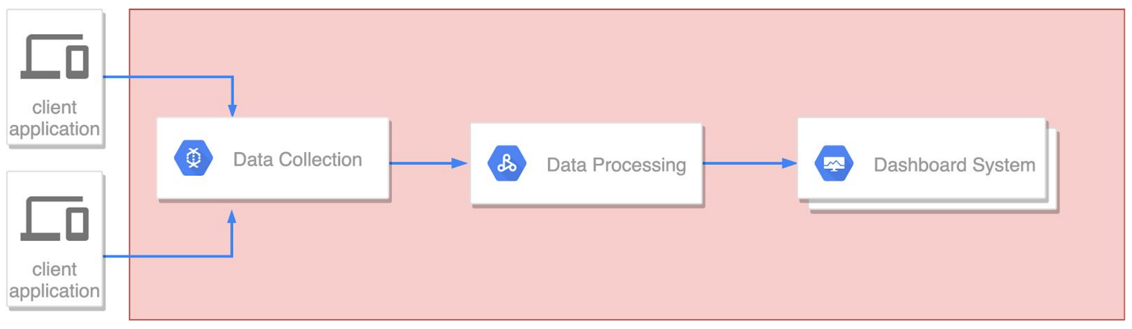
Design Question

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.

1. A high-level view of system components



There will be 3 main parts of the system:

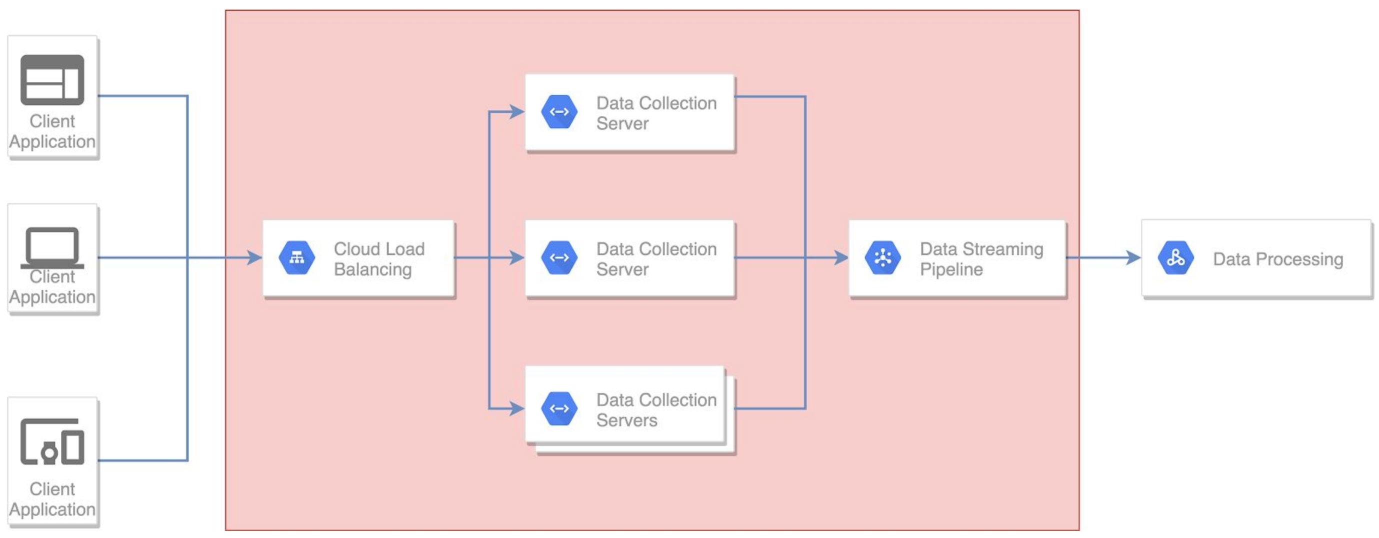
Data Collection System is to receive data from client applications. This system needs to able to handle an extremely high volume of writing requests, and be stable enough so that could always provide reliable service to clients.

Data Processing System is to do data polishing, re-formatting, analysis, and deliver useful data to other systems. This system must be fast enough in order to meet the 'at most one hour delay**' requirement and need to have ability to re-processing historical data.**

Dashboard System provides user visualized data and data analysis results. This system needs to take care of a huge amount of queries.

2. Components Details

2.1 Data Collection System



Data Collection is just a service provided API for client applications for receiving data. It is mainly responsible for authentication checking for clients, and pushing data into a Data Streaming Middleware. It could also do some basic data validation, re-formatting, quick calculations or categories data if necessary, but these operations must be really simple and could be done very fast in order to handle a huge volume of incoming data.

Considering the the fact that there will billion write requests per day and system needs to maintain high availability, there should be multiple servers spinning out. And we need a load balancer in front of those servers.

The data streaming pipelines must satisfy the following conditions:

- Support pub-sub strategy for streams of records
- High-throughput. Could accept a huge amount of data pushed from all data collection servers.
- Provide data persistence. There will a lot of data accumulated into the pipeline. We do not want to lose data for any reasons.
- Low-latency
- High availability and easy to scale.

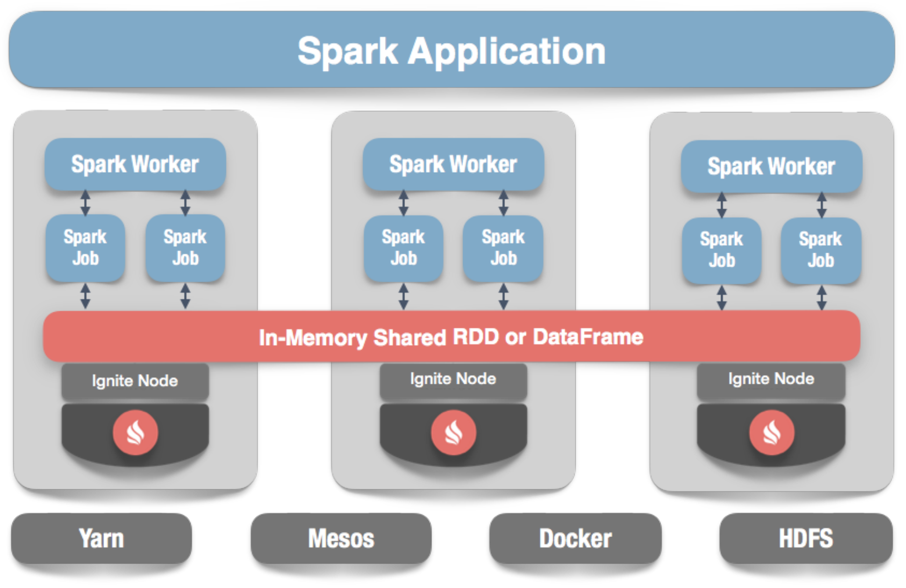
Kafka is pretty fast, lightweight and meets all the requirements we have.

2.2 Data Processing System

Data Processing System is put in place between collection and dashboard visualization. It needs to process incoming raw data, re-format them in order to provide valid and clean data for analytics and/or monitoring propose. It works with the following process:

1. Data process phase 1: Polishing row data. E.g. data validation, de-dup, reformatting.
2. Persistent data: In order to provide historical data, we need to save the data on hardware. Also allowed independency of later processing.
3. Data process phase 2: We might have demand to do data analytics, machine learning on collected data.
4. Deliver data to dashboard systems with time-series

Could use Spark for this system. It provides high performance on processing both streaming data and more structured data, also has distributed machine-learning framework supported.



Picture found from Ignite website: https://ignite.apache.org/use-cases/spark/shared-memory-layer.html

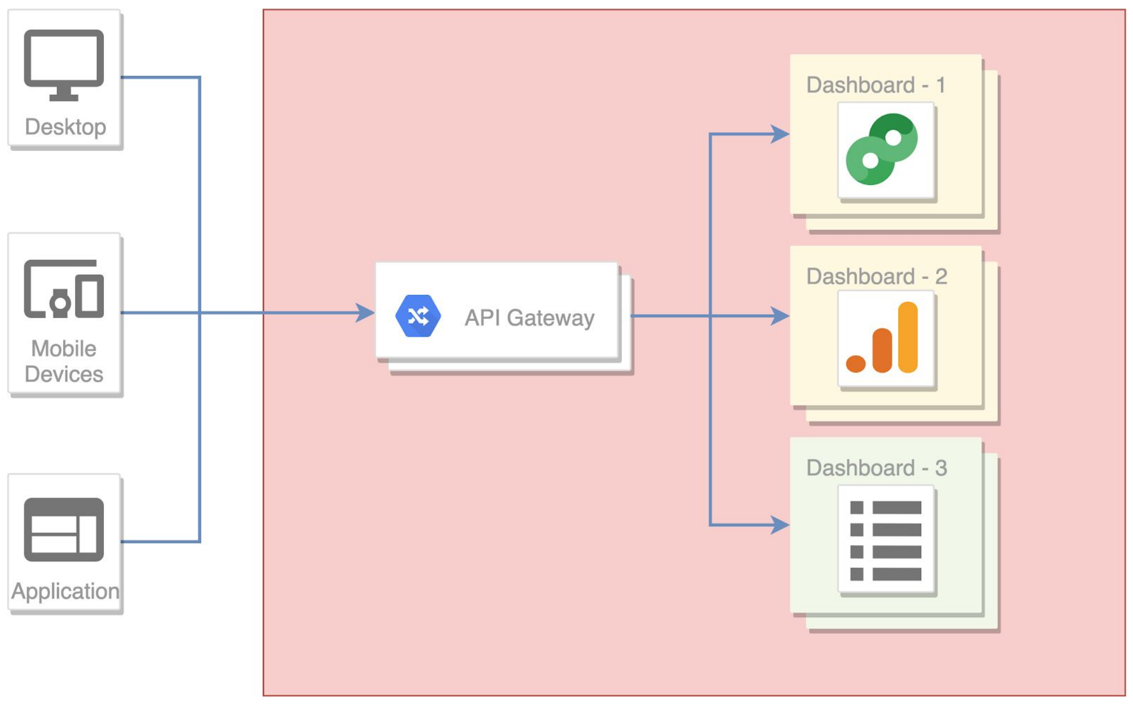
In order to increase Spark works' performance, we could use Apache Ignite which achieve in-memory performance and allows data and state shared among Spark jobs.

For data persistence writes from Ignite, Apache Cassandra should be a good fit. It is a wide-column datastore which is suitable for data with or without relationships. Cassandra also has really good write and read performance.

Data sharding by timestamps.

2.3 Dashboard System

This is the system that works for users directly, will contain various business requirements. Considering scalability, it's designed as micro-services based system. Different type of dashboard could be built with different service, e.g. dashboard shown user clicks on web-pages and trace of cloud space using for applications.



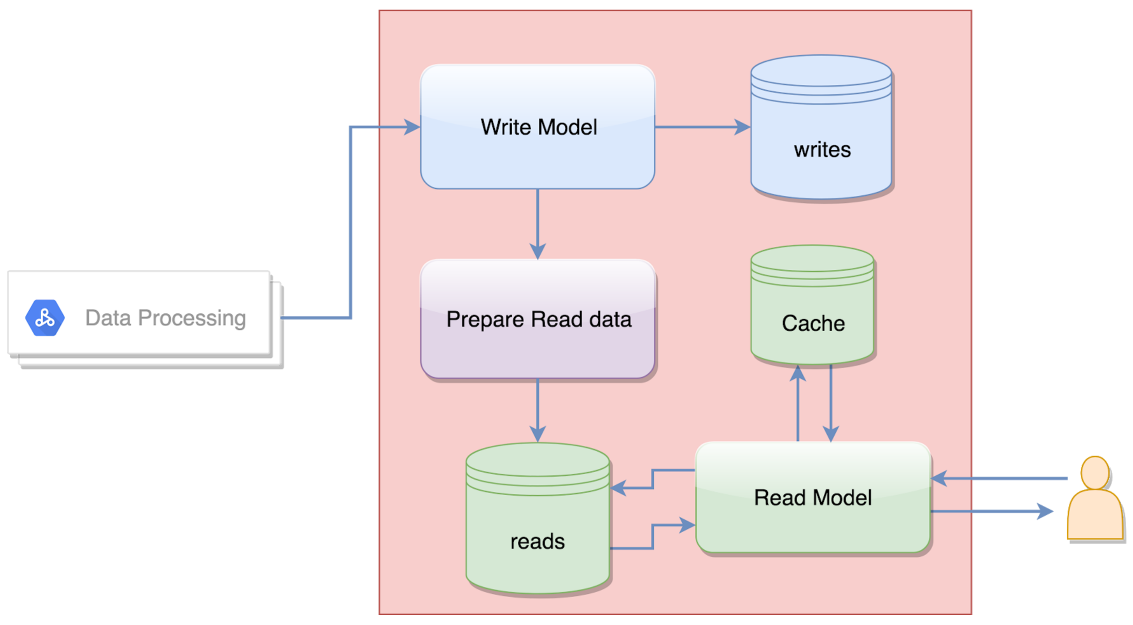
Since the number of microservices might increase quickly, let clients known all endpoints addresses on each of the microservices becomes quite painful. And there are some logics needs to be done on each service that is exactly same, it's redundant work to implement them on each service.
In order to solve this problem, it's a good idea to provide a API Gateway infrastructure. The API Gateway will support the following features mainly:

- Provide a single entry point for all a cluster of microservices
- Routing requests to the appropriate service
- Limit the number of incoming requests to prevent the server from crashing caused by too many requests
- User authorization
- System healthy monitoring

Netflix Zuul is a well-known open-source API Gateway solution. There are a set of OSS provided by Netflix could support all the functionalities listed above.
Using Ribbon as a client-side load balancer.
Eureka with ribbon could support service discovery
Hystrix could provide circuit breaker control.
Turbine and Hystrix Dashboard will visualize monitoring information.

Database chosen for each microservice could be different based on the type of queries the service needs to provide. Non-relational DB will be more suitable for this type of services.

The high volume of reading requests has to be taken into consideration. Same as the data collection system, each micro-services of dashboard system also need to spinning out multiple servers to increase capability for serving queries.
Separating Writing model and reading model is another approach needs to be taken if the information shown to end-users requires a lot of computation or complex queries. As shown below:


Data Processing
Write Model
writes
Prepare Read data
Cache
reads
Read Model