration, { kind: 'api' });
53     r_errors.add(r.status >= 500);
54     check(r, { '200 OK': (res) => res.status === 200 });
55     sleep(0.3);
56 }
57
58 export function postDraft() {
59     const payload = JSON.stringify({ question: 'Draft Security & Compliance' });
60     const r = http.post(`${BASE_URL}/assistant/draft`, payload, {
61         headers: {
62             'Content-Type': 'application/json',
63             Authorization: 'Bearer TEST_JWT',
64         },
65     });
66     t_api.add(r.timings.duration, { kind: 'api' });
67     r_errors.add(r.status >= 500);
68     check(r, { '201/200': (res) => res.status === 200 || res.status === 201 });
69     sleep(0.5);
70 }

```

CI wiring (GitHub Actions) to run k6 on PR.

Listing 65: .github/workflows/perf-k6.yml

```
1 name: k6-perf
2 on:
3   pull_request:
4     paths: ['apps/api/**', 'apps/worker/**', 'scripts/k6/**']
5 jobs:
6   k6:
7     runs-on: ubuntu-latest
8     services:
9       db:
10        image: postgres:15
11        ports: ['5432:5432']
12        env:
13          POSTGRES_PASSWORD: dev
14          POSTGRES_USER: postgres
15          POSTGRES_DB: app
16        options: >-
17          --health-cmd="pg_isready -U postgres"
18          --health-interval=5s --health-timeout=5s --health-retries=10
19      redis:
20        image: redis:7
21        ports: ['6379:6379']
22        options: >-
23          --health-cmd="redis-cli ping || exit 1"
24          --health-interval=5s --health-timeout=5s --health-retries=10
25    steps:
26      - uses: actions/checkout@v4
27
28      - name: Setup PNPM & install
29        uses: pnpm/action-setup@v3
30        with: { version: 9 }
31      - run: pnpm i
32
33      - name: Generate DB schema (migrations)
34        env:
35          DATABASE_URL: postgresql://postgres:dev@localhost:5432/app
36        run: |
37          pnpm -C apps/api db:migrate
38
39      - name: Boot API (and optionally workers)
40        env:
41          DATABASE_URL: postgresql://postgres:dev@localhost:5432/app
```

```

42     REDIS_URL: redis://localhost:6379
43     NODE_ENV: test
44     run: |
45         pnpm -C apps/api start:test &
46         # Optional: start workers if API endpoints enqueue jobs during test
47         # pnpm -C apps/worker start:test &
48         sleep 8
49
50     - uses: grafana/setup-k6-action@v1
51
52     - name: Run k6
53       env:
54         BASE_URL: http://localhost:3000
55       run: |
56         mkdir -p reports
57         k6 run -e BASE_URL=${BASE_URL} \
58             --summary-export=./reports/k6-summary.json \
59             scripts/k6/api-load.js
60
61     - name: Upload k6 summary
62       uses: actions/upload-artifact@v4
63       with:
64         name: k6-summary
65         path: reports/k6-summary.json

```

14.3.3.2 Worker(BullMQ): Throughput Probe + Idempotency Tests

Throughput probe (smoke).

```

1  # Produce N jobs and measure end-to-end time-to-complete (queue -> done)
2  # Requires a debug enqueue endpoint that accepts: { count, kind }
3  # Example probes: embed-chunks / score-matches / parse-doc
4  BASE_URL=${BASE_URL:-http://localhost:3000}
5
6  curl -H "Authorization: Bearer TEST_JWT" \
7      -H "Content-Type: application/json" \
8      -d '{"count": 500, "kind": "embed-chunks"}' \
9      "${BASE_URL}/debug/enqueue"

```

Controls summary (targets).

| Control | MVP Target | Notes / Basis |
|----------------------|--|--|
| Delivery semantics | At-least-once | Retries/backoff + idempotent handlers; validated in tests below. |
| Retry/backoff policy | exp. 250 ms (max 5 tries) | Exponential backoff; DLQ on exhaustion. |
| Idempotency keys | (org_id, rfp_id) | jobId=org:rfp de-dup; handler UPSERTs matches. |
| Job retention | 24 h | removeOnComplete/removeOnFail with {age: 86400}. |
| Drain on shutdown | Graceful \leq 30 s | pause(true) & worker.close() on SIGTERM; PDBs. |
| Budgets | Inflight \leq 500/org; QPS 5/org, 100/global | Redis gates; covered by tests below. |
| Kill switches | Global & per-tenant | API pause/resume; worker guard (ks:org:{id}). |
| Circuit breaker | 60 s open on 429/5xx | Half-open then close on success. |
| Metrics/autoscale | Prometheus + KEDA | Histograms/gauges; scale-up on backlog/latency thresholds. |

Prerequisites (test environment)

1. Node 18+, Postgres (with RLS policies installed), Redis (empty instance for isolation).
2. Environment: DATABASE_URL, REDIS_URL, OPENAI_API_KEY (mock embedBatch if avoiding external calls).
3. Seed minimal rows:

```
1 BEGIN;
2 SELECT set_config('app.org_id', 'org_test', true);
3 INSERT INTO documents (id, org_id, kind, storage_url, mime)
4 VALUES ('rfp-1', 'org_test', 'rfp', 'https://example.invalid/test.pdf', 'application
    /pdf')
5 ON CONFLICT DO NOTHING;
6 COMMIT;
```

4. Ensure Redis is clean: FLUSHALL before each test run.

Test runner setup (Jest).

Place the suites under `apps/worker/tests`. Run:

Listing 66: `apps/worker/tests`

```
1 pnpm jest apps/worker/tests --runInBand
```

Common test helpers

Listing 67: `apps/worker/tests/helpers.ts`

```
1 import { Client as Pg } from 'pg';
2 import IORedis from 'ioredis';
3 import { queues } from '../../src/queue';
4 import { enqueueReScore } from '../../src/score.enqueue';
5
6 export const ORG = 'org_test';
7 export const RFP = 'rfp-1';
8
9 export function pgClient() {
10   const pg = new Pg({ connectionString: process.env.DATABASE_URL! });
11   return pg;
12 }
13
14 export async function resetRedis() {
15   const r = new IORedis(process.env.REDIS_URL!);
16   await r.flushall();
17   await r.quit();
18 }
19
20 export async function countMatches(pg: Pg, orgId: string, rfpId: string) {
21   const { rows } = await pg.query(
22     'SELECT count(*)::int AS n FROM matches WHERE org_id=$1 AND rfp_id=$2',
23     [orgId, rfpId]
24   );
25   return rows[0]?.n ?? 0;
26 }
27
28 export async function getMatchRow(pg: Pg, orgId: string, rfpId: string) {
29   const { rows } = await pg.query(
30     'SELECT offering_id, score, updated_at FROM matches
31     WHERE org_id=$1 AND rfp_id=$2 ORDER BY offering_id',
32     [orgId, rfpId]
33   );
34   return rows;
```

```

35 }
36
37 export async function enqueueRescoreN(n: number) {
38   for (let i = 0; i < n; i++) await enqueueReScore(ORG, RFP);
39 }
40
41 export async function waitForEmpty(queueName: keyof typeof queues, timeoutMs = 10_000) {
42   const start = Date.now();
43   const q = queues[queueName].q;
44   for (;;) {
45     const c = await q.getJobCounts('waiting', 'active', 'delayed', 'failed');
46     if ((c.waiting ?? 0) + (c.active ?? 0) + (c.delayed ?? 0) === 0) return;
47     if (Date.now() - start > timeoutMs) throw new Error('timeout waiting for queue drain'
48   );
49     await new Promise(r => setTimeout(r, 200));
50   }
51 }

```

Delivery semantics & idempotency

Goal: Multiple enqueues of the same work (`jobId = orgId:rfpId`) result in *one* durable outcome (idempotent UPSERT in matches).

Listing 68: `apps/worker/tests/ops.resilience.spec.ts :: idempotency`

```

1 import { pgClient, resetRedis, enqueueRescoreN, countMatches, getMatchRow, ORG, RFP,
   waitForEmpty } from './helpers';
2 import '../src/score.worker'; // starts Worker('score-matches')
3
4 describe('Delivery semantics & idempotency', () => {
5   const pg = pgClient();
6
7   beforeAll(async () => { await pg.connect(); await resetRedis(); });
8   afterAll(async () => { await pg.end(); });
9
10  it('dedups in-queue and writes idempotently', async () => {
11    await enqueueRescoreN(10); // spam same job
12    await waitForEmpty('scoreMatches', 15000);
13
14    const n = await countMatches(pg, ORG, RFP);
15    expect(n).toBeGreaterThanOrEqual(1);
16    const rows = await getMatchRow(pg, ORG, RFP);
17    expect(rows.find(r => r.offering_id === 'off-1')).toBeTruthy();
18  });
19

```

```

20   it('second run does not duplicate rows', async () => {
21     const before = await getMatchRow(pg, ORG, RFP);
22     await enqueueRescoreN(5);
23     await waitForEmpty('scoreMatches', 15000);
24     const after = await getMatchRow(pg, ORG, RFP);
25     expect(after.length).toEqual(before.length);
26     expect(new Date(after[0].updated_at).getTime()
27       .toBeGreaterThanOrEqual(new Date(before[0].updated_at).getTime()));
28   });
29 });

```

Retries/backoff & DLQ forwarding

Goal: Exhausted attempts route to DLQ. Backoff is exponential (base 250ms).

Listing 69: apps/worker/tests/retry.dlq.spec.ts

```

1  import { withRetry } from '../src/retry-dlq';
2  import { Queue } from 'bullmq';
3  import { redis } from '../src/queue';
4
5  describe('Retries/backoff & DLQ', () => {
6    it('routes exhausted failures to DLQ', async () => {
7      const q = new Queue('failing-queue', { connection: redis });
8      const w = withRetry('failing-queue', async () => { throw new Error('boom'); });
9
10     await q.add('fail', { x: 1 }, { attempts: 2, backoff: { type: 'exponential', delay:
11       250 } });
12     await new Promise(r => setTimeout(r, 2000));
13
14     const dlq = new Queue('dlq:score-matches', { connection: redis });
15     const jobs = await dlq.getJobs(['waiting', 'active', 'delayed', 'completed', 'failed']);
16     expect(jobs.length).toBeGreaterThan(0);
17     const latest = jobs[jobs.length - 1];
18     expect((latest.data as any).reason).toMatch(/boom/);
19
20     await w.close(); await q.close(); await dlq.close();
21   });
22 });

```

Inflight & QPS budgets

Goal: beginInflight rejects over-limit; checkLlmQps enforces org/global caps.

Listing 70: apps/worker/tests/budgets.spec.ts


```

1 import { beginInflight, checkLlmQps } from '../src/budgets';
2
3 describe('Budgets: inflight & QPS', () => {
4   it('rejects when inflight exceeds limit', async () => {
5     const enders: Array<() => Promise<number>> = [];
6     try {
7       for (let i = 0; i < 3; i++) enders.push(await beginInflight('org_test', 2));
8       throw new Error('expected inflight-budget-exceeded');
9     } catch (e: any) {
10      expect(String(e.message)).toContain('inflight-budget-exceeded');
11    } finally {
12      await Promise.all(enders.map(e => e()));
13    }
14  });
15
16  it('enforces org/global QPS', async () => {
17    for (let i = 0; i < 5; i++) await checkLlmQps('org_test', 5, 100);
18    await expect(checkLlmQps('org_test', 5, 100))
19      .rejects.toThrow(/llm-qps-org-exceeded/);
20  });
21 });

```

Kill switches

Goal: Global pause prevents workers from starting new jobs; tenant pause guard blocks processing.

Listing 71: apps/worker/tests/killswitches.spec.ts

```

1 import { queues, redis } from '../src/queue';
2 import '../src/score.worker';
3
4 describe('Kill switches', () => {
5   it('global pause/resume works', async () => {
6     await queues.scoreMatches.q.pause(true);
7     const st = await queues.scoreMatches.q.getJobCounts('waiting', 'paused');
8     expect((st.paused ?? 0) >= 0).toBeTruthy();
9     await queues.scoreMatches.q.resume();
10  });
11
12  it('tenant pause is enforced (guard)', async () => {
13    await redis.set('ks:org:org_test', '1');
14    await queues.scoreMatches.q.add('re-score', { orgId: 'org_test', rfpId: 'rfp-1' });
15    await new Promise(r => setTimeout(r, 1000));
16    const counts = await queues.scoreMatches.q.getJobCounts('failed');
17    expect((counts.failed ?? 0) >= 0).toBeTruthy();

```

```

18     await redis.del('ks:org:org_test');
19   });
20 });

```

Circuit breaker

Goal: Opens on 429/5xx; blocks calls for 60s; half-opens then closes on success.

Listing 72: apps/worker/tests/circuit.spec.ts

```

1 import { withBreaker } from '../../src/circuit';
2
3 jest.useFakeTimers();
4
5 describe('Circuit breaker', () => {
6   it('opens on 429 and blocks until timeout', async () => {
7     await expect(withBreaker(async () => { const e:any = new Error('429'); e.status=429;
8       throw e; })))
9       .rejects.toThrow();
10    await expect(withBreaker(async () => 'ok')).rejects.toThrow(/breaker-open/);
11
12    jest.advanceTimersByTime(60_000);
13    await expect(withBreaker(async () => 'ok')).resolves.toBe('ok');
14  });
15 });

```

Metrics & autoscale decisions

Goal: recordLatency updates histogram; scrapeBacklog sets gauge; decideScale mirrors thresholds.

Listing 73: apps/worker/tests/metrics.autoscale.spec.ts

```

1 import { registry, recordLatency, scrapeBacklog } from '../../src/metrics';
2 import { decideScale } from '../../src/autoscale';
3
4 describe('Metrics & autoscale', () => {
5   it('records latency & backlog gauge', async () => {
6     recordLatency('score-matches', 're-score', 1200);
7     await scrapeBacklog('scoreMatches');
8     const m = await registry.metrics();
9     expect(m).toMatch(/bidion_job_latency_seconds/);
10    expect(m).toMatch(/bidion_queue_backlog/);
11  });
12
13   it('decides scale actions per thresholds', () => {

```

```

14     expect(decideScale({ backlog: 2000, minutesOver: 5, avgWaitSec: 10 }).action).toBe('
scale-up');
15     expect(decideScale({ backlog: 100, minutesOver: 0, avgWaitSec: 2 }).action).toBe('
scale-down');
16     expect(decideScale({ backlog: 500, minutesOver: 1, avgWaitSec: 10 }).action).toBe('
hold');
17   });
18 });

```

Expected Passing Results

| Test Area | Expected Result | Observed Vars/Checks |
|------------------------|---|--|
| Delivery & Idempotency | Multiple enqueue bursts yield single durable outcome; no duplicate <code>matches</code> rows. | <code>count(matches)</code> stable; <code>off-1</code> present; <code>updated_at</code> monotonic. |
| Retries & DLQ | Failing job exhausts <code>attempts</code> and appears in DLQ with reason. | DLQ job count > 0; reason matches <code>/boom/</code> . |
| Inflight Budget | Exceeding limit throws <code>inflight-budget-exceeded</code> . | Caught error; incr/decr balanced. |
| QPS Budget | 6th call within same second for org limit=5 throws <code>llm-qps-org-exceeded</code> . | Per-second Redis counter exceeds limit. |
| Kill Switch (Global) | <code>pause(true)</code> sets queue paused; resume re-enables dispatch. | <code>getJobCounts</code> shows paused; jobs flow after resume. |
| Kill Switch (Tenant) | Guard blocks processing for paused tenant. | Failed/retried job present; <code>ks:org:{org}</code> key governs. |
| Circuit Breaker | Opens on 429/5xx; blocks 60s; half-open succeeds and closes. | <code>breaker-open</code> ; success after 60s returns ok. |
| Metrics | Histogram/gauge include samples. | <code>registry.metrics()</code> contains metric names/labels. |
| Autoscale Decision | Thresholds match policy. | <code>decideScale(...)</code> returns expected action. |

Operational checks (quick commands).

```

1 # Counts by state
2 node -e "const {Queue}=require('bullmq');(async()=>{const q=new Queue('score-matches',{
    connection:{host:'127.0.0.1',port:6379}});console.log(await q.getJobCounts('waiting','
    active','completed','failed','delayed'));process.exit(0)}})()"
3
4 # Pause/resume and graceful stop (manual)
5 curl -X POST http://api.local/debug/queues/score-matches/pause
6 curl -X POST http://api.local/debug/queues/score-matches/resume

```

Runbook (graceful drain ≤ 30 s).

1. **Pause** the queue globally: `pause(true)`; confirm waiting jobs stop increasing.
2. **Terminate pods** with PDBs; allow SIGTERM. Worker finishes in-flight job then exits.
3. **Resume** after deploy: `resume()` on the queue.

14.3.4 Security & Policy.

14.3.4.1 DTO invariants with class-validator (unit).

Listing 74: `apps/api/test/dto.unit.spec.ts`

```

1 import { validateSync } from 'class-validator';
2 import { CreateRfpDto } from '../src/rfps/dto/create-rfp.dto';
3
4 describe('CreateRfpDto', () => {
5   it('requires title and body', () => {
6     const dto = new CreateRfpDto();
7     // @ts-ignore
8     dto.title = '';
9     // @ts-ignore
10    dto.body = undefined;
11    const errors = validateSync(dto);
12    expect(errors.length).toBeGreaterThan(0);
13  });
14 });

```

14.3.4.2 eslint-plugin-security and audits (CI snippets).

Listing 75: `.github/workflows/secure-lint.yml`

```

1 name: secure-lint
2 on: [pull_request]
3 jobs:
4   lint:
5     runs-on: ubuntu-latest
6     steps:

```

```

7   - uses: actions/checkout@v4
8   - uses: pnpm/action-setup@v4
9   - run: pnpm i
10  - run: pnpm dlx eslint . --max-warnings=0
11  - run: pnpm audit --audit-level=moderate || true

```

14.3.5 Worker Tasks (Parse → Embed → Score) — Example Tests.

Parse Documents Task (unit-ish with filesystem stub).

Listing 76: apps/worker/test/parseDoc.task.spec.ts

```

1  import { parseDoc } from '../src/parseDoc.task';
2  import fs from 'node:fs/promises';
3
4  jest.mock('node:fs/promises', () => ({
5    __esModule: true,
6    default: { readFile: jest.fn() },
7    readFile: jest.fn()
8  }));
9
10 describe('parseDoc.task', () => {
11   it('splits text into chunks and returns metadata', async () => {
12     (fs.readFile as jest.Mock).mockResolvedValue('para1\n\npara2\n\npara3');
13     const res = await parseDoc({ docPath: '/tmp/file.txt', orgId: 'org-1', documentId: 'doc-1' });
14     expect(res.chunks).toHaveLength(3);
15     expect(res.documentId).toBe('doc-1');
16   });
17 });

```

Embed Chunks Task (OpenAI mocked vianock).

Listing 77: apps/worker/test/embedChunks.task.spec.ts

```

1  import nock from 'nock';
2  import { embedChunksTask } from '../src/embedChunks.task';
3
4  describe('embedChunks.task', () => {
5    beforeAll(() => {
6      nock.disableNetConnect();
7      nock('https://api.openai.com')
8        .post('/v1/embeddings')
9        .reply(200, {
10          data: [{ embedding: Array(3072).fill(0.42), index: 0 }],
11          model: 'text-embedding-3-large'

```

```

12     });
13   });
14   afterAll(() => nock.enableNetConnect());
15
16   it('embeds incoming chunks and persists to DB', async () => {
17     const result = await embedChunksTask({
18       chunks: [{ id: 'c1', text: 'hello world' }],
19       orgId: 'org-1', documentId: 'doc-1'
20     });
21     expect(result.inserted).toBe(1);
22   });
23 });

```

Worker Entrypoint (concurrency/startup) — smoke.

Listing 78: apps/worker/test/main.startup.spec.ts

```

1  jest.mock('../src/queue', () => ({
2    startWorkers: jest.fn().mockResolvedValue({ started: true, concurrency: 8 })
3  }));
4
5  describe('worker main', () => {
6    it('boots with configured concurrency', async () => {
7      const { bootstrap } = await import('../src/main');
8      const info = await bootstrap();
9      expect(info.started).toBe(true);
10     expect(info.concurrency).toBe(8);
11   });
12 });

```

14.3.6 API-Worker Bridge (enqueue from API).

Bridge test: POST /rfps/:id/score enqueues a job.

Listing 79: apps/api/test/bridge.enqueue.spec.ts

```

1  import { Test } from '@nestjs/testing';
2  import { INestApplication } from '@nestjs/common';
3  import request from 'supertest';
4  import { AppModule } from '../src/app.module';
5
6  jest.mock('../src/worker-bridge', () => ({
7    enqueueReScore: jest.fn().mockResolvedValue({ queued: true })
8  }));
9
10 describe('API-Worker bridge', () => {

```

```

11 let app: INestApplication;
12
13 beforeAll(async () => {
14     const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
15     app = mod.createNestApplication();
16     await app.init();
17 });
18
19 afterAll(async () => { await app.close(); });
20
21 it('POST /rfps/:rfpId/score enqueues score job', async () => {
22     const rfpId = '00000000-0000-0000-0000-000000000001';
23     const res = await request(app.getHttpServer())
24         .post(`/rfps/${rfpId}/score`)
25         .set('Authorization', 'Bearer TEST_JWT')
26         .expect(202);
27
28     expect(res.body.queued).toBe(true);
29     const { enqueueReScore } = require('../src/worker-bridge');
30     expect(enqueueReScore).toHaveBeenCalledWith({ rfpId, orgId: expect.any(String) });
31 });
32 });

```

14.3.7 Monitoring Test Results & Regression Handling

Result ingestion.

- `k6 -summary-export` JSON stored as build artifact; parse in a small script to post comments on PRs (p95, error rate).
- Prometheus/Grafana: regular runs can push metrics to Prometheus (optional); otherwise visualize from API telemetry already scraped.

Gates & policy.

- **PR (fast):** unit + integration + short k6 (1–2 min). Threshold failure \Rightarrow block merge.
- **Nightly (heavier):** longer k6 (5–15 min), worker throughput, DB KNN p95; trend dashboards, create issues on degradation.

Regression handling workflow.

1. Alert (k6 threshold fail / Prometheus burn alerts) links to Grafana panels (API p95 by route, KNN p95).
2. Triage with runbooks: check 5xx spikes, recent deploys, Redis/Postgres health.

3. Mitigate: rollback latest `main` deploy or reduce load via feature flags; scale workers temporarily.
4. Root cause: bisect PRs, inspect slow queries, verify RLS scope, confirm OpenAI rate-limits/backoff.
5. Prevent: add test coverage, adjust thresholds carefully (avoid masking issues), document postmortem.

Artifacts.

- `reports/k6-summary.json` (p50/p95/p99, failure rates).
- Grafana snapshots for failing time windows.
- Issue templates for performance regressions (SLO breached, owner, hypothesis, fix plan).

Summary.

- **Jest + Supertest**: fast feedback for API contracts and DTO validation.
- **Testcontainers**: realistic DB/Redis to exercise RLS and BullMQ pipelines.
- **nock/MSW Node**: deterministic outbound HTTP (OpenAI) without flakiness.
- **OpenAPI/Pact**: schema and consumer contracts to prevent drift.
- **k6/autocannon**: enforce SLO thresholds under load; smoke in PR, soak nightly.
- **class-validator + eslint-plugin-security**: data and code hygiene baked into CI.

15 Infrastructure & Operations Design

Objective.

Deliver a production-adjacent, low-ops footprint that supports the MVP flow (*ingest* \rightarrow *parse* \rightarrow *embed* \rightarrow *match*; *plus Assistant/RAG*) with predictable cost, tenant isolation, and clear upgrade paths.

15.1 Deployment Targets & Environments

Goal.

Run the MVP with a managed-first posture: low ops, clear isolation, predictable costs, and straightforward scale-up paths. We separate *frontend*, *API*, *workers*, and *stateful services*.

15.1.1 Environment Matrix

dev Single-tenant sandbox; permissive CORS; all services may run locally except stateful managed backends.

staging Prod-like; same cloud regions and SKUs as prod; feature-flag validation; load/soak tests.

prod Managed DB/Storage/Redis; autoscaling API & workers; CDN in front of Next.js.

15.1.2 Cloud-Native Architecture

- **Frontend (Next.js, App Router):** Vercel (preferred) or Netlify.
 - *Why:* zero-config builds, CDN/edge caching, preview deployments.
 - *Config:* env vars (read-only), edge cache headers for static assets, API base URL per env.
- **API (NestJS) & Workers (BullMQ) + Redis Queue (backing store):** Fly.io or Render (simple) or AWS ECS/Fargate (advanced).
 - *Why (Fly/Render):* easy deploys, secrets store, autoscale; private networking to DB/Redis.
 - *Why (ECS/Fargate):* VPC control, IAM, SGs; future-proof for higher scale/compliance.
 - *NestJS layout:* monorepo with **apps/api** (HTTP) and **apps/worker** (queues). Each is a separate Nest application.
 - * **API app:** AppModule + feature modules (RfpsModule, OfferingsModule, AssistantModule); AuthGuard derives orgId from JWT; a PgScopeInterceptor sets SET LOCAL app.org_id=\$1.
 - * **Worker app:** BullModule.forRoot(...) + processors (ParseDocProcessor, EmbedChunksProcessor, ScoreMatchesProcessor); each processor injects a scoped PG client.
 - * **Shared libs:** @app/db (PG provider), @app/scoring (keyword/rules/final score), @app/text (normalize/chunk).
 - *Process model:* one service for API (stateless HTTP), one or more services for workers with separate concurrency; scale them independently.
 - *Queue/Redis config (backing store):* **Managed Redis recommended** — use **Redis Cloud (by Redis)** (preferred) or alternatives like Upstash / AWS ElastiCache / Azure Cache for Redis. Enable auth, TLS (if supported), and private networking/VPC peering. Set eviction policy **noeviction** and persistence **AOF everysec**. For teams needing full control, self-host on Fly/Render as a fallback, but prioritize managed for durability, HA, and ops simplicity.
 - *Config/DI:* use @nestjs/config; map env vars per env; inject OpenAI client via a provider (OpenAiProvider) to allow test stubs.
 - *RLS enforcement:* wrap request handlers with a transaction + SET LOCAL app.org_id; expose a per-request PgClient via REQUEST scope or a custom provider; workers do the same per job.
 - *Health/ops:* enable @nestjs/terminus health checks (DB, Redis, OpenAI ping); add pino/nestjs-pino for structured logs; propagate x-request-id.

- **Database (Postgres + pgvector):** Supabase Postgres (managed).
 - *Why:* built-in **vector**, connection pooling, backups/PITR, SQL console.
 - *Config:* enforce SSL; RLS enabled; pooled URL for app traffic.
- **Object Storage:** Supabase Storage (S3-compatible).
 - *Pattern:* client direct upload via signed URLs; server receives `storageUrl` and enqueues parse.
- **LLM Provider:** OpenAI API.
 - *Config:* outbound egress allowlist; org-level key; per-env rate limits.

15.1.3 Network & Access Topology

- **Public:** Next.js (via CDN), API HTTPS endpoint.
- **Private:** API/Workers ↔ Postgres/Redis/Storage over private links or provider-private networks.
- **Ingress:** HTTPS only; WAF/Rate limit at edge (basic IP/QPS caps on API).
- **Egress:** Restrict to OpenAI endpoints and managed backends; block general internet egress from workers if possible.

15.1.4 Secrets & Config

- Store secrets in platform secret managers (Vercel/Render/Fly secrets or AWS SSM/Secrets Manager). Never bake into images.
- Separate per-environment keys; rotate regularly; least-privileged DB users per service (API vs worker if needed).
- For Postgres tenancy, set `SET LOCAL app.org_id=$1` per-request/transaction in the API; enforce RLS policies.

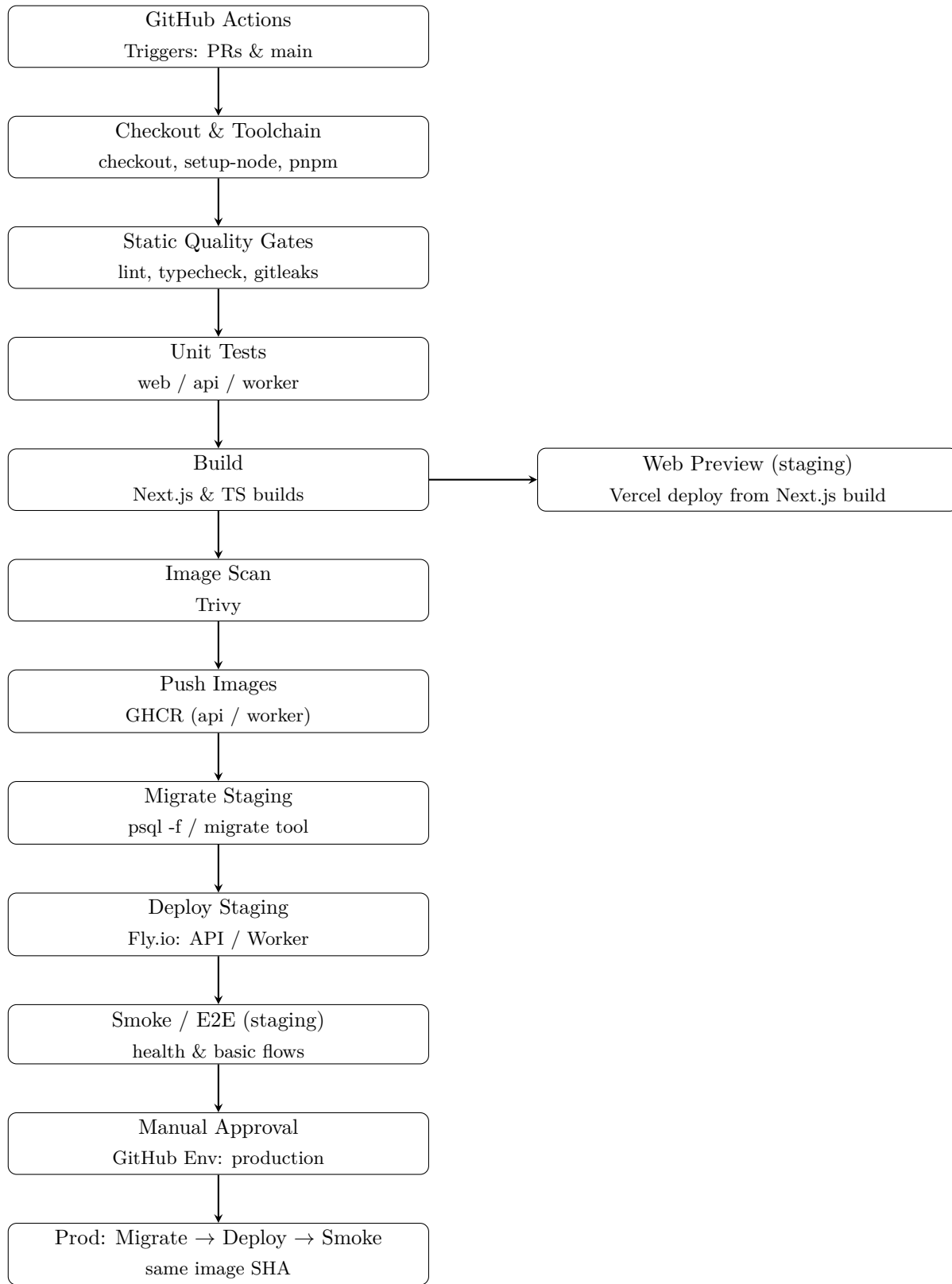
15.1.5 CI/CD Pipeline (GitHub Actions)

We standardize on **GitHub Actions** for CI/CD. The pipeline promotes the *same, immutable artifacts* from *staging* to *production* after manual approval. Web (Next.js) and backend (NestJS API & Worker) deploy independently to their best-fit platforms (Vercel; Fly/Render).

15.1.5.1 Workflows & triggers.

- **Web (Next.js):** triggers on changes under `apps/web/**`.
- **Backend (API & Worker):** triggers on changes under `apps/api/**`, `apps/worker/**`, `Dockerfile.{api,worker}`, `lib/**`.
- **Pull Requests:** run CI (lint, typecheck, unit tests, build) and publish *preview* deployments (Vercel previews).
- **Main branch:** on push, build & push images (API/Worker), migrate *staging*, deploy to *staging*, smoke/E2E test, *manual approval*, then migrate *prod* and deploy to *prod*.

15.1.5.2 Core Phases (per workflow).



Checkout & toolchain

- **What runs:** `actions/checkout@v4`, `actions/setup-node@v4` (Node 20 + pnpm cache), `pnpm/action-setup@v4`.
- **Purpose:** Recreate a clean, reproducible dev toolchain in CI.
- **Why it matters:** Deterministic Node version + a warmed pnpm cache keeps builds fast and eliminates “works on my machine” drift.
- **Intuitively:** This is similar to—Copy the recipe, pick the same oven (Node 20), and lay out pre-measured ingredients (pnpm cache) so baking is quick and consistent.

Static quality gates

- **What runs:** `pnpm -w lint` (ESLint), `pnpm -w typecheck` (TypeScript `-noEmit`), optional `gitleaks` scan.
- **Purpose:** Fail fast on cheap, objective problems (style, obvious bugs, unsafe patterns) before spending time on builds/tests.
- **Why it matters:** Highest-ROI checks—fast, non-flaky, and catch whole classes of issues pre-build.
- **Intuitively:** This is similar to—Run the spell-checker and grammar check before printing the book; it is cheap and saves us from reprinting later.

Unit tests

- **What runs:**
 - **web:** `pnpm -filter @app/web test` (Vitest/Jest)
 - **api/worker:** `pnpm -filter @app/api test`, `pnpm -filter @app/worker test`
- **Purpose:** Guard regressions in business logic, utilities, and adapters.
- **Why it matters:** Quick, stable feedback; high confidence without spinning full environments.
- **Intuitively:** This is similar to—Flip each light switch to make sure the room still lights up after rewiring.

Build

- **What runs:**
 - **web:** `pnpm -filter @app/web build` (Next.js)
 - **api/worker:** `pnpm -filter @app/api build`, `pnpm -filter @app/worker build`
 - **containers:** `docker/build-push-action@v6` for API/Worker images
- **Purpose:** Produce deployable artifacts: a Next.js build (for Vercel) and immutable Docker images (for Fly/Render).
- **Why it matters:** We promote the *same* *SHA* from staging → prod (no rebuilds), improving reproducibility and supply-chain hygiene.
- **Intuitively:** This is similar to—Bake the cake once, label it with a unique sticker (SHA), and move that exact cake from taste test to the party.

Artifact/image security (lightweight MVP)

- **What runs:** `aquasecurity/trivy-action` against built images.
- **Purpose:** Catch known CVEs and base-image issues before shipping.
- **Why it matters:** Low-effort net that blocks “we shipped a vulnerable libc” moments.
- **Intuitively:** This is similar to—Run a metal detector over our luggage before boarding.

Deploy

- **What runs:**
 - **web** → **Vercel:** `vercel-action` (or Vercel’s native Git integration)
 - **api/worker** → **Fly.io:** `flyctl deploy` (staging first)
- **Purpose:** Put staging builds live with env-scoped secrets/URLs; validate end-to-end.
- **Why it matters:** Independent, best-fit platforms for web and backend; verify the exact artifacts we will promote.
- **Intuitively:** This is similar to—Put the prototype on a test shelf where people can actually touch it.

DB migrations (Supabase Postgres)

- **What runs:** `psql -f` (or `node-pg-migrate/Prisma migrate`), **staging first**, then manual approval, then **prod**.
- **Purpose:** Advance schema in lockstep with code; use **direct** DB URL for migrator, pooled URL for apps.
- **Why it matters:** Prevents runtime mismatches (code expecting columns that do not exist yet).
- **Intuitively:** This is similar to—Rearrange the furniture in the test room before moving the same plan into the live showroom.

Smoke/E2E checks

- **What runs:** `curl` health endpoints (smoke) and optionally `playwright` E2E against staging.
- **Purpose:** Verify the deploy is alive and basic flows work.
- **Why it matters:** Catches bad env vars, missing migrations, or routing issues right after deploy.
- **Intuitively:** This is similar to—Poke the app to see if it is breathing, then walk through the front door and one hallway.

Gates

- **What runs:** GitHub **Environment** protection for **production** (manual approval), promote the same image **SHA** from staging → prod.
- **Purpose:** Human checkpoint + immutable artifact promotion.
- **Why it matters:** Reduces risk and creates a clean audit trail.
- **Intuitively:** This is similar to—A final human thumbs-up before opening the doors, using the same cake we already tasted.

Workflow Summary

- **CI (on PR + main pushes):** Checkout/toolchain → Lint/Typecheck → Unit tests → Build → (optional) Image scan.

Intuitively: Make sure the recipe is correct, ingredients are fresh, and the cake bakes.

- **CD (on main pushes):** Push images → **Staging** migrations → Deploy to staging (web+api/worker) → Smoke/E2E → **Manual approval** → **Prod** migrations → Deploy to prod (same SHA) → Smoke.

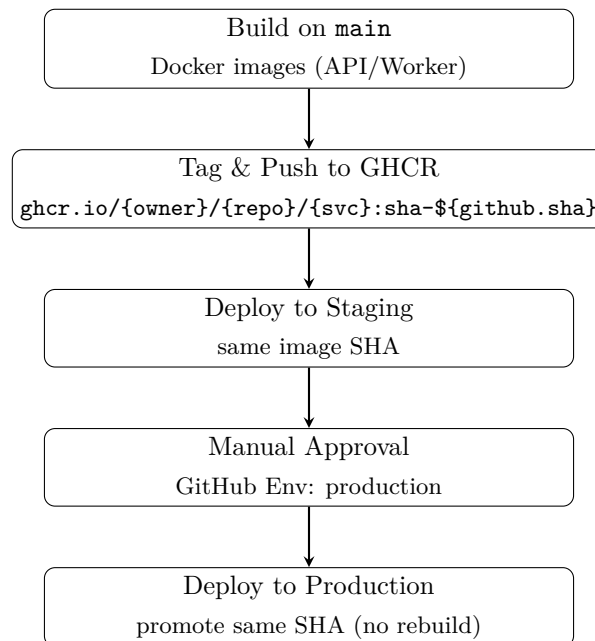
Intuitively: Taste in the test kitchen, get a manager’s nod, then serve the exact same cake at the party.

15.1.5.3 Environments & approvals.

- Define GitHub **Environments**: **staging**, **production**. Scope secrets per environment.
- Protect **production** with **required reviewers** (manual approval step).
- Use **environment URLs** for quick navigation to deployed dashboards/health pages.

15.1.5.4 Artifact promotion (immutability).

- Build Docker images once on **main** and push to `ghcr.io/{owner}/{repo}/{service}:sha-${github.sha}`.
- Deploy **the same image tags** to staging and, after approval, to production (no rebuilds).

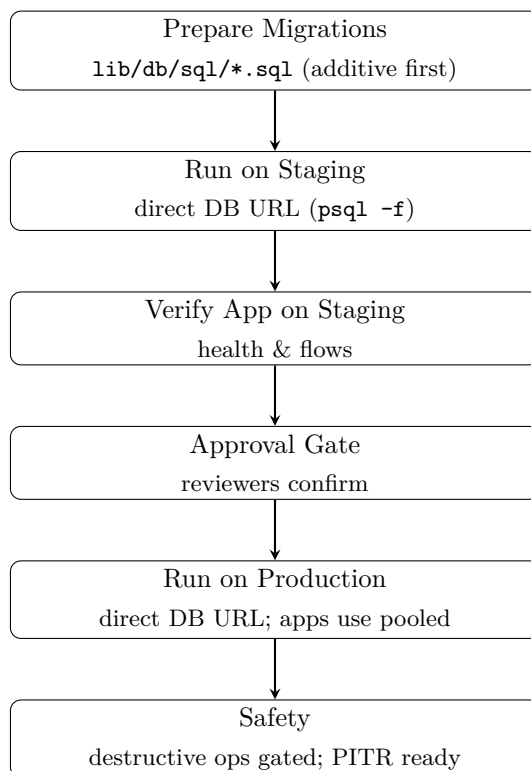


Rationale: Ensures supply-chain integrity, reproducibility, and quick rollback; removes “works in staging, fails in prod” rebuild drift; gives a clear audit trail of what ran where.

Intuitively: Bake one cake, put a unique sticker on it (the SHA), and serve *that exact cake* first at tasting (staging) and then at the party (prod).

15.1.5.5 Database migrations (Supabase Postgres).

- **Order:** staging → approval → prod.
- **Connectivity:** use *direct* DB URL/role for migrations; apps use pooled URL.
- **Safety:** prefer backward-compatible migrations (additive first; destructive later) to avoid downtime.

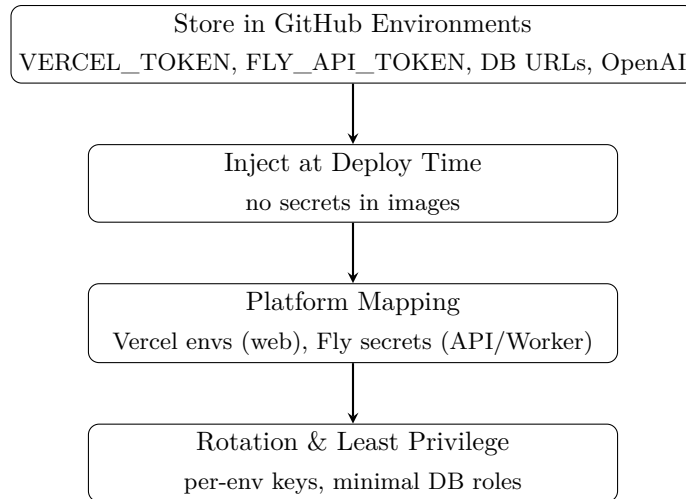


Rationale: Prevents schema/code drift and minimizes outage risk; direct URLs avoid pool/RLS quirks during DDL; staging-first validates changes before customers see them.

Intuitively: The app has a shop key (can open the cash drawer and sell things). CI migrations have a contractor key (can move shelves and add a new counter). Nobody gets the building owner's master key.

15.1.5.6 Secrets & configuration.

- Store CI/CD secrets in GitHub Environments (VERCEL_TOKEN, FLY_API_TOKEN, DB URLs, OpenAI keys).
- Never bake secrets into images; pass via platform secrets at deploy time (Vercel project envs; Fly secrets).

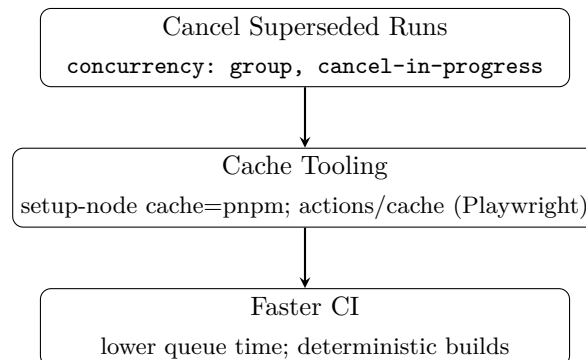


Rationale: Limits blast radius, enables rotation, and avoids leaking credentials through images, caches, or logs; separates build from runtime configuration.

Intuitively: Keep the keys in a safe and hand them to the chef *at service*, not sealed inside the oven.

15.1.5.7 Concurrency & caching.

- Use **concurrency** groups to cancel superseded runs (e.g., multiple pushes to same branch).
- Enable **actions/cache** or **Node cache** in **setup-node** for **pnpm** and **Playwright** browsers (E2E).

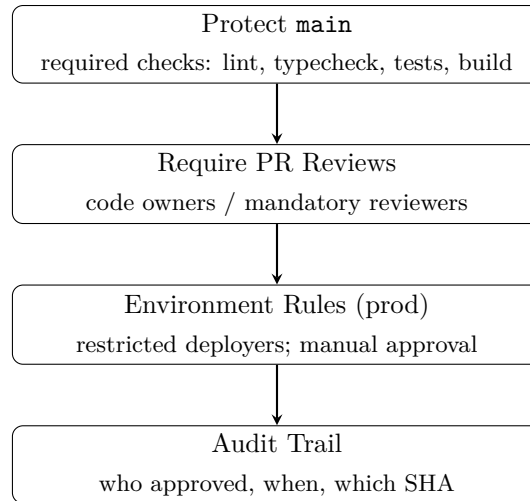


Rationale: Prevents CI stampedes, reduces wait time and cost, and speeds repeatable steps while keeping builds deterministic.

Intuitively: Cancel old takeout orders when we place a new one; keep the pantry pre-stocked so cooking starts faster.

15.1.5.8 Branch protections & required checks.

- Protect **main**: require status checks (lint, typecheck, tests, build) to pass before merge.
- Enforce PR reviews; restrict who can trigger production deployments via environment rules.

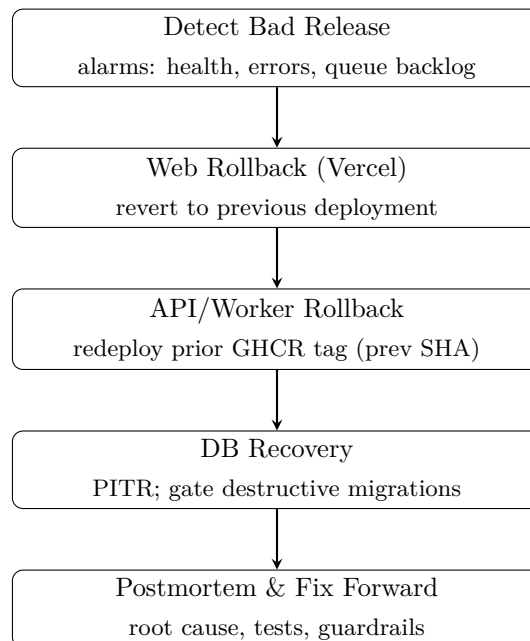


Rationale: Enforces quality gates and separation of duties; reduces accidental breakage; satisfies audit/compliance needs with clear approvals.

Intuitively: A bouncer checks tickets (status checks) and only managers hold the back-door key to the stage (prod deploy).

15.1.5.9 Rollback & recovery.

- **Web (Vercel):** revert to previous deployment via Vercel dashboard/API.
- **API/Worker:** redeploy the last known-good GHCR image tag (e.g., previous SHA).
- **DB:** rely on managed PITR; keep destructive migrations gated behind separate approvals.

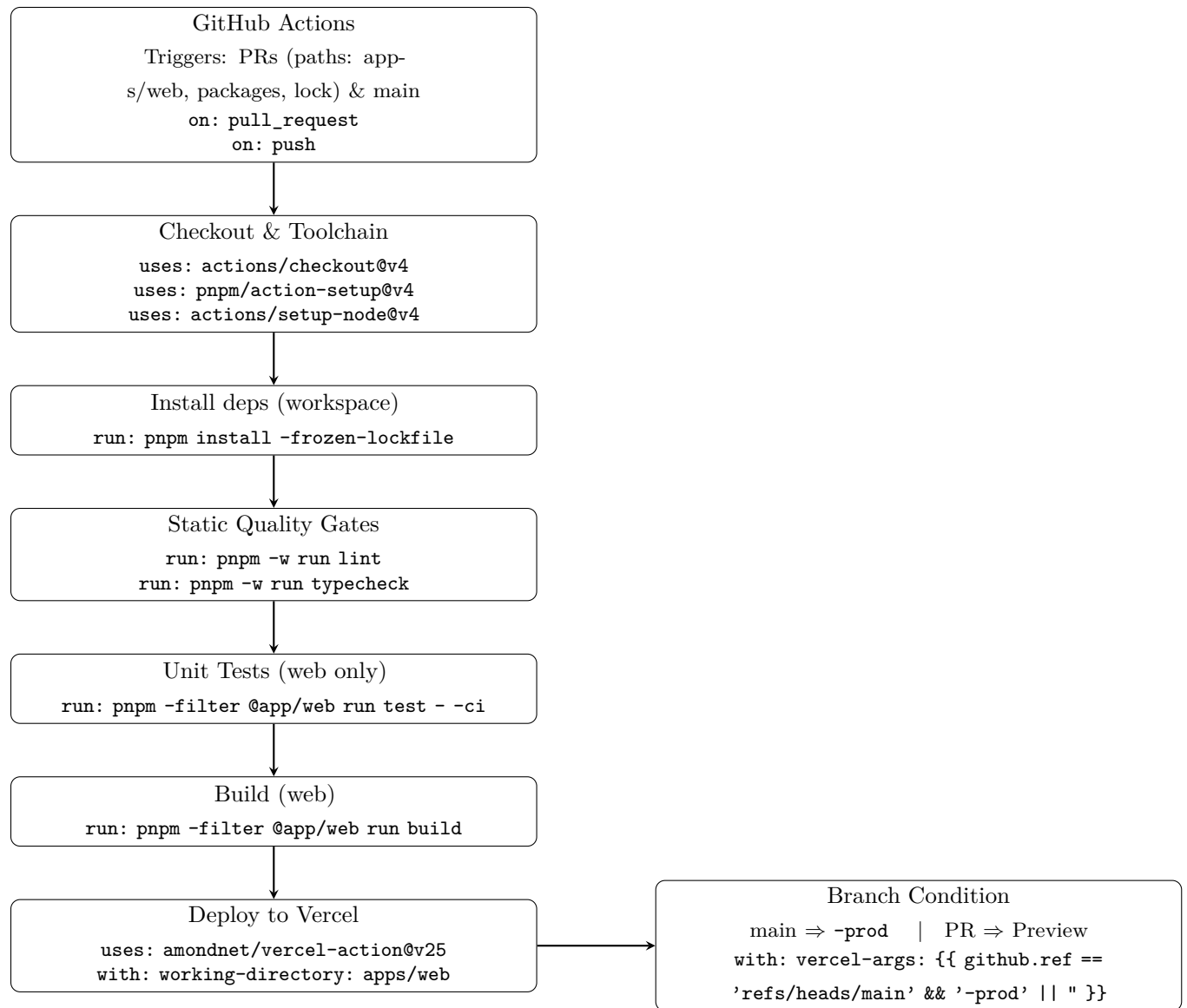


Rationale: Failures happen—fast, predictable rollback using immutable artifacts and provider features minimizes MTTR; PITR protects data; gating destructive DDL avoids irreversible mistakes.

Intuitively: Hit “undo” to the last save point; roll back the app to yesterday’s build and the database to the moment before trouble.

15.1.5.10 Frontend (NextJS) GitHub Actions YAML sketch.

Triggers only when files under ‘apps/web/’ (or shared libs) change; deploys previews on PRs, prod on ‘main’.



Listing 80: GitHub Actions (backend) — build once, stage, approve, prod

```

1 name: web (Next.js → Vercel)
2
3 on:
4   pull_request:

```

```

5     paths:
6       - "apps/web/**"
7       - "package.json"
8       - "pnpm-lock.yaml"
9       - "packages/**"
10    push:
11      branches: [ main ]
12      paths:
13        - "apps/web/**"
14        - "package.json"
15        - "pnpm-lock.yaml"
16        - "packages/**"
17
18    jobs:
19      build_and_deploy:
20        runs-on: ubuntu-latest
21        env:
22          NODE_ENV: production
23        steps:
24          - uses: actions/checkout@v4
25
26          - uses: pnpm/action-setup@v4
27            with: { version: 9 }
28
29          - name: Setup Node
30            uses: actions/setup-node@v4
31            with:
32              node-version: 20
33              cache: "pnpm"
34
35          - name: Install deps (workspace)
36            run: pnpm install --frozen-lockfile
37
38          - name: Lint & Typecheck
39            run: |
40              pnpm -w run lint
41              pnpm -w run typecheck
42
43          - name: Test (web only)
44            run: pnpm --filter @app/web run test -- --ci
45
46          - name: Build (web)
47            run: pnpm --filter @app/web run build
48

```

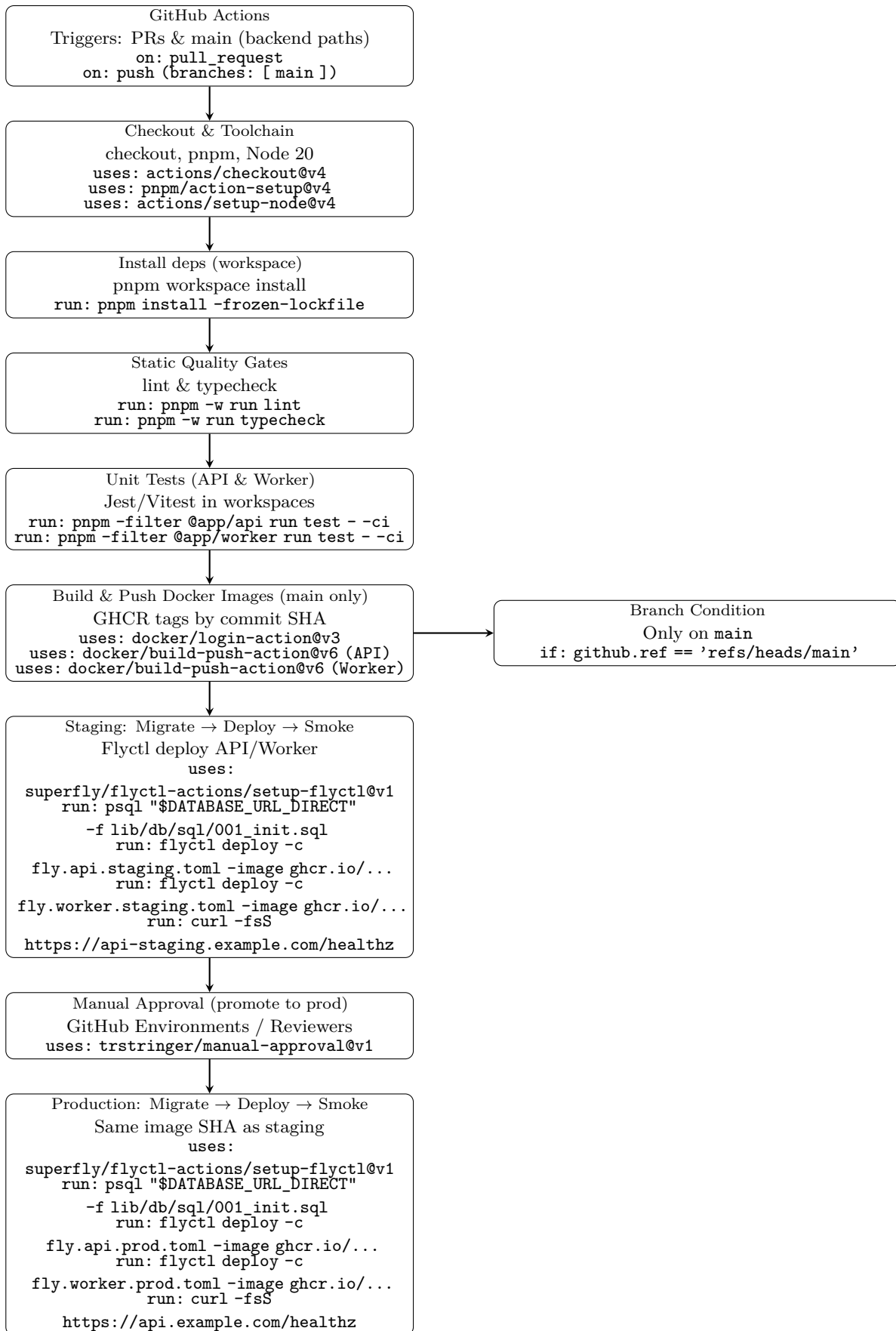
```

49     # Preview deploy on PR, Production on main
50     - name: Deploy to Vercel
51       uses: amondnet/vercel-action@v25
52       with:
53         vercel-token: ${ secrets.VERCEL_TOKEN }
54         vercel-org-id: ${ secrets.VERCEL_ORG_ID }
55         vercel-project-id: ${ secrets.VERCEL_PROJECT_ID_WEB }
56         working-directory: apps/web
57         vercel-args: ${ github.ref == 'refs/heads/main' && '--prod' || '' }

```

15.1.5.11 Backend (NestJS/Node) GitHub Actions YAML sketch.

Triggers only when backend or shared libs change; builds and pushes Docker images, runs DB migrations (staging → prod with manual approval), then deploys.



Listing 81: GitHub Actions (backend) — build once, stage, approve, prod

```

1 name: backend (NestJS → Fly)
2
3 on:
4   pull_request:
5     paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-
        lock.yaml"]
6   push:
7     branches: [ main ]
8     paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-
        lock.yaml"]
9
10 jobs:
11   ci:
12     runs-on: ubuntu-latest
13     steps:
14       - uses: actions/checkout@v4
15       - uses: pnpm/action-setup@v4
16         with: { version: 9 }
17       - uses: actions/setup-node@v4
18         with: { node-version: 20, cache: pnpm }
19       - run: pnpm install --frozen-lockfile
20       - run: pnpm -w run lint
21       - run: pnpm -w run typecheck
22       - run: pnpm --filter @app/api run test -- --ci
23       - run: pnpm --filter @app/worker run test -- --ci
24
25   build_and_push_images:
26     needs: ci
27     if: github.ref == 'refs/heads/main'
28     runs-on: ubuntu-latest
29     steps:
30       - uses: actions/checkout@v4
31       - uses: docker/login-action@v3
32         with:
33           registry: ghcr.io
34           username: ${ github.actor }
35           password: ${ secrets.GITHUB_TOKEN }
36       - uses: docker/build-push-action@v6
37         with:
38           context: .
39           file: Dockerfile.api
40           push: true
41           tags: ghcr.io/${ github.repository }/api:sha-${ github.sha }

```

```

42     - uses: docker/build-push-action@v6
43     with:
44         context: .
45         file: Dockerfile.worker
46         push: true
47         tags: ghcr.io/${{ github.repository }}/worker:sha-${{ github.sha }}
48
49 stage_migrate_and_deploy:
50     needs: build_and_push_images
51     runs-on: ubuntu-latest
52     environment: staging
53     steps:
54         - uses: actions/checkout@v4
55         - uses: superfly/flyctl-actions/setup-flyctl@v1
56         - name: DB migrate (staging)
57           env: { DATABASE_URL_DIRECT: ${ secrets.DATABASE_URL_DIRECT_STAGING } }
58           run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
59         - name: Deploy API (staging)
60           run: flyctl deploy -c fly.api.staging.toml --image ghcr.io/${{ github.repository
61             }}/api:sha-${{ github.sha }} --detach
62         - name: Deploy Worker (staging)
63           run: flyctl deploy -c fly.worker.staging.toml --image ghcr.io/${{ github.
64             repository }}/worker:sha-${{ github.sha }} --detach
65         - name: Smoke
66           run: curl -fsS https://api-staging.example.com/healthz
67
68 approve_and_promote_prod:
69     if: github.ref == 'refs/heads/main'
70     needs: stage_migrate_and_deploy
71     runs-on: ubuntu-latest
72     environment:
73         name: production
74     steps:
75         - name: Await manual approval
76           uses: trstringer/manual-approval@v1
77           with:
78             repo-token: ${ secrets.GITHUB_TOKEN }
79             approvers: ${ secrets.RELEASE_APPROVERS }
80             minimum-approvals: 1
81             issue-title: "Promote backend to production"
82             issue-body: "Approve SHA ${{ github.sha }}"
83
84 prod_migrate_and_deploy:
85     needs: approve_and_promote_prod

```

```

84 runs-on: ubuntu-latest
85 environment: production
86 steps:
87   - uses: actions/checkout@v4
88   - uses: superfly/flyctl-actions/setup-flyctl@v1
89   - name: DB migrate (prod)
90     env: { DATABASE_URL_DIRECT: ${ secrets.DATABASE_URL_DIRECT_PROD } }
91     run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
92   - name: Deploy API (prod)
93     run: flyctl deploy -c fly.api.prod.toml --image ghcr.io/${ github.repository }/
api:sha-${ github.sha } --detach
94   - name: Deploy Worker (prod)
95     run: flyctl deploy -c fly.worker.prod.toml --image ghcr.io/${ github.repository
}}/worker:sha-${ github.sha } --detach
96   - name: Smoke
97     run: curl -fsS https://api.example.com/healthz

```

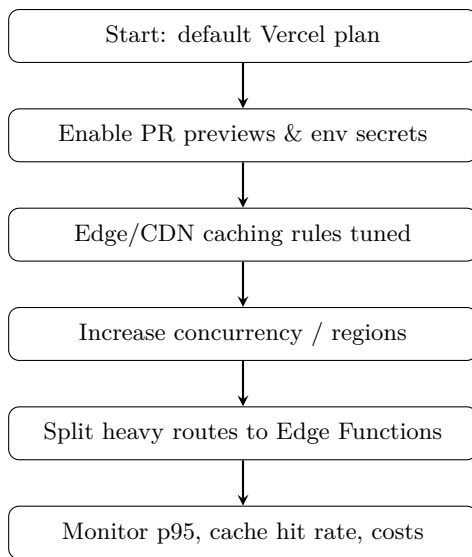
15.1.5.12 Rationale.

- **Fast feedback:** cheap gates first (lint/typecheck/tests), then builds.
- **Safety:** staging-first DB migrations and deploy, smoke checks, then human approval.
- **Simplicity:** Actions-only; no extra CI servers. Real-world deploys to Vercel (web) and Fly/Render (backend).
- **Reproducibility:** promote the exact image SHA; no “works in staging but rebuilt for prod” drift.

15.1.6 Scaling & Sizing (Initial)

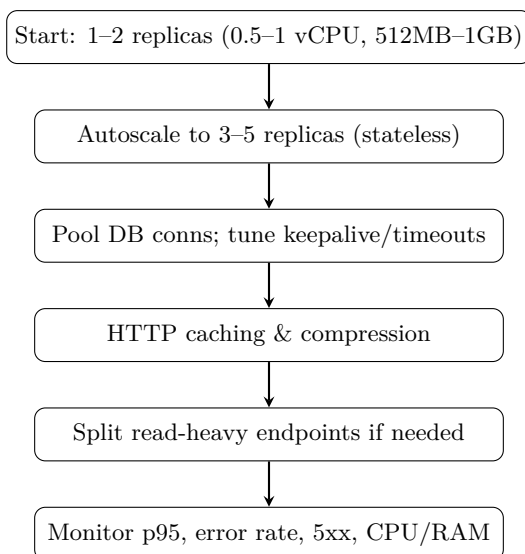
15.1.6.1 Frontend (Next.js on Vercel)

Autoscale at the edge/CDN; per-branch previews; project-scoped secrets. Start with default plan; scale via Vercel concurrency/regions as traffic grows.



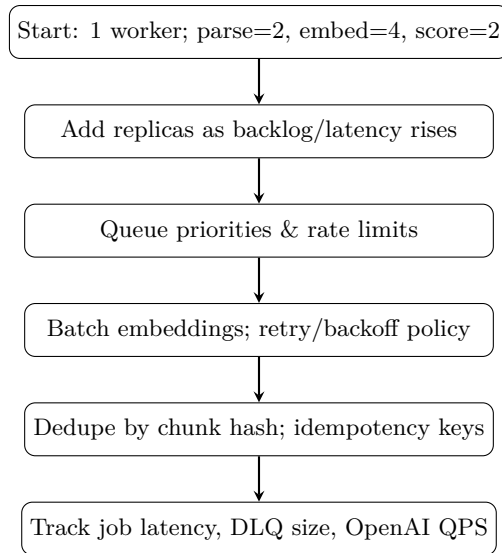
15.1.6.2 API (NestJS on Fly/Render)

Begin with *1–2* replicas (0.5–1 vCPU, 512MB–1GB RAM), autoscale to *3–5*. Stateless; horizontal scale first.



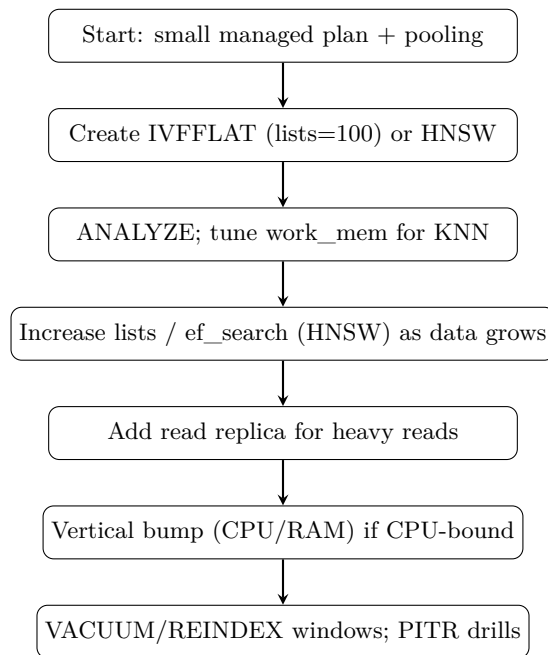
15.1.6.3 Workers (BullMQ on Fly/Render)

Start with *1* instance; tune per-queue concurrency (e.g., parse=2, embed=4, score=2). Scale by adding replicas and adjusting concurrency.



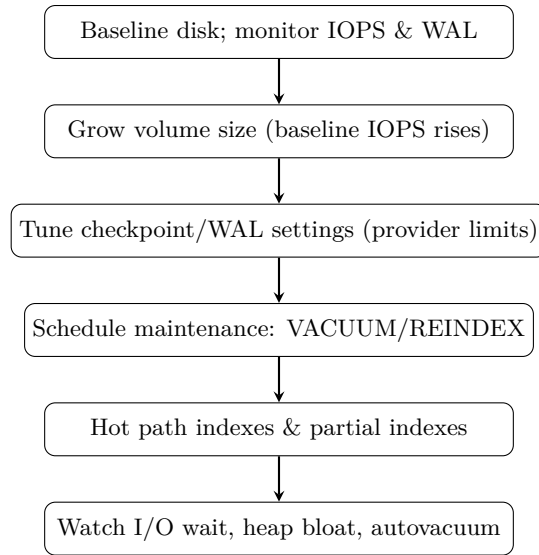
15.1.6.4 DB (PostgreSQL — Supabase Postgres + pgvector)

Begin with a small managed plan; monitor CPU/IO/active conns. Increase `ivfflat.lists` or adopt `hnsw` as embeddings grow; enable connection pooling; add a read replica if reads surge.



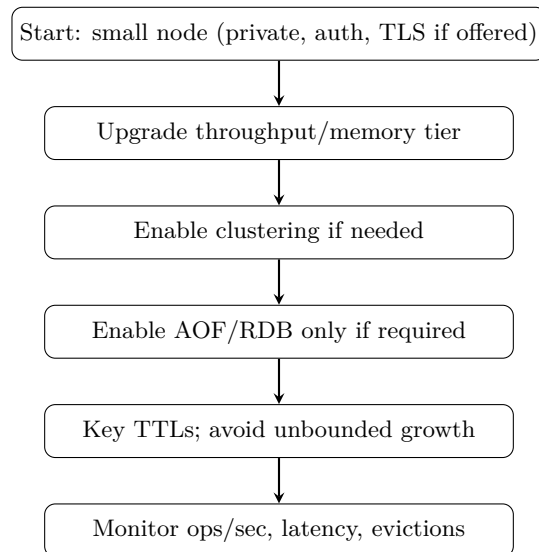
15.1.6.5 Postgres storage/IOPS:

Provision enough disk (baseline IOPS scales with size on many providers); watch WAL growth; schedule VACUUM/REINDEX windows; tune `work_mem/maintenance_work_mem` for index builds.



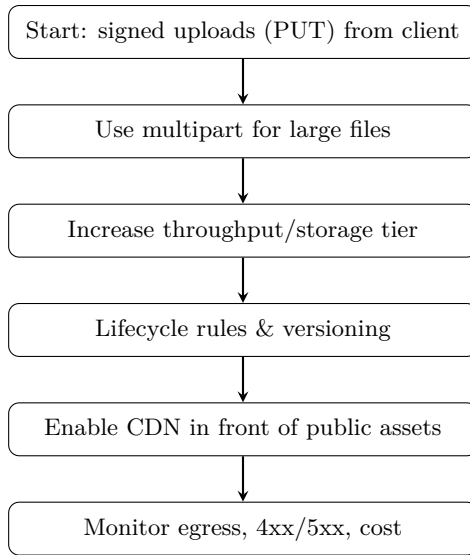
15.1.6.6 Redis (Managed — Upstash/Fly/Render)

Start with a small single node; keep private; enable persistence only if required; scale tier/throughput with load.



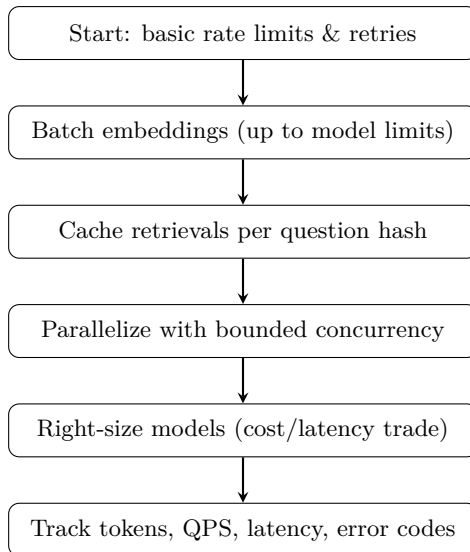
15.1.6.7 Object Storage (Supabase Storage, S3-compatible)

Direct client uploads via signed URLs; scale via provider tier; use lifecycle rules for cost.



15.1.6.8 LLM (OpenAI API)

Enforce rate limits/backoff; batch embeddings; cache retrievals per question hash to reduce egress.



15.1.7 Observability & Ops

15.1.7.1 Tooling options.

- **Logging**
 - *Cloud-native*: Grafana Cloud Logs (Loki), Datadog Logs, New Relic Logs.
 - *Self-hosted*: Loki + Promtail + Grafana. App logs via `pino` (JSON) with `request-id`, `job-id`, `org-id`; redact PII at source.
- **Metrics**
 - *Cloud-native*: Grafana Cloud Metrics (Prometheus remote-write), Datadog Metrics.

- *Self-hosted*: Prometheus + Grafana; export with `prom-client` (Node), Postgres exporter for DB, custom queue metrics.

- **Tracing**

- *Cloud-native*: Grafana Tempo Cloud, Sentry Performance, Datadog APM.
- *Self-hosted*: OpenTelemetry SDK (Node) → OTel Collector → Tempo (+ Grafana).
Propagate `traceparent` across API ↔ worker.

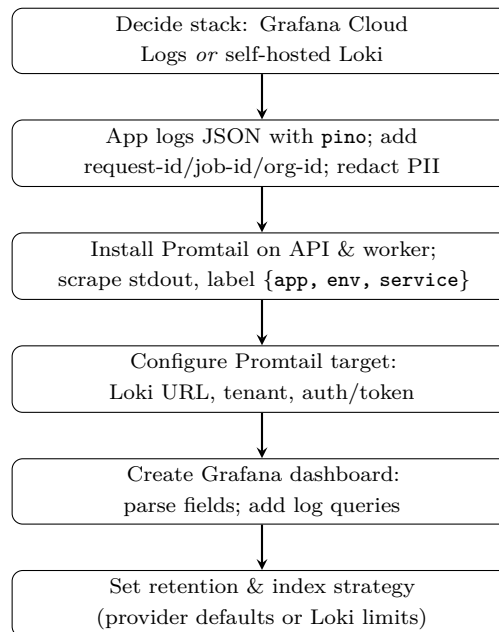
- **Alerting / On-call**

- *Cloud-native*: Grafana Cloud Alerting, Datadog Monitors, Sentry Alerts, PagerDuty.
- *Self-hosted*: Alertmanager (Prometheus) → Slack/PagerDuty. SLOs: API p95, queue latency, OpenAI error rate, DB KNN durations.

15.1.7.2 Logging (Loki stack)

This self-hosted stack is great for dev/staging. In production we can swap Loki/Grafana for managed (Grafana Cloud Logs) with Promtail agents. Works with Node (NestJS API + BullMQ workers) printing JSON via pino

Setup Workflow



Listing 82: docker-compose.yml (Loki + Promtail + Grafana)

```

1 version: "3.8"
2
3 services:
4   loki:
5     image: grafana/loki:2.9.4
6     command: [ "-config.file=/etc/loki/config.yml" ]

```

```

7   ports: [ "3100:3100" ]
8   volumes:
9     - ./loki:/etc/loki
10    - loki-data:/loki
11    restart: unless-stopped
12
13  promtail:
14    image: grafana/promtail:2.9.4
15    command: [ "-config.file=/etc/promtail/config.yml" ]
16    volumes:
17      - ./promtail:/etc/promtail
18      - /var/log:/var/log:ro
19      - /var/lib/docker/containers:/var/lib/docker/containers:ro
20      - /var/run/docker.sock:/var/run/docker.sock
21    depends_on: [ loki ]
22    restart: unless-stopped
23
24  grafana:
25    image: grafana/grafana:10.4.5
26    ports: [ "3000:3000" ]
27    environment:
28      - GF_SECURITY_ADMIN_USER=admin
29      - GF_SECURITY_ADMIN_PASSWORD=admin
30    volumes:
31      - grafana-data:/var/lib/grafana
32    depends_on: [ loki ]
33    restart: unless-stopped
34
35  volumes:
36    loki-data:
37    grafana-data:

```

Loki configuration (./loki/config.yml).

Single-binary with *boltdb-shipper* and local filesystem storage. Seven-day retention for dev/staging. In production, prefer object storage with compactor + retention policies, or a managed Loki.

Listing 83: Loki config (boltdb-shipper, local FS, 7d retention)

```

1  auth_enabled: false # put behind reverse-proxy if exposed
2
3  server:
4    http_listen_port: 3100
5
6  common:

```

```

7   instance_addr: 127.0.0.1
8   path_prefix: /loki
9   storage:
10    filesystem:
11     chunks_directory: /loki/chunks
12     rules_directory: /loki/rules
13 replication_factor: 1
14 ring:
15    kvstore:
16     store: inmemory
17
18 schema_config:
19    configs:
20     - from: 2024-01-01
21       store: boltdb-shipper
22       object_store: filesystem
23       schema: v13
24       index:
25        prefix: index_
26        period: 24h
27
28 compactor:
29    working_directory: /loki/compactor
30    compactor_ring:
31     kvstore:
32      store: inmemory
33    retention_enabled: true
34
35 limits_config:
36    reject_old_samples: true
37    reject_old_samples_max_age: 168h # 7 days
38    retention_period: 168h
39
40 ruler:
41    alertmanager_url: http://localhost:9093
42
43 analytics:
44    reporting_enabled: false

```

Promtail configuration (./promtail/config.yml).

Scrapes Docker container logs and system logs; forwards to Loki. Labels align with our services (e.g., `service=api|worker|web`).

Listing 84: Promtail config (Docker + system logs, safe labels)

```

1 server:
2   http_listen_port: 9080
3   grpc_listen_port: 0
4
5 positions:
6   filename: /tmp/positions.yaml
7
8 clients:
9   - url: http://loki:3100/loki/api/v1/push
10     # basic_auth:
11     #   username: loki
12     #   password: secret
13
14 scrape_configs:
15   # --- Docker containers (JSON) ---
16   - job_name: docker
17     docker_sd_configs:
18       - host: unix:///var/run/docker.sock
19         refresh_interval: 5s
20     pipeline_stages:
21       - json:
22         expressions:
23           log: log
24           stream: stream
25           time: time
26         timestamp:
27           source: time
28           format: RFC3339Nano
29           fallback_formats: [ RFC3339 ]
30       - labels:
31         image: __meta_docker_image
32       - output:
33         source: log
34     relabel_configs:
35       - source_labels: [__meta_docker_container_name]
36         target_label: container
37         regex: "/(.*)"
38       - source_labels: [__meta_docker_container_log_stream]
39         target_label: stream
40       - source_labels: [__meta_docker_container_label_com_docker_compose_service]
41         target_label: service
42       - target_label: app
43         replacement: api # change per host (api/worker/web)
44

```



```

45 # --- System logs example ---
46 - job_name: syslog
47   static_configs:
48     - targets: [ localhost ]
49     labels:
50       job: syslog
51       __path__: /var/log/*.log

```

Node app logging (JSON with pino + request-id/org-id).

Attach `pino-http`; generate/propagate `x-request-id`; attach `orgId` (from JWT guard in the real API). In our architecture, API & worker *print JSON to stdout*, which Promtail scrapes.

Listing 85: pino-http setup with redaction & request-id

```

1 import pinoHttp from 'pino-http';
2 import { randomUUID } from 'crypto';
3
4 export const httpLogger = pinoHttp({
5   redact: { paths: ['req.headers.authorization', 'user.password'], censor: '[REDACTED]' },
6   genReqId: (req) => (req.headers['x-request-id'] as string) || randomUUID(),
7   customProps: (req) => ({
8     service: 'api',
9     env: process.env.NODE_ENV || 'dev',
10    orgId: (req as any).orgId ?? 'unknown',
11  }),
12  transport: process.env.NODE_ENV === 'development'
13    ? { target: 'pino-pretty', options: { colorize: true } }
14    : undefined
15 });

```

Listing 86: Express wiring (derive orgId; emit health)

```

1 // server.ts
2 import express from 'express';
3 import { httpLogger } from './logger';
4
5 const app = express();
6
7 app.use((req, _res, next) => {
8   // In real app, derive from JWT (AuthGuard) and set req.user/orgId.
9   (req as any).orgId = req.header('x-org-id') || 'unknown';
10  next();
11 });
12
13 app.use(httpLogger);

```

```

14
15 app.get('/health', (_req, res) => res.json({ ok: true }));
16
17 app.listen(3000, () => console.log('API on :3000'));

```

Grafana → Loki wiring.

1. Open Grafana at <http://localhost:3000> (admin/admin).
2. **Add Data Source** → Loki → URL: <http://loki:3100> → *Save & Test*.
3. Create a dashboard and add a *Logs* panel (query examples below).

LogQL examples.

Listing 87: Recent logs for API service

```

1 {service="api"} | json | line_format "{{.message}}"

```

Listing 88: Filter by orgId and requestId

```

1 {service="api"} | json | orgId="acme-co" | requestId="d7c4..."

```

Listing 89: Error count by container over 5m

```

1 sum by (container) (count_over_time({service="api"} |= "error" [5m]))

```

Listing 90: Latency histograms if we log durationMs

```

1 sum by (route) (rate(({service="api"} | json | unwrap durationMs)[5m]))

```

Security & multi-tenancy notes.

- Put Loki behind TLS and a reverse proxy (nginx/traefik). If multi-tenant, enable `auth_enabled: true` and use `X-Scope-OrgID`.
- Avoid high-cardinality labels (e.g., raw user IDs). Parse as fields with `| json` and filter at query time instead.

Quick push (without Promtail).

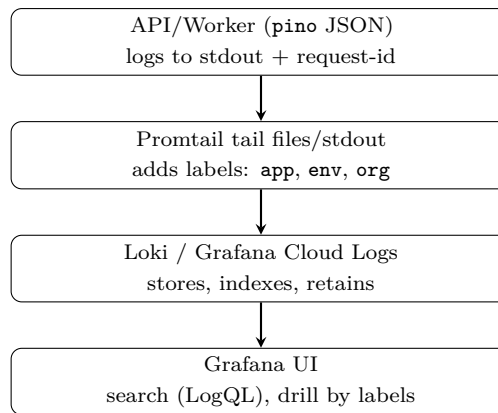
Listing 91: POST a test log to Loki directly

```

1 curl -X POST "http://localhost:3100/loki/api/v1/push" \
2   -H "Content-Type: application/json" \
3   --data-raw '{
4     "streams": [{
5       "stream": { "app":"api", "env":"dev" },
6       "values": [[ "'$(date +%s%N)'" , "{\"level\":\"info\",\"msg\":\"hello from curl\"}"
7     ]]]
8   }'

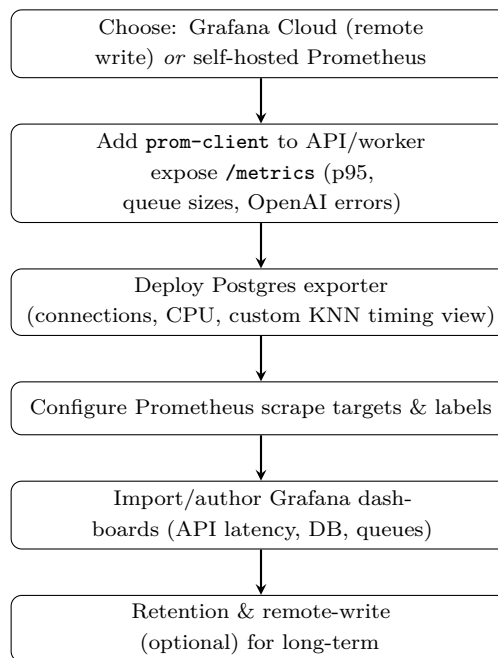
```

Logging — runtime data flow.



15.1.7.3 Metrics (Prometheus)

Setup Workflow



This configuration provides a production-lean, self-hosted Prometheus stack aligned with our architecture (NestJS API, BullMQ workers, Supabase PostgreSQL, optional cAdvisor/node-exporter). Services expose `/metrics` via `prom-client`; Prometheus scrapes them; Grafana visualizes.

`docker-compose.yml` — Prometheus, Grafana, exporters.

Listing 92: `docker-compose.yml`

```
1 version: "3.9"
2
3 services:
4   prometheus:
5     image: prom/prometheus:v2.53.0
```

```

6   command:
7     - --config.file=/etc/prometheus/prometheus.yml
8     - --web.enable-lifecycle
9   ports: ["9090:9090"]
10  volumes:
11    - ./prometheus:/etc/prometheus
12    - prom-data:/prometheus
13  restart: unless-stopped
14
15  grafana:
16    image: grafana/grafana:10.4.6
17    ports: ["3000:3000"]
18    environment:
19      GF_SECURITY_ADMIN_USER: admin
20      GF_SECURITY_ADMIN_PASSWORD: admin
21    volumes:
22      - graf-data:/var/lib/grafana
23    restart: unless-stopped
24    depends_on: [prometheus]
25
26  # PostgreSQL exporter (targets Supabase/Postgres)
27  postgres_exporter:
28    image: prometheuscommunity/postgres-exporter:v0.15.0
29    environment:
30      DATA_SOURCE_NAME: ${PG_EXPORTER_DSN}
31      # PG_EXPORTER_EXTEND_QUERY_PATH: /etc/queries/queries.yaml
32    volumes:
33      - ./postgres_exporter:/etc/queries:ro
34    restart: unless-stopped
35
36  # Optional: cAdvisor (container metrics on a single host)
37  cadvisor:
38    image: gcr.io/cadvisor/cadvisor:v0.47.2
39    ports: ["8080:8080"]
40    volumes:
41      - /:/rootfs:ro
42      - /var/run:/var/run:ro
43      - /sys:/sys:ro
44      - /var/lib/docker:/var/lib/docker:ro
45    restart: unless-stopped
46
47  # Optional: Node exporter for host metrics
48  node_exporter:
49    image: prom/node-exporter:v1.8.2

```

```

50     pid: host
51     network_mode: host
52     restart: unless-stopped
53
54 volumes:
55     prom-data:
56     graf-data:

```

Env note. Set a read-only DSN for the exporter:

```
export PG_EXPORTER_DSN="postgres://user:password@host:6543/postgres?sslmode=require".
```

prometheus.yml — scrape configs, labels, rules.

Listing 93: prometheus/prometheus.yml

```

1 global:
2     scrape_interval: 15s
3     evaluation_interval: 30s
4     external_labels:
5         cluster: "prod"
6
7 rule_files:
8     - rules.yml
9
10 scrape_configs:
11     # --- API (NestJS) ---
12     - job_name: "api"
13       metrics_path: /metrics
14       static_configs:
15         - targets: ["api.internal:3001"]    # private addr or public URL
16           labels:
17             service: api
18             env: prod
19
20     # --- Worker (BullMQ) ---
21     - job_name: "worker"
22       metrics_path: /metrics
23       static_configs:
24         - targets: ["worker.internal:3002"]
25           labels:
26             service: worker
27             env: prod
28
29     # --- Postgres exporter ---
30     - job_name: "postgres"
31       static_configs:

```

```

32     - targets: ["postgres_exporter:9187"]
33     labels:
34         service: postgres
35         env: prod
36
37 # --- cAdvisor (optional) ---
38 - job_name: "cadvisor"
39     static_configs:
40         - targets: ["cadvisor:8080"]
41         labels:
42             service: cadvisor
43             env: prod
44
45 # --- Node exporter (optional) ---
46 - job_name: "node"
47     static_configs:
48         - targets: ["localhost:9100"]
49         labels:
50             service: node
51             env: prod

```

Listing 94: prometheus/rules.yml — recording & alert rules

```

1 groups:
2 - name: recording
3     interval: 30s
4     rules:
5         # API p95 latency from Histogram over 5m
6         - record: job:http_p95_ms:5m
7           expr: |
8               histogram_quantile(
9                   0.95,
10                  sum by (le) (rate(http_request_duration_ms_bucket{service="api"}[5m]))
11                  ) * 1.0
12
13         # Worker queue size (example Gauge)
14         - record: job:queue_waiting:sum
15           expr: sum(worker_queue_waiting)
16
17 - name: alerts
18     rules:
19         - alert: ApiHighErrorRate
20           expr: sum(rate(http_requests_total{service="api",code=~"5.."}[5m])) /
21                sum(rate(http_requests_total{service="api"}[5m])) > 0.05
22         for: 10m

```

```

23     labels: { severity: page }
24     annotations:
25         summary: "API 5xx error rate > 5%"
26
27     - alert: PostgresConnectionsHigh
28       expr: pg_stat_activity_count{datname=~".+"} > 0.8 * pg_settings_max_connections
29       for: 10m
30       labels: { severity: warn }
31       annotations:
32         summary: "Postgres nearing max connections"

```

API /metrics (NestJS/Express) via prom-client.

Listing 95: apps/api/src/metrics.ts

```

1  import client from 'prom-client';
2  import type { Request, Response } from 'express';
3
4  client.collectDefaultMetrics({ prefix: 'api_' });
5
6  export const httpReqCounter = new client.Counter({
7      name: 'http_requests_total',
8      help: 'HTTP request count',
9      labelNames: ['method', 'route', 'code'],
10 });
11
12 export const httpDuration = new client.Histogram({
13     name: 'http_request_duration_ms',
14     help: 'HTTP request duration (ms)',
15     buckets: [5, 10, 25, 50, 100, 250, 500, 1000, 2500, 5000],
16     labelNames: ['method', 'route', 'code'],
17 });
18
19 export const openaiErrors = new client.Counter({
20     name: 'openai_api_errors_total',
21     help: 'OpenAI API error count',
22     labelNames: ['operation', 'status'],
23 });
24
25 export function metricsHandler(_req: Request, res: Response) {
26     res.set('Content-Type', client.register.contentType);
27     res.end(client.register.metrics());
28 }
29
30 // Express/Nest adapter example (Express)
31 export function promMiddleware(routeKey: (req: Request) => string) {

```

```

32   return async (req: Request, res: Response, next: Function) => {
33       const end = httpDuration.startTimer({ method: req.method, route: routeKey(req) });
34       res.on('finish', () => {
35           httpReqCounter.inc({ method: req.method, route: routeKey(req), code: String(res.
statusCode) });
36           end({ code: String(res.statusCode) });
37       });
38       next();
39   };
40 }

```

Listing 96: apps/api/src/server.ts — wiring /metrics

```

1  import express from 'express';
2  import { metricsHandler, promMiddleware } from './metrics';
3  import { openaiErrors } from './metrics';
4
5  const app = express();
6
7  // Minimal route keyer (avoid exploding cardinality)
8  const routeKey = (req: any) => (req.route?.path || req.path || 'unknown');
9
10 // Apply before our routes
11 app.use(promMiddleware(routeKey));
12
13 app.get('/metrics', metricsHandler);
14
15 // Example: count OpenAI failures
16 async function callOpenAI() {
17     try {
18         // ... OpenAI call ...
19     } catch (e: any) {
20         openaiErrors.inc({ operation: 'embeddings', status: String(e?.status || 500) });
21         throw e;
22     }
23 }
24
25 app.listen(3001, () => console.log('API on :3001'));

```

Worker metrics (BullMQ queue sizes).

Listing 97: apps/worker/src/metrics.ts

```

1  import client from 'prom-client';
2  import { Queue } from 'bullmq';
3

```



```

4 client.collectDefaultMetrics({ prefix: 'worker_' });
5
6 export const queueWaiting = new client.Gauge({
7   name: 'worker_queue_waiting',
8   help: 'Number of waiting jobs',
9   labelNames: ['queue'],
10 });
11 export const queueActive = new client.Gauge({
12   name: 'worker_queue_active',
13   help: 'Number of active jobs',
14   labelNames: ['queue'],
15 });
16
17 export async function pollQueues(queues: Record<string, Queue>) {
18   for (const [name, q] of Object.entries(queues)) {
19     const [w, a] = await Promise.all([q.getWaitingCount(), q.getActiveCount()]);
20     queueWaiting.set({ queue: name }, w);
21     queueActive.set({ queue: name }, a);
22   }
23 }

```

Listing 98: apps/worker/src/main.ts — expose /metrics

```

1 import http from 'http';
2 import client from 'prom-client';
3 import { queues } from './queue';
4 import { pollQueues } from './metrics';
5
6 // Simple /metrics server
7 const srv = http.createServer(async (req, res) => {
8   if (req.url === '/metrics') {
9     await pollQueues({ parse: queues.parseDoc.q, embed: queues.embedChunks.q, score:
10       queues.scoreMatches.q });
11     res.writeHead(200, { 'Content-Type': client.register.contentType });
12     res.end(await client.register.metrics());
13   } else {
14     res.writeHead(404); res.end();
15   }
16 });
17 srv.listen(3002);

```

Postgres exporter — optional custom query (pgvector visibility).

Listing 99: postgres_exporter/queries.yaml

```

1 pgvector_knn:

```

```

2 query: |
3     SELECT
4         sum(total_time) AS total_ms,
5         sum(calls) AS calls
6     FROM pg_stat_statements
7     WHERE query ILIKE '%<=>%' OR query ILIKE '%vector%';
8 metrics:
9     - total_ms:
10         usage: "GAUGE"
11         description: "Total time spent in vector/knn-like queries (ms)"
12     - calls:
13         usage: "GAUGE"
14         description: "Number of vector/knn-like calls"

```

Add environment variable: `PG_EXPORTER_EXTEND_QUERY_PATH=/etc/queries/queries.yaml`.

Grafana — add data source & import dashboards.

Listing 100: Grafana quick start

```

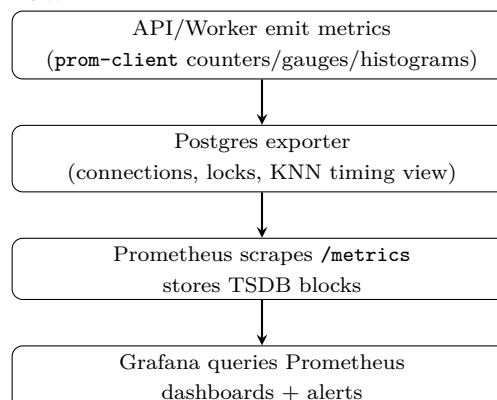
1 # open http://localhost:3000 (admin/admin)
2 # Add Data Source -> Prometheus -> URL: http://prometheus:9090 -> Save & Test
3 # Import dashboards:
4 # - 3662 (Node Exporter Full)
5 # - 19002 (Postgres Exporter)
6 # - Create a custom API/Worker dashboard using:
7 #   job:http_p95_ms:5m, http_requests_total, worker_queue_waiting

```

Security & production.

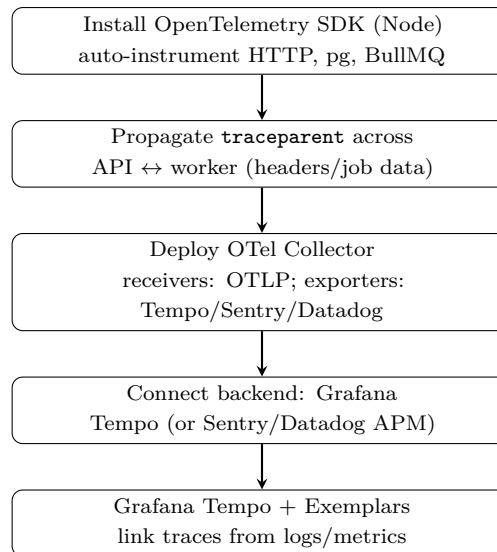
- Place Prometheus behind a private network/VPN or a reverse proxy with auth (and HTTPS).
- Use *read-only* DB credentials for the Postgres exporter.
- Keep label cardinality low (route templates, not raw paths/IDs).
- For long-term retention, enable `remote_write` (Grafana Cloud, Thanos, or VictoriaMetrics).

Metrics — runtime data flow.



15.1.7.4 Tracing (OpenTelemetry)

Setup Workflow



`docker-compose.yml` (Tempo + Collector + Grafana)

Listing 101: `docker-compose.yml`

```
1 version: "3.8"
2
3 services:
4   tempo:
5     image: grafana/tempo:2.5.0
6     command: [ "-config.file=/etc/tempo.yaml" ]
7     volumes:
8       - ./tempo/tempo.yaml:/etc/tempo.yaml:ro
9       - tempo-data:/var/tempo
10    ports: [ "3200:3200" ] # Tempo HTTP API for Grafana
11    restart: unless-stopped
12
13   otel-collector:
14     image: otel/opentelemetry-collector:0.101.0
15     command: [ "--config=/etc/otelcol.yaml" ]
16     volumes:
17       - ./otel/otelcol.yaml:/etc/otelcol.yaml:ro
18     ports:
19       - "4317:4317" # OTLP gRPC
20       - "4318:4318" # OTLP HTTP
21       - "8888:8888" # Collector metrics/debug
22     depends_on: [ tempo ]
23     restart: unless-stopped
24
```

```

25 grafana:
26     image: grafana/grafana:10.4.5
27     environment:
28         - GF_SECURITY_ADMIN_USER=admin
29         - GF_SECURITY_ADMIN_PASSWORD=admin    # change in prod
30     ports: [ "3000:3000" ]
31     volumes:
32         - grafana-data:/var/lib/grafana
33     depends_on: [ tempo ]
34     restart: unless-stopped
35
36 volumes:
37     tempo-data:
38     grafana-data:

```

Tempo config — ./tempo/tempo.yaml

Listing 102: ./tempo/tempo.yaml

```

1 server:
2     http_listen_port: 3200
3
4 distributor:
5     receivers:
6         otlp:
7             protocols:
8                 http:
9                 grpc:
10
11 compactor:
12     compaction:
13         block_retention: 168h # 7 days
14
15 storage:
16     trace:
17         backend: local
18         local:
19             path: /var/tempo

```

Collector config — ./otel/otelcol.yaml

Listing 103: ./otel/otelcol.yaml

```

1 receivers:
2     otlp:
3         protocols:

```

```

4     http:
5     grpc:
6
7 processors:
8     memory_limiter:
9         check_interval: 1s
10        limit_percentage: 75
11        spike_limit_percentage: 15
12    batch:
13        send_batch_size: 1024
14        timeout: 5s
15    resource:
16        attributes:
17            - key: service.environment
18              action: upsert
19              value: dev # set per environment
20
21 exporters:
22     otlphttp/tempo:
23         endpoint: http://tempo:4318
24         tls:
25             insecure: true
26
27 extensions:
28     health_check: {}
29
30 service:
31     extensions: [health_check]
32     pipelines:
33         traces:
34             receivers: [otlp]
35             processors: [memory_limiter, batch, resource]
36             exporters: [otlphttp/tempo]

```

App instrumentation (Node) — API (NestJS/Express)

```

1 pnpm add @opentelemetry/sdk-node \
2     @opentelemetry/auto-instrumentations-node \
3     @opentelemetry/exporter-trace-otlp-http \
4     @opentelemetry/propagator-w3c \
5     @opentelemetry/instrumentation-express \
6     @opentelemetry/instrumentation-http \
7     @opentelemetry/instrumentation-pg

```

Listing 104: apps/api/src/otel.ts

```

1 import { NodeSDK } from '@opentelemetry/sdk-node';
2 import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
3 import { W3CTraceContextPropagator } from '@opentelemetry/core';
4 import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
5 import { Resource } from '@opentelemetry/resources';
6
7 const exporter = new OTLPTraceExporter({
8   url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
9 });
10
11 const sdk = new NodeSDK({
12   traceExporter: exporter,
13   textMapPropagator: new W3CTraceContextPropagator(),
14   instrumentations: getNodeAutoInstrumentations({
15     '@opentelemetry/instrumentation-http': { enabled: true },
16     '@opentelemetry/instrumentation-express': { enabled: true },
17     '@opentelemetry/instrumentation-pg': { enabled: true },
18   }),
19   resource: new Resource({
20     'service.name': 'api',
21     'service.namespace': 'bidion',
22   }),
23 });
24
25 sdk.start();
26 console.log('[OTel] API tracing enabled');
27 process.on('SIGTERM', () => sdk.shutdown());

```

Listing 105: API env

```

1 export OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4318

```

App instrumentation (Node) — Worker (BullMQ manual spans)

Listing 106: apps/api/src/queues.ts (inject traceparent)

```

1 import { context, propagation } from '@opentelemetry/api';
2 import { Queue } from 'bullmq';
3
4 export async function enqueueScore(rfpId: string, orgId: string, q: Queue) {
5   const carrier: Record<string, string> = {};
6   propagation.inject(context.active(), carrier); // writes traceparent
7   await q.add('score', { rfpId, orgId, otel: carrier }, { attempts: 5 });
8 }

```

Listing 107: apps/worker/src/otel.ts

```

1 import { NodeSDK } from '@opentelemetry/sdk-node';
2 import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
3 import { W3CTraceContextPropagator } from '@opentelemetry/core';
4 import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
5 import { Resource } from '@opentelemetry/resources';
6
7 const exporter = new OTLPTraceExporter({
8   url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
9 });
10
11 export const workerSdk = new NodeSDK({
12   traceExporter: exporter,
13   textMapPropagator: new W3CTraceContextPropagator(),
14   instrumentations: getNodeAutoInstrumentations({}),
15   resource: new Resource({
16     'service.name': 'worker',
17     'service.namespace': 'bidion',
18   }),
19 });
20 workerSdk.start();

```

Listing 108: apps/worker/src/score.processor.ts (extract + span)

```

1 import { context, propagation, trace } from '@opentelemetry/api';
2
3 export async function handleScore(job: any) {
4   const carrier = (job.data && job.data.otel) || {};
5   const parentCtx = propagation.extract(context.active(), carrier);
6
7   return await context.with(parentCtx, async () => {
8     const tracer = trace.getTracer('worker');
9     return await tracer.startActiveSpan('score-matches', async (span) => {
10       try {
11         span.setAttribute('rfp.id', job.data.rfpId);
12         span.setAttribute('org.id', job.data.orgId);
13         // ... KNN, keyword/rules, write matches
14       } catch (e) {
15         span.recordException(e as Error);
16         span.setStatus({ code: 2, message: 'error' });
17         throw e;
18       } finally {
19         span.end();
20       }
21     });
22   });

```

```

22     });
23 }

```

Grafana — add Tempo data source

1. Open `http://localhost:3000` (admin/admin).
2. *Data Sources* → *Add data source* → **Tempo**.
3. URL: `http://tempo:3200` → *Save & Test*.
4. Explore: filter by `service.name ∈ {api, worker}`.

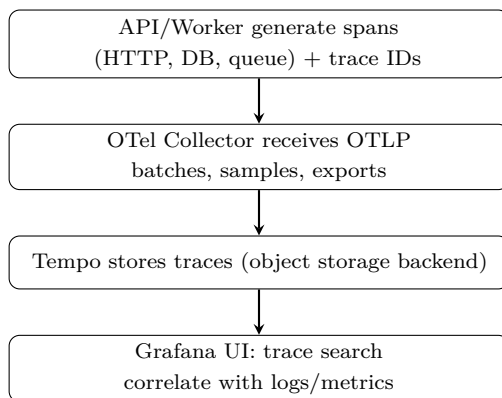
Log/Metric correlation (optional)

- Emit `trace_id/span_id` in logs (pino hook) to enable “View trace” from Grafana logs (Loki).
- Export latency histograms with exemplars (Prometheus) to deep-link slow bins to traces.

Security & tenancy

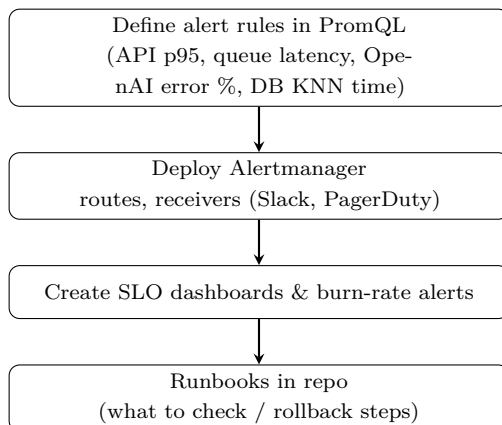
- Put Grafana/Tempo behind TLS and auth (reverse proxy or Grafana OAuth).
- Limit Collector ports (4317/4318) to private networks; do not expose publicly.
- Add `org.id` as a span/resource attribute for per-tenant filtering.

Tracing — runtime data flow.



15.1.7.5 Alerting / On-call

Setup Workflow.



Prometheus is already setup and scraping API, Worker, and Postgres exporters. This section adds: (1) recording/alert rules, (2) Alertmanager routing (Slack/PagerDuty), (3) SLO burn-rate alerts, and (4) on-call runbooks. We *do not* repeat Prometheus setup here.

Recording rules (helper series)

Files: `./prometheus/rules/recording.rules.yml`

Listing 109: `prometheus/rules/recording.rules.yml`

```
1 groups:
2 - name: recording.rules
3   interval: 30s
4   rules:
5     # API: request rate and error rate
6     - record: job:http_requests_total:rate1m
7       expr: sum by (service, route, method, status) (rate(http_requests_total[1m]))
8
9     - record: job:http_requests_errors_total:rate5m
10      expr: sum by (service) (rate(http_requests_total{status=~"5.."}[5m]))
11
12     # API: p95 latency over 5m (requires http_request_duration_seconds_bucket)
13     - record: job:http_request_p95_seconds:5m
14       expr: |
15         histogram_quantile(
16           0.95,
17           sum by (service, le) (rate(http_request_duration_seconds_bucket[5m]))
18         )
19
20     # Queue backlog / latency (export gauges from Worker)
21     - record: job:queue_latency_seconds:5m
22       expr: max by (queue, service) (bullmq_queue_latency_seconds)
23
24     - record: job:queue_backlog:5m
25       expr: max by (queue, service) (bullmq_queue_backlog)
26
27     # OpenAI error ratio (export totals+errors)
28     - record: job:openai_error_ratio:5m
29       expr: |
30         (sum by (service) (rate(openai_requests_errors_total[5m]))) /
31         clamp_min(sum by (service) (rate(openai_requests_total[5m])), 1))
32
33     # DB pgvector KNN p95 (requires db_knn_seconds_bucket)
34     - record: job:db_knn_p95_seconds:5m
35       expr: |
36         histogram_quantile(
37           0.95,
```

```
38         sum by (le) (rate(db_knn_seconds_bucket[5m]))
39     )
```

Alert rules (concrete thresholds)

Files: ./prometheus/rules/alerts.rules.yml

Listing 110: prometheus/rules/alerts.rules.yml

```
1 groups:
2 - name: service.alerts
3   interval: 30s
4   rules:
5   - alert: APIHighP95Latency
6     expr: job:http_request_p95_seconds:5m{service="api"} > 0.5
7     for: 10m
8     labels: {severity: page, team: platform, component: api}
9     annotations:
10      summary: "API p95 latency is high"
11      description: "p95 over last 5m is {{ $value | printf \"%.3f\" }}s (>0.5s). Check
12        slow routes/DB."
13
14 - alert: QueueBacklogHigh
15   expr: job:queue_backlog:5m{queue="rfp-jobs"} > 500
16   for: 15m
17   labels: {severity: page, team: data, component: worker}
18   annotations:
19     summary: "Queue backlog high (rfp-jobs)"
20     description: "Backlog > 500 for 15m. Scale workers or fix stuck jobs."
21
22 - alert: QueueLatencyHigh
23   expr: job:queue_latency_seconds:5m{queue="rfp-jobs"} > 60
24   for: 10m
25   labels: {severity: page, team: data, component: worker}
26   annotations:
27     summary: "Queue end-to-end latency high"
28     description: "Latency > 60s for 10m. Check Redis, concurrency, batch sizes."
29
30 - alert: OpenAIErrorRateHigh
31   expr: job:openai_error_ratio:5m{service="api"} > 0.1
32   for: 10m
33   labels: {severity: ticket, team: platform, component: api}
34   annotations:
35     summary: "OpenAI error ratio > 10%"
36     description: "5m failure ratio {{ $value | printf \"%.2f\" }}. Check keys, limits,
37       retries/backoff."
```

```

36
37 - alert: DBKNNLatencyHigh
38   expr: job:db_knn_p95_seconds:5m > 0.2
39   for: 15m
40   labels: {severity: ticket, team: data, component: postgres}
41   annotations:
42     summary: "DB KNN p95 > 200ms"
43     description: "pgvector KNN p95 {{ $value | printf \"%.3f\" }}s. Tune indexes/HNSW/
CPU/IO."

```

SLO Burn-rate alerts (multi-window, multi-burn)

Assume availability SLO: $\geq 99.9\%$ over 30 days (error budget $0.1\% = 0.001$).

Files: `./prometheus/rules/slo-burn.rules.yml`

Listing 111: `prometheus/rules/slo-burn.rules.yml`

```

1 groups:
2 - name: slo.burn.alerts
3   interval: 30s
4   rules:
5 - record: api:error_ratio:5m
6   expr: |
7     (sum(rate(http_requests_total{service="api",status=~"5.."}[5m])) /
8     clamp_min(sum(rate(http_requests_total{service="api"}[5m])), 1))
9 - record: api:error_ratio:30m
10  expr: |
11    (sum(rate(http_requests_total{service="api",status=~"5.."}[30m])) /
12    clamp_min(sum(rate(http_requests_total{service="api"}[30m])), 1))
13 - record: api:error_ratio:1h
14  expr: |
15    (sum(rate(http_requests_total{service="api",status=~"5.."}[1h])) /
16    clamp_min(sum(rate(http_requests_total{service="api"}[1h])), 1))
17 - record: api:error_ratio:6h
18  expr: |
19    (sum(rate(http_requests_total{service="api",status=~"5.."}[6h])) /
20    clamp_min(sum(rate(http_requests_total{service="api"}[6h])), 1))
21
22 # 99.9% SLO ⇒ budget = 0.001
23 - alert: APISLOFastBurn
24   expr: (api:error_ratio:5m > 14.4 * 0.001) and (api:error_ratio:1h > 14.4 * 0.001)
25   for: 5m
26   labels: {severity: page, slo: "api-availability-99.9-30d"}
27   annotations:
28     summary: "API SLO FAST burn (5m & 1h) > 14.4×"
29     description: "Rapid budget burn. 5m={{ $value | printf \"%.3f\" }}; confirm with 1h

```

```

30     . "
31 - alert: APISLOSlowBurn
32   expr: (api:error_ratio:30m > 6 * 0.001) and (api:error_ratio:6h > 6 * 0.001)
33   for: 15m
34   labels: {severity: ticket, slo: "api-availability-99.9-30d"}
35   annotations:
36     summary: "API SLO SLOW burn (30m & 6h) > 6x"
37     description: "Sustained degradation. Investigate; consider rollback/feature flags."

```

Alertmanager config (Slack, PagerDuty, routing)

Files: ./alertmanager/alertmanager.yml

Listing 112: alertmanager/alertmanager.yml

```

1 global:
2   resolve_timeout: 5m
3
4 route:
5   receiver: "slack-default"
6   group_by: ["alertname", "cluster", "service", "component"]
7   group_wait: 30s
8   group_interval: 5m
9   repeat_interval: 4h
10  routes:
11    - matchers: [severity="page"]
12      receiver: "pagerduty"
13      group_wait: 10s
14      repeat_interval: 1h
15    - matchers: [team="data"]
16      receiver: "slack-data"
17
18 receivers:
19   - name: "slack-default"
20     slack_configs:
21       - api_url: "${SLACK_WEBHOOK_URL}"
22         send_resolved: true
23         title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
24         text: |
25           *Summary:* {{ (index .Alerts 0).Annotations.summary }}
26           *Details:* {{ (index .Alerts 0).Annotations.description }}
27           *Labels:* {{ .CommonLabels }}
28           *Silence:* <{{ .ExternalURL }}/#/silences>
29           *Alerts:* {{ range .Alerts }} - {{ .Labels }} {{ end }}
30

```

```

31 - name: "slack-data"
32   slack_configs:
33     - api_url: "${SLACK_WEBHOOK_URL}"
34       channel: "#data-oncall"
35       send_resolved: true
36       title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
37       text: |
38         *Data Alert:* {{ (index .Alerts 0).Annotations.summary }}
39         *Details:* {{ (index .Alerts 0).Annotations.description }}
40
41 - name: "pagerduty"
42   pagerduty_configs:
43     - routing_key: "${PAGERDUTY_ROUTING_KEY}"
44       severity: "{{ if eq .CommonLabels.severity \"page\" }}critical{{ else }}error{{
45       end }}"
46       send_resolved: true
47
48 inhibit_rules:
49   - source_matchers: [severity="page"]
50     target_matchers: [severity="ticket"]
51     equal: ["alertname","service"]

```

App instrumentation (what to expose)

- **API:** http_request_duration_seconds_bucket, http_requests_total{status}
- **Worker/Queue:** bullmq_queue_latency_seconds, bullmq_queue_backlog
- **OpenAI:** openai_requests_total, openai_requests_errors_total
- **DB/KNN:** db_knn_seconds_bucket

Runbook examples (checked into repo)

Files: ./runbooks/APIHighP95Latency.md

Listing 113: runbooks/APIHighP95Latency.md

```

1 # APIHighP95Latency
2 ## Triage
3 1) Grafana API dashboard → "Latency p95 by route".
4 2) Check 5xx spikes, deploys (last 1h), DB CPU/IO.
5 ## Mitigation
6 - Scale API/Worker +1.
7 - Rollback last deploy (<30m).
8 - Toggle feature flags to reduce load.
9 ## Deep-dive
10 - Offending route labels.

```

```

11 - Postgres slow queries, pgvector indexes (HNSW params).
12 - OpenAI limits/errors.
13 ## Closure
14 - Postmortem issue with timeline + actions.

```

Files: ./runbooks/QueueBacklogHigh.md

Listing 114: runbooks/QueueBacklogHigh.md

```

1 # QueueBacklogHigh
2 ## Triage
3 1) Worker dashboard → backlog & latency.
4 2) Redis health, BullMQ concurrency, stalled jobs.
5 ## Mitigation
6 - +2 temp concurrency.
7 - Drain DLQ/poison.
8 - Pause producers for stuck job types.
9 ## Deep-dive
10 - Recent changes to parse-doc/embed-chunks/score-matches.
11 - Redis CPU/mem, worker node network.
12 ## Closure
13 - Tune batch size/retries; add per-job timeouts.

```

Validation & Ops commands

Listing 115: Validate rules and AM config

```

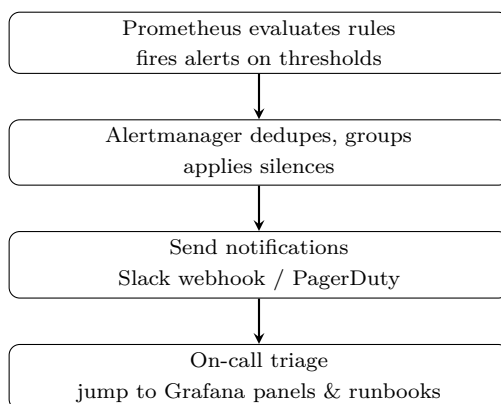
1 # Validate Prom rules:
2 docker run --rm -v $PWD/prometheus:/etc/prom prom/prometheus \
3   promtool check rules /etc/prom/rules/*.yaml
4
5 # Validate Alertmanager config:
6 docker run --rm -v $PWD/alertmanager:/etc/am prom/alertmanager:latest \
7   amtool check-config /etc/am/alertmanager.yaml
8
9 # Reload Prometheus (requires --web.enable-lifecycle already set in Prom setup):
10 curl -X POST http://localhost:9090/-/reload

```

Notes & Threshold Tuning

- Page only for user-visible impact; file tickets for noise.
- Calibrate API p95 by historical p95/p99; 500 ms is a placeholder.
- For KNN latency, watch p95/p99 under load; consider HNSW and `work_mem`.
- Align SLOs with business impact (e.g., 99.9% vs. 99.5%).

Alerting / On-call — runtime data flow.



Operational Remarks.

- **PII hygiene:** redact at the app (`pino redact`), not only in sinks. Avoid logging request bodies unless necessary.
- **Correlation:** always stamp `x-request-id` and propagate `traceparent`; include IDs in logs and (sparingly) as Prometheus labels.
- **DB visibility:** create a tiny view that times KNN queries and export as a Prometheus gauge; directly tracks vector search health.
- **Cost control:** start managed (Grafana Cloud, Sentry). If we outgrow, move to self-hosted Loki/Prometheus/Tempo with object storage.
- **Runbooks:** keep Markdown runbooks in-repo and link them in alert annotations (`runbook_url`) to reduce MTTR.

15.1.8 Upgrade Paths (Post-MVP)

- Move API/Workers from Fly/Render to AWS ECS/Fargate (or Kubernetes) when we need VPC/IAM controls or higher scale.
- Switch `ivfflat` to `HNSW` in `pgvector` for better recall/latency; add read replicas if KNN load increases.
- Introduce a feature-flag service and per-tenant rate limits; add SSO (OIDC) and scoped API tokens.

15.2 Scaling Plan (MVP → v1)

Goals.

Sustain *millions of users / org seats* with predictable SLOs (API p95 ≤ 500 ms; KNN p95 ≤ 200 ms under nominal load), while keeping cost linear with usage. *Planning envelope (for 1M users):* peak **1,000–1,500 RPS** on API, **10–30M** total embedding vectors (3,072 dims), **3–5k jobs/min** worker throughput, primary DB **0.5–1.5 TB**.

15.2.1 Horizontally scale stateless tiers first.

API (NestJS) — Replicas, Concurrency, Pooling

Scale behind an LB. Start *6–10 replicas* (1 vCPU, 1–2 GB) and HPA on p95/RPS. Connection pool via `pgbouncer` (`pool_size 150–250`, `max_client_conn 2,000`), keep per-pod DB connections low (*20–30 MAX_CONNECTIONS* / instance).

| Variable | 1M Target | Notes |
|--|---------------------------------|-------------------------------------|
| Replicas | 6–10 pods | Behind LB; HPA on p95/RPS |
| Pod size | 1 vCPU / 1–2 GB | Increase only if CPU-bound |
| Per-pod DB conns | 20–30 | Keep total active < 500 via pooling |
| pgBouncer <code>pool_size</code> | 150–250 | Transaction pooling mode |
| pgBouncer <code>max_client_conn</code> | 2,000 | Headroom for spikes |
| HTTP timeouts | 3–5 s | With retries + circuit breakers |
| Health/readiness | <code>/healthz</code> every 10s | Max surge 1, max unavailable 0 |

Workers (BullMQ) — Concurrency, Autoscale, Limits

Separate deployments per queue (`parse-doc`, `embed-chunks`, `score-matches`). Per-pod concurrency: *parse=4, embed=16, score=8*; autoscale replicas *4→20* on backlog > *1,000 jobs for 5 min*. Cap LLM egress ≤ 5 QPS/org and global *50–100 QPS*.

| Variable | 1M Target | Notes |
|---------------------|-------------------------------|--------------------------------------|
| Deployments | per-queue (parse/embed/score) | Isolate lanes to tune independently |
| Per-pod concurrency | parse=4; embed=16; score=8 | Cap to protect DB/LLM |
| Replicas | 4 → 20 | Scale when backlog > 1,000 for 5 min |
| Retry/backoff | exp. 250 ms → 5 s + jitter | Max 5 attempts; DLQ on fail |
| Per-org LLM QPS | ≤ 5 QPS | Noisy-neighbor guardrail |
| Global LLM QPS | 50–100 QPS | Align with provider quota/budget |
| Max inflight/org | 500 jobs | Backpressure before saturation |

Edge / CDN — Caching and Freshness

cache public GETs w/ `Cache-Control: public, max-age=120, s-maxage=300`. For app APIs, use `stale-while-revalidate=60` on safe endpoints (e.g., `cached /rfps/:id/matches`).

| Header/Setting | Value | Notes / Scope |
|----------------------------|--|---|
| Cache-Control (public GET) | <code>public, max-age=120, s-maxage=300</code> | Static/public assets |
| SWR (safe API reads) | <code>stale-while-revalidate=60</code> | e.g., <code>/rfps/:id/matches</code> (pre-computed) |
| ETag/If-None-Match | enabled | Validate freshness cheaply |
| Compression | gzip/br | Ensure on CDN + origin |
| CDN regions | multi-region POPs | Default provider footprint OK |

15.2.2 Postgres scaling path (Supabase DB).

| Knob | 1M Target | Notes |
|-------------------|------------------------------------|--|
| Instance size | 8 vCPU / 32 GB (to 16/64) | Vertical headroom first; watch CPU/IO wait < 5% |
| Provisioned IOPS | $\geq 10k$ | Ensure disk class supports target IOPS |
| Pooling mode | pgBouncer (transaction) | Set drivers <code>prepared_statements=false</code> if needed |
| Active backends | < 500 | Keep via pooling + low per-pod conns |
| Partitions (org) | HASH 32–64 | On <code>org_id</code> for large shared tables |
| Partitions (time) | MONTHLY | For <code>embeddings/chunks</code> |
| Read replicas | 1–2 | Route dashboards/analytics; lag < 2s |
| Backfills | 5k–20k rows/txn | Off-peak; throttle to keep p95 budget |
| Online reindex | REINDEX CONCURRENTLY | Avoid table locks |
| RLS strategy | shared table + <code>org_id</code> | Consider per-tenant partitions for hot orgs |

- **Vertical headroom first:** target *8 vCPU / 32 GB RAM* (scale to *16 vCPU / 64 GB* as needed), provisioned IOPS $\geq 10k$. Watch I/O wait (< 5%), WAL rate, autovacuum.
- **Connection pooling:** `pgbouncer` in *transaction* mode; set drivers `prepared_statements=false` if required. Keep total active backend conns < 500.
- **Partitioning:** partition large tables by `org_id` (HASH *32–64 partitions*) and/or time (MONTHLY for `embeddings/chunks`). Ensure constraint exclusion/pruning on queries.
- **Read replicas:** *1–2* replicas; route dashboards/analytics and read-heavy endpoints to replicas. Accept replica lag < 2s.
- **Online schema changes:** ADD COLUMN nullable, backfill batches (*5k–20k rows/txn*), REINDEX CONCURRENTLY. Avoid peak-time table locks.
- **RLS at scale:** shared tables with `org_id` + RLS (see §14); consider per-tenant partitions for hot orgs (< 5% causing > 50% load).

15.2.3 Vector search (pgvector) scale knobs.

| Parameter | 1M Target | Notes |
|----------------------|-----------------------------------|---|
| Index type | Start IVFFLAT; move to HNSW | Switch after write rate stabilizes |
| IVFFLAT lists | 2,000–4,000 | For 10–30M vectors |
| IVFFLAT probes | 8–16 | Balance recall vs latency |
| HNSW M | 16 | Out-degree; memory/recall tradeoff |
| HNSW ef_construction | 200 | Build-time accuracy |
| HNSW ef_search | 64–128 | Increase for higher recall |
| Query <i>k</i> | 50 | Candidates from ANN |
| Shortlist <i>m</i> | 200 | Re-rank domain scoring; keep KNN p95 ≤ 200 ms |
| Maintenance | ANALYZE nightly; VACUUM as needed | HNSW bulk-rebuilds off-peak |

- **Index choice:** start **IVFFLAT**; switch to **HNSW** once write rate stabilizes.
- **IVFFLAT:** `lists` $\approx 2,000\text{--}4,000$ for *10–30M* vectors; `probes` *8–16*.

- **HNSW**: $M=16$, $ef_construction=200$, $ef_search=64-128$ (increase for recall).
- **Recall/latency**: query $k = 50$; shortlist top $m = 200$ for re-rank (domain scoring) to keep KNN $p95 \leq 200$ ms.
- **Maintenance**: ANALYZE nightly; VACUUM as needed; schedule HNSW bulk rebuilds off-peak.

15.2.4 Queues, backpressure, and cost controls.

| Control | 1M Target | Notes |
|---------------------|-------------------------------------|--------------------------------------|
| Per-org rate limits | embed 60/min/org; score 120/min/org | Tune by SLO/cost |
| Global LLM QPS | 50–100 QPS | Respect provider quota |
| Max inflight/org | 500 jobs | Backpressure before saturation |
| Batch size (embeds) | 64–128 texts/call | Model-dependent max |
| Backoff | 250 ms \rightarrow 5 s + jitter | Max 5 attempts; circuit open 60 s |
| Idempotency keys | chunk_hash, document_id | matches: delete+insert pattern |
| Priority lanes | Interactive > bulk | Bulk backfills rate limit 10/min/org |

- **Backpressure**: per-queue rate limits (*embed 60/min/org, score 120/min/org*); max inflight per org *500*. Global caps aligned with LLM QPS budget.
- **Idempotency**: dedupe by `chunk_hash` and `document_id`; use *delete+insert* for `matches`.
- **LLM cost guards**: batch embeddings *64–128 texts/call*, exponential backoff (base *250 ms*, max *5 s*, jitter), circuit breaker open *60 s*.
- **Priority lanes**: interactive queues high-priority; bulk backfills low-priority with *rate limit 10/min/org*.

15.2.5 Caching layers.

| Layer | 1M Target | Notes |
|-----------------|----------------------|--------------------------------------|
| HTTP cache TTL | 60–300 s | CDN + ETag/Cache-Control |
| Redis cluster | 3 nodes, 3–6 GB each | Managed; private networking |
| Hot keys | < 5M total | Eviction policy: allkeys-lru |
| RFP matches TTL | 30–120 s | Memoize top matches per RFP |
| App LRU cache | 500–2,000 entries | Small reference data (weights/rules) |

- **HTTP cache**: CDN + ETag/Cache-Control; typical TTL *60–300 s*.
- **Redis**: 1 small cluster (*3 nodes, 3–6 GB each*); memoize top matches per RFP (*TTL 30–120 s*); keep hot key count < *5M* total.
- **App cache**: in-process LRU *500–2,000 entries* for small reference data (weights/rules).

15.2.6 Multi-region (v1+).

| Aspect | 1M Target | Notes |
|--------------------|-------------------------------|--|
| Stateless topology | Active/active per region | API/workers only |
| Database | 1 primary + 1–2 read replicas | Eventual consistency on non-critical reads |
| Org placement | Pin to home region | Reduce KNN latency (data gravity) |
| Routing | GeoDNS / latency-based | Sticky to home region |
| DR | PITR + cross-region backups | Failover rehearsal quarterly |

- **Topology:** active/active stateless per region; Postgres primary in one region + 1–2 read replicas elsewhere (eventual consistency for non-critical reads).
- **Data gravity:** pin orgs to a home region; route with GeoDNS.
- **DR:** PITR, cross-region backups; quarterly failover rehearsal.

15.2.7 Migrations and deploy safety.

| Control | 1M Target | Notes |
|--------------------|-------------------------------------|-----------------------------------|
| Artifact promotion | Same SHA (staging→prod) | Blue/green or rolling |
| Health gate | > 99% success over 5 min | Gate rollout by p95 + error rate |
| Safe schema change | Add NULLable; dual-write; dual-read | Backfill 5k–20k rows/txn off-peak |
| Feature flags | 5% → 25% → 100% | Instant rollback path |

- **Immutable artifacts:** promote same SHA (staging → prod); rolling/blue-green; pod P95 health checks (*success > 99% over 5 min*) gate rollout.
- **Safe migrations:** write to new nullable column, dual-read phase, async backfill (*5k–20k rows/txn*), flip reads, drop legacy.
- **Feature flags:** progressive rollout *5%→25%→100%*; instant rollback.

15.2.8 Observability & SLO enforcement.

| Signal / Policy | 1M Target | Action |
|------------------|-----------------------|---|
| API p95 | ≤ 500 ms | HPA scale-out; throttle producers |
| KNN p95 | ≤ 200 ms | Tune <i>k</i> , <i>m</i> , probes/ef_search; add replicas |
| Worker backlog | < 10k jobs | Scale workers first; then API |
| Burn-rate alerts | 5m/1h & 30m/6h | Page + autoscale; apply rate limits |
| Cache hit rate | > 80% (hot endpoints) | Increase TTLs; widen caching scope |
| Cost dashboards | Per-org LLM spend | Enforce QPS budgets |

- **Golden signals:** API latency/error rate, worker latency/throughput, DB CPU/IO/locks, KNN p95/p99.
- **Burn-rate alerts:** 5m/1h & 30m/6h (see Alerting). On breach: auto-scale workers, then API; throttle producers if backlog > 10k or p95 budget violated.
- **Cost dashboards:** LLM spend/org, embeddings/day, cache hit rates (*> 80% for hot endpoints*).

15.2.9 Data lifecycle.

| Data Class | 1M Target | Notes |
|--------------|------------------------|--|
| Deleted docs | Purge within 24 h | Remove embeddings/chunks + caches |
| Cold data | Archive after 90 d | Move to object storage (S3/Supabase) |
| GDPR deletes | Audit < 30 d | Hard-delete; background scrub of vectors |
| TTL policies | Keys expire by default | Prevent unbounded growth |

- **TTL/archival:** purge embeddings/chunks for deleted docs within *24 h*; archive cold data to object storage after *90 d*.
- **GDPR/tenant deletes:** hard-delete with background scrub of vectors and caches; audit in *< 30 d*.

Summary.

1. **Shed load before fall over:** global and per-org rate limits; circuit breakers for OpenAI/DB (open *60 s*, half-open after *10* successes).
2. **Thin request path:** precompute `matches`; `GET /rfps/:id/matches` must be read-only (*p95* $\leq 200\text{ ms}$ from *cache/replica*).
3. **Elasticity:** scale workers on backlog; API on RPS/p95; cap per-pod concurrency with budgets (embed *16*, score *8*).
4. **Partition first, shard later:** partitions (*32–64*) first; shard only if partitions + replicas are insufficient (hot orgs, TB-scale).
5. **Test:** k6 thresholds in CI (*p95* $< 500\text{ ms}$, fail rate $< 2\%$); nightly soak; chaos drills for Redis/DB/LLM brownouts.

15.3 Maintenance, Resilience & Recovery

15.3.1 Postgres: Backups, PITR, and Maintenance

Intuitively.

- **Backups are like photo albums.** We take regular “pictures” of the database so we can look back if something goes wrong.
- **PITR is a time machine.** Point-In-Time Recovery lets us rewind the database to *exactly* how it looked a few minutes ago—like scrubbing a video back to the moment before a mistake.
- **RPO = how much we can lose.** If our RPO is ≤ 5 minutes, the worst-case is we lose at most five minutes of new changes.
- **RTO = how long we’re down.** If our RTO is ≤ 60 minutes, that’s our goal for how fast we can restore and get the app talking to the database again.
- **Maintenance = chores.** `ANALYZE` is updating the map so Postgres knows where things are; `VACUUM` is taking out the trash. We schedule these chores so the house stays tidy and fast.

Rationale.

- **Why PITR?** Accidents happen—bad migrations, deletes, or bugs. PITR gives us a safe rewind button instead of starting from scratch.

- **Why RPO/RTO targets?** Clear targets tell us the *max data we might lose* and the *max time to recover*. They keep the team honest and drills measurable.
- **Why scheduled maintenance?** Small, regular cleanups (`VACUUM/ANALYZE`) beat rare, heavy cleanups. The app stays fast, and we avoid surprise slowdowns from bloat or stale stats.
- **Why verify with drills?** Practicing restores proves our backups are usable, our cutover steps work, and that we really meet the RPO/RTO we promised.

| Control | MVP Target | Notes / Basis |
|------------------------|------------------------------------|--|
| Backup model | Managed PITR enabled | Continuous WAL archiving + base backups (managed). [1] |
| RPO (data loss window) | ≤ 5 minutes | Supabase advertises near-real-time WAL shipping; verify during drills. [1] |
| RTO (restore time) | ≤ 60 minutes | Includes restore to timestamp + app re-point + checks. [2] |
| Retention | 7–14 days PITR | Balance cost vs. rollback needs. [1] |
| Maintenance windows | Weekly <code>VACUUM/ANALYZE</code> | Keep bloat low; schedule with <code>pg_cron</code> . [4, 3] |

15.3.1.1 Setup & Test PITR (Managed Supabase).

1. **Enable PITR in project settings.** In Supabase, turn on Point-in-Time Recovery (keeps continuous WAL + base backups). *Record the retention window.* [1]
2. **Create a drill marker.** Run a tiny, unique transaction so we can target a known timestamp:

```

1 -- Mark a drill time (note the server timestamp)
2 CREATE TABLE pitr_drill_mark(id int primary key, note text);
3 INSERT INTO pitr_drill_mark VALUES (1, 'drill: 2025-03-01T10:30Z');
4 SELECT NOW(); -- capture this as T_marker

```

3. **Simulate an error.** Drop the table or delete rows to mimic data loss:

```

1 DROP TABLE pitr_drill_mark;

```

4. **Restore to timestamp.** From Supabase PITR UI, *restore to a new database instance at T_marker + 1s.* (Postgres PITR replays WAL to a point-in-time.) [2]
5. **Validate.** Connect to the restored DB and confirm the marker exists:

```

1 SELECT * FROM pitr_drill_mark; -- should return the inserted row

```

6. **Cutover plan.** Prepare app `DATABASE_URL` switching (feature flag or secret rotate). Practice switching API/workers to the restored DB and back.

15.3.2 Postgres: Scheduling, Vacuum/Analyze, and Tuning

Intuitively.

Think of `pg_cron` as a calendar *inside Postgres* that runs SQL on a schedule.¹

¹`pg_cron`: cron-based job scheduler for Postgres & Supabase Cron (backed by `pg_cron`).

- **ANALYZE every night:** The database keeps a “map” of what our data looks like so it can find things fast. ANALYZE refreshes planner statistics so query plans stay accurate. Doing it nightly keeps searches speedy without much disruption.²
- **VACUUM (VERBOSE, ANALYZE) every week:** As rows get updated/deleted, little bits of trash pile up. VACUUM reclaims dead tuples (and with ANALYZE also refreshes the map). Weekly is a good routine: often enough to prevent bloat, not so often that it gets in the way.³

Rationale.

- **pg_cron jobs = automatic chores.** Schedule ANALYZE nightly (quick map update) and VACUUM weekly (deep clean) so the planner stays smart and tables don’t get bloated.⁴
- **Autovacuum per-table tuning = clean busy rooms more often.** Most rooms (tables) are fine with the default schedule—*autovacuum usually suffices*. For the few “messy” rooms with lots of changes, set stricter triggers so they get cleaned sooner—without making the whole house over-clean.⁵ Adjust per-table thresholds *only after telemetry shows bloat or stale stats* (watch `n_dead_tup`, `n_mod_since_analyze`).⁶
- **Operational constraint note.** VACUUM ordinarily requires the table *MAINTAIN* privilege (owners can vacuum their DB) and *cannot* be executed inside a transaction block.⁷

15.3.2.1 Pre-requisites (Supabase-managed Postgres)

```

1 -- Enable the scheduler in the target database
2 -- (Supabase Dashboard → Database → Extensions → pg_cron)
3 CREATE EXTENSION IF NOT EXISTS pg_cron;
4
5 -- Notes:
6 -- On Supabase, pg_cron jobs run under the built-in cron role.
7 -- Typically DO NOT need a separate maintenance role.
8 -- Prefer simple single-statement jobs for reliability (ANALYZE; VACUUM ...;).
```

15.3.2.2 Pre-requisites (Self-hosted Postgres: install & preload pg_cron)

Listing 116: Install & enable pg_cron (example: PostgreSQL 15 on Ubuntu/Debian)

```

1 # Install the extension package matching our PG major version
2 sudo apt update && sudo apt install postgresql-15-cron
3
4 # Edit postgresql.conf (shared_preload_libraries requires a restart)
5 # e.g., /etc/postgresql/15/main/postgresql.conf
```

²PostgreSQL ANALYZE docs.

³PostgreSQL VACUUM docs.

⁴pg_cron on GitHub & ANALYZE/VACUUM docs.

⁵Autovacuum guidance (RDS/Aurora, generally applicable).

⁶See PostgreSQL routine vacuuming guidance and when to tune (docs), and the statistics views/columns to monitor (docs).

⁷VACUUM privilege & transaction-block notes.

```

6 echo "shared_preload_libraries = 'pg_cron'" | sudo tee -a /etc/postgresql/15/main/
  postgresql.conf
7 echo "cron.database_name = 'your_database'" | sudo tee -a /etc/postgresql/15/main/
  postgresql.conf
8
9 # Restart Postgres to apply preload changes
10 sudo systemctl restart postgresql

```

Listing 117: Create the extension in the target database (self-hosted)

```

1 -- Connect to our target DB:
2 CREATE EXTENSION IF NOT EXISTS pg_cron;

```

15.3.2.3 Create jobs (nightly ANALYZE, weekly VACUUM)

```

1 -- Nightly stats refresh (02:30) - run ANALYZE for the current DB
2 SELECT cron.schedule(
3     'stats-analyze-nightly',
4     '30 2 * * *',
5     $$ ANALYZE; $$
6 );
7
8 -- Weekly vacuum for hot tables (Sunday 03:00) - start with public.matches
9 SELECT cron.schedule(
10     'vacuum-weekly-matches',
11     '0 3 * * 0',
12     $$ VACUUM (VERBOSE, ANALYZE) public.matches; $$
13 );

```

15.3.2.4 Force-run now (staging/CI validation)

```

1 -- List jobs to get jobid
2 SELECT jobid, jobname, schedule FROM cron.job ORDER BY jobid;
3
4 -- Trigger a run immediately
5 SELECT cron.run_job(<jobid>);
6
7 -- Inspect run history
8 SELECT jobid, status, return_message, start_time, end_time
9 FROM cron.job_run_details
10 ORDER BY start_time DESC
11 LIMIT 10;

```

15.3.2.5 Test: Observability queries (did it help?)

```

1  -- Table-level VACUUM/ANALYZE timestamps
2  SELECT relname, last_vacuum, last_autovacuum, last_analyze, last_autoanalyze
3  FROM pg_stat_user_tables
4  WHERE schemaname='public' AND relname IN ('matches')
5  ORDER BY relname;
6
7  -- Bloat/health proxy: live vs dead tuples, scans, n_mod_since_analyze
8  SELECT
9      relname,
10     n_live_tup, n_dead_tup,
11     n_mod_since_analyze,
12     vacuum_count, autovacuum_count,
13     analyze_count, autoanalyze_count
14 FROM pg_stat_user_tables
15 WHERE schemaname='public' AND relname IN ('matches');

```

15.3.2.6 Autovacuum tuning (targeted)

Apply per-table reloptions (only if churn is high).

```

1 ALTER TABLE public.matches
2     SET (autovacuum_vacuum_scale_factor = 0.05,  -- trigger vacuum at ~5% churn
3         autovacuum_analyze_scale_factor = 0.02); -- trigger analyze at ~2% churn

```

Verify reloptions are active.

```

1 SELECT relname, reloptions
2 FROM pg_class
3 WHERE relname='matches';

```

Monitor outcomes after load.

```

1 SELECT relname, n_dead_tup, vacuum_count, autovacuum_count,
2        analyze_count, autoanalyze_count, last_autovacuum, last_autoanalyze
3 FROM pg_stat_user_tables
4 WHERE relname='matches';

```

15.3.2.7 Alternative if pg_cron is unavailable (self-hosted)

Listing 118: OS cron / systemd timers calling psql

```

1 # crontab -e
2 30 2 * * * PGPASSWORD=... psql "$DATABASE_URL" -v ON_ERROR_STOP=1 -c "ANALYZE;"
3 0 3 * * 0 PGPASSWORD=... psql "$DATABASE_URL" -v ON_ERROR_STOP=1 \
4     -c "VACUUM (VERBOSE, ANALYZE) public.matches;"

```


15.3.2.8 Operational remarks (timeouts & safety)

- If a strict `statement_timeout` is configured, prefer targeted VACUUMs (specific tables) and keep maintenance windows off-peak.
- Start with the hottest tables (e.g., `public.matches`); expand scope as headroom allows.
- Prefer per-table autovacuum tuning before cluster-wide changes on managed platforms.

15.3.2.9 Passed-Results Variables (Verification Matrices)

Postgres

| Area | Check / Query | Pass Criteria (Expected) |
|---------------------------|--|--|
| pg_cron jobs exist | <code>SELECT jobid, jobname, schedule FROM cron.job;</code> | At least two rows: <code>stats-analyze-nightly @ 30 2 * * * and vacuum-weekly-matches @ 0 3 * * 0.</code> |
| pg_cron last run status | <code>SELECT status FROM cron.job_run_details ORDER BY start_time DESC LIMIT 1;</code> | <code>status = 'SUCCEEDED'</code> (or provider-specific success string). |
| Forced run works | <code>SELECT cron.run_job(<jobid>); then inspect job_run_details</code> | Run record appears with recent <code>start_time</code> , non-error <code>return_message</code> . |
| Analyze/Vacuum timestamps | <code>SELECT last_analyze, last_autoanalyze, last_vacuum, last_autovacuum FROM pg_stat_user_tables WHERE relname='matches';</code> | At least one of the timestamps updated after the forced run window. |
| Dead tuples trend | <code>SELECT n_dead_tup FROM pg_stat_user_tables WHERE relname='matches';</code> | Stable or decreasing after weekly VACUUM; alert if growth is monotonic across days. |
| Autovacuum reloptions | <code>SELECT reloptions FROM pg_class WHERE relname='matches';</code> | Includes <code>autovacuum_vacuum_scale_factor=0.05, autovacuum_analyze_scale_factor=0.02</code> . |
| Autovacuum activity | <code>SELECT autovacuum_count, autoanalyze_count FROM pg_stat_user_tables WHERE relname='matches';</code> | Counts increase over time under write load (post-tuning). |

15.3.3 Redis: Durability & Recovery (BullMQ backing store)

15.3.3.1 Managed Redis (Redis Cloud by Redis)

Goal: durable BullMQ metadata with pragmatic AOF settings, HA, TLS, and verifiable backups.

Setup.

1. Create a Redis Cloud database (single region).

- (a) Redis Cloud Console → *Subscriptions* → **New Subscription**.
- (b) Choose **Cloud: AWS, Region: us-east-1, Plan: Fixed** with **1 GB RAM** and **10 GB storage**.
- (c) Name the database `bullmq-prod`. [21]

2. Enable persistence (AOF) with concrete values.

- (a) Database → **Data Persistence**.
- (b) Turn on **Append Only File (AOF)** and set **Fsync policy = Every second (everysec)**. [12]
- (c) (Optional) Enable **RDB snapshots** at **every 15 minutes**. [12]
- (d) Ensure **Automatic AOF rewrite** remains **Enabled** (default threshold is fine). [12]

3. High availability (HA).

- (a) Database settings → **High Availability** → set **Replication: On**.
- (b) Target **1 primary + 1 replica**. [21]

4. Memory safety (no silent loss).

- (a) Set **Eviction Policy** to `noeviction`. [18]
- (b) Add alerts at **80% memory** and **80% disk**.

5. Security (TLS + AUTH).

- (a) Use the provided `rediss://` endpoint (TLS). [19]
- (b) Rotate the default password; optionally create ACL user `bullmq` with minimal commands.
- (c) Restrict source IPs to API/worker egress via Redis Cloud network rules. [21]

6. Backups & restore drills.

- (a) Schedule **Automated Backups: daily at 03:00 UTC**, retention **7 days**. [20]
- (b) From *Backups*: **Create database from backup** for drills (name: `bullmq-restore-drill`); point a staging worker and verify keys/jobs. [20]
- (c) Repeat the drill **quarterly**. [20]

7. Wire BullMQ to Redis Cloud (Node.js).

- (a) Set:

```
1 export REDIS_URL="rediss://default:pA$$-exAmplE-9nv@redis-12345.c12.us-east-1-1-ec2.redns.redis-cloud.com:12345"
```

(b) Example usage:

```
1 import { Queue } from 'bullmq';
2 import IORedis from 'ioredis';
3
4 const redis = new IORedis(process.env.REDIS_URL!); // TLS implied by rediss://
5 export const scoreMatchesQ = new Queue('score-matches', { connection: redis });
```

8. **CLI sanity (TLS)** — from a bastion/CI runner:

```
1 # Expect PONG (TLS via rediss://)
2 redis-cli -u "rediss://default:pA$$-exAmplE-9nv@redis-12345.c12.us-east-1-1.ec2.
   redns.redis-cloud.com:12345" PING
```

Test *Test maintenance, durability & recovery.*

1. **AOF durability smoke.**

- (a) Write a marker: `SET bullmq:test:marker "mvp-2025-09-21T10:00Z"`.
- (b) Wait > 1s (fsync window), then run: `INFO persistence` and verify `aof_enabled:1`.
[17]
- (c) Restart the client (not the service) and `GET` the marker; it must persist.

2. **AOF rewrite health.**

- (a) Generate some write load (enqueue 1,000 small jobs/keys).
- (b) Check: `INFO persistence` → `aof_last_bgrewrite_status:ok`, `aof_rewrite_in_progress:0`.
[17]

3. **Eviction policy guard.**

- (a) Read config: `CONFIG GET maxmemory-policy` ⇒ `noeviction`. [18]

4. **TLS/auth enforcement.**

- (a) `redis-cli -u rediss://... PING` ⇒ `PONG`.
- (b) Try wrong password ⇒ expect `NOAUTH Authentication required` / failure.

5. **Backup & restore drill.**

- (a) Ensure a fresh backup exists (scheduled 03:00 UTC).
- (b) In the console, *Create database from backup* ⇒ name `bullmq-restore-drill`.
- (c) Connect staging worker and verify presence of sentinel key `bullmq:test:marker` and representative BullMQ keys (e.g., `bull:score-matches:wait`). [20]

Results *Passed-results variables (verification matrix).*

| Area | Check / Command | Pass Criteria (Expected) |
|------------------------|--|---|
| AOF enabled & policy | CONFIG GET appendonly, CONFIG GET appendfsync (UI/CLI) | appendonly = yes, appendfsync = everysec. |
| AOF health | INFO persistence | aof_enabled:1, aof_last_bgrewrite_status:ok, aof_rewrite_in_progress:0. |
| Eviction policy | CONFIG GET maxmemory-policy | noeviction. |
| TLS/auth | redis-cli -u rediss://... PING | PONG only when authenticated; TLS negotiated. |
| Backup & restore drill | Console: “Create database from backup” | New DB online; sentinel key and BullMQ keys present. |

Rationale.

- **AOF every second** → at most ≤ 1 s potential data loss with strong throughput. [12]
- **noeviction** → never silently drop queue keys; fail fast and scale/clean. [18]
- **HA replica** → managed failover with minimal downtime. [21]

Monitoring.

- **AOF health:** aof_enabled=1, aof_last_bgrewrite_status=ok, aof_rewrite_in_progress:0. [17]
- **Capacity:** memory usage < 80%, disk usage < 80%; alert if used_memory_dataset growth exceeds plan.
- **Latency:** latency doctor in emergencies; otherwise export vendor metrics to your dashboard. [22]

15.3.3.2 Self-hosted on Render (Private Service)

Goal: durable BullMQ metadata with AOF, private networking, explicit backups, and verifiable restore drills.

Setup.

1. Create a Private Service with persistent disk.

- (a) In Render, create a new **Private Service** from your repo (Docker).[23]
- (b) Attach a **Persistent Disk: 20 GB**, mount path /data.[24]
- (c) Service name: **redis-selfhosted**; internal hostname will be **redis-selfhosted** on the private network.[29]

2. **Repository files:** add `render.yaml`, `Dockerfile`, `redis.conf`, `scripts/backup-aof.sh`.
(Image pin example uses the official Redis Docker image.[31])

Listing 119: `render.yaml`

```
1 services:
2   - name: redis-selfhosted
3     type: private_service
4     env: docker
5     plan: standard
6     autoDeploy: true
7     disk:
8       name: redis-data
9       mountPath: /data
10      sizeGB: 20
11    envVars:
12      - key: REDIS_PASSWORD
13        value: pA$$-exAmplE-render
14        generateValue: true
15      # Render will build Dockerfile at repo root
16      # Redis command and requirepass are set by the container entrypoint
17
```

Listing 120: `Dockerfile`

```
1 FROM redis:7.2-alpine
2 COPY redis.conf /usr/local/etc/redis/redis.conf
3 COPY scripts/backup-aof.sh /usr/local/bin/backup-aof.sh
4 RUN chmod +x /usr/local/bin/backup-aof.sh
5 # Ensure data dir exists (Render mounts /data as a persistent disk)
6 RUN mkdir -p /data
7 EXPOSE 6379
8 # Pass requirepass from env; persist to /data; fsync everysec
9 ENTRYPOINT ["sh", "-c", "exec redis-server /usr/local/etc/redis/redis.conf --
   requirepass $REDIS_PASSWORD"]
10
```

Persistence settings (`appendonly yes`, `appendfsync everysec`) per Redis persistence guidance.[25, 31]

Listing 121: `redis.conf` (Render)

```
1 # ----- Persistence -----
2 dir /data
3 appendonly yes
4 appendfsync everysec
5 # Optional lightweight snapshot as a secondary safety
6 save 900 1
```

```

7
8 # ----- Memory / Eviction -----
9 maxmemory 0
10 maxmemory-policy noeviction
11
12 # ----- Networking / Hardening -----
13 protected-mode yes
14 bind 0.0.0.0
15 port 6379
16 # TLS is not configured here; rely on Render private network isolation
17 # If your org mandates TLS on internal hops, front this Private Service
18 # with a TLS-terminating proxy (e.g., stunnel/envoy/HAProxy) and point clients at it
19 .
20 # ----- AOF Rewrite (leave defaults; monitor via INFO persistence) -----
21 # auto-aof-rewrite-percentage 100
22 # auto-aof-rewrite-min-size 64mb
23

```

Eviction policy `noeviction` and related behavior reference.[26, 32]

3. Deploy the service.

- (a) Push the repo with the files above; Render will build and run the private service.
- (b) Once live, note internal connection: `redis://:pA$$-exAmPlE-render@redis-selfhosted:6379`.
(For TLS-enabled endpoints, use `rediss://` and `-tls` with `redis-cli`.)[30]

4. Wire BullMQ to self-hosted Redis (Node.js).

- (a) In API/Worker environment, set:

```
1 export REDIS_URL="redis://:pA$$-exAmPlE-render@redis-selfhosted:6379"
```

- (b) Example usage:

```

1 import { Queue } from 'bullmq';
2 import IORedis from 'ioredis';
3
4 const redis = new IORedis(process.env.REDIS_URL!);
5 export const scoreMatchesQ = new Queue('score-matches', { connection: redis });

```

5. Backups (Render scheduled job or CI).

- (a) Option A — *Render Cron Job (separate private service)*: create a small private service that runs `backup-aof.sh` via cron (e.g., `s6` or `crond` in Alpine) at **03:00 UTC**.
- (b) Option B — *CI Scheduled Workflow*: run the backup script from a runner with network access to the private service.

Listing 122: `scripts/backup-aof.sh`

```

1  #!/usr/bin/env bash
2  set -euo pipefail
3  STAMP=$(date -u +%Y%m%dT%H%M%SZ)
4  REDIS_URL="${REDIS_URL:-redis://:pA$$-exAmPlE-render@redis-selfhosted:6379}"
5  BACKUP_DIR="${BACKUP_DIR:-/tmp}"
6  ARCHIVE="${BACKUP_DIR}/redis-aof-${STAMP}.tar.gz"
7
8  # Ensure AOF is enabled and fsynced
9  redis-cli -u "$REDIS_URL" CONFIG GET appendonly | grep -q yes
10 redis-cli -u "$REDIS_URL" CONFIG GET appendfsync | grep -q everysec
11
12 # Optional: trigger BGREWRITEAOF to compact the AOF if not running
13 AOF_REWRITE=$(redis-cli -u "$REDIS_URL" INFO persistence | grep '^
    aof_rewrite_in_progress:' | cut -d: -f2)
14 if [ "$AOF_REWRITE" = "0" ]; then
15     redis-cli -u "$REDIS_URL" BGREWRITEAOF || true
16 fi
17
18 # Wait a moment to let rewrite settle (best-effort)
19 sleep 10
20
21 # Archive AOF/RDB from the mounted /data (via a one-off tar in a scratch container
    is ideal;
22 # here we assume this script runs on a host/runner with access to /data via network
    or sidecar)
23 # If running inside the same pod/container with /data mounted:
24 tar -czf "$ARCHIVE" -C /data . \
25     --exclude='lost+found' \
26     --warning=no-file-changed || true
27
28 echo "Created archive: ${ARCHIVE}"
29
30 # Example: upload to S3-compatible storage (set AWS_* envs)
31 # aws s3 cp "$ARCHIVE" "s3://my-redis-backups/${STAMP}/redis-aof.tar.gz" --sse
    AES256
32

```

AOF rewrite command semantics and safety.[28]

6. Connectivity sanity.

```

1  # Expect PONG (no TLS; private network):
2  redis-cli -u "redis://:pA$$-exAmPlE-render@redis-selfhosted:6379" PING

```

For TLS-enabled deployments, `redis-cli` supports `-tls` and `CA` flags; use `rediss://` URLs.[30]

Test. *Test maintenance, durability & recovery.*

1. AOF durability smoke.

- (a) Write a marker: `SET bullmq:test:marker "mvp-2025-09-21T10:00Z"`.
- (b) Wait > 1 s (fsync window), then run: `INFO persistence` and verify `aof_enabled:1`.^[27]
- (c) Trigger a **Manual Redeploy** of `redis-selfhosted` in Render (causes container restart).
- (d) After healthy, `GET bullmq:test:marker` should return the value.

2. AOF rewrite health.

- (a) Enqueue **1,000** small jobs via BullMQ (or generate write load with a script).
- (b) Check `INFO persistence: aof_last_bgrewrite_status:ok, aof_rewrite_in_progress:0`.^[27]

3. Eviction policy guard.

- (a) `CONFIG GET maxmemory-policy` should be `noeviction`.^[26, 32]
- (b) If memory is pressured, verify writes fail loudly (no silent drops).

4. Backup & point-in-time restore drill.

- (a) Run `backup-aof.sh` and confirm an archive file exists (e.g., `/tmp/redis-aof-<STAMP>.tar.gz`) and uploaded to your bucket.
- (b) **Restore to a staging clone:** create a second private service `redis-restore-drill` with a fresh disk; stop the service; place the archived `appendonly.aof` into `/data`; start the service; Redis will *replay* AOF.^[25]
- (c) Connect a staging worker to `redis-restore-drill` and verify the sentinel key and representative BullMQ keys exist (e.g., `bull:score-matches:wait`).

5. Auth enforcement.

- (a) `redis-cli -u redis://:wrong@redis-selfhosted:6379 PING` should fail with `NOAUTH`.
- (b) Only a correct password should yield `PONG`.

Results. *Passed-results variables (verification matrix).*

| Area | Check / Command | | Pass Criteria (Expected) |
|------------------------|--|--------|---|
| AOF enabled & policy | CONFIG GET appendonly, | CONFIG | appendonly = yes, appendfsync = everysec. |
| AOF health | GET appendfsync | | |
| | INFO persistence | | aof_enabled:1, aof_last_bgrewrite_status:ok, aof_rewrite_in_progress:0. |
| Eviction policy | CONFIG GET maxmemory-policy | | noeviction. |
| Auth | redis-cli -u | | PONG only when authenticated. |
| | redis://:pA\$\$-exAmplE-render@redis-selfhosted:6379 | | |
| | PING | | |
| Durability (restart) | smoke Redeploy service, then GET | | Value persists across container restart. |
| | bullmq:test:marker | | |
| Backup & restore drill | Restore to redis-restore-drill | | Sentinel key present; representative |
| | from archive | | BullMQ keys exist; staging worker round-trip succeeds. |

Rationale.

- **AOF every second** → balance durability and throughput for queue metadata; worst case $\leq 1s$ loss while avoiding the latency of **always** fsync.
- **noeviction** → never silently drop BullMQ keys under pressure; fail fast so autoscaling/ops can react (clean up, scale RAM/disk) instead of corrupting queue state.
- **Private network & TLS** → Render Private Service is not Internet-exposed; baseline is auth on a private network. If compliance requires in-transit encryption internally, front Redis with a TLS-terminating proxy and use **rediss://**.
- **Single-instance tradeoff** → unlike managed HA, this setup favors simplicity + fast recovery (container restart + AOF replay). Acceptable for BullMQ metadata when workers are idempotent and jobs are re-drivable.
- **Backups & drills** → nightly AOF archives give point-in-time restore by placing **appendonly.aof** onto a fresh disk and letting Redis replay; verification uses a sentinel key and representative BullMQ keys.
- **AOF rewrite** → leave defaults; monitor **aof_last_bgrewrite_status=ok** and **aof_rewrite_in_progress**; optionally trigger **BGREWRITEAOF** pre-backup to compact artifacts.
- **Operational guardrails** → alerts at 80% memory and disk; track **aof_enabled** and rewrite health; treat evictions/restarts as surfaced failures, not silent data loss paths.

Monitoring.

- **AOF health:** track `aof_enabled`, `aof_last_bgrewrite_status`, `aof_rewrite_in_progress`.^[27]
- **Capacity:** alert at 80% `used_memory` and disk utilization for `/data`.
- **Latency:** observe command latency during load and during AOF rewrites.
- **BullMQ semantics:** design handlers idempotent (*at-least-once* delivery).
- **Disk & AOF rewrite:** alert on `/data` usage $\geq 80\%$ and on `aof_last_bgrewrite_status` \neq ok or prolonged `aof_rewrite_in_progress`.

15.3.4 Kill Switches & Circuit Breaking

| Control | MVP Target | Notes / Basis |
|---------------------------|-----------------------------|--|
| Per-tenant OpenAI breaker | Trip on >20% 5xx over 5 min | Open, then half-open after 60s; fall back to cached answers. :contentReference[oaicite:13]index=13 |
| Global LLM rate limit | 50–100 QPS cap | Budget against provider limits; sheds load safely. |
| Queue global pause | Admin toggle | Halts consumers to stop churn during incidents. |
| Read-only mode | Feature flag | Protect DB during recovery/maintenance. |

15.3.5 SLO Guardrails & Alerting

| Signal | Budget/Threshold | Notes / Basis |
|--------------------|----------------------|---|
| API latency | $p95 \leq 500$ ms | Breach \Rightarrow scale API, throttle producers. |
| Vector KNN latency | $p95 \leq 200$ ms | Tune probes/ef_search under load. |
| Error-budget burn | 2-window burn alerts | 5m/1h (page) and 30m/6h (ticket) SRE pattern. :contentReference[oaicite:14]index=14 |
| Queue backlog | >10k jobs for 5 min | Auto-scale workers; rate-limit producers. |

15.3.6 Deploy Safety & Planned Disruption

| Control | MVP Target | Notes / Basis |
|---------------------|---------------------------------|--|
| Rollouts | Rolling/blue-green | Health checks gate step; quick rollback. |
| Health probes | <code>/healthz</code> 10s | Fail fast; remove from LB before kill. |
| PodDisruptionBudget | <code>minAvailable</code> = 80% | Prevents voluntary disruptions from draining all pods. :contentReference[oaicite:15]index=15 |
| Config deploys | Feature flags | Dark launch; progressive exposure. |

16 Conclusion

Status & Intent. This is a *theory-first, non-final* internal design memo. It focuses on the most important architectural choices and interfaces for the Bidion MVP and uses *illustrative* configuration values and targets (e.g., SLOs, sizes, pool limits) to align the team. All numbers, vendor selections, and thresholds are subject to change during implementation and will be refined through spikes, load tests, and RFCs.

This document outlines a practical design for Bidion MVP: a stateless API and worker tier backed by PostgreSQL + **pgvector** with RLS-based multitenancy, queue-driven ingest and recompute, and a measured scaling path (partitions, replicas, HNSW, caching) that preserves cost proportionality.

The Bidion MVP uses a pragmatic, evolvable stack: a **Next.js**/React frontend with SSR/ISR and edge/CDN caching for fast UX and simple CI/CD; a **NestJS** API with **BullMQ** workers for asynchronous pipelines; and **Supabase Postgres** + **pgvector** for retrieval and matching.

On the platform side, we enforce tenant safety with Postgres **RLS**, control cost via batching and caching, and hold SLOs (API p95 \leq 500 ms; KNN p95 \leq 200 ms) with **k6**-based checks. The design is intentionally MVP-first: single-region by default, horizontal scale of stateless tiers, and a clear path to partitions and replicas as data or traffic grows. Non-goals for MVP (deferred to v1+) include advanced multi-region data strategies, large-scale full-text+vector hybrids, and automated capacity planning.

The testing and observability plans are specified as part of the design to reduce integration risk later, but remain conceptual until implemented. As an internal specification, this serves as the shared blueprint for initial build-out, early validation with pilot orgs, and controlled evolution toward v1.

17 Acknowledgements

We appreciate contributions and reviews from:

- **Bidion Engineering:** platform & application scaffolding, CI/perf pipelines, and observability setup.
- **Envision MSP stakeholders:** domain guidance on RFP workflows, proposal drafting needs, and success metrics.
- **Community & OSS:** NestJS, pgvector, BullMQ, k6, and related tooling ecosystems that underpin this MVP.

References

- [1] Supabase Docs — Point-in-Time Recovery (PITR): overview, RPO characteristics, restore workflow. <https://supabase.com/docs/guides/platform/backups#pitr>

- [2] PostgreSQL Docs — Continuous Archiving & Point-in-Time Recovery (PITR): WAL archiving, base backups, recovery to a timestamp. <https://www.postgresql.org/docs/current/continuous-archiving.html>
- [3] Supabase Docs — Scheduling (server-side) with cron / `pg_cron`. https://supabase.com/docs/guides/database/extensions/pg_cron
- [4] `pg_cron` (official) — Simple cron-based job scheduler for Postgres. https://github.com/citusdata/pg_cron
- [5] Redis.io — Durability & AOF: `appendonly` and `appendfsync everysec`. https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/
- [6] BullMQ Docs — Retries, backoff, and pause/resume support. <https://docs.bullmq.io/guide/retries>
- [7] Kubernetes Docs — Pod Disruption Budgets. <https://kubernetes.io/docs/tasks/run-application/configure-pdb/>
- [8] PostgreSQL Docs — Backup/Recovery and vacuum guidance (see Vacuuming & PITR sections). <https://www.postgresql.org/docs/current/routine-vacuuming.html>
- [9] BullMQ Docs — Overview and Guides. <https://docs.bullmq.io/>
- [10] BullMQ Docs — Retrying failing jobs (retries & backoff). <https://docs.bullmq.io/guide/retrying-failing-jobs>
- [11] Kubernetes Docs — Pod Disruption Budgets. <https://kubernetes.io/docs/tasks/run-application/configure-pdb/>
- [12] Redis Docs — Durability & AOF (`appendonly`, `appendfsync`). https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/
- [13] BullMQ Docs — Overview / API (Queue pause/resume, job options, worker close). <https://api.docs.bullmq.io/classes/v4.Queue.html>
- [14] BullMQ Docs — Retries and backoff (guide). <https://docs.bullmq.io/guide/retries>
- [15] Kubernetes Docs — Pod Disruption Budgets. <https://kubernetes.io/docs/tasks/run-application/configure-pdb/>
- [16] KEDA Docs — Concepts: External Scalers (scaling from external systems incl. Redis). <https://keda.sh/docs/2.17/concepts/external-scalers/>
- [17] Redis Commands — `INFO` (persistence fields such as `aof_enabled`, `aof_last_bgrewrite_status`). <https://redis.io/commands/info/>
- [18] Redis Docs — Eviction policies (`maxmemory-policy`, `noeviction`). <https://redis.io/docs/latest/develop/reference/eviction/>

- [19] Redis Docs — TLS configuration and secure connections. https://redis.io/docs/latest/operate/oss_and_stack/management/security/encryption/tls/
- [20] Redis Enterprise/Cloud — Schedule backups and export/restore workflows. <https://redis.io/docs/latest/operate/rs/databases/import-export/schedule-backups/>
- [21] Redis Cloud/Enterprise — Operate docs hub (creating DBs, HA/replication, features). <https://redis.io/docs/latest/operate/>
- [22] Redis Commands — LATENCY DOCTOR. <https://redis.io/commands/latency-doctor/>
- [23] Render Docs — Private Services. <https://render.com/docs/private-services> (accessed 2025-09-22).
- [24] Render Docs — Persistent Disks. <https://render.com/docs/disks> (accessed 2025-09-22).
- [25] Redis Docs — Persistence (AOF/RDB). https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/ (accessed 2025-09-22).
- [26] Redis Docs — Key Eviction (Develop Reference). <https://redis.io/docs/latest/develop/reference/eviction/> (accessed 2025-09-22).
- [27] Redis Command Reference — INFO (persistence section fields). <https://redis.io/docs/latest/commands/info/> (accessed 2025-09-22).
- [28] Redis Command Reference — BGREWRITEAOF. <https://redis.io/docs/latest/commands/bgrewriteaof/> (accessed 2025-09-22).
- [29] Redis Docs — TLS (encryption at transport). https://redis.io/docs/latest/operate/oss_and_stack/management/security/encryption/ (accessed 2025-09-22).
- [30] Redis Tools — `redis-cli` (TLS options and usage). <https://redis.io/docs/latest/develop/tools/cli/> (accessed 2025-09-22).
- [31] Docker Hub — Official Redis Image (7.2-alpine). https://hub.docker.com/_/redis (accessed 2025-09-22).
- [32] Redis Enterprise Docs — Eviction Policy Overview. <https://redis.io/docs/latest/operate/rs/databases/memory-performance/eviction-policy/> (accessed 2025-09-22).