Topic

A high level solution design for a Google Analytic like Backend System.

Assignment Requirement

Here is the high level requirement for google analytic like backend system

i. Handle large write volume: Billions write events per day.
ii. Handle large read/query volume: Millions merchants want to get insight about their business. Read/Query patterns are time-series related metrics.
iii. Provide metrics to customers with at most one hour delay.
iv. Run with minimum downtime.
v. Have the ability to reprocess historical data in case of bugs in the processing logic.

Soultion - Micro-service architecture

- purpose :
    - To allow multiple teams across region co-work in real-time.
    - To be flexible of function salability.

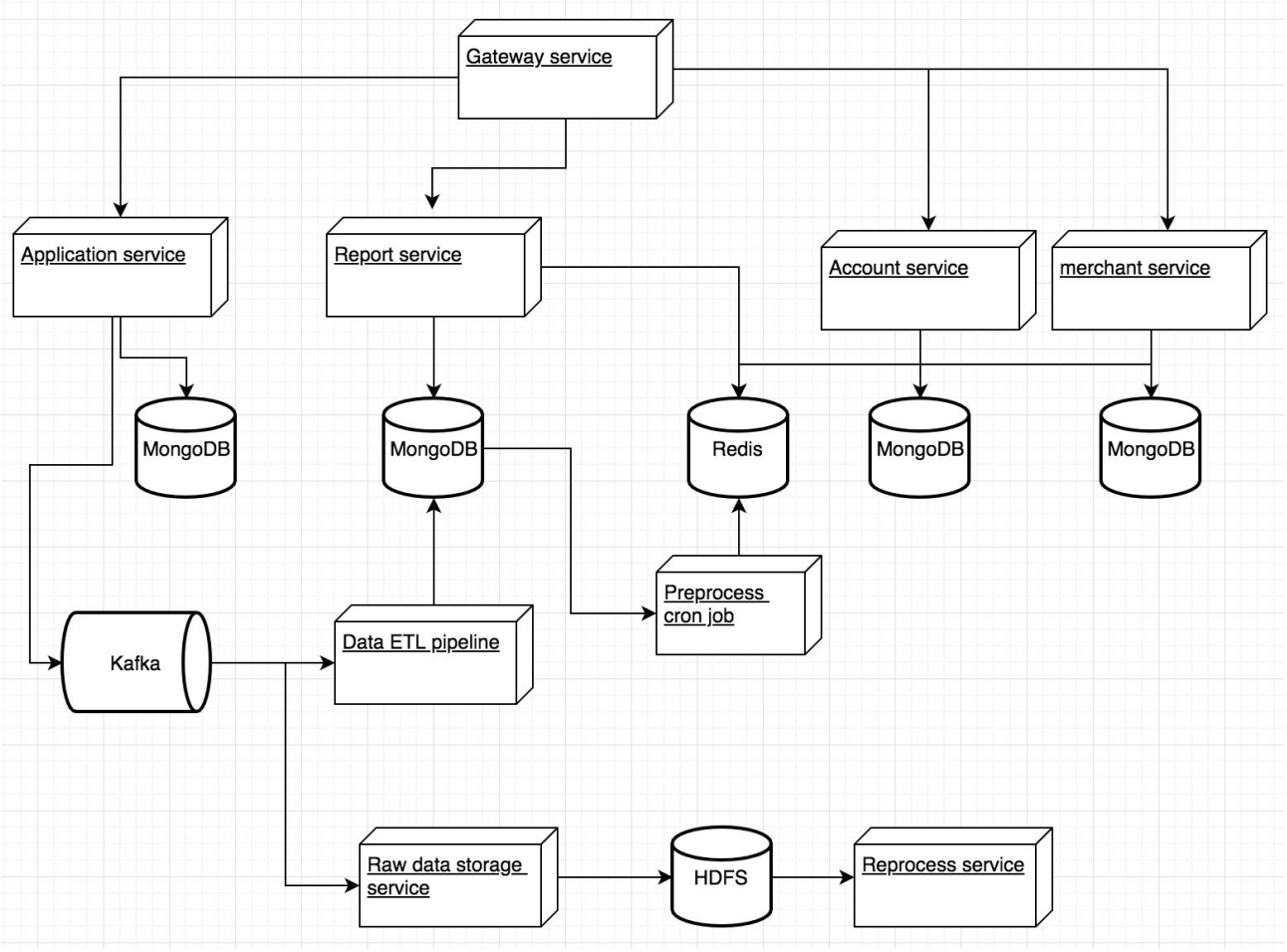
Assumptions for the system

Below description of assumptions for the solution :

1. Number of write volume per day: 1 billion
2. QPS for write events: 11575
3. Number of merchants: 1 million
4. Number of read/query events per hour: 5 million
5. QPS for read/query events: 3500

Architecture

Describe architecture of the system.



Explanation of each component :

Service	Technology	Detail
Gateway service	Scala Akka	Gateway service handles all requests in this system.
Application service	Scala Akka	Application service focuses on handling requests from tracking code and makes sure that raw data store into MongoDB and pass to queue.
Report service	Java spring boot	Report service handles all requests related to reports. Report service gets processed data from MongoDB and summarized data from Redis.
Account service	Java spring boot	Account service handles all user account data, including password, permission and other information.

Merchant service	Java spring boot	Merchant service handles all merchant data, including merchant configurations and merchant information.
Queue service	Apache kafka	Queue service provides great support to decouple different modules with different functions and assists communication between modules in micro-service architecture. It increase the ability to reduce latency as well.
Data ETL pipeline	Python	Data ETL pipelines processes, analyzes data and stores processed data into MongoDB.
Raw data storage service	Java spring boot	Raw data storage service consumes raw data from kafka broker and stores raw data to HDFS.
Preprocess cron job	Java	Preprocess cron job preporcesses summary data for each merchants and common queries by users to reduce latency.
Reprocess service	Scala spark	Reprocess service reprocesses historical raw data from HDFS if the system has bugs.

Tech Stack

All services in the system are developed based on active-active high availability cluster. In this case, the system would increase availability and reduce downtime.

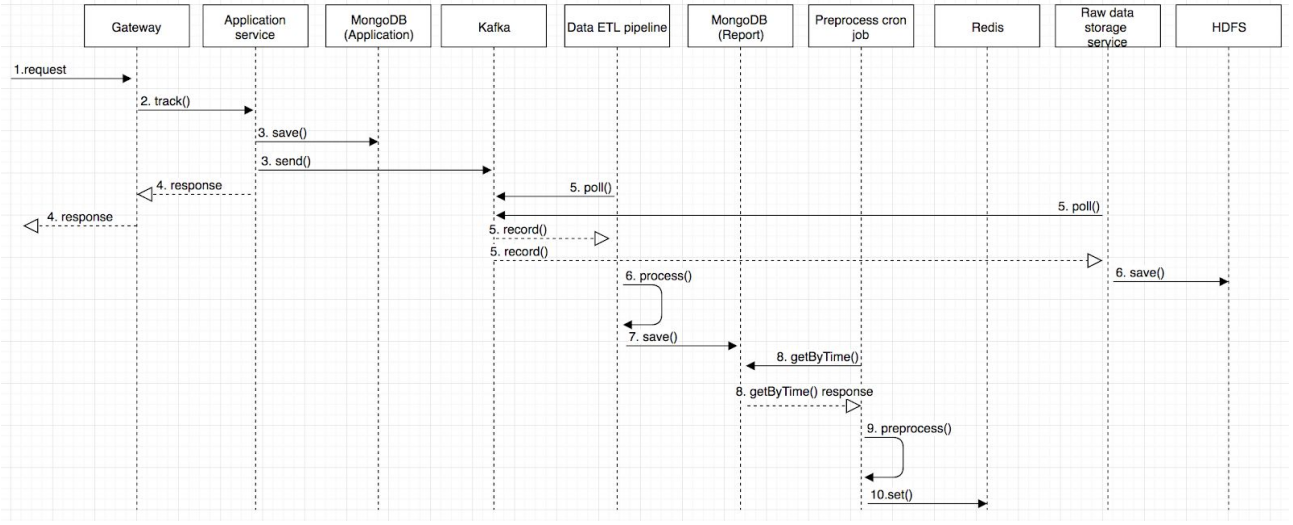
Technology	Explanation
F5	F5 works as load balancer and it is based on hardware which provides faster performance. F5 can easily control requests and response when upgrading services to decrease downtime.
Consul	Consul works as service discovery.
Grafana	Grafana monitors services in the system and support alert notifications as well.
Let's encrypt	Let's encrypt offers free SSL certificates.
Ansible	Ansible is a great tool to support CI/CD work flow automatically.
Docker	Docker is fast, flexible and good at version control. Use docker images to deploy services reduce the effort when setting up services in new machines.
Java spring boot	Java and spring boot offer simple and strong solutions to handle complicated requirements and both have rich ecosystem.
Scala Akka	Scala and Akka are great to deal with large amount of traffic. Non-blocking, asynchronous and actor make handling huge amount of traffic more easier.
Python	Python works to process and analyze data. Python contains a lot of libraries, frameworks and tools in data analysis.
Apache kafka	Apache kafka is great for real-time data pipelines, horizontally scaling and fault-tolerant.
Apache Spark	Apache spark is a fast and distributed framework in processing and

	dealling with huge amount of data on disk.

Data storage	Explanation
MongoDB	MongoDB is a NoSQL database and it has great performance in both writing and reading. Due to this system meets high traffic pressure, MongoDB could solve this problem. Another reason is MongoDB is schemaless. If we need to change API requests or data to store, the flexibility of MongoDB is a great fit to us.
Redis	Redis offers a great deal of support for developing efficient caching mechanisms and it also provides persistent function which is easy to restore data.
HDFS	HDFS is a popular way to store large amount of data. If reprocessing historical data is needed, it is faster and convenient to run spark with HDFS to reprocess historical raw data.

Sequence Diagram

Describes how the system handles write requests :



Steps :

1. All requests will be handled by gateway service.
2. Gateway service will redirect requests to appliction service.
3. Application service will do two actions. One is to save requests raw data into Mongodb and another one is to send requests raw data to kafka broker.
4. After application service finishes step 3, it will return response back to gateway service. Gateway service will return response back to client.
5. Data ETL pipeline and raw data storage pipeline will both consume requests raw data from kafka broker.

6. Raw data storage pipeline will save requests raw data into HDFS. Data ETL pipeline will process requests raw data.
7. After data ETL pipeline finishes processing requests raw data, it will save processed data into MongoDB.
8. Preprocess cron job service will get processed data from MongoDB every hour.
9. Preprocess cron job service will summarize processed data and preprocess common queries data from users.
10. Preprocess cron job service will save summarized data and preprocess common queries data into Redis.

What's More - How many Redis are required to support this system?

Based on the requirement :

- Read/Query patterns are time-series related metrics.

Design to reach the above reqirement :

- Preprocess cron job generates summarized data and common queries data of the past 1 hour.
- Preprocess cron job stores data into Redis.
- Based on selected time range, the data would be aggregated by report service and composed to JSON response.

To support this design :

- Hypothese
    - Each merchant generates 1KB of summarized data every hour
    - With 1 year TTL, the total amount of data would be ~8.5TB in Redis.
- Result
    - Based on the QPS assumption - Number of read/query events per hour: 5 million- the Redis specification would be 125 machines with 4 cores CPU and 72GB ram