Database Choice for Microservices Architecture

Contents

Introduction

In the design and implementation of an enterprise level software, the choice of database system is very important, particularly when using microservices. This document will show a detailed rationale for selecting Cassandra and PostgreSQL as the database systems for the project Movimingle. These choices are aligned with Learning outcome 7 – Distributed Data, by focusing on best practices for handling and storing large amounts of various data types, while also sticking to legal and ethical considerations.

Isolation

To improve the system's speed and to reduce possible failures, I have implemented isolation across different microservices. This means that each service will manage its data independently and operate autonomously. By doing this, I ensure that each service is insulated by disruptions from the others. for example, if the activity in one service is very high, the rest of the services' performance or availability will not be affected by that.

Initial Consideration of Two Databases

Initially the plan was to use Cassandra for the Voting and Movie Selection service and PostgreSQL for the User Management and Favorites Management service. I had decided to go with this approach because of Cassandra's high throughput and PostgreSQL's strong transactional integrity for structured data.

Reevaluating the Need for Two Databases

Upon deeper analysis of Movimingle's requirements and use cases, it became evident that the complexity introduced by using two different databases is not justified. The main reasons are:

1. Lack of need for high transactional integrity:
    - Movimingle does not handle sensitive financial or medical data where transactional integrity is crucial.
    - The data involved, such as user profiles and favorites lists, do not require the strict consistency and transactional guarantees provided by PostgreSQL.
2. Unnecessary complexity:
    - Managing and maintaining two different database systems increases the complexity of the system architecture.
    - It requires additional configuration, monitoring, and backup processes, which can complicate development and operational efforts.
3. Sufficient performance with a single database
    - A single NoSQL database like Cassandra can handle the read and write operations for both voting activities and movie data management efficiently.
    - Theoretically it provides the necessary scalability and performance to support a large user base. The performance difference between using single database and the current setup will have to be tested to determine which is best in this case.

Simplifying with a single database

In order to simplify the architecture and overall system, the option of using only Cassandra for all database needs that the software demands is sufficient. Below is some of the reasoning for the change to take place.

Cassandra for All Services:

- Scalability: Cassandra's ability to scale horizontally is one of its great benefits, which help with handling a growing user base from 1 million to over 100 million users.
- High Throughput: It efficiently manages high rates of read and write operations, which are common in both voting activities and user interactions with movies and watchlists.
- Fault Tolerance: Cassandra's replication mechanisms provide high availability and resilience, which is crucial for maintaining a smooth user experience.
- Simplified Architecture: Using a single database reduces the complexity of managing multiple systems, making development, deployment, and maintenance more straightforward.

When users vote for movies, update their profiles, or add movies to their favorites list, Cassandra can handle these operations with high efficiency. This ensures that all user actions are processed quickly, maintaining a consistent and responsive application experience.

Cassandra for Voting and Movie Selection Service

Due to the high rates of read and write, Cassandra is very suitable option for the Voting service. In a scenario with millions of users creating watch parties and the service containing algorithm that determines the movie to watch a lot of information will need to be processed. Cassandra's architecture ensures that data writes are handled efficiently, even under the strain of high traffic.

The real strength of Cassandra comes with its ability to scale horizontally. It can scale out across multiple nodes to distribute load effectively as user base grows. This capability is very important to keep the voting system's responsiveness. It ensures that user interactions and algorithm for choosing a movie run smoothly. Because of its robust replication mechanism, the data availability and fault tolerance are increased, because data us replicated across multiple nodes. This makes the possibility of process disruptions less likely.

PostgreSQL for User Management and Favorites Management

For the handling of highly structured data, PostgreSQL is very suitable. These may be the cases like the relationships in the user profiles and the favorites list. It provides many data integrity features, including foreign keys, transactional integrity and sophisticated locking mechanisms. These features are very important for the User Management and Favorites Management services, where maintaining data accuracy and consistency is priority.

For instance when a user updates their profile or adds movies to their favorites list, PostgreSQL ensures that these transactions are processed reliably. This guarantees that the system reflects a consistent state, even if multiple users are updating their preferences at the same time. PostgreSQL's support for complex queries also allows for efficient retrieval of personalized data, enabling features like customized movie recommendations based on user preferences and search history.

Integrating Both Databases with Microservices

To integrate Cassandra and PostgreSQL databases in a microservice architecture means that the communication and data flow mechanisms used are efficient and secure. This is accomplished by using API Gateway that routes the request from the client to the services that are responsible for the actions needed. By using API Gateway the development and maintenance processes of each microservice are simplified and also enhances the security aspect by centralizing authentication and rate limiting.

For the consistency of the data and the integrity between the services when using different databases, I have implemented an event-driven architecture. This means that I have a message broker (RabbitMQ), which is used to store messages that the services can send or receive. This communication is asynchronous and helps decouple the services, which allows them to operate independently while still maintaining data synchronization.

Conclusion

By simplifying the database architecture to use only Cassandra, Movimingle can achieve high performance, reliability, and scalability without the unnecessary complexity of managing two different database systems.