Implementation Approach

Database : Postgresql ; Server : Uvicon + FastAPI (for endpoints)

We put exception handling in place and handled any issues with the database updates by rolling back the updates so that the database is always in a consistent state. We made sure a producer cannot access a topic they are not registered for.

Tradeoffs

We used 4 different tables in the database schema :
Topic : name, timestamps corresponding to the create and update
Producer_Topic : producer_id, name, timestamps corresponding to the create and update
Consumer_Topic : consumer_id, name, pos(signifying the next log to be consumed), timestamps corresponding to the create and update
Queue : index (sorted according to this to maintain the FIFO order), message (log texts), topic name and the timestamps

We do not need a Topic table, as we could get all the listed topics by querying on the Producer_Topic but we used extra tables to make sure the queries are processed faster as the Producer_Topic table would typically be much larger than the Topic file.

Testing

Client library

Built the client library unit test suite using unittest mocking the responses using patch

End to End

Multiple producer threads are made to log several messages from corresponding log files simultaneously to the broker and at the same time multiple consumers are made to poll for various topics consuming them whenever the log messages are available

Challenges

Faced some issues while trying to make some of the database transactions atomic (for persistence), had to use db.rollback() in case of any issues with the update queries. Faced some issues with dockerization.