Design Question

Design A Google Analytic like Backend System. We need to provide Google Analytic like services to our customers. Please provide a high-level solution design for the backend system. Feel free to choose any open source tools as you want.

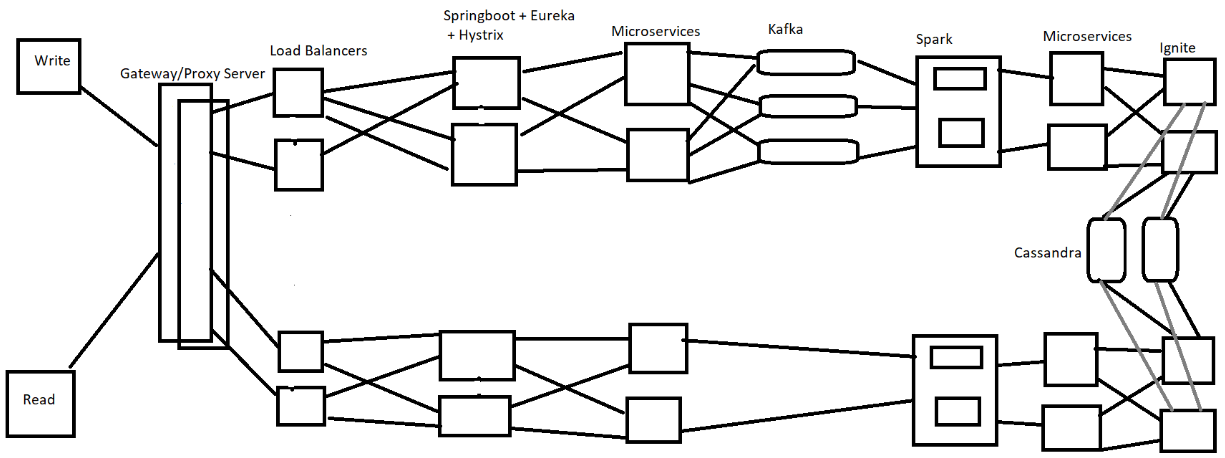
Requirements

1. Handle large write volume: Billions of write events per day.
2. Handle large read/query volume: Millions of merchants wish to gain insight into their business. Read/Query patterns are time-series related metrics.
3. Provide metrics to customers with at most one-hour delay.
4. Run with minimum downtime.
5. Have the ability to reprocess historical data in case of bugs in the processing logic.

Design Solution

- To Design something of such large scale, consistency, reliability as well has fault tolerant, we will utilize microservices based architecture that will be distributed in groups among different regions based on the needs. The regions can be in one country or around the globe based on the needs.
- We will have two APIs for our system, one for read and one for write large amounts of analytics data.
- Right after the client, we will have a load balancer with high availability, something like NGINX Plus Load Balancing or HAProxy Load Balancer along with sticky sessions.
- Next the traffic will be at our microservices based servers, utilizing spring boot to ensure system is scalable and resilient. These group of services would be mainly for data processing and parsing on read/write events.
  - Implemented with Eureka Zuul and Hystrix for fault tolerance. The reason I chose these is because they work well with spring boot, otherwise zookeeper would have worked as well. Eureka will be set up in a high available way.
  - With micro services, one can deploy modules to multiple servers for increased availability and performance.
  - System should be able to handle multiple failures, this is where Hystrix circuit breakers will come into play.
  - Deployments of these would also be done using Kubernetes cluster as well.
- The other requirement for our architecture is metrics. Thus, for that we can also use Spring Boot actuator for metrics purposes or Splunk for logging.
- From our servers, we will attach Message Queues either RabbitMQ or Kafka could be utilized, I will choose Kafka due to the following reasons:
  - High volume publish-subscribe messages and streams platform
  - Kafka offers much higher performance than message brokers like RabbitMQ. It uses sequential disk I/O to boost performance, making it a suitable option for implementing queues. It can achieve high throughput (millions of messages per second) with limited resources, a necessity for big data use cases.
  - This can be used to store billions of records with low latency.
- After that we will process our data through Apache Spark for batch processing. This will further make things easier to handle billions of events as required. This will allow:
  - Fast parallel stream processing
  - Provides highly reliable fast in memory computation
  - Fault tolerance capabilities because of immutable primary abstraction named RDD.
- The parallel batch processing jobs will send data to microservices based servers that will help process depositing/retrieving data to/from our cache system as well our database. Each group of servers can handle a part of the request.
- The caching mechanism is highly desirable for faster process and retrieval of data. Here we will select Apache Ignite as compared to Redis for the following reasons:
  - Trigger based (cache interceptors and events)
    - Since these are events (read/write)
  - Replicated cache

    - Provides Immediate Consistency
- As for the database we will use a Time series-based DB, Cassandra. The reasons for us to pick this database for our case are:
    - cluster of nodes
    - Data partitioning
    - Distributed across regions
        - One in each
        - Or a group of machines in regions
        - Or all in one region
    - These are resilient and fault tolerance
    - Uses consistent hashing as well
    - Cassandra is a large scalable database that can be used to hold historical data for reprocessing. This will also be a time processed event, we can hold data for a decided period of time based on requirements. We may also include file system for archiving if required.
- The description of the design provided thus far has mainly been for the write process, for the read process (dashboard view) we will have the same process expect we won't need kafka for this and our Apache Spark will help us with Timeseries based data processing combined with the efforts of Cassandra and Ignite on this aspect.
- The diagram below describes the model explained in the points above. Please note I do not mean just two servers or machine at a time, I mean to show they can be scaled to a desired size at each stage. Moreover, as mentioned before as well, we will have replicas of this system throughout regions.


Write
Read
Gateway/Proxy Server
Load Balancers
Springboot + Eureka + Hystrix
Microservices
Kafka
Spark
Microservices
Ignite
Cassandra