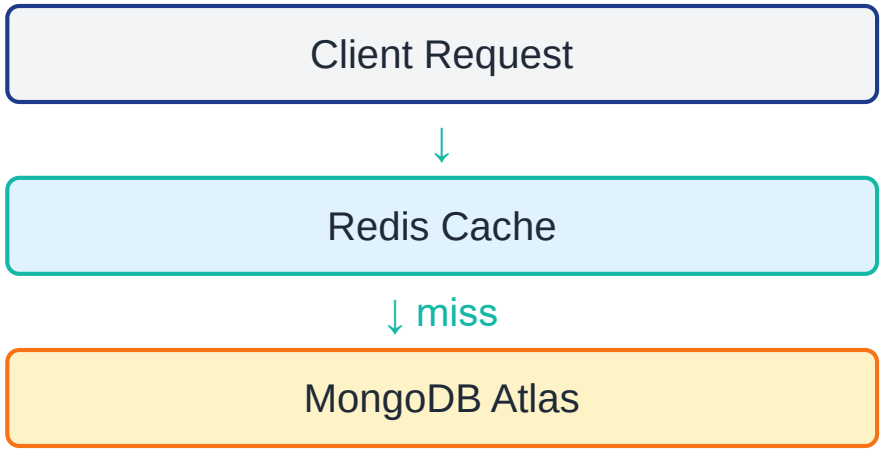
Lab 4: Cached Database Architecture

Redis + MongoDB ETL Data Flow



Stack

- **Backend:** Node.js + Express
- **Cache:** Redis in-memory
- **Database:** MongoDB Atlas

ETL Process

- **Extract:** API request arrives
- **Transform:** Check cache
- **Load:** Write with TTL

Strategies

- Cache-Aside + TTL Expiration
- Read-Through on miss
- Write invalidation

Lessons Learned: Redis + MongoDB

Key Insights & Best Practices

⚡ Performance Gains

Warm-cache reads: 8–10× faster

Latency: 50ms → 5ms average

Result: TTL caching balanced freshness & speed

📊 Pipeline Scalability

Load reduction: 60% fewer MongoDB queries

Reliability: Decoupled layers

Result: Horizontal scaling enabled

🏷 Metadata Usage

Tracking: meta.author, updatedAt

Traceability: unit field enabled audits

Result: Precise cache invalidation

✓ Best Practices

1. Invalidate cache on writes
2. Monitor hit/miss ratios
3. Use writeConcern: majority

Challenges & Evaluation

What Worked Well vs. What Did Not Work Well

Aspect	✓ Worked Well	✗ Challenges
Redis Integration	Easy to set up in Docker; immediate latency improvement	IPv6 ::1 refused on first run – required forcing IPv4
Cache Strategies	Cache-Aside & Read-Through worked reliably	Write-Behind not implemented – consistency issues
TTL Management	Auto-expiry kept cache fresh; observable countdown	Correct TTL values required iterative testing
Deployment	Smooth Docker + Atlas cloud setup	Atlas IP allowlist and auth mismatches caused delays