# Problem Statement and Design Specifications

## • Problem Description

***Design A Google Analytics like Backend System. We need to provide Google Analytics like services to our customers.***

***The system needs to:***

1. ***handle large write volume: Billions write events per day.***
2. ***handle large read/query volume: Millions of merchants want to get an 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.***

## • Summary

Taking the reference to Google Analytics as a functioning model we need to design a system that fulfills the company's requirement of tracking all the data flowing through their network and

_______their websites for any given user(s). Basically what GA does is categorized into the following things. We will be designing a System on the same aspects and will be elaborating on what each component is meant to do.

1. Data Collection
2. Data Configuration
3. Data Processing
4. Report Generation

### Data Collection:

Whenever a user browses over the internet by means, the user is basically being tracked by a clientside script which can be embedded in the website. This will track all the user's data and then this data will be collected in terms of various attributes. Each time a user navigates on a website, clicks links on the website, watches a video on the website, redirected from one page to another and so on, it is considered to be a HIT. A hit could also be called as an EVENT. Every time the user interacts with the website a hit request is generated. A hit gathers all of the information about the interaction at that exact particular moment – a snapshot of information, and sends this information to the collection server which does the work of collecting such important and uncategorized data.

### Data Configuration:

Data Configuration can be thought of as the settings you apply to customize the data being collected. It's about setting up the rules for data processing. Under the hood, you may want to include various features to process that data according to market conditions. Different properties can also be set to categorize the data into meaningful sections. The data configuration can vary depending upon the intent and goals of the System. This would again be another microservice-based backend service that we will have in the whole ecosystem.

### Data Processing:

Once the data has been captured by the data collection backend services, this module of data processing decides the future of that data. What all data needs to be taken in and what all data needs to be omitted is decided by this module. Taking help of Data configuration unit this module will filter out the data for further processing and finally storing it into a database for future references. This module will be a very complex module of all and will need to be intelligent enough to process humongous data and transform into meaningful chunks. The collected data would be categorized in the following four characteristics.

1. User-level (related to actions by each user)
2. Session level (each individual visit)
3. Pageview level (each individual page visited)
4. Event level (button clicks, video views, etc)

### Report Generation:

An important module from the perspective of a Service Provider. The service provider would want to see the processed data in the form of meaningfully related Reports. This module would be another micro-service which would be responsible for generating various reports as per standards, business industry, business growth. Basically understanding the consumer behavior, creating patterns out of it. Using some configurational data and processed data reports would be generated. The reports could also be categorized into the following two sub-sections.

1. User Acquisition Data: report containing data about your users *before* they visit your website
2. User Behavior Data: report containing data about your users *when* they visit your website

## • Solution Concept

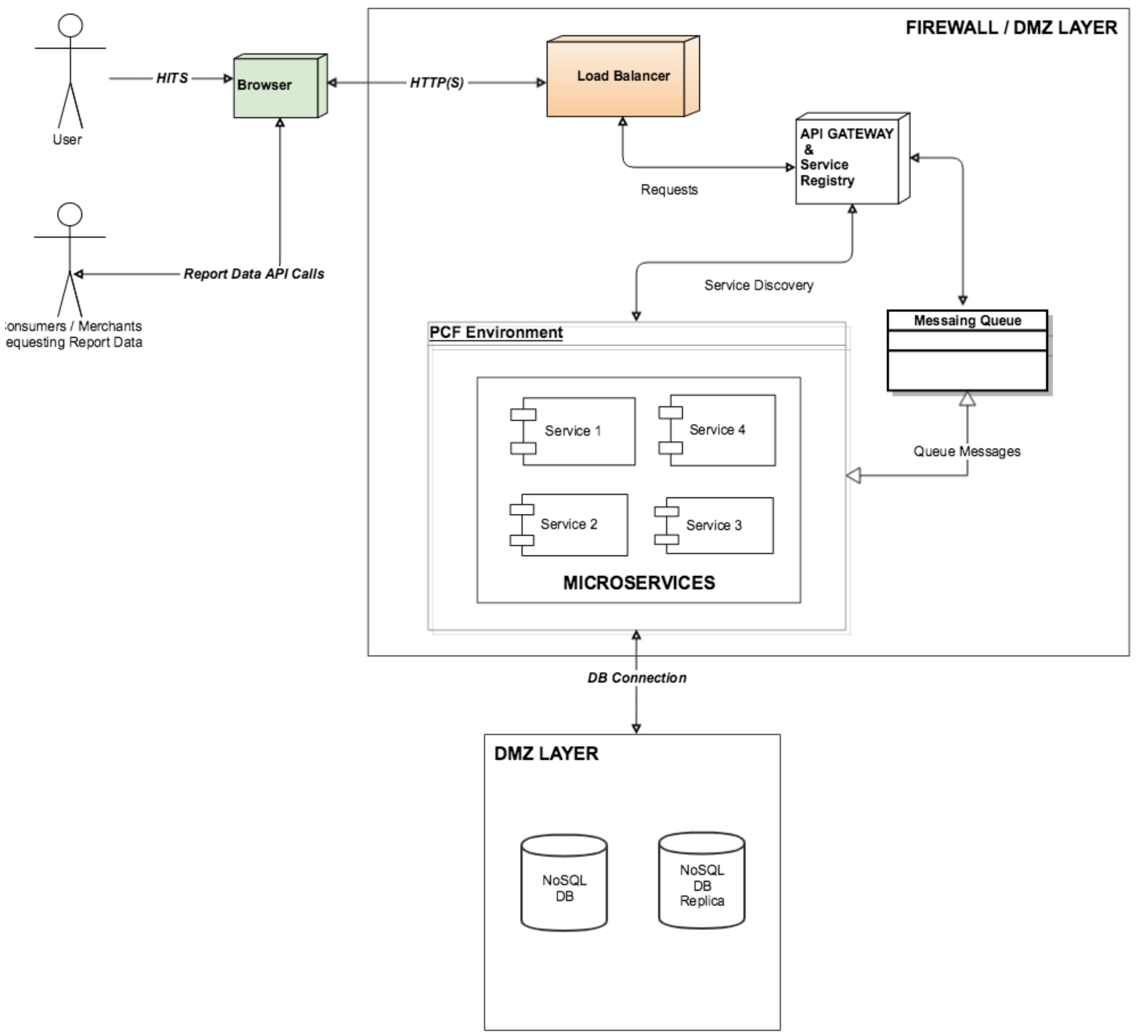
The solution would comprise of a Backend System which would be a combination of **distinct microservices**. These microservices can talk to each other using **cross-service communication**. Each microservice will have its **own DB and each DB will have its own schema**. **NoSQL database** would be a preferred choice considering the huge amount of Data. **Service registry or Service Discovery** would manage all the microservice's discovery or registry. **PCF would be the Cloud platform to be used as Platform As A Service(PaaS).** PCF would also allow the microservice to be scaled horizontally in case of higher traffic or extreme load. **Messaging queues** would be an important part of this ecosystem and would be used wherever necessary. **Load Balancers** will do the work of offloading the traffic to various nodes to keep the system balanced.

## • Backend Micro-Services

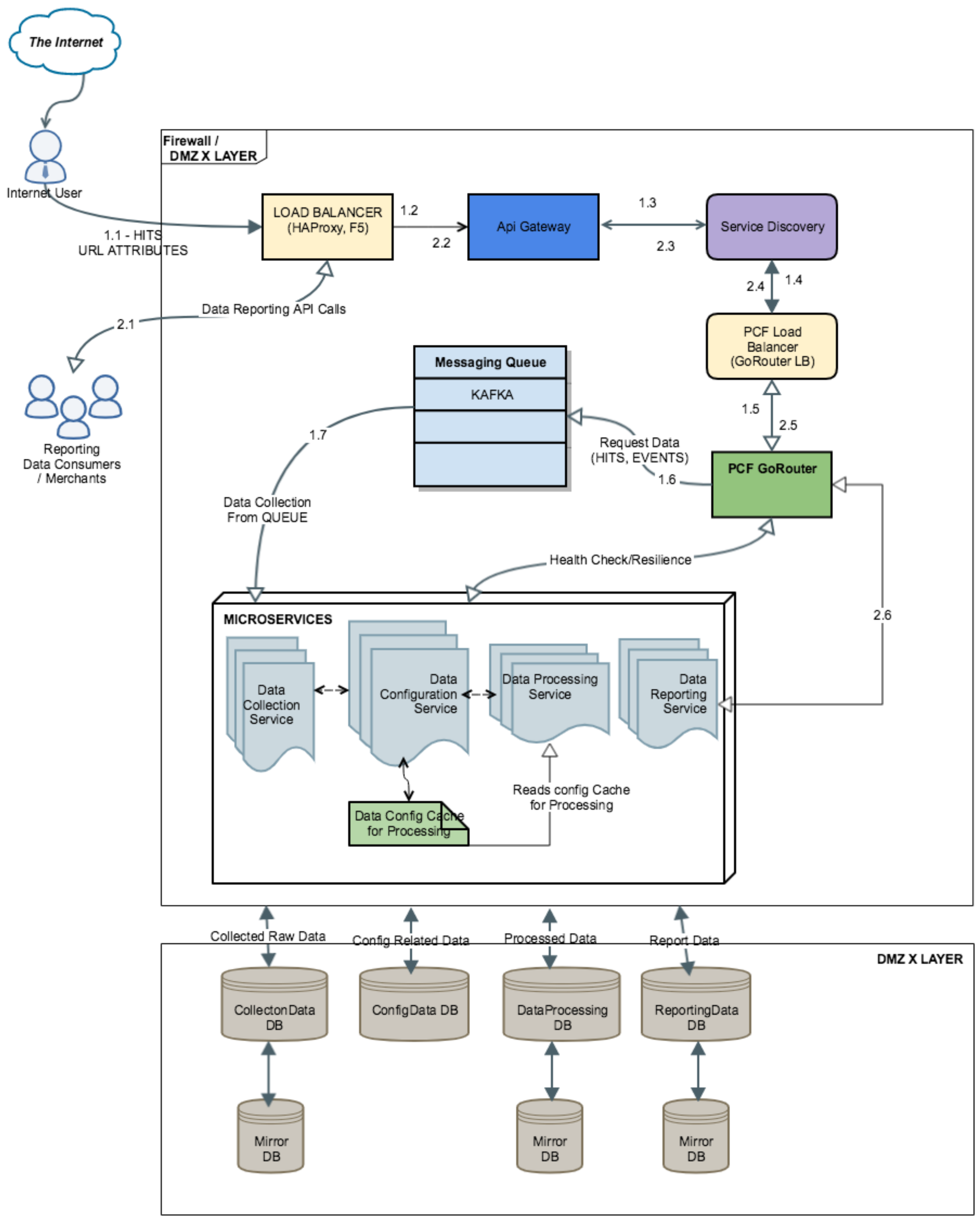
1. **Data Collection Service**
2. **Data Configuration Service**
3. **Data Processing Service**
4. **Report Generation Service**
    1. **User Acquisition Data Report Service**
    2. **User Behavior Data Report Service**

## • High-Level Diagram (HLD)

High Level Diagram



- Technical Architecture Diagram (TAD)



## • Architectural Components Breakdown:

The components from TAD can be broken down into several functional units.

### 1. Web User/Visitor Tracking Code

We need to embed tracking code that collects data about the visitor. It loads an async script that assigns a tracking cookie to the user if it is not set. It also sends an XHR request for every user interaction.\

## 2. Load Balancer

HAProxy/F5, is a popular software TCP/HTTP Load Balancer and proxying solution. Its most common use is to improve the performance and reliability of a server environment by distributing the workload across multiple servers. HAProxy routes the requests coming from the Web/Mobile Visitor site to the Zuul API Gateway of the solution. Given the nature of a distributed system built for scalability and stateless request and response handling, we can distribute the Zuul API gateways spread across geographies

## 3. API Gateway

Spring Boot & Netflix OSS Eureka + Zuul

Zuul is an API gateway and edge service that proxies requests to multiple backing services. It provides a unified “front door” to the application ecosystem, which allows any browser, mobile app or another user interface to consume services from multiple hosts. Zuul is integrated with other Netflix stack components like Hystrix for fault tolerance and Eureka for service discovery or use it to manage routing rules, filters and load balancing across your system. Most importantly all of those components are well adapted by Spring framework through Spring Boot/Cloud approach. An API gateway is a layer 7 (HTTP) router that acts as a reverse proxy for upstream services that will reside in our your platform. API gateways are typically configured to route traffic based on URI paths.

Eureka server will act as a registry and will allow all clients to register themselves and could be used for Service Discovery to be able to find IP addresses and ports of other services if they want to talk to.

## 4. Spring Boot Microservices

Using a microservices approach to application development can improve resilience and expedite the time to market. Microservices offer increased **modularity**, making applications easier to develop, test, deploy, and, more importantly, change and maintain. With microservices, the code is broken into independent services that run as separate processes. Scalability is the key aspect of microservices. Because each service is a separate component, we can scale up a single function or service without having to scale the entire application. When a failure arises, the troubled service should still run in a degraded functionality without crashing the entire system. Hystrix Circuit-breaker will come into rescue in such failure scenarios. The core processing logic of the backend system is designed for **scalability, high availability, resilience and fault-tolerance** using distributed Streaming Processing, the microservices will ingest data to **Kafka Messaging Queues**.

## 5. Apache Kafka for Messaging

Apache Kafka is used for building real-time streaming data pipelines that reliably get data between many independent systems or applications. It allows Publishing and subscribing to streams(messages) of records. Secondly Storing streams /messages of records in a fault-tolerant, durable way. Kafka uses Zookeeper to store metadata about brokers, topics, and partitions. Kafka Streams is a pretty fast, lightweight stream processing solution that works best if all of the data ingestion is coming through Apache Kafka.

## 6. PCF - Platform as a Service for Cloud

Pivotal Cloud Foundry is the best choice for managing your microservices. Pivotal gives you the multi-cloud platform that has many inbuilt features like

- Application portability.
- Application auto-scaling.
- Centralized platform administration.
- Centralized logging.
- Dynamic routing.
- Application health management.
- Integration with external logging components like Elasticsearch and Logstash.

## 7. NoSQL DB: Apache Cassandra

Apache Cassandra as storage for persistence high volume data. Apache Cassandra is a highly scalable and available distributed database that facilitates and allows storing and managing high velocity structured data across multiple commodity servers without a single point of failure.

The Apache Cassandra is an extremely powerful open-source distributed database system that works extremely well to handle huge volumes of records spread across multiple commodity servers. It can be easily scaled to meet a sudden increase in demand, by deploying multi-node Cassandra clusters, meets high availability requirements, and there is no single point of failure. **Apache Cassandra has best write and read performance.**

Characteristics of Cassandra:

- It is a column-oriented database
- Highly consistent, fault-tolerant, and scalable
- The data model is based on Google Bigtable
- The distributed design is based on Amazon Dynamo

> ***In addition to the above components, we can use in-memory cache for maintaining Data config which can be used by the Data Processing unit while transforming the Data Collected from the Data Collection Unit.***
>
> ***Data processing Unit can connect to this cache and listen for any updates in the config to do the data transformation with the latest configuration.***
>
> *Addition to the Reporting Service we can have a Dashboard which will connect with this Reporting Service and will display the analytical data into the Analytics Dashboard using various techniques and configuration. We could add another Reporting Configuration Service and DB for improving Reporting Service's capabilities.*