Car Domain App Plan — Logical & Physical Workflow with Ontology/KG Standards

# 1) Logical application workflow (text diagram)

[CQs] → define must-answer questions & KPIs  
 │  
 ▼  
[Ontology (OWL/RDFS + SKOS terms)]  
 │ (classes: VehicleModel, Trim, Feature, Part…; properties: belongsTo, hasFeature…)  
 │  
 ▼  
[Ingest 8 sources → Lakehouse Bronze/Silver/Gold]  
 Sales(DB) | R&D PDFs | OCR PDFs | Images/Design | XML/JSON | Reviews | Features/Parts | Patents  
 - Parse (Tika/GROBID/JSON/XML) - OCR (Tesseract/Textract) - Clean/normalize - IDs  
 - Map enums → SKOS concept schemes (features, markets)  
 - Store as Parquet (Delta/Iceberg)  
 │  
 ▼  
[KG Build (RDF)]  
 - Map Gold tables/XML/JSON → RDF (R2RML/RML/SPARQL-Generate)  
 - Entity resolution (same model/trim/part across sources)  
 - Write triples to Graph DB  
 │  
 ▼  
[Validation]  
 - SHACL shapes (cardinality, datatypes) → block bad loads  
 - Reasoner checks (consistency) on OWL  
 │  
 ▼  
[Indexes for unstructured]  
 - Text chunks (R&D/OCR/Patents/Reviews) → BM25 + Vector store  
 - Image captions/embeddings → Vector store  
 │  
 ▼  
[Q&A Runtime (Hybrid)]  
 Router → (a) SPARQL over Graph DB for facts  
 (b) Vector/BM25 for doc/image evidence  
 (c) Federated SPARQL to external endpoints (e.g., Wikidata) when needed  
 Merge + re-rank → Answer composer → Citations (KG URIs + doc/page)

# 2) Physical deployment (AWS-first; OSS in parentheses)

[Data Sources]  
 Sales(DB), PDFs, Images, XML/JSON, Reviews/Twitter, Features/Parts, Patents  
  
[Landing & Storage]  
 S3 Data Lake (MinIO) + Delta/Iceberg ← raw/bronze, silver, gold  
  
[Ingestion/ETL/Orchestration]  
 AWS Glue/Spark (Apache Spark), MWAA/Airflow, DMS, Lambda, Step Functions (NiFi/Airbyte/Kafka)  
  
[Document/OCR/NLP]  
 Textract (Tesseract) • Tika/GROBID • spaCy/HF NER • chunker  
  
[Ontology/KG]  
 Ontology authoring: Protégé + OWL/RDFS/SKOS + Git/CI  
 Graph DB: Amazon Neptune (GraphDB/Jena/Blazegraph/Neo4j\*)  
 RDF mapping: R2RML/RML/SPARQL-Generate (Morph-KGC)  
  
[Search/Retrieval]  
 OpenSearch BM25 + kNN • Vector store: OpenSearch vector / Qdrant / Milvus / FAISS  
  
[Serving/Apps]  
 API Gateway + Lambda/FastAPI on ECS/EKS • Q&A Orchestrator (LangChain/LlamaIndex/Haystack)  
 Auth (Cognito/IAM), Cache (Redis/ElastiCache)  
  
[Observability/Governance]  
 CloudWatch + Prometheus/Grafana • Great Expectations • OpenLineage •  
 SHACL validation job • Reasoner in CI (HermiT/Pellet)  
  
\*If using Neo4j (property graph), mirror the RDF KG or ingest directly; keep OWL/SHACL in RDF side.

# 3) Where each term fits (phase + purpose)

* CQs (Competency Questions) — Plan: define scope (e.g., 'Which 2023 trims of Model X have ACC and what were CA sales?').
* Ontology (OWL + RDFS) — Model: schema of classes/relations/constraints to align all sources.
* RDF — Represent: triple data model (subject–predicate–object).
* RDFS — Basic schema: types, subclassing, domain/range.
* OWL — Rich semantics: cardinalities, equivalence, disjointness; enables reasoning/consistency.
* SKOS — Controlled vocabularies: feature/market terms, synonyms with prefLabel/altLabel.
* SHACL — Validate instances: shapes enforce data quality (e.g., each Trim has exactly one belongsTo).
* Knowledge Graph (KG) — Data layer: instance data built using your ontology.
* Graph DB — Storage/Query: Neptune/GraphDB/Jena (SPARQL) or Neo4j/Janus (Cypher/Gremlin).
* SPARQL — Query: retrieve structured facts; supports federated queries via SERVICE.
* Graph DB Federated Queries — Reach external KGs (e.g., Wikidata) inside one query.

# 4) Standards relationship & order (why/when)

[ XML / JSON(-LD) ] → serialization formats from sources/APIs  
 │  
 ▼  
[ RDF ] → data model (triples) you’ll load into the KG  
 │  
 ├─[ RDFS ] → basic schema (classes, properties, domain/range)  
 │  
 ├─[ SKOS ] → controlled vocabularies (features, markets, synonyms)  
 │  
 └─[ OWL ] → rich ontology constraints/inference over your RDFS model  
 │  
 ▼  
[ SHACL ] → runtime validation of instance data against shapes

# 5) Tesla internal + BMW USA external (federation workflow)

Goal: “List 2023 Tesla Model Y trims with ACC and summarize U.S. BMW sedan context.”  
  
(1) Internal Tesla facts first  
 - Entity link: 'Tesla Model Y' → internal URIs (ex:model/TESLA\_MODEL\_Y, ex:feature/ACC).  
 - SPARQL (internal Graph DB): get trims, features, sales by region/date.  
 - Produce authoritative internal facts + citations to internal docs.  
  
(2) Augment with external BMW USA  
 - Federated SPARQL using SERVICE to Wikidata (or another SPARQL endpoint) for BMW entities.  
 - If sales numbers aren't in a SPARQL endpoint, use web RAG: crawl official reports/news → chunk → embed → retrieve → summarize; link extracted facts via owl:sameAs (ex:org/BMW\_NA owl:sameAs wd:Q256747).  
 - Merge: internal Tesla results + external BMW context → final answer with dual citations.  
  
Federated SPARQL sketch:  
SELECT ?teslaTrim ?year ?bmwModel ?bmwLabel WHERE {  
 SERVICE <https://your-internal-neptune/sparql> {  
 ?teslaTrim a ex:Trim ; ex:belongsTo ex:TESLA\_MODEL\_Y ;  
 ex:hasFeature ex:ACC ; ex:modelYear "2023"^^xsd:gYear .  
 }  
 SERVICE <https://query.wikidata.org/sparql> {  
 ?bmwModel wdt:P176 wd:Q153 ; # manufacturer BMW  
 rdfs:label ?bmwLabel .  
 FILTER(LANG(?bmwLabel) = "en")  
 }  
}

# 6) Minimal data structures (stable identifiers)

URIs  
 http://example.com/model/MODEL\_Y  
 http://example.com/trim/TESLA\_M3P\_2023  
 http://example.com/feature/ACC  
 http://example.com/market/US-CA  
  
Key RDF triples  
 <…/trim/TESLA\_M3P\_2023> ex:belongsTo <…/model/MODEL\_3> .  
 <…/trim/TESLA\_M3P\_2023> ex:hasFeature <…/feature/ACC> .  
 <…/sale/S1> ex:forTrim <…/trim/TESLA\_M3P\_2023> ;  
 ex:soldIn <…/market/US-CA> ;  
 ex:quantity "120"^^xsd:integer .  
  
SKOS example  
 <…/feature/ACC> skos:prefLabel "Adaptive Cruise Control" ;  
 skos:altLabel "ACC" .  
  
SHACL guardrail  
 Every ex:Trim has exactly one ex:belongsTo and at least one ex:modelYear.

# TL;DR

Model with OWL/RDFS + SKOS, store facts in RDF inside a Graph DB, validate with SHACL, answer with SPARQL and augment externally via federated SPARQL (and RAG when no SPARQL).

Generated on: 2025-08-31 04:01 UTC