Skills-Graph Architecture

Revision v1.2

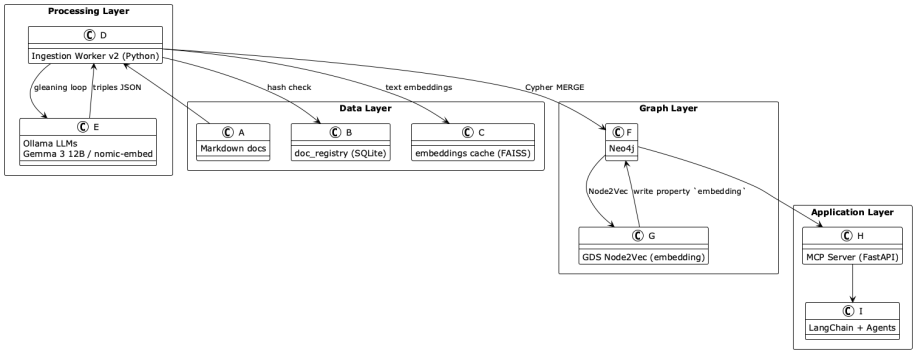
Bernd Prager

2025-05-22

Abstract

End-to-end design for turning Markdown experience records into a Hypergraph-of-Thought (Neo4j) and exposing it through an MCP API that local LLM agents (Ollama) can query.

1 Logical View (high-level)



2 Ingestion Worker v2 (detailed steps)

1. **SHA-256 change detection** – skip unchanged docs (SQLite `doc_registry`).
2. **Chunk & embed** – 1 500-word chunks / 200-word overlap → `nomic-embed-text` vectors → optional FAISS.
3. **Gleaning loop extraction** – up to **3 LLM passes** (Gemma 3 12 B) per chunk; each pass only requests *new* triples.
4. **Cypher MERGE insert** – deterministic `MERGE` for nodes/relations; alias map normalisation.
5. **Registry update** – store new hash & timestamp (UTC).
6. **Node2Vec batch job** – after all files processed: GDS `node2vec.write()` (128-dim, 10 × 20 walks) → node property `embedding`.

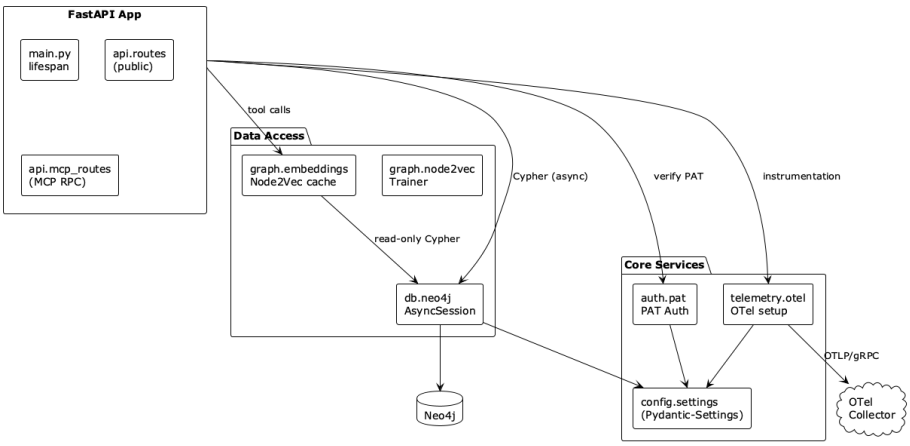
7. **(optional)** Add graph embeddings to FAISS for hybrid doc + structural search.

   Performance note – With gleaning + Node2Vec the first full build takes ~3 × the v1 time, but incremental runs only pay the Node2Vec cost if *any* doc changed.

3 MCP Server (FastAPI) Architecture

The **MCP Server** is a modular FastAPI service that exposes the Skills-Graph through both generic REST endpoints and an *MCP RPC* interface designed for LLM agents.

- **Entry point** – `main.py` bootstraps the FastAPI app, sets up CORS, registers routers and configures graceful startup/shutdown via an async lifespan handler.
- **Configuration** – `config/settings.py` (and a CLI-friendly twin inside `mcp_server.py`) load strongly-typed settings from environment variables or a `.env` file using **Pydantic v2 BaseSettings**, cached with `lru_cache`.
- **Authentication** – `auth/pat.py` implements an in-memory *Personal Access Token* registry and a reusable FastAPI `Depends` guard (`get_current_token`).
- **API surface**
  - `/api/v1/*` – CRUD helpers (e.g., `/skills`) for interactive exploration.
  - `/api/mcp/rpc/*` – RPC endpoints (`initialize`, `resources.*`, `tools.dispatch`) that follow the *Model Context Protocol* spec.
- **Database layer** – `db/neo4j.py` supplies an async Neo4j driver plus a per-request `AsyncSession` dependency (`db.deps.get_db_session`).
- **Graph/ML helpers**
  - `graph.node2vec.py` – full Node2Vec trainer (for offline jobs).
  - `graph.embeddings.py` – lightweight in-process vector index used by the `graph.search` tool.
- **Observability** – `telemetry/otel.py` sets up an OTLP exporter; all routes are auto-instrumented when the relevant OpenTelemetry packages are present.



3.1 Runtime Behaviour

1. **Startup**
   - `lifespan()` reads configuration, sets up OpenTelemetry and verifies Neo4j connectivity; on failure the process exits fast.
2. **Request flow**
   1. Client sends HTTP request with `Authorization: Bearer <PAT>`.
   2. `auth.pat.get_current_token` validates the token.
   3. Route handler obtains an `AsyncSession` via dependency injection.
   4. Business logic executes Cypher queries or vector search.
   5. Successful responses are returned; spans are exported via OTLP.
3. **Shutdown**
   - The Neo4j driver is closed and remaining spans are flushed.

4 Updated Infrastructure Topology

Host	Stack	Ports
odin	Neo4j 5.15 + GDS 2.x	7474 / 7687
odin	Ollama 0.6.8 (local models & `/api/embed`)	11434
odin	Ingestion Worker v2 (systemd)	–
odin	FastAPI MCP server	**8000**

5 Maintenance Jobs

Job	Schedule	Notes
`nightly_dedupe`	03:00	APOC `refactor.mergeNodes`
`node2vec_refresh`	After *any* ingest	Triggered automatically by worker
`refresh_embeddings`	Weekly	Re-runs text embeddings if model upgraded

6 Future Enhancements (next, ordered)

1. **Edge weighting & centrality pre-compute** for richer MCP ranking.
2. **Auto-summary blurb** (store summary on Entity)
3. **Embedding-aware LLM cache** to avoid redundant Gemma calls.
4. **Incremental Node2Vec** once graph size or runtime makes full runs painful
5. **Async ingestion + two-pass RAG** when we start serving high-QPS MCP queries