Task

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.

Functional requirements:

- The system shall provide an API to capture clients browser activity like clicks;
- The system shall provide an API to receive data for reports;
- The system shall provide an API to configure custom data queries per merchant.

Non-functional requirements:

- The system shall be able to process billions of events a day, which means:
  - 1 billion of events a day is 11574 events a second on average or transaction per second (TPS)
  - 25000 TPS during peak hours considering during hot period load is doubled
- The system shall allow 5 millions of merchants to run queries on data during a day. What means report service shall handle 4 requests * 5 million merchants a day or 250 TPS or 500 TPS in peak hours
- The system shall have a minimum downtime
- Systems batch processing shall provide metrics for reporting service up to one hour after ingestion
- The system shall persist raw events into a scalable and durable store

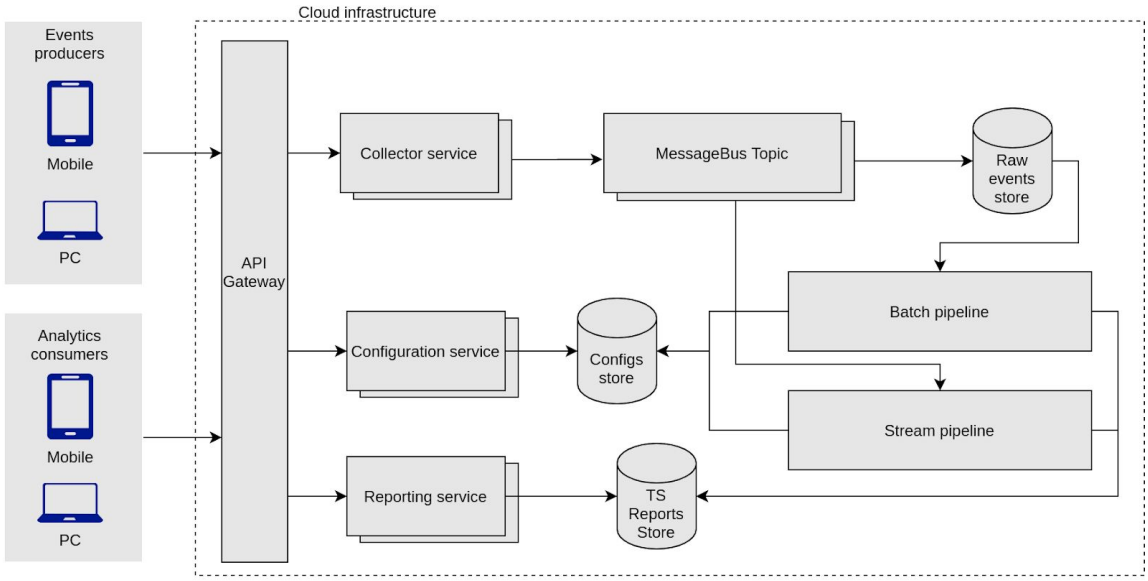
Assumptions:

- Public cloud infrastructure will be used to provide high availability and scalability
- Multi-zone/multi-region deployment will be considered in system design
- The system will utilize long-term, reliable and scalable cloud store for raw data

Out of scope:

- Authorization and authentication
- Personal identification concerns

High-level architecture



Components:

Cloud infrastructure – AWS, GCP, OCI or Azure computational resources and other services will be used to build the system. This allows much easily scale, extend, configure and distribution of the system and its components.

API Gateway – will be used as a single, configurable, scalable entry point to the system that will route external requests to the dedicated services. Can be provided with Apigee, Kong or managed services from a cloud provider

Collector service – light-weight REST service used to transport events to the message bus and able to handle a large volume of requests. Designed and implemented following micro-services architecture in a non-blocking fashion. Distributed in form of Docker images and orchestrated by Kubernetes cluster. Proposed tools are SpringBoot, Akka or Vert.x.

Configuration Service – REST service to configure events processing and custom user reports. Distributed in form of Docker images and orchestrated by Kubernetes cluster. Suggested implementation framework is SpringBoot or Vert.x.

Configuration store – Scalable and read-intensive distributed database to store users configurations. Proposed datastore is MongoDB as it can persist s

Message Bus – Distributed, resilient and high performing message bus used as a central point of integration for all processing components. Can be implemented with Kafka or Amazon Kinesis

Events store – Durable and long-term store of raw events messages to enable reprocessing of event streams in case of fixes, debugging or custom reports. Hbase can be used for this purpose as a distributed database on top HDFS or AWS S3 service can be used alternatively.

Stream pipeline – A pipeline to generate business insight in near real-time and satisfy requirements to deliver data up to one hour. Spark will be used for this purpose and store time-series metrics data into TS Reports store.

Batch pipeline – An additional to near real-time ‘stream’ pipeline used to reprocess historical data from the Event store to correct or run custom user reports. Airflow or Spark or AWS Glue can be used to run custom on-demand processing. Results of such processing will be stored into TS Reports store.

TS Reports store – A scalable time-series distributed database able to efficiently respond to a high volume of queries. InfluxBD or Druid can be a good candidate for this component.

Reports service - REST service to process standard and custom user queries and deliver the report as a time series values collection. Distributed in form of Docker images and orchestrated by Kubernetes cluster. Suggested implementation framework is SpringBoot or Vert.x.