**Executive Brief — Choosing Neo4j (LPG) vs RDF Triple Store for Enterprise Sharing**

Date: 2025-09-01 21:14 UTC

Audience: Executive & Platform Leadership • Goal: Decide how to model, serve, and share knowledge across BUs and with partners/public.

**1) Snapshot Recommendation**

• Use RDF (triples + SPARQL + SHACL + JSON-LD/DCAT) as the publication/contract layer for cross‑BU and external sharing.

• Use Neo4j (LPG/Cypher + GDS + vector) as the app/runtime graph for fast traversals, algorithms, and GenAI RAG.

• If you want one managed engine, consider Amazon Neptune (SPARQL + openCypher in one cluster).

**2) Decision Grid (What matters most?)**

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Neo4j (LPG/Cypher) | RDF Triple Store (SPARQL) | Hybrid (Amazon Neptune) |
| Primary win | App traversals, GDS algorithms, vector RAG | Interoperability, reasoning, SHACL, federation | One platform, trade-offs on specialization |
| Latency target | ms-level multi-hop | Varies; inference can add cost | Good; varies by model/query |
| Semantics | Pragmatic; no native OWL | RDFS/OWL inference native | RDF + property graph |
| Contracts/Quality | DB constraints + app checks | SHACL shapes native | SHACL via RDF side |
| Sharing externally | Export RDF/JSON-LD via n10s | Native RDF/JSON-LD/SPARQL | RDF & openCypher APIs |
| Developer UX | App-friendly (Cypher) | Ontology/semantic skillset | Both worlds; simpler ops |

**3) Reference Patterns**

A) Knowledge-first (RDF as System of Record; Neo4j as App Graph)

[RDF store: SPARQL, SHACL, DCAT/JSON-LD]  
 │ (publish to BUs/partners/public)  
 └── export RDF snapshots/streams  
 │  
 ▼  
 [Neo4j: Cypher, GDS, Vector] → product APIs & GenAI

B) App-first (Neo4j internal; publish RDF/JSON-LD outward)

[Neo4j: LPG + Cypher/GDS/Vector] → product APIs  
 ├─ n10s export → RDF/JSON-LD → partners/public  
 └─ n10s import ← RDF from partners

C) Single managed engine

[Amazon Neptune]  
 ├─ RDF graphs (SPARQL)  
 └─ Property graphs (openCypher/Gremlin)  
(Front with API Gateway; publish JSON-LD/DCAT)

**4) 30‑Day Rollout Plan**

Week 1 — Scope & Schema: Pick 1–2 CQs; define minimal ontology and LPG labels/relations. Deliverable: signed schema + test queries.

Week 2 — Ingest & Load: Build a small ETL (docs → sections → chunks; entities); load to RDF store and Neo4j; set SHACL & Neo4j constraints. Deliverable: first working graph both sides.

Week 3 — Serving & Share: Expose Cypher APIs (Neo4j) and SPARQL + JSON‑LD (RDF). Add vector search for RAG. Deliverable: demo answering CQs with citations.

Week 4 — Governance & Scale: Add DCAT catalog, access control, metrics (latency, recall/precision), and a runbook. Deliverable: go/no‑go and budget to scale.

**5) Risks & Mitigations**

1) Model drift → SHACL (RDF) + nightly Cypher checks (Neo4j).

2) Two‑stack complexity → Automate RDF↔Neo4j sync (n10s) and CI checks.

3) Latency surprises → Cache hot queries; precompute paths; tune indexes.

4) Skills gap → Pair ontologists with app teams; training on SPARQL & Cypher.

5) Security & quotas → Rate‑limit SPARQL; auth on JSON‑LD; access policies.

**6) Quick Selector**

• Share to many orgs with standards/validation → Start RDF. • Build fast product features → Start Neo4j. • Need both with less ops → Neptune.