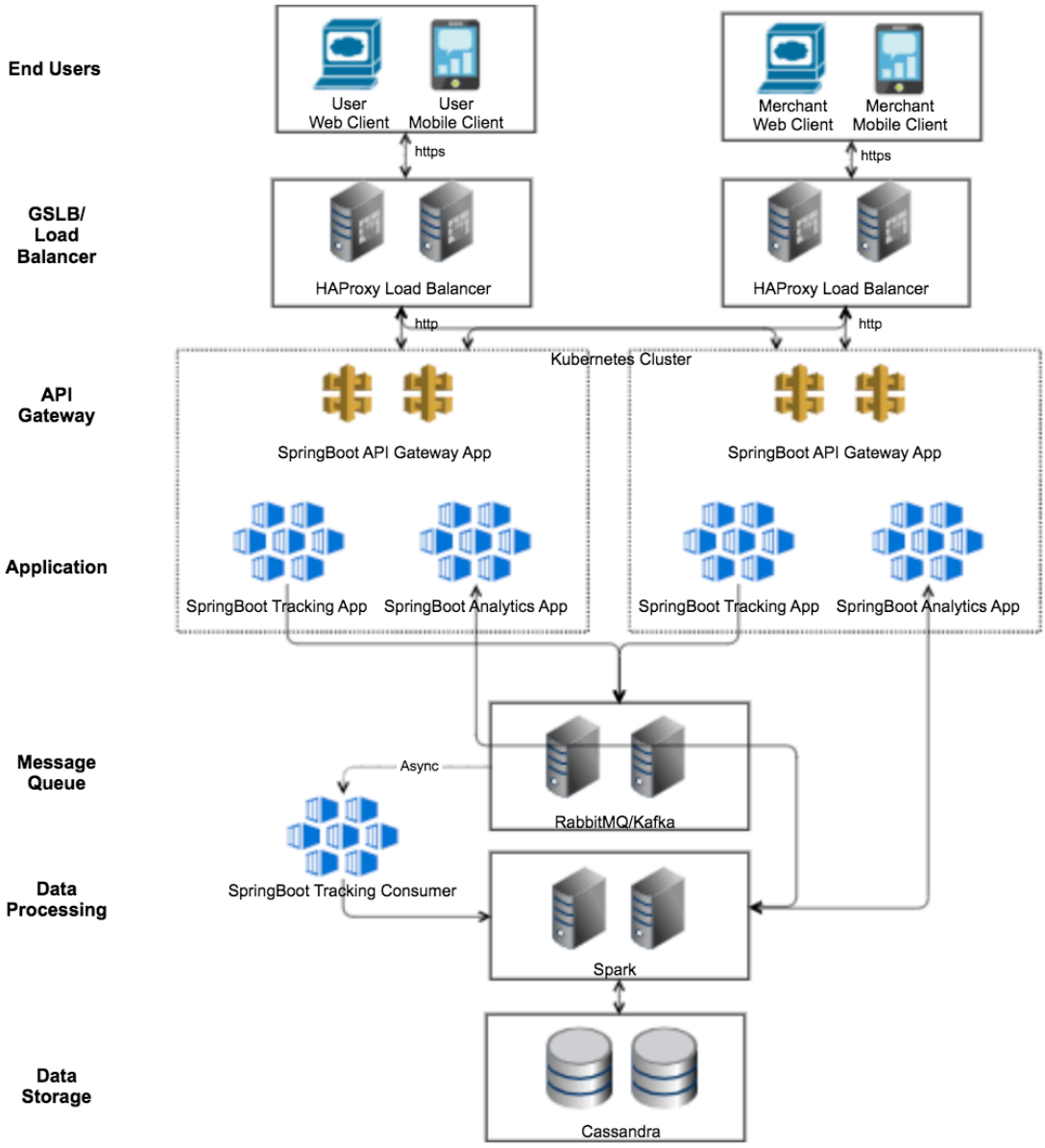
Design a Google Analytics like Backend System

Design Overview



Handling Tracking Write Events

End Users

With client library, for each user interaction, user web/mobile clients send HTTPS with JSON request to the system.

GSLB/Load Balancer
Global load balancer routes the request to the nearest upstream application clusters/servers. In case there is a failure, the load balancer will fail-over the traffic to another upstream clusters/servers for high availability.

API Gateway
API Gateway check the user requests to identify the merchant, and check if the request is authenticated and authorized. Once this is done, it records some statistics and route the request to the corresponding service.

Application – Tracking Service
The Spring application processes the tracking information in the requests as needed. Then it publish the information to message queue.

Message Queue
The message queue cluster receives the tracking information and stores it with fault-tolerant and durability. The information will be asynchronously subscribed and consumed by the Spring worker application – tracking consumer. Then, the consumer sends the information to Spark.

Data Processing and Storage
With Spark, data is processed and manipulated as per the data model. Then, it is written in Cassandra.

Handling Analytics Read Events

End Users
Merchant web/mobile clients send HTTPS with JSON request to pull analytics information from system.

GSLB/Load Balancer
Same as above.

API Gateway
Same as above.

Application – Analytics Service
The Spring application processes the analytics information required in the requests as needed. Then it pull the responding information from Spark with queries.

Data Processing and Storage
Data are pulled out from Cassandra, and are re-assembled with map-reduce in Spark.

How it fulfils the system requirement?

1. Requirement 1 and 2 – Billion scale of write and read volume, with write >> read
    a. With billion scale of requests per day, database will be the bottom neck.
    b. Unlike monetary transactions, ACID is not required for most of the cases in analytics. Hence, it will be simpler to use NoSQL database with AP(availability and partition tolerance) to achieve the volume.
    c. Cassandra is a distributed database with linear scalability and high availability. It is also highly performant, especially for write. Hence it is selected in the design. On top of it, Spark is useful for data processing for analytics purpose.
2. Requirement 3 – Provide metrics to customers with at most 1h delay
    a. With the delay tolerance, RabbitMQ/Kafka is added to improve system reliability and handle peak traffic.
    b. To handle sharp and sudden peak of requests, depending on the bottleneck, we could add consumer pods or Spark/Cassandra nodes. Each of the components in the system can be scaled easily, compared to traditional enterprise components.
3. Requirement 4 – Run with minimum downtime
    a. The load balancer layer enables multi-DC fail-over and cluster fail-over.
    b. Designed to be stateless, the application pods can be scaled easily if any of them is down.
    c. The message queue and data store could support multi-DC fail-over, with a higher cost of latency and bandwidth.
4. Requirement 5 – Able to re-process historical data
    a. Designed to support idempotency, application should be able to process data multiple times with the same result state.
    b. A logging system is needed, e.g. ELK stack (Elastic search, Logstash, Kibana). The applications should log the request into the logging system or in the database, so that we can manually fix and replay the request.