Uber System Design

Architecture and Micro services

Uber - Architecture

Moving from Monolithic Service to Micro Services

	Monolithic Service	Micro Services
Productivity, when teams and codebases are small	High	Low
Productivity, when teams and codebases are large	Low	High (Conway's law)
Requirements on Engineering Quality	High (under-qualified devs break down the system easily)	Low (runtimes are segregated)
Dependency Bump	Fast (centrally managed)	Slow
Multi-tenancy support / Production-staging Segregation	Easy	Hard (each individual service has to either 1) build staging env connected to others in staging 2) Multi-tenancy support across the request contexts and data storage)
Debuggability, assuming same modules, metrics, logs	Low	High (w/ distributed tracing)
Latency	Low (local)	High (remote)
DevOps Costs	Low (High on building tools)	High (capacity planning is hard)

Table 1 : Comparison Monolithic & Micro Services(source https://tianpan.co/notes/120-designing-uber)

Uber – System Design

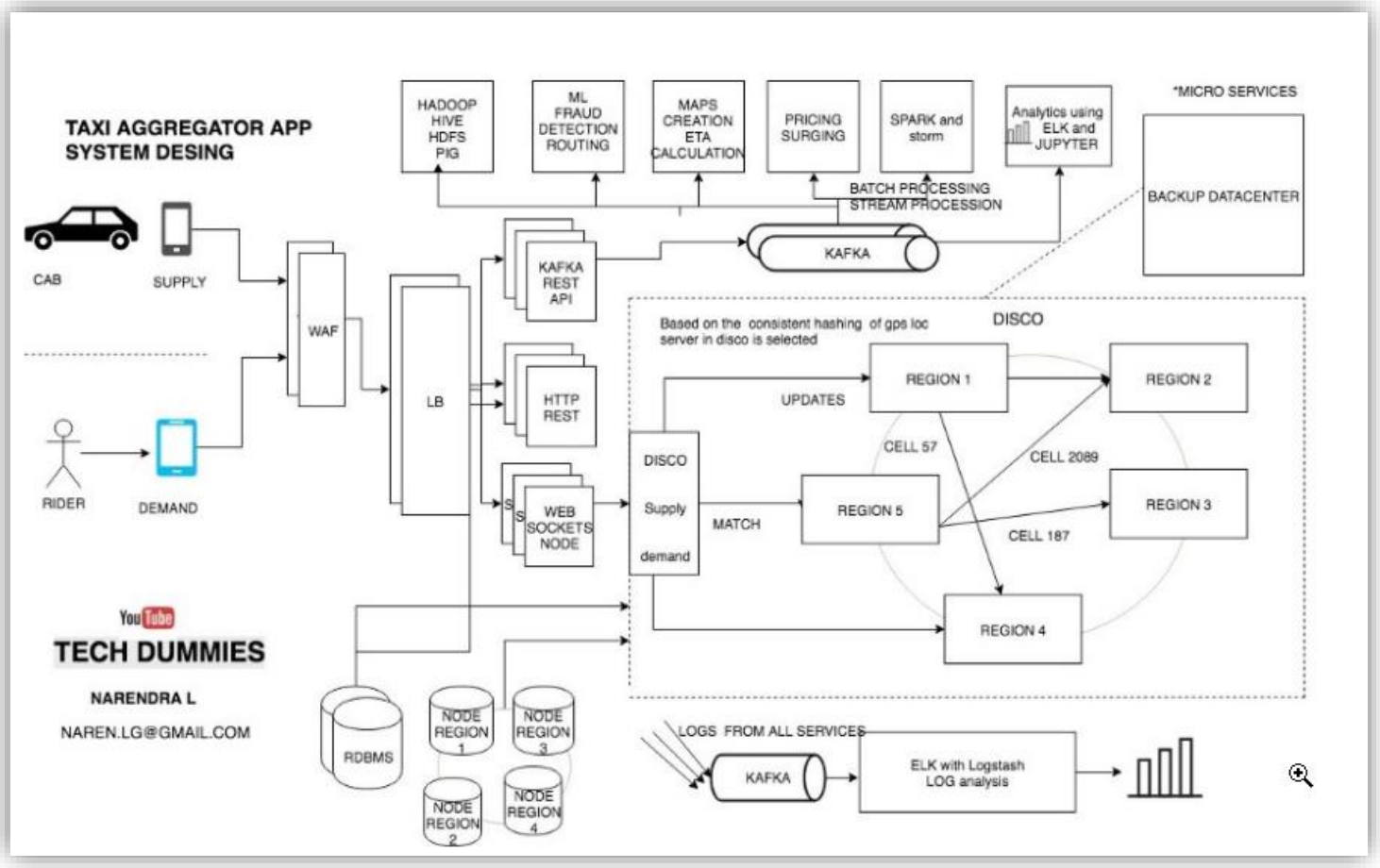
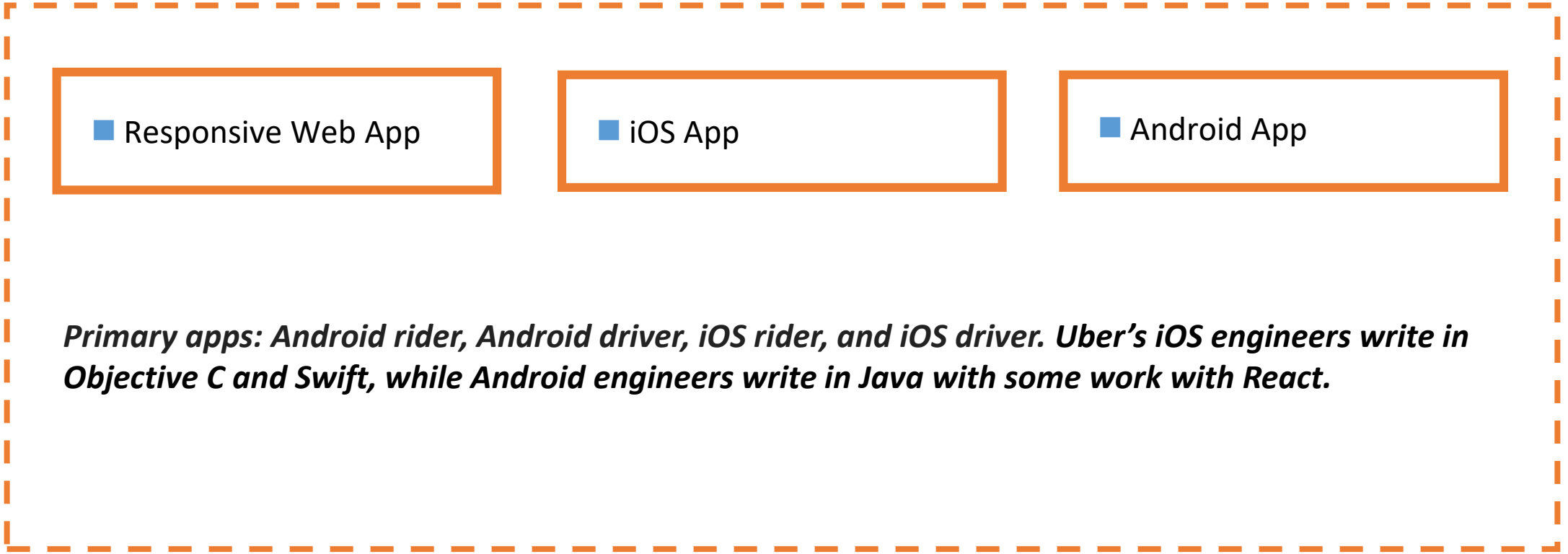


Figure 1 : Comparison Monolithic & Micro Services(Sources from https://medium.com/@narengowda/uber-system-design-8b2bc95e2cfe)

Uber – Application



Uber – Core Services

- **Supply services (Drivers)**
  - Track cars using (lat and lang : active cab will keep sending lat-long to server every 5 sec
  - State machines of supply was kept in memory
  - Attribute set : number of seats, type of vehicle, available car seat for child and etc.

- **Demand services (Customer)**
  - Track GPS location of user le requested the service
  - Special requirement: car size big/small or pooling
  - Demand requirement to match with supply

Uber –Micro Services

Example of supporting services related to main process flow that connected using KAFKA Cluster

- DISCO - Dispatch Optimization Service
  - consistent hashing sharded by geohash.
  - data is transient, in memory single-threaded or locked matching in a shard to prevent double dispatching

- Trip Service
  - Must have low latency with caching mechanism

- Payment Service
  - Must have async design due to payment system have long latency

- User Profile Service
  - to serve users in increasing types (driver, rider, restaurant owner, eater, etc)

- KAFKA cluster
  - As logging: for services interaction with each other and to mobile devices.
  - This interaction needed for internal usage like messaging system, geo-replication and stream processing
  - The data then been achieved into Hadoop before it expired in Kafka
  - This data is also ingested in real time by various services and indexed into an ELK(Elastisearch, Logstash & Kibana) stack for searching and visualizations.

Uber –Micro Services

Example of supporting services not related to main process flow that connected using KAFKA Cluster

- Collect ratings
  - Rate the driver and services

- Send emails
  - Online billing, services info or discount to customer

- KAFKA cluster
  - As logging: for services interaction with each other and to mobile devices.
  - This interaction needed for internal usage like messaging system, geo-replication and stream processing
  - The data then been achieved into Hadoop before it expired in Kafka
  - This data is also ingested in real time by various services and indexed into an ELK(Elastisearch, Logstash & Kibana) stack for searching and visualizations.

- Update databases
  - Store in RDBMS and moved to NoSQL DB to ensure horizontal scaling, manage different regions on which the services was been established.
  - **Data then send to Hadoop via Kafka Cluster for analytics**

- Schedule payments

Uber – Availability & Scalability

- **Load Balancing set on various layer to ensure stable load**
  - Application Server Load
  - DNS & IP Load
  - IP address Load

- **Regional Node Setup**
  - To ensure low latency for user from various region.

- **Databases**
  - Set to Primary DC and Secondary DC as backup
  - With backup datacenter for disaster recovery solution
  - **Ringpop used as cooperative distributed systems** It gives the high-availability, partition-tolerant properties of distributed databases like DynamoDB or Riak to developers at the application level.

References

- https://medium.com/@narengowda/uber-system-design-8b2bc95e2cfe
- https://eng.uber.com/tech-stack-part-one-foundation/
- https://www.linkedin.com/pulse/uber-system-design-demysified-rajesh-s/
- https://tianpan.co/notes/120-designing-uber
- https://eng.uber.com/tech-stack-part-one-foundation/
- https://eng.uber.com/uber-tech-stack-part-two/
- https://www.quora.com/Systems-design-What-is-the-software-design-of-the-Uber-App
- https://medium.com/@narengowda/uber-system-design-8b2bc95e2cfe
- https://www.youtube.com/watch?v=umWABit-wbk
- https://www.youtube.com/watch?v=J3DY3Te3A_A
- https://www.linkedin.com/pulse/uber-system-design-demysified-rajesh-s

WebSockets	Send and receive messages, so anytime client can send the message to server or server can send and whenever it wants to
Ringpop	Ringpop is a library that brings cooperation and coordination to distributed applications. It maintains a consistent hash ring on top of a membership protocol and provides request forwarding as a routing convenience
Apache Kafka	Is used as the data hub supply or cabs uses Kafka's APIS to send there accurate GPS locations to the datacenter.
HADOOP	Analytic Soultion once data collected from RDBMS
DISCO - Dispatch Optimization	Dispatch system, ends a request to geo by supply to match riders to drivers or just display cars on a map