Skill based matchmaking service

Members

Name: Hemanth Chenna **Email id:** hech9374@colorado.edu

Project overview and goals

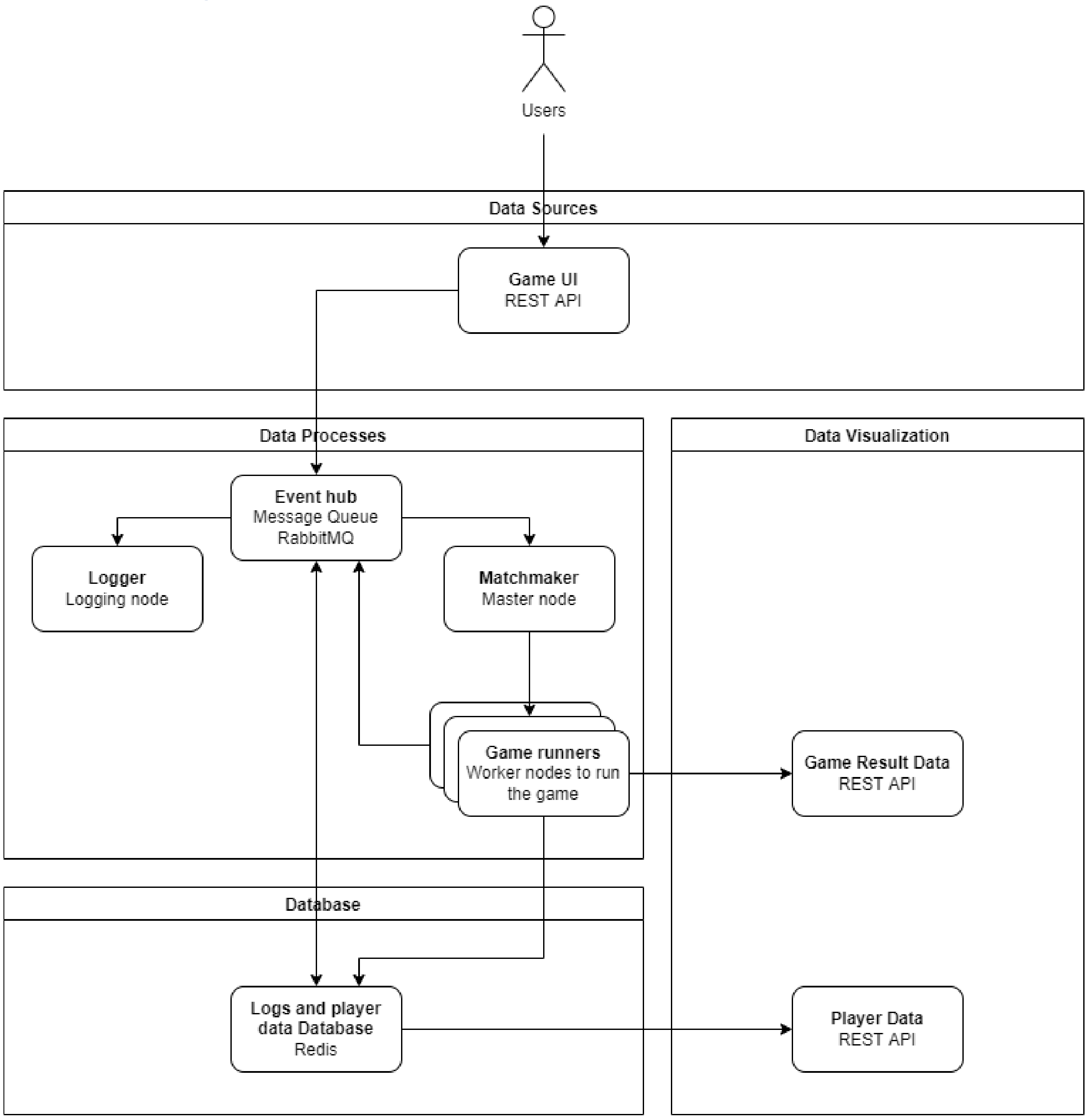
The goal of my project is to create a service for online games to aggregate and group similarly skilled players into the same lobbies and run servers for the created lobbies. Subsequently, the results of these games should be stored back into the database when the game completes. The game itself is a placeholder for any massively multiplayer game over the internet where skill based matchmaking is required, which is becoming an increasingly common occurrence in today's internet driven world.

Components

- Kubernetes and Docker
    - **Purpose –** This is used for scalability and being able to deploy all the other components to a cloud infrastructure like Google Cloud Platform
    - **Advantages –** Kubernetes is the most widely used deployment tool, is open-source and reliably provides the features that are most useful to scale my project
    - **Interactions –** Kubernetes uses Docker containers to deploy the Redis DB with its dbnode, RabbitMQ message queues, REST server and ingress, and the matchmaker with the worker nodes
- Redis DB
    - **Purpose –** This is used to store the player data and game results after the games have been completed. A dbnode is present to handle the incoming RabbitMQ messages and serialize them so the DB does not overload or have any contentions
    - **Advantages –** Redis is faster than other DBs and provides the option to store to persistent volumes in snapshots
    - **Disadvantages –** Since this is only persisted in memory until a snapshot is stored, any data between snapshots may be lost in the event of failure
    - **Interactions –** There are 2 DBs, the first for storing player data which is updated whenever the matchmaker signals a new player or when a game finishes and scores need to be updated; the second to store a history of games played to be retrieved on demand by the REST API calls
- RabbitMQ
    - **Purpose –** This is used to exchange messages between the different components that have been deployed
    - **Advantages –** RabbitMQ allowed for more flexible routing of JSON messages, and the transfer of these messages was also more reliable
    - **Disadvantages –** Something like Apache Kafka may have been more efficient and allow for more distributed operation

    - **Interactions –** There are 3 main message queues: toMatchmaker, toPlayerDb, toWorker and as the names state, they queue messages to the matchmaker, the dbnode of the redis DB and worker nodes respectively.
- REST API and ingress
    - **Purpose –** Serves as an interface with the end user, either through a python file or a web interface
    - **Advantages –** Simple and easy to implement. Can also be combined with a load balancer to improve performance
    - **Disadvantages –** Faster options are available with the possibility of developing a GUI for the endpoint
    - **Interactions –** Sends add player requests to the matchmaker through RabbitMQ and performs data retrieval requests to the Redis DB to display results to the user
- Matchmaker and worker nodes
    - **Purpose –** Performs the task of skill based grouping and running games to get results.
    - **Interactions –** The matchmaker interacts with the REST server to get the add player requests, with the DB to get player history, and to the worker nodes to run the game. The worker nodes get the game ID and player list from the matchmaker, run the game and send the result to the DB

Architecture Diagram



Capabilities and Limitations

As per my testing, the code can handle 1000 players in the queue in a feasible amount of time while choosing 60 players per game (tested by loading the DB and queue through a .csv file)

The system has the following limitations/future enhancements:

- The system is limited in time by python's sorting algorithm. Can improve this by using faster sorting algorithms/libraries and inserting into sorted lists.

- Queue wait times can be reduced by providing some leeway in the acceptance threshold based on how much time has elapsed in the queue (Increasing the threshold for players who have been waiting a very long time)
- Integrate a game GUI and game animations into the game runner to display scores as they are received
- Run the game runners through on-demand jobs to be able to spawn as many servers as necessary
- Run a CronJob to periodically store the DB values into a persistent volume