Architecture & ETL Data Flow (Redis → MongoDB)

Architecture Overview:

Data Sources: IoT sensors / APIs generate readings.

ETL Flow:

- **Extract:** Sensor data is captured via Node.js server.
- **Transform:** Data formatted into JSON with fields – sensorId, reading, unit, updatedAt, meta.author.
- **Load:**

1. **Redis Cache Layer:** Stores frequently accessed readings for fast lookup.
2. **MongoDB Atlas:** Acts as the data lake for long-term storage and analytics.

Integration:

- Cache-aside strategy used — data fetched from Redis; if missing, pulled from MongoDB and cached.
- ETL/Cache refresh occurs periodically or on data updates.

Lessons Learned

Improved Performance: Redis dramatically reduced read latency for frequent queries.
Scalable Design: Decoupling cache and database improved throughput under load.
Hands-on with ETL: Gained practical experience building Extract–Transform–Load pipelines between in-memory and persistent layers.
Monitoring Importance: Learned to measure cache hit/miss ratio and TTL performance.
Data Consistency: Understood trade-offs between immediate consistency vs eventual sync in cache refresh cycles.

Challenges & Observations

What Worked Well:

- Redis integration using Node.js client was seamless.
- TTL-based expiration ensured cache freshness.
- MongoDB Compass helped validate stored readings and metadata

What Did Not Work Well:

- Handling cache invalidation when updates occurred simultaneously.
- Difficulty managing Redis memory limits (eviction policies).
- Maintaining consistent timestamps between Redis and MongoDB.

Next Steps / Improvements:

- Automate cache invalidation triggers via pub/sub.
- Implement Redis Cluster for scaling.
- Add monitoring dashboard for ETL performance metrics.