Redis First Architecture with Binlog Denormalization and Active Failover

Architecture Overview
This design powers a ticket system with very fast reads from Redis, reliable writes to the database, and continuous synchronization through binlogs. It also includes proactive failover management to ensure no downtime.

Core Pattern

Read Path
API → Primary Redis → Instant Response (sub millisecond)
Write Path
API → Database → Binlog Detection → Data Enrichment → Redis Update
Failover Path
Health Monitor → Secondary Redis → Traffic Switch → Primary Recovery

Key Components and Design Choices

1. Redis Storage Strategy: Hash Structure

- **How it works**: Store each ticket in a Redis Hash with the ticket ID as the key.
- **Why Hash**
  - O(1) operations for ticket read or update
  - Memory efficient for large data
  - Atomic field level updates without pulling the whole object
  - Scales better than storing full JSON strings

- **Why not JSON string**
  - Updating requires fetching and rewriting the entire object
  - More CPU overhead for parsing

2. Data Denormalization with Binlog

- **How it works**: A binlog listener detects database changes, enriches ticket data, and pushes it to Redis.
- **Enriched Data Includes**
  - Ticket fields
  - Status name
  - Priority name
  - Assigned user name

    - Creator name
- **Why denormalize**
    - Removes runtime joins in API
    - Improves read speed by precomputing related fields
    - Enables instant API responses
- **Why not runtime joins**
    - Adds database load
    - Slower response times

3. Active Health Management

- **How it works**: A microservice checks Redis health and manages failover.
- **Steps**
    - Detect issues in Primary Redis
    - Migrate data to Secondary Redis
    - Switch traffic to Secondary
    - Recover and repopulate Primary
- **Why active management**
    - Prevents issues before a crash
    - Zero downtime
    - Failover can be triggered even on performance drop
    - Keeps Secondary fully in sync
- **Why not snapshots**
    - Only help after failure
    - Recovery takes downtime
    - Data loss possible between snapshot intervals

4. Dual Redis Setup

- **How it works**: One Redis acts as Primary for traffic, the other as Secondary for failover readiness.
- **Why dual Redis**
    - Enables failover without downtime
    - Allows maintenance on one while the other serves traffic
    - Supports performance testing safely
- **Why not single Redis**
    - Creates a single point of failure

5. Database as Source of Truth

- **How it works**: All writes go to the database first. Redis acts as a computed cache.
- **Why database first**
    - Guarantees durability and ACID compliance
    - Supports business rules and constraints
    - Provides audit history and backups

- **Why not Redis first**
    - Risk of data loss
    - No transaction safety

Data Flow

Normal Operations

1. API reads from Primary Redis with O(1) lookups
2. API writes go to Database
3. Binlog listener detects changes
4. Data is enriched and updated in both Primary and Secondary Redis

Failover Operations

1. Health Monitor detects Primary Redis issue
2. Confirms Secondary Redis has current data
3. Switches traffic to Secondary Redis
4. Rebuilds Primary Redis from Secondary
5. Switches traffic back to Primary

Performance Benefits

- Sub millisecond ticket retrieval
- Database read load reduced to near zero
- Consistent O(1) operations
- Fast user experience with instant ticket loading

Reliability Benefits

- Zero downtime during failover
- Consistent data through binlog sync
- Proactive monitoring prevents failures
- Recovery options from Database or Secondary Redis

Trade offs

- Higher Redis memory usage due to denormalized data
- More complex binlog processing logic
- Need for a second Redis and monitoring service
- More setup effort compared to a simple cache aside pattern

Why This Architecture

This approach is designed for applications where fast responses and reliability are more important than simple writes. By removing database joins from read operations and adding proactive failover, the system provides instant data access with no downtime. It balances speed, safety, and reliability in a way that supports enterprise scale operations.