Research

Individual Project Semester 6
Veronika Valeva 4090349
Fontys, Eindhoven
25/4/2023

Distributed Data and Scalability – Research in the means of the Quiz Game.

Table of Contents

Introduction to the case

Distributed systems are becoming increasingly common in enterprise software development, and there is a growing need for effective communication and data sharing between different services and components. In the case of the Quiz Game there is a bottleneck in the sharing of data between the Quiz Game Logic Service and the Quiz Game Scoreboard Service. The issue comes from the way the workflow is constructed. When a user registers in the game his data is saved in SQL database connected to the Quiz Game Logic Service, this user has to be saved in the NoSQL database connected to the Quiz Game Scoreboard Service. Moreover, when this user plays quiz games and builds up/ collects score point – his/her gameplay has to be saved in the SQL database connected to the Quiz Game Logic Service and his/her current game point score has to be saved/updated in the NoSQL database connected to the Quiz Game Scoreboard Service. The data saved by the two services seems the same, however, the two services operate with different parts of the user data. Moreover, the two services use different structures of data storing and accessing. The two services need to share user data in a consistent and efficient way. This requires a mechanism for asynchronous communication between the services that can handle varying message sizes and delivery rates, while ensuring that messages are delivered reliably and without loss. From a scalability point of view, the Quiz Game Logic Service needs to be able to handle a large number of concurrent user requests, while the Quiz Game Scoreboard Service needs to be able to handle a large volume of updates to user data. This requires a mechanism for load balancing and workload distribution that can scale horizontally and distribute processing across multiple instances of a service.

RabbitMQ

RabbitMQ is a distributed message broker that enables asynchronous communication between applications or services in a distributed environment. It provides a robust messaging infrastructure that can handle large volumes of messages, making it a popular choice for building scalable and reliable distributed systems. When it comes to distributed data, RabbitMQ can help in several ways. Firstly, it allows messages to be sent and received asynchronously, which means that data can be distributed across different services without the need for tight coupling between them. This makes it easier to build distributed systems that are flexible and adaptable, as each service can operate independently and can be scaled up or down as required. In addition, RabbitMQ provides features such as message persistence and delivery guarantees, which are essential for ensuring that data is reliable and available when required. Messages can be stored in RabbitMQ queues until they are successfully delivered to the intended recipient, ensuring that data is not lost if a service goes down or a network connection is lost. Finally, RabbitMQ can help to ensure that distributed data is compliant with legal and ethical requirements. For instance, GDPR requires that personal data is processed in a manner that is lawful, transparent, and secure. RabbitMQ can help in building a distributed system that is GDPR-compliant by ensuring that personal data is stored securely and is only accessible by authorized services.

How can RabbitMQ ensure Distributed Data?

GDPR requires that personal data is processed in a lawful, transparent, and secure manner. This means that data controllers and processors must take appropriate technical and organizational measures to ensure that personal data is protected against unauthorized access, use, or disclosure. In a distributed system, RabbitMQ can help to ensure that personal data is processed and stored securely by implementing appropriate security measures. Here are some examples of how RabbitMQ can be used to achieve GDPR compliance:

- RabbitMQ supports SSL/TLS encryption for secure communication between services. By enabling SSL/TLS, messages can be transmitted over a secure channel, ensuring that personal data is not intercepted or compromised during transmission.
- RabbitMQ provides authentication and authorization mechanisms that can be used to control access to queues and messages. By using access control, you can ensure that only authorized services have access to personal data, reducing the risk of unauthorized access or disclosure.
- RabbitMQ supports message-level encryption, which can be used to encrypt sensitive data such as personal data. By encrypting messages containing personal data, you can ensure that the data is protected even if it is intercepted by unauthorized parties.
- RabbitMQ provides message persistence, which can be used to ensure that messages containing personal data are not lost due to system failures or crashes. By retaining messages in durable queues, you can ensure that personal data is available when required, reducing the risk of data loss or corruption.
- RabbitMQ provides extensive monitoring capabilities that can be used to monitor compliance with GDPR requirements.

In conclusion, RabbitMQ can help to ensure that distributed data is compliant with legal and ethical requirements such as GDPR by providing a robust messaging infrastructure that enables secure communication, access control, message encryption, message retention, and compliance monitoring. By using RabbitMQ to build a distributed system, the Quiz Game can ensure that personal data is processed and stored securely, reducing the risk of unauthorized access or disclosure.

How can RabbitMQ ensure Scalability?

RabbitMQ is well-suited for building scalable architectures because it provides a robust messaging infrastructure that can be used to build distributed systems. By using RabbitMQ to decouple services and communicate asynchronously, you can build a system that can scale horizontally by adding additional services or nodes as needed. Here are some examples of how RabbitMQ can be used to build scalable architectures:

- RabbitMQ supports load balancing through the use of message queues. By distributing workloads across multiple queues and services, you can ensure that each service is handling an equal share of the workload. This can help to prevent bottlenecks and ensure that the system can handle increasing amounts of traffic.
- RabbitMQ supports fault tolerance through the use of clustering and replication. By clustering multiple RabbitMQ nodes together, you can ensure that the system can

continue to operate even if one or more nodes fail. Additionally, by replicating messages across multiple nodes, you can ensure that messages are not lost due to system failures.

- RabbitMQ supports elastic scaling through the use of message queues and auto-scaling policies. By monitoring queue lengths and service utilization, you can automatically add or remove services as needed to ensure that the system can handle increasing amounts of traffic.
- RabbitMQ is well-suited for building microservices architectures, which involve breaking down monolithic applications into smaller, independent services that can be developed and deployed independently. By using RabbitMQ to communicate between microservices, you can ensure that each service is decoupled and can be scaled independently as needed.

In conclusion, RabbitMQ is a powerful tool for building scalable architectures that can handle increasing amounts of traffic and workloads. By using RabbitMQ to build distributed systems that are decoupled and asynchronous, the Quiz Game can ensure that your system can scale horizontally by adding additional services or nodes as needed.

RabbitMQ in the means of the Quiz Game

The Quiz Game consists of couple of services, two of which the Quiz Game Logic Service and the Quiz Game Scoreboard Service. To ensure that these services can operate independently and asynchronously, RabbitMQ messaging was implemented using the Routing pattern.

By using RabbitMQ messaging, I have decoupled the Quiz Game Logic Service and Quiz Game Scoreboard Service. This means that updates to user data in the Quiz Game Logic Service are communicated to the Quiz Game Scoreboard Service via messages, which can be stored in a NoSQL database connected to the Scoreboard Service. This decoupling ensures that each service has its own but different copy of the data, which helps to improve availability and reliability.

Using RabbitMQ messaging has also enabled asynchronous communication between the services. This means that messages can be sent and received without requiring immediate responses, which can help to improve the overall performance and efficiency of the system.

Since messages are sent every time a new user registers or updates their score, this ensures that the data stored in the NoSQL database connected to the Quiz Game Scoreboard Service is consistent with the data in the Quiz Game Logic Service. This helps to ensure that users can view accurate and up-to-date data on the Quiz Game Scoreboard.

Using RabbitMQ messaging also contributes to the scalability of our Quiz Game. By enabling the Quiz Game Scoreboard Service to scale horizontally, we can add more Scoreboard Service instances as needed to handle increasing amounts of traffic. By distributing workloads across multiple Quiz Game Scoreboard Service instances using RabbitMQ message queues, we can ensure that each service is handling an equal share of the workload, which helps to prevent bottlenecks and ensure that the system can handle increasing amounts of traffic.

Finally, by using clustering and replication in RabbitMQ, we can ensure that the Quiz Game Scoreboard Service can continue to operate even if one or more nodes fail. This helps to ensure that the system can handle failures and maintain high availability.

Overall, our implementation of RabbitMQ messaging using the Routing pattern has contributed to the distributed data and scalability of our Quiz Game, ensuring that our system can handle increasing amounts of traffic and data while maintaining high availability and reliability.

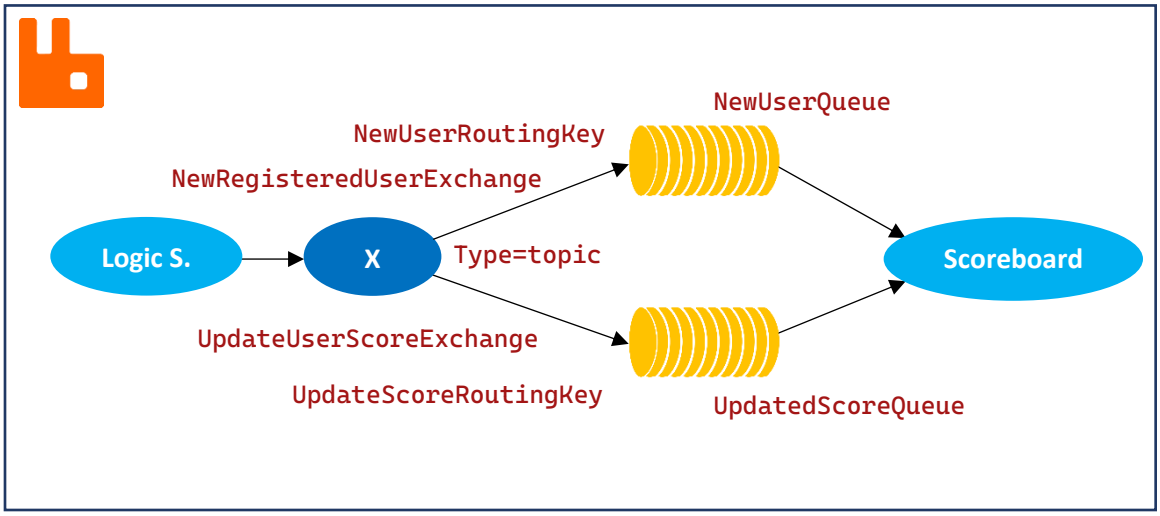

Figure 1 - A Diagram of the RabbitMQ Messaging implementation. The produces - Quiz Game Logic Service using 'topic' exchange sends data to two different queues. The Consumers of the two queues are both the Quiz Game Scoreboard Service. A message – containing user/score data is sent to a queues whoose binding keys exactly matches the routhing queues of the message.

Conclusion

In conclusion, using RabbitMQ in the Quiz Game has enabled asynchronous communication between services, improved scalability, reliability, and compliance with legal and ethical requirements for distributed data.