CRAD —Microservices Blueprint

# 0) Purpose

Design a production-ready microservices architecture for CRAD that uses Next.js (TypeScript) for the frontend, Node.js (TypeScript) for backend services, Drizzle ORM + PostgreSQL for persistence, and OpenAI-powered reasoning (RAG over a vector database). This document is formatted as an engineering blueprint you can drop into a Replit repo.

# 1) High-Level Architecture

Core components:

* Web App: Next.js (app router, TypeScript).
* API Gateway / BFF (Backend for Frontend): Node.js (Fastify or NestJS), auth, request aggregation.
* Microservices (Node.js, TypeScript):
* Farmer Profile Service (Drizzle + Postgres).
* Soil & Sensor Ingest Service.
* Recommendation Service (LLM Orchestrator + domain tools).
* Knowledge Service (RAG: embeddings + vector DB).
* Market & Policy Service.
* Scraper & Ingestion Service (docs/pdfs/txt/images/links).
* Notifications Service (email/SMS/WhatsApp, optional).
* Database: PostgreSQL (Drizzle ORM migrations/models).
* Vector Database: (Pinecone / Weaviate / pgvector).
* Event Bus: NATS or RabbitMQ (service-to-service async).
* Cache/Queue: Redis (rate limits, jobs).
* Observability: OpenTelemetry + Prometheus-compatible metrics.
* Containerization: Docker; Orchestration optional (K8s, Railway, Fly.io, Render).

# 2) Microservices and Responsibilities

## 2.1 API Gateway / BFF

* Auth (JWT/OAuth2), RBAC enforcement for roles: super\_admin, agronomist, farmer, viewer.
* Routes facade for the frontend; fan-out to services.
* Rate limiting, schema validation (Zod).
* Edge-friendly endpoints for Next.js server actions.

## 2.2 Farmer Profile Service

* Entities: farms, users, rotations, field plots.
* Drizzle ORM schemas + migrations.
* CRUD for farm metadata, geodata (PostGIS optional).

## 2.3 Soil & Sensor Ingest Service

* Ingest soil samples (pH, N, P, K, OM, texture, moisture, salinity, depth, compaction, microbes).
* Stream sensor data; normalize units; store latest + history.
* Webhook/CSV upload endpoints; validation + lineage metadata.

## 2.4 Recommendation Service (LLM Orchestrator)

* Implements crop suitability, fertilizer optimization, pest/disease risk, irrigation planning.
* Calls tools: weather, latest soil, crop rules, policy guardrails.
* Uses OpenAI chat with tool/function calling; composes responses with clear rationale.
* Never emits numeric doses without data; validates via rule engine.

## 2.5 Knowledge Service (RAG)

* Embeddings (OpenAI text-embedding-3-large or latest).
* Vector DB namespaces: crops\_knowledge, fertilizers, pesticides, policies, howto.
* Ingests passages from Scraper & Ingestion Service with metadata (region, crop, source, version).
* Retrieval for /chat and /recommend endpoints.

## 2.6 Market & Policy Service

* Fetches/ingests market prices and demand indicators.
* Stores policy documents (pesticide labels, fertilizer limits, subsidies).
* Exposes policy-check API used by Recommendation Service.

## 2.7 Scraper & Ingestion Service

* Accepts uploads (pdf, docx/txt, images) and URLs for crawling/scraping.
* Extracts text (PDF/Docx parsers, OCR for images), de-duplicates, chunks, cleans.
* Metadata tagging (region, crop, source, date, license).
* Embeds chunks and upserts into Vector DB + stores raw in object storage.
* Admin UI for provenance review, takedown, and re-indexing.

## 2.8 Notifications Service (optional)

* Sends alerts for pest/disease risk windows, irrigation timing, policy changes.

# 3) Tech Stack

* Frontend: Next.js 14+, TypeScript, App Router, Server Actions, Tailwind, shadcn/ui.
* Backend: Node.js 20+, TypeScript, Fastify or NestJS (per service).
* ORM: Drizzle ORM (PostgreSQL).
* DB: PostgreSQL 15+ (optionally with pgvector).
* Messaging: NATS or RabbitMQ; Redis for cache/queues.
* Vector DB: Pinecone, Weaviate, or PostgreSQL + pgvector.
* Storage: S3-compatible for raw documents and artifacts.
* Infra: Docker; CI with GitHub Actions; Replit for dev.

# 4) Data Model (Drizzle ORM, PostgreSQL)

* users(id, email, password\_hash, role, created\_at)
* farms(id, owner\_id->users.id, name, location\_geojson, area\_ha, altitude\_m, slope\_pct)
* soil\_samples(id, farm\_id, date, ph, n\_ppm, p\_ppm, k\_ppm, om\_pct, texture, moisture\_pct, salinity\_ds\_m, depth\_cm, compaction\_kpa, microbes\_json)
* crops(id, common\_name, variety, temp\_min\_c, temp\_max\_c, ph\_min, ph\_max, water\_need\_mm, season\_window, notes)
* rotations(id, farm\_id, season, crop\_id, intercropping\_json)
* recommendations(id, farm\_id, request\_json, response\_json, created\_at)
* market\_prices(id, region, crop\_id, price\_per\_kg, date)
* policies(id, region, type, payload\_json, effective\_date)
* documents(id, owner\_id, kind, uri, mime, hash, meta\_json, created\_at)
* embeddings(id, document\_id, chunk\_id, vector, meta\_json)

# 5) API Contracts (Gateway → Services)

## 5.1 Auth & Roles

* Roles: super\_admin, agronomist, farmer, viewer.
* JWT with short expiry + refresh; RBAC enforced at gateway and service level.
* Zod schemas for request/response validation.

## 5.2 Endpoints (selected)

* POST /auth/login
* POST /farms { create } | GET /farms/:id
* POST /soil/ingest | GET /soil/latest?farm\_id=
* GET /recommend/crops?farm\_id=&season=
* POST /recommend/fertilizer
* POST /recommend/pest-disease
* POST /recommend/irrigation
* POST /chat (RAG + tool calls)
* POST /admin/documents/upload (pdf/docx/txt/images)
* POST /admin/documents/scrape { url, scope, depth }
* POST /admin/documents/reindex { document\_id? filters? }
* GET /admin/documents/status/:id

# 6) Super Admin Capabilities

* User & Role Management: create users, assign roles, revoke access.
* Data Governance: approve/ban sources, manage policies, retention rules.
* Observability: view service health, error rates, ingestion queue, vector DB stats.

## 6.1 Resource Ingestion & Web Scraping (Super Admin UI)

* Upload Center: PDF, DOCX/TXT, images (OCR), CSV; 2 GB per file (configurable).
* Link Scraping: submit URLs or sitemaps; configure domain scope, depth, schedule.
* Extraction Pipeline: detect MIME → parse → OCR (images) → clean → chunk → embed → upsert.
* Metadata Form: region, crop, language, license, version, review\_status.
* Quality & Safety: antivirus scan, size/type limits, PII redaction, license tagging.
* Re-index Controls: full/partial; rollback specific versions.
* Provenance & Citations: store hashes/URIs; show sources in answers.

# 7) LLM Orchestration (OpenAI + Tools)

* System Prompt tuned for agronomy and sustainability; locale-aware (Rwanda-first).
* Tools: get\_weather, get\_latest\_soil, calc\_crop\_suitability, optimize\_fertilizer, predict\_pests, plan\_irrigation, retrieve\_knowledge.
* RAG: top-k passages with metadata shown in answers; avoid hallucinated doses.
* Guardrails: policy checks for pesticides/fertilizers; numeric validations.

# 8) Developer Experience & Repo Layout

* Monorepo (pnpm workspaces): apps/web (Next.js), apps/gateway, services/\*, packages/\* (schemas, clients).
* Infra as Code: docker-compose for local; Helm/K8s optional later.
* Testing: Vitest/Jest for TS; Playwright for e2e; k6 for load tests.
* CI: typecheck, lint (eslint), test, build, containerize, scan, deploy.

# 9) Environment Variables (.env examples)

* OPENAI\_API\_KEY=...
* POSTGRES\_URL=postgresql://user:pass@host:5432/crad
* DRIZZLE\_MIGRATIONS\_DIR=./drizzle
* VECTOR\_DB\_URL=... or use pgvector
* NATS\_URL=nats://localhost:4222
* REDIS\_URL=redis://localhost:6379
* JWT\_SECRET=change\_me
* TZ\_DEFAULT=Africa/Kigali
* REGION\_DEFAULT=RW

# 10) Security & Compliance

* RBAC enforced at gateway and services; least privilege db roles.
* Input validation (Zod), output encoding, prompt injection defenses for tool params.
* Pesticide/fertilizer policy enforcement before emitting doses.
* Audit trails: admin actions, ingestion provenance, model outputs.

# 11) Quickstart (Replit / Local Dev)

1. Create a Node.js (TypeScript) Replit; initialize pnpm workspace.
2. Add apps/web (Next.js) and apps/gateway (Fastify/NestJS).
3. Add services: farmer, soil, recommend, knowledge, market-policy, scraper.
4. Wire Drizzle ORM in each service; generate migrations; run Postgres (Docker).
5. Implement /admin/documents/upload and /admin/documents/scrape first.
6. Set up embeddings + vector DB (pgvector or Pinecone).
7. Implement /chat via Gateway → Knowledge → Recommendation Service.
8. Add role-based UI in Next.js; expose Super Admin console for ingestion/scraping.
9. Add tests and CI pipeline; containerize everything.

# 12) Milestones

* M1: Monorepo scaffold; Auth; Gateway; Postgres + Drizzle; Super Admin ingestion UI (uploads + scrape).
* M2: Knowledge service with RAG; embeddings; citations in chat.
* M3: Recommendation service v0 (crop suitability + irrigation heuristic).
* M4: Fertilizer optimizer + policy guardrails; market integration.
* M5: Pest/disease risk; notifications; observability dashboards.

# 13) Appendix: Drizzle ORM Example (TypeScript)

import { pgTable, serial, integer, varchar, jsonb, timestamp, numeric } from "drizzle-orm/pg-core";  
  
export const soilSamples = pgTable("soil\_samples", {  
 id: serial("id").primaryKey(),  
 farmId: integer("farm\_id").notNull(),  
 date: timestamp("date", { withTimezone: true }).defaultNow(),  
 ph: numeric("ph", { precision: 4, scale: 2 }),  
 nPpm: integer("n\_ppm"),  
 pPpm: integer("p\_ppm"),  
 kPpm: integer("k\_ppm"),  
 omPct: numeric("om\_pct", { precision: 5, scale: 2 }),  
 texture: varchar("texture", { length: 32 }),  
 moisturePct: numeric("moisture\_pct", { precision: 5, scale: 2 }),  
 salinityDsM: numeric("salinity\_ds\_m", { precision: 6, scale: 3 }),  
 depthCm: integer("depth\_cm"),  
 compactionKpa: integer("compaction\_kpa"),  
 microbesJson: jsonb("microbes\_json")  
});