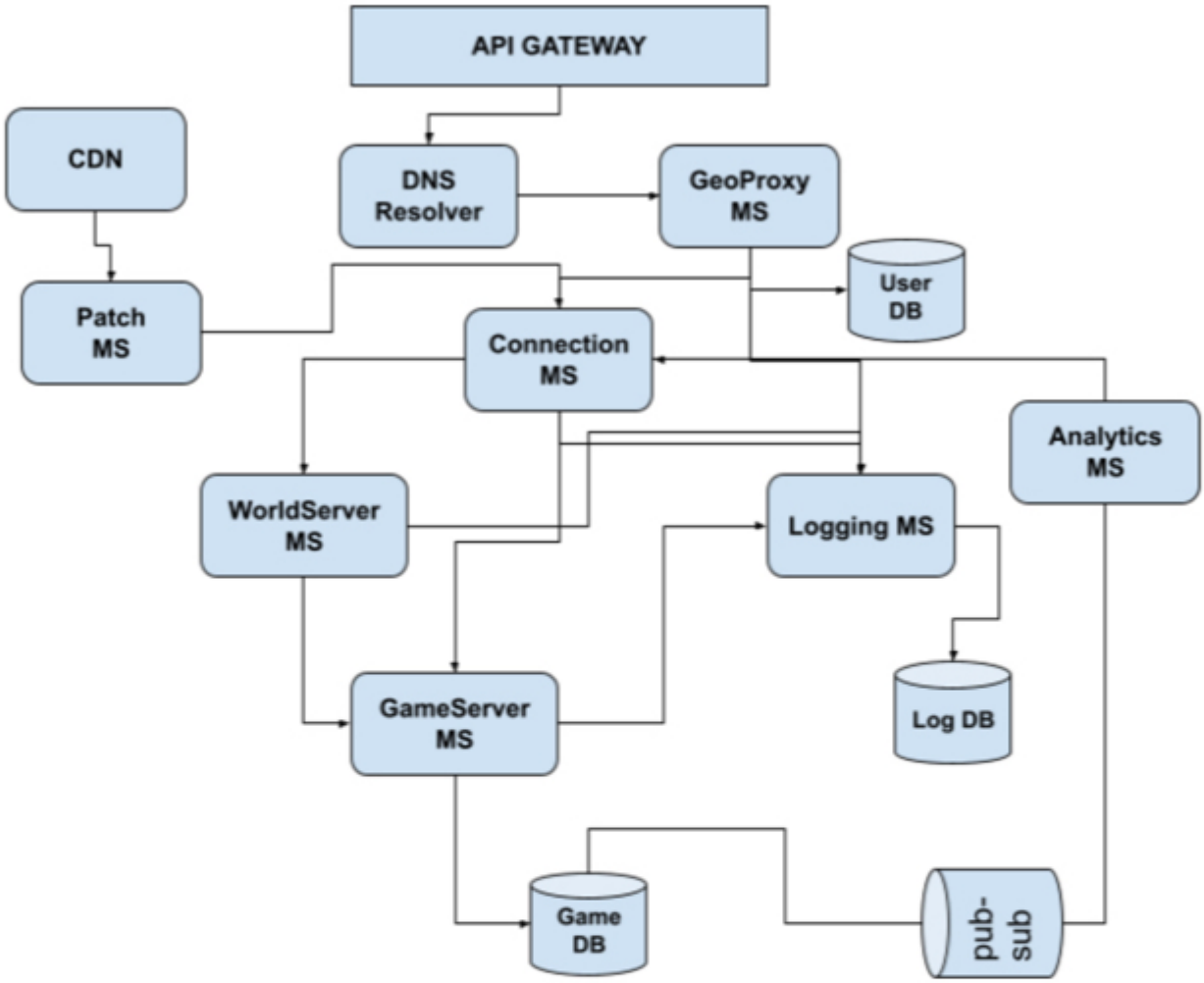
Functional Requirements

a. Functional
    i. Play Game
    ii. Add to existing Game
    iii. Publish Scores / Leaderboard

b. Non-Functional
    i. Latency has to be in acceptable levels (200ms)
    ii. Minimize bandwidth consumption for synchronizing the state of the game
    iii. Minimal jitter when updating the state.

Logical Architecture Diagram

When a user connects to the online game using a web-browser, a REST API request is sent to the **API Gateway** Interface. Based on the geography the user is connecting from, the respective **Geo-proxy microservice** comes into action. This service forwards the request to the **Connection microservice**. The Connection Microservice validates the user as per the credentials stored in the **User DB.** The **Patch Microservice** checks to see if client software is up to date, else a newer version of the software is pushed. This software is stored in **CDN** closer to the user for quickest response. The CDN also stores static images, video, music that are part of the game. All these steps help to alleviate the latency experienced by the user. Also this Patch microservice can be used to push if any emergency fixes while the game is in progress too.Based upon the successful validation, a session is created in the **World Server Microservice**. Based upon the game requested, the World microservice returns the **Game Server** instance along with other users who are part of this game. The updates as part of the game are handled in the **Game Server Microservice**. The World Server Microservice is the source of truth for the game instance that is currently running. The current state of the game is relayed to all the players retrieved from the **Game DB**.

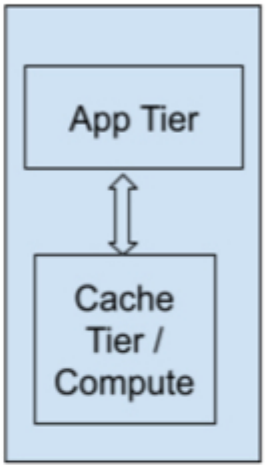


Detailed Design of Microservices

Microservice	Type of Technology
Connection Microservice	K-V Workloads, Simple Data Storage on Disk and Hash Table K-V in memory
Patch Microservice	K-V Workloads, Simple Data Storage on Disk and Hash Table K-V in memory
Geo-Proxy Microservice	K-V Workloads, Simple Data Storage on Disk and Hash Table K-V in memory
Logging Microservice	Offline Compute Intensive, Online: Streaming Analytics

Analytics Microservice	Offline Compute Intensive, Online: Streaming Analytics Workload
Game Server Microservice	Compute Intensive Workload, In-Memory Processing
World Server Microservice	Compute Intensive Workload, In-Memory Processing

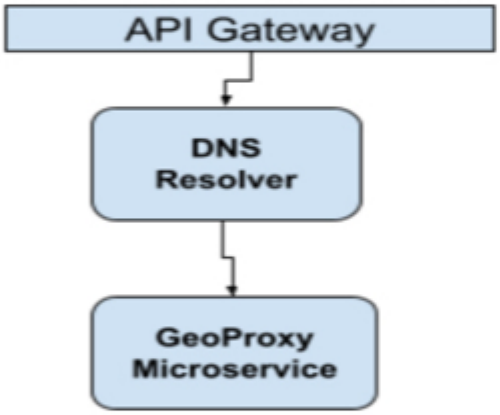
Geo-Proxy Microservice



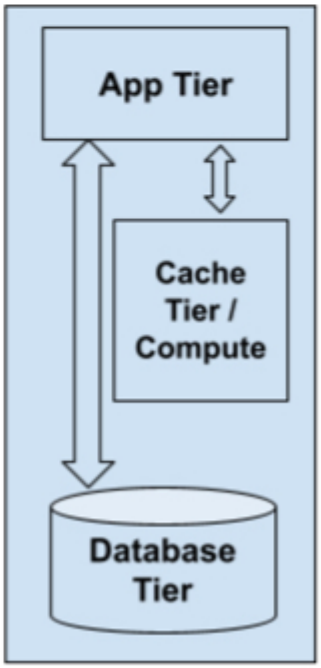
When a user connects to the online gaming application, the API Gateway forwards the request based upon the Geo presence of the gamer. The DNS resolver present will appropriately redirect to the correct Geo-proxy server location.

Data Design

a. requestConnectionToGameServer(userId, gameTitle)



Connection Microservice



The Connection Microservice is one of the critical services that Social Gaming needs to support.

Data Design

a. createUserAccount(userName, email, password, phoneNumber, address, country)

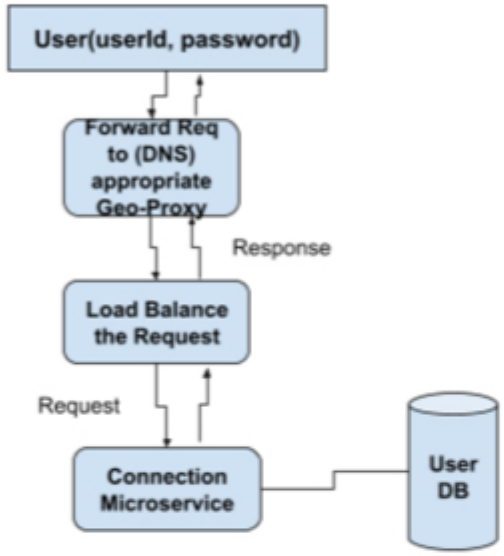
This method creates the user account for the first time. It stores the information in a simple K-V workload CRUD based database. On successful creation of the account, a unique userId is generated and stored against the created user account. The database could store the uniquely generated userId to index into the database for faster lookup

and information retrieval. Based upon the location of the user, the request is routed through the appropriate Geo-Proxy service.

b. updateUserAccount(userName)

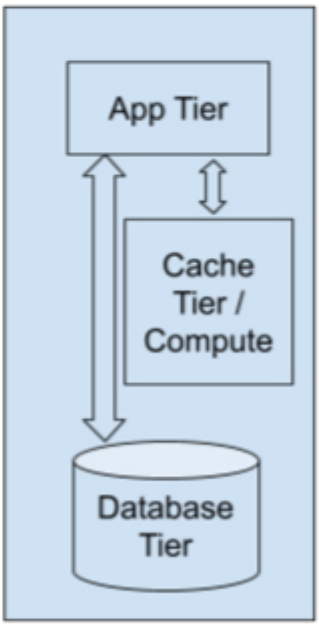
This method is used to modify any of the fields except the userName and email address combination to avoid adding duplicate records in the database.

c. loginToGamingServer(userId, password)



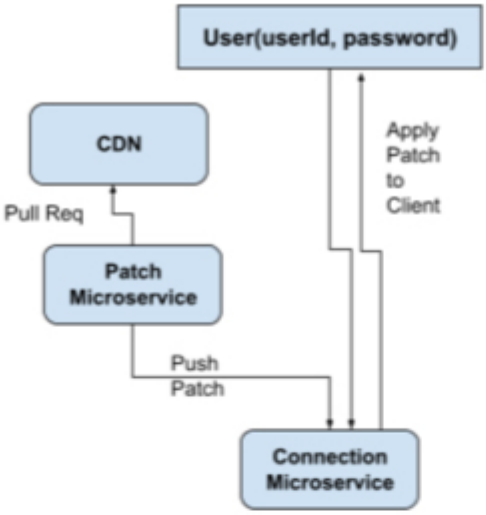
Patch Microservice

One of the important services part of the Social Gaming platform is the Patch Microservice. The role of this service is to deliver any software patches that the Gaming client may require after successfully connecting to the Gaming service.



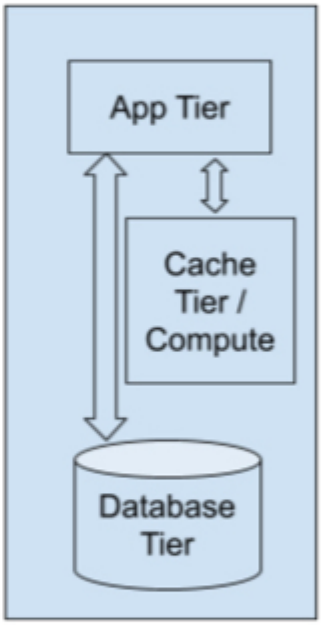
Data Design

a. pushPatchToClient(userId, patchPath, status)
When a connection to the Game Server is successful, the Patch microservice determines if the client software requires any updates or fixes. If it determines that the version that the client is running needs updates or fixes, it downloads them from the CDN and pushes it to the client.



Logging Microservice

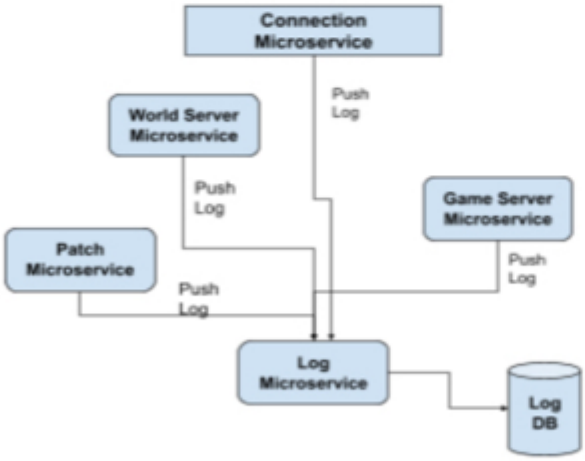
The logging microservice is a Stream Based Workload for log analytics and Compute Intensive workload for offline log processing. This is a write-heavy service.



online

Data Design

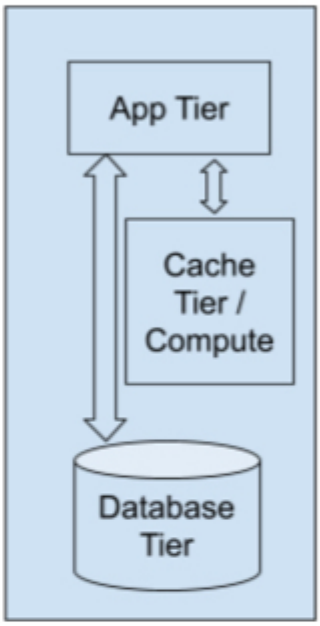
a. writeLogToBuffer(serviceId, userId, logLevel, log, timeStamp)



LogId (Key)	ServiceId	Log Message	Severity

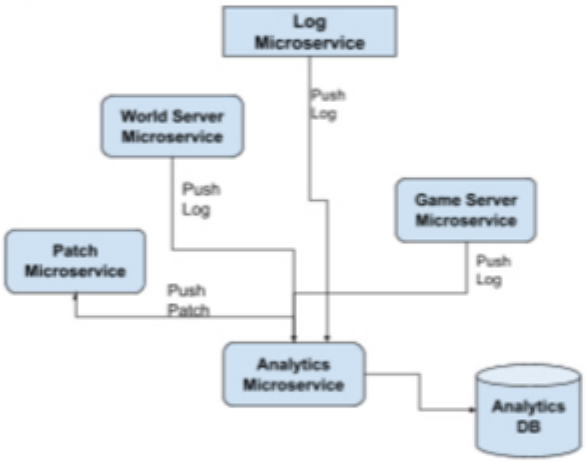
Analytics Microservice

This is one of the services that does not directly interface with the User actions. It proactively analyzes the games that users play, the moves that they are interested in, and accordingly create new features of the game or enhance the user experience. This service also helps to identify any anomalies that were found as part of the game and can push patch software to the clients while the game is being played. This service is Compute Intensive for offline streaming processing workload.



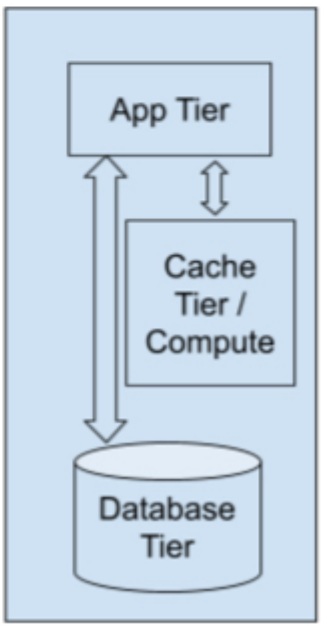
Data Design

a. fetchLogForAnalytics(logId, logMessage, logSource, analyticResutls)



World Server Microservice

The WorldServer is a Compute Intensive workload microservice. When the user has been authenticated and all the patches have been successfully applied, if applicable, the WorldServer based upon the geo-location of the user, creates a game instance on the Game Server.



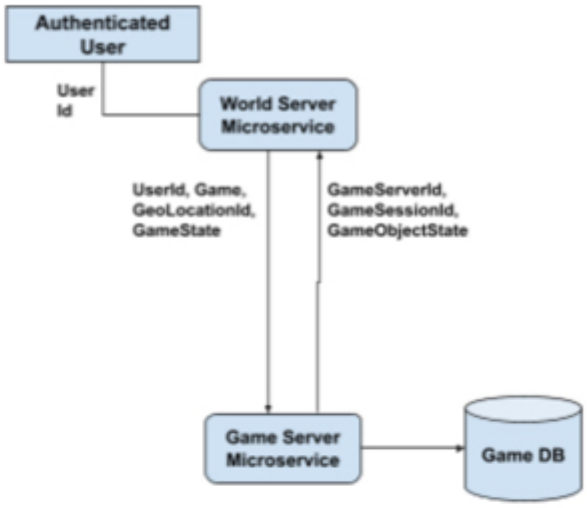
Data Design

a. createGameSessionId(userId, game, userGeoLocation, gameState)

This API associates the specific user to the Game that has been requested, with a corresponding GameState.
Possible GameState includes:

a. NewGame
b. ResumeGame
c. CancelGame

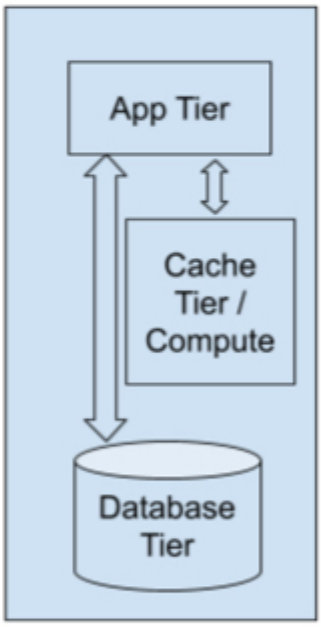
The GameServer returns the gameServerId, gameSessionId, and the gameObjectState. The gameObjectState. The gameObjectState represents the current state of gameScreen with respect to all the players part of this game.



b. endGameSessionId(userId, gameSessionId, gameServerId, ttl)

Game Server Microservice

The GameServer Microservice is the heart and mind of any Social Gaming infrastructure. The aforementioned microservices support this microservice for realizing a successful gaming experience. The GameServer is also called the Authoritative Server (AS) which validates the game in case a user hacks the system. Without this role of the AS, the game may not be played according to the stated rules of the game.



Data Design

a. MatchId createMatchId(userId, game, <List> players, gameState)
b. calculateTickRate(userId, game, gameState, <List> players)
c. determineMatchState(userId, matchState, <List> players)
d. matchPlayersToGame(<List> players, <List> userId, gameId, matchId)

When the WorldServer requests for adding a user to an existing game or to create a new game, the Game Server returns a matchId as handler. MatchId is nothing but an instance of GameId. In the Gameserver microserver, a tight loop runs, which periodically wakes on a tick(order of few milliseconds) to process various events that are received in the current match being played. Typically, for a game the tickRate would be around 30 ticks per second. In the woken up state, the GameServer thread determines the state of the match and updates the match state to each player's part of the game.

Need to Scale

The GameServer Microservice is a Compute Intensive workload, as such needs to be scaled for:

Throughput

The throughput in the context of Game Server can be thought of as reduced bandwidth consumption when propagating the game state to all players in the match. This can be achieved by having a 64-bit vector for possible actions. This can be applied to all the players of the match with (vector_info, timestamp, <List> players)

Also another option is to use UDP as connection protocol, Websockets at the GameServer end to keep the connections open towards the clients.

Single Point of Failure

To reduce the cases where Single Point of Failure of the Game Server can happen, a typical 3 Game Server redundancy can be supported for each geographically separated social gaming sites.

Latency Reduction, as outlined in the NFR

To address the Latency to be within acceptable limits, we need to replicate the GameServers in a geographically distributed fashion. With the help of DNS resolver service, the appropriate GameServer can be mapped to the user.

Future Enhancements

a. LeaderBoards
b. Game Tournaments
c. Friends Graph