Service Oriented Architectures

Microservices in Industry

Assignment 1

ACIT2855

Adel Kuanysheva
A01258780
October 16th, 2022

1 - Microservices Definition

A microservices architecture is a type of application setup that is developed as a series of separate small services based on subsets of business logic. Each service is a logical component that has their own unique responsibility, built on their own programming language, use their own data storage techniques, processes, resources, and communicate with each other through secure APIs. The services can also be developed and deployed independently from each other, which makes this an appealing setup for larger application that now more than ever require an architecture to scale depending on demand.

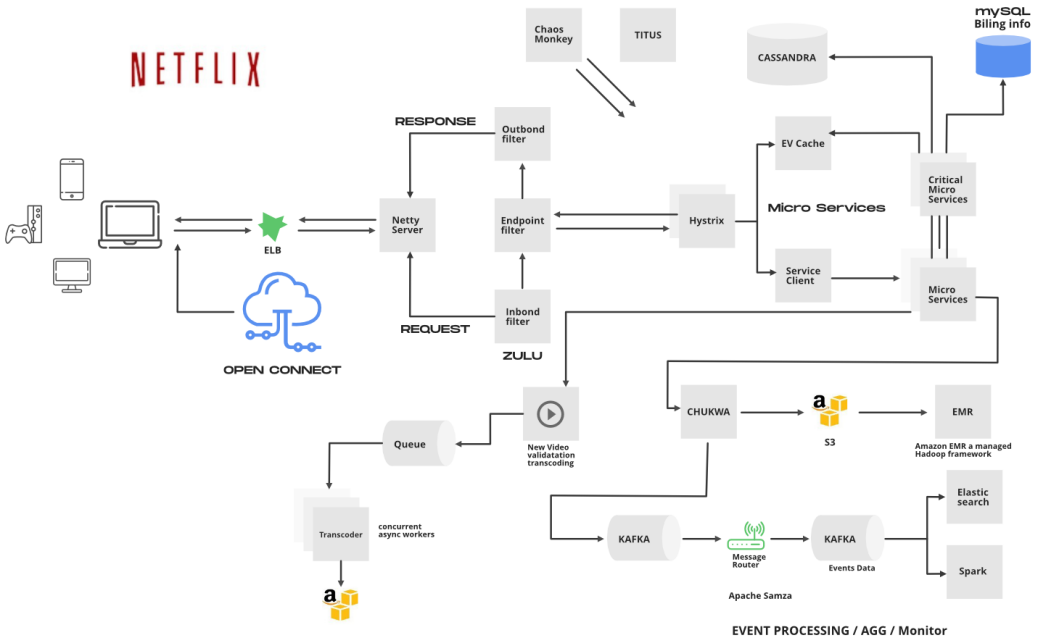
This type of structure benefits large, fast growing, enterprise-like company applications that need to serve many different clients that may have differing operating systems, browsers, and technologies. Since a microservice architecture is "language and technology agnostic" [1] it accomplishes this cross-functionality requirement seamlessly. If a large team of developers were all working on platform of the same service, there would be high risk of merge conflicts and overlapping code. This is why having separate small services is advantageous being that developers can concentrate on individual parts rather than the application as a whole. Another way that this separated service architecture is beneficial is the fact that services are able to keep updated and be replaced with ease. Since communication between services are loosely coupled, there is low dependency when tearing down a system component. For the same reason, adding features and redeployment do not take much time at all since changes are made to a contained, distinct area. An application will not need to be re-deployed in its entirety just to adjust a certain function. This characteristic of easy manipulation is what makes services able to adjust their resources accordingly and the system as a whole can keep up with modern techniques.

There are many situations in which microservices should belong, however this structure should not be implemented just for the sake of being used. Most of the time when you are first developing an application, you should keep it as simple as possible. If an application needs to be deployed as a unit, microservices can add unneeded complexity because deploying multiple artifacts is quite laborious. Microservices are based on business functions, therefore, if there is no way to split a system into unique tasks then the application is most likely better off as a monolith. An example of a system that is better off without a microservices architecture is an authentication system. For instance, if there is a service that requires user login to access information to send out

to another service. But what happens when the development team does not own all the information required by their applications? This over complicates the business process by needing authorization every time information is accessed. Hence, this system and set of business processes should be implemented as a monolith architecture. Microservices are often overkills for new applications and can create communication complexities, be quite expensive to run, and be hard to distribute.

2 - Netflix, the Success Story

Netflix is the most popular online video subscription streaming service in the world that allows its users to watch TV shows and movies without commercials on an internet connected device. [2] Netflix is responsible for 15% of the Internet's bandwidth, spanning over numerous regions with 180M+ users in 200+ countries, this company serves as a fantastic example of a success story related to cloud-based microservices architecture. This high-level system works on two cloud platforms, AWS and Open Connect. It's data centres exist of the public cloud and their original monolithic program was broken up into small services, this change influenced the incredible success of this company. Netflix switched to this architecture below to improve their user experience, creating a reliable structure with no single point of failure. [4]



Netflix Microservices Architecture Diagram [3]

This decision was made after their data centre shutdown for three days due to a service outage. Netflix also uses a very complex system called Hystrix. This helps the platform control the loose communication between services by running it with low latency and fault tolerance.

Some functional requirements that Netflix possess are:

- Allowing users to create accounts and buy a subscription plan,
- Allowing users to have multiple accounts on one plan,
- Allowing users to watch movies and TV shows, and
- Having a platform that allows developers to upload videos to their storage and make it available to their customers.

Some non-functional requirements of Netflix include:

- Minimal wait times (buffering), real time video streaming,
- Reliability, Availability, and Scalability.

The functional requirements are satisfied with the separate business components that are outlined in the structure. Logically, there are 3 main parts that their architecture is broken down into. First, a client device logs in and browses their movie selection. The front end is coded with React JS which is known to be great on loading speeds and performance. Then once a video is selected and played, Open Connect handles this request as a content delivery network (CDN). This entails Open Connect retrieving the video from its nearest server to display on a user's device. Lastly, the Database handles all other actions that do not fall under video streaming such as posting content, processing videos, and distributing them to multiple servers.

The components of the Netflix structure also improve the reliability, availability, and scalability of their product. Availability is partially achieved with their use of load balancers that route traffic to multiple proxy servers. Along with their multi-region AWS operations and OCAs servers, they are able to respond to requests in a small window of time. Reliability ties in with their ability to mitigate the risk of failures. The API Gateway, Zuul, is used to handle requests and perform the dynamic routing of their services so that they can detect and resolve these failures. Netflix's architecture is able to scale using horizontal scaling, parallel execution, and database partitioning. Horizontal scaling is through EC2 instances in AWS, as demand increases, the Netflix is

able to deploy more instances to handle many requests. This is also how parallel execution is executed, with many applications running concurrent requests.

The success is evident for Netflix, in the beginning of 2012 the company decide to adopt this structure because of rapidly increasing data and user information. It was simply impossible for their data centres of their original monolithic approach to withstand the application's popularity, as well as the risk of single point failure. The company did experience lots of challenges during their process of changing to a microservices architecture, taking almost 3 years to finally migrating into the cloud with now over 100 services. This was difficult to carry out for their team due to the fact that the company had to run both cloud services and their original structure in parallel, this costed them with many sunk costs. In tangent, the company had to deal with the transferring of lots of data over from local data centers to the cloud. This causes latency issues on their web pages. This movement however turned out to be the correct decision for Netflix since now they are one of the cloud technology leaders. Netflix made the best choice with changing over to this concept of microservices. They are now able to scale effectively and serve many users without risk of buffering or one point failure. Analyzing their Return of Investments in quite trivial, the company's success continued to skyrocket once they had fully migrated in 2011.

3 - GroundZero, the Failure Story

GroundZero decided to make the move to a microservices architecture after running on a monolithic architecture that they built in their early years of development. GroundZero is a platform that is used by developers to publish, integrate, aggregate, and manage their APIs in one single place. The reason for decomposing their monolith was from necessity. They were experiencing higher than normal requests with their growth in popularity, they needed their system to be able to handle this without any risk of failures.

Some functional requirements that GroundZero possess are:

- Allowing users to publish, integrate, aggregate, and manage their APIs,
- Allowing users to define and perform one-to-many transformations and normalize fields across elements, and
- Allowing users to create workflow templates, synchronize and automate processes across multiple systems.

Some non-functional requirements of GroundZero include:

- Fast response times,
- Low resource usage, and
- Reliability, Availability, and Scalability.

How the company formatted their structure was by using Docker and Minikube to run their services locally and use integrated load balancers to handle requests. This helped them deploy their services faster and allow developers of their products to work more efficiently by separating their workflows. This brought high performance and the ability to forecast failures in operations. However, this structure of their system created massive overhead since the application became increasingly larger in size. GroundZero also split their one system with help from a third-party technology company, hence none of their existing staff were well versed in topics such as loose coupling, smart endpoints, or decentralized data management. These are some of the most important characteristics to microservices, hence an expensive cost in re-training their employees. Their distributed system became too complex and inconsistent. Since each service in their system read and wrote independently, updating their APIs rules and specifications became time consuming and had lots of risk for oversight.

This company overall made a poor choice to migrate to a microservices architecture. GroundZero played into the hype of this structure and perhaps did not thoroughly investigate the impact, cost to benefits and ROI that this change will induce. The benefits of having multiple mini services did not overweigh the money and time spent for implementation and maintenance, instead caused more headaches. It turned out that their original monolith structure was easier to deploy and would have provided a better solution to their business model if it was given more resources. If they had stayed with their monolithic codebase, users of their product would maybe suffer more wait times however but would benefit from less integration steps such as authentication and authorization. The mistake made by GroundZero in this case was that they thought that microservices would turn into a modular system, however instead they were left with the same problem but spread over multiple services.

Netflix vs. GroundZero

Similarities	D*ifferences*
Both companies used microservices to improve their user experience with regards to availability and data integrity.	Netflix thoroughly investigated the cost to benefit ratio before decomposing. GroundZero instead followed the popularity of Microservices in the technology sector.
Both companies decomposed their monolithic architecture so that there were no failures happening from one service alone.	Netflix split up their architecture to 3 logical components, which made sense to their user experience. GroundZero was not able to split up their business requirements effectively.
Both companies deployed their services to run concurrent using load balancing and proxy servers.	GroundZero focused on a user's workflow but instead should have focused on the modularity of their system.
Both companies had large teams that all worked in the same codebase. This made it difficult for developers to merge their code without lots of conflicts.	Netflix created redundancy in their data centers to minimize the risk of failure.
Both companies solved their need for scalability by using load balancers to build and tear services as needed.	Netflix spent longer creating their microservices architecture. However, this meant that their team members were better equipped to maintain their structure
Both companies had architectures that were deployed on the cloud, which meant that they did not require physical resources to operate.	GroundZero relied on a third-party application, hence their team members lacked to knowledge to continue maintenance.

Overall, it is clear that Netflix's approach to microservices architecture was appropriate for their specific business model, "The leadership demonstrated by Netflix can't be overstated." [3] What GroundZero should have done was to compare their business's requirements with Netflix and see how this successful company used microservices to solve their problems. A successful use of MSA is one that is carefully research and simplified when needed. It is important to make sure, at the end of the day, the architecture you are implementing is in fact improving your business instead of hindering it, this might be more difficult than original anticipation. Speaking to the

claims made in Section 1, microservices only succeed when it reflects clear business logic. When a system is decomposed with this clear structure, it is very often that a company will not see the benefits that they expected.

References

[1] Kamaruzzaman, M. (2020, December 15). *Microservice Architecture: A brief overview and why you should use it in your next project*. Medium. Retrieved October 16, 2022, from https://towardsdatascience.com/microservice-architecture-a-brief-overview-and-why-you-should-use-it-in-your-next-project-a17b6e19adfd

[2] Netflix Dev. (n.d.). *What is netflix?* Help Center. Retrieved October 16, 2022, from https://help.netflix.com/en/node/412

[3] System design netflix - A complete architecture. GeeksforGeeks. (2022, August 2). Retrieved October 16, 2022, from https://www.geeksforgeeks.org/system-design-netflix-a-complete-architecture/#:~:text=Netflix's%20architectural%20style%20is%20built,for%20applications20and%20Web%20apps.

[4] Nguyen, C. D. (2020, May 5). *A design analysis of cloud-based microservices architecture at Netflix*. Medium. Retrieved October 16, 2022, from https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f

[5] Scalable microservices at Netflix. challenges and tools of the Trade. Scalable Microservices at Netflix. Challenges and Tools of the Trade | Software Development Conference QCon San Francisco. (n.d.). Retrieved October 16, 2022, from https://qconsf.com/sf2014/presentation/scalable-microservices-netflix-challenges-and-tools-trade.html