Design Question

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.

The system needs to:

i. handle large write volume: Billions write events per day.

ii. handle large read/query volume: Millions of 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.

Solution

1. **Basic Steps:** Before designing the system, lets first briefly discuss about working of Google Analytics. At a high level it involves following steps:

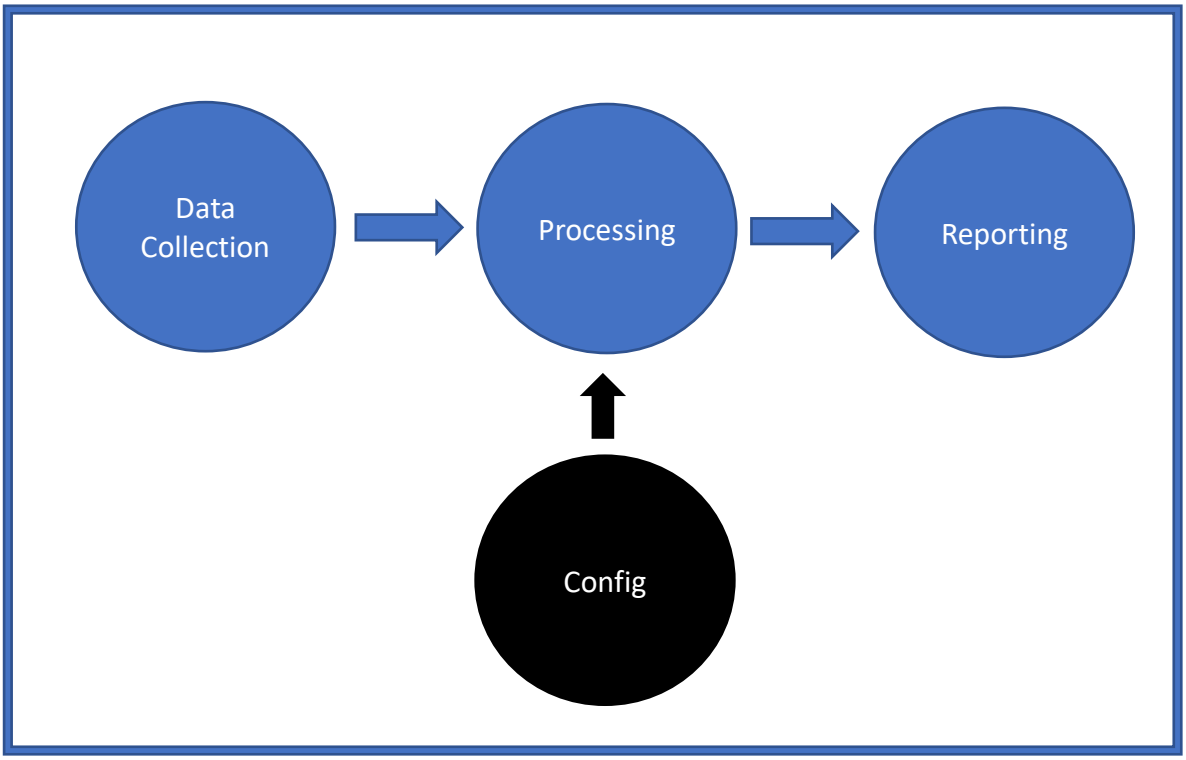

Figure 1: Basic Steps

a. **Data collection:** User's behavioral data is collected from website across different touchpoints.
b. **Processing:** Data is cleansed(filtered/validated), ETL based on merchant's config.
c. **Reporting:** Data is reported via web portal where data can be visualized in the form of graphs, charts and dashboards to uncover business insights.

2. **System Flow:** Before details system design the abstract flow should look like:

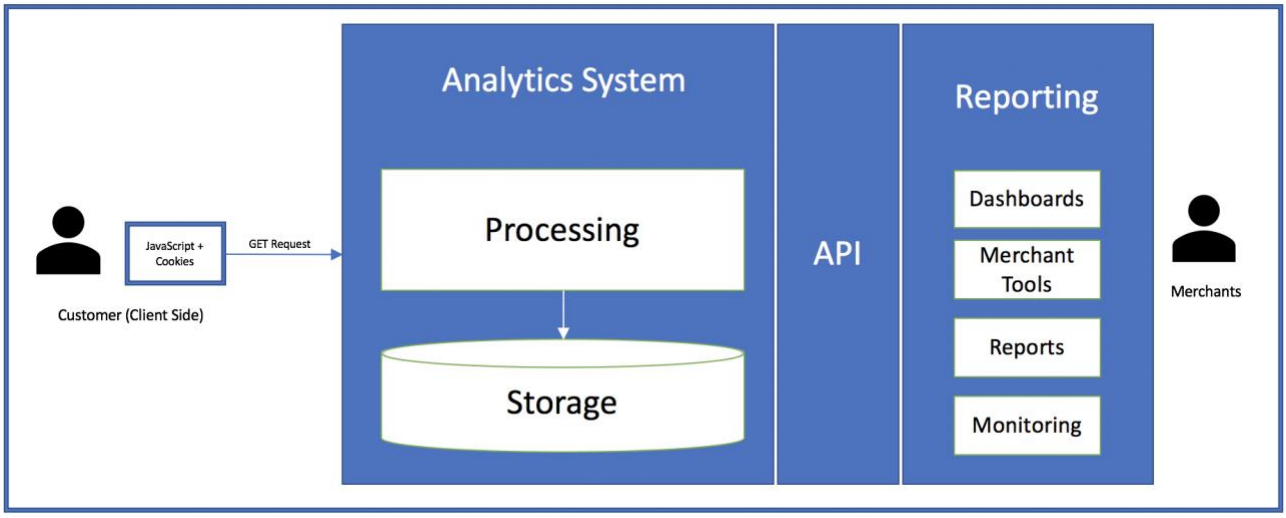

Figure 2: Brief System Flow

Step 0: ID Generation and Integration at Merchant side
- Merchants needs to generate unique ID and implement a JavaScript code on their website.

Step 1: User's behavioral data collection
- When user visits the website the JavaScript checks/updates cookie and makes asynchronous GET request to analytics system.

Step 2: Request is processed and ETL
- The system extracts user's information based on query parameters, cookie and request headers (IP address, User agent, etc.)
- Data is cleansed, filters are run, data is sessionised, merchant configurations are applied, data is stored in persistent storage

Step 3: API layer serves as gateway for reporting
- We can place an API layer that queries our system to feed into our reporting system or web portal for Merchants.
- Merchant access the reporting tools which will call APIs to render data/reports.

3. **Detail System Design:**

1. **Users Browser**
- When user access the webpage, browser makes XHR request to our analytics system

2. Load Balancer

- Load balancers distributes the request across the web servers.

3. Web Servers

- Web servers can be Lighttpd or Ngnix for high performance.
- These web servers receive request and write it to log files immediately.
- These logs continue to accumulate and at regular interval let's say 5-10 minutes are first transformed using Logstash and queued using Kafka to be processed further.

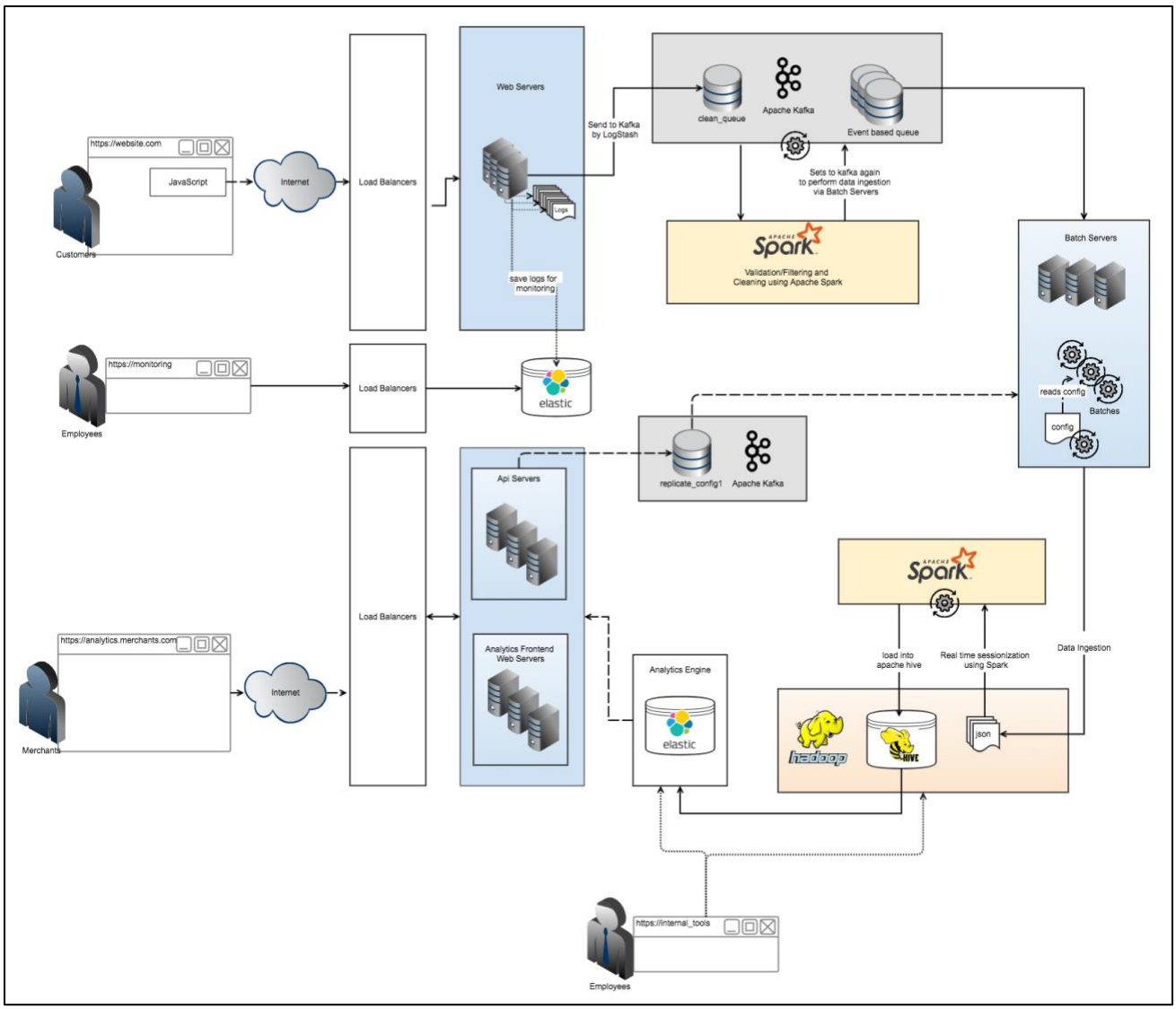

Figure 3: System Design

- 5-10 minutes interval would allow us to collect large data set (of course interval can be modified depending on usage scale and load) for batch processing and offer near real time data reflection on analytics portal with minimum delay.
- The logs are set to Kafka via Logstash for cleaning and validation of parameters.
- Request filtering if applied by merchant should also happen here.
- We will also store the raw logs to separate Elastic Search DB for monitoring and investigation purpose in case of claim or trouble.

4. Kafka:

- For data queueing purpose we use Kafka

5. **Apache Spark:**

- We will upstream the data for cleaning and validating using Apache Spark.
- We will filter and validate the data and will discard invalid if ant
- Note: The webserver store the raw request to separate elastic DB, so in case of wrong processing we can still extract and process the data.
- After filtering we will set several Kafka Queues
- For each type of event and target metric separate queue will be set. This include: Page scroll, Page Visited, video played, checkout made, transaction placed and so on.

6. **Batch Servers (Running residential scripts)**

- Batch server will have separate batches to process each processing queue.
- They handle different processing needed for each request based on configuration.
- Note: this config can be updated in real time from analytics portal or via APIs. We will revisit about it below.
- Each batch will watch its queue, fetch data as they arrive and after processing places it on Hadoop in JSON format.
- This can also be aggregated as list of json based on requirement.
- Programming Language for this batch can be: Python, Perl, Java or Frink

7. **Sessionisation in Hadoop**

- Json file will be sessionized on real-time using Apache Spark again and store in persistent storage on Apache Hive.
- Apache Hive can be used for analytical query processing.

8. **Load Hive data to Elastic**

- For retrieval and querying from behind the APIs/ Web portal we will store data in Elastic as Hot load.

9. **Analytics Portal (API Server and Analytic servers)**

- Analytics server presents the frontend for our merchant.
- Note: Like Google analytics merchant can be an organization itself and thus different access control need to be managed here. Like:
    - Edit/manage reports
    - Collaborate
    - Read and Analyze only
    - Add new properties, goals
    - Analyst
- Web portal will make API calls to fetch and draw graphs, analyze metrics and so on.
- APIS can also be used independently using.

10. **Replication for Modify APIS**

- APIs like updating filter, creating properties and goals need to replicate them at different stages.
- Here again we will use Kafka to set multiple queues and their respective batches will process it.