Applied Cloud Programming Essay (coursework 2)

Table of Contents

1. What to consider when implementing a microservice architecture approach?

1.1 Introduction

The term ‘microservice architecture’ refers to a method of application development in which the application is divided into a series of independent services [1]. In this setup, every microservice operates as an autonomous service designed to perform a specific feature of the application and manage distinct tasks. However, there are numerous factors to consider when adopting a microservices approach. This essay will cover some of these important considerations, particularly focusing on the strategies suggested by ChatGPT. The aim of this essay is to thoroughly examine the accuracy of the solutions proposed by ChatGPT, showcasing why they are sensible.

1.2 Main arguments

The main six ideas proposed by ChatGPT are ***business domain decomposition***, ***service boundaries***, ***data management***, ***communication protocols***, ***fault tolerance and resilience*** and ***team organization and culture***. However, in this section, we are going to only consider two of these aspects, namely ***service boundaries*** and ***communication protocols***. We believe these aspects cover a wide range of considerations for implementing a microservice architecture, whilst also allowing for an in-depth analysis per aspect.

1.2.1 Service Boundaries

Service boundaries in a microservice architecture define how these services interact with each other and what responsibilities each service has. In relation to service boundaries, ChatGPT’s suggestion was to define clear boundaries for each microservice to ensure loose coupling and high cohesion, thus ensuring lower complexity and higher scalability.

This suggestion is perfectly reasonable, as a set of clear boundaries is curcial for ensuring scalability [2]. For instance, incorrect boundary settings can easily undermine the benefits of a microservice architecture by causing very tight coupling. Moreover, vaguely defined boundaries can result in functionality overlaps and data consistency challenges across services, further hindering efficiency and scalability [3].

Given these problems, we can clearly see how strategic boundary delineation preserves architectural integrity and helps facilitate seamless functionality. A clear illustration of how clear service boundaries are hugely beneficial can be seen in the financial sector. In this sector, adaptibility and scalability are crucial for gaining a competitive advantage and retaining customer satisfaction. This is why banking applications that use a microservices architecture employ very precise service boundaries, as they enable independent scaling for services such as account management, fraud detection, and credit risk assessments to name a few.

Based on these examples, we can see how ChatGPT's recommendation for establishing clear service boundaries within microservices architecture aligns with best practices for achieving scalability and reducing complexity.

1.2.2 Communication Protocols

Communication protocols in a microservice architecture determine how microservices communicate with each other. ChatGPT's suggestion with regards to communication protocols is to consider factors such as latency, reliability, and consistency when deciding between synchronous and asynchronous communication, as well as to use lightweight protocols such as *HTTP/REST* for efficient and scalable communication.

Although the consideration of latency, reliability and consistency is key for ensuring a well-fitted protocol, using lightweight protocols may in fact result in more difficulties. For instance, *HTTP/REST* may not be suitable for internal communications due to the additional completixity it brings to microservices interactions [4]. In the case of Internet of Things (IoT) applications, a protocol like MQTT would be preferred due to its efficiency when handling data, whereas *HTTP/REST* would be better when dealing with web applications [5].

Given that ill-fitted communication protocols can result in challenges with system complexity and scalability [6], careful protocol selection is vital for fully leveraging the benefits of a microservices architecture. Therefore, ChatGPT's recommendation of always using lightweight protocols may not be the most sensible, as a more complex protocol that is tailored to the system's requirements and use cases may in fact result in less difficulties and lower complexity.

1.3 Conclusion

Throughout this essay, we explored two critical aspects of implementing a microservice architecture, focusing on ChatGPT's recommendations for the necessity for clear service demarcation and the careful selection of communication protocols. We saw how these recommendations generally align with best practices for scalability and complexity reduction, although additional factors must also be considered. As microservice architectures continue to evolve, balancing their scalability and flexibility against potential challenges such as system fragmentation and increased management complexity will be of utmost importance. Future advancements must navigate these considerations, ensuring that microservices still provide an opportunity for organizations to innovate their software architecture, fostering more adaptive, resilient, and customer-focused solutions.

References for Q1

1. Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M., 2016. Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc.".

2. Matias, T., Correia, F.F., Fritzsch, J., Bogner, J., Ferreira, H.S. and Restivo, A., 2020. Determining microservice boundaries: a case study using static and dynamic software analysis. In Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings 14 (pp. 315-332). Springer International Publishing.

3. Mattila, M. and Hanin, A., 2014, October. The real value of electronic banking. In *Proceedings of the 2000 Academy of Marketing Science (AMS) Annual Conference* (pp. 398-402). Cham: Springer International Publishing.

4. Kumar, P.K., Agarwal, R., Shivaprasad, R., Sitaram, D. and Kalambur, S., 2021, September. Performance characterization of communication protocols in microservice applications. In 2021 International Conference on Smart Applications, Communications and Networking (SmartNets) (pp. 1-5). IEEE.

5. Grgić, K., Špeh, I. and Heđi, I., 2016, October. A web-based IoT solution for monitoring data using MQTT protocol. In *2016 international conference on smart systems and technologies (SST)* (pp. 249-253). IEEE.

6. Söylemez, M., Tekinerdogan, B. and Kolukısa Tarhan, A., 2022. Challenges and solution directions of microservice architectures: A systematic literature review. *Applied sciences*, *12*(11), p.5507.

2. How could you use Kafka in an event-driven service-based architecture?

2.1 Introduction

In an event-driven architecture, decoupled services interact and communicate through changes in state or updates (otherwise known as events) [1]. This architectural style is common in modern applications that are built with microservices. Apache Kafka, on the other hand, is a distributed platform designed for event storage and stream processing. Its goal is to offer a platform capable of high throughput and low latency for processing live data feeds [2]. In an event-driven service-based architecture, Kafka can serve as a powerful messaging system facilitating communication between microservices through the use of events. Throughout this essay, we will explore the utilization of Kafka within an event-driven architecture, particularly focusing on the solutions proposed by ChatGPT. The aim of this essay is to thoroughly examine the accuracy of these solutions and assess how sensible they are.

2.2 Main arguments

The main four ideas proposed by ChatGPT are are ***Kafka's role as an event bus***, ***its tools for event sourcing***, ***its potential for event-driven workflow orchestration*** and ***its ability to provide integration with external systems***. However, in this section, we are going to only consider two of these aspects, namely ***Kafka's role as an event bus*** and ***its tools for event sourcing***. We believe these aspects cover a wide range of considerations for the uses of Kafka in an event-driven service-based architecture, whilst also allowing for an in-depth analysis per aspect.

2.2.1 Event Bus

ChatGPT argues in favour of using Kafka as a centralized event bus, as it enables asynchronous communication between microservices: each microservice can publish events to Kafka topics, and other microservices can subscribe to these topics to consume the events. This setup would help decouple the producers and consumers of events, in turn allowing for greater flexibility and scalability [3].

However, despite these benefits, navigating Kafka's complexities requires deep insight into system design to prevent congestion and avoid errors. For instance, one major challenge of decoupled systems is that they struggle to achieve strong data consistency [4]. This could be very costly in cases where immediate consistency across events is needed, such as e-commerce, which requires managing vast event volumes to ensure a smooth user experience [5].

Given these concerns, appropriate steps must be taken in order to effectively leverage Kafka as a centralized event bus. Although it has the potential for providing high felxibility and scalability, strategic planning and continuous monitoring are very much crucial to avoiding latency and maintaining operational integrity. Therefore, this calls for a more calculated approach when using Kafka as an event bus in an event-driven service-based architecture.

2.2.2 Event Sourcing

Besides acting as a centralized event bus, ChatGPT also argues that Kafka can be used for implementing event sourcing, where changes to the state of a microservice are captured as a sequence of events. These events would be stored in Kafka topics and could be replayed to reconstruct the state of the microservice at any point in time, thus enabling auditability, traceability, and the ability to recover from failures [6].

There are undeniable benefits to using Kafka for event sourcing. For instance, Kafka's ability to provide a comprehensive log supports both transparency as well as compliance with strict regulations [7]. Additionally, the added traceability and failure recovery are critical in industries such as finance, where the inability to recover lost data can have a large negative impact that may not be limited to one company [8].

However, in the context of event sourcing, Kafka introduces new challenges pertaining to the high storage requirements and the complexity of managing event replays. Although there has been a decrease in storage costs, the large volume of data, together with the necessity for efficient processing strategies, would still result in notable operational costs and require considerable computational power [9].

Therefore, leveraging Kafka's event sourcing effectively goes beyond recognizing its benefits: a thorough grasp of system dynamics, as well as meticulous data management, are both needed to effectively navigate the added complexities in order to fully harness the advantages provided by Kafka. Similar to the event bus implementation, this calls for a more strategic approach than that proposed by ChatGPT.

2.3 Conclusion

This essay has shown that Kafka can play a central role in event-driven architectures, enabling effective asynchronous communication between microservices as well as traceability and the ability to recover from failures. Future Kafka advancements promise enhanced scalability, fault tolerance, and data handling efficiencies, which are all vital for applications and sectors demanding quick responsiveness, such as streaming or the Internet of Things (IoT). Nonetheless, there still exist challenges that need addressing, such as complex data management and hight storage requirements. These challenges could, however, be addressed by recent machine learning and AI advancements in system optimisation, which can in turn help pave the way for leveraging Kafka to create dynamic, intelligent systems ready to meet the digital demands of the future.

References for Q2

1. Michelson, B.M., 2006. Event-driven architecture overview. *Patricia Seybold Group*, *2*(12), pp.10-1571.

2. Garg, N., 2013. *Apache Kafka* (pp. 30-31). Birmingham, UK: Packt Publishing.

3. Kul, S., Tashiev, I., Şentaş, A. and Sayar, A., 2021. Event-based microservices with Apache Kafka streams: A real-time vehicle detection system based on type, colour, and speed attributes. *IEEE Access*, *9*, pp.83137-83148.

4. Yu, L., Su, S., Luo, S. and Su, Y., 2008, June. Completeness and consistency analysis on requirements of distributed event-driven systems. In *2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*(pp. 241-244). IEEE.

5. Pal, G., Li, G. and Atkinson, K., 2018, October. Big data real-time clickstream data ingestion paradigm for e-commerce analytics. In *2018 4th International Conference for Convergence in Technology (I2CT)* (pp. 1-5). IEEE.

6. Rishaug, R. and Wika, O.L., 2018. *Event Sourcing* (Bachelor's thesis, NTNU).

7. Scott, D., Gamov, V. and Klein, D., 2022. *Kafka in Action*. Simon and Schuster.

8. Bryzgalova, S., Lerner, S., Lettau, M. and Pelger, M., 2022. Missing financial data. *Available at SSRN*, *4106794*.

9. Rybicki, J. and JSC, J.S.C., 2018. Application of event sourcing in research data management. *ALLDATA*, pp.22-26.