Design Airbnb,
OYO,
Booking.com

(System Design)

(1) Functional Requirements:

Hotel:

(1) Add Hotel (on-boarding)
(2) Add Room
(3) Update Hotel Details
(4) Update Room Details
(5) Get room details
(6) Get Booking details
etc...

USER:

(1) Search for a hotel
(2) Booking
(3) Get Booking status.

(2) Non-Functional Requirements:

(1) Low Latency (User should see available hotels as fast as possible)

(2) High Availability (System should be highly available to ensure trust)

(3) High consistency (Ensure Trust)

(3) Capacity Estimation:

No. of Hotels >= 500 M (all around the globe)

No. of rooms >= 500 X 1000 rooms per hotel

Daily Active User ≅ 1000 M or 1 B

List of Services:

(1) Hotel service: To assist Hotels
(2) Search service: To retrive hotels
(3) Booking service: To Book a room
(4) BookingStatus Service: To get booking status.
(5) Notification service: To notify Hotel and user

List of Databases:

(1) MySQL Cluster: To maintain details pertaining to the Hotels.

(2) MySQL Cluster: To Keep track of booking details of the hotel or room by user ✓

Both are mysql clusters because the data is structured & the need of ACID made me choose mysql cluster.

Mysql cluster contains a master and a set of slave DBs

(3) Elastic search cluster: To maintain details for Search.
Why ES cluster ?? To support fuzzy search

(4) <u>cassandra cluster:</u>

TO maintain track of all transactions that happened on hotels.

<u>List of cache:</u>

(1) Redis or. memcached :

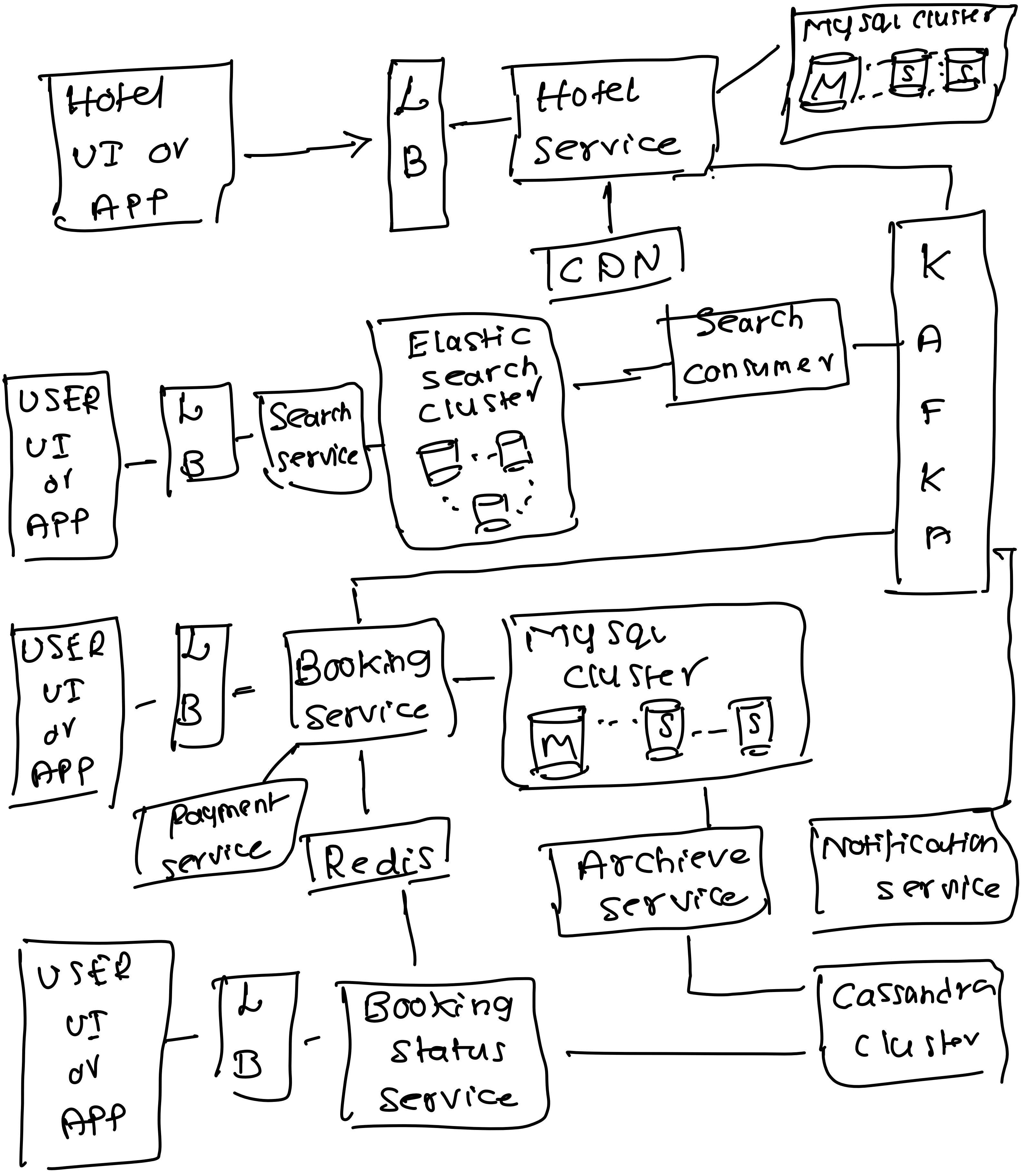
While Booking a room, we need to have a session timer (TTL) for payment At that time (Booking_id, TTL) will be in the cache

<u>List of communicators:</u>

kafka cluster : TO support distributed features and multiple consumers for multiple producers. so kafka topic can help me here

<u>Extra DB:</u>

we can maintain all details of Booking in Hadoop DB or Bigdata Storage. upon performing data Analytics we can come up with pricings for rooms may be based on demand & supply



Load Balancer follows consistent Hashing (IP Hash)
Every service is distributed & scalable.
When demand increases, No. of servers will be added

APIS & DB Deep Dive:

Hotel:

(1) POST /hotel (Add Hotel)
(2) POST /hotel/room/id (Add room)
(3) PUT /hotel (Update Hotel)
(4) PUT /hotel/room/id (update or add) etc...

DB:

(1) Hotel Table:
[hotel-id, Locality-id, no.of rooms, Original images, display.images, available rooms]
≥ 0

(2) Room Table:
[hotel id, room id, original image, display, room-facilities)

(3) room facilities:
[room-id, water, AC, Porn]

(4) Hotel Facilities:

(5) Locality
[locality-id, zipcode, state, country)

USER: (APIs)

While booking

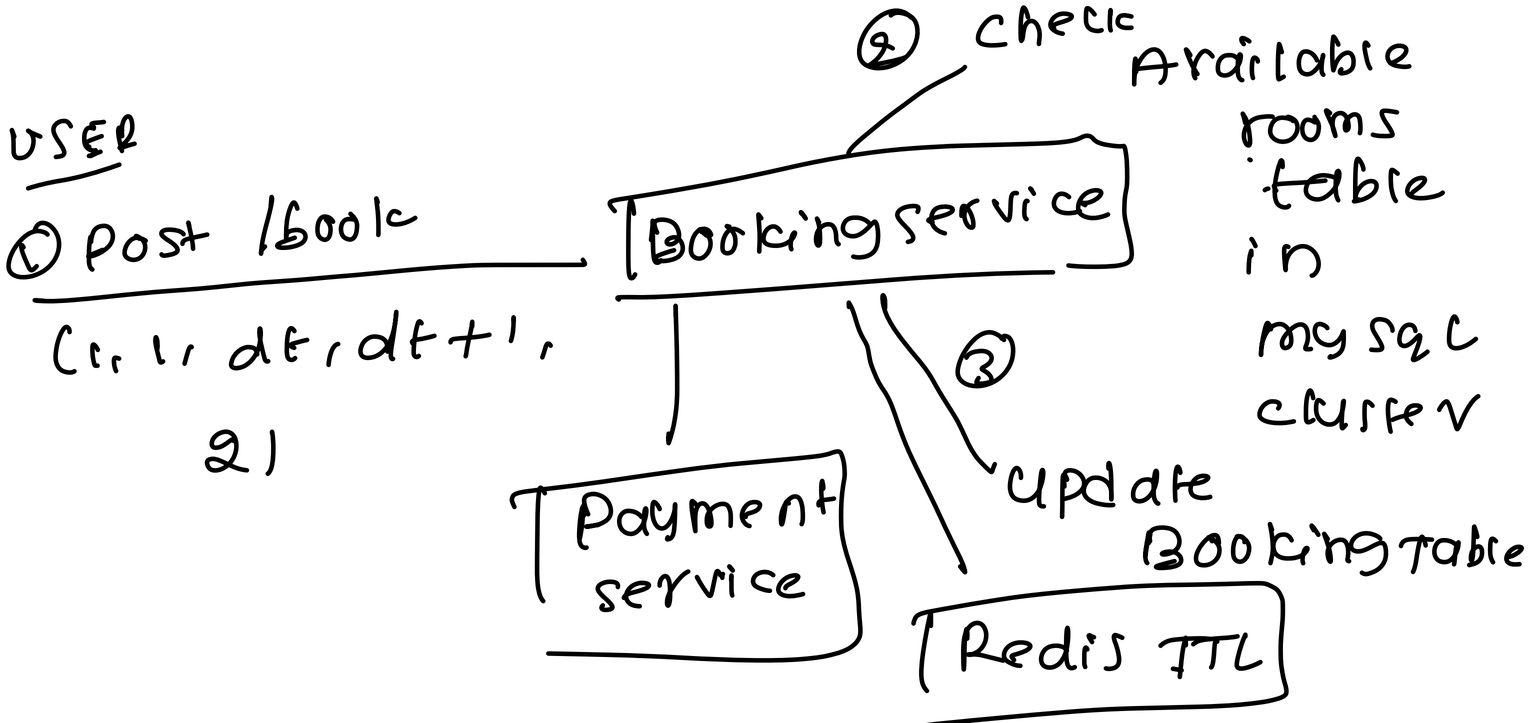
(1) Post /book
(user-id, room-id,
start date, end date,
quantity)

DB:

(1) Available-Rooms:
[room-id, date, available quantity]

(2) Booking-Table:
[booking-id, room-id, start date, end date,
quantity, status]

(3) Status: [Reserved, booked, cancelled,
completed]



Dry Run:

(1) USER POST to book a room

(2) Booking service will look in to the Available rooms table.

if rooms > requested quantity
- ↳ proceed to payment
- ↳ update Redis
- ↳ update Booking Table

else, Notify that rooms count is low

Available Rooms:

① [r, dt, 7]   7 > 2

available quantity > requested

② update Booking Table

[r, l, dt, dt+1, 2, Reserved]

③ create a session (TTL) and wait for payment

Booking Table
Status
column

Possibilities:

① payment success → booked

② Payment failure → cancelled

③ Time expired in Redis → cancelled

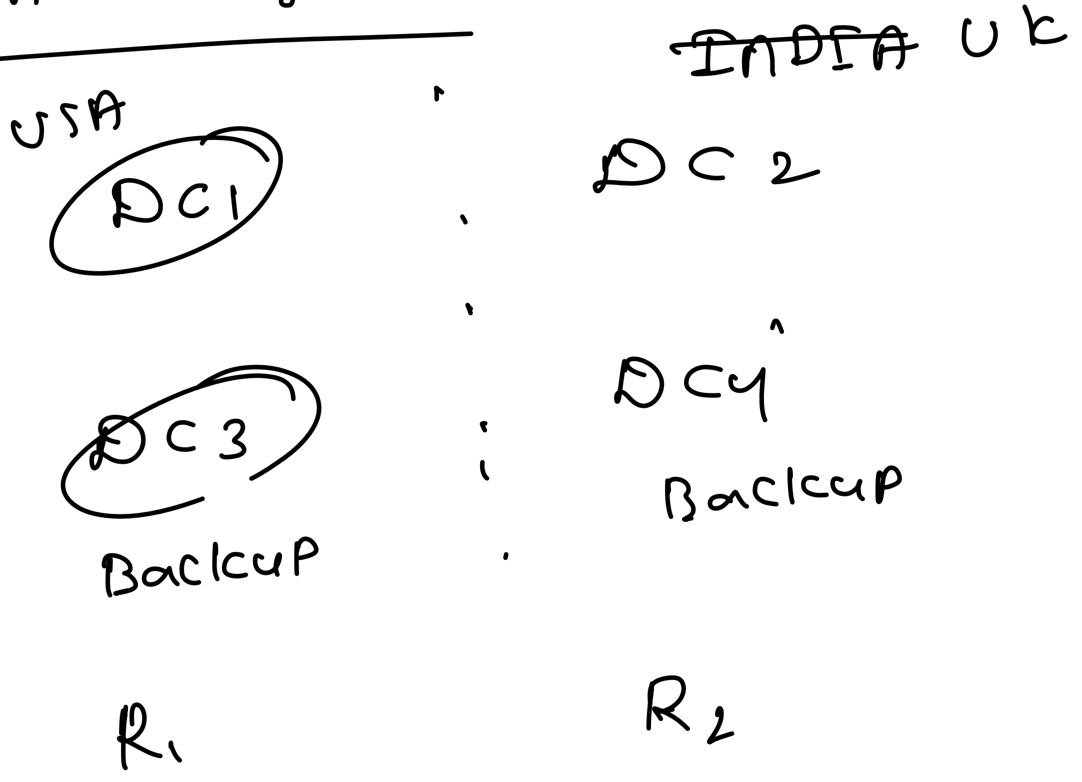
④ ③ & ① } Time expired & payment successful

ⓐ revert payment

ⓑ book another room

⑤ ① & ③ → booked.

Data center Regioning:



Keeping all data in one data center is not optimal. only 25% is operating remaining 75% is backup.

so, Divide globe into two regions.

Country belongs to R1

↳ keep the details of hotels in $\boxed{\text{DC1}}$

with DC3 as backup

and viceversa.