MONOLITHIC:

Low latency.

ASYNCHRONOUS:

Best effort.

MESSAGE QUEUES:

CACHING:

Don't use:

- High Consistency
- Write heavy
- Low Repitions

Metrics

→ Size

→ Latency

→ Cache hit rate.

Step 1 : Monolithic vs Microservices

Step 2 : Inter-Process Communication

Sync vs Async vs Hybrid

↓

Mandatory - Sync.

Step 3 : Client-Server Communication Protocol

HTTP(s) (with REST API) vs WebSocket

- Client driven
- Low throughput/user
- Low infra cost

eg: Amazon

eg: Messaging, Uber (High throughput - driver location)

Step 4 : Decide the Database

- Caching : Key-val NO-SQL
- File Storage : Blob Storage + CDN
- Search Capability : Search Engine

- Metric tracking / Logging: Time Series Database
- Analytics: Data Warehouse

Otherwise: Structured Data

→ Need ACID

→ RDBMS

→ No

→ Columnar DB

Not Structured

→ Many data types, many queries (JSON)

→ Document DB

→ Less queries but ever increasing data

→ Columnar DB

Step 4: Use Cache or not

→ Consistency, Write / Read heavy, Repetitions

Message Queue : Kafka

Cache: Redis , Memcached.

Database:

- → Blob : Amazon S3 , Azure Blob
- → Search Engine : Elastic Search , Solr
- → Time series : OpenTSDB , Influx DB
- → Data Warehouse : Hadoop
- → RDBMS : MySQL , Postgre SQL
- → NoSQL → Key - Val : Redis , Memcached
  - → Columnar : Cassandra , HBase, Azure Tables
    eg: Tweets
  - → Document : Mongo DB
    eg: Amazon items
  - → Graph

1. Functional Requirements

2. NFR :

Availability

Consistency

Scalability

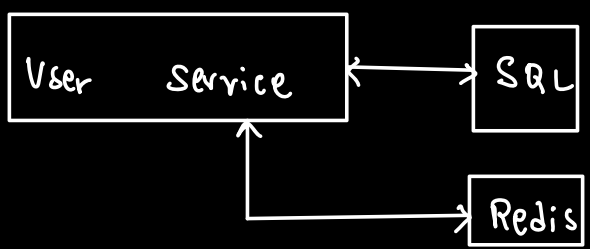
Latency

Read / Write Heavy

3. Services :

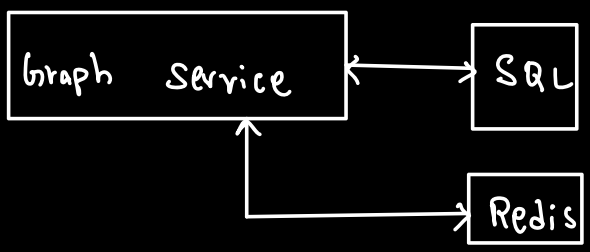
i. User service

Stores user data in SQL with Redis cache.

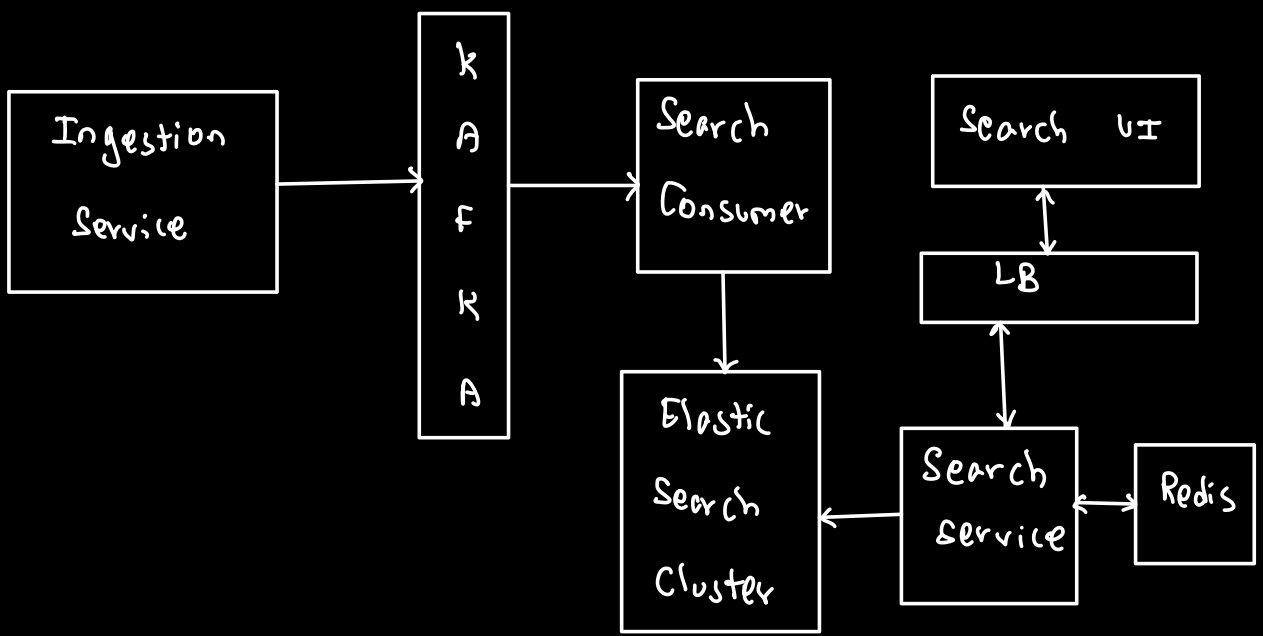


ii. Graph Service :

Relation between users, follow / friend etc.



iii. Search Service



iv. Analytics