Healthcare Monitoring System

Instructor: Hani salah
Name: Momen Mohammed Bhais
St. Number: 221073
Course: NoSQL
Date: May 2025

Scenario and Problem Statement:

Scenario Overview:
Modern healthcare systems deal with diverse types of data: structured (like medical records), semi-structured (like sensor data), and complex relationships (like doctor-patient connections). Traditional databases fail to efficiently handle this variety.

NoSQL

Proposed Solution:

Multi-Database Architecture for Healthcare Monitoring

- MongoDB Atlas (Document Database): Stores detailed patient files and their geographical location
- InfluxDB Cloud (Time-Series Database): Real-time monitoring of vital signs like heart rate and blood pressure
- Neo4j Aura (Graph Database): Models and queries complex relationships between doctors, patients, and treatments

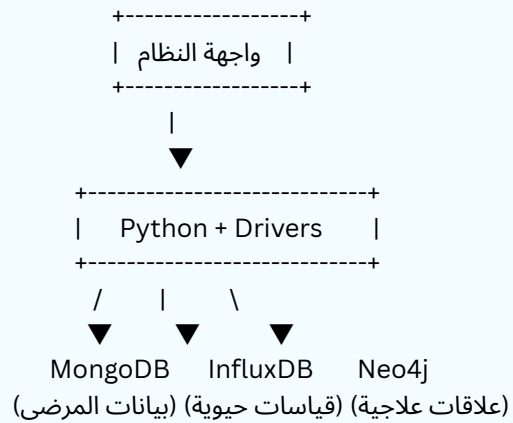
System Goals:

- High availability via reliable cloud services
- Fault tolerance and automatic recovery
- Scalable to support more patients and devices
- Balanced consistency, availability, and performance based on CAP principle

NoSQL

Architecture Diagram:

- Python uses official libraries:
  - pymongo for MongoDB
  - influxdb-client for InfluxDB
  - neo4j Driver for Neo4j
- Data flow:
  - MongoDB: medical records
  - InfluxDB: vital device data
  - Neo4j: relationships
- All hosted on managed cloud platforms to ensure redundancy, availability, and performance flexibility



NoSQL

Justification for Database Choices:

- MongoDB

A flexible NoSQL database that supports both structured and unstructured medical data (e.g., radiology images, medical history, clinical notes). It ensures strong consistency with regional replication via MongoDB Atlas, enhancing system stability and responsiveness locally and globally — a critical feature for sensitive healthcare applications.

- InfluxDB

Specifically designed to handle time-series data such as real-time vital signs. It offers high write performance and automatic time-based partitioning, enabling instant analytics and event-driven alerts — essential in health monitoring systems and emergency response scenarios.

- Neo4j

A graph database specialized in identifying complex patterns and relationships. It provides a robust graph structure that supports causal analysis and intricate medical relationships (e.g., comorbidities, drug interactions, genetic factors). Through Aura, it ensures high-availability global replication, making it ideal for outbreak tracking and healthcare crisis management.

NoSQL

Detailed Mechanisms to Achieve System Goals:

- High Availability: Using DBaaS like Atlas, Cloud, and Aura, which replicate data across availability zones automatically

- Fault Tolerance:
  - MongoDB Atlas: Uses Replica Sets (minimum of 3 nodes). If the primary fails, a secondary is elected as new primary. Data is synced in real-time.
  - InfluxDB Cloud: Distributes data across nodes, manages copies internally based on time windows. Redirects traffic to other nodes automatically.
  - Neo4j Aura: Graph replication allows reading from multiple nodes and causal-consistent writing.
- Scalability: All services leverage cloud-native architectures that allow seamless horizontal and vertical scaling.
- CAP Balance: Trade-offs made based on data type: favor availability over strict consistency (InfluxDB), or favor strong consistency (MongoDB).

Data Models:

MongoDB (Documents):

```
{
  "name": "Ahmed Mohammed",
  "age": 35,
  "region": "South",
  "medical_history": ["Diabetes", "High Blood Pressure"]
}
```

InfluxDB (Time-Series):

measurement: health_metrics
tags: patient = Ahmed Mohammed
fields: heart_rate = 72, blood_pressure = "120/80"

Neo4j (Graph):

(Dr. Sarah)-[:TREATS]->(Ahmed Mohammed)

- **Sharding:**

- **MongoDB:**

Sharded by region

Atlas automatically distributes and balances shards based on patient location

- **InfluxDB:**

Uses time-based sharding

Allows querying recent data without scanning old entries

- **Neo4j:**

No traditional sharding, but uses logical graph distribution

Aura distributes nodes and relationships across servers while ensuring fast relationship queries

Replication and High Availability:

- **MongoDB Atlas:** Uses Replica Sets with at least 2 replicas. Secondary nodes serve reads. New primary is elected within 5 seconds if the main node fails.
- **InfluxDB Cloud:** Uses distributed storage with automatic replication based on time capacity. Data is partitioned and copied within the cloud service.
- **Neo4j Aura:** Provides replicated graph instances across regions. Uses Paxos algorithms to ensure consistency even during failures.

Result: System continues without interruption using fallback nodes automatically.

Consistency Strategy in Databases:

- **MongoDB:**

Confirms writes and reads from multiple nodes to ensure always up-to-date data — critical for medical systems.

- **InfluxDB:**

Writes data instantly, then replicates it shortly after — enables fast performance with reliable analytics.

- **Neo4j:**

Maintains the order of related events — perfect for understanding complex medical relationships in context.

Queries and Demonstrations:

MongoDB:

```
db.patients.insertOne({"name": "Khaled", "region": "North"})
db.patients.updateOne({"name": "Khaled"}, {"$set": {"region": "Central"}})
```

InfluxDB:

```
from(bucket:"health")
  |> range(start: -1h)
  |> filter(fn: (r) => r["_measurement"] == "health_metrics")
```

Neo4j:

```
MATCH (d:Doctor)-[:TREATS]->(p:Patient)
RETURN d.name, p.name
```

Performance Optimization Strategy:

- **MongoDB – Regional Indexing:**

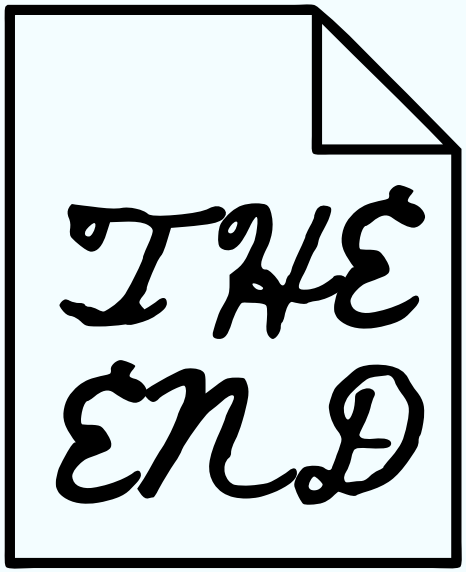
Uses B-Tree indexing to minimize geo-based query time — ideal for fast patient record access by location.

- **InfluxDB – Time Window Filtering:**

Uses Flux language to filter data by time range — drastically reduces query load in real-time monitoring.

- **Neo4j – Node Name Indexing:**

Automatically indexes node names — boosts relationship lookups across large medical graphs.



NoSQL