Design Google Analytics like Backend System

Author: Lukas Fam

For this google analytics like backend system design, I prefer microservice architecture rather than a monolithic architecture since microservice architecture is more scalable and reliable. Application built as microservices can be split into several component services thus each services can be deployed/redeployed independently without affecting application integrity. It means that microservice architecture gives developers the freedom to develop and deployed services independently. The main benefit using microservice architecture is if one microservice fails, the others will still work. As for this design concept, I use Java Spring Framework for implementing microservices.

Here my high level architecture design for the backend system.

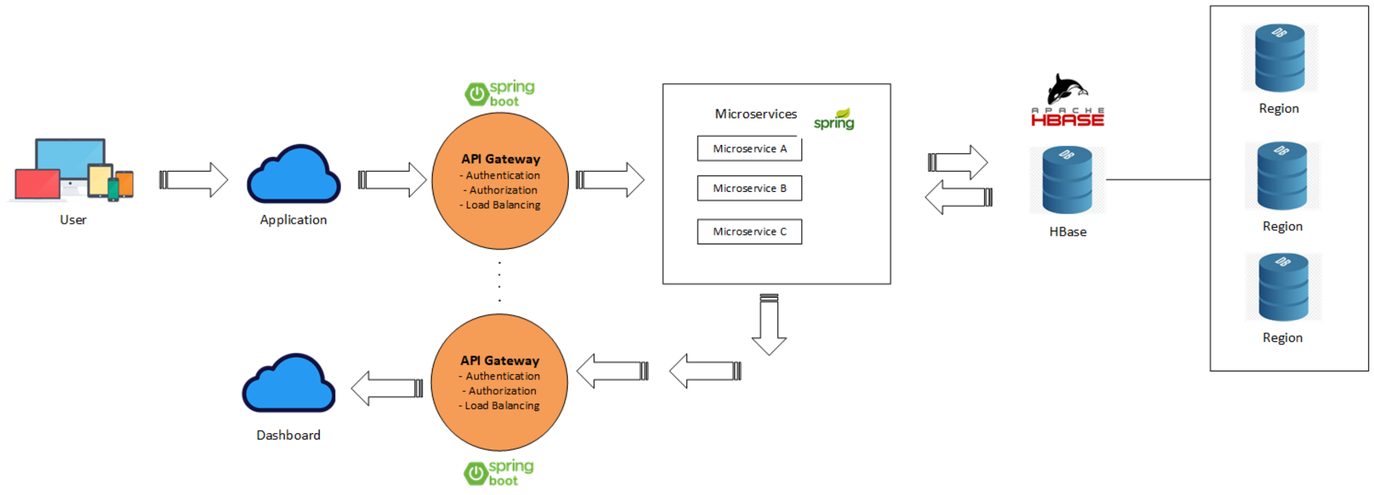

Figure: Architecture for Google Analytics like Backend System.

Component Breakdown

1. User and Application

User access the application through internet by using their devices such as desktop web browser or smartphone (mobile). Every web page or mobile site embed tracking code (or ID) which collects data visitor, for instance session duration, pages per session, bounce rate (percentage of visitors who enter a site and then leave rather than continuing to view other pages within the same site), etc.

2. API Gateway

API gateway coordinates how all the requests are processed in microservice architecture. When an API gateway receives a request, it looks up for the microservice which can serve the request and delivers it to relevant part. Aside from routing task, API gateway can also be a part that performs authentication, authorization, load balancing and other tasks. By adding API Gateway between client and microservices, it simplifies the implementation of distributed system.

In order to use API Gateway, we can use an open source tool to implement this, for example **Spring Cloud API Gateway or Tyk**. These tools are simple yet effective to route to APIs and provide crosscutting concern such as security, monitoring/metrics and resiliency. Spring Cloud API Gateway using Eureka Discovery Server Microservice to route HTTP requests. However, we can also implement API Gateway by ourselves such as authentication (sign in) using JWT (JSON Web Token) and then check if the client has a privilege to access service, monitoring, etc.

3. **Microservices**

Microservices offer increased modularity, making applications easier to develop, test, deploy, change and maintain. The code is broken into independent services that run as separate process.

Scalability is the main key of microservices. Because each service is a separate component, we can scale up a single service without having to scale the whole application. Business-critical services can be deployed on multiple servers for increased availability and performance without impacting the performance of other services. We should be prepared to handle multiple failure issues, such as system downtime, slow service and unexpected responses. Load balancing is really important here. When a failure occurs, the troubled service should still run in a degraded functionality without crashing the entire system. API Gateway will use Circuit Breaker to invoke the failure service.

Each microservice has a task, for example Microservice A's functionality is to track all activities of a user in a site and then store them into database (including session duration, pages per session and how long user stay and switch between pages) and Microservice B's functionality is to fetch data activity users in all pages in a site. Let's say there's already many data (more than one billion) in database and we want to display the overview of today's data. Even though *HBase* is fast for indexing data when we fetch from database for processing data (in terms of querying data), we can still make it more efficient by improving our algorithms for each services. When fetching/querying data from database, we should fetch what we only need, we should avoid fetch unnecessary data. We often fetch/query unnecessary data using ORM (Object Relational Mapping), which is provided by any framework nowadays. ORM by default fetch all data tables with all its relations. With such a big data, when we fetch tables with all its relations, it will take more time to query it. In order to prevent this issue, we should limit all table relations and just fetch the necessary data table.

Generally speaking, there might be a case of which the system has a bug that affecting data processing in analytic dashboard. To handle that such kind of case, we need to implement a service which can correcting the incorrect data. However, it will be difficult to implement if we have incomplete data in order to fix the historical data. Otherwise, we can create a service that correcting the incorrect historical data by uploading the correct data to database (import). If the incorrect data is a large number, we can fix it gradually using *cron-job* until it's done correcting whole data.

4. **Database**

For database architecture, MySQL is definitely impossible to handle billions data daily. So I pick NoSQL for managing big data, precisely HBase (Hadoop Database) since we need to quick look up over billion rows of data. HBase is a distributed database that uses the Hadoop file system for storing data. Let's get an example, Facebook, is one of the famous social media in the world right now and has billion users, uses HBase for their database.

As I explained about execution time for processing big data, actually there is still another factor to affect execution time of processing data beside algorithm and query data. As time goes by, the velocity of data increases more and queries become slower as whole tables need to be scanned. In order to handle this issue, we can do *table partitioning*. By this doing this, there is no need for a full scan, for example we can do table partitioning per date in order to make more efficient execution of processing data.

5. **Analytics Dashboard**

   The request will be routed from analytics dashboard through microservices. Microservices will do processing of time series data and the result will be sent to the dashboard for visualization.

Icon Credits

- Device (User) : https://cdn.dribbble.com
- Application (Dashboard) : https://www.flaticon.com
- API Gateway : https://www.thecuriousdev.org
- Microservices : https://www.opencodez.com
- Apache HBase : https://mapr.com