Shark Bake-Off

Database Benchmarking Project Plan

Comparing Neo4j, Memgraph, and PostgreSQL
for Knowledge Base Architecture

Integrated Planning Team

Version 1.0

January 2026

Status: Planning Complete - Ready for Implementation

This document establishes the foundation for database benchmarking comparing graph and relational approaches for a Knowledge Base system.

Contents

Shark Bake-Off: Database Benchmarking Project Plan

Version: 1.0 **Date:** 2026-01-02 **Status:** Planning Complete - Ready for Implementation

---

Table of Contents

Executive Summary

Central Question

"Is Graph Fast Enough?"

This benchmark evaluates whether a graph database (Neo4j, Memgraph) can meet the performance requirements for a Knowledge Base (KB) system supporting real-time military tracking, while preserving the curation flexibility that graph databases provide.

Context

- **Current state:** Legacy PostgreSQL-based KB with significant curation pain points (30+ min update delays, no relationship CRUD, schema changes require PG Admin)
- **New dataset:** Being curated in Neo4j with excellent results (flexible schema, relationship management, batch operations)
- **Decision needed:** Determine the best KB architecture for the production application

Approach

1. **Phase A:** Find optimal PostgreSQL configuration and optimal Graph configuration independently
2. **Phase B:** Head-to-head comparison across all evaluation dimensions
3. **Phase C:** Apply "Is Graph Fast Enough?" decision criteria
4. **Phase D:** Implement mitigation strategies if needed

Project Overview

What We're Comparing

Database	Type	Role in Benchmark
Neo4j	Graph	Primary graph candidate (curator favorite, proven with new dataset)
Memgraph	Graph	Graph alternative (performance comparison, Bolt-compatible)
PostgreSQL	Relational	Baseline comparison, potential hybrid component

Domains Under Test

Domain	Entities	Identifiers
Air	Aircraft (existing dataset)	Mode_S, Tail Number
Maritime	Ships (~450 with identifiers)	MMSI, SCONUM

Key Constraints

- **AuraDB not an option** (proprietary constraints)
- **Neo4j Community Edition** viable for curation-focused hybrid
- **Curator preference:** Neo4j tooling (Bloom) strongly preferred over alternatives

---

Evaluation Dimensions

Dimension	Weight	Description
Performance	25%	Latency (p50/p95/p99), throughput, “fast enough” pass/fail
Curation Capability	20%	Relationship CRUD, visualization, curator workflow
Schema Evolution	15%	Self-service additions, migration complexity, lag time
Cost	15%	3-year TCO (infrastructure, licensing, development, maintenance)
Data Integration Fit	10%	Property metadata support, flexible mapping, lineage capability
Development Complexity	10%	Initial build effort, developer availability
Maintenance Burden	5%	Ongoing code changes, operational overhead

---

Databases Under Test

Neo4j

Aspect	Details
Type	Native graph database
Query language	Cypher

Aspect	Details
Storage	Disk-based with caching
Tooling	Bloom (visualization), Browser, APOC procedures
Licensing	Community (free, limited) / Enterprise ($150K+/year)
Current status	Curator favorite, new dataset working well

Memgraph

Aspect	Details
Type	In-memory graph database
Query language	Cypher (compatible)
Storage	In-memory with persistence
Tooling	Lab (visualization), MAGE algorithms
Licensing	Community (free) / Enterprise (varies)
Performance claim	Up to 120x faster than Neo4j for some workloads
Bolt compatible	Yes (Neo4j drivers work)

PostgreSQL

Aspect	Details
Type	Relational database
Query language	SQL
Storage	Disk-based with extensive caching
Tooling	PgAdmin, extensive ecosystem
Licensing	Open source (free)
Graph queries	Recursive CTEs, junction tables

Languages and Drivers

Implementation Matrix

Language	PostgreSQL Raw	PostgreSQL ORM	Neo4j Raw	Neo4j OGM	Memgraph
Python	asyncpg	SQLAlchemy	neo4j-driver	neomodel	GQLAlchemy
Go	pgx	Ent	neo4j-go-driver	(none)	neo4j-go-driver*
Java	JDBC	Spring Data JPA	Neo4j Java Driver	Spring Data Neo4j	Neo4j Java Driver*

Language	PostgreSQL Raw	PostgreSQL ORM	Neo4j Raw	Neo4j OGM	Memgraph
Rust	tokio-postgres	Diesel/SeaORM	neo4rs	(none)	neo4rs*

*Memgraph is Bolt-compatible; Neo4j drivers work

Language Rationale

Language	Purpose
Python	Developer productivity baseline, common in data engineering
Go	Modern compiled language, excellent concurrency
Java	Enterprise standard, mature ecosystem
Rust	Performance ceiling (maximum possible speed)

Performance Requirements

"Fast Enough" Thresholds

Query Type	Target (p50)	Acceptable (p95)	Maximum (p99)
Identifier Lookup	< 10ms	< 50ms	< 100ms
2-hop Traversal	< 50ms	< 150ms	< 300ms
3-hop Traversal	< 100ms	< 300ms	< 500ms
6-hop Traversal	< 500ms	< 1000ms	< 2000ms
Property Write	< 50ms	< 200ms	< 500ms
Relationship Write	< 100ms	< 300ms	< 500ms

Throughput Requirements

Scenario	Baseline	Peak	Stress Test
Identifier lookups/sec	625	1,000	1,750
Analytics queries/sec	100	200	500
Writes/sec	30	100	500

Pass/Fail Criteria

```
PASS: Graph database meets all p99 thresholds at peak load (1,000 qps)

CONDITIONAL PASS: Graph meets thresholds with Redis caching
                  (minimum 80% cache hit rate achievable)
```

```
FAIL: Graph cannot meet thresholds even with caching
      → Trigger hybrid architecture evaluation
```

Curation Requirements

Current Pain Points

Issue	Current State	Impact
Update delay	30+ minutes after approval	Curator frustration, stale data
Relationship CRUD	Not possible in app	Essential functionality missing
Schema changes	Requires PG Admin + DBA	Days of delay, bottleneck
Cache invalidation	Slow, unpredictable	Manual deployment required

Current Workflow

```
SME suggests → Curator reviews → Manual deploy → Cache invalidates
                                 (bottleneck)    (30+ min delay)
```

Desired Workflow

```
SME suggests → Curator reviews → Immediately available
               (ontology check)   (seconds to minutes)
```

Curation Capability Requirements

Must Have (Non-negotiable)

ID	Requirement	Target
C1	Relationship CRUD through UI	Functional
C2	Property updates visible	< 5 minutes
C3	Node creation visible	< 5 minutes
C4	Add new properties without DBA	Self-service
C5	Add new relationship types without DBA	Self-service
C6	Visualization of entity relationships	Bloom-quality

Should Have (Strongly Desired)

ID	Requirement	Target
C7	Property updates visible	< 1 minute
C8	Batch operations (Cypher or equivalent)	Supported
C9	SME direct access for low-risk changes	Configurable RBAC

ID	Requirement	Target
C10	Automated schema change tests	CI/CD integration
C11	Audit trail (who changed what, when)	Complete

Conflict Resolution

- **Policy:** Last write wins
- **Implementation:** Timestamp-based, optimistic concurrency
- **Audit:** Full trail of all changes

Access Model

Actor	Permissions
SME	Direct access for some low-risk changes
Curator	Reviews for ontology/schema issues, approves changes

Change Volume

- **Current:** 20-50 corrections/day (property, label, relationship)
- **Future:** Rapid intermittent expansion expected
- **Knowledge generation:** Exponential relationship growth over 2 years

---

Schema Evolution Requirements

The Requirement

Aspect	Specification
Frequency	Potentially daily schema changes
Who	Curators (not developers)
Acceptable lag	< 24 hours (faster preferred)
Examples	New properties, new relationship types, new entity types

Comparison

Operation	PostgreSQL	Neo4j	Memgraph
Add property	ALTER TABLE / migration	Just add it	Just add it
Add relationship type	New junction table + migration	Just create it	Just create it

Operation	PostgreSQL	Neo4j	Memgraph
Add entity type	New table + migration	Just create nodes	Just create nodes
Deployment	CI/CD pipeline, possible downtime	Immediate	Immediate
Curator self-service	No (requires developer)	Yes	Yes
Impact on running system	Table locks possible	None	None

Data Integration Context

> **Note:** Reconciliation engine implementation is a separate project. The benchmark evaluates how well each database supports reconciliation needs.

External Data Sources

Aspect	Specification
Source count	15+ external sources
Update frequency	Varies: few daily, most monthly or longer
Trust model	Partially defined (needs completion)

Reconciliation Requirements (for evaluation)

Requirement	Description
Per-property metadata	Track source, timestamp, curator flags
Curator lock behavior	Prevent source overwrites of curator corrections
Conflict detection	Identify when source data conflicts with KB
Flexible mapping	Handle diverse source schemas

Data Lineage (per property)

```
Property: "tail_number"
Value: "N12345"

Metadata:
├── source: "FAA_Registry"
├── source_timestamp: 2025-12-15
├── ingested_at: 2025-12-16
├── curator_modified: true
```

```
├── curator_id: "jsmith"
├── curator_timestamp: 2025-12-20
├── curator_reason: "FAA typo"
└── locked: true (prevent source overwrites)
```

Benchmark Scenarios

Performance Scenarios

ID	Scenario	Description	Primary Metric
S1	Single identifier lookup	Mode_S or MMSI → properties	p99 latency
S2	Batch identifier lookup	100 IDs at once	Throughput
S3	2-hop traversal	Operator network query	p99 latency
S4	3-hop traversal	Shared operator aircraft	p99 latency
S5	6-hop traversal	Full platform heritage	p99 latency
S6	Cross-domain query	Air + Maritime by country	p99 latency
S7	Property update	Single property write	p99 latency
S8	Relationship creation	New relationship	p99 latency
S9	Mixed workload	Parametric sweep	Throughput, latency

Curation Scenarios

ID	Scenario	Description	Metric
CT1	Property update latency	Time until visible in app	Seconds/minutes
CT2	Node creation latency	Time until queryable	Seconds/minutes
CT3	Relationship creation	Create via UI, verify	Functional + latency
CT4	Schema addition (property)	Add without DBA	Time + steps
CT5	Schema addition (rel type)	Add without DBA	Time + steps
CT6	Batch import	1,000 entities with relationships	Time
CT7	Visualization quality	Curator subjective assessment	Rating

Sync Scenarios (for hybrid)

ID	Scenario	Description	Target
ST1	Curation-to-query sync	Change in Neo4j → visible in query system	< 5 minutes
ST2	Schema sync	New property → queryable everywhere	< 24 hours

---

Test Matrix

Dimension 1: Databases

- Neo4j
- Memgraph
- PostgreSQL

Dimension 2: Languages

- Python (FastAPI + asyncio)
- Go (standard library HTTP + goroutines)
- Java (Spring Boot)
- Rust (Actix-web or Axum)

Dimension 3: Drivers

- Raw drivers (baseline)
- ORM/OGM (overhead measurement)

Dimension 4: Caching

- No cache (baseline)
- Redis cache
- In-memory cache (Python comparison)

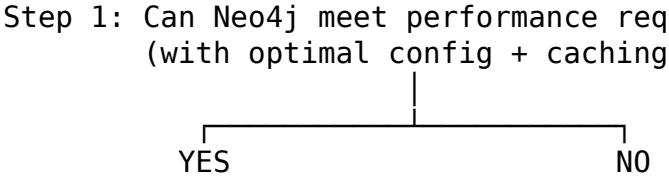
Dimension 5: Workload Mix (Parametric Sweep)

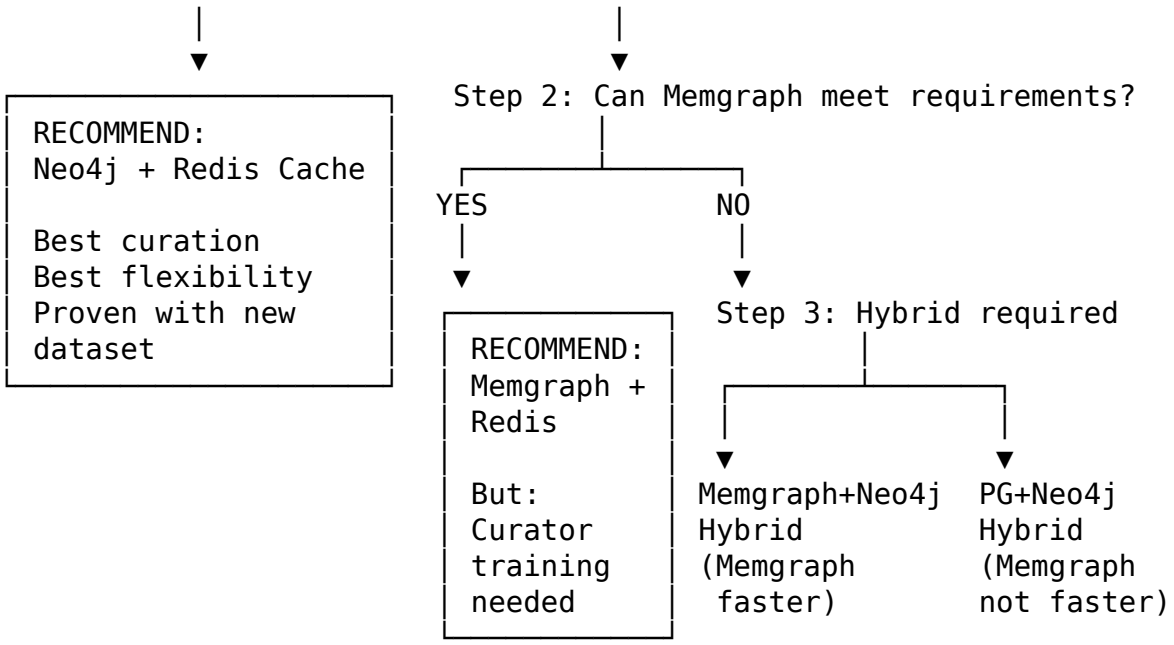
```
Lookup-heavy:     [95/4/1, 90/8/2, 85/12/3, 80/15/5, 75/20/5]
Balanced:         [60/35/5, 50/40/10, 40/45/15]
Analytics-heavy:  [30/60/10, 20/70/10, 10/80/10]
Write-heavy:      [50/20/30, 40/20/40, 30/20/50]
```

Dimension 6: Domains

- Air (aircraft with Mode_S, Tail Number)
- Maritime (ships with MMSI, SCONUM)
- Cross-domain queries

Architecture Decision Framework





Mitigation Strategies

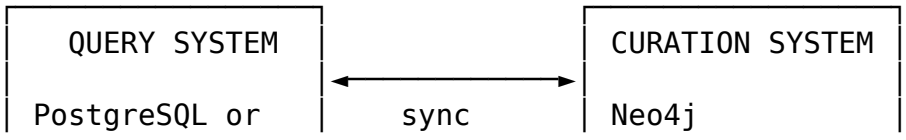
When Triggered

Mitigation is needed if graph database fails to meet p99 thresholds at peak load (1,000 qps) even with Redis caching at 80%+ hit rate.

Options

Option	Description	When to Use
D1: Aggressive Caching	Redis L2 + in-process L1, extended TTL, cache warming	Graph close to thresholds
D2: Query Optimization	Composite indexes, denormalization, APOC procedures	Specific queries slow
D3: PostgreSQL + Neo4j Hybrid	PG for lookups, Neo4j Community for curation	Graph far from thresholds
D4: Memgraph + Neo4j Hybrid	Memgraph for performance, Neo4j for curation (Bloom)	Memgraph much faster, curators need Bloom

Hybrid Architecture (D3/D4)



Sync Strategies

Approach	Complexity	Latency	Recommendation
Periodic refresh	Low	5-60 min	Start here
CDC (Debezium)	High	Seconds	Upgrade if needed
Application dual-write	Medium	Seconds	Risk of inconsistency

Cost Analysis Framework

Infrastructure Costs (AWS, estimated annual)

Component	Graph-Only	Relational-Only	Hybrid
Neo4j (self-hosted)	$3,000	-	$3,000
PostgreSQL (RDS)	-	$4,800	$4,800
Redis (ElastiCache)	$1,800	$1,800	$1,800
Kafka (MSK)	$3,600	$3,600	$3,600
Compute (Fargate)	$1,800	$1,800	$2,400
Monitoring	$600	$600	$900
Annual Total	**~$10,800**	**~$12,600**	**~$16,500**

Licensing Costs (annual)

License	Graph-Only	Relational-Only	Hybrid
Neo4j Enterprise	$150K-300K	-	-
Neo4j Community	$0	-	$0
PostgreSQL	$0	$0	$0
Memgraph (if used)	Varies	-	Varies

Development Costs (estimated hours)

Task	Graph-Only	Relational-Only	Hybrid
Core API	200	200	280
Caching layer	40	40	40
Curation UI	0 (Bloom)	360	0 (Bloom)
Knowledge gen pipeline	80	120	100
Sync mechanism	0	0	120
Testing	80	80	120
Total Hours	**~400**	**~800**	**~660**

Implementation Phases

Phase 1: Foundation

- ☐ Create project structure and git repo
- ☐ Set up Docker Compose (Neo4j, Memgraph, PostgreSQL, Redis, Kafka)
- ☐ Design PostgreSQL schemas (lookup + activity)
- ☐ Add Maritime domain schema
- ☐ Implement Neo4j → PostgreSQL migration
- ☐ Validate Air + Maritime data (~450 ships with MMSI/SCONUM)
- ☐ Create multi-hop query implementations for all DBs
- ☐ Define cross-domain queries

Phase 2: Python Implementation

- ☐ Neo4j: raw driver AND neomodel OGM
- ☐ Memgraph: raw driver AND GQLAlchemy
- ☐ PostgreSQL: asyncpg AND SQLAlchemy
- ☐ FastAPI with cache abstraction
- ☐ Redis + in-memory cache
- ☐ Kafka producer for activity logging
- ☐ Tests

Phase 3: Go Implementation

- ☐ Repository interfaces
- ☐ pgx, neo4j-go-driver (raw)
- ☐ Ent ORM for PostgreSQL
- ☐ HTTP server
- ☐ Redis cache + Kafka producer

Phase 4: Java Implementation

- ☐ Spring Boot setup
- ☐ Raw JDBC/Neo4j Driver AND Spring Data
- ☐ Redis cache with Spring Cache

- ☐ Kafka with Spring Kafka

Phase 5: Rust Implementation

- ☐ Actix-web or Axum HTTP server
- ☐ tokio-postgres / sqlx for PostgreSQL
- ☐ neo4rs for Neo4j/Memgraph
- ☐ Redis with deadpool-redis
- ☐ Establish performance ceiling

Phase 6: Activity Storage & Message Queue

- ☐ Kafka setup
- ☐ Activity consumer (batch writer)
- ☐ Activity storage implementation
- ☐ Read replica configuration

Phase 7: Benchmark Harness

- ☐ HDR Histogram metrics collector
- ☐ Parametric workload generator
- ☐ Pass/fail evaluation against thresholds
- ☐ Multi-hop query benchmarks
- ☐ Cross-domain query benchmarks
- ☐ Locust load test scripts
- ☐ Prometheus + Grafana dashboards

Phase 8: Curation Testing

- ☐ Curation scenario tests (CT1-CT7)
- ☐ Schema evolution tests
- ☐ Visualization assessment

Phase 9: Analysis - Phase A (Optimization)

- ☐ Find optimal PostgreSQL configuration
- ☐ Find optimal Neo4j configuration
- ☐ Find optimal Memgraph configuration
- ☐ ORM overhead quantification
- ☐ Language performance comparison

Phase 10: Analysis - Phase B (Comparison)

- ☐ Head-to-head performance comparison
- ☐ Parametric workload analysis
- ☐ Crossover point identification
- ☐ Curation capability assessment

- ☐ Schema evolution assessment
- ☐ Data integration fit assessment

Phase 11: Analysis - Phase C (Decision)

- ☐ Apply “Is Graph Fast Enough?” criteria
- ☐ Total Cost of Ownership calculation
- ☐ Architecture recommendation

Phase 12: Mitigation (Conditional)

- ☐ Implement mitigation if needed (D1-D4)
- ☐ Validate hybrid architecture if required

Phase 13: Final Report

- ☐ Architecture decision documentation
- ☐ Recommendations report
- ☐ Infrastructure as Code finalization

Deliverables

1. Working Implementations

- Python, Go, Java, Rust implementations
- Both raw drivers and ORMs
- Neo4j, Memgraph, and PostgreSQL
- Air and Maritime domains

2. Performance Validation

- Pass/fail against “fast enough” thresholds
- Parametric analysis across workload mixes
- Crossover point identification
- ORM overhead quantification

3. Curation Assessment

- Capability comparison matrix
- Schema evolution evaluation
- Visualization quality assessment
- Curator workflow evaluation

4. **Cost Analysis**

   - 3-year TCO for each architecture option
   - Development cost comparison
   - Licensing analysis (with Community Edition options)

5. **Architecture Recommendation**

   - Primary recommendation with rationale
   - Contingency options if primary doesn't meet requirements
   - Decision matrix with weighted criteria

6. **Infrastructure as Code**

   - Docker Compose for local development
   - Terraform modules for AWS (if cloud deployment needed)
   - Monitoring dashboards (Prometheus + Grafana)

Appendix A: Identifier Formats

Air Domain

Mode_S (Mode Select) - 6 hexadecimal characters - Example: `A1B2C3`, `4CA123`

Tail Number - Alphanumeric, country-specific format - Example: `N12345` (US), `G-ABCD` (UK)

Maritime Domain

MMSI (Maritime Mobile Service Identity) - 9 digits - First 3 digits = MID (country code) - Example: 366123456 (US), 235987654 (UK)

SCONUM (Ship Control Number) - Military/NATO identifier - Alphanumeric - Example: `FFG-62`, `DDG-1001`

Appendix B: PostgreSQL Schema (Reference)

```sql
-- Denormalized lookup table
CREATE TABLE air_instance_lookup (
    id BIGSERIAL PRIMARY KEY,
    mode_s VARCHAR(8),
    tail_number VARCHAR(20),
    shark_name VARCHAR(255) NOT NULL,
    platform VARCHAR(255),
    affiliation VARCHAR(100),
    nationality VARCHAR(100),
```

```
    operator VARCHAR(255),
    air_type VARCHAR(255),
    air_model VARCHAR(255),
    neo4j_node_id BIGINT
);

-- Covering index for zero-heap-fetch lookups
CREATE INDEX idx_air_lookup_mode_s_covering
ON air_instance_lookup(mode_s)
INCLUDE (shark_name, platform, affiliation, nationality, operator)
WHERE mode_s IS NOT NULL;

-- Maritime lookup table
CREATE TABLE ship_instance_lookup (
    id BIGSERIAL PRIMARY KEY,
    mmsi VARCHAR(9),
    sconum VARCHAR(20),
    shark_name VARCHAR(255) NOT NULL,
    platform VARCHAR(255),
    affiliation VARCHAR(100),
    nationality VARCHAR(100),
    operator VARCHAR(255),
    ship_type VARCHAR(255),
    ship_class VARCHAR(255),
    neo4j_node_id BIGINT
);

-- Activity log (for knowledge generation)
CREATE TABLE track_activity_log (
    id BIGSERIAL PRIMARY KEY,
    track_id VARCHAR(50) NOT NULL,
    event_type VARCHAR(50) NOT NULL,
    kb_object_id BIGINT,
    mode_s VARCHAR(8),
    mmsi VARCHAR(9),
    timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    properties JSONB
);
```

Appendix C: Glossary

Term	Definition
Cypher	Neo4j's query language for graph databases
ORM	Object-Relational Mapper - abstracts database access
OGM	Object-Graph Mapper - ORM equivalent for graph databases

Term	Definition
CDC	Change Data Capture - tracking database changes in real-time
CTE	Common Table Expression - SQL feature for recursive queries
p99	99th percentile - 99% of requests are faster than this
QPS	Queries Per Second - throughput measure
TTL	Time To Live - cache expiration time
Bolt	Neo4j's binary protocol (also used by Memgraph)
APOC	"Awesome Procedures on Cypher" - Neo4j extension library

---

Document History

Version	Date	Author	Changes
1.0	2026-01-02	Planning Session	Initial comprehensive plan