High Level Design Document For

Hotel Reservation System

Table of Contents

[1. Customer User Journey 1](#_Toc188905161)

[2. Admin User Journey 1](#_Toc188905162)

[3. Functional Requirements 2](#_Toc188905163)

[4. Non-Functional Requirements 2](#_Toc188905164)

[5. Some of the APIs 3](#_Toc188905165)

[6. Choosing the right Database 4](#_Toc188905166)

[7. Architecture Diagram 5](#_Toc188905167)

[8. ER diagram 6](#_Toc188905168)

[9. Different Services and their goals 6](#_Toc188905169)

[10. Booking Flow 7](#_Toc188905170)

[11. Scenarios to address 8](#_Toc188905171)

[12. Additional Design Consideration 9](#_Toc188905172)

## Customer User Journey

1. Search for the hotels
2. Fetch hotels details
3. Fetch room details
4. Create a booking order for room
5. Payment
6. Fetch bookings

## Admin User Journey

1. Add/remove/update a hotel
2. Add/remove/update a room
3. Fetch bookings

## Functional Requirements

Users

1. User can search hotel in the platform.
2. User can get hotels details (like querying location as Bhubaneswar, then list of hotels should come up)
3. User should get the room details (query with any tag like summing pool or king size bed etc.)
4. Create a booking/reservation of the room
5. Make a payment (with Time to Live)
6. Get booking details in UI, SMS or email notification

Admins

1. Create /update/delete hotels details.
2. Create/update/delete room details.
3. Fetch the booking details based on hotel id, date, user email or etc.

## Non-Functional Requirements

1. High Concurrency, High Available and High Consistent
2. Low/Moderate latency

**Estimations and Constraints**

5,000 hotels with 1 million rooms (stats provided)

User will be reading multiple times from platform but write once while booking a room.

**Assumption**

Occupancy in any day = 80 %

Average booking = 4 Nights

Seconds in day = 86400 seconds (for the ease of calculation) = 10^5

Transaction per second= (10^6 \*0.8)/ (4\*10^5) = **2 TPS** (Transaction per second).

For each booking there will be 100th times of search

Query per seconds = **200 QPS** (Query per second)

* Create order will have **200 QPS** (read) and transaction will have **2 TPS** (write)

## Some of the APIs

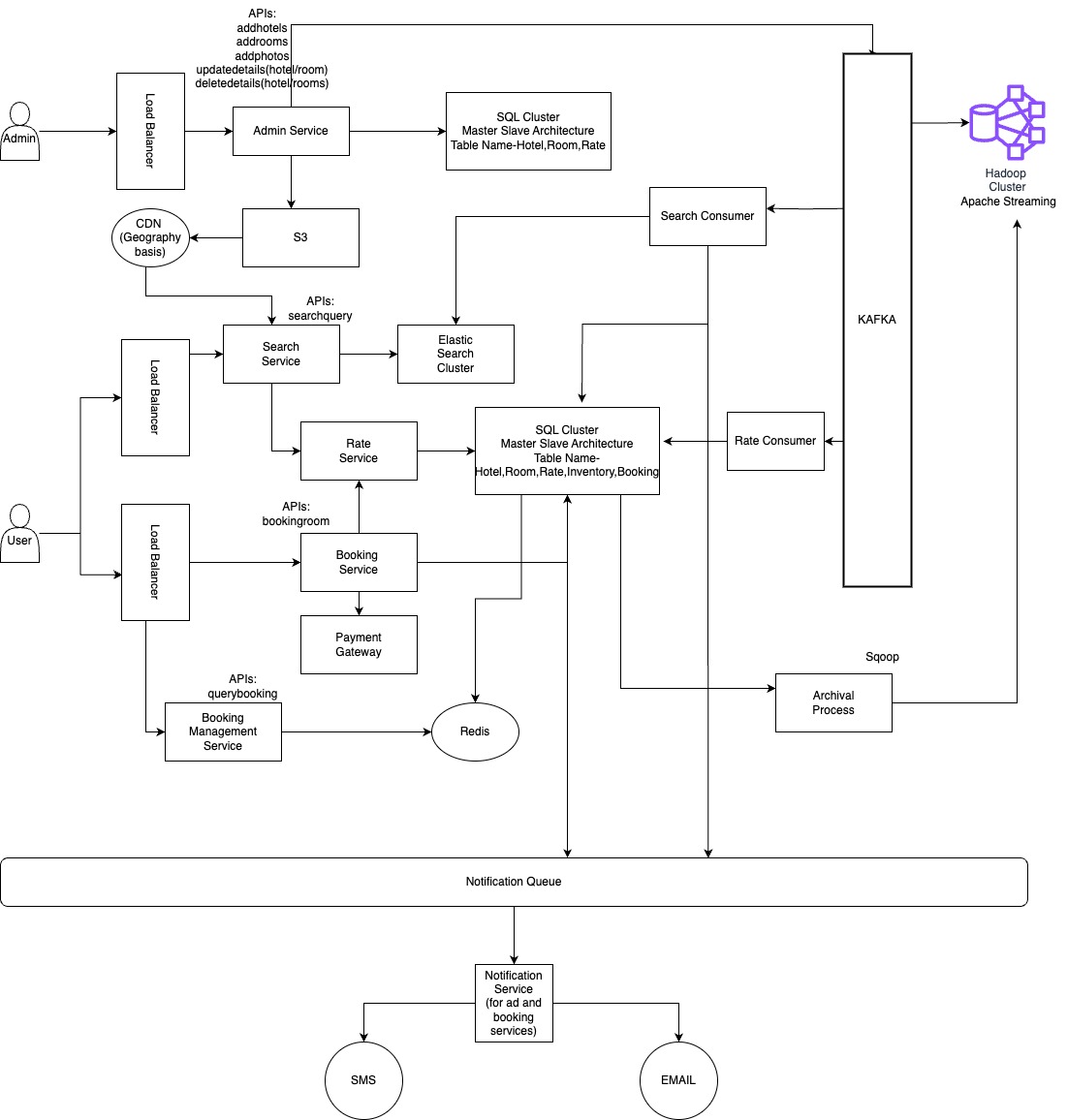
1. **Hotel APIs**
   1. POST <platform\_name>/hotels ---creation of hotels (Admins Service)
   2. GET <platform\_name>/hotels/hotel\_id ---searching of hotels (Search Service)
   3. PUT <platform\_name>/hotels/hotel\_id ---updating of hotels details (Admins Service)
   4. DELETE <platform\_name>/hotels/hotel\_id ---deletion of hotels (Admins Service)
2. **Rooms API**
   1. POST <platform\_name>/ hotels/hotel\_id/rooms ---creation of rooms (Admins Service)
   2. GET <platform\_name>/hotels/hotel\_id/rooms/room\_type\_id ---searching of the hotel room type (Search Service)
   3. GET <platform\_name>/hotels/hotel\_id/rooms ---searching the types of hotels and room (Search service)
   4. PUT <platform\_name>/hotels/hotel\_id/rooms/room\_type\_id ---updating of rooms (Admins Service)
   5. DELETE <platform\_name>/hotels/hotel\_id/rooms/room\_type\_id ----deletion of rooms (Admins Service)
3. **Booking APIs**
   1. POST <platform\_name>/bookings ---for booking the room type in the hotel (Booking Service)
   2. GET <platform\_name>/bookings/<booking\_id/user\_id/hotel\_id> ---for getting the booking details (Booking Management Service)
4. **Search APIs**
   1. GET <platform\_name>/search/query --- to cater all the search query

## Choosing the right Database

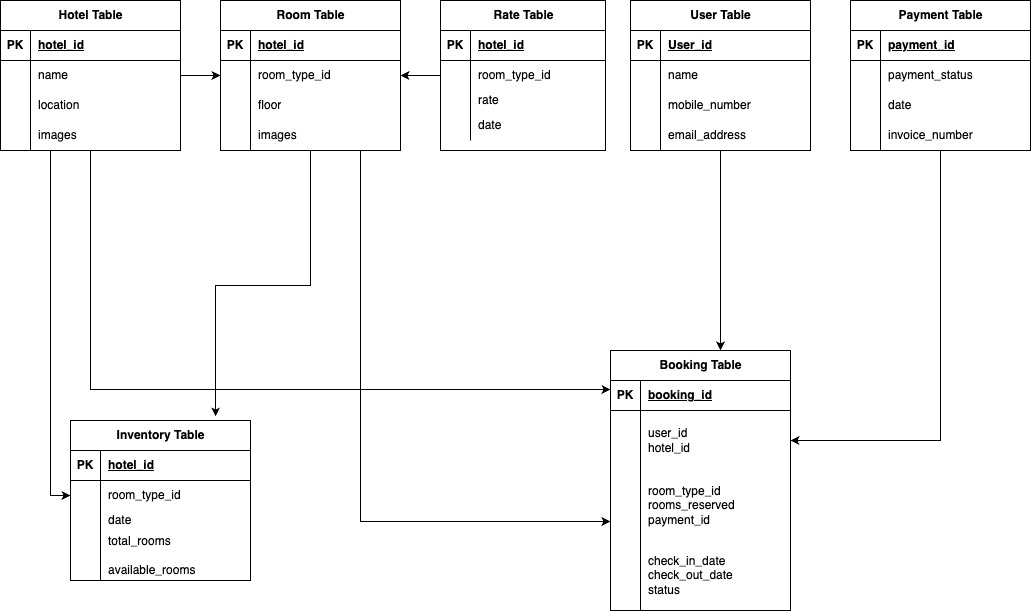
1. Write – 2 TPS (NO SQL is not necessary).
2. Entities can be modelled and correlated in SQL in better way.
3. ACID properties to be used.

Any SQL database we can choose (e.g. MySQL, AWS Aurora, Oracle etc.)

## Architecture Diagram



## ER diagram



## Different Services and their goals

* + - 1. Show hotel and room details to customers

1. **Service Services:** It helps to search hotels and room with the help of Elastic Search cluster (for addressing the typo and querying with different tag like swimming pool etc. to address fuzzy and wild card search)

Elastic search cluster get the input from Search Consumer of Kafka messaging tool for all the modification of hotel admin service.

For e.g. If any change in the rooms, the same got percolated to the Elastic Search cluster and same has been shared with CDN for low latency.

* + - 1. Reserve a hotel room for a specific date range

1. **Booking Services:** It helps the user to book/reserve the rooms for the provided check in and checkout date with a look up with Inventory table for rooms are available and booking table for reserving the room type.
2. **Booking Managements Service**: When Customer or admin want to fetch the bookings (for a particular day or user or hotel), it will be from Redis cluster which get the data from the SQL cluster Booking table)

Whenever the booking is completed (means the customer has utilized the hotels service) then there will be a shift in the status of the Booking table from booked to completed.

After the lifespan of data is completed, the data is moved to Hadoop cluster for EDW by using Archival Service (For e.g. Sqoop)

* + - 1. Admin functionality to add, remove, or update hotel and room information

1. **Hotel Services:** Admins have access to create/update/delete for hotels and rooms
   * + 1. Support dynamic room pricing based on expected occupancy rates
2. **Rate Service**: All the data (like hotels, rooms, traffic, holiday, demand etc.) has been pushed from different services to Kafka and then to Hadoop cluster for consuming the streaming data to predict/generate price for the specific room type of the hotels for a specific date.
   * + 1. Allow up to 10% overbooking to account for possible cancellations
3. **Booking services**: As the booking of the room is depended on the Inventory table (Column: totalcount), while reserving any room type we can make a functional logic to have a threshold of +10% of the total count of the specific room type .If no booking got cancelled within a specific deadline date , then the Hotel management have to allocate different room to the customer or have to refund the same with some freebies for the apologies.

## Booking Flow

1. Customer is trying to book a room in hotel.
2. Check the inventory table with specific date (check in and check out) for specific room type and fetch the available count should not be less than threshold value( for e.g.: if total count of rooms are 100 then the available count should be greater than -10 to address overbooking) , if the response is true move to next step or provide user with different room type of hotel.
3. Check inventory and reduce the inventory (available\_count) for the respective room type.
4. Make an entry into the Booking table with all the entries with status as reserved. (Allowed entries reserved, booked, cancelled and completed)
5. Make payment and marked the booking entry to booked (having Redis cluster for Time to live)
6. If booking is cancelled for any scenarios, then marked the booking as failed and increase the available\_count entry in the inventory table.

## Scenarios to address

**User Story 1: Room reservation with Dynamic pricing**

**Solution**: All the data (like hotels, rooms, traffic, holiday, demand etc.) has been pushed from different services to Kafka and then to Hadoop cluster for consuming the streaming data to predict/generate price for the specific room type of the hotels for a specific date. Tools required are Hadoop cluster, Spark streaming and Machine learning algorithm for analyzing real time data for the price of a room type.

**User Story 2: Concurrency Control**

**Solutions:** Optimistic locking, for each fresh row inserted in the inventory table the records will be marked as v1 and when multiple users try to reserve the same room type then same version v1 will be read and who every commit first will increment the version v1 to v2 and when the user2 tries to book the same room type then the service will be in such way that if there is a version mis matching with the version of the inventory table then data will be pulled again from the inventory table and new booking will be allowed if the room available value is not less than threshold value(as mentioned in the above) else failure message will be pop up to the user with alternative room type in same hotel.

**User Story 3: Overbooking**

**Solution:** As the booking of the room is depended on the Inventory table(Column: totalcount), while reserving any room type we can make a functional logic to have a threshold of +10% of the total count of the specific room type .If no booking got cancelled within a specific deadline date , then the Hotel management have to allocate different room to the customer or have to refund the same with some freebies f or the apologies.

**User Story 4: Idempotency of reservation**

**Solution**: We will have Idempotent service Api configure in Booking service. It will flag out the double booking from server for same input parameters and disabling the Reserve button in front end to the customer.

Input parameters (user\_id, hotel\_id, room\_type\_id, num\_room, checkin\_date, checkout\_date)

The above input parameters hashed value will be used for generating booking\_id for booking table, it will be unique for each booking and the booking\_id will be primary key in booking table as a result the idempotent scenarios will be stopped.

**User Story 5: Handling Cancelations**

**Solutions:** Whenever a cancellation event occurred, the customer will be refunded with the booking amount and the inventory for the specific room type will be rollback with the increment value in the available unit.

**User Story 6: Database Scalability**

**Solutions:** As the database has been normalized and modelled (as in ER diagram), it can handle huge load. To scale the database, we are using SQL cluster is in Master slave architecture which help us for high availability and to tackle high read and write operation. Further to tackle the resource usage in databases we have divided the SQL tables to two group and the same has been published in the ER diagram

To further improve the performance, we can move the old data to Hadoop cluster for archival process and EDW process.

## Additional Design Consideration

1)**To sync the data is multiple databases**: We will be using SSIS tools (SQL Server Integration Services) or ETL tools for syncing of the data in the different cluster. This tool helps us to sync the data from one system to another within few milliseconds and the strategy will be incremental synchronization (whenever an entry is made in the databases the same will be copied to the other)

**2)Approach to manage peak in user traffic:** The whole ecosystem is built by software/technology which are horizontal scalable, and all services are microservices. We can scale any application to any limit and load balancers are added to address huge traffic.