Real-time Gaming Leaderboard

EDS: +1 point if win the Match
each month kick off → New leaderboard
10 users display
5 Mill DAU, 25 Mill Monthly users
10 Matches per day
display real-time results

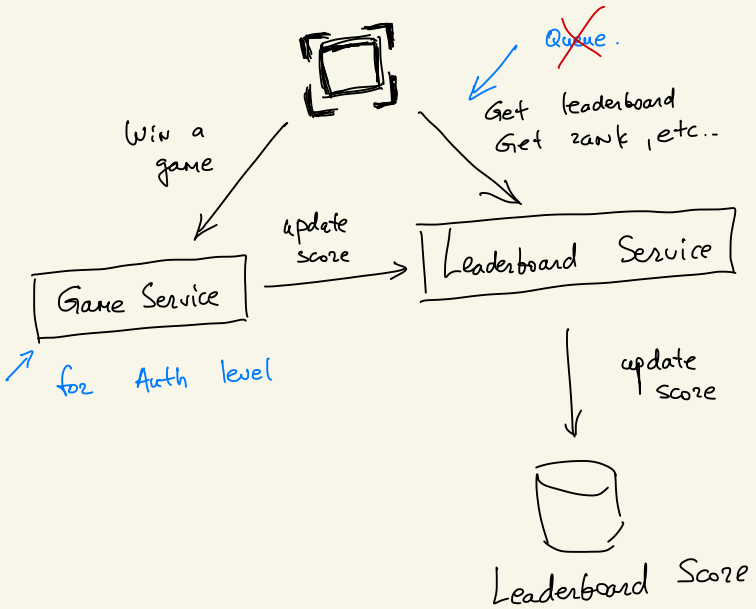
NFR: Real-time update / real-time reflected on the leaderboard
General scalability, availability, reliability

BEE: $\frac{5.000.000 \text{ DAU}}{10^5 \text{ sec}} \approx 50$ QPS; Peak time: QPS × 5 = 250 QPS
× = 500 QPS
× = 2500 QPS

HLD: API design:

POST	/v1/scores	update user position on the leaderboard
GET	/v1/scores	fetch top 10 players
GET	/v1/scores/{:userId}	fetch rank of specific user

HLA:



Database:

RDS — heavy changing data, need to sort over millions of rows — speed 10s.
No caching for constantly changing data.
Index + Do not performant to scale.
Requires the table scan to determine the rank.

Redis : in-memory . Sorted Sets data type . O(N) for insertion, removal, search.

to improve performance use Binary Search. O(log N)

Implementation using Redis SortedSet :

ZADD - insert user
ZINCBY - increment the score
ZRANGE / ZREVRANGE - fetch range of users
ZRANK / ZREVRANK - fetch user position

ZINCBY <key> <increment> <user>

Storage requirement:

name_id 24 char string } 26 bytes
score 16 bit int (2 bytes) }

25 mill * 26 bytes = 650 MB

CPU, I/O usage:

peak: 250 QPS . This is well within the performance envelope of a single Redis server.

Redis do not support persistence.
Use Redis read replica in configuration.

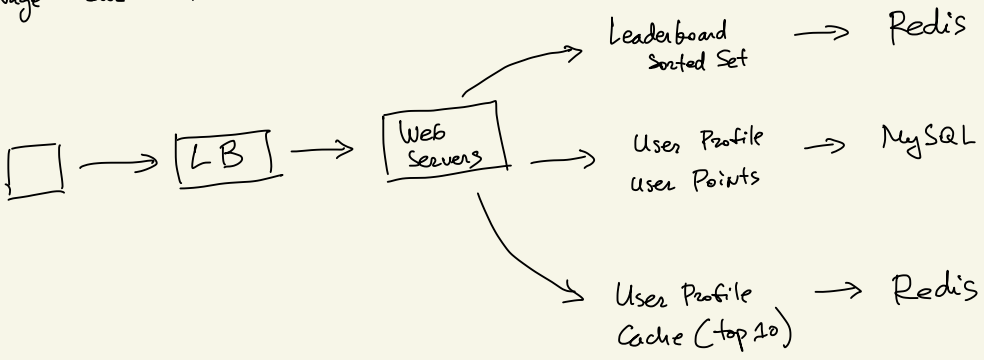
MySQL : user_id
score
timestamp
← data for play history, etc.. services

Additional cache for top 10 players.

DDD :

1. Cloud ? AWS ?
2. Scaling Redis.
3. No SQL ?

Manage our own services :



Using AWS: API gateway → REST endpoints
Lambda Function

Auto Scaling +

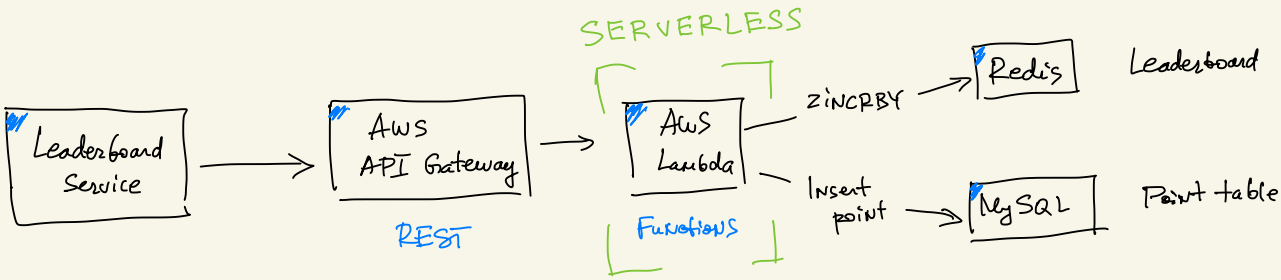
API

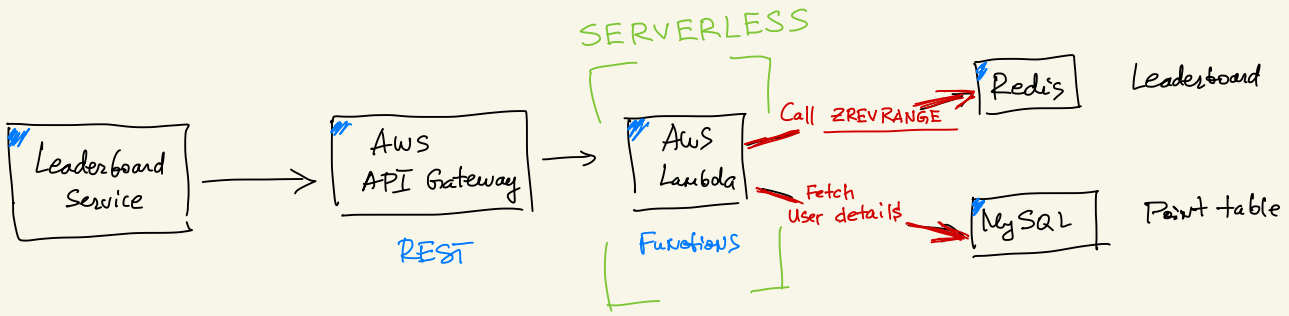
GET /v1/scores
GET /v1/scores/{:userId}
POST /v1/scores

Lambda Function

Leaderboard Fetch Top 10
Leaderboard Fetch Player Rank
Leaderboard Update Score

} invoke Redis, MySQL commands





Scaling: 5 mill DAU no need to scale.
↗ x 100 = 500 mill DAU
↗ 65GB x 100 ; ↗ if QPS 250.000 !

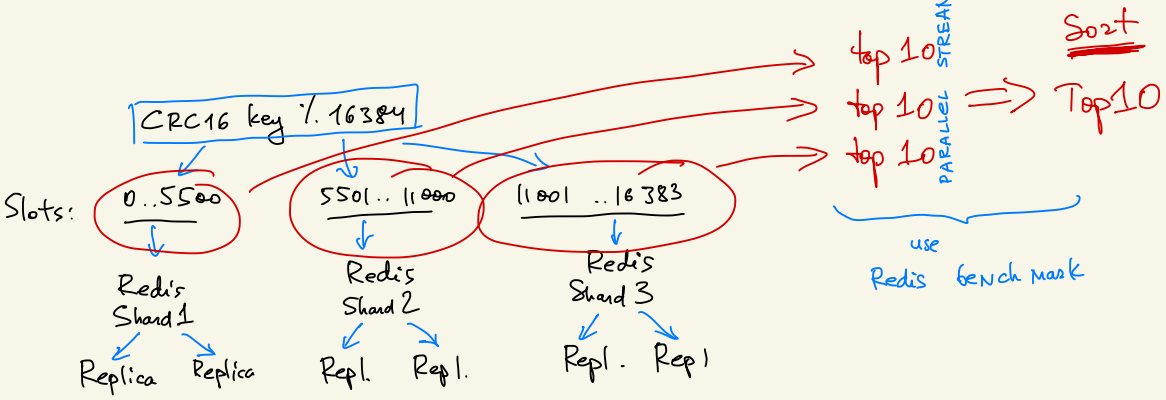
Sharding: 1 ... 1000 (points)

Fixed partition:

1..100 , 100..200 , ... , 900 – 1000 .
Sorted Set Sorted Set Sorted Set

Will need to fetch top 10 from the last shard
O(1)

Hash partition:
Redis cluster, hash slots 16384.



Using NoSQL:

1. Optimized for writes
2. Efficiently sort

Amazon DynamoDB, Cassandra, MongoDB