



ekoDB White Paper

A Multi-Model Database System

By Sean M. Vazquez, Creator of ekoDB

Executive Summary

ekoDB is a ~50MB binary that replaces your document store, cache, search engine, vector database, server-side functions, and auth service. Built from scratch in Rust for memory safety and bare-metal performance, it combines in-memory speed with full durability guarantees through Write-Ahead Logging (WAL) — no garbage collector pauses, no null pointer crashes, no data races.

No separate Redis for sessions, no Elasticsearch for search, no Pinecone for vectors, no custom auth middleware. One binary, one API, one set of client libraries across six languages.

Performance that matters: ekoDB doesn't just match the databases it replaces — it outperforms them while doing more work per request:

What ekoDB Replaces	ekoDB	Competitor	Advantage
PostgreSQL (writes)	39K ops/sec	5K ops/sec	7.3x faster
MongoDB (writes)	39K ops/sec	11K ops/sec	3.7x faster
PostgreSQL (reads)	124K ops/sec	107K ops/sec	15% faster, ~2x less CPU
Redis (writes)	39K ops/sec	6K ops/sec	6.5x faster
MySQL (writes)	39K ops/sec	2.5K ops/sec	15.7x faster
Redis (features)	Auth, encryption, search built-in	None included	Full stack in one binary

YCSB benchmarks: 1M records, 64 threads, full durability, 6 databases tested. Every ekoDB request includes JWT auth, AES-GCM encryption, full-text indexing, and vector indexing — work that competitors skip entirely.

Use less, get more: ekoDB achieves 2-5x better CPU efficiency than PostgreSQL, MongoDB, MySQL, and Redis across all workloads. That means smaller cloud instances, lower monthly bills, and more headroom for your application — not your database.

Scale Your Way

Run one ekoDB for your entire application, or run many. Connect multiple ekoDB instances via **Ripple** for cross-region replication, use **Functions** for server-side business logic — from simple queries to full workflow orchestration with AI, caching, and graph traversal — or orchestrate from your own application API. Start with a single node and scale horizontally as your needs grow — no re-architecture required.

Less to Learn, More to Build

One query language. One set of client libraries. All capabilities listed below accessible through the same API. Your team learns one database instead of five.

Key Capabilities:

- **Multi-Model Architecture:** Document storage, key-value, full-text search, vector search, real-time subscriptions, and server-side functions in a single system
- **Proven Performance:** 37-127K ops/sec with full durability, leading every YCSB workload against 4 competitors, 2-5x better CPU efficiency
- **Configurable Durability:** Non-durable mode for high throughput or durable mode for guaranteed persistence
- **Adaptive Scaling:** Deployments from IoT devices to enterprise servers
- **Secure by Default:** HTTPS/WSS-only, AES-GCM encryption at rest, TLS/SSL in transit
- **Adaptive Memory:** 1%-80% of available RAM with automatic management
- **Distributed:** Ripple system for horizontal scaling and cross-database propagation
- **Server-Side Logic:** Composable Functions for business logic, AI workflows, SWR caching, graph traversal, and external API calls — executed server-side in a single request
- **Memory-Safe Foundation:** Built entirely in Rust — no garbage collector pauses, guaranteed memory safety

1. Introduction

1.1 The Problem

Applications often require multiple database systems to handle different workloads:

- MongoDB for document storage
- Redis for caching and real-time data
- Elasticsearch for full-text search
- Pinecone for vector search
- PostgreSQL for relational data

This multi-database approach introduces complexity, operational overhead, and integration challenges.

1.2 The Solution

ekoDB consolidates the capabilities described in the [Executive Summary](#) into a single Rust binary — managed through a unified API with client libraries across six languages.

1.3 History

ekoDB originated from a practical challenge: integrating multiple databases required complex layers of abstraction to achieve feature parity and consistent usage patterns.

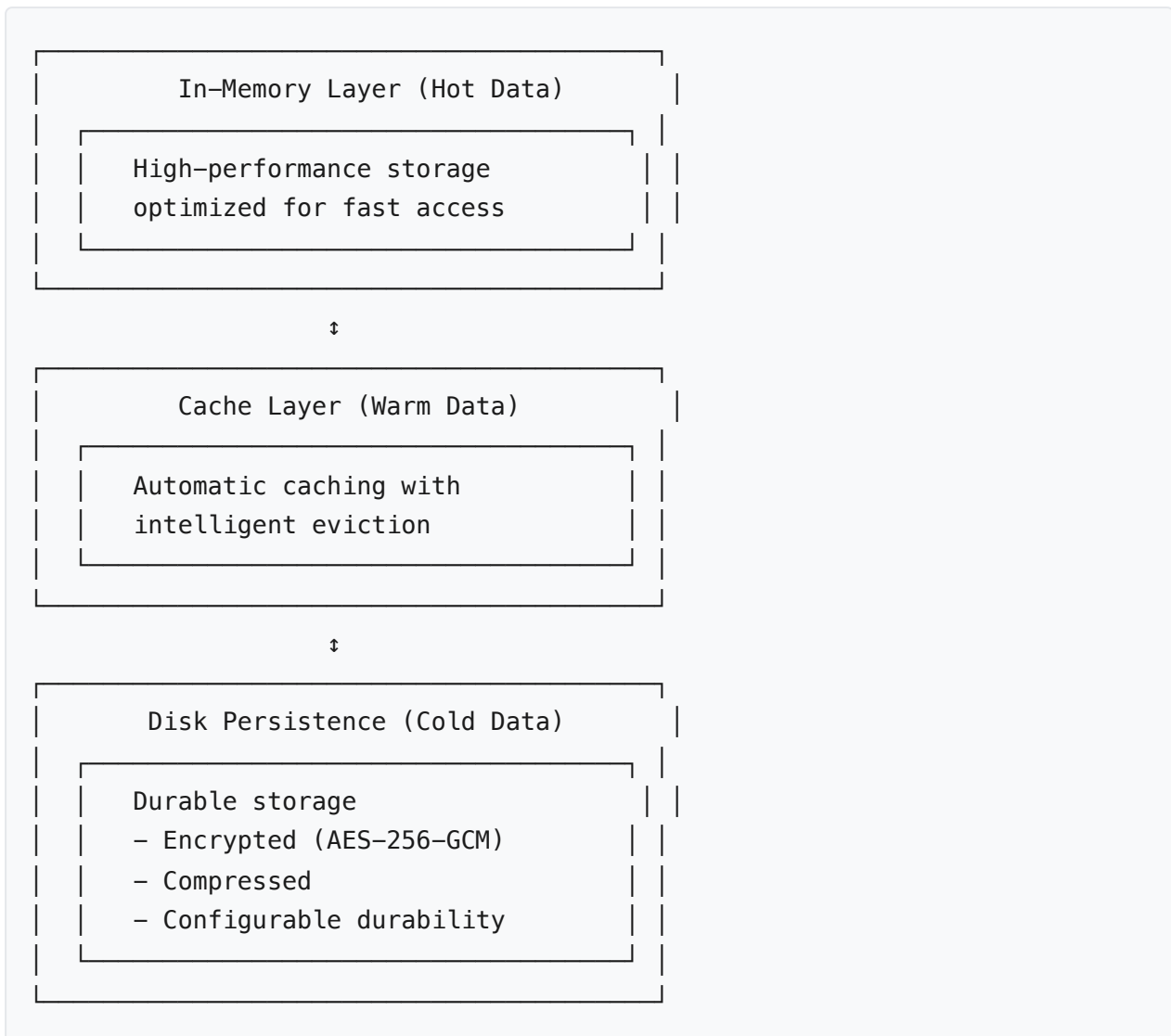
Development Timeline:

- **2013:** Initial development as SOLO (Single Object Language Operator), an API gateway for multi-database integration
 - **2013-2022:** SOLO operated as an API gateway connecting various database systems
 - **2022:** Decision to eliminate the abstraction layer and build a unified database
 - **2022-2025:** Complete from-scratch rewrite as ekoDB
 - **Current:** Active development and production use
-

2. Architecture

2.1 Storage Architecture

ekoDB uses a **hybrid in-memory architecture** with disk persistence:



Key Features:

- **Larger-than-Memory Support:** Datasets can exceed available RAM
- **Smart Caching:** Frequently accessed data stays in memory for fast access
- **Automatic Eviction:** Intelligent cache eviction when memory limits are reached

2.2 Data Model

ekoDB supports multiple data models in a unified system:

Document Model

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "tags": ["developer", "rust"],
  "address": {
    "city": "San Francisco",
    "country": "USA"
  }
}
```

Key-Value Model

```
{
  "session:user123": {
    "token": "abc123",
    "expires": "2025-10-15T00:00:00Z"
  }
}
```

Vector Model

```
{
  "content": "ekoDB is a high-performance database",
  "embedding": [0.1, 0.2, 0.3, ...], // 384-dimensional vector
  "metadata": {
    "source": "documentation"
  }
}
```

2.3 Type System

ekoDB provides a flexible type system with per-collection per-field type enforcement:

Note:

- Type enforcement occurs at the time of write and is not enforced at the time of read.
- Type enforcement is optional at key-value level vs required at document level.
- All types are dynamically inferred at write time or can be explicitly specified via `/schemas` endpoint.

Supported Types:

ekoDB supports 16 comprehensive data types organized into four categories:

Basic Types

- **String** - UTF-8 encoded text data for names, descriptions, and general text content
- **Integer** - 64-bit signed integers (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- **Float** - 64-bit IEEE 754 floating-point numbers for decimal values
- **Boolean** - Binary true/false values for logical operations

Advanced Numeric Types

- **Number** - Flexible numeric type that automatically handles both integers and floats, inferred at write time
- **Decimal** - Arbitrary-precision decimal numbers that avoid floating-point rounding errors. Essential for financial calculations (e.g., currency, accounting), scientific computations requiring exact decimal representation, and any scenario where $0.1 + 0.2$ must equal exactly 0.3 . Unlike **Float**, **Decimal** uses precision-preserving internal storage, ensuring mathematical precision without binary floating-point approximation issues

Temporal Types

- **DateTime** - RFC 3339 formatted date-time values with timezone support (e.g., `2024-01-01T00:00:00Z`)
- **Duration** - Time duration values for representing time spans (e.g., `30s`, `5m`, `2h`)

Collection Types

- **Array** - Ordered lists of heterogeneous elements, preserving insertion order
- **Set** - Unordered collections of unique values with automatic deduplication
- **Vector** - Fixed-dimension numeric arrays optimized for embeddings and vector similarity search
- **Object** - Nested documents/maps with key-value pairs for complex structured data

Specialized Types

- **UUID** - Universally unique identifiers (RFC 4122) for globally unique record identification
- **Binary** - Base64-encoded binary data for images, files, and other binary content
- **Bytes** - Raw byte arrays for unencoded binary data storage
- **Null** - Explicit null/empty values for optional fields

Response Formats:

- **Typed:** Includes type metadata (e.g., `{"type": "String", "value": "text"}`)
- **Non-Typed:** Traditional NoSQL format (e.g., `"text"`)

2.4 Configuration Options

ekoDB provides flexible configuration options to tune behavior for different use cases:

Response Format Configuration

- **Typed Responses:** Include type metadata for strong typing
- **Non-Typed Responses:** Traditional NoSQL format for simplicity
- **Default:** Typed responses (includes type metadata)
- **Configuration:** Configure via configuration API or ekoDB App (<https://app.ekodb.io>)

Durability Configuration

- **Durable** (`durable_operations: true`): Guaranteed persistence, every write confirmed to disk
- **Non-Durable** (`durable_operations: false`): Higher throughput, async persistence

Storage Mode Configuration

ekoDB provides three storage modes optimized for different workloads:

Mode	Description	Best For
Fast	In-memory with WAL durability	Maximum throughput, general workloads
Balanced	In-memory with periodic disk persistence	General purpose, mixed workloads
Cold	Disk-optimized append-only storage	Write-heavy, archival, time-series data

Fast Mode (`storage_mode: "fast"`):

- Fastest write and read performance
- Data recoverable from WAL on restart
- Ideal for caches, sessions, and high-throughput ingestion

Balanced Mode (`storage_mode: "balanced"`):

- Near fast-mode performance with eventual disk persistence
- Good balance of speed and storage efficiency

Cold Mode (`storage_mode: "cold"`):

- Optimized for write-heavy and append-only workloads
- Efficient disk space usage
- Ideal for IoT, logs, time-series, and archival data

Configuration Example:

```
{  
  "storage_mode": "balanced",  
  "durable_operations": true  
}
```

Storage mode can be changed at runtime via the configuration API. Each mode works with both durability settings (durable or async).

3. ACID Compliance

ekoDB is **fully ACID-compliant**, providing the same data guarantees as traditional relational databases while maintaining NoSQL-level performance.

What is ACID?

ACID stands for Atomicity, Consistency, Isolation, and Durability - the four properties that guarantee reliable database transactions:

- **Atomicity:** Operations either complete fully or not at all. If any part of a transaction fails, the entire transaction is rolled back.

- **Consistency:** Data always moves from one valid state to another. Schema constraints and validation rules are enforced.
- **Isolation:** Concurrent operations don't interfere with each other. Multiple users can work simultaneously without conflicts.
- **Durability:** Once committed, data persists even if the system crashes. Write-Ahead Logging (WAL) ensures no data loss.

How ekoDB Implements ACID

Atomicity: Every operation either completes fully or not at all. If a failure occurs, the database automatically recovers to ensure all-or-nothing semantics.

Consistency via Schema Constraints: ekoDB supports 16 field types and 7 constraint types (required, unique, min/max, enum, regex, default, null handling) to maintain data integrity.

Isolation via Concurrency Control: ekoDB's concurrency control ensures that concurrent operations don't create race conditions or data corruption.

Durability via Configurable Persistence: Write-Ahead Logging with configurable durability modes (see [Section 2.4](#)).

Transactions

ekoDB supports multi-document transactions with:

- Multiple isolation levels (ReadUncommitted, ReadCommitted, RepeatableRead, Serializable)
- Savepoints for nested transactions
- Automatic rollback on errors
- Full audit trail for all operations

For detailed transaction usage, see [Transactions Documentation](#).

4. Performance

4.1 Write Performance

Write throughput depends on the [durability mode](#) selected. In durable mode, ekoDB leads all tested competitors on YCSB workloads with equivalent persistence guarantees. In non-

durable mode, throughput increases further for workloads where crash recovery is not critical.

Dual-Node Strategy: Deploy a primary node in non-durable mode and a secondary node in durable mode (via Ripple replication) to achieve both high performance and durability.

4.2 Read Performance

- **Indexed Lookups:** Fast hash and B-tree index lookups
- **Point Queries:** Sub-millisecond latency
- **Range Queries:** Efficient range scans via B-tree indexes
- **Full-Text Search:** Fast term-based retrieval
- **Vector Search:** Efficient nearest neighbor search

4.3 Memory Efficiency

- **Base Memory:** Low memory footprint optimized for efficiency
 - **Compression:** Significant space savings with configurable compression
 - **Adaptive Allocation:** 1%-80% of available RAM
 - **Automatic Eviction:** Intelligent memory management
-

5. Indexing

5.1 Index Types

ekoDB implements multiple index types optimized for different query patterns:

Hash Indexes (Default)

- **Use Case:** `WHERE field = value` — fast equality lookups
- **Automatic:** Created for frequently queried fields

B-Tree Indexes

- **Use Case:** `WHERE field > value`, sorting, range scans
- **Features:** Supports `<`, `>`, `<=`, `>=`, `BETWEEN`

Inverted Indexes

- **Use Case:** Full-text search
- **Features:** Stemming, fuzzy matching, tokenization

Vector Indexes

- **Use Case:** Semantic similarity search, embeddings, AI/ML workloads
- **Metrics:** Cosine similarity, Euclidean distance, dot product
- **Search Modes:** Approximate nearest neighbor and exact search

5.2 Index Management

- **Automatic Creation:** Indexes created based on query patterns
 - **Automatic Maintenance:** Updated on insert/update/delete
 - **Concurrent Access:** Thread-safe operations
 - **Memory Efficient:** Automatic memory management and eviction
-

6. Concurrency & Isolation

6.1 Concurrency Control

ekoDB implements collection-level concurrency control:

- **Collection-Level Granularity:** Concurrent access managed per collection
- **Concurrent Reads:** Multiple readers can access data simultaneously
- **Write Coordination:** Ensures data consistency during modifications
- **Cross-Collection Independence:** Operations on different collections don't block each other

6.2 Isolation Levels

Within a Single Collection: Serializable

- No dirty reads, non-repeatable reads, or phantom reads
- Full serializable isolation guarantees

Across Multiple Collections: Read Uncommitted (effectively)

- No coordination between collections
- Applications must handle cross-collection consistency at the application level

7. Durability & Recovery

7.1 Write-Ahead Logging (WAL)

All writes are recorded to a Write-Ahead Log before execution. The [durability_mode](#) controls whether each WAL entry is fsynced before acknowledging the client (durable) or batched asynchronously (non-durable).

WAL Management:

- Automatic rotation and compaction
- Manual rotation available
- Automatic cleanup after replication

7.2 Recovery Process

ekoDB performs automatic crash recovery. The system validates data integrity and restores the database to its last consistent state. Recovery time depends on data volume and system resources.

8. Search Capabilities

Built on the [index types described in Section 5](#), ekoDB exposes three search modes through a unified query API:

8.1 Full-Text Search

- **Field Weighting:** Prioritize specific fields in relevance scoring
- **Minimum Score:** Filter results by relevance threshold
- **Fuzzy Matching:** Typo tolerance via Levenshtein distance

8.2 Vector Search

- **Embedding Support:** 384, 768, 1536 dimensions (configurable)
- **Metadata Filtering:** Combine vector similarity with field filters
- **Top-K Results:** Efficient nearest-neighbor selection

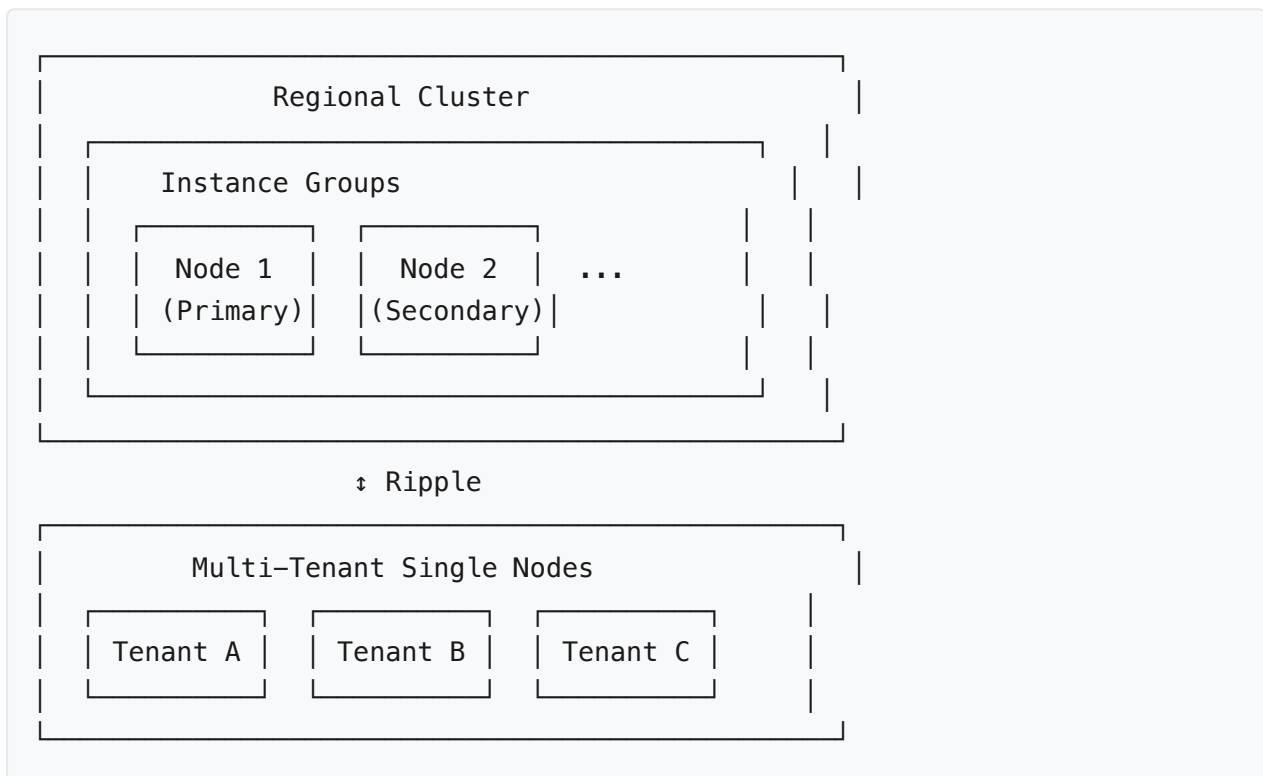
8.3 Hybrid Search

Combine full-text and vector search in a single query — keyword matching for precision, semantic similarity for recall, with unified scoring and ranking.

9. Distributed Architecture

9.1 Ripple System

ekoDB's distributed architecture uses the **Ripple** system for real-time data propagation and horizontal scaling:



Core Features:

- **Real-Time Propagation:** Operations replicate immediately as they occur
- **Cross-Database Propagation:** Replicate data across ekoDB instances
- **Horizontal Scaling:** Add nodes for increased read/write capacity
- **Regional Distribution:** Deploy across geographic regions for low-latency access
- **Automatic Failover:** Secondary nodes take over on primary failure
- **Loop Prevention:** Automatic deduplication prevents infinite operation loops in multi-node deployments

- **Configurable Modes:** Send-only, receive-only, bidirectional, or isolated nodes

Real-World Use Cases:

Geographic Distribution (Multi-Region) Deploy primary nodes in US, EU, and Asia with cross-region ripples. Users read from their nearest region for low latency while writes propagate globally for consistency.

Read Scaling (Analytics) Configure a primary write node in durable mode sending ripples to multiple read replicas in non-durable mode. Production traffic doesn't slow down while replicas handle heavy reporting queries.

High Availability (Active-Active) Two primary nodes in bidirectional mode accept writes simultaneously. If one fails, the other continues operations without manual intervention.

Data Pipelines (IoT) Dedicated ingestion nodes in non-durable mode propagate to storage nodes in durable mode and processing nodes for real-time analytics. Each layer scales independently.

Development/Staging Production primary sends ripples to staging replica for realistic testing. Test writes in staging don't propagate back to production.

See [Ripples - Data Propagation](#) documentation for comprehensive architecture guides and configuration examples

9.2 Replication

- **Asynchronous Replication:** Non-blocking writes
- **Configurable Targets:** Replicate to multiple destinations
- **Selective Replication:** Choose which collections to replicate
- **Conflict Resolution:** Last-write-wins strategy

10. Functions

ekoDB's **Functions** system is a server-side execution engine for composable business logic. A single Function call can query data, transform results, invoke AI models, and call external APIs — all in one request, with no additional network round-trips.

10.1 Composable Server-Side Logic

Functions are JSON-defined, parameterized, versioned, and callable from any client library or the REST API. Each Function contains one or more operations (Query, Insert, Update, Chat, Embed, HttpRequest, and more) that execute sequentially with results flowing between steps. Functions compose freely: a Function can call other Functions via `CallFunction`, enabling modular workflows without duplicating logic across clients.

Built-in conditional execution (`If`), iteration (`ForEach`), and error handling give Functions the control flow needed for real business logic — not just simple queries.

10.2 Caching & Stale-While-Revalidate

Functions integrate naturally with the **stale-while-revalidate (SWR)** pattern. Cached Function results are served instantly while fresh data is computed in the background. This turns expensive multi-step operations — dashboard summaries, analytics aggregations, enriched user profiles — into instant responses with automatic freshness. For workloads that don't change on every request, SWR eliminates perceived latency entirely.

10.3 Graph Traversal with Vectors

By combining **vector search** with **conditional logic** and **Function composition**, you can build graph-like traversals entirely within ekoDB. Use vector similarity to discover related nodes, chain Functions to walk relationships, and apply conditional branching to shape the traversal — all server-side, in a single call. This enables recommendation engines, knowledge graphs, and relationship discovery without a separate graph database.

10.4 AI Integration

Functions natively support **Chat** (LLM completions) and **Embed** (vector generation) operations alongside standard database operations. Build complete RAG pipelines, content moderation workflows, automated summarization, or any AI-augmented logic — without leaving the database layer or adding external orchestration.

For detailed architecture, operation types, and code examples, see [Functions Architecture](#).

11. Security

11.1 Network Security

HTTPS/WSS Only: Unlike traditional databases that use direct TCP connections, ekoDB exclusively uses HTTPS and WSS (Secure WebSocket) protocols.

Benefits:

- No direct TCP exposure
- Built-in encryption (TLS/SSL)
- Standard web protocols
- Firewall friendly (port 443)
- Certificate-based security
- Protection against man-in-the-middle attacks

11.2 Encryption

At Rest:

- AES-256-GCM encryption for all stored data
- Volume-level encryption provided by ekoDB managed infrastructure

In Transit:

- TLS/SSL for all network communication
- Certificate validation
- Perfect forward secrecy

11.3 Authentication

- **JWT Tokens:** Industry-standard authentication
 - **API Keys:** Simple authentication for services
 - **Role-Based Access:** Collection-level and field-level permissions
 - **Token Expiration:** Configurable TTL
-

12. Use Cases

12.1 AI Agents & Workflows

- **Vector Search:** Store and query embeddings
- **Chat History:** Built-in chat session management
- **Context Management:** Efficient retrieval of relevant context

- **Real-Time Updates:** WebSocket for live agent interactions

12.2 Real-Time Analytics

- **High Throughput:** High-speed ingestion for real-time analytics
- **In-Memory Processing:** Sub-millisecond queries
- **Time-Series Data:** Efficient storage and retrieval
- **Aggregations:** Fast analytical queries

12.3 IoT Data Processing

- **Adaptive Scaling:** Runs on resource-constrained devices
- **Edge Computing:** Deploy close to data sources
- **Efficient Storage:** Compression reduces disk usage
- **Batch Operations:** Handle bursts of sensor data

12.4 Session Management

- **TTL Support:** Automatic session expiration
- **Fast Lookups:** $O(1)$ key-value operations
- **High Concurrency:** Handle thousands of sessions
- **Persistence:** Optional durability for sessions

12.5 Content Delivery

- **Distributed Caching:** Ripple for multi-region deployment
- **Fast Reads:** In-memory performance
- **Compression:** Reduce bandwidth usage
- **Real-Time Updates:** WebSocket for live content

13. Getting Started

13.1 Quick Start

```
# Install client library
npm install @ekodb/ekodb-client
```

```
# Connect to ekoDB

const client = new EkoDBClient({
  baseUrl: "https://your-subdomain.ekodb.net",
  apiKey: "your-api-key"
});

await client.init();

// Insert a document
await client.insert("users", {
  name: "John Doe",
  email: "john@example.com"
});

// Query documents using ekoDB query builder
const query = {
  filter: {
    type: "Condition",
    content: {
      field: "age",
      operator: "Gt",
      value: 25
    }
  }
};
const users = await client.find("users", query);
```

13.2 Resources

- **Homepage:** <https://ekodb.io>
- **Documentation:** <https://docs.ekodb.io>
- **Management Console:** <https://app.ekodb.io>
- **Support:** support@ekodb.io

14. Architecture Deep Dives

For detailed technical documentation on specific subsystems:

- [Functions Architecture](#) - Server-side execution, operation types, workflows, and composition patterns
 - [Transactions Architecture](#) - ACID compliance, isolation levels, savepoints, and rollback mechanisms
 - [Advanced Operations](#) - Client library usage, search capabilities, chat integration, and real-time features
-

15. About the Author

Sean M. Vazquez is the creator and lead developer of ekoDB. His journey in technology began at age 10, writing HTML, CSS, and JavaScript on an old Compaq computer. Those early years were filled with screen-by-screen adventure games and tinkering with 8-bit and 16-bit consoles (Game Boy, SNES, Sega Genesis), sparking a lifelong passion for building software and the endless wonder of console modding. To this day, he still treasures his N64, playing Mario Kart, Ocarina of Time, 007, and Super Smash Bros with game mods.

As he grew, Sean expanded into C++ and JavaScript, and discovered Linux, a discovery that would shape his approach to systems engineering. He enrolled at Stevens Institute of Technology as a mechanical engineer, but the pull of code was too strong. Remembering the joy of those early programming days, he switched to computer science.

Career Progression:

- **Stevens Institute of Technology:** B.S. in Computer Science, 2013
- **Thomson Reuters:** QA Engineer Intern, working on C# tax software where he learned the importance of quality and testing at scale
- **UBS:** eLearning software implementation and automation, building systems that reached thousands of employees
- **TMP Worldwide:** Full Stack Engineer focused on front-end development for enterprise recruitment platforms
- **Metacake:** Full Stack Engineer driving growth engineering and building products from scratch for contract clients
- **American Express:** Grew from Full Stack Engineer to Senior Engineering Manager and Lead Software Engineer/Architect, leading both product development and internal tooling engineering with a focus on automation

The Origin of ekoDB:

ekoDB began in 2013 as SOLO (Single Object Language Operator), an ambitious vision to create "one API gateway to rule them all." But Sean's vision went further: why just abstract databases when you could make all data universally accessible? The goal became empowering engineers to build their best backends without being constrained by database choices. After nearly a decade as an API gateway, SOLO was rewritten from scratch as ekoDB, a unified database platform that eliminates the complexity of multi-database architectures.

Today, Sean works from a setup that reflects his roots: a Filco tenkeyless keyboard, Raspberry Pis running Arch Linux alongside Debian and Ubuntu, and macOS for daily work. He still has a soft spot for the iPod.

Contact: sean@ekodb.io

Features and specifications are subject to change as ekoDB continues to evolve.