ACIT 3855 – Assignment 1 – Microservices In Industry

Mike Mulder

Markus Afonso – Set C

February 22, 2024

Microservices Definition

Microservices Architecture (MSA) is a software development approach where a complex application is decomposed into smaller, independently deployable services, each focused on a specific business function. These services are designed to be loosely coupled, meaning they can be developed, deployed, and scaled independently (What is Microservices Architecture?, n.d.). Communication between services typically occurs through lightweight protocols like HTTP or messaging queues. Each microservice is responsible for its own data storage, ensuring autonomy and resilience (Communication in a microservice architecture, 2022). Overall, MSA aims to improve agility, scalability, and maintainability by breaking down monolithic applications into smaller, manageable components .

When to use them

MSA is especially useful for large, complex systems with different deployment and scaling needs that consist of several interconnected components. Dividing the monolithic architecture of such systems into microservices has multiple advantages. First of all, it makes it possible for teams to work independently on specific components, which speeds up the development and deployment cycles. Because one microservice can be created, tested, and deployed independently of the others, it becomes easier to coordinate changes throughout the entire system. Furthermore, because improvements to one service are less likely to affect other services, the modular design of microservices makes maintenance and upgrades simpler. Better fault isolation is another benefit of this modular design, since it reduces the possibility that system-wide failures may originate from a single microservice. Agility, scalability, and maintainability are generally increased by breaking down complex systems into microservices making it easier to adapt to evolving business requirements and technological changes.

Many modern applications, particularly those with rapidly growing workloads or varying demands, have scalability as their primary need. Because of its inherent flexibility and granularity, MSA offers substantial advantages in handling scaling requirements. MSA allow individual services to be scaled horizontally as needed without affecting the application as a whole, in contrast to monolithic systems where the entire program must be scaled as a single unit. This allows for the dynamic allocation of resources to specific services in response to demand, resulting in more efficient resource usage and improved management of varying workloads. Also, auto-scaling mechanisms—in which services dynamically modify their capacity in response to variations in workload patterns or traffic—are made easier to deploy with the help of microservices. Scalability issues can be separated from the rest of the system by MSA, which allows businesses to grow their applications more effectively, ensuring the most optimal performance and resource utilization even under high load conditions.

The diverse array of technologies, platforms, and programming languages that applications must interface with is common in today's diverse computing configurations. The versatile MSA offers a way to integrate many technology stacks into a single application. Without being influenced by the decisions made for other system components, each microservice can be developed using the tools and technologies most suited for its unique set of needs. This flexibility helps teams make the most of their knowledge of several technologies, which boosts developer productivity and innovation. Additionally, it makes it easier for companies to implement new frameworks and technologies because they can make small, gradual modifications to individual microservices without disrupting the system as a whole. MSA encourages modularity, extensibility, and future-proofing by separating technology decisions from the overall architecture. This allows businesses to adapt to changing technological landscapes and market trends more effectively.

When not to use them

In simple, small systems with low scalability needs and little functionality, the cost of setting up and maintaining microservices could outweight the advantages. In these situations, a monolithic design might be a better choice. Due to the tight integration of all components into a single codebase, a monolithic architecture provides simplicity and easier initial development. Developers may concentrate on creating and delivering the application instead of adding the hassle of managing many services when there are fewer moving components. Additionally, for straightforward applications that don't need scalability or regular changes, the overhead of deploying and monitoring microservices might not be warranted.

Dividing the application into microservices could add needless complexity if there are significant connections between parts of it. When components share resources or have strong interdependencies, there may be a tight coupling between them. This could make it difficult to maintain transactional consistency, communication, and overall system performance. A monolithic architecture might be more suitable in these situations since it enables tighter component integration, makes data exchange easier, and guarantees transactional consistency. Splitting tightly linked components into microservices could lead to more expense and complexity without a noticeable improvement in scalability or maintainability.

Developing and maintaining a microservices architecture requires additional expertise, infrastructure, and operational overhead compared to a monolithic architecture. Organizations with limited resources, such as budget constraints, a shortage of skilled personnel, or inadequate infrastructure, may struggle to adopt and manage MSA effectively. Implementing microservices involves setting up and managing a distributed system, which requires expertise in containerization, orchestration, and cloud infrastructure. Additionally, ensuring the availability, reliability, and security of microservices applications demands ongoing monitoring, maintenance, and support. Organizations lacking the necessary resources may find it challenging to invest in the infrastructure, training, and ongoing support required for successful adoption and management of microservices.

In scenarios where multiple services need to access the same data or maintain consistency across transactions, administrating data sharing and synchronization in a microservices environment can be hard. Microservices typically encapsulate their own data storage, leading to data duplication and potential inconsistencies between services. Ensuring data consistency and integrity across distributed services requires implementing complex coordination mechanisms, such as distributed transactions or eventual consistency models. This complexity may outweigh the benefits of decomposition, particularly if the application's primary focus is on data processing and consistency. In such cases, a monolithic architecture may offer better control over data access and consistency since all components share the same data storage and processing logic.

Company 1 (Success Story) – Uber

Uber is a multinational transportation network company that operates a mobile application connecting passengers with drivers for hire. The company offers a range of services, including ride-sharing, food delivery, package delivery, and bicycle sharing. Uber's software platform facilitates millions of transactions daily, coordinating the movement of passengers and drivers across various locations worldwide (Blystone, 2024).

Uber utilizes microservices architecture to power its diverse range of services, including its flagship ride-sharing platform, food delivery service (Uber Eats), and other ancillary offerings. Each service provided by Uber, such as ride requests, driver allocations, payment processing, and trip tracking, is encapsulated within individual microservices. These microservices work together cohesively to deliver a seamless and reliable user experience across multiple platforms and devices (Gancarz, 2023).

Uber's adoption of microservices involves cross-functional teams responsible for developing, deploying, and maintaining specific microservices. These teams are organized around business capabilities, with each team having end-to-end ownership of the services they manage. Uber's technology stack includes a wide range of tools and technologies tailored to the requirements of each microservice, including languages like Java, Python, Go, and frameworks like Spring Boot and Flask. Additionally, Uber employs containerization technologies such as Docker and orchestration platforms like Kubernetes to manage and scale its microservices efficiently (Gancarz, 2023).

Uber chose to adopt a microservices architecture to address the challenges associated with managing its rapidly growing and diverse set of services. By decomposing its monolithic application into smaller, independently deployable services, Uber gained greater flexibility, scalability, and resilience. Microservices allowed Uber to innovate rapidly, scale its services independently, and adapt to changing market dynamics more effectively.

Company 2 (Failure Story) – ExpressRide

ExpressRide is a fictional transportation startup company that aimed to compete with established ride-hailing platforms like Uber and Lyft. Founded with the vision of providing fast, reliable, and affordable transportation services, ExpressRide pursued to distinguish itself through innovative technology solutions and superior customer experiences. ExpressRide's business model closely resembled that of its competitors, focusing on providing on-demand transportation services to customers through a mobile app platform. The company aimed to connect riders with drivers in their vicinity, offering a range of vehicle options to meet diverse transportation needs. ExpressRide operated in select cities, targeting urban areas with high demand for transportation services.

ExpressRide initially adopted MSA with the intention of building a scalable, flexible, and responsive platform capable of competing with industry giants like Uber. However, the implementation of microservices at ExpressRide encountered significant challenges, leading to operational issues and ultimately contributing to the company's failure.

ExpressRide's implementation of microservices involved a decentralized approach, where different teams were responsible for developing and maintaining individual microservices corresponding to specific functional components of the platform. Each microservice was developed using modern technologies and frameworks, such as Node.js, Docker, and Kubernetes, to ensure agility, scalability, and resilience. Teams followed agile development methodologies and DevOps practices to iterate rapidly and deploy changes frequently.

ExpressRide chose to adopt microservices architecture (MSA) primarily to achieve scalability, flexibility, and agility in its software development and operations. By decomposing the monolithic application into smaller, independently deployable services, ExpressRide aimed to accelerate development cycles, improve fault tolerance, and scale its platform to meet growing demand. Additionally, MSA offered the promise of easier maintenance, better resource utilization, and faster time-to-market, aligning with ExpressRide's business objectives and competitive strategy.

The decentralized nature of microservices architecture introduced significant complexity overhead for ExpressRide. With multiple teams responsible for developing and maintaining individual microservices, coordination and integration efforts became challenging. Ensuring consistency, reliability, and performance across the platform required extensive coordination between teams and services. This complexity made it difficult to identify and address issues promptly, leading to operational inefficiencies and increased maintenance overhead. Despite efforts to streamline communication and collaboration, the inherent complexity of managing a large number of microservices remained a persistent challenge for ExpressRide.

As ExpressRide's platform evolved and the number of microservices increased, managing dependencies and communication between services became increasingly complex. Changes made to one microservice often triggered cascading impacts on other dependent services, resulting in unpredictable behavior and degraded user experiences. ExpressRide struggled to maintain a clear understanding of service dependencies, leading to challenges in versioning, compatibility, and deployment. Despite efforts to implement service isolation and encapsulation, interdependencies between services remained a significant hurdle, hindering the company's ability to deliver reliable and consistent services to its customers.

While microservices architecture promised scalability, ExpressRide encountered challenges in scaling its platform efficiently to meet growing demand. Bottlenecks and performance limitations in critical services hindered the company's ability to handle increased traffic and workload effectively. Inadequate resource allocation, inefficient load balancing, and poor service orchestration exacerbated scalability issues, leading to degraded system performance and increased response times during peak demand periods. Despite investing in infrastructure upgrades and optimization efforts, ExpressRide struggled to achieve the level of scalability required to support its expanding user base, impacting customer satisfaction and retention.

Managing a large number of microservices distributed across multiple environments posed significant operational challenges for ExpressRide. The complexity of deploying, monitoring, and troubleshooting microservices in production environments strained the company's resources and capabilities. Manual intervention was often required to address issues such as service failures, performance degradation, and resource contention, leading to increased downtime and maintenance costs. Additionally, the lack of standardized processes and tools for managing microservices further compounded operational challenges, hindering efficiency and scalability. Despite efforts to implement automation and streamline operations, ExpressRide struggled to cope with the operational overhead associated with managing a microservices-based architecture, impacting its ability to deliver reliable and responsive services to its customers.

ExpressRide made a poor choice in adopting MSA primarily due to the complexity overhead and operational challenges it introduced. While MSA promised scalability, flexibility, and agility, ExpressRide struggled to effectively manage the decentralized nature of microservices, leading to difficulties in coordination, integration, and maintenance across multiple teams and services. Additionally, the interdependencies between microservices and scalability issues further compounded operational challenges, resulting in unpredictable behavior, service outages, and degraded user experiences. Ultimately, the decision to adopt MSA proved detrimental to ExpressRide's success, highlighting the importance of careful planning and consideration when implementing distributed systems architectures.

Compare and Contrast

Both Uber and ExpressRide adopted MSA with the aim of achieving scalability, flexibility, and agility in their respective transportation platforms. They both decomposed their monolithic applications into smaller, independently deployable services to enable faster development cycles, improve fault tolerance, and scale their platforms to meet growing demand. Additionally, both companies leveraged modern technologies and frameworks to implement microservices, such as Docker, Kubernetes, and Node.js, to ensure agility, scalability, and resilience. Furthermore, both Uber and ExpressRide faced challenges related to complexity overhead, service interdependencies, and operational scalability as they scaled their microservices architectures

Despite these similarities, there are notable differences in how Uber and ExpressRide approached the implementation and management of microservices architecture. Uber, as an established industry leader, had the advantage of experience and resources to navigate the complexities of MSA effectively. The company invested heavily in building a robust microservices ecosystem, with dedicated teams and infrastructure to support its platform's growth. In contrast, ExpressRide, as a startup, lacked the expertise, resources, and infrastructure necessary to manage the complexities of MSA successfully. The company struggled with operational challenges, scalability issues, and service interdependencies, ultimately leading to its failure.

While Uber's use of MSA proved to be effective and appropriate, ExpressRide's experience highlights the importance of careful planning and consideration when adopting distributed systems architectures. Uber's successful implementation of microservices architecture enabled the company to achieve scalability, flexibility, and agility, enhancing its competitiveness and customer satisfaction. The company's use of MSA aligns with the defined use cases, including complex systems, scalability requirements, and diverse technology stacks. In contrast, ExpressRide's poor choice in adopting MSA resulted in operational challenges and limitations that challenged its success. The company's struggles with complexity overhead, service interdependencies, and scalability issues highlight the risks associated with adopting MSA without adequate expertise, resources, and infrastructure. Therefore, while MSA can be effective and appropriate when implemented correctly, it requires careful planning, execution, and management to realize its full potential and deliver tangible benefits to organizations.

References

Blystone, D. (2024, February 14). *The History of Uber*. Retrieved from Investopedia: https://www.investopedia.com/articles/personal-finance/111015/story-uber.asp

ChatGPT. (n.d.). Retrieved from OpenAI: https://chat.openai.com/

Communication in a microservice architecture. (2022, April 14). Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

Gancarz, R. (2023, October 18). *Uber Migrates 4000+ Microservices to a New Multi-Cloud Platform Running Kubernetes and Mesos*. Retrieved from InfoQ: https://www.infoq.com/news/2023/10/uber-up-cloud-microservices/

What is Microservices Architecture? (n.d.). Retrieved from Google Cloud: https://cloud.google.com/learn/what-is-microservices-architecture