Bidion MVP System Design Specification

Theoretical Framework and Design Rationale

Jason Ramirez

September 20, 2025

Contents

1 Definitions

Table 1: Definitions of Terms and Acronyms

Term	Definition
API	Application Programming Interface; your 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 `pgvector` (*Hierarchical Navigable Small World* / *Inverted File*; 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

Disclaimer. This document is a *theory-first, non-final* internal design memo for the Bidion MVP. It captures architectural intent, guiding principles, and illustrative configuration targets to align the team. It is *not* a commitment to specific vendors, exact capacities, or dates; all parameters (e.g., pool sizes, SLOs, index settings) are placeholders to be validated.

Scope. We focus on the core system shape—Next.js frontend, NestJS API, BullMQ workers, PostgreSQL+`pgvector` with RLS—and the critical interfaces between them (ingest, matching, retrieval, observability). Detailed implementation steps, production runbooks, and security audits are out of scope for this revision.

Audience. Bidion engineering and adjacent stakeholders (product, infra). The goal is to provide enough specificity to start implementation spikes and create RFCs, while avoiding premature lock-in.

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 $\geq$ *6 months* (alpha $\sim$ 3 months, beta $\sim$ 4–5 months, GA $\geq$ 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–25k MAU $\times$ ~40–60 reads/day $\times$ 30 days $\approx$ 12M–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–10k RFPs $\times$ (initial compute + ~2 updates) $\approx$ *18k–30k* recomputes.

Size per Entity (storage planning)

- **RFP (median)**: 40 pages, ~12k words $\Rightarrow$ ~50 chunks (avg 240 words/chunk).
- **Embedding vector (3072-d float32)**: 3072 $\times$ 4 bytes = **12 KB** per vector.
- **Embeddings per RFP**: 50 chunks $\times$ 12 KB $\approx$ **600 KB** (vectors only).
- **Text/PDF storage (RFP)**: median **2–5 MB** per file (depends on scans/figures).
- **Offering record**: ~2 KB metadata; embedding ~12 KB.
- **KB doc (median)**: 10 pages $\Rightarrow$ ~12 chunks $\Rightarrow$ ~144 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× the averages and are absorbed via CDN and queues.

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

Handy Conversion

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

Performance-Critical Operations (Bid/Proposal Context)

- **RFP create/upload**: immediate user feedback; background parsing/embedding starts within $< 5\,\mathrm{s}$.
- **Matches fetch**: cached and paginated; P95 $\leq 500\,\mathrm{ms}$.
- **Assistant ask**: shortlist + re-rank ⇒ P95 KNN $\leq 200\,\mathrm{ms}$, end-to-end answer target $< 2\,\mathrm{s}$.
- **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 $\leq$ *250 ms*; heavy work async via queue.
- **Enqueue parse/embed jobs:** $\leq$ *150 ms* API acknowledgement; visible queue status in $\leq$ *1 s*.
- **Parse → 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 s*; API backing calls *p95* $\leq$ *400 ms.*
- **Export (CSV/Docx) kickoff:** $\leq$ *200 ms* to acknowledge; download ready notification *P95* $\leq$ *60 s.*
- **Auth/session resume:** $\leq$ *150 ms* token refresh; cold login $\leq$ *600 ms* server time.

4 MVP Scope

4.1 In Scope (MVP Deliverables)

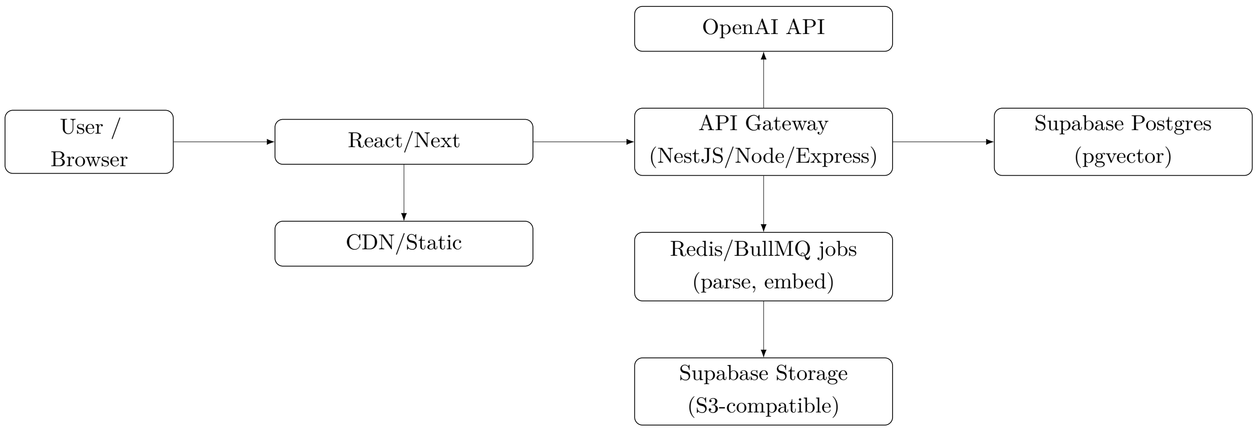
- AuthN/AuthZ, org/user accounts
- Data ingestion: Offerings (forms + CSV), RFPs (URL/PDF/manual), KB uploads
- Matching engine v1: normalization → embeddings → vector search; hybrid score
- Match review UI: score, snippet highlights, rationale
- Draft response assistant (RAG over KB + RFP)
- Dashboard & 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.


OpenAI API
User / Browser
React/Next
API Gateway (NestJS/Node/Express)
Supabase Postgres (pgvector)
CDN/Static
Redis/BullMQ jobs (parse, embed)
Supabase Storage (S3-compatible)

Data flow:

1. Ingest RFPs/offerings/KB → *parse* → *embed* → *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	timestamptz		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	→ `orgs.id`
created_at	timestamptz		default `now()`

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	timestamptz		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	timestamptz		default `now()`
updated_at	timestamptz		default `now()`

Table 7: Table: `embeddings`

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		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	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
rfp_id	uuid	FK	→ `documents.id` (`kind='rfp'`)
offering_id	uuid	FK	→ `offerings.id`
score	double precision		final hybrid score $[0, 1]$
reasons	jsonb		snippets/keywords/penalties
created_at	timestamptz		default `now()`

Table 9: Table: `score_config`

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

Table 10: Table: `work_items`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
rfp_id	uuid	FK	→ `documents.id` (`kind='rfp'`)
title	text		
status	text		`new|in_progress|review|done`
assignee_user_id	uuid	FK	→ `users.id` (nullable)
created_at	timestamptz		default `now()`

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	timestamptz		default `now()`

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

Listing 1: Core tables, vector storage, and indexes

```
-- Supabase: enable required extensions in your project
CREATE EXTENSION IF NOT EXISTS pgcrypto;
CREATE EXTENSION IF NOT EXISTS vector;

-- documents & chunks
CREATE TABLE IF NOT EXISTS documents (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),
  ref_id      uuid,                 -- optional foreign link (e.g., offering id)
  filename    text,
  storage_url text,
  mime        text,
  created_at  timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE IF NOT EXISTS chunks (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  document_id uuid NOT NULL REFERENCES documents(id) ON DELETE CASCADE,
  idx         int  NOT NULL,        -- chunk order
  section     text,                 -- optional 'Requirements', etc.
  text        text NOT NULL
);

-- embeddings (pgvector)
CREATE TABLE IF NOT EXISTS embeddings (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),
  ref_id      uuid NOT NULL,           -- document_id (or offering id if we embed inline)
  chunk_id    uuid,                    -- null for offering-wide vectors if we store one
  vector      vector(3072) NOT NULL,
  text        text NOT NULL,
  meta        jsonb DEFAULT '{}'::jsonb
);

-- matches
CREATE TABLE IF NOT EXISTS matches (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  rfp_id      uuid NOT NULL,           -- documents.id where kind='rfp'
  offering_id uuid NOT NULL,           -- documents.id where kind='offering'
  score       double precision NOT NULL,
  reasons     jsonb NOT NULL DEFAULT '{}'::jsonb,
  created_at  timestamptz NOT NULL DEFAULT now()
);

-- keyword/rules (simple MVP)
CREATE TABLE IF NOT EXISTS score_config (
  org_id      uuid PRIMARY KEY,
  boosts      jsonb NOT NULL DEFAULT '{}'::jsonb,  -- {"soc2":0.25, "24/7":0.15}
  required    jsonb NOT NULL DEFAULT '[]'::jsonb,  -- ["soc2","24/7"]
  forbidden   jsonb NOT NULL DEFAULT '[]'::jsonb   -- ["on-prem only"]
```

```sql
);

-- offering catalog (you can also model as documents(kind='offering'))
CREATE TABLE IF NOT EXISTS offerings (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  title       text NOT NULL,
  description text NOT NULL,
  tags        text[] DEFAULT '{}',
  created_at  timestamptz NOT NULL DEFAULT now(),
  updated_at  timestamptz NOT NULL DEFAULT now()
);

-- indexes
CREATE INDEX IF NOT EXISTS idx_chunks_doc ON chunks(document_id, idx);
CREATE INDEX IF NOT EXISTS idx_embeddings_kind_org ON embeddings(kind, org_id);
CREATE INDEX IF NOT EXISTS idx_embeddings_vector ON embeddings USING ivfflat (vector
    vector_cosine_ops) WITH (lists=100);
CREATE INDEX IF NOT EXISTS idx_matches_rfp ON matches(org_id, rfp_id, score DESC);


-- Row-level security (multitenancy)
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_rfps ON documents
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid AND kind = 'rfp');

ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_matches ON matches
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);

ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_crud_offerings ON offerings
  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 inserts/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`.

How it fits the 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 → chunk → 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

```
# Supabase Postgres (managed)
# Use the pooled URL (port 6543) for app traffic. Keep sslmode=require.
DATABASE_URL=postgresql://postgres:<password>@db.<project-ref>.supabase.co:6543/postgres?
    sslmode=require

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# Storage (S3-compatible; MinIO in compose)
S3_ENDPOINT=http://localhost:9000
S3_BUCKET=bucket
S3_ACCESS_KEY=dev
S3_SECRET_KEY=devdevdev

# OpenAI
OPENAI_API_KEY=sk-...

# App
NODE_ENV=development
```

7.2 Docker Compose (Redis)

Listing 3: docker-compose.yml for Redis

```
version: "3.9"

networks:
  backend:
    driver: bridge
    internal: true   # prevents external access

volumes:
  redis_data:

services:
  redis:
    image: redis:7
    command: >
```

```
      redis-server
      --appendonly yes
      --requirepass ${REDIS_PASSWORD}
      --maxmemory-policy allkeys-lru
      --protected-mode yes
    healthcheck:
      test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}", "PING"]
      interval: 10s
      timeout: 3s
      retries: 5
    volumes:
      - redis_data:/data
    restart: always
    networks: [backend]
    # 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

```
import { Pool } from 'pg';
export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
// Ensure DATABASE_URL includes sslmode=require for Supabase.

export async function withTx<T>(fn: (c: import('pg').PoolClient) => Promise<T>) {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    const r = await fn(client);
    await client.query('COMMIT');
    return r;
  } catch (e) {
    await client.query('ROLLBACK'); throw e;
  } finally {
    client.release();
  }
}
```

8 Shared Libraries

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

Listing 5: Normalize strings and chunk long text

```
export function normalizeForKeywords(raw: string) {
  return raw
    .normalize('NFKC')
    .toLowerCase()
    .replace(/[^\p{L}\p{N}\s.%/-]/gu, ' ')
    .replace(/\s+/g, ' ')
    .trim();
}

// ~800-1200 tokens ≈ ~3500-5000 chars (rough)
export function chunkByChars(s: string, maxChars = 4000) {
  const parts: string[] = [];
  let buf = '';
  for (const p of s.split(/\n{2,}/)) {
    if ((buf + '\n\n' + p).length > maxChars) {
      if (buf) parts.push(buf.trim());
      if (p.length > maxChars) {
        for (let i = 0; i < p.length; i += maxChars) {
          parts.push(p.slice(i, i + maxChars));
        }
        buf = '';
      } else {
        buf = p;
      }
    } else {
      buf = buf ? `${buf}\n\n${p}` : p;
    }
  }
  if (buf) parts.push(buf.trim());
  return parts;
}
```

9 Matching Engine (v1)

9.1 Retrieval pipeline: normalization → embeddings → 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 $1 -$ 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 $1 -$ 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:

$$S_{\text{sem}}^{(\text{pair})} = \max\{0,\ \text{sim}_{\cos}(\mathbf{a},\mathbf{b})\},$$

$$S_{\text{kw}} = \min\Big(1,\ \sum_t w_t\, \mathbf{1}_{\text{substr}(\text{normalize}(t),\, \text{normalize}(\texttt{text}))}\Big),$$

$$S_{\text{rule}} = \max\big(0,\ \min(1,\ 1-\text{penalty})\big).$$

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:

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

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

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

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

- $S_{\text{sem}}^{\max}(\text{off}) = \max_{\text{chunks}} \text{sim}$ (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 $R = \{r_1, \ldots, r_{|R|}\}$ be RFP chunks and $O = \{o_1, \ldots, o_{|O|}\}$ 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: $\widetilde{S}_{\text{sem}} = \frac{1}{|R|}\sum_{i=1}^{|R|} m_i$.

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

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

```
-- Inputs: :rfp_id, :offering_id, :org_id, :k
WITH rfp_chunks AS (
  SELECT e.chunk_id, e.vector
  FROM embeddings e
  WHERE e.org_id = :org_id AND e.kind = 'rfp' AND e.ref_id = :rfp_id
),
off_chunks AS (
  SELECT e.chunk_id, e.vector
  FROM embeddings e
  WHERE e.org_id = :org_id AND e.kind = 'offering' AND e.ref_id = :offering_id
),
topk_per_rfp AS (
  SELECT
    r.chunk_id AS r_chunk_id,
    AVG(1 - (r.vector <=> o.vector)) AS mean_topk
  FROM rfp_chunks r
  CROSS JOIN LATERAL (
    SELECT o.vector
    FROM off_chunks o
    ORDER BY r.vector <=> o.vector
    LIMIT :k
  ) o
  GROUP BY r.chunk_id
)
SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
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):

$$\mathrm{sim}(r_1, o_1) = 0.88,\ \mathrm{sim}(r_1, o_2) = 0.42,\ \mathrm{sim}(r_1, o_3) = 0.66,$$
$$\mathrm{sim}(r_2, o_1) = 0.35,\ \mathrm{sim}(r_2, o_2) = 0.71,\ \mathrm{sim}(r_2, o_3) = 0.62.$$

With k=2: $m_1 = (0.88 + 0.66)/2 = 0.77$, $m_2 = (0.71 + 0.62)/2 = 0.665$. Then $\widetilde{S}_{\text{sem}} = (0.77 + 0.665)/2 = 0.7175$, 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* $S_{\text{kw}} \in [0,1]$

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:

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

where norm$(\cdot)$ applies text normalization (NFKC, lowercase, preserve `.%/-`, collapse spaces).

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

```
{
  "boosts": {
    "soc2": 0.25,
    "hipaa": 0.20,
    "24/7": 0.15,
    "99.99%": 0.10,
    "iso 27001": 0.20
  }
}
```

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 $\to$ lowercase $\to$ remove non-alphanumerics except `.%/-` $\to$ collapse spaces.

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

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

Listing 9: Keyword score with normalization and capping

```
import { normalizeForKeywords } from './text'; // NFKC -> lower -> keep .%/- -> collapse
    spaces

export function keywordScore(offTxt: string, boosts: Record<string, number>) {
  const text = normalizeForKeywords(offTxt);
  let score = 0;
  for (const [term, w] of Object.entries(boosts)) {
    if (text.includes(normalizeForKeywords(term))) score += w;
  }
  return Math.min(1, score); // S_kw in [0,1]
}
```

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

```sql
SELECT boosts
FROM score_config
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

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

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* $S_{\text{rule}} \in [0, 1]$

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`

```sql
-- Per-org configuration of rules and keyword boosts
-- (Boosts are used by the keyword component, not the rules component.)
```

```
org_id    uuid  PRIMARY KEY,
boosts    jsonb NOT NULL DEFAULT '{}'::jsonb,  -- e.g., {"soc2":0.25,"24/7":0.15}
required  jsonb NOT NULL DEFAULT '[]'::jsonb,  -- e.g., ["soc2","99.99% uptime"]
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**:
  - $p_{\text{Req}} = \texttt{miss} / |\text{required}|$ (or 0 if no required terms)
  - $p_{\text{For}} = \texttt{forb} / |\text{forbidden}|$ (or 0 if no forbidden terms)
- **Weighted penalty**: penalty $= 0.7\, p_{\text{Req}} + 0.3\, p_{\text{For}}$.
- **Rule score**: $S_{\text{rule}} = \max\big(0, \min(1, 1 - \text{penalty})\big)$.

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

```
import { normalizeForKeywords } from './text';

function ruleCompliance(text: string, required: string[], forbidden: string[]) {
  const norm = normalizeForKeywords(text);
  const miss = required.filter(r => !norm.includes(normalizeForKeywords(r))).length;
  const forb = forbidden.filter(f => norm.includes(normalizeForKeywords(f))).length;
  const pReq = required.length ? miss / required.length : 0;
  const pFor = forbidden.length ? forb / forbidden.length : 0;
  const penalty = 0.7 * pReq + 0.3 * pFor;
  return Math.max(0, Math.min(1, 1 - penalty));
}
```

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 $S_{\text{sem}}, S_{\text{kw}}, S_{\text{rule}}$ 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 $= 0 \Rightarrow \mathbf{S_{\text{rule}} = 1.00}$.

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$= 0.7 \Rightarrow \mathbf{S}_{\text{rule}} = \mathbf{0.30}$.

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 $1-$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

```
import { pool } from '../../../lib/db';
import { keywordScore, ruleCompliance, finalScore } from '../../../lib/shared/scoring';

async function getScoreConfig(orgId: string) {
  const { rows } = await pool.query(
    `SELECT boosts, required, forbidden FROM score_config WHERE org_id=$1`, [orgId]
  );
  if (!rows[0]) return { boosts:{}, required:[], forbidden:[] };
  const { boosts, required, forbidden } = rows[0];
  return { boosts, required, forbidden };
}

async function semanticForOffering(orgId: string, rfpId: string, offeringId: string, k =
    3): Promise<number> {
  const { rows } = await pool.query(`
    WITH rfp_chunks AS (
      SELECT vector
      FROM embeddings
      WHERE org_id = $1 AND kind = 'rfp' AND ref_id = $2
    ),
    off_chunks AS (
      SELECT vector
      FROM embeddings
      WHERE org_id = $1 AND kind = 'offering' AND ref_id = $3
    ),
    topk_per_rfp AS (
      SELECT AVG(1 - (r.vector <=> o.vector)) AS mean_topk
      FROM rfp_chunks r
      CROSS JOIN LATERAL (
        SELECT o.vector
        FROM off_chunks o
        ORDER BY r.vector <=> o.vector
        LIMIT $4
      ) o
      GROUP BY r.vector
    )
    SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
    FROM topk_per_rfp
  `, [orgId, rfpId, offeringId, k]);

  return Number(rows[0]?.s_sem ?? 0);
}

async function offeringText(offeringId: string) {
  const { rows } = await pool.query(
    `SELECT title, description FROM offerings WHERE id=$1`, [offeringId]
  );
  if (!rows[0]) return '';
  return `${rows[0].title}\n\n${rows[0].description}`;
}

export async function scoreMatchesTask({ rfpId, orgId }:{ rfpId:string; orgId:string }) {
  const offs = await pool.query(`SELECT id FROM offerings WHERE org_id=$1`, [orgId]);
  const cfg = await getScoreConfig(orgId);

  const results: Array<{ offering_id:string; score:number; reasons:any }> = [];

  for (const o of offs.rows) {
    const offId = o.id as string;
    const sem  = await semanticForOffering(orgId, rfpId, offId);
    const text = await offeringText(offId);
    const kw   = keywordScore(text, cfg.boosts);
    const rule = ruleCompliance(text, cfg.required, cfg.forbidden);
    const score= finalScore(sem, kw, rule);

    const topChunk = await pool.query(`
      SELECT e.text, (1 - (e.vector <=> r.vector)) AS sim
      FROM embeddings e
      JOIN embeddings r ON r.ref_id=$1 AND r.kind='rfp' AND r.org_id=$2
      WHERE e.ref_id=$3 AND e.kind='offering' AND e.org_id=$2
      ORDER BY sim DESC
      LIMIT 1
    `, [rfpId, orgId, offId]);

    results.push({
      offering_id: offId,
      score,
      reasons: {
        semantic_top_sim: Number(topChunk.rows[0]?.sim ?? 0),
        semantic_snippet: topChunk.rows[0]?.text ?? null,
        keyword_hits: Object.keys(cfg.boosts).filter(k => text.toLowerCase().includes(k.
    toLowerCase())),
        required_missing: cfg.required.filter(r => !text.toLowerCase().includes(r.
    toLowerCase())),
        forbidden_hit: cfg.forbidden.filter(f => text.toLowerCase().includes(f.
    toLowerCase())),
      }
    });
  }

  results.sort((a,b)=>b.score - a.score);
  const top = results.slice(0,10);
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    await client.query(`DELETE FROM matches WHERE org_id=$1 AND rfp_id=$2`, [orgId, rfpId
    ]);
    for (const r of top) {
      await client.query(`
        INSERT INTO matches(org_id, rfp_id, offering_id, score, reasons)
        VALUES ($1,$2,$3,$4,$5)
      `, [orgId, rfpId, r.offering_id, r.score, r.reasons]);
    }
    await client.query('COMMIT');
  } catch (e) { await client.query('ROLLBACK'); throw e; }
  finally { client.release(); }

  return { count: top.length };
}
```

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`.
3. **ScoreMatchesWorker**: for RFP, compute hybrid scores vs offerings → write `matches`.

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

Listing 14: BullMQ queue setup and job payloads

```
import { Queue, Worker, JobsOptions, QueueEvents } from 'bullmq';
import IORedis from 'ioredis';

export const redis = new IORedis({
  host: process.env.REDIS_HOST, port: Number(process.env.REDIS_PORT || 6379),
});

export function makeQueue(name: string) {
  const q = new Queue(name, { connection: redis });
  const events = new QueueEvents(name, { connection: redis });
  return { q, events };
}

export type ParseDocPayload    = { documentId: string };
export type EmbedChunksPayload = { documentId: string, orgId: string, kind: 'rfp'|'kb'|'
    offering' };
export type ScoreMatchesPayload= { rfpId: string, orgId: string };

export const queues = {
  parseDoc:    makeQueue('parse-doc'),
  embedChunks: makeQueue('embed-chunks'),
  scoreMatches:makeQueue('score-matches'),
};

export const defaultJobOpts: JobsOptions = { attempts: 5, backoff: { type: 'exponential',
     delay: 2000 } };
```

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

Listing 15: Batch embed texts with OpenAI

```
import OpenAI from 'openai';
const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

export async function embedBatch(texts: string[]) {
  const res = await client.embeddings.create({
    model: 'text-embedding-3-large',
    input: texts
  });
  return res.data.map(d => d.embedding);
}
```

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

Listing 16: Extract text, chunk, enqueue embeddings

```
import { pool } from '../../../lib/db';
import { chunkByChars } from '../../../lib/shared/text';
import { queues, defaultJobOpts } from './queue';
import pdf from 'pdf-parse';
import fetch from 'node-fetch';

```

```
async function fetchFile(storageUrl: string): Promise<Buffer> {
  const r = await fetch(storageUrl);
  if (!r.ok) throw new Error(‘fetch failed: ${r.status}‘);
  return Buffer.from(await r.arrayBuffer());
}

export async function parseDoc({ documentId }: { documentId: string }) {
  const { rows } = await pool.query(
    ‘SELECT id, org_id, kind, storage_url, mime FROM documents WHERE id=$1‘,
    [documentId]
  );
  if (!rows[0]) throw new Error(’doc not found’);
  const { org_id: orgId, kind, storage_url, mime } = rows[0];

  const buf = await fetchFile(storage_url);
  let text = ’’;
  if (mime?.includes(’pdf’)) {
    const data = await pdf(buf);
    text = data.text || ’’;
  } else {
    text = buf.toString(’utf8’);
  }

  const parts = chunkByChars(text, 4000);
  for (let i = 0; i < parts.length; i++) {
    await pool.query(
      ‘INSERT INTO chunks(document_id, idx, text) VALUES ($1, $2, $3)‘,
      [documentId, i, parts[i]]
    );
  }

  await queues.embedChunks.q.add(’embed’, { documentId, orgId, kind }, defaultJobOpts);
  return { chunks: parts.length };
}
```

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

Listing 17: Embed chunk texts and persist vectors

```
import { pool } from ’../../../lib/db’;
import { embedBatch } from ’./embeddings’;
import { queues, defaultJobOpts } from ’./queue’;

export async function embedChunksTask({ documentId, orgId, kind }:
  { documentId: string; orgId: string; kind: 'rfp'|'kb'|'offering' }) {

  const chunks = await pool.query(
    `SELECT id, text FROM chunks WHERE document_id=$1 ORDER BY idx`,
    [documentId]
  );

  const batchSize = 64;
  for (let i = 0; i < chunks.rowCount; i += batchSize) {
    const slice = chunks.rows.slice(i, i + batchSize);
    const vecs = await embedBatch(slice.map(r => r.text));
    const textAndVec = slice.map((r, j) => ({ chunkId: r.id, text: r.text, vec: vecs[j]
    }));

    const client = await pool.connect();
    try {
      await client.query('BEGIN');
      for (const tv of textAndVec) {
        await client.query(
          `INSERT INTO embeddings(org_id, kind, ref_id, chunk_id, vector, text)
           VALUES ($1,$2,$3,$4,$5,$6)`,
          [orgId, kind, documentId, tv.chunkId, `[${tv.vec.join(',')}]`, tv.text]
        );
      }
      await client.query('COMMIT');
    } catch (e) {
      await client.query('ROLLBACK'); throw e;
    } finally {
      client.release();
    }
  }

  if (kind === 'rfp') {
    await queues.scoreMatches.q.add('score', { rfpId: documentId, orgId }, defaultJobOpts
    );
  }
}
```

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

Listing 18: Start BullMQ workers with concurrency

```
import { Worker } from 'bullmq';
import { redis } from './queue';
import { parseDoc } from './parseDoc.task';
import { embedChunksTask } from './embedChunks.task';
import { scoreMatchesTask } from './scoreMatches.task';

new Worker('parse-doc', async job => parseDoc(job.data), { connection: redis, concurrency
    : 4 });
new Worker('embed-chunks', async job => embedChunksTask(job.data), { connection: redis,
    concurrency: 8 });
new Worker('score-matches', async job => scoreMatchesTask(job.data), { connection: redis,
     concurrency: 4 });

console.log('Workers up: parse-doc, embed-chunks, score-matches');
```

11 API Design

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

Listing 19: Minimal Express setup with JSON body parsing

```
import express from 'express';
import bodyParser from 'body-parser';
import { pool } from '../../lib/db';
import { queues, defaultJobOpts } from '../worker-bridge';
import { draftAssistant } from './svc.assistant';

const app = express();
app.use(bodyParser.json({ limit: '10mb' }));

// ... API endpoints here ...

app.listen(3001, () => console.log('API listening on :3001'));
```

11.2 RFP Endpoints

11.2.1 POST `/rfps` — Create Manual-Text RFP

Listing 20: Express route: POST /rfps (manual text RFP)

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

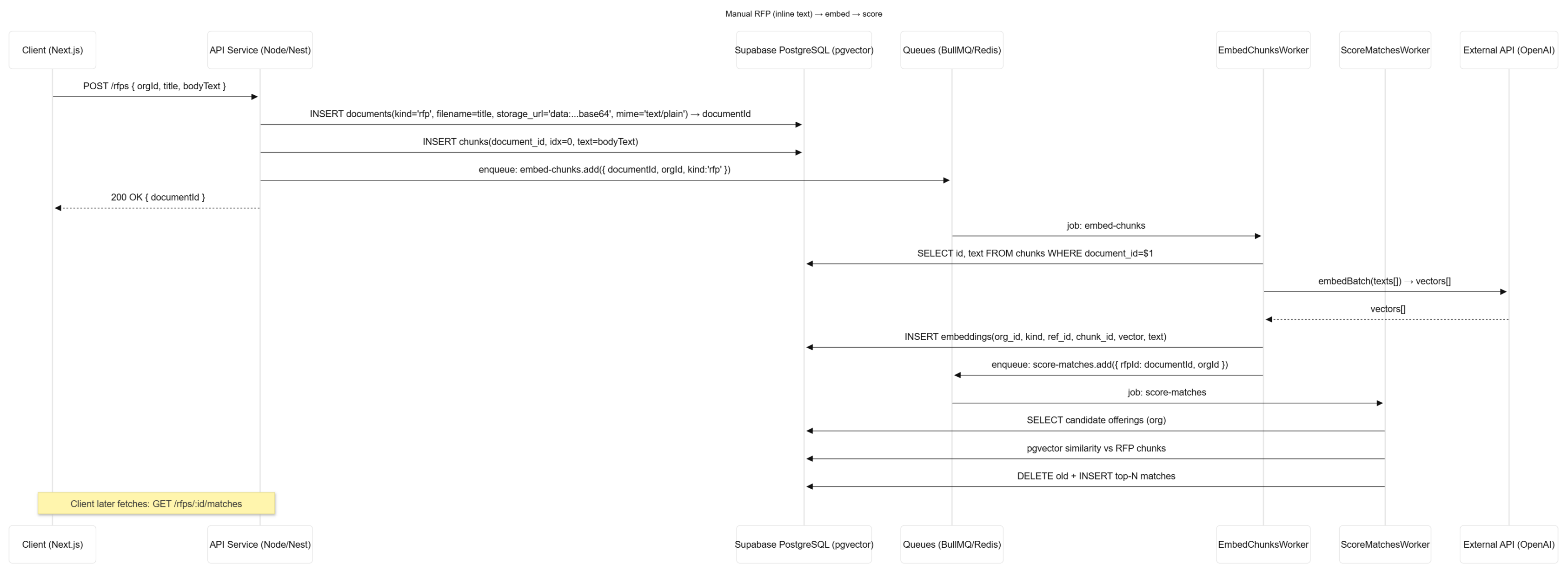

Manual RFP (inline text) → embed → score
Client (Next.js)
API Service (Node/Nest)
Supabase PostgreSQL (pgvector)
Queues (BullMQ/Redis)
EmbedChunksWorker
ScoreMatchesWorker
External API (OpenAI)
POST /rfps { orgId, title, bodyText }
INSERT documents(kind='rfp', filename=title, storage_url='data:...base64', mime='text/plain') → documentId
INSERT chunks(document_id, idx=0, text=bodyText)
enqueue: embed-chunks.add({ documentId, orgId, kind:'rfp' })
200 OK { documentId }
job: embed-chunks
SELECT id, text FROM chunks WHERE document_id=$1
embedBatch(texts[]) → vectors[]
vectors[]
INSERT embeddings(org_id, kind, ref_id, chunk_id, vector, text)
enqueue: score-matches.add({ rfpId: documentId, orgId })
job: score-matches
SELECT candidate offerings (org)
pgvector similarity vs RFP chunks
DELETE old + INSERT top-N matches
Client later fetches: GET /rfps/:id/matches

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

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

Listing 21: Express route: POST /rfps/upload (file-backed RFP)

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

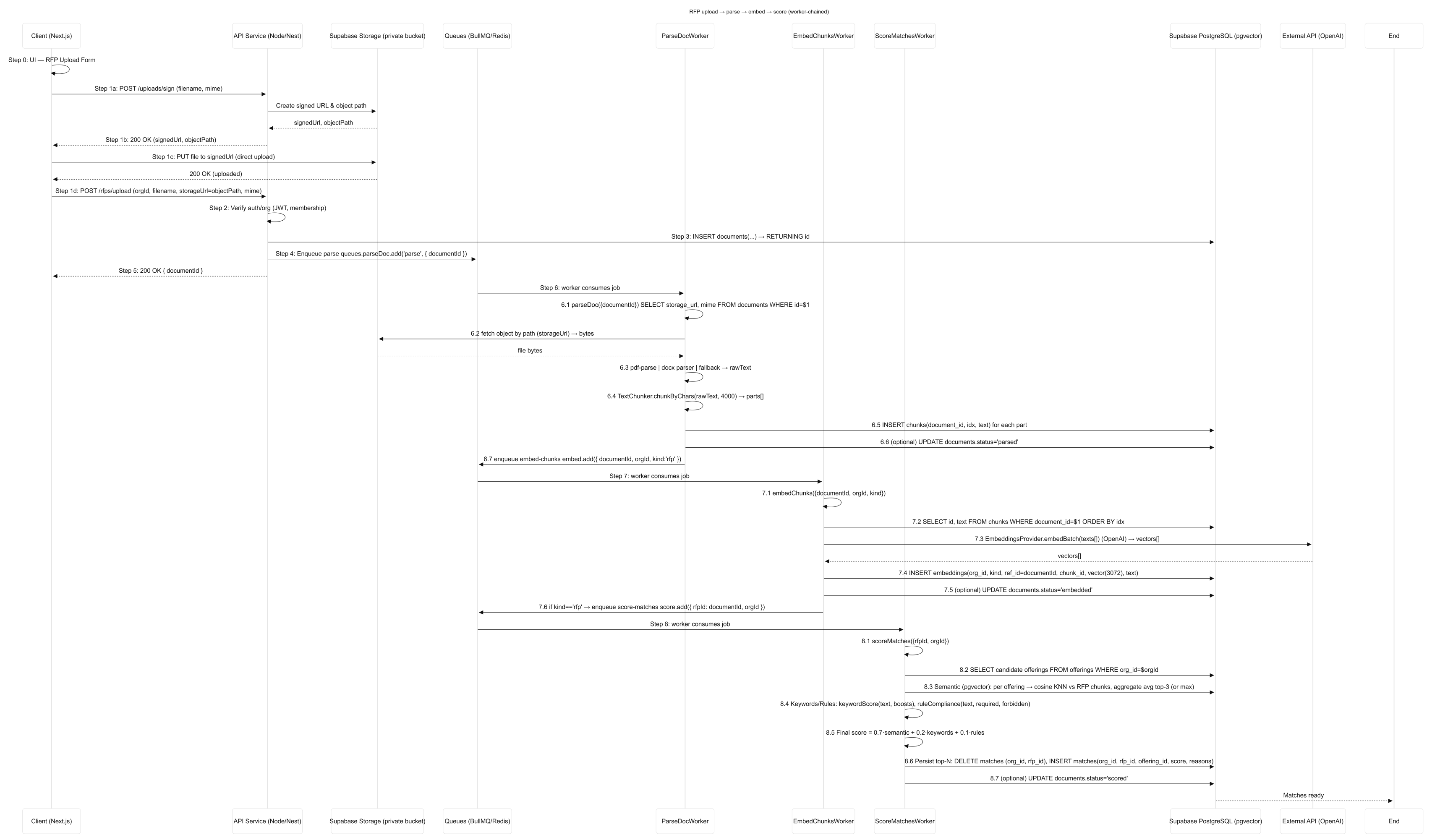

RFP upload → parse → embed → score (worker-chained)
Client (Next.js)
API Service (Node/Nest)
Supabase Storage (private bucket)
Queues (BullMQ/Redis)
ParseDocWorker
EmbedChunksWorker
ScoreMatchesWorker
Supabase PostgreSQL (pgvector)
External API (OpenAI)
End
Step 0: UI — RFP Upload Form
Step 1a: POST /uploads/sign (filename, mime)
Create signed URL & object path
signedUrl, objectPath
Step 1b: 200 OK (signedUrl, objectPath)
Step 1c: PUT file to signedUrl (direct upload)
200 OK (uploaded)
Step 1d: POST /rfps/upload (orgId, filename, storageUrl=objectPath, mime)
Step 2: Verify auth/org (JWT, membership)
Step 3: INSERT documents(...) → RETURNING id
Step 4: Enqueue parse queues.parseDoc.add('parse', { documentId })
Step 5: 200 OK { documentId }
Step 6: worker consumes job
6.1 parseDoc({documentId}) SELECT storage_url, mime FROM documents WHERE id=$1
6.2 fetch object by path (storageUrl) → bytes
file bytes
6.3 pdf-parse | docx parser | fallback → rawText
6.4 TextChunker.chunkByChars(rawText, 4000) → parts[]
6.5 INSERT chunks(document_id, idx, text) for each part
6.6 (optional) UPDATE documents.status='parsed'
6.7 enqueue embed-chunks embed.add({ documentId, orgId, kind:'rfp' })
Step 7: worker consumes job
7.1 embedChunks({documentId, orgId, kind})
7.2 SELECT id, text FROM chunks WHERE document_id=$1 ORDER BY idx
7.3 EmbeddingsProvider.embedBatch(texts[]) (OpenAI) → vectors[]
vectors[]
7.4 INSERT embeddings(org_id, kind, ref_id=documentId, chunk_id, vector(3072), text)
7.5 (optional) UPDATE documents.status='embedded'
7.6 if kind=='rfp' → enqueue score-matches score.add({ rfpId: documentId, orgId })
Step 8: worker consumes job
8.1 scoreMatches({rfpId, orgId})
8.2 SELECT candidate offerings FROM offerings WHERE org_id=$orgId
8.3 Semantic (pgvector): per offering → cosine KNN vs RFP chunks, aggregate avg top-3 (or max)
8.4 Keywords/Rules: keywordScore(text, boosts), ruleCompliance(text, required, forbidden)
8.5 Final score = 0.7·semantic + 0.2·keywords + 0.1·rules
8.6 Persist top-N: DELETE matches (org_id, rfp_id), INSERT matches(org_id, rfp_id, offering_id, score, reasons)
8.7 (optional) UPDATE documents.status='scored'
Matches ready
Client (Next.js)
API Service (Node/Nest)
Supabase Storage (private bucket)
Queues (BullMQ/Redis)
ParseDocWorker
EmbedChunksWorker
ScoreMatchesWorker
Supabase PostgreSQL (pgvector)
External API (OpenAI)
End

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

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

Listing 22: Express route: POST /rfps/:rfpId/score

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

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

```
// GET /rfps?limit=20&cursor=2025-09-01T10:00:00Z|f9b1...&q=soc2
app.get('/rfps', async (req, res) => {
  // NEVER trust orgId from query; derive from auth/JWT or RLS
  const orgId = req.user.orgId; // e.g., set by auth middleware

  const limit = Math.min(parseInt(String(req.query.limit ?? '20'), 10) || 20, 100);
  const cursor = typeof req.query.cursor === 'string' ? req.query.cursor : undefined;
  const q      = typeof req.query.q === 'string' ? req.query.q.trim() : '';

  // Parse cursor "created_at|id"
  let cursorCreatedAt: string | null = null;
  let cursorId: string | null = null;
  if (cursor) {
    const [ca, id] = cursor.split('|');
    cursorCreatedAt = ca || null;
    cursorId = id || null;
  }

  // Build dynamic WHERE
  const params: any[] = [orgId];
  let where = `d.org_id = $1 AND d.kind = 'rfp'`;
  if (q) {
    params.push(`%${q}%`);
    where += ‘ AND (d.filename ILIKE $${params.length})‘;
  }

  // Cursor predicate: stable tuple (created_at DESC, id DESC)
  let cursorSql = ’’;
  if (cursorCreatedAt && cursorId) {
    params.push(cursorCreatedAt, cursorId);
    cursorSql = ‘
      AND (d.created_at, d.id) < ($${params.length-1}::timestamptz, $${params.length}::uuid)
    ‘;
  }

  params.push(limit);

  const { rows } = await pool.query(
    ‘
    SELECT
      d.id                AS rfp_id,
      d.filename         AS title,
      d.created_at,
      COALESCE(r.last_scored_at, NULL) AS last_scored_at,
      -- lightweight stats
      (SELECT COUNT(*)::int FROM matches m WHERE m.org_id = d.org_id AND m.rfp_id = d.id) AS matches_count
    FROM documents d
    LEFT JOIN rfp_meta r
      ON r.org_id = d.org_id AND r.rfp_id = d.id
    WHERE ${where}
      ${cursorSql}
    ORDER BY d.created_at DESC, d.id DESC
    LIMIT $${params.length}
    ‘,
    params
  );

  // Next cursor is last row’s (created_at|id)
  const next =
    rows.length === limit
      ? ‘${rows[rows.length - 1].created_at.toISOString()}|${rows[rows.length - 1].rfp_id}‘
      : null;

  res.json({ items: rows, nextCursor: next });
```

```
});
```

Indexes & RLS.

Listing 24: Indexes/RLS to support GET /rfps

```
-- Speed up listing + cursor ordering
CREATE INDEX IF NOT EXISTS documents_rfp_list_idx
  ON documents (org_id, kind, created_at DESC, id);

-- Optional metadata for freshness
-- rfp_meta(org_id uuid, rfp_id uuid, last_scored_at timestamptz, PRIMARY KEY (org_id,
    rfp_id))

-- Row-level security (multitenancy)
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_rfps ON documents
  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 25: Express route: GET /rfps/:rfpId/matches

```
// GET /rfps/:rfpId/matches?limit=20&cursor=<offering_id>|null
app.get('/rfps/:rfpId/matches', async (req, res) => {
  const auth = req.user!;                    // derive orgId from auth
  const orgId = auth.orgId;                  // NOT from query
  const limit = Math.min(Number(req.query.limit ?? 20), 100);
  const cursor = req.query.cursor as string | undefined;

  const params: any[] = [orgId, req.params.rfpId, limit];
  const whereCursor = cursor ? `AND (m.score, m.offering_id) < (
      SELECT score, offering_id FROM matches
      WHERE org_id=$1 AND rfp_id=$2 AND offering_id=$4
    )` : ``;
  if (cursor) params.push(cursor);

  const { rows } = await pool.query(
    `
    SELECT m.offering_id,
           o.title,
           m.score,
           m.reasons,
           r.last_scored_at
    FROM matches m
    JOIN offerings o ON o.id = m.offering_id
    JOIN rfp_meta r ON r.org_id = m.org_id AND r.rfp_id = m.rfp_id
    WHERE m.org_id = $1 AND m.rfp_id = $2
      ${whereCursor}
    ORDER BY m.score DESC, m.offering_id ASC
    LIMIT $3
    `,
    params
  );

  // next cursor = last offering_id of this page (stable because of tiebreaker)
  const nextCursor = rows.length === limit ? rows[rows.length - 1].offering_id : null;
  res.json({ items: rows, nextCursor });
});
```

Indexes & RLS.

Listing 26: Indexes/RLS to support GET /rfps

```
CREATE INDEX IF NOT EXISTS matches_rfpid_rank_idx
  ON matches (org_id, rfp_id, score DESC, offering_id);

-- Optional: store when the worker last wrote results
-- rfp_meta(org_id, rfp_id, last_scored_at timestamptz)

ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_matches ON matches
  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 27: Express route: POST /rfps/:rfpId/match (idempotent enqueue)

```
// Admin-only; orgId must come from auth (e.g., req.user.orgId)
app.post('/rfps/:rfpId/match', async (req, res) => {
  const orgId = req.user.orgId; // derive from JWT/session; do NOT trust client input
  const rfpId = req.params.rfpId;

  // Optional tuning overrides (validated & bounded)
  const k     = Math.max(1, Math.min(Number(req.body?.k ?? 3), 10));      // top-k for
    semantic pooling
  const topN  = Math.max(1, Math.min(Number(req.body?.topN ?? 10), 100)); // number of
    matches to persist
  const reason = String(req.body?.reason ?? 'manual');

  // Existence check (and org scoping)
  const { rows: rfpRows } = await pool.query(
    `SELECT 1 FROM documents WHERE id=$1 AND org_id=$2 AND kind='rfp'`,
    [rfpId, orgId]
  );
  if (rfpRows.length === 0) {
    return res.status(404).json({ ok:false, error:'rfp_not_found' });
  }

  // Embeddings readiness (optional but helpful)
  const { rows: embRows } = await pool.query(
    `SELECT 1 FROM embeddings WHERE org_id=$1 AND kind='rfp' AND ref_id=$2 LIMIT 1`,
    [orgId, rfpId]
  );
  if (embRows.length === 0) {
    return res.status(409).json({ ok:false, error:'embeddings_not_ready' });
  }

  // Idempotent enqueue (one in-flight job per (org, rfp))
  const jobId = `score:${orgId}:${rfpId}`;
  await queues.scoreMatches.add(
    'score',
    { orgId, rfpId, k, topN, reason },
    {
      jobId,                    // dedupe
      attempts: 3,
      backoff: { type: 'exponential', delay: 2000 },
      removeOnComplete: true,
      removeOnFail: false
    }
  );

  return res.status(200).json({ ok:true, enqueued:true, jobId });
});
```

- **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 28: Express route: POST /offerings (create + embed enqueue)

```
// Admin/user auth middleware must set req.user.orgId (do not trust client-sent orgId)
app.post('/offerings', async (req, res) => {
  const orgId = req.user.orgId;
  const { title, description, tags } = req.body ?? {};

  // Basic validation
  if (typeof title !== 'string' || !title.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_title' });
  }
  if (typeof description !== 'string' || !description.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_description' });
  }
  if (tags && !Array.isArray(tags)) {
    return res.status(400).json({ ok:false, error:'invalid_tags' });
  }

  // Create offering
  const off = await pool.query(
    `INSERT INTO offerings(org_id, title, description, tags)
     VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
     RETURNING id`,
    [orgId, title, description, tags ?? []]
  );
  const offeringId = off.rows[0].id;

  // Create document(kind='offering') linked to offering
  const doc = await pool.query(
    `INSERT INTO documents(org_id, kind, ref_id, filename, mime)
     VALUES ($1, 'offering', $2, $3, 'text/plain')
     RETURNING id`,
    [orgId, offeringId, title]
  );
  const documentId = doc.rows[0].id;

  // Insert a single chunk (title + description); we can split further if long
  await pool.query(
    `INSERT INTO chunks(document_id, idx, text)
     VALUES ($1, 0, $2)`,
    [documentId, `${title}\n\n${description}`]
  );

  // Enqueue embeddings for this offering document
  await queues.embedChunks.add(
    'embed',
    { documentId, orgId, kind: 'offering' },
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
    }
  );

  return res.status(201).json({ ok:true, offeringId, documentId });
});
```

- **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 29: Indexes and RLS policy sketches

```sql
-- Fast lookup by org and recency
CREATE INDEX IF NOT EXISTS offerings_org_created_idx
  ON offerings (org_id, created_at DESC);

-- Documents/chunks for offerings
CREATE INDEX IF NOT EXISTS documents_offerings_idx
  ON documents (org_id, kind, ref_id);

ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_crud_offerings ON offerings
  USING (org_id = current_setting('app.org_id', true)::uuid)
  WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);
```

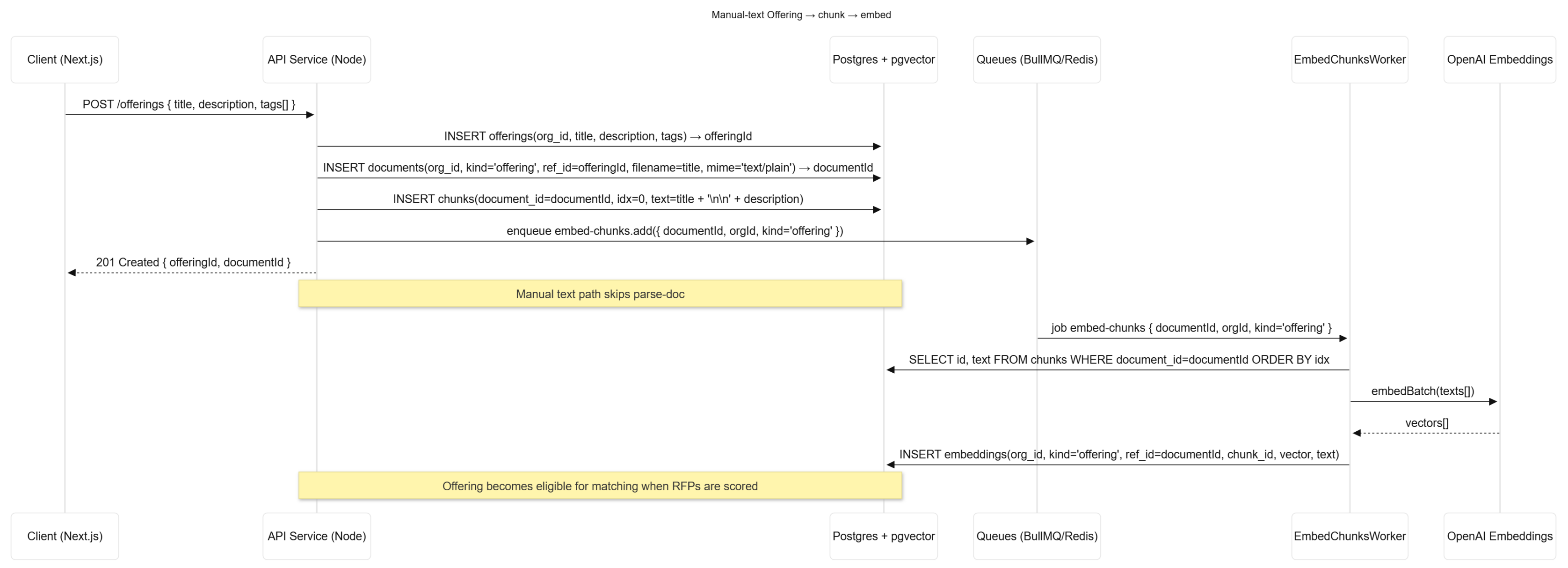

Manual-text Offering → chunk → embed
Client (Next.js)
API Service (Node)
Postgres + pgvector
Queues (BullMQ/Redis)
EmbedChunksWorker
OpenAI Embeddings
POST /offerings { title, description, tags[] }
INSERT offerings(org_id, title, description, tags) → offeringId
INSERT documents(org_id, kind='offering', ref_id=offeringId, filename=title, mime='text/plain') → documentId
INSERT chunks(document_id=documentId, idx=0, text=title + '\n\n' + description)
enqueue embed-chunks.add({ documentId, orgId, kind='offering' })
201 Created { offeringId, documentId }
Manual text path skips parse-doc
job embed-chunks { documentId, orgId, kind='offering' }
SELECT id, text FROM chunks WHERE document_id=documentId ORDER BY idx
embedBatch(texts[])
vectors[]
INSERT embeddings(org_id, kind='offering', ref_id=documentId, chunk_id, vector, text)
Offering becomes eligible for matching when RFPs are scored

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

```
// Auth middleware sets req.user.orgId; never trust client-sent orgId
app.post('/offerings/upload', async (req, res) => {
  const orgId = req.user.orgId;
  const { title, storageUrl, mime, tags } = req.body ?? {};

  // Basic validation
  if (typeof title !== 'string' || !title.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_title' });
  }
  if (typeof storageUrl !== 'string' || !storageUrl.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_storageUrl' });
  }
  if (typeof mime !== 'string' || !mime.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_mime' });
  }
  if (tags && !Array.isArray(tags)) {
    return res.status(400).json({ ok:false, error:'invalid_tags' });
  }

  // Create offering row
  const off = await pool.query(
    `INSERT INTO offerings(org_id, title, description, tags)
     VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
     RETURNING id`,
    [orgId, title, /* description */ '', tags ?? []]
  );
  const offeringId = off.rows[0].id;

  // Create document(kind='offering') pointing to the file
  const doc = await pool.query(
    `INSERT INTO documents(org_id, kind, ref_id, filename, storage_url, mime)
     VALUES ($1, 'offering', $2, $3, $4, $5)
    RETURNING id`,
    [orgId, offeringId, title, storageUrl, mime]
  );
  const documentId = doc.rows[0].id;

  // Enqueue parse → (embed) → (score-later when RFPs are scored)
  await queues.parseDoc.add(
    'parse',
    { documentId }, // parseDoc will enqueue embed-chunks
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
    }
  );

  return res.status(201).json({ ok:true, offeringId, documentId });
});
```

- **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.

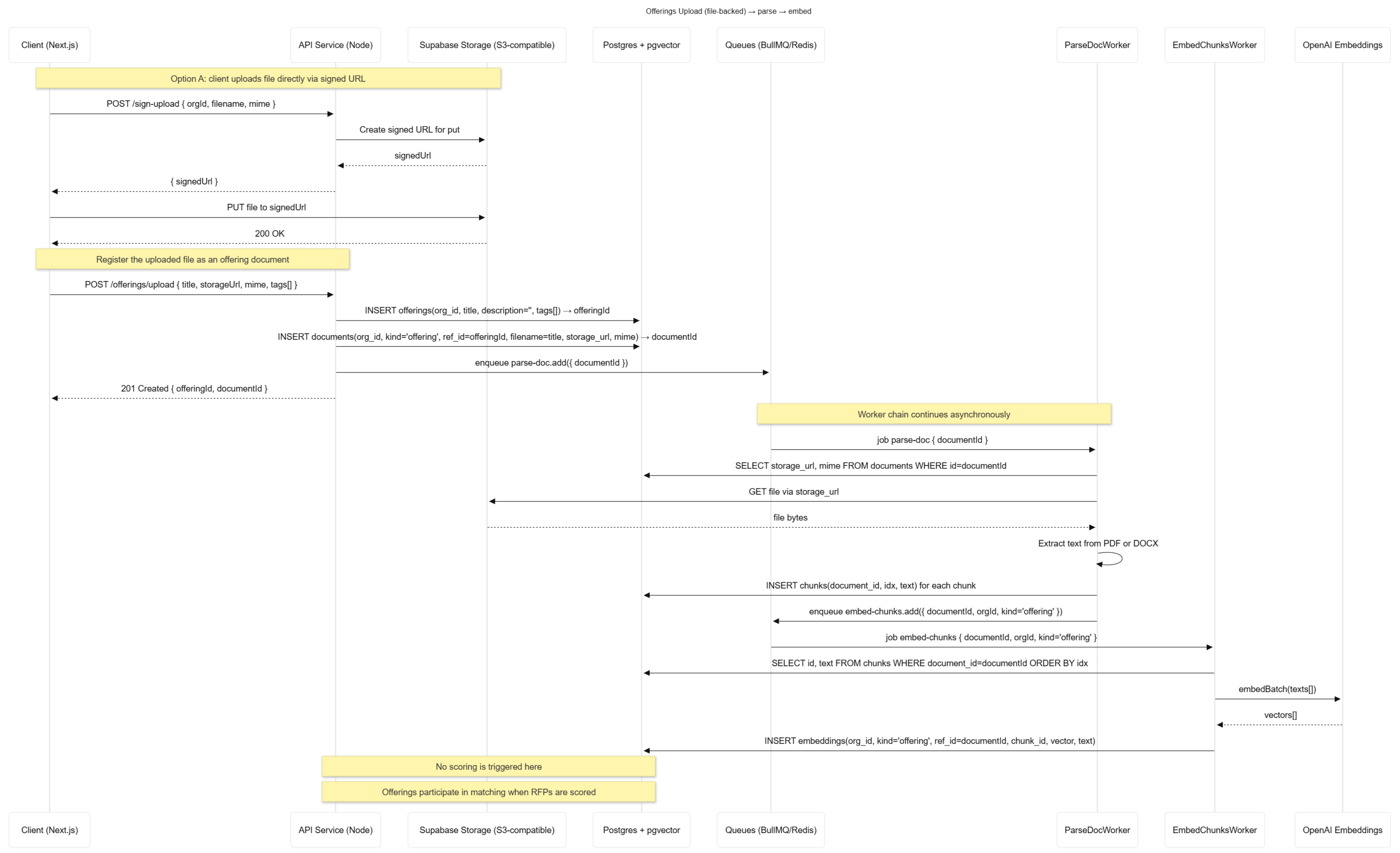

Offerings Upload (file-backed) → parse → embed
Client (Next.js)
API Service (Node)
Supabase Storage (S3-compatible)
Postgres + pgvector
Queues (BullMQ/Redis)
ParseDocWorker
EmbedChunksWorker
OpenAI Embeddings
Option A: client uploads file directly via signed URL
POST /sign-upload { orgId, filename, mime }
Create signed URL for put
signedUrl
{ signedUrl }
PUT file to signedUrl
200 OK
Register the uploaded file as an offering document
POST /offerings/upload { title, storageUrl, mime, tags[] }
INSERT offerings(org_id, title, description='', tags[]) → offeringId
INSERT documents(org_id, kind='offering', ref_id=offeringId, filename=title, storage_url, mime) → documentId
enqueue parse-doc.add({ documentId })
201 Created { offeringId, documentId }
Worker chain continues asynchronously
job parse-doc { documentId }
SELECT storage_url, mime FROM documents WHERE id=documentId
GET file via storage_url
file bytes
Extract text from PDF or DOCX
INSERT chunks(document_id, idx, text) for each chunk
enqueue embed-chunks.add({ documentId, orgId, kind='offering' })
job embed-chunks { documentId, orgId, kind='offering' }
SELECT id, text FROM chunks WHERE document_id=documentId ORDER BY idx
embedBatch(texts[])
vectors[]
INSERT embeddings(org_id, kind='offering', ref_id=documentId, chunk_id, vector, text)
No scoring is triggered here
Offerings participate in matching when RFPs are scored

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 31: Express route: POST /kb/upload

```
app.post('/kb/upload', requireAuth, withPgScope, async (req, res) => {
  const { title, storageUrl, mime } = req.body ?? {};
  if (typeof title !== 'string' || !title.trim())       return res.status(400).json({
    error:'invalid_title' });
  if (typeof storageUrl !== 'string' || !storageUrl)    return res.status(400).json({
    error:'invalid_storageUrl' });
  if (typeof mime !== 'string' || !mime.trim())         return res.status(400).json({
    error:'invalid_mime' });

  // Materialize a KB document (no orgId in SQL; RLS handles tenancy)
  const doc = await (req as any).pg.query(`
    INSERT INTO documents(kind, filename, storage_url, mime)
    VALUES ('kb', $1, $2, $3)
    RETURNING id
  `, [title, storageUrl, mime]);

  const documentId = doc.rows[0].id;

  // Enqueue parse -> (embed) via worker chain
  await queues.parseDoc.add(
    'parse',
    { documentId },
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
    }
  );

  return res.status(201).json({ ok: true, documentId });
});
```

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 32: Express route: GET /kb (list)

```
app.get('/kb', requireAuth, withPgScope, async (req, res) => {
  const q = (req.query.q as string | undefined)?.trim();
  const limit  = Math.min(Math.max(parseInt((req.query.limit as string) || '20', 10), 1),
     100);
  const offset = Math.max(parseInt((req.query.offset as string) || '0', 10), 0);

  const params: any[] = [];
  let where = `kind = 'kb'`;
  if (q) {
    params.push(`%${q.toLowerCase()}%`);
    where += ` AND lower(filename) LIKE $${params.length}`;
  }
  params.push(limit, offset);

  const { rows } = await (req as any).pg.query(
    `
    SELECT id AS document_id, filename, mime, created_at
    FROM documents
    WHERE ${where}
    ORDER BY created_at DESC
    LIMIT $${params.length - 1} OFFSET $${params.length}
    `,
    params
  );

  res.json({ items: rows, limit, offset });
});
```

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

```
// apps/api/src/server.ts
import { requireAuth } from './auth';
import { withPgScope } from './pg-scope';
import { draftAssistant } from './svc.assistant';

app.post('/assistant/draft', requireAuth, withPgScope, async (req, res) => {
  // orgId is derived from JWT by requireAuth and applied to the PG session by
    withPgScope
  const { rfpId, question } = req.body || {};
  if (!rfpId || !question) {
    return res.status(400).json({ error: 'rfpId_and_question_required' });
  }

  const out = await draftAssistant({
    rfpId,
    question,
    client: (req as any).pg   // request-scoped PG client with SET LOCAL app.org_id
    already applied
  });

  res.json(out);
});
```

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 34: Example: POST /assistant/draft (HTTP)

```
POST /assistant/draft
```

```
Content-Type: application/json
Authorization: Bearer <JWT>

{
  "rfpId": "7be3a4fa-5db9-4a19-8b4b-7a7b3f9c1d10",
  "question": "Draft the Security & Compliance section focusing on SOC 2 Type II and 24/7
     support."
}
```

Listing 35: Example response

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

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

Listing 36: Expose queues to API layer

```
import { makeQueue, defaultJobOpts } from '../worker/src/queue';
export const queues = {
  parseDoc: makeQueue('parse-doc').q,
  embedChunks: makeQueue('embed-chunks').q,
  scoreMatches: makeQueue('score-matches').q,
};
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 37: Retrieve top chunks (RFP + KB) semantically and call OpenAI

```
// apps/api/src/svc.assistant.ts
import OpenAI from 'openai';
import type { PoolClient } from 'pg';

const oai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

/**
 * NOTE:
 *  - `orgId` is NOT accepted from the client. Tenant scoping is enforced by Postgres RLS
    .
 *  - Use the request-scoped `client` (middleware sets `SET LOCAL app.org_id = ...`).
 */
export async function draftAssistant({
  rfpId,
  question,
  client
}: {
  rfpId: string;
  question: string;
  client: PoolClient;   // request-scoped client with RLS org set
}) {
  // Embed the question once
  const emb = await oai.embeddings.create({
    model: 'text-embedding-3-large',
    input: question
  });
  const qvec = ‘[${emb.data[0].embedding.join(’,’)}]‘;

  // Semantic KNN over this RFP + org KB (RLS limits rows to caller’s org)
  const { rows: hits } = await client.query(
    ‘
    WITH cand AS (
      SELECT kind, ref_id, text, vector
      FROM embeddings
      WHERE (kind = ’rfp’ AND ref_id = $1) OR kind = ’kb’
    )
    SELECT kind,
           ref_id,
           text,
           1 - (vector <=> $2::vector) AS sim
    FROM cand
    ORDER BY vector <=> $2::vector      -- cosine distance asc
    LIMIT 12
    ‘,
    [rfpId, qvec]
  );

  // Light dedupe + clamp to token budget
  const seen = new Set<string>();
  const top: Array<{ kind: string; ref_id: string; text: string; sim: number }> = [];
  for (const h of hits) {
    const text = (h.text || ’’) as string;
    const key = text.slice(0, 160).toLowerCase(); // cheap near-dup heuristic
    if (seen.has(key)) continue;
    seen.add(key);
    top.push({ kind: h.kind, ref_id: h.ref_id, text, sim: Number(h.sim) });
    if (top.length >= 8) break;
  }
  const ctxTexts = top.map(h => h.text.slice(0, 1200)); // trim long chunks

  // Prompt
  const messages = [
    {
      role: ’system’ as const,
      content:
        ’You are a proposal assistant. Answer strictly using the provided CONTEXT. ’ +
        ’If information is missing, state what is needed.’
    },
    {
      role: 'user' as const,
      content: `CONTEXT:\n${ctxTexts.join('\n---\n')}\n\nQUESTION:\n${question}`
    }
  ];

  // Call LLM
  const resp = await oai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages,
    temperature: 0.2
  });

  // Return draft + useful citations
  return {
    text: resp.choices[0]?.message?.content ?? '',
    sources: top.map((h, i) => ({
      rank: i + 1,
      kind: h.kind,
      refId: h.ref_id,
      sim: h.sim
    }))
  };
}
```

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 38: Create a manual-text RFP and fetch matches

```
'use client';
import { useState } from 'react';

export default function NewRfpPage() {
  const [title, setTitle] = useState('');
  const [body, setBody]   = useState('');
  const [rfpDocId, setRfpDocId] = useState();
  const orgId = '00000000-0000-0000-0000-000000000001'; // from session in real app

  async function createRfp() {
    const r = await fetch('http://localhost:3001/rfps', {
      method:'POST', headers:{'Content-Type':'application/json'},
      body: JSON.stringify({ orgId, title, bodyText: body })
    }).then(r=>r.json());

    setRfpDocId(r.documentId);
    alert('RFP created. Parsing, embeddings, and scoring are queued automatically.');
  }

  return (
    <div style={{maxWidth:720, margin:'2rem auto'}}>
      <h1>New RFP</h1>
      <input value={title} onChange={e=>setTitle(e.target.value)} placeholder="Title"
    style={{width:'100%', padding:8}}/>
      <textarea value={body} onChange={e=>setBody(e.target.value)} placeholder="Paste the
     RFP body..." rows={12} style={{width:'100%', marginTop:8}}/>
      <button onClick={createRfp} style={{marginTop:12}}>Create & Queue</button>
      {rfpDocId && <Matches rfpId={rfpDocId} orgId={orgId} />}
    </div>
  );
}

function Matches({ rfpId, orgId }) {
  const [items, setItems] = useState([]);
  async function refresh() {
    const r = await fetch(`http://localhost:3001/rfps/${rfpId}/matches?orgId=${orgId}`).
    then(r=>r.json());
    setItems(r);
  }
  return (
    <div style={{marginTop:24}}>
      <button onClick={refresh}>Refresh Matches</button>
      <ul>
        {items.map(x=>(
          <li key={x.offering_id}>
            <strong>{x.title}</strong> -- score {x.score.toFixed(3)}
          </li>
        ))}
      </ul>
    </div>
  );
```

```
}
```

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

Listing 39: Ask the assistant to draft grounded content

```tsx
'use client';
import { useState } from 'react';

export default function AssistantPage() {
  const [question, setQ] = useState('Draft the Security & Compliance section.');
  const [answer, setAns] = useState('');
  const orgId = '00000000-0000-0000-0000-000000000001';
  const rfpId = '...'; // pass from navigation/state

  async function ask() {
    const r = await fetch('http://localhost:3001/assistant/draft', {
      method:'POST', headers:{'Content-Type':'application/json'},
      body: JSON.stringify({ orgId, rfpId, question })
    }).then(r=>r.json());
    setAns(r.text);
  }

  return (
    <div style={{maxWidth:720, margin:'2rem auto'}}>
      <h1>Assistant</h1>
      <textarea value={question} onChange={e=>setQ(e.target.value)} rows={5} style={{
    width:'100%'}}/>
      <button onClick={ask} style={{marginTop:8}}>Generate</button>
      <pre style={{whiteSpace:'pre-wrap', background:'#fafafa', padding:12, marginTop
    :12}}>{answer}</pre>
    </div>
  );
}
```

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).

```
'use client';
import { useState } from 'react';
import { useRouter } from 'next/navigation';

export default function LoginPage() {
  const r = useRouter();
  const [email, setEmail] = useState('jane@acme.com');
  const [password, setPassword] = useState('');
  const [msg, setMsg] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);

  async function onSubmit(e: React.FormEvent) {
    e.preventDefault();
    setLoading(true);
    setMsg('Signing you in...');
    const res = await fetch('/api/login', {
      method: 'POST',
      body: JSON.stringify({ email, password }),
      headers: { 'Content-Type': 'application/json' }
    });
    setLoading(false);
    if (res.ok) {
      setMsg('Redirecting to dashboard...');
      r.push('/dashboard');
    } else {
      const { error } = await res.json();
      setMsg(error ?? 'Login failed');
    }
  }

  return (
    <section style={{ maxWidth: 420, margin: '80px auto' }}>
      <h2>Welcome back</h2>
      <p className="muted">Sign in to continue</p>
      <form onSubmit={onSubmit}>
        <label>
          Email
          <input
            aria-label="Email"
            value={email}
            onChange={e => setEmail(e.target.value)}
          />
        </label>
        <label>
          Password
          <input
            aria-label="Password"
            type="password"
            value={password}
            onChange={e => setPassword(e.target.value)}
          />
        </label>
        <button type="submit" aria-label="Sign In" disabled={loading}>
          Sign In
        </button>
      </form>
      {msg && <div role="status">{msg}</div>}
      <p><a href="#">Forgot password?</a></p>
    </section>
  );
}
```

14.2.2 Unit & Component Tests.

14.2.2.1 Vitest + React Testing Library

Install.

```
pnpm add -D vitest @testing-library/react @testing-library/jest-dom jsdom @types/jest
    user-event
```

vitest.config.ts.

```
import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
    environment: 'jsdom',
    setupFiles: ['./vitest.setup.ts'],
    globals: true,
    css: true,
    coverage: { reporter: ['text', 'lcov'] }
  }
});
```

vitest.setup.ts.

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

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

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

describe('Login page', () => {
  it('renders fields, actions, and defaults', () => {
    render(<LoginPage />);
    expect(screen.getByRole('heading', { name: /welcome back/i })).toBeInTheDocument();
    expect(screen.getByText(/sign in to continue/i)).toBeInTheDocument();

    const email = screen.getByLabelText(/email/i);
    const password = screen.getByLabelText(/password/i);
    const submit = screen.getByRole('button', { name: /sign in/i });
    const forgot = screen.getByRole('link', { name: /forgot password\?/i });

    expect(email).toBeInTheDocument();
    expect(password).toBeInTheDocument();
    expect(submit).toBeEnabled();
    expect(forgot).toHaveAttribute('href', '#');
  });

  it('accepts input and shows a submitting state', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.clear(screen.getByLabelText(/email/i));
    await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
    await user.clear(screen.getByLabelText(/password/i));
    await user.type(screen.getByLabelText(/password/i), 's3cret!');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByRole('status')).toHaveTextContent(/signing you in/i);
  });
});
```

14.2.2.2 MSW (Mock Service Worker)

tests/msw/handlers.ts.

```
import { http, HttpResponse } from 'msw';

export const handlers = [
  http.post('/api/login', async ({ request }) => {
    const body = await request.json() as { email: string; password: string };
    if (body.email === 'dev@bidion.io' && body.password === 's3cret!') {
      return HttpResponse.json({ token: 'jwt-123', user: { email: body.email } }, {
    status: 200 });
    }
    return HttpResponse.json({ error: 'Invalid credentials' }, { status: 401 });
  }),
];
```

tests/msw/server.ts.

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

Wire MSW in vitest.setup.ts.

```
import '@testing-library/jest-dom';
import { server } from './tests/msw/server';

beforeAll(() => server.listen({ onUnhandledRequest: 'error' }));
afterEach(() => server.resetHandlers());
afterAll(() => server.close());
```

page.msw.spec.tsx.

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

```

describe('Login with MSW', () => {
  it('redirects to dashboard on success', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
    await user.type(screen.getByLabelText(/password/i), 's3cret!');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByText(/redirecting to dashboard/i)).toBeInTheDocument();
  });

  it('shows error on invalid credentials', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.type(screen.getByLabelText(/email/i), 'wrong@user.io');
    await user.type(screen.getByLabelText(/password/i), 'nope');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByRole('alert')).toHaveTextContent(/invalid credentials/i);
  });
});
```

14.2.3 E2E & Browser Automation.

14.2.3.1 Playwright (E2E)

playwright.config.ts (optional).

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

e2e/login.spec.ts.

```
import { test, expect } from '@playwright/test';

test('login success routes to dashboard', async ({ page }) => {
  await page.goto('/login');
  await expect(page.getByRole('heading', { name: /welcome back/i })).toBeVisible();
  await page.getByLabel('Email').fill('dev@bidion.io');
  await page.getByLabel('Password').fill('s3cret!');
  await page.getByRole('button', { name: /sign in/i }).click();
  await expect(page).toHaveURL(/\/dashboard$/);
  await expect(page.getByRole('heading', { name: /dashboard/i })).toBeVisible();
});
```

```

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

14.2.3.2 Cypress (Alternative E2E)

cypress/e2e/login.cy.ts.

```
describe('Login', () => {
  it('logs in successfully', () => {
    cy.visit('/login');
    cy.findByRole('heading', { name: /welcome back/i }).should('be.visible');
    cy.findByLabelText(/email/i).clear().type('dev@bidion.io');
    cy.findByLabelText(/password/i).clear().type('s3cret!');
    cy.findByRole('button', { name: /sign in/i }).click();
    cy.url().should('match', /\/dashboard$/);
    cy.findByRole('heading', { name: /dashboard/i }).should('be.visible');
  });

  it('shows error on bad creds', () => {
    cy.visit('/login');
    cy.findByLabelText(/email/i).type('wrong@user.io');
    cy.findByLabelText(/password/i).type('nope');
    cy.findByRole('button', { name: /sign in/i }).click();
    cy.findByRole('alert').should('contain.text', 'Invalid credentials');
  });
});
```

14.2.4 Visual & Accessibility.

14.2.4.1 Storybook (Visual Catalog)

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

```
import type { Meta, StoryObj } from '@storybook/react';
import LoginPage from './page';

const meta: Meta<typeof LoginPage> = {
  title: 'Auth/LoginPage',
  component: LoginPage,
  parameters: { layout: 'centered' }
};
export default meta;

export const Default: StoryObj<typeof LoginPage> = {};
export const ErrorState: StoryObj<typeof LoginPage> = {
  args: { initialError: 'Invalid credentials' }
};
```

14.2.4.2 Accessibility

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

```
import { render } from '@testing-library/react';
import { axe, toHaveNoViolations } from 'jest-axe';
import LoginPage from './page';

expect.extend(toHaveNoViolations);

it('has no obvious a11y violations', async () => {
  const { container } = render(<LoginPage />);
  const results = await axe(container);
  expect(results).toHaveNoViolations();
});
```

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

```
import { test, expect } from '@playwright/test';
import AxeBuilder from '@axe-core/playwright';

test('login page accessibility', async ({ page }) => {
  await page.goto('/login');
  const results = await new AxeBuilder({ page }).analyze();
  expect(results.violations).toEqual([]);
});
```

14.2.4.3 Lighthouse CI (Perf/A11y Budgets)

lighthouserc.json.

```
{
  "ci": {
    "collect": {
      "numberOfRuns": 2,
      "url": ["http://localhost:3000/login"]
    },
    "assert": {
      "assertions": {
        "categories:performance": ["warn", { "minScore": 0.8 }],
        "categories:accessibility": ["error", { "minScore": 0.9 }]
      }
    },
    "upload": { "target": "temporary-public-storage" }
  }
}
```

Run.

```
npx lhci autorun
```

14.2.5 Static Analysis & Type Safety

14.2.5.1 TypeScript strict.

```
{
  "compilerOptions": {
    "strict": true,
    "noUncheckedIndexedAccess": true,
    "noImplicitOverride": true
  }
}
```

14.2.5.2 ESLint (Next.js base).

```
module.exports = {
  extends: ['next/core-web-vitals'],
  rules: {
    '@next/next/no-img-element': 'off'
  }
};
```

14.2.5.3 Dead code scan (knip).

```
{
  "scripts": {
    "scan:dead": "knip --reporter compact"
  },
  "devDependencies": { "knip": "^5.0.0" }
}
```

14.2.5.4 Import rules and cycles (dependency-cruiser).

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

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 40: docker-compose.test.yml

```
# docker-compose.test.yml
version: "3.9"
services:
  db:
    image: supabase/postgres:15
    environment:
      POSTGRES_PASSWORD: dev
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
  api:
    build: ./apps/api
    environment:
      DATABASE_URL: postgres://postgres:dev@db:5432/postgres
      REDIS_URL: redis://redis:6379
      NODE_ENV: test
    depends_on: [db, redis]
    command: ["npm","run","test:integration"]
```

Database schema & minimal seed for tests.

Listing 41: test/schema.sql

```sql
-- test/schema.sql (run via migration tool for test)
CREATE SCHEMA IF NOT EXISTS test;
-- Enable extensions used by MVP:
CREATE EXTENSION IF NOT EXISTS vector;

-- RLS sample toggle (ensure we test with RLS ON):
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
-- ... policies reference current_setting('app.org_id', true);

-- Fast seed (org, 1 RFP, 2 offerings):
-- (Use transactions; keep deterministic IDs for tests)
```

Org-scope flag for DB operations.

Listing 42: Set org scope in tests

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

14.3.1 Unit & Integration.

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

Listing 43: apps/api/test/rfps.e2e-lite.spec.ts

```typescript
import { Test } from '@nestjs/testing';
import { INestApplication, ValidationPipe } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';

describe('RFP Matches (e2e-lite)', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
    await app.init();
  });

  afterAll(async () => { await app.close(); });

  it('GET /rfps/:id/matches returns ranked list', async () => {
    const rfpId = '00000000-0000-0000-0000-000000000001';
    await request(app.getHttpServer())
      .get(`/rfps/${rfpId}/matches`)
      .set('Authorization', 'Bearer TEST_JWT')
      .expect(200)
      .expect(res => {
        expect(Array.isArray(res.body)).toBe(true);
        for (let i = 1; i < res.body.length; i++) {
          expect(res.body[i - 1].score).toBeGreaterThanOrEqual(res.body[i].score);
        }
      });
  });
});
```

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

Listing 44: apps/api/test/rfps.dto.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication, ValidationPipe } from '@nestjs/common'];
import request from 'supertest';
import { AppModule } from '../src/app.module';

describe('RFP Create DTO validation', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true
    }));
    await app.init();
  });

  afterAll(async () => { await app.close(); });

  it('rejects invalid body', async () => {
    await request(app.getHttpServer())
      .post('/rfps')
      .set('Authorization', 'Bearer TEST_JWT')
      .send({ badField: 'nope' })
      .expect(400);
  });
});
```

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

Listing 45: apps/api/test/tc.setup.ts (Jest globalSetup)

```
import { GenericContainer, StartedTestContainer } from 'testcontainers';
import { PostgreSqlContainer, StartedPostgreSqlContainer } from '@testcontainers/
    postgresql';

declare global {
  // eslint-disable-next-line no-var
  var __PG__: StartedPostgreSqlContainer;
  // eslint-disable-next-line no-var
  var __REDIS__: StartedTestContainer;
}

export default async function globalSetup() {
  const pg = await new PostgreSqlContainer('postgres:15')
    .withEnvironment({ POSTGRES_PASSWORD: 'dev' })
    .start();

  // Create pgvector + base schema
  const { Client } = await import('pg');
  const client = new Client({
    connectionString: pg.getConnectionUri(),
  });
  await client.connect();
  await client.query(`CREATE EXTENSION IF NOT EXISTS vector;`);
  await client.query(`SET ROLE postgres;`); // adjust if needed
  // Minimal tenant env: org scope GUC and tables must exist in migrations
  await client.end();

  const redis = await new GenericContainer('redis:7').withExposedPorts(6379).start();

  process.env.DATABASE_URL = pg.getConnectionUri();
  process.env.REDIS_URL = `redis://127.0.0.1:${redis.getMappedPort(6379)}`;

  // Expose on global for teardown
  // @ts-ignore
  global.__PG__ = pg;
  // @ts-ignore
  global.__REDIS__ = redis;
}
```

Listing 46: apps/api/test/tc.teardown.ts (Jest globalTeardown)

```
export default async function globalTeardown() {
  // @ts-ignore
  const pg = global.__PG__;
  // @ts-ignore
  const redis = global.__REDIS__;
  if (pg) await pg.stop();
  if (redis) await redis.stop();
}
```

Listing 47: apps/api/jest.config.ts (hook setup/teardown)

```
import type { Config } from 'jest';

const config: Config = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  globalSetup: '<rootDir>/test/tc.setup.ts',
  globalTeardown: '<rootDir>/test/tc.teardown.ts',
  setupFilesAfterEnv: ['<rootDir>/test/setup.env.ts'],
};

export default config;
```

Listing 48: apps/api/test/setup.env.ts

```
process.env.NODE_ENV = 'test';
process.env.APP_JWT_PUBLIC_KEY = 'TEST_KEY';
// ...anything else the app reads
```

14.3.1.4 Worker Queue Test (BullMQ Roundtrip) with Redis.

Listing 49: apps/worker/test/queue.roundtrip.spec.ts

```
import { Queue, Worker, Job } from 'bullmq';
import IORedis from 'ioredis';

describe('BullMQ roundtrip', () => {
  const connection = new IORedis(process.env.REDIS_URL as string);
  const qName = 'test-q';
  let queue: Queue;
  let results: any[] = [];

  beforeAll(async () => {
    queue = new Queue(qName, { connection });
    // Worker that echoes payload and writes to Postgres (stubbed here)
    new Worker(qName, async (job: Job) => {
      // simulate some work
      return { received: job.data };
    }, { connection }).on('completed', (_, res) => results.push(res));
  });

  afterAll(async () => {
    await queue.drain(true);
    await queue.close();
    await connection.quit();
  });

  it('processes embed-chunks job', async () => {
    await queue.add('embed-chunks', { documentId: 'doc-1', orgId: 'org-1' });
    // naive wait for worker to run in test, better: event gating
    await new Promise(res => setTimeout(res, 500));
    expect(results[0]).toEqual({ received: { documentId: 'doc-1', orgId: 'org-1' } });
  });
});
```

14.3.1.5 Mock outbound HTTP (OpenAI) with nock.

Listing 50: apps/worker/test/openai.mock.spec.ts

```
import nock from 'nock';
import { embedChunks } from '../src/embeddings'; // your adapter that calls OpenAI REST

describe('OpenAI embeddings adapter', () => {
  beforeAll(() => {
    nock.disableNetConnect(); // block real network
    nock('https://api.openai.com')
      .post('/v1/embeddings')
      .reply(200, {
        data: [{ embedding: Array(3072).fill(0.01), index: 0 }],
        model: 'text-embedding-3-large',
        object: 'list'
      });
  });

  afterAll(() => nock.enableNetConnect());

  it('returns embeddings from mocked OpenAI', async () => {
    const vec = await embedChunks(['hello world']);
    expect(vec).toHaveLength(1);
    expect(vec[0]).toHaveLength(3072);
  });
});
```

14.3.2 Contract & API Schema.

14.3.2.1 OpenAPI Schema Checks (jest-openapi).

Listing 51: Install jest-openapi

```
pnpm add -D jest-openapi supertest
```

Listing 52: apps/api/test/openapi.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';
import 'jest-openapi';

describe('OpenAPI contract', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    await app.init();

    // Load the generated swagger JSON (exported at build or served in dev)
    const openapi = require('../openapi.json'); // ensure generated before test
    expect(openapi).toBeDefined();
    expect.extend({ toSatisfyApiSpec: (global as any).toSatisfyApiSpec });
    (global as any).jestOpenAPI(openapi);
  });

  afterAll(async () => { await app.close(); });

  it('GET /rfps satisfies OpenAPI spec', async () => {
    const res = await request(app.getHttpServer())
      .get('/rfps')
      .set('Authorization', 'Bearer TEST_JWT')
      .expect(200);

    expect(res.body).toSatisfyApiSpec();
  });
});
```

14.3.2.2 Pact (consumer-driven contract) example.

Listing 53: contracts/consumer.rfps.pact.spec.ts

```
import path from 'path';
import { Pact } from '@pact-foundation/pact';
import fetch from 'node-fetch';

describe('Consumer pact for /rfps', () => {
  const provider = new Pact({
    consumer: 'BidionWeb',
    provider: 'BidionAPI',
    dir: path.resolve(process.cwd(), 'pacts'),
    logLevel: 'warn',
  });

  beforeAll(() => provider.setup());
  afterAll(() => provider.finalize());

  it('lists RFPs', async () => {
    await provider.addInteraction({
      state: 'there are RFPs',
      uponReceiving: 'a request for list of RFPs',
      withRequest: { method: 'GET', path: '/rfps', headers: { Authorization: 'Bearer TEST_JWT' } },
      willRespondWith: {
        status: 200,
        headers: { 'Content-Type': 'application/json; charset=utf-8' },
        body: [{ id: 'rfp-1', title: 'Sample RFP' }]
      }
    });

    const res = await fetch(`${provider.mockService.baseUrl}/rfps`, {
      headers: { Authorization: 'Bearer TEST_JWT' }
    });
    const json = await res.json();
    expect(json[0]).toBeDefined();
    expect(json[0].id).toBe('rfp-1');

    await provider.verify();
  });
});
```

14.3.3 Load & Resilience.

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

We use **k6** to generate realistic API and background load. Targets reflect SLOs tracked by Prometheus (p95 latency $\leq 500\,\text{ms}$ for API; KNN p95 $\leq 200\,\text{ms}$ under nominal load). Thresholds fail builds on breach.

14.3.3.2 k6 script (scenarios + thresholds).

```
/**
 * k6 run -e BASE_URL=http://localhost:3000 \
 *        --vus 50 --duration 5m \
 *        --summary-export=./reports/k6-summary.json \
 *        script.js
 *
 * Optionally emit JSON:  --out json=./reports/k6.json
 */
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Trend, Rate } from 'k6/metrics';

const BASE_URL = __ENV.BASE_URL || 'http://localhost:3000';
const t_api = new Trend('api_latency');
const r_errors = new Rate('api_errors');

export const options = {
  thresholds: {
    'http_req_failed': ['rate<0.02'],            // <2% request failures
    'http_req_duration{kind:api}': ['p(95)<500'],// p95 < 500ms
    'api_latency': ['p(95)<500'],
    'api_errors': ['rate<0.02'],
  },
  scenarios: {
    readMatches: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '1m', target: 20 },
        { duration: '3m', target: 50 },
        { duration: '1m', target: 0 },
      ],
      exec: 'getMatches',
      tags: { kind: 'api' },
    },
    draftAssistant: {
      executor: 'constant-arrival-rate',
      rate: 20, timeUnit: '1s',
      duration: '3m',
      preAllocatedVUs: 50, maxVUs: 100,
      exec: 'postDraft',
      tags: { kind: 'api' },
    },
  },
};

export function getMatches() {
  const r = http.get(`${BASE_URL}/rfps/00000000-0000-0000-0000-000000000001/matches`, {
    headers: { Authorization: 'Bearer TEST_JWT' },
  });
  t_api.add(r.timings.duration, { kind: 'api' });
  r_errors.add(r.status >= 500);
  check(r, { '200 OK': (res) => res.status === 200 });
  sleep(0.3);
}

export function postDraft() {
  const payload = JSON.stringify({ question: 'Draft Security & Compliance' });
  const r = http.post(`${BASE_URL}/assistant/draft`, payload, {
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer TEST_JWT'
    },
  });
  t_api.add(r.timings.duration, { kind: 'api' });
  r_errors.add(r.status >= 500);
  check(r, { '201/200': (res) => res.status === 200 || res.status === 201 });
  sleep(0.5);
}
```

14.3.3.3 Worker/queue throughput probe (smoke).

```
# Produce N jobs and measure end-to-end TTL (queue -> done)
curl -H "Authorization: Bearer TEST_JWT" \
  -H "Content-Type: application/json" \
  -d '{"count": 500, "kind": "embed-chunks"}' \
  "${BASE_URL}/debug/enqueue"
```

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

Listing 54: .github/workflows/perf-k6.yml

```
# .github/workflows/perf-k6.yml
name: k6-perf
on:
  pull_request:
    paths: ['apps/api/**','apps/worker/**']
jobs:
  k6:
    runs-on: ubuntu-latest
    services:
      db:
        image: supabase/postgres:15
        ports: ['5432:5432']
        env: { POSTGRES_PASSWORD: dev }
      redis:
        image: redis:7
        ports: ['6379:6379']
    steps:
      - uses: actions/checkout@v4
      - uses: grafana/setup-k6-action@v1
      - name: Boot API
        run: |
          pnpm i && pnpm -C apps/api start:test &
          sleep 8
      - name: Run k6
        run: |
          k6 run --vus 20 --duration 1m \
            -e BASE_URL=http://localhost:3000 \
            --summary-export=./reports/k6-summary.json \
            scripts/k6/script.js
      - name: Upload k6 summary
        uses: actions/upload-artifact@v4
        with: { name: k6-summary, path: reports/k6-summary.json }
```

14.3.4 Security & Policy.

14.3.4.1 DTO invariants with class-validator (unit).

Listing 55: apps/api/test/dto.unit.spec.ts

```
import { validateSync } from 'class-validator';
import { CreateRfpDto } from '../src/rfps/dto/create-rfp.dto';

describe('CreateRfpDto', () => {
  it('requires title and body', () => {
    const dto = new CreateRfpDto();
    // @ts-ignore
    dto.title = '';
    // @ts-ignore
    dto.body = undefined;
    const errors = validateSync(dto);
    expect(errors.length).toBeGreaterThan(0);
  });
});
```

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

Listing 56: .github/workflows/secure-lint.yml

```
name: secure-lint
on: [pull_request]
jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: pnpm/action-setup@v4
      - run: pnpm i
      - run: pnpm dlx eslint . --max-warnings=0
      - 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 57: apps/worker/test/parseDoc.task.spec.ts

```
import { parseDoc } from '../src/parseDoc.task';
import fs from 'node:fs/promises';

jest.mock('node:fs/promises', () => ({
  __esModule: true,
  default: { readFile: jest.fn() },
  readFile: jest.fn()
}));

describe('parseDoc.task', () => {
  it('splits text into chunks and returns metadata', async () => {
    (fs.readFile as jest.Mock).mockResolvedValue('para1\n\npara2\n\npara3');
    const res = await parseDoc({ docPath: '/tmp/file.txt', orgId: 'org-1', documentId: '
      doc-1' });
    expect(res.chunks).toHaveLength(3);
    expect(res.documentId).toBe('doc-1');
  });
});
```

Embed Chunks Task (OpenAI mocked via nock).

Listing 58: apps/worker/test/embedChunks.task.spec.ts

```
import nock from 'nock';
import { embedChunksTask } from '../src/embedChunks.task';

describe('embedChunks.task', () => {
  beforeAll(() => {
    nock.disableNetConnect();
    nock('https://api.openai.com')
      .post('/v1/embeddings')
      .reply(200, {
        data: [{ embedding: Array(3072).fill(0.42), index: 0 }],
        model: 'text-embedding-3-large'
      });
  });
  afterAll(() => nock.enableNetConnect());

  it('embeds incoming chunks and persists to DB', async () => {
    const result = await embedChunksTask({
      chunks: [{ id: 'c1', text: 'hello world' }],
      orgId: 'org-1', documentId: 'doc-1'
    });
    expect(result.inserted).toBe(1);
  });
});
```

Worker Entrypoint (concurrency/startup) — smoke.

Listing 59: apps/worker/test/main.startup.spec.ts

```
jest.mock('../src/queue', () => ({
  startWorkers: jest.fn().mockResolvedValue({ started: true, concurrency: 8 })
}));

describe('worker main', () => {
  it('boots with configured concurrency', async () => {
    const { bootstrap } = await import('../src/main');
    const info = await bootstrap();
    expect(info.started).toBe(true);
    expect(info.concurrency).toBe(8);
  });
});
```

14.3.6 API–Worker Bridge (enqueue from API).

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

Listing 60: apps/api/test/bridge.enqueue.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';

jest.mock('../src/worker-bridge', () => ({
  enqueueReScore: jest.fn().mockResolvedValue({ queued: true })
}));

describe('API-Worker bridge', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    await app.init();
  });

  afterAll(async () => { await app.close(); });

  it('POST /rfps/:rfpId/score enqueues score job', async () => {
    const rfpId = '00000000-0000-0000-0000-000000000001';
    const res = await request(app.getHttpServer())
      .post(`/rfps/${rfpId}/score`)
      .set('Authorization', 'Bearer TEST_JWT')
      .expect(202);

    expect(res.body.queued).toBe(true);
    const { enqueueReScore } = require('../src/worker-bridge');
    expect(enqueueReScore).toHaveBeenCalledWith({ rfpId, orgId: expect.any(String) });
  });
});
```

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 ⇒ 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 → parse → embed → 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):** 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.

  - *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` or `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.
- **Queues:** Managed Redis (e.g., Upstash, ElastiCache, or Fly/Render Redis with auth & TLS).
  - *Config:* private network/VPC peering; requirepass; TLS if provider supports.
- **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)

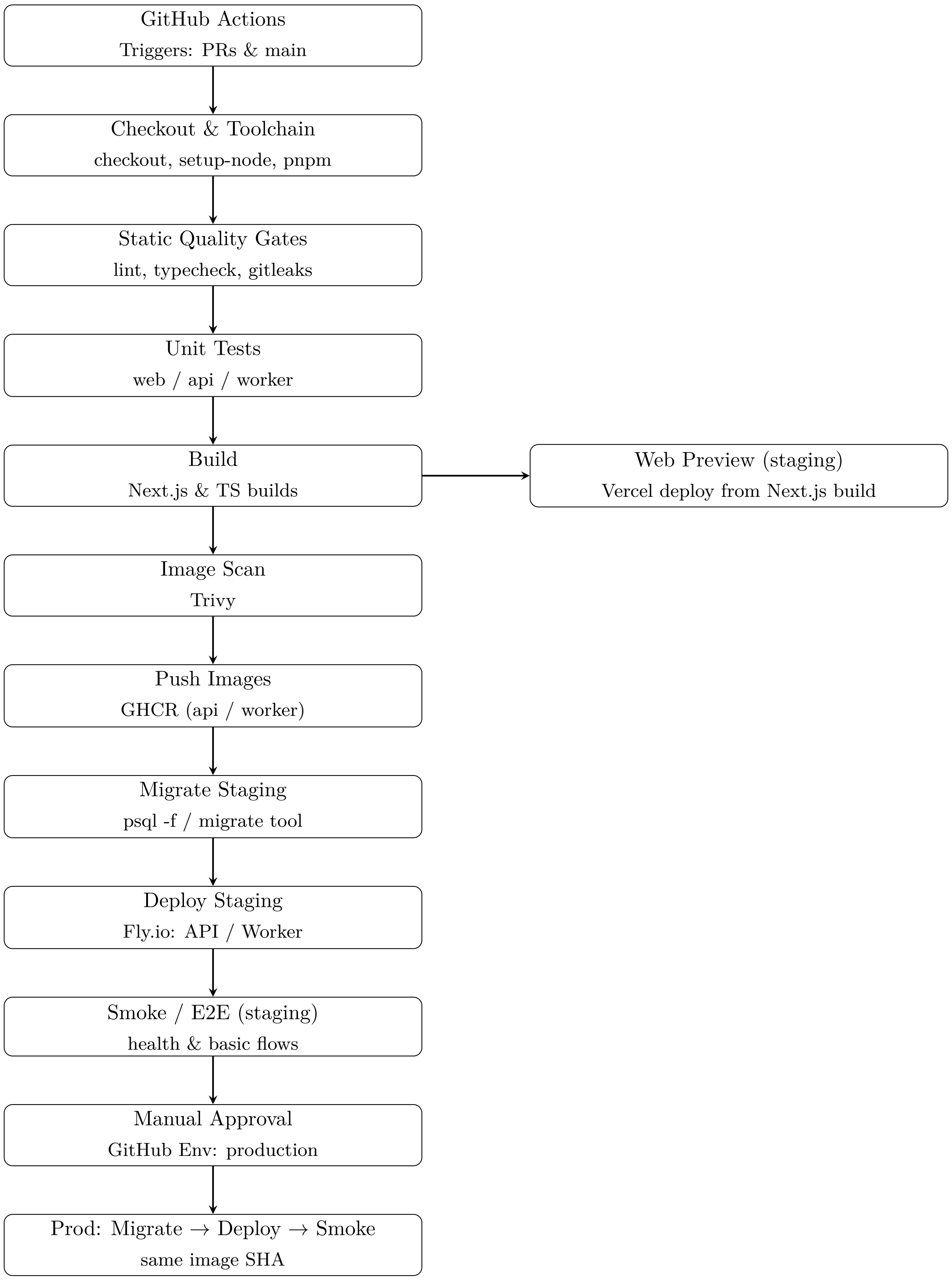
We standardize on **GitHub Actions** for CI/CD. The pipeline promotes the *same, immutable artifacts* from *staging* to *production* after manual approval. Web (Next.js) and backend (NestJS

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 your 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

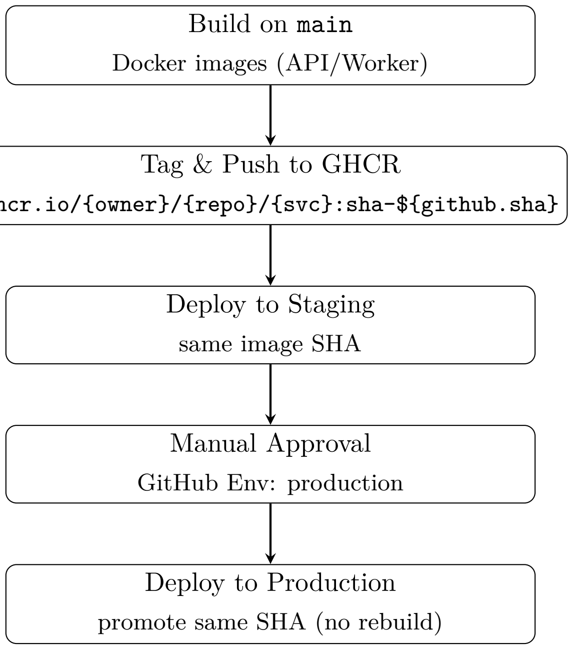
- **CI (on PR + main pushes):** Checkout/toolchain → Lint/Typecheck → Unit tests → Build → (optional) Image scan.
  Intuitively: Make sure the recipe is correct, ingredients are fresh, and the cake bakes.
- **CD (on main pushes):** Push images → **Staging** migrations → Deploy to staging (web+api/worker) → 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).

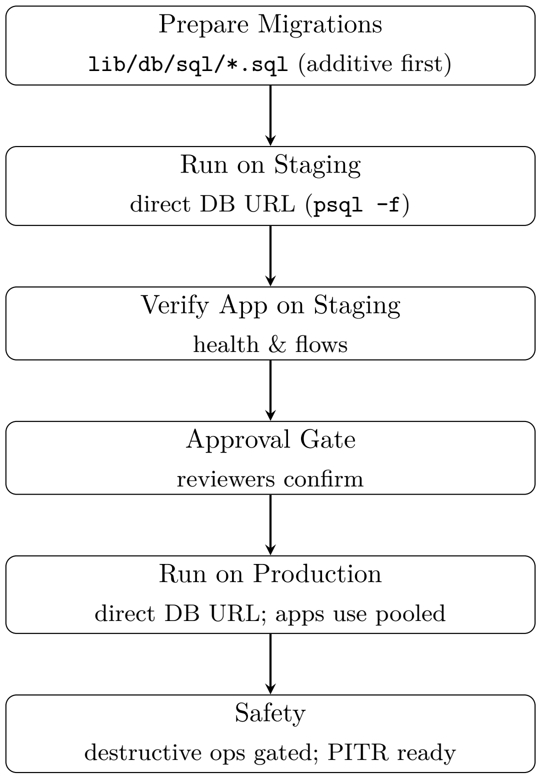
- Build Docker images once on `main` and push to `ghcr.io/{owner}/{repo}/{service}:sha-${github.sha}`.
- Deploy **the same image tags** to staging and, after approval, to production (no rebuilds).



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).

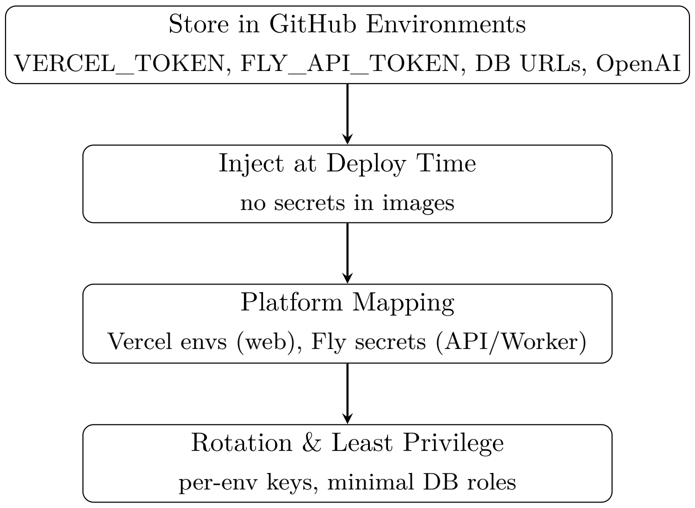
- **Order**: staging → approval → prod.
- **Connectivity**: use *direct* DB URL/role for migrations; apps use pooled URL.
- **Safety**: prefer backward-compatible migrations (additive first; destructive later) to avoid downtime.



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.

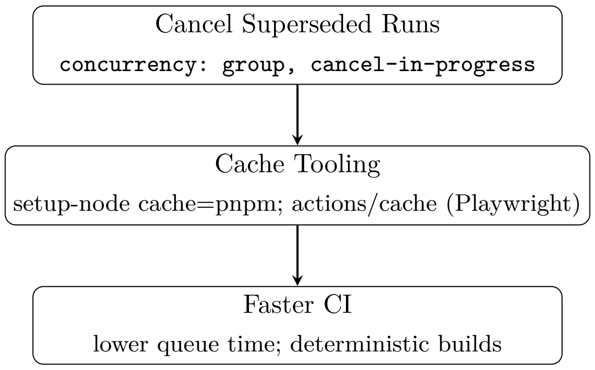
- Store CI/CD secrets in GitHub Environments (`VERCEL_TOKEN`, `FLY_API_TOKEN`, DB URLs, OpenAI keys).
- Never bake secrets into images; pass via platform secrets at deploy time (Vercel project envs; Fly secrets).


Store in GitHub Environments
VERCEL_TOKEN, FLY_API_TOKEN, DB URLs, OpenAI
Inject at Deploy Time
no secrets in images
Platform Mapping
Vercel envs (web), Fly secrets (API/Worker)
Rotation & Least Privilege
per-env keys, minimal DB roles

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.

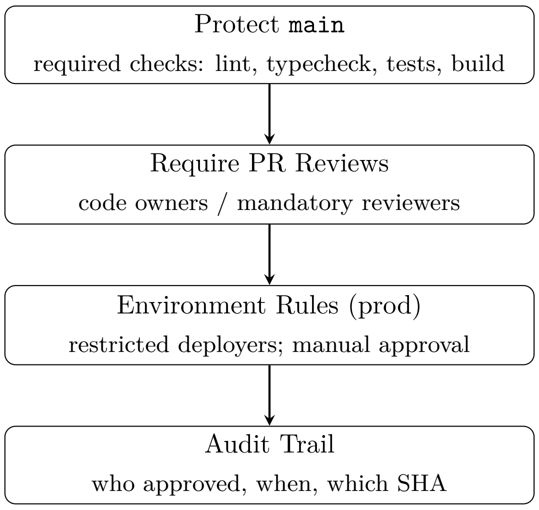
- Use `concurrency` groups to cancel superseded runs (e.g., multiple pushes to same branch).
- Enable `actions/cache` or Node cache in `setup-node` for `pnpm` and Playwright browsers (E2E).


Cancel Superseded Runs
concurrency: group, cancel-in-progress
Cache Tooling
setup-node cache=pnpm; actions/cache (Playwright)
Faster CI
lower queue time; deterministic builds

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.

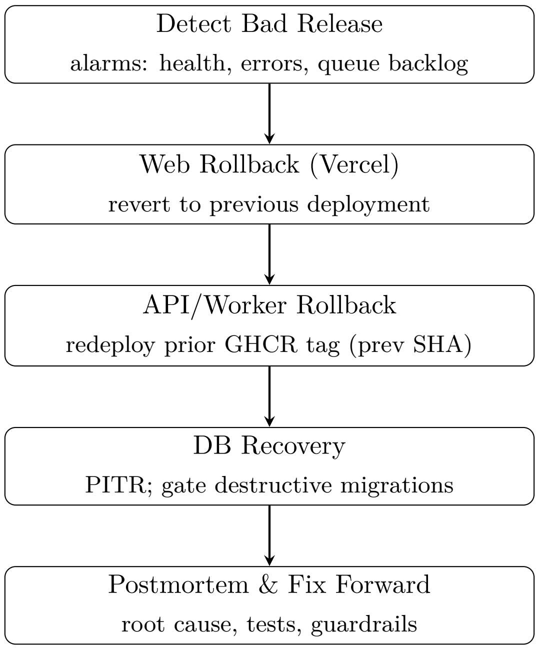
- Protect `main`: require status checks (lint, typecheck, tests, build) to pass before merge.
- Enforce PR reviews; restrict who can trigger production deployments via environment rules.


Protect main
required checks: lint, typecheck, tests, build
Require PR Reviews
code owners / mandatory reviewers
Environment Rules (prod)
restricted deployers; manual approval
Audit Trail
who approved, when, which SHA

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.

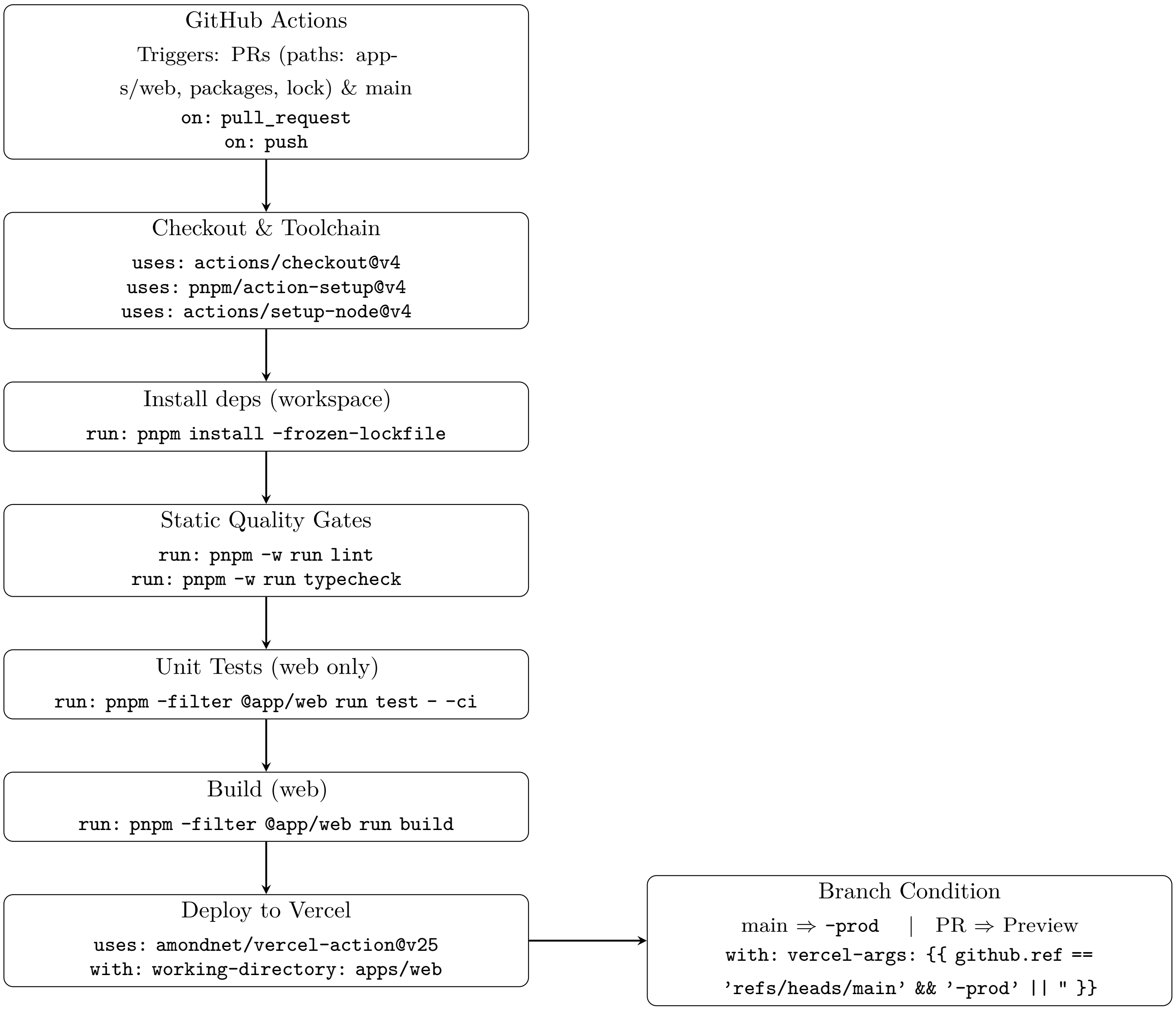

Detect Bad Release
alarms: health, errors, queue backlog
Web Rollback (Vercel)
revert to previous deployment
API/Worker Rollback
redeploy prior GHCR tag (prev SHA)
DB Recovery
PITR; gate destructive migrations
Postmortem & Fix Forward
root cause, tests, guardrails

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‘.



GitHub Actions
Triggers: PRs (paths: apps/web, packages, lock) & main
on: pull_request
on: push
Checkout & Toolchain
uses: actions/checkout@v4
uses: pnpm/action-setup@v4
uses: actions/setup-node@v4
Install deps (workspace)
run: pnpm install -frozen-lockfile
Static Quality Gates
run: pnpm -w run lint
run: pnpm -w run typecheck
Unit Tests (web only)
run: pnpm -filter @app/web run test - -ci
Build (web)
run: pnpm -filter @app/web run build
Deploy to Vercel
uses: amondnet/vercel-action@v25
with: working-directory: apps/web
Branch Condition
main ⇒ -prod | PR ⇒ Preview
with: vercel-args: {{ github.ref == 'refs/heads/main' && '-prod' || " }}

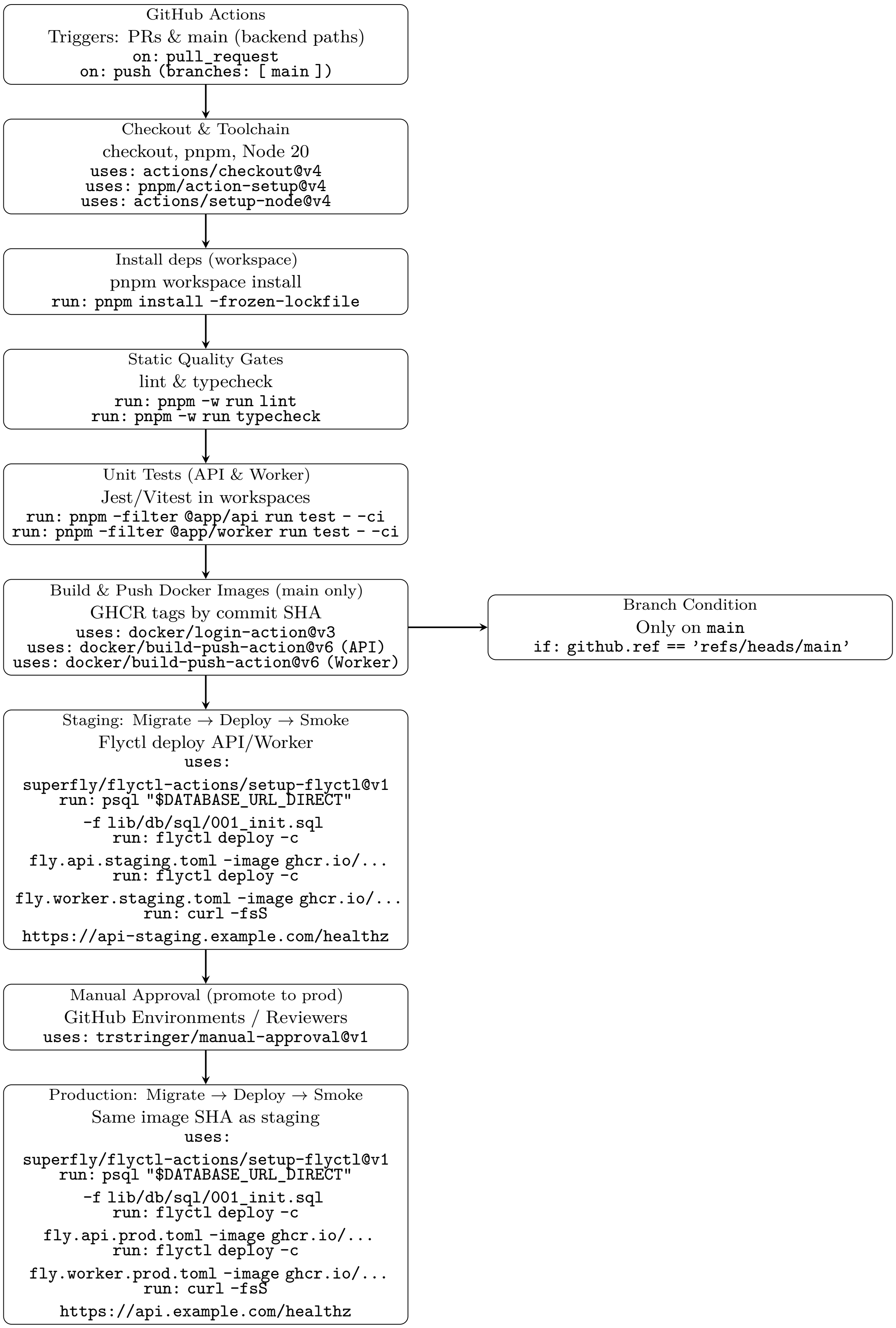
Listing 61: GitHub Actions (backend) — build once, stage, approve, prod

```
name: web (Next.js → Vercel)

on:
  pull_request:
```

```yaml
    paths:
      - "apps/web/**"
      - "package.json"
      - "pnpm-lock.yaml"
      - "packages/**"
  push:
    branches: [ main ]
    paths:
      - "apps/web/**"
      - "package.json"
      - "pnpm-lock.yaml"
      - "packages/**"

jobs:
  build_and_deploy:
    runs-on: ubuntu-latest
    env:
      NODE_ENV: production
    steps:
      - uses: actions/checkout@v4

      - uses: pnpm/action-setup@v4
        with: { version: 9 }

      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: "pnpm"

      - name: Install deps (workspace)
        run: pnpm install --frozen-lockfile

      - name: Lint & Typecheck
        run: |
          pnpm -w run lint
          pnpm -w run typecheck

      - name: Test (web only)
        run: pnpm --filter @app/web run test -- --ci

      - name: Build (web)
        run: pnpm --filter @app/web run build

      # Preview deploy on PR, Production on main
      - name: Deploy to Vercel
        uses: amondnet/vercel-action@v25
        with:
          vercel-token: ${{ secrets.VERCEL_TOKEN }}
          vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
          vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID_WEB }}
          working-directory: apps/web
          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.



```
on: pull_request
on: push (branches: [ main ])
```

```
uses: actions/checkout@v4
uses: pnpm/action-setup@v4
uses: actions/setup-node@v4
```

```
run: pnpm install -frozen-lockfile
```

```
run: pnpm -w run lint
run: pnpm -w run typecheck
```

```
run: pnpm -filter @app/api run test - -ci
run: pnpm -filter @app/worker run test - -ci
```

```
uses: docker/login-action@v3
uses: docker/build-push-action@v6 (API)
uses: docker/build-push-action@v6 (Worker)
```

```
if: github.ref == 'refs/heads/main'
```

```
uses:
superfly/flyctl-actions/setup-flyctl@v1
run: psql "$DATABASE_URL_DIRECT"
-f lib/db/sql/001_init.sql
run: flyctl deploy -c
fly.api.staging.toml -image ghcr.io/...
run: flyctl deploy -c
fly.worker.staging.toml -image ghcr.io/...
run: curl -fsS
https://api-staging.example.com/healthz
```

Manual Approval (promote to prod)
GitHub Environments / Reviewers

```
uses: trstringer/manual-approval@v1
```

Production: Migrate → Deploy → Smoke
Same image SHA as staging

```
uses:
superfly/flyctl-actions/setup-flyctl@v1
run: psql "$DATABASE_URL_DIRECT"
-f lib/db/sql/001_init.sql
run: flyctl deploy -c
fly.api.prod.toml -image ghcr.io/...
run: flyctl deploy -c
fly.worker.prod.toml -image ghcr.io/...
run: curl -fsS
https://api.example.com/healthz
```

Listing 62: GitHub Actions (backend) — build once, stage, approve, prod

```
name: backend (NestJS → Fly)

on:
  pull_request:
    paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-
    lock.yaml"]
  push:
    branches: [ main ]
    paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-
    lock.yaml"]

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: pnpm/action-setup@v4
        with: { version: 9 }
      - uses: actions/setup-node@v4
        with: { node-version: 20, cache: pnpm }
      - run: pnpm install --frozen-lockfile
      - run: pnpm -w run lint
      - run: pnpm -w run typecheck
      - run: pnpm --filter @app/api run test -- --ci
      - run: pnpm --filter @app/worker run test -- --ci

  build_and_push_images:
    needs: ci
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v6
        with:
          context: .
          file: Dockerfile.api
          push: true
          tags: ghcr.io/${{ github.repository }}/api:sha-${{ github.sha }}
```

```yaml
      - uses: docker/build-push-action@v6
        with:
          context: .
          file: Dockerfile.worker
          push: true
          tags: ghcr.io/${{ github.repository }}/worker:sha-${{ github.sha }}

  stage_migrate_and_deploy:
    needs: build_and_push_images
    runs-on: ubuntu-latest
    environment: staging
    steps:
      - uses: actions/checkout@v4
      - uses: superfly/flyctl-actions/setup-flyctl@v1
      - name: DB migrate (staging)
        env: { DATABASE_URL_DIRECT: ${{ secrets.DATABASE_URL_DIRECT_STAGING }} }
        run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
      - name: Deploy API (staging)
        run: flyctl deploy -c fly.api.staging.toml --image ghcr.io/${{ github.repository }}/api:sha-${{ github.sha }} --detach
      - name: Deploy Worker (staging)
        run: flyctl deploy -c fly.worker.staging.toml --image ghcr.io/${{ github.repository }}/worker:sha-${{ github.sha }} --detach
      - name: Smoke
        run: curl -fsS https://api-staging.example.com/healthz

  approve_and_promote_prod:
    if: github.ref == 'refs/heads/main'
    needs: stage_migrate_and_deploy
    runs-on: ubuntu-latest
    environment:
      name: production
    steps:
      - name: Await manual approval
        uses: trstringer/manual-approval@v1
        with:
          repo-token: ${{ secrets.GITHUB_TOKEN }}
          approvers: ${{ secrets.RELEASE_APPROVERS }}
          minimum-approvals: 1
          issue-title: "Promote backend to production"
          issue-body: "Approve SHA ${{ github.sha }}"

  prod_migrate_and_deploy:
    needs: approve_and_promote_prod
    runs-on: ubuntu-latest
    environment: production
    steps:
      - uses: actions/checkout@v4
      - uses: superfly/flyctl-actions/setup-flyctl@v1
      - name: DB migrate (prod)
        env: { DATABASE_URL_DIRECT: ${{ secrets.DATABASE_URL_DIRECT_PROD }} }
        run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
      - name: Deploy API (prod)
        run: flyctl deploy -c fly.api.prod.toml --image ghcr.io/${{ github.repository }}/
    api:sha-${{ github.sha }} --detach
      - name: Deploy Worker (prod)
        run: flyctl deploy -c fly.worker.prod.toml --image ghcr.io/${{ github.repository
    }}/worker:sha-${{ github.sha }} --detach
      - name: Smoke
        run: curl -fsS https://api.example.com/healthz
```

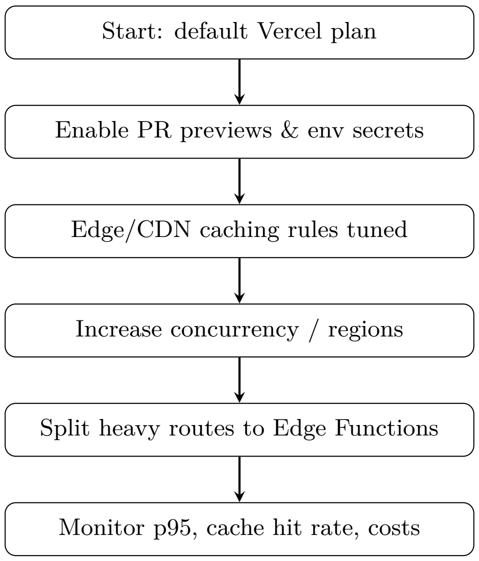
15.1.5.12 Rationale.

- **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)

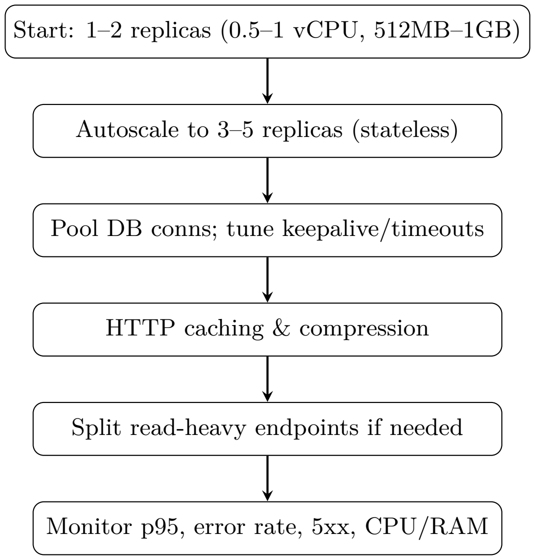
15.1.6.1 Frontend (Next.js on Vercel)

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


Start: default Vercel plan
Enable PR previews & env secrets
Edge/CDN caching rules tuned
Increase concurrency / regions
Split heavy routes to Edge Functions
Monitor p95, cache hit rate, costs

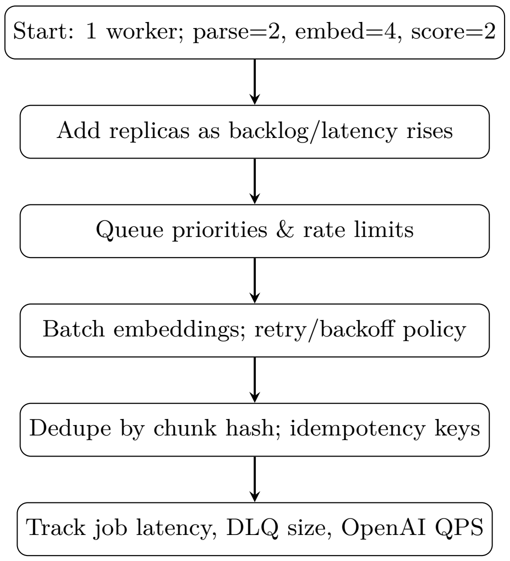
15.1.6.2 API (NestJS on Fly/Render)

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


Start: 1–2 replicas (0.5–1 vCPU, 512MB–1GB)
Autoscale to 3–5 replicas (stateless)
Pool DB conns; tune keepalive/timeouts
HTTP caching & compression
Split read-heavy endpoints if needed
Monitor p95, error rate, 5xx, CPU/RAM

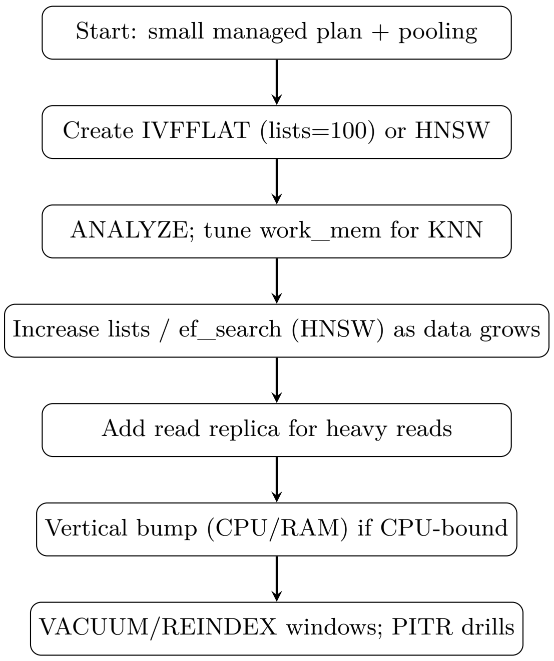
15.1.6.3 Workers (BullMQ on Fly/Render)

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


Start: 1 worker; parse=2, embed=4, score=2
Add replicas as backlog/latency rises
Queue priorities & rate limits
Batch embeddings; retry/backoff policy
Dedupe by chunk hash; idempotency keys
Track job latency, DLQ size, OpenAI QPS

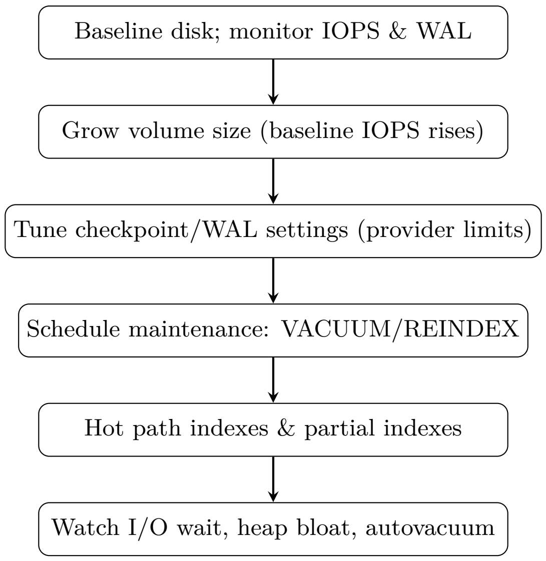
15.1.6.4 DB (PostgreSQL — Supabase Postgres + `pgvector`)

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.


Start: small managed plan + pooling
Create IVFFLAT (lists=100) or HNSW
ANALYZE; tune work_mem for KNN
Increase lists / ef_search (HNSW) as data grows
Add read replica for heavy reads
Vertical bump (CPU/RAM) if CPU-bound
VACUUM/REINDEX windows; PITR drills

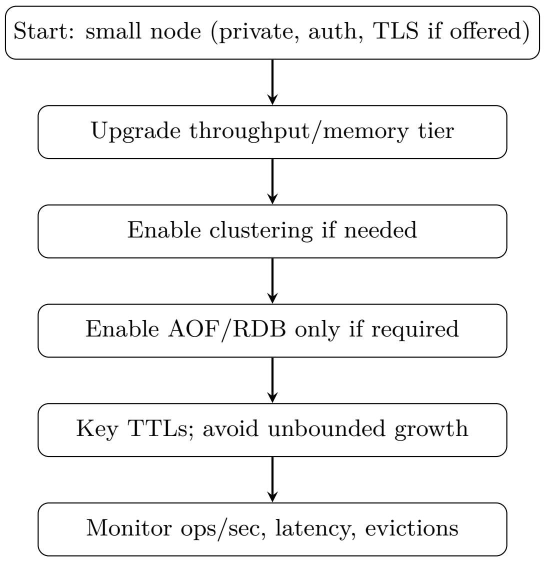
15.1.6.5 Postgres storage/IOPS:

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.


Baseline disk; monitor IOPS & WAL
Grow volume size (baseline IOPS rises)
Tune checkpoint/WAL settings (provider limits)
Schedule maintenance: VACUUM/REINDEX
Hot path indexes & partial indexes
Watch I/O wait, heap bloat, autovacuum

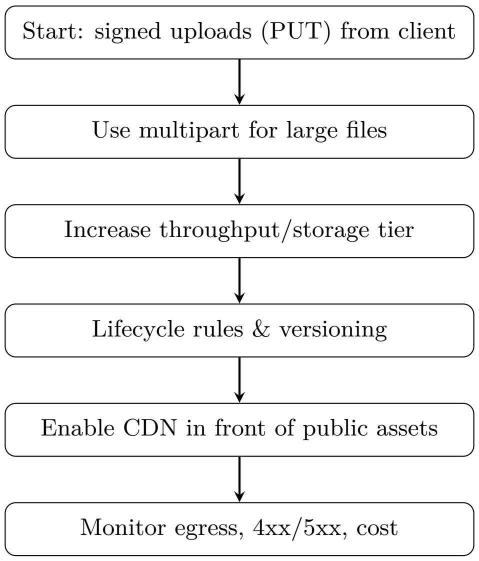
15.1.6.6 Redis (Managed — Upstash/Fly/Render)

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


Start: small node (private, auth, TLS if offered)
Upgrade throughput/memory tier
Enable clustering if needed
Enable AOF/RDB only if required
Key TTLs; avoid unbounded growth
Monitor ops/sec, latency, evictions

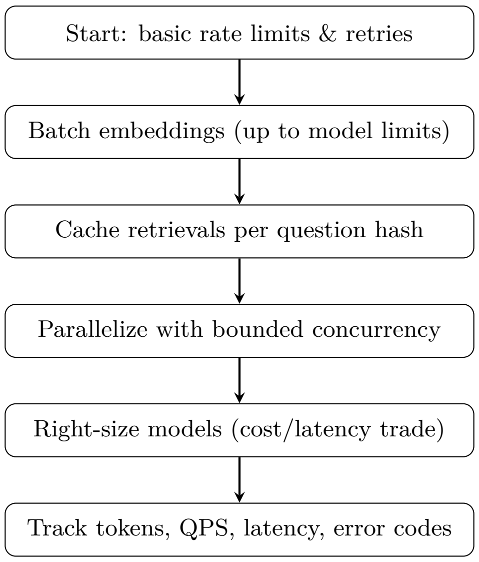
15.1.6.7 Object Storage (Supabase Storage, S3-compatible)

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


Start: signed uploads (PUT) from client
Use multipart for large files
Increase throughput/storage tier
Lifecycle rules & versioning
Enable CDN in front of public assets
Monitor egress, 4xx/5xx, cost

15.1.6.8 LLM (OpenAI API)

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


Start: basic rate limits & retries
Batch embeddings (up to model limits)
Cache retrievals per question hash
Parallelize with bounded concurrency
Right-size models (cost/latency trade)
Track tokens, QPS, latency, error codes

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.

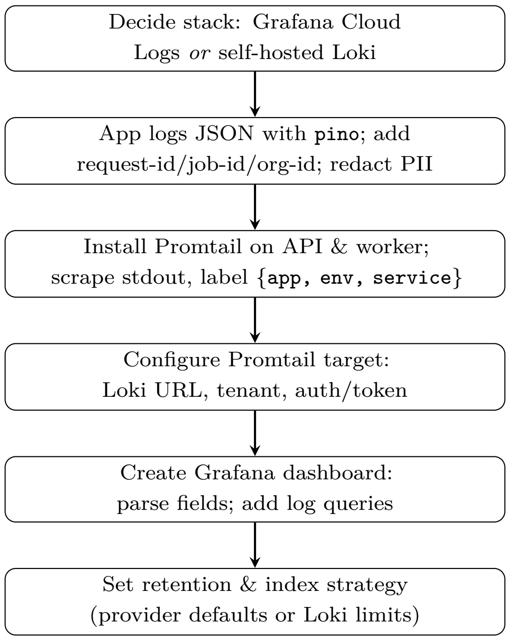
- **Tracing**
  - *Cloud-native:* Grafana Tempo Cloud, Sentry Performance, Datadog APM.
  - *Self-hosted:* OpenTelemetry SDK (Node) → OTel Collector → Tempo (+ Grafana). Propagate `traceparent` across API ↔ worker.

- **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`


Decide stack: Grafana Cloud Logs *or* self-hosted Loki
App logs JSON with `pino`; add request-id/job-id/org-id; redact PII
Install Promtail on API & worker; scrape stdout, label {`app, env, service`}
Configure Promtail target: Loki URL, tenant, auth/token
Create Grafana dashboard: parse fields; add log queries
Set retention & index strategy (provider defaults or Loki limits)

Listing 63: docker-compose.yml (Loki + Promtail + Grafana)

```
version: "3.8"

services:
  loki:
    image: grafana/loki:2.9.4
    command: [ "-config.file=/etc/loki/config.yml" ]
    ports: [ "3100:3100" ]
    volumes:
      - ./loki:/etc/loki
      - loki-data:/loki
    restart: unless-stopped

  promtail:
    image: grafana/promtail:2.9.4
    command: [ "-config.file=/etc/promtail/config.yml" ]
    volumes:
      - ./promtail:/etc/promtail
      - /var/log:/var/log:ro
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on: [ loki ]
    restart: unless-stopped

  grafana:
    image: grafana/grafana:10.4.5
    ports: [ "3000:3000" ]
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-data:/var/lib/grafana
    depends_on: [ loki ]
    restart: unless-stopped

volumes:
  loki-data:
  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 64: Loki config (boltdb-shipper, local FS, 7d retention)

```
auth_enabled: false  # put behind reverse-proxy if exposed

server:
  http_listen_port: 3100

common:
  instance_addr: 127.0.0.1
  path_prefix: /loki
  storage:
    filesystem:
      chunks_directory: /loki/chunks
      rules_directory: /loki/rules
  replication_factor: 1
  ring:
    kvstore:
      store: inmemory

schema_config:
  configs:
    - from: 2024-01-01
      store: boltdb-shipper
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h

compactor:
  working_directory: /loki/compactor
  compactor_ring:
    kvstore:
      store: inmemory
  retention_enabled: true

limits_config:
  reject_old_samples: true
  reject_old_samples_max_age: 168h  # 7 days
  retention_period: 168h

ruler:
  alertmanager_url: http://localhost:9093

analytics:
  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 65: Promtail config (Docker + system logs, safe labels)

```yaml
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push
    # basic_auth:
    #   username: loki
    #   password: secret

scrape_configs:
  # --- Docker containers (JSON) ---
  - job_name: docker
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
        refresh_interval: 5s
    pipeline_stages:
      - json:
          expressions:
            log: log
            stream: stream
            time: time
          timestamp:
            source: time
            format: RFC3339Nano
            fallback_formats: [ RFC3339 ]
      - labels:
          image: __meta_docker_image
      - output:
          source: log
    relabel_configs:
      - source_labels: [__meta_docker_container_name]
        target_label: container
        regex: "/(.*)"
      - source_labels: [__meta_docker_container_log_stream]
        target_label: stream
      - source_labels: [__meta_docker_container_label_com_docker_compose_service]
        target_label: service
      - target_label: app
        replacement: api  # change per host (api/worker/web)

  # --- System logs example ---
  - job_name: syslog
    static_configs:
      - targets: [ localhost ]
        labels:
          job: syslog
          __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 66: pino-http setup with redaction & request-id

```
import pinoHttp from 'pino-http';
import { randomUUID } from 'crypto';

export const httpLogger = pinoHttp({
  redact: { paths: ['req.headers.authorization', 'user.password'], censor: '[REDACTED]'
    },
  genReqId: (req) => (req.headers['x-request-id'] as string) || randomUUID(),
  customProps: (req) => ({
    service: 'api',
    env: process.env.NODE_ENV || 'dev',
    orgId: (req as any).orgId ?? 'unknown',
  }),
  transport: process.env.NODE_ENV === 'development'
    ? { target: 'pino-pretty', options: { colorize: true } }
    : undefined
});
```

Listing 67: Express wiring (derive orgId; emit health)

```
// server.ts
import express from 'express';
import { httpLogger } from './logger';

const app = express();

app.use((req, _res, next) => {
  // In real app, derive from JWT (AuthGuard) and set req.user/orgId.
  (req as any).orgId = req.header('x-org-id') || 'unknown';
  next();
});

app.use(httpLogger);

app.get('/health', (_req, res) => res.json({ ok: true }));

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 68: Recent logs for API service

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

Listing 69: Filter by orgId and requestId

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

Listing 70: Error count by container over 5m

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

Listing 71: Latency histograms if we log durationMs

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

Security & multi-tenancy notes.

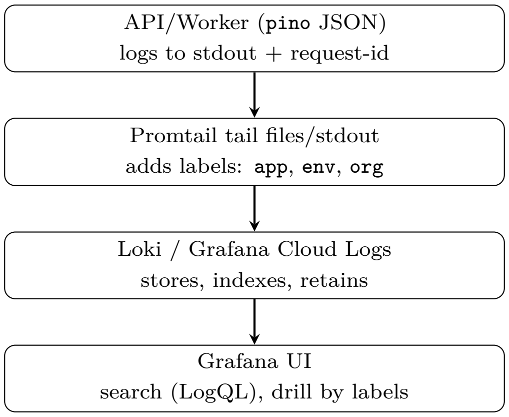
- Put Loki behind TLS and a reverse proxy (nginx/traefik). If multi-tenant, enable `auth_enabled: true` and use `X-Scope-OrgID`.
- Avoid high-cardinality labels (e.g., raw user IDs). Parse as fields with `| json` and filter at query time instead.

Quick push (without Promtail).

Listing 72: POST a test log to Loki directly

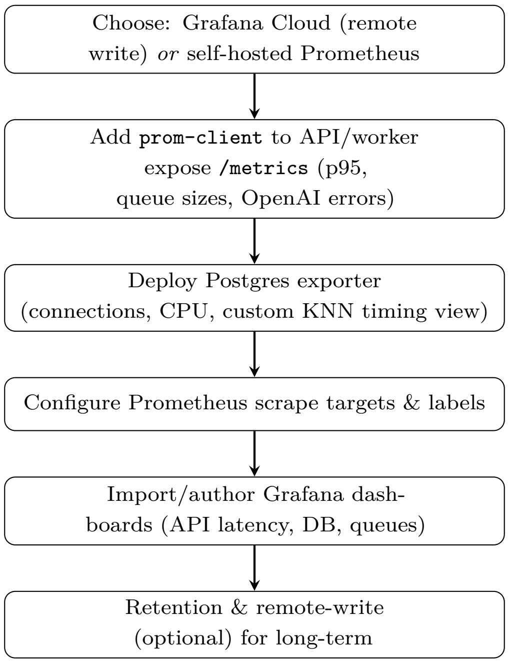
```
curl -X POST "http://localhost:3100/loki/api/v1/push" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "streams": [{
      "stream": { "app":"api", "env":"dev" },
      "values": [[ "'$(date +%s%N)'", "{\"level\":\"info\",\"msg\":\"hello from curl\"}" ]]
    }]
  }'
```

Logging — runtime data flow.


API/Worker (pino JSON)
logs to stdout + request-id
Promtail tail files/stdout
adds labels: app, env, org
Loki / Grafana Cloud Logs
stores, indexes, retains
Grafana UI
search (LogQL), drill by labels

15.1.7.3 Metrics (Prometheus)

Setup Workflow


Choose: Grafana Cloud (remote write) or self-hosted Prometheus
Add prom-client to API/worker expose /metrics (p95, queue sizes, OpenAI errors)
Deploy Postgres exporter (connections, CPU, custom KNN timing view)
Configure Prometheus scrape targets & labels
Import/author Grafana dashboards (API latency, DB, queues)
Retention & remote-write (optional) for long-term

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 73: docker-compose.yml

```
version: "3.9"

services:
  prometheus:
    image: prom/prometheus:v2.53.0
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --web.enable-lifecycle
    ports: ["9090:9090"]
    volumes:
      - ./prometheus:/etc/prometheus
      - prom-data:/prometheus
    restart: unless-stopped

  grafana:
    image: grafana/grafana:10.4.6
    ports: ["3000:3000"]
    environment:
      GF_SECURITY_ADMIN_USER: admin
      GF_SECURITY_ADMIN_PASSWORD: admin
    volumes:
      - graf-data:/var/lib/grafana
    restart: unless-stopped
    depends_on: [prometheus]

  # PostgreSQL exporter (targets Supabase/Postgres)
  postgres_exporter:
    image: prometheuscommunity/postgres-exporter:v0.15.0
    environment:
      DATA_SOURCE_NAME: ${PG_EXPORTER_DSN}
      # PG_EXPORTER_EXTEND_QUERY_PATH: /etc/queries/queries.yaml
    volumes:
      - ./postgres_exporter:/etc/queries:ro
    restart: unless-stopped

  # Optional: cAdvisor (container metrics on a single host)
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.47.2
    ports: ["8080:8080"]
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    restart: unless-stopped

  # Optional: Node exporter for host metrics
  node_exporter:
    image: prom/node-exporter:v1.8.2
    pid: host
    network_mode: host
    restart: unless-stopped

volumes:
  prom-data:
  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 74: prometheus/prometheus.yml

```
global:
  scrape_interval: 15s
  evaluation_interval: 30s
  external_labels:
    cluster: "prod"

rule_files:
  - rules.yml

scrape_configs:
  # --- API (NestJS) ---
  - job_name: "api"
    metrics_path: /metrics
    static_configs:
      - targets: ["api.internal:3001"]   # private addr or public URL
        labels:
          service: api
          env: prod

  # --- Worker (BullMQ) ---
  - job_name: "worker"
    metrics_path: /metrics
    static_configs:
      - targets: ["worker.internal:3002"]
        labels:
          service: worker
          env: prod

  # --- Postgres exporter ---
  - job_name: "postgres"
    static_configs:
      - targets: ["postgres_exporter:9187"]
        labels:
          service: postgres
          env: prod

  # --- cAdvisor (optional) ---
  - job_name: "cadvisor"
    static_configs:
      - targets: ["cadvisor:8080"]
        labels:
          service: cadvisor
          env: prod

  # --- Node exporter (optional) ---
  - job_name: "node"
    static_configs:
      - targets: ["localhost:9100"]
        labels:
          service: node
          env: prod
```

Listing 75: prometheus/rules.yml — recording & alert rules

```
groups:
- name: recording
  interval: 30s
  rules:
    # API p95 latency from Histogram over 5m
    - record: job:http_p95_ms:5m
      expr: |
        histogram_quantile(
          0.95,
          sum by (le) (rate(http_request_duration_ms_bucket{service="api"}[5m]))
        ) * 1.0

    # Worker queue size (example Gauge)
    - record: job:queue_waiting:sum
      expr: sum(worker_queue_waiting)

- name: alerts
  rules:
    - alert: ApiHighErrorRate
      expr: sum(rate(http_requests_total{service="api",code=~"5.."}[5m])) /
            sum(rate(http_requests_total{service="api"}[5m])) > 0.05
      for: 10m
      labels: { severity: page }
      annotations:
        summary: "API 5xx error rate > 5%"

    - alert: PostgresConnectionsHigh
      expr: pg_stat_activity_count{datname=~".+"} > 0.8 * pg_settings_max_connections
      for: 10m
      labels: { severity: warn }
      annotations:
        summary: "Postgres nearing max connections"
```

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

Listing 76: apps/api/src/metrics.ts

```
import client from 'prom-client';
import type { Request, Response } from 'express';

client.collectDefaultMetrics({ prefix: 'api_' });

export const httpReqCounter = new client.Counter({
  name: 'http_requests_total',
  help: 'HTTP request count',
  labelNames: ['method','route','code'],
});

export const httpDuration = new client.Histogram({
  name: 'http_request_duration_ms',
  help: 'HTTP request duration (ms)',
  buckets: [5,10,25,50,100,250,500,1000,2500,5000],
  labelNames: ['method','route','code'],
});

export const openaiErrors = new client.Counter({
  name: 'openai_api_errors_total',
  help: 'OpenAI API error count',
  labelNames: ['operation','status'],
});

export function metricsHandler(_req: Request, res: Response) {
  res.set('Content-Type', client.register.contentType);
  res.end(client.register.metrics());
}

// Express/Nest adapter example (Express)
export function promMiddleware(routeKey: (req: Request)=>string) {
  return async (req: Request, res: Response, next: Function) => {
    const end = httpDuration.startTimer({ method: req.method, route: routeKey(req) });
    res.on('finish', () => {
      httpReqCounter.inc({ method: req.method, route: routeKey(req), code: String(res.statusCode) });
      end({ code: String(res.statusCode) });
    });
    next();
  };
}
```

Listing 77: apps/api/src/server.ts — wiring /metrics

```
import express from 'express';
import { metricsHandler, promMiddleware } from './metrics';
import { openaiErrors } from './metrics';

const app = express();

// Minimal route keyer (avoid exploding cardinality)
const routeKey = (req: any) => (req.route?.path || req.path || 'unknown');

// Apply before your routes
app.use(promMiddleware(routeKey));

app.get('/metrics', metricsHandler);

// Example: count OpenAI failures
async function callOpenAI() {
  try {
    // ... OpenAI call ...
  } catch (e: any) {
    openaiErrors.inc({ operation: 'embeddings', status: String(e?.status || 500) });
    throw e;
  }
}

app.listen(3001, () => console.log('API on :3001'));
```

Worker metrics (BullMQ queue sizes).

Listing 78: apps/worker/src/metrics.ts

```
import client from 'prom-client';
import { Queue } from 'bullmq';

```

```
client.collectDefaultMetrics({ prefix: 'worker_' });

export const queueWaiting = new client.Gauge({
  name: 'worker_queue_waiting',
  help: 'Number of waiting jobs',
  labelNames: ['queue'],
});
export const queueActive = new client.Gauge({
  name: 'worker_queue_active',
  help: 'Number of active jobs',
  labelNames: ['queue'],
});

export async function pollQueues(queues: Record<string, Queue>) {
  for (const [name, q] of Object.entries(queues)) {
    const [w, a] = await Promise.all([q.getWaitingCount(), q.getActiveCount()]);
    queueWaiting.set({ queue: name }, w);
    queueActive.set({ queue: name }, a);
  }
}
```

Listing 79: apps/worker/src/main.ts — expose /metrics

```
import http from 'http';
import client from 'prom-client';
import { queues } from './queue';
import { pollQueues } from './metrics';

// Simple /metrics server
const srv = http.createServer(async (req, res) => {
  if (req.url === '/metrics') {
    await pollQueues({ parse: queues.parseDoc.q, embed: queues.embedChunks.q, score:
      queues.scoreMatches.q });
    res.writeHead(200, { 'Content-Type': client.register.contentType });
    res.end(await client.register.metrics());
  } else {
    res.writeHead(404); res.end();
  }
});
srv.listen(3002);
```

Postgres exporter — optional custom query (pgvector visibility).

Listing 80: postgres_exporter/queries.yaml

```
pgvector_knn:
  query: |
    SELECT
      sum(total_time) AS total_ms,
      sum(calls) AS calls
    FROM pg_stat_statements
    WHERE query ILIKE '%<=>%' OR query ILIKE '%vector%';
  metrics:
    - total_ms:
        usage: "GAUGE"
        description: "Total time spent in vector/knn-like queries (ms)"
    - calls:
        usage: "GAUGE"
        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.

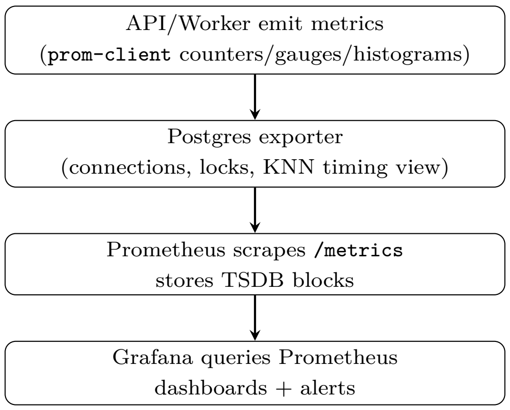
Listing 81: Grafana quick start

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

Security & production.

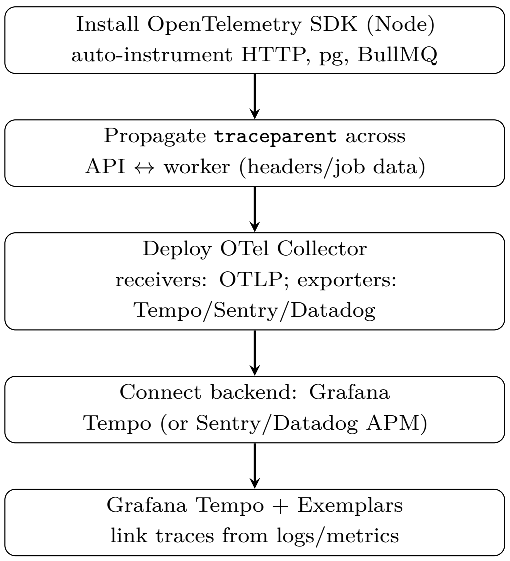
- Place Prometheus behind a private network/VPN or a reverse proxy with auth (and HTTPS).
- Use *read-only* DB credentials for the Postgres exporter.
- Keep label cardinality low (route templates, not raw paths/IDs).
- For long-term retention, enable `remote_write` (Grafana Cloud, Thanos, or VictoriaMetrics).

Metrics — runtime data flow.


API/Worker emit metrics
(prom-client counters/gauges/histograms)
Postgres exporter
(connections, locks, KNN timing view)
Prometheus scrapes /metrics
stores TSDB blocks
Grafana queries Prometheus
dashboards + alerts

15.1.7.4 Tracing (OpenTelemetry)

Setup Workflow


Install OpenTelemetry SDK (Node) auto-instrument HTTP, pg, BullMQ
Propagate traceparent across API ↔ worker (headers/job data)
Deploy OTel Collector receivers: OTLP; exporters: Tempo/Sentry/Datadog
Connect backend: Grafana Tempo (or Sentry/Datadog APM)
Grafana Tempo + Exemplars link traces from logs/metrics

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

Listing 82: docker-compose.yml

```yaml
version: "3.8"

services:
  tempo:
    image: grafana/tempo:2.5.0
    command: [ "-config.file=/etc/tempo.yaml" ]
    volumes:
      - ./tempo/tempo.yaml:/etc/tempo.yaml:ro
      - tempo-data:/var/tempo
    ports: [ "3200:3200" ]  # Tempo HTTP API for Grafana
    restart: unless-stopped

  otel-collector:
    image: otel/opentelemetry-collector:0.101.0
    command: [ "--config=/etc/otelcol.yaml" ]
    volumes:
      - ./otel/otelcol.yaml:/etc/otelcol.yaml:ro
    ports:
      - "4317:4317"  # OTLP gRPC
      - "4318:4318"  # OTLP HTTP
      - "8888:8888"  # Collector metrics/debug
    depends_on: [ tempo ]
    restart: unless-stopped

  grafana:
    image: grafana/grafana:10.4.5
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin   # change in prod
    ports: [ "3000:3000" ]
    volumes:
      - grafana-data:/var/lib/grafana
    depends_on: [ tempo ]
    restart: unless-stopped

volumes:
  tempo-data:
  grafana-data:
```

Tempo config — `./tempo/tempo.yaml`

Listing 83: ./tempo/tempo.yam

```yaml
server:
  http_listen_port: 3200

distributor:
  receivers:
    otlp:
      protocols:
        http:
        grpc:

compactor:
  compaction:
    block_retention: 168h  # 7 days

storage:
  trace:
    backend: local
    local:
      path: /var/tempo
```

Collector config — `./otel/otelcol.yaml`

Listing 84: ./otel/otelcol.yaml

```yaml
receivers:
  otlp:
    protocols:
      http:
      grpc:

processors:
  memory_limiter:
    check_interval: 1s
    limit_percentage: 75
    spike_limit_percentage: 15
  batch:
    send_batch_size: 1024
    timeout: 5s
  resource:
    attributes:
      - key: service.environment
        action: upsert
        value: dev  # set per environment

exporters:
  otlphttp/tempo:
    endpoint: http://tempo:4318
    tls:
      insecure: true

extensions:
  health_check: {}

service:
  extensions: [health_check]
  pipelines:
    traces:
      receivers: [otlp]
      processors: [memory_limiter, batch, resource]
      exporters: [otlphttp/tempo]
```

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

```
pnpm add @opentelemetry/sdk-node \
         @opentelemetry/auto-instrumentations-node \
         @opentelemetry/exporter-trace-otlp-http \
         @opentelemetry/propagator-w3c \
         @opentelemetry/instrumentation-express \
         @opentelemetry/instrumentation-http \
         @opentelemetry/instrumentation-pg
```

Listing 85: apps/api/src/otel.ts

```
import { NodeSDK } from '@opentelemetry/sdk-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
import { W3CTraceContextPropagator } from '@opentelemetry/core';
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { Resource } from '@opentelemetry/resources';

const exporter = new OTLPTraceExporter({
  url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
});

const sdk = new NodeSDK({
  traceExporter: exporter,
  textMapPropagator: new W3CTraceContextPropagator(),
  instrumentations: getNodeAutoInstrumentations({
    '@opentelemetry/instrumentation-http': { enabled: true },
    '@opentelemetry/instrumentation-express': { enabled: true },
    '@opentelemetry/instrumentation-pg': { enabled: true },
  }),
  resource: new Resource({
    'service.name': 'api',
    'service.namespace': 'bidion',
  }),
});

sdk.start();
console.log('[OTel] API tracing enabled');
process.on('SIGTERM', () => sdk.shutdown());
```

Listing 86: API env

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

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

Listing 87: apps/api/src/queues.ts (inject traceparent)

```
import { context, propagation } from '@opentelemetry/api';
import { Queue } from 'bullmq';

export async function enqueueScore(rfpId: string, orgId: string, q: Queue) {
  const carrier: Record<string, string> = {};
  propagation.inject(context.active(), carrier); // writes traceparent
  await q.add('score', { rfpId, orgId, otel: carrier }, { attempts: 5 });
}
```

Listing 88: apps/worker/src/otel.ts

```
import { NodeSDK } from '@opentelemetry/sdk-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
import { W3CTraceContextPropagator } from '@opentelemetry/core';
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { Resource } from '@opentelemetry/resources';

const exporter = new OTLPTraceExporter({
  url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
});

export const workerSdk = new NodeSDK({
  traceExporter: exporter,
  textMapPropagator: new W3CTraceContextPropagator(),
  instrumentations: getNodeAutoInstrumentations({}),
  resource: new Resource({
    'service.name': 'worker',
    'service.namespace': 'bidion',
  }),
});
workerSdk.start();
```

Listing 89: apps/worker/src/score.processor.ts (extract + span)

```
import { context, propagation, trace } from '@opentelemetry/api';

export async function handleScore(job: any) {
  const carrier = (job.data && job.data.otel) || {};
  const parentCtx = propagation.extract(context.active(), carrier);

  return await context.with(parentCtx, async () => {
    const tracer = trace.getTracer('worker');
    return await tracer.startActiveSpan('score-matches', async (span) => {
      try {
        span.setAttribute('rfp.id', job.data.rfpId);
        span.setAttribute('org.id', job.data.orgId);
        // ... KNN, keyword/rules, write matches
      } catch (e) {
        span.recordException(e as Error);
        span.setStatus({ code: 2, message: 'error' });
        throw e;
      } finally {
        span.end();
      }
    });
  });
}
```

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`}.

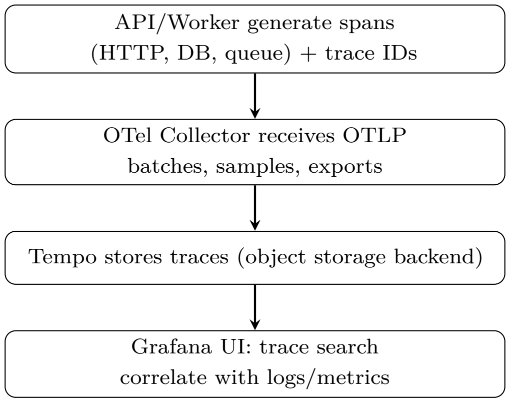
Log/Metric correlation (optional)

- 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

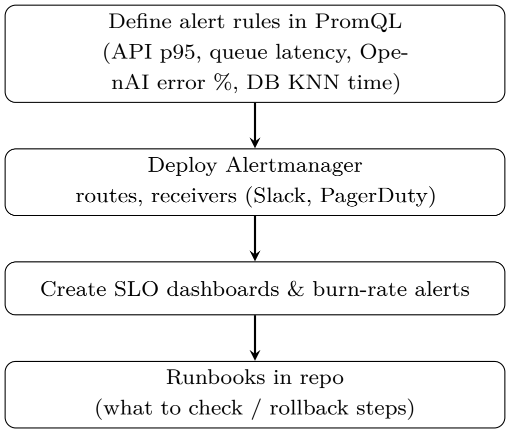
- Put Grafana/Tempo behind TLS and auth (reverse proxy or Grafana OAuth).
- Limit Collector ports (4317/4318) to private networks; do not expose publicly.
- Add `org.id` as a span/resource attribute for per-tenant filtering.

Tracing — runtime data flow.


API/Worker generate spans
(HTTP, DB, queue) + trace IDs
OTel Collector receives OTLP
batches, samples, exports
Tempo stores traces (object storage backend)
Grafana UI: trace search
correlate with logs/metrics

15.1.7.5 Alerting / On-call

Setup Workflow.


Define alert rules in PromQL
(API p95, queue latency, OpenAI error %, DB KNN time)
Deploy Alertmanager
routes, receivers (Slack, PagerDuty)
Create SLO dashboards & burn-rate alerts
Runbooks in repo
(what to check / rollback steps)

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 90: prometheus/rules/recording.rules.yml

```yaml
groups:
- name: recording.rules
  interval: 30s
  rules:
  # API: request rate and error rate
  - record: job:http_requests_total:rate1m
    expr: sum by (service, route, method, status) (rate(http_requests_total[1m]))

  - record: job:http_requests_errors_total:rate5m
    expr: sum by (service) (rate(http_requests_total{status=~"5.."}[5m]))

  # API: p95 latency over 5m (requires http_request_duration_seconds_bucket)
  - record: job:http_request_p95_seconds:5m
    expr: |
      histogram_quantile(
        0.95,
        sum by (service, le) (rate(http_request_duration_seconds_bucket[5m]))
      )

  # Queue backlog / latency (export gauges from Worker)
  - record: job:queue_latency_seconds:5m
    expr: max by (queue, service) (bullmq_queue_latency_seconds)

  - record: job:queue_backlog:5m
    expr: max by (queue, service) (bullmq_queue_backlog)

  # OpenAI error ratio (export totals+errors)
  - record: job:openai_error_ratio:5m
    expr: |
      (sum by (service) (rate(openai_requests_errors_total[5m])) /
       clamp_min(sum by (service) (rate(openai_requests_total[5m])), 1))

  # DB pgvector KNN p95 (requires db_knn_seconds_bucket)
  - record: job:db_knn_p95_seconds:5m
    expr: |
      histogram_quantile(
        0.95,
        sum by (le) (rate(db_knn_seconds_bucket[5m]))
      )
```

Alert rules (concrete thresholds)

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

Listing 91: prometheus/rules/alerts.rules.yml

```
groups:
- name: service.alerts
  interval: 30s
  rules:
  - alert: APIHighP95Latency
    expr: job:http_request_p95_seconds:5m{service="api"} > 0.5
    for: 10m
    labels: {severity: page, team: platform, component: api}
    annotations:
      summary: "API p95 latency is high"
      description: "p95 over last 5m is {{ $value | printf \"%.3f\" }}s (>0.5s). Check
    slow routes/DB."

  - alert: QueueBacklogHigh
    expr: job:queue_backlog:5m{queue="rfp-jobs"} > 500
    for: 15m
    labels: {severity: page, team: data, component: worker}
    annotations:
      summary: "Queue backlog high (rfp-jobs)"
      description: "Backlog > 500 for 15m. Scale workers or fix stuck jobs."

  - alert: QueueLatencyHigh
    expr: job:queue_latency_seconds:5m{queue="rfp-jobs"} > 60
    for: 10m
    labels: {severity: page, team: data, component: worker}
    annotations:
      summary: "Queue end-to-end latency high"
      description: "Latency > 60s for 10m. Check Redis, concurrency, batch sizes."

  - alert: OpenAIErrorRateHigh
    expr: job:openai_error_ratio:5m{service="api"} > 0.1
    for: 10m
    labels: {severity: ticket, team: platform, component: api}
    annotations:
      summary: "OpenAI error ratio > 10%"
      description: "5m failure ratio {{ $value | printf \"%.2f\" }}. Check keys, limits,
    retries/backoff."

  - alert: DBKNNLatencyHigh
    expr: job:db_knn_p95_seconds:5m > 0.2
    for: 15m
    labels: {severity: ticket, team: data, component: postgres}
    annotations:
      summary: "DB KNN p95 > 200ms"
      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 92: prometheus/rules/slo-burn.rules.yml

```
groups:
- name: slo.burn.alerts
  interval: 30s
  rules:
  - record: api:error_ratio:5m
    expr: |
      (sum(rate(http_requests_total{service="api",status=~"5.."}[5m])) /
       clamp_min(sum(rate(http_requests_total{service="api"}[5m])), 1))
  - record: api:error_ratio:30m
    expr: |
      (sum(rate(http_requests_total{service="api",status=~"5.."}[30m])) /
       clamp_min(sum(rate(http_requests_total{service="api"}[30m])), 1))
  - record: api:error_ratio:1h
    expr: |
      (sum(rate(http_requests_total{service="api",status=~"5.."}[1h])) /
       clamp_min(sum(rate(http_requests_total{service="api"}[1h])), 1))
  - record: api:error_ratio:6h
    expr: |
      (sum(rate(http_requests_total{service="api",status=~"5.."}[6h])) /
       clamp_min(sum(rate(http_requests_total{service="api"}[6h])), 1))

  # 99.9% SLO ⇒ budget = 0.001
  - alert: APISLOFastBurn
    expr: (api:error_ratio:5m > 14.4 * 0.001) and (api:error_ratio:1h > 14.4 * 0.001)
    for: 5m
    labels: {severity: page, slo: "api-availability-99.9-30d"}
    annotations:
      summary: "API SLO FAST burn (5m & 1h) > 14.4×"
      description: "Rapid budget burn. 5m={{ $value | printf \"%.3f\" }}; confirm with 1h
    ."

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

Alertmanager config (Slack, PagerDuty, routing)

Files: `./alertmanager/alertmanager.yml`

Listing 93: alertmanager/alertmanager.yml

```
global:
  resolve_timeout: 5m

route:
  receiver: "slack-default"
  group_by: ["alertname","cluster","service","component"]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  routes:
    - matchers: [severity="page"]
      receiver: "pagerduty"
      group_wait: 10s
      repeat_interval: 1h
    - matchers: [team="data"]
      receiver: "slack-data"

receivers:
  - name: "slack-default"
    slack_configs:
      - api_url: "${SLACK_WEBHOOK_URL}"
        send_resolved: true
        title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
        text: |
          *Summary:* {{ (index .Alerts 0).Annotations.summary }}
          *Details:* {{ (index .Alerts 0).Annotations.description }}
          *Labels:* {{ .CommonLabels }}
          *Silence:* <{{ .ExternalURL }}/#/silences>
          *Alerts:* {{ range .Alerts }} - {{ .Labels }} {{ end }}

  - name: "slack-data"
    slack_configs:
      - api_url: "${SLACK_WEBHOOK_URL}"
        channel: "#data-oncall"
        send_resolved: true
        title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
        text: |
          *Data Alert:* {{ (index .Alerts 0).Annotations.summary }}
          *Details:* {{ (index .Alerts 0).Annotations.description }}

  - name: "pagerduty"
    pagerduty_configs:
      - routing_key: "${PAGERDUTY_ROUTING_KEY}"
        severity: "{{ if eq .CommonLabels.severity \"page\" }}critical{{ else }}error{{ end }}"
        send_resolved: true

inhibit_rules:
  - source_matchers: [severity="page"]
    target_matchers: [severity="ticket"]
    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 94: runbooks/APIHighP95Latency.md

```
# APIHighP95Latency
## Triage
1) Grafana API dashboard → "Latency p95 by route".
2) Check 5xx spikes, deploys (last 1h), DB CPU/IO.
## Mitigation
- Scale API/Worker +1.
- Rollback last deploy (<30m).
- Toggle feature flags to reduce load.
## Deep-dive
- Offending route labels.
- Postgres slow queries, pgvector indexes (HNSW params).
- OpenAI limits/errors.
## Closure
- Postmortem issue with timeline + actions.
```

Files: `./runbooks/QueueBacklogHigh.md`

Listing 95: runbooks/QueueBacklogHigh.md

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

Validation & Ops commands

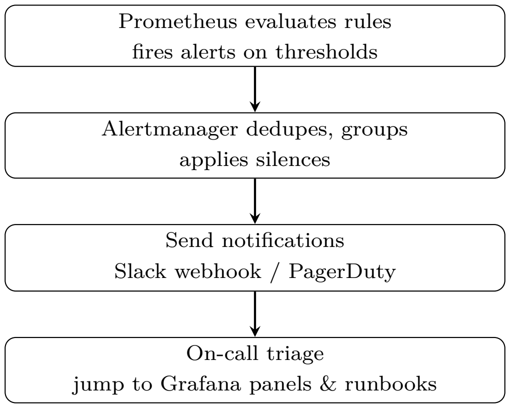
Listing 96: Validate rules and AM config

```
# Validate Prom rules:
docker run --rm -v $PWD/prometheus:/etc/prom prom/prometheus \
  promtool check rules /etc/prom/rules/*.yml

# Validate Alertmanager config:
docker run --rm -v $PWD/alertmanager:/etc/am prom/alertmanager:latest \
  amtool check-config /etc/am/alertmanager.yml

# Reload Prometheus (requires --web.enable-lifecycle already set in Prom setup):
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.


Prometheus evaluates rules
fires alerts on thresholds
Alertmanager dedupes, groups
applies silences
Send notifications
Slack webhook / PagerDuty
On-call triage
jump to Grafana panels & runbooks

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 $\leq$ 500 ms; KNN p95 $\leq$ 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 pool_size	150–250	Transaction pooling mode
pgBouncer max_client_conn	2,000	Headroom for spikes
HTTP timeouts	3–5 s	With retries + circuit breakers
Health/readiness	`/healthz` 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 $\leq$ *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)	`public, max-age=120, s-maxage=300`	Static/public assets
SWR (safe API reads)	`stale-while-revalidate=60`	e.g., `/rfps/:id/matches` (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 `prepared_statements=false` if needed
Active backends	< 500	Keep via pooling + low per-pod conns
Partitions (org)	HASH 32–64	On `org_id` for large shared tables
Partitions (time)	MONTHLY	For `embeddings/chunks`
Read replicas	1–2	Route dashboards/analytics; lag < 2 s
Backfills	5k–20k rows/txn	Off-peak; throttle to keep p95 budget
Online reindex	`REINDEX CONCURRENTLY`	Avoid table locks
RLS strategy	shared table + `org_id`	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 $<$ *2 s*.
- **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 k	50	Candidates from ANN
Shortlist m	200	Re-rank domain scoring; keep KNN p95 $\leq$ 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` $\approx$ *2,000–4,000* 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 → 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	$\leq$ 500 ms	HPA scale-out; throttle producers
KNN p95	$\leq$ 200 ms	Tune k, m, 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 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 ms*, fail rate $<$ *2%*); nightly soak; chaos drills for Redis/DB/LLM brownouts.

15.3 Resilience & Recovery

- **Backups:** rely on managed DB PITR; document RTO/RPO expectations.
- **Idempotency:** re-running `score-matches` replaces prior rows for (`org_id`, `rfp_id`).
- **Kill switches:** per-tenant circuit breaker for OpenAI; global queue pause.

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 $\leq 500\,\text{ms}$; KNN p95 $\leq 200\,\text{ms}$) 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.

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.