# Google Analytics Type Backend System Design

Contents

[Google Analytics Type Backend System Design 1](#_Toc23992581)

[Assumptions 1](#_Toc23992582)

[Functional Workflow 1](#_Toc23992583)

[Functional Approach 1](#_Toc23992584)

[Design 2](#_Toc23992585)

[Explanation of write flow: 4](#_Toc23992586)

[Explanation of read flow: 4](#_Toc23992587)

[Concerns 4](#_Toc23992588)

## Assumptions

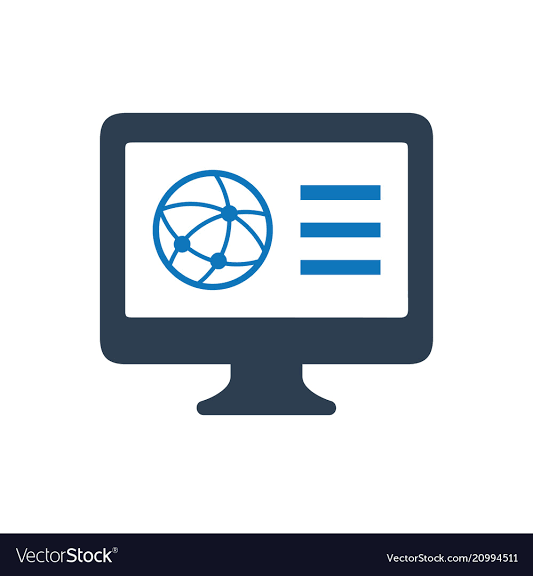
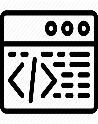
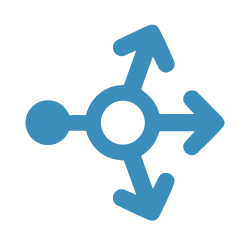
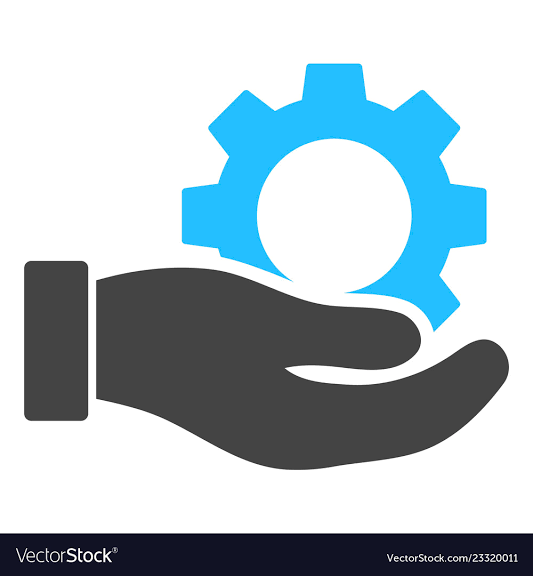
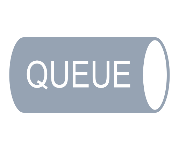
1. Both write and read volume is high. However, the write volume is much higher than read volume (write heavy).
2. Analytics queries can tolerate a delay of 1 hour.
3. Proprietary software is not desirable.
4. Number of time series queries are limited.

## Functional Workflow

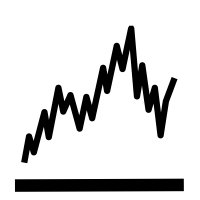
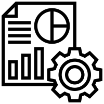
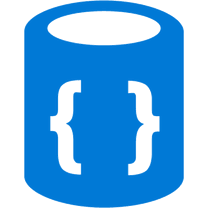
The basic premise of the system is based on collecting analytics data points collected from end user interactions. This data points constitute the events which are persisted in a data store from which time series-based queries can be executed and displayed via some dashboard visualizations.

## Functional Approach

A general outline of how the system would look like functionally is shown below:

     writeflow

Tracker scripts Load balancer Core Service

Readflow   

Time Series Processing DB

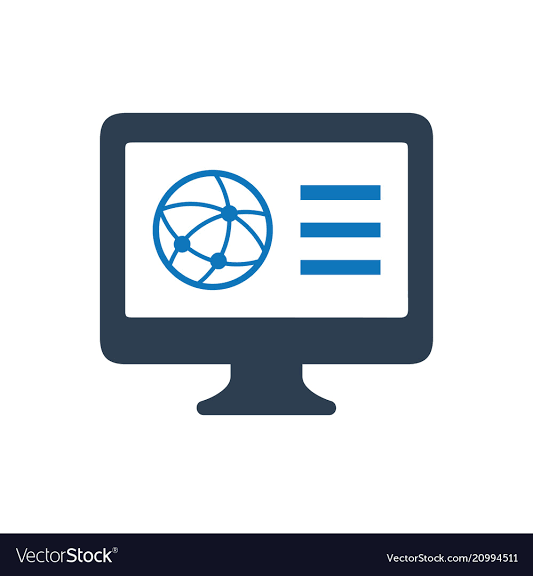
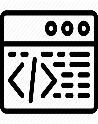
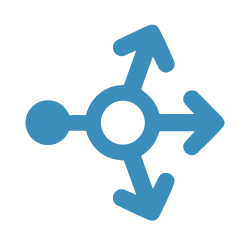
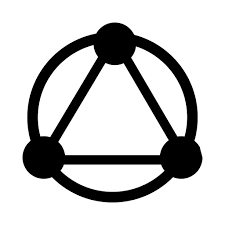
The various broad level components in the system are:

1. Tracker scripts – These scripts are responsible for collecting user data points and sending them to the back end for processing.
2. External Load Balancer - This will be used to load balance and distribute the load among the service instances.
3. Service layer – This may be composed of smaller microservices which orchestrate the job of write flows (persisting to DB) and read flow (execute time series queries on the database).
4. Queue – This is to support asynchronous handling of data events which will be used to persist in database
5. Database – This is a write optimized database for persisting huge data quickly and in a resilient manner.
6. Time series processing layer – This mainly corresponds to the read flow (visualization of analytics).

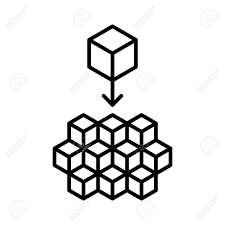
## 

## Design

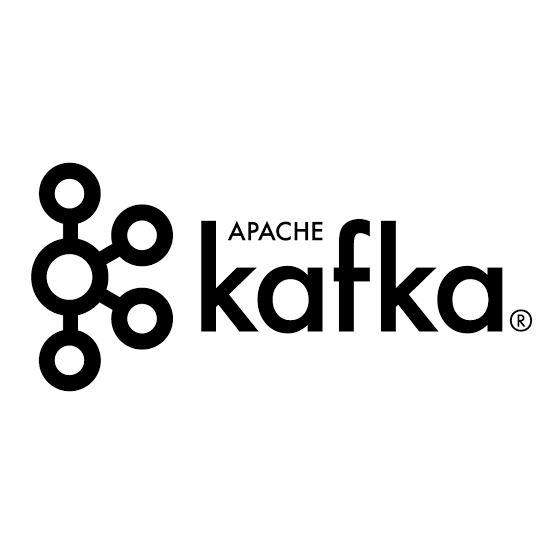
The basic design approach can be categorized into two flows write and read. The write flow is outlined below:

(open source LB like HA Proxy) (API Gateway like Kong or Zuul)

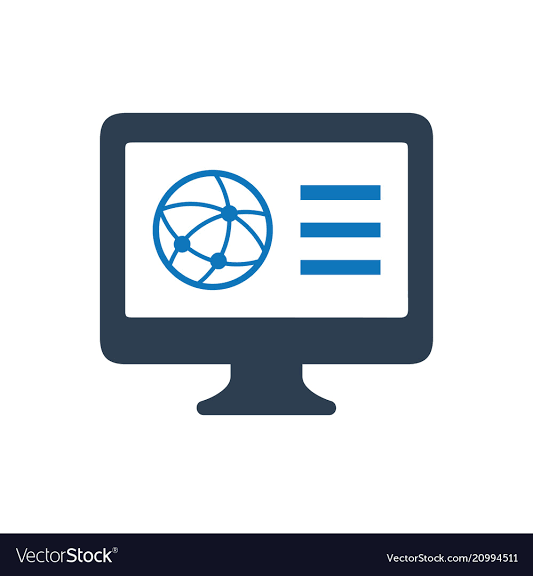
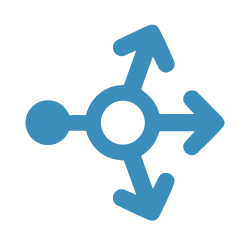
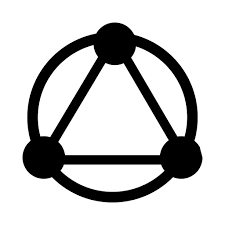
 

(Application microservices) (Service Discovery)

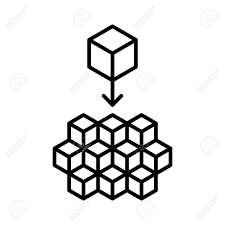
 



The read flow is outlined below:

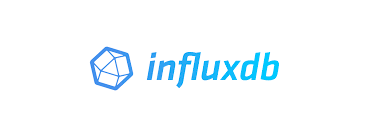
(open source LB like HA Proxy) (API Gateway like Kong or Zuul)

(Application microservices) (Service Discovery)





Decision point to consider same db for read/write or use separate ones. Use a time series optimized db like influx db in case of separate read flow

### 

### Explanation of write flow:

We want to support availability of the back end and hence we will have multiple back end services split over geographies, HA proxy comes to the mind and can also be used to monitor health status of individual backend service nodes. The business functionality will be served by a set of microservices front ended, preferably by an API gateway (like Zuul), this is because we can abstract out the complexity and adopt of a flexible configuration model backed by service discovery to route requests to the backend services. The backend services can be implemented using SpringBoot or some micro framework like Spark. In its most minimal form, the read and write flows will be handled by separate microservices (this may be split further on domain lines). We can use a Kafka Spark combination to enable fast persistence of data point events/messages into the database. Since the system is assumed to be write heavy something like Cassandra comes up as a good choice, with its promise of linear scalability and resilience.

### Explanation of read flow:

Ideally for time series calculations, we would prefer to use something like Influx DB . However, I am not sure whether building another offline pipeline to load from Cassandra to Influx will be a good idea. However, if we see that read performance is heavily impaired, we can switch to different DBs for read and write. In the current approach, we use Spark again for time series processing from the Cassandra db and use that for our data visualizations.

## Concerns

1. Having the same database for read and write might not be ideal. We might have to separate this out in a CQRS kind of approach.
2. All components in the system need to support failover either by replication or some other means.
3. There should be some operational/monitoring for the whole system as a whole to investigate and troubleshoot issues.