HLD for Swiggy--

The major functional requirements that we will be covering are :

**1.** Restaurant On Boarding.

**2.** Food and Menu (Uploaded by Restaurant, Approved by App Admin)

**3.** Dynamic Pricing (Add delivery cost, discounts, promos, etc.)

**4.** Door to Door Delivery (Driver selection, continuous location update of order, etc.)

**5.** User Registration, Roles and Security

**6.**Search and Recommendations based on user activity, generated orders, user locations and ongoing promos.

**7.** Notification (Notify customer for every event).

**8.** Integration with Payment Gateway.

**9.** Reviews, Ratings and Surveys.

**10**. Order handling, cart and Checkout.

**11.** Customer Support and improvements.

**12.** Design that can be enhanced to support B2C E-Com Application.

**13. S**ales Report

**14.**Billing and Documents

**Non Functional Aspects**

1. System can support more than million orders and x10 search requests.

2. SLA for search should be <500 ms along with recommendations.

3. No data loss for transactional data.

**Design decisions and Technology choices**

**1.** **Cloud platform :** Hybrid cloud. AWS and native cloud (kubernetes) with direct connect between AWS and on-prim Data Centre. Cloud services used from**AWS cloud are SQS, SNS, AWS Lambda, AWS ADM, VPC, etc.**to support the notification system.

**2.** **Kafka Streams :** used to achieve high throughput in streaming events between read and write repositories of order management, sending real time or batch events to recommendation engine and publishing approved catalog and products(menu and foods) to elastic index.

**3.** **RabbitMQ :**To achieve transactions between distributed microservices and avoid data/transaction loss

**4.** **Apache Spark/Beam :**For running map-reduce jobs to select driver for efficient delivery and generate recommendations based on continuous streams of user activity, order activity, pricing and promo changes, etc.

**5.** **Clickhouse DB :** All the processed data will be stored in intermediate DB which can deliver good performance in high throughput(both read and write). Cassandra could have been a choice but because of its low read performance while reading the data from multiple SS tables in case of huge datasets didn’t make that a efficient candidate for the purpose. Aerospike could have been the second choice but as this is processed data and we are looking to store this in relational form to generate queries, we settled for this.

**6.** **Aero Spike DB :**To support high throughput in read and write in real time.

**7.** **Redis Cache :**To store the location and segments information of all the restaurants in the city. As this is very frequently read and non-changing data, we had chose to setup this in Redis to avoid multiple DB calls. For cart also, we will be storing all the selected items in Redis.

**8.** **Elastic Search** : Elastic indexes are used to store all published catalogs and products along with recommended items and placed order queries to achieve the targeted SLA for search and browse requests. Elastic search will also be used for indexing customer calls records and metadata for the transcription.

**9. PostgresDB :**To Store all transactional data. We have chosen PostgresDB to support native cloud approach but other options like public cloud managed storages can also be considered to achieve high scalability.

**10. GraphQL :**To achieve the targeted SLA, we have designed the system to support different repository for read and write, and have made the write repository more aligned towards domain data and read repository towards data that will be requested from UI. To do so, we have bundled all the product and order related data into their respective documents inside the elastic indexes. Now to decrease the load and send only relevant data to UI, we have used GraphQL query language to query selective data from the tree instead of complete data. Same goes for recommendation engine as well.

**11. Web Sockets :** Will be used to deliver the continuous update of driver location who had picked the delivery, to the customer.

**12. Rest API(s)**: Other the cased specified above for web sockets and graphQL, all other request will be rest api(s).

**13. MongoDB :**To store review, ratings, notifications and surveys related data that doesn’t have a fixed schema and changed overtime.

**14. Containers :** Spring Boot/Web flux based services.

**15. Cloud Storage :** for Blob and file storage storages like transcriptions, documents, reports, templates, etc.

**16. NodeJS :**Light weight API(s) exposed to perform quick and efficient IO operations and support scalable traffic with controlled memory. Java Reactive/Spring Webflux can also be considered as alternate

**17. Map API :**Map API will provide the city information and segments in which the restaurant is located. Also that can be used to determine the possible routes between restaurant from where the food is ordered and where the customer is located.

**18. Config Server :**For Externalization of properties/configurations.

**19. API Gateway :**Exposing the proxy API(s) of external systems. 3 different API gateways will be exposed with different level of security and rbac checks.

**20. Kubernetes**(Hosted on Public Cloud or On-Prim) : Kubernetes cluster will be deployed in On-prim system or over public cloud and all the container services will be hosted on the cluster. For Infra, terraform is preferred for inter portability of infra services and components between different clouds. Containers are docker based.

**21. Service Mesh :** for Inter service communication, certificate management, service discovery, version management and rate limiting (if required).

**22. Load Balancer (Network and Service) :**Network load balancer will be doing using the URL based routing to a service and service load balancer will be redirecting the network traffic to deployed service instances according to the configured mode(round robin or load based).

**23. Network Firewall :**External firewall placed in top of API gateway to secure internal infrastructure.

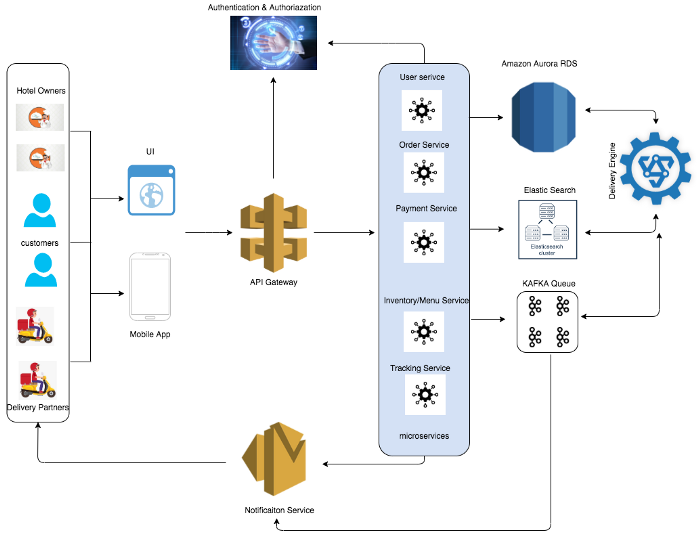
**24. Active Directory :**Will be storing all registered user credential and RBAC to manage authentication and authorization.

**25. DevOps Tools :** Terraform (infrastructure management), Vault (Secret Management), Jenkins (CI/CD), Docker, Kubernetes, AWS, Datadog and Prometheus (Monitoring and metrices), ELK/Splunk (logging)

**26. Amazon Athena/Hive**: To query the logs and transcriptions stored in the blob/cloud storage.

**27. CDN** : Akamai/ Amazon CDN for static content and cache

**28. Rule Engine :** For configuring pricing and transcription rules.



# Core Features of Customer App

* **Searching menu:**Allow your users to search for different restaurants, cafes by location, and cuisines. Using the search filter, users can easily find their favorite eating places, list menu, offers, etc.
* **Order placement:**The user can place an order of selected dishes and food. They just need to cross-verify their preferred dish, delivery time, and proceed check-out.
* **Tracking Delivery Partners:**With real-time tracking features, it becomes easy for users to track delivery drivers and know their real-time location information. Users can check the time taken by the food delivery executive to deliver their parcel.
* **Payment gateway integration:** You provide the users with multiple payment options like credit/debit cards, different wallets like Google Pay, Paytm, Phonepe, UPI, etc

# Core Features of Delivery Partner

* **Delivery Partner’s profile:**Through this feature, a driver can keep his profile update. It contains his full name, address, email, contact number, photo, and other personal information.
* **Notification for orders:**Through push notifications, drivers can get constant updates & alerts for new orders. It will help in the accurate delivery service of your restaurant.
* **Map for the delivery route:** Integrate Google Map or other providers and allow drivers to choose the shortest and fastest routes to reach the location.

# Core Features of Food Partners/ Restaurants

* **Restaurant Profile/Menu:**Through this feature, a restaurant owner can add their restaurant details, menu and its availability, price, preparation times, etc
* **Notification for orders:**Through push notifications, Restaurants can get constant updates & alerts for new orders. It will help in the accurate delivery service of your restaurant.
* **Notifications for Pickup Partners:**They will get alerts about delivery partners, their location when they will pick up, etc.
* **Payment Details:**Information about the payment received from the food delivery system for their orders

# Core Features for Food Ordering System Admin

* **Restaurant management:**Being on the admin panel, one can directly manage all the restaurants by adding, updating, and removing any eating joint from the list. He can also check active restaurant status.
* **Analytics & report generation:**Using analysis and report features, you can get real-time insights of reports and other accounting information which helps you to identify the growth and opportunities to expand reach.
* **Monitoring every action:**Monitor all the food orders, delivery partners, deliveries, reviews & ratings of delivery partners, cancelled orders, and other important data related to the driver’s performance.

# **Application Flow**

We are considering here microservices-based architecture. Different services are listed in the architecture diagram

1. All requests made from a mobile app or UI will go to different services via the API gateway. API gateway will take care of load balancing and routing requests to services. This will authenticate and authorize the user and send back the token ID. This token is used for further communication
2. Different services like, user registration and management service, order service, payment service will use transactional databases. We will use the Amazon Aurora relational database. This is a highly scalable database service to manage users and concurrent orders etc.
3. Information about different restaurants, their menu, price, offers, etc will be stored in JSON document storage in Elastic Search. We can use a multi-node cluster here. Whenever a customer searches for a menu/cuisines it will be fetched from elastic search. Elastic search provides fast scalable search options
4. Once the user selects the dishes and quantity from the restaurant. He will go to the checkout option and then do payment. Different payment gateways and payment options are integration with the system and upon successful payments, the order is successfully placed
5. Once the order is placed all the information is sent to the central message Queue like Kafka. The order processing unit reads the order info and then notifies the selected restaurant about the order. At the same time, it searches for available delivery partners to nearby locations to pick up the order. It also gets the information like preparation time from the restaurant and estimated pickup time from the delivery partner based on his location and other details. it will select the best available delivery partner and he is notified about order and restaurant details
6. The user gets push notification about the order. The order processing and tracking service will work together and the user can track their order status, live location of the delivery person, etc
7. Delivery person pickup order and deliver to customers. Customer is real-time notified with ETA for the order.