**Enterprise Guide: Building a Graph-Backed App for CQ Answering (Neptune/Neo4j)**

*Data -> KG -> Serving | No-NAT VPC | Cognito JWT | Neptune (SPARQL) | Neo4j (optional) | OpenSearch (Snippets)*

Generated: 2025-08-31 19:13 UTC

**[0] Competency Questions (CQs)**

**Objective**

Define the precise questions the system must answer (e.g., “Which 2023 Falcon X trims have ACC and California sales on 2023-03-15?”). CQs drive ontology scope, data mappings, and tests.

**Tools/Service: Definition + Simple Example**

* CQs: a curated list of must-answer questions. Example: “List trims with feature = ACC for model year 2023 and show units sold in CA on 2023-03-15.”
* Issue Tracker/Docs (e.g., Git, Markdown): store and version the CQ list; each CQ gets an ID (CQ-001).

**Purpose / Why / When**

Start here to align stakeholders, fix scope, and make every downstream step testable. Use CQs before ingest begins; update only via change control.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Markdown/Docs, GitHub Issues | AWS CodeCommit/CodeCatalyst, Confluence (self-managed) |
| Docusaurus/Wiki |  |

**Input -> Output -> Next**

* Input: business objectives, examples from SMEs.
* Output: CQ catalog (IDs, phrasing, acceptance criteria, success metrics).
* Next: design ontology terms and data needed to answer each CQ.

**[1] Sources (8 inputs) -> Bronze**

**Objective**

Land raw data exactly as received for traceability and replay.

**Tools/Service: Definition + Simple Example**

* Airbyte/NiFi/Glue: connectors/pipelines to pull Sales DB (JDBC), fetch PDFs, crawl reviews, and drop to S3.
* OCR (Tesseract/Textract): convert scanned PDFs to text.
* Apache Tika: extract text/metadata from digital PDFs.

Example landing layout (S3):  
s3://auto-bronze/  
 sales/2023/03/15/sales\_20230315.csv  
 rnd\_pdfs/RND-42.pdf  
 ocr\_pdfs/INV-771.pdf  
 images/cad/front\_view.png  
 json/parts/parts\_2023-03.json  
 patents/US1234567A.txt  
 reviews/twitter/2023/03/15.jsonl

**Purpose / Why / When**

Create an immutable source of truth. Land everything first; do not cleanse here.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Airbyte, Apache NiFi, Apache Tika, Tesseract OCR | AWS Glue, Amazon Textract, AWS DMS |

**Input -> Output -> Next**

* Input: the eight source types.
* Output: raw files/streams in S3 (“Bronze”).
* Next: Step 2 for cleaning/parsing.

**[2] Ingest & Clean (Silver)**

**Objective**

Standardize formats, parse documents, and apply basic quality checks.

**Tools/Service: Definition + Simple Example**

* Spark/Databricks/Glue ETL: parse CSV/JSON/XML; normalize dates, types; extract text from PDFs.
* Great Expectations: declarative data tests (e.g., ‘quantity must be integer and non-negative’).

Example: Silver schema (CSV)  
sales\_silver.csv:  
 date,sold\_in,trim\_id,quantity  
 2023-03-15,CA,T101,120

**Purpose / Why / When**

Ensure downstream modeling has clean inputs; fail fast if quality gates are violated.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Apache Spark, Great Expectations | AWS Glue (Spark), AWS Step Functions (orchestration) |

**Input -> Output -> Next**

* Input: Bronze files.
* Output: typed, cleaned Silver data (CSV/Parquet + extracted text).
* Next: Step 3 to build conformed ‘Gold’ tables.

**[3] Conform (Gold analytics layer)**

**Objective**

Model the domain as conformed dimensions/facts and bridge tables to align entities across sources.

**Tools/Service: Definition + Simple Example**

* dbt/Spark SQL: transform Silver into Gold (dim\_vehicle\_model, dim\_trim, dim\_feature, fact\_sales, etc.).
* Entity resolution: deterministic or fuzzy matching to deduplicate models/trims/parts.

Gold tables (samples)  
dim\_trim(id, model\_id, name, model\_year)  
dim\_feature(id, name, skos\_pref, skos\_alt[])  
bridge\_trim\_feature(trim\_id, feature\_id)  
fact\_sales(date, region\_id, trim\_id, quantity)

**Purpose / Why / When**

Provide a stable, query-friendly substrate to generate triples and to support analytics baselines.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| dbt Core, Apache Spark, DuckDB (small scale) | AWS Glue/Spark, Amazon Redshift (optional staging) |

**Input -> Output -> Next**

* Input: Silver data.
* Output: Gold dimension/fact tables.
* Next: Step 4-5 to map Gold to RDF using the ontology.

**[4] Ontology & Vocabulary**

**Objective**

Formalize classes, properties, and terms needed to answer CQs; add SKOS labels/synonyms for robust entity linking.

**Tools/Service: Definition + Simple Example**

* OWL/RDFS: define classes (VehicleModel, Trim, Feature, Part, SaleRecord, Region) and relations (belongsTo, hasFeature, soldIn).
* SKOS: preferred/alternate labels (ACC <-> Adaptive Cruise Control).

Turtle snippet  
@prefix ex: <http://example.com/auto#> .  
ex:Trim a rdfs:Class .  
ex:belongsTo a rdf:Property ; rdfs:domain ex:Trim ; rdfs:range ex:VehicleModel .  
ex:ACC a skos:Concept ; skos:prefLabel "Adaptive Cruise Control" ; skos:altLabel "ACC" .

**Purpose / Why / When**

Make semantics explicit so SPARQL can be written once and data can evolve underneath.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Protege, RDF4J/Jena, SKOS vocabularies | Host ontology files in S3/CodeCommit; use Neptune for reasoning where applicable |

**Input -> Output -> Next**

* Input: CQ list + Gold schemas.
* Output: ontology.ttl, skos.ttl.
* Next: Step 5 to map rows -> triples.

**[5] RDF Mapping & Triple Generation**

**Objective**

Convert Gold rows into RDF triples with minted URIs and controlled vocabularies.

**Tools/Service: Definition + Simple Example**

* R2RML/RML: declarative mapping from tables to RDF (or a small Python mapper).
* Serialization: N-Triples (.nt) or Turtle (.ttl) for bulk load.

Sample triples (.nt)  
<http://example.com/trim/T101> <http://example.com/auto#belongsTo> <http://example.com/model/M1> .  
<http://example.com/trim/T101> <http://example.com/auto#hasFeature> <http://example.com/feature/F1> .  
<http://example.com/sale/S1> <http://example.com/auto#forTrim> <http://example.com/trim/T101> .

**Purpose / Why / When**

RDF is the wire format for Neptune’s SPARQL and for cross-dataset linking.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| RMLMapper, RDFLib (Python) | Use AWS Glue/Spark + custom mapper; store outputs in S3 |

**Input -> Output -> Next**

* Input: Gold tables + ontology/SKOS.
* Output: RDF files (N-Triples/Turtle).
* Next: Step 6 for SHACL quality gating.

**[6] Validation & Reasoning (SHACL/OWL)**

**Objective**

Enforce graph quality and optional inference before loading to production KG.

**Tools/Service: Definition + Simple Example**

* SHACL: shapes/constraints (e.g., Trim must belongTo exactly one VehicleModel; SaleRecord requires forTrim, soldIn, saleDate, quantity).
* Reasoners (Hermit/ELK): materialize inferences (optional).

SHACL shape (snippet)  
ex:SaleRecordShape a sh:NodeShape ;  
 sh:targetClass ex:SaleRecord ;  
 sh:property [ sh:path ex:quantity ; sh:datatype xsd:integer ; sh:minInclusive 0 ] .

**Purpose / Why / When**

Catch structural/systemic issues early; provide compliance evidence to stakeholders.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| pySHACL, Apache Jena SHACL | Run validators in Glue/SageMaker/Containers; store reports in S3 |

**Input -> Output -> Next**

* Input: RDF triples; SHACL shapes.
* Output: pass/fail report; cleaned triples if needed.
* Next: Step 7 to load Neptune/Neo4j.

**[7] Load Graph DB (Neptune or Neo4j)**

**Objective**

Load validated triples into a graph store and expose SPARQL/Cypher endpoints.

**Tools/Service: Definition + Simple Example**

* Neptune: SPARQL 1.1 endpoint; bulk loader from S3; IAM auth (optional).
* Neo4j: with n10s plugin to import RDF to labeled property graph.

Neptune bulk loader payload (simplified)  
{  
 "source": "s3://bucket/neptune\_sample.nt",  
 "format": "ntriples",  
 "iamRoleArn": "arn:aws:iam::<acct>:role/neptune-loader"  
}

**Purpose / Why / When**

Centralize facts and relations for low-latency CQ answering and graph traversals.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Neo4j Community + n10s, Apache Jena TDB2 | Amazon Neptune |

**Input -> Output -> Next**

* Input: validated RDF.
* Output: KG in Neptune/Neo4j; SPARQL/Cypher endpoints.
* Next: Step 8A/8B for supporting snippets/vector search.

**[8A] Text/Search Index (Snippets)**

**Objective**

Index R&D/OCR/manual text for evidence snippets to accompany KG answers.

**Tools/Service: Definition + Simple Example**

* OpenSearch: BM25 keyword ranking; store docId to entity links.

Sample document  
{ "doc\_id":"RND-42", "title":"Falcon X ACC Tuning",  
 "text":"Improved ACC sensor fusion for Falcon X Pro...",  
 "tags":["Falcon X","ACC","T101"] }

**Purpose / Why / When**

Provide human-readable evidence alongside structured facts; improves trust.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Elasticsearch OSS, Whoosh (small) | Amazon OpenSearch Service (VPC domain) |

**Input -> Output -> Next**

* Input: extracted text + links from Step 5/3.
* Output: searchable index ('docs').
* Next: Step 9 orchestrator queries this for snippets.

**[8B] Vector Index (RAG, optional)**

**Objective**

Enable semantic retrieval for open-ended questions and disambiguation.

**Tools/Service: Definition + Simple Example**

* Embeddings: sentence transformers/OpenAI; kNN index in OpenSearch or Qdrant.

Chunk record  
{ "chunk\_id":"c001", "embedding":[...], "entities":["T101","ACC"], "text":"..." }

**Purpose / Why / When**

Use when keywords are insufficient or for long-form answers; combine with KG grounding.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| Qdrant, FAISS, sentence-transformers | OpenSearch kNN, Bedrock Titan Embeddings (optional) |

**Input -> Output -> Next**

* Input: cleaned text chunks; entity metadata.
* Output: vector index.
* Next: Step 9 can choose vector path for open-ended queries.

**[9] Query Orchestrator (App/Service Layer)**

**Objective**

Turn a natural language question into a precise, answerable plan using the KG, text, and optional federation.

**Tools/Service: Definition + Simple Example**

* Cognito + API Gateway (JWT): secure endpoints.
* Lambda (query\_handler): choose path (SPARQL vs BM25/vector); call Neptune/OpenSearch; optional SPARQL SERVICE to Wikidata.

Example logic (pseudo)  
if looks\_like\_fact\_CQ(q):  
 trims = SPARQL(CQ1\_using\_SKOS\_labels)  
 sales = SPARQL(CQ2\_on\_trim\_set)  
 snippet = OpenSearch('Falcon X ACC')  
else:  
 passages = VectorSearch(q); snippet = passages[0]  
return compose\_answer(trims, sales, snippet)

**Purpose / Why / When**

Encodes routing, joins multi-source results, and returns a single coherent answer with citations.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| FastAPI/Flask, LangChain/LlamaIndex (optional) | Amazon API Gateway, AWS Lambda, AWS Step Functions |

**Input -> Output -> Next**

* Input: user question + JWT; KG and search endpoints.
* Output: JSON answer (facts + evidence + citations).
* Next: Step 10 renders to the user.

**[10] Final Answer to User (UI)**

**Objective**

Present structured facts with evidence and links; enable follow-ups and drill-downs.

**Tools/Service: Definition + Simple Example**

* Static web (S3/CloudFront) or React app; login via Cognito Hosted UI or custom form.

Response (example)  
{  
 "answer": {"T101":120,"T102":55},  
 "trims": ["T101","T102"],  
 "evidence": [{"doc\_id":"RND-42","text":"Improved ACC sensor fusion..."}]  
}

**Purpose / Why / When**

Human-centered presentation with provenance builds trust and enables decisions.

**Open Source vs AWS**

|  |  |
| --- | --- |
| Open Source Tools | AWS Services |
| React/Vue, Keycloak (alt IdP) | Amazon S3 + CloudFront (static), Amazon Cognito |

**Input -> Output -> Next**

* Input: JSON from the orchestrator.
* Output: UI view with tables, chips, and citations.
* Next: optional feedback loop to flag data/linking issues.

**Cross-Cutting Considerations (Enterprise)**

**Security & Networking**

* No-NAT VPC; Interface VPC Endpoints (cognito-idp, logs, sts), S3 Gateway endpoint; SG: Lambda->Neptune:8182, Lambda->OpenSearch:443.
* JWT on /query; enable Neptune IAM auth + SigV4; least-privilege IAM for loader and Lambdas.

**Governance & Catalog**

* Track ontology/SKOS versions; catalog datasets and provenance; data retention policies.

**Quality & Testing**

* Great Expectations for tabular; SHACL for graph; CQ regression suite (expected answers).

**Observability**

* CloudWatch logs/metrics; query latency SLOs; loader job metrics; index health.

**Performance & Cost**

* Right-size Neptune/OpenSearch; batch bulk-loads; compress triples; prune indices; cache frequent SPARQL results.

**Change Management**

* Branch ontology and mappings; backfill pipelines; version URIs when necessary.