## IoT System Diagram

Uber ride and user system architecture -

**Maps Creation ETA Calculation**

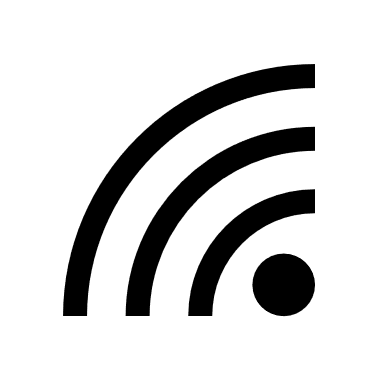
**ML Fraud Detection**

**Analytics**

**Price Surging**

**Kafka Rest API**

**JSON over HTTP vis TCP/IP**



**Receives Cab GPS data every 4 sec**

**Apache Kafka**

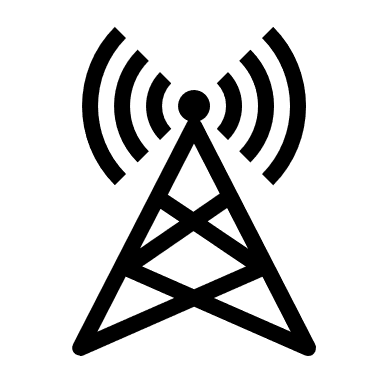
**Cellular Network**

**Batch Processing/Stream Processing**

**CAB**

**GPS (Global Positioning System)**

**Active cab location data**



**Google S2 Library**

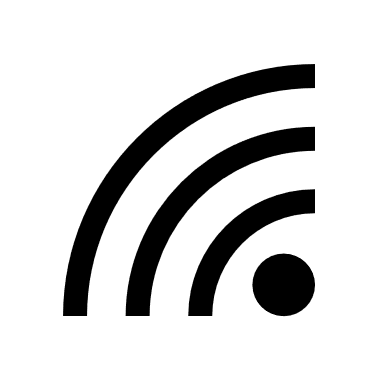
**Supply Services**

**Mobile Application**

**Load Balancer**

**WebSocket**

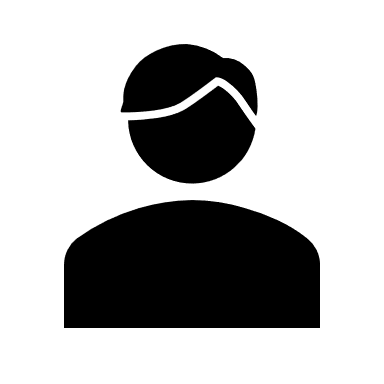
**Provide ETA to Supply**



**Multi Region Datacenters**

**Dispatch System (DISCO)**

**Web Application Firewall**



**Demand Services**

**HTTPS via TCP/IP**

**Cellular Network**

**Wi-Fi Network**

**Identify nearby cabs to server the request, calculates ETA**

**HTTP via TCP/IP**

**User**

**Request for ride**

**GPS (Global Positioning System)**

**Mobile Application**

**Cloud Database (Having several regions)**

**RDBMS**

**No SQL Database**

In the above diagram, Cab is the supply and User is the demand where the User request the Driver.

### System Components –

**Client applications** –

* Users and Uber Drivers communicate with Uber platform through mobile applications installed on their smartphones. Mobile devices connect to the internet using Wi-Fi or cellular network which allows real time communication. Mobile applications communicate with Uber's backend servers using HTTPS protocol, ensuring secure communication over the internet.
* Supply services (Cabs) will be tracked by geolocation. All active cabs send the location to the server once every 4 seconds. The accurate GPS location is sent to the data center through Kafka’s Rest APIs.

Web Application Firewall- This is the security layer which block the requests from the blocked IPs, Bots, and regions which is not supported.

Load Balancer – Various types (Layer3, 4, 5 ) are used to distribute the load

**Backend System** – Dispatch System (Dispatch Optimization/DISCO)

Web Socket - The requirement here is that we should have connection established between the Cab Application to the Server or The User to the Server. Web Socket keeps the connection opens for all of the Uber Application and based on the changes that happen in the DISCO or any component in the server the data will be exchanged between the Application and the Server. Mainly written in NodeJS (Asynchronous and event-driven framework).

Dispatch System – It is mainly written in NodeJS. So that server can send/push the messages to the Application whenever it wants. When a user requests a ride, the request is directed to the Web Socket, which then forwards it to the Demand Service. The Demand Service assesses the ride requirements and communicates them to the Supply Service along with relevant information like ride type, quantity, and location. Utilizing Consistent Hashing, location data is evenly distributed among servers. The Supply Service, equipped with the user's location (cell ID), queries one of the servers in the server ring. It calculates ETA values to identify nearby cabs for the rider. Subsequently, the Supply Service notifies cabs via Web Socket, prompting them to accept the ride request. Upon acceptance by a driver, the trip is assigned to both the rider and driver.

**Databases** – Uber's database architecture prioritizes horizontal scalability, enabling seamless addition of capacity by incorporating more servers. Given the frequent updates from cabs every 4 seconds, the system must efficiently handle massive volumes of reads and writes without downtime, ensuring high availability across all operations.

* Redis for both caching and queuing. Some are behind Twemproxy (which provides scalability of the caching layer). Some are behind a custom clustering system.
* Uber uses Schemaless (built in-house on top of MySQL), Riak, and Cassandra. Schemaless is for long-term data storage. Riak and Cassandra meet high-availability, low-latency demands.
* MySQL database.
* Uber is building their own distributed column store that’s orchestrating a bunch of MySQL instances.

**Analytics** – Kafka seamlessly integrates historical and real-time data streams. Once data expires from Kafka, it's archived into Hadoop for long-term storage. Additionally, Uber indexes this data into an Elasticsearch stack, facilitating efficient searching and visualization. Elasticsearch, along with tools like Kibana and Grafana, performs log analysis for insights.

Uber employs various tools and frameworks for comprehensive analyses, including:

* Track HTTP APIs
* Manage profile
* Collect feedback and ratings
* Promotion and coupons etc
* Fraud detection
* Payment fraud

Ref - <https://medium.com/nerd-for-tech/uber-architecture-and-system-design-e8ac26690dfc>

<https://www.geeksforgeeks.org/system-design-of-uber-app-uber-system-architecture/>