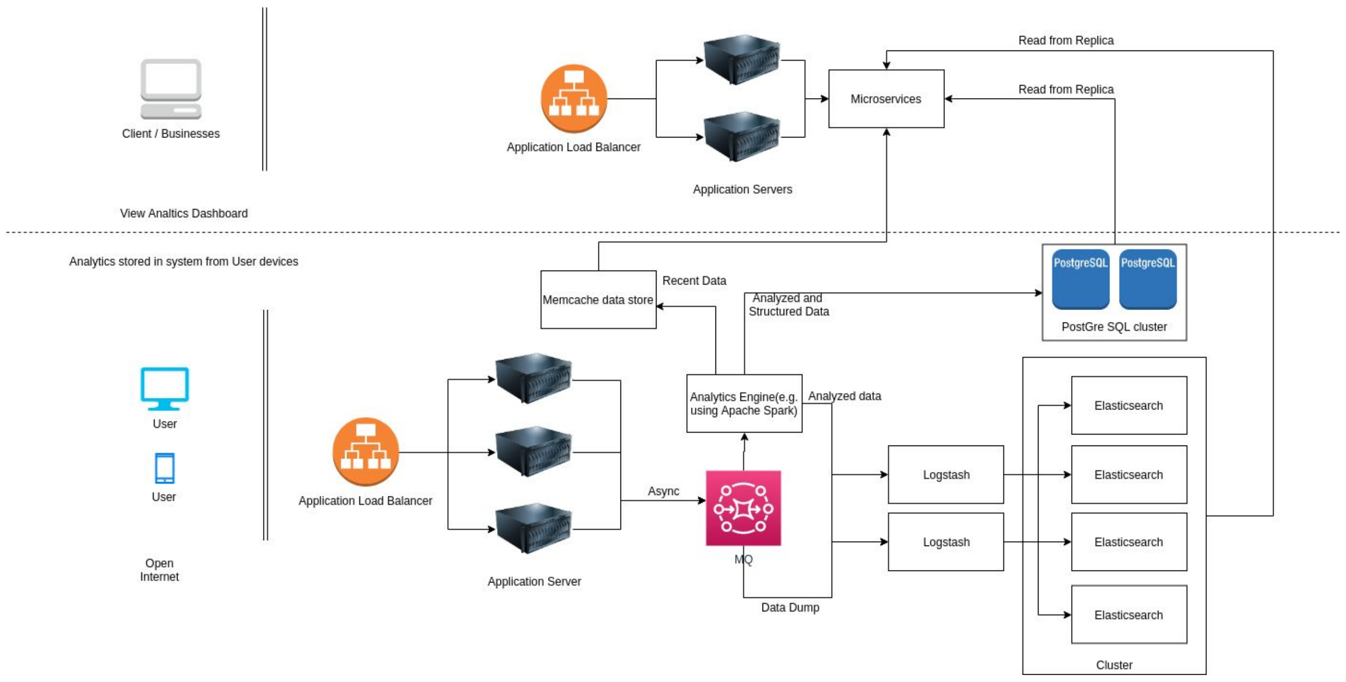
Problem Statement: Design A Google Analytic like Backend System. We need to provide Google Analytic like services to our customers. Pls provide a high level solution design for the backend system. Feel free to choose any open source tools as you want.



System Design:

1. Application Load Balancer manages the load across servers over synchronous apis.
2. The Application Server responds to API calls from user's device by sending a 200 OK status or an appropriate one, without processing the whole request. It delegates the task(s) to be performed to the MQ. The server also horizontally scales the microservices when load is high.
3. The Messaging Queue (e.g. RabbitMQ) manages the queue in which the messages are queued before they are processed. One copy of the message is sent to keep as dump in Elasticsearch (through Logstash). The same message is also sent to the Analytics Engine.
4. The Analytics Engine (powered by Apache Spark) also runs as a microservice and supports horizontal scaling to meet peak traffic demands. Apache Spark parallelization features help to speed up analysis. The analyses may be performed periodically such that if an analysis take a few milliseconds to perform, they are scheduled hourly to ensure important metrics are available within an hour.
5. Logstash nodes are more than one to guarantee high availability. They connect to multiple Elasticsearch nodes (master and data nodes) managed inside a cluster. Because of a high traffic application, writes are always performed on the data nodes whereas reads are performed on the replica nodes. The master nodes manage the data nodes.

6. Recent data that have been collected, analyzed and are highly likely to be queried very often are stored on the cache server (memcached). This may include metrics such as count of visitors on a specific day.
7. Analytics Engine also store structured data on relational database cluster like PostGreSQL. This helps to utilize the full power of relational databases, joins, etc.
8. Clients (or businesses) log into the web application and queries for data regularly.
9. A load balancer is present there to ensure the loads are equally distributed.
10. Application servers hold the microservices capable of scaling.
11. The microservice collects necessary data from only the replicas to ensure write operation is not obstructed. A cache server helps to gather important/frequently used metrics/data be calculated and present for fast service.

i. **handle large write volume**: Billions write events per day. - Load Balancing with MQ help to queue requests as they are processed by the analytics engine. Logstash also maintains its own queue and will further help to buffer the pipeline. The application servers ensure that the user requests are not blocked for long. The queue and the rest of the pipeline work in asynchronous mode.
ii. **handle large read/query volume: Millions merchants want to get insight about their business. Read/Query patterns are time-series related metrics.** - Replica databases and replica Elasticsearch instances help to make sure that the read process is not hampered. Elasticsearch is good with time series data as it maintains the time on its own.
iii. **provide metrics to customers with at most one hour delay.** - Use of Elasticsearch enables features such as Near Real Time data availability meaning the replica instances of the data have new incoming data within seconds. That means the replica instance is capable of providing data that has just been added. Analysis may run periodically so that data is accumulated and then the analysis can be performed once on the whole data collected in the meanwhile. This should be scheduled such that one hour delay time is not crossed by the engine.
iv. **run with minimum downtime.** - Elasticsearch facilitates very low downtimes since the available replicas become primary until the primary instance becomes available once again, hence not interrupting service at all.
v. **have the ability to reprocess historical data in case of bugs in the processing logic.** - A dump is kept of the original data so that it may be referred back to in case there are bugs. These data may be stored in compressed state and even archived after a certain time has passed.

Please note that for the ease of representation, API hits (write) from clients to application servers (to save/update data or config) are not shown above in the diagram.