IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 6.252

Volume 2, Issue 2, June 2022

Literature Review on: A Modern Approach for Software Development using Microservices

Saurabh Kawli and Lalitkumar Choudhary

Students, Master of Computer Application

Late Bhausaheb Hiray S S Trust's Hiray Institute of Computer Application, Mumbai, India

Abstract: *Building complex systems as a composite of small, loosely linked components that communicate with one another using language-independent APIs is the idea behind microservice-based architecture. Due to its benefits, including higher scalability of deployed systems and increased software development agility, this architectural approach is currently growing in popularity within the industry. Our goal in this work is to compile and organize best practices, problems, and some current solutions to these problems used by practitioners successfully creating microservice-based applications for commercial usage. The microservices paradigm focuses on changing how software is perceived, developed, and built. Scalability is one of the fundamental traits of this new, promising paradigm, as compared to monolithic architecture. In this paper, we will see the approach to microservices and challenges in implementation.*

Keywords: Microservice.

I. INTRODUCTION

Software engineers are becoming more interested in solutions based on the microservice architectural style as its underlying pattern promises to make it possible to create highly reliable and scalable applications with less work than with conventional methods. Adopting a microservices-based architecture has obvious advantages, particularly during the development stage. This new strategy, which differs from monolithic applications by requiring an application to be organized as a collection of distributed, communicative microservices, has accelerated the adoption of agile approaches. Microservices cooperate at runtime to satisfy the program's overall goals (such as efficiency, security, and scalability) based on the design of the application, and an efficient monitoring system is needed to verify whether these goals are actually met. While adopting the microservice architectural style provides a number of advantages in terms of agility, it also makes it more difficult to meet microservice-based applications' monitoring needs. Actually, the diverse characteristics of these microservices and the independence of the teams participating in their development lead to different monitoring needs. Microservices might be CPU-intensive in some situations or storage-intensive in others. As a result, many metrics with various details can be needed. These various requirements have an impact on deployment since the microservices must be situated in a location where the cloud provider's capabilities correspond to the needs of the application owner. Furthermore, because of the scalable and flexible nature of microservice-based applications, the place where they are installed must also provide a scalable monitoring solution. The previous few decades have seen a gradual trend in software architectures toward distribution, modularization, and loose coupling with the aim of producing more code reuse and robustness, ultimately a necessity a requirement imposed by the need to improve software quality, not only in applications that are safety- and money-critical but also in more widely used off-the-shelf software products. The most recent step in this process appears to be referred to as microservice architecture, a form of service-oriented computing that has lately begun to gain prominence to alter how software is viewed, conceived of, and designed. In service oriented architectures, the emphasis was mostly on cross boundaries inter-organization technology-agnostic communication, and on orchestration of business processes. The research community dedicated major attention and effort on foundational aspects, such as correctness and verifiability of service composition. Little effort was spent on defining and bounding the nature of the internal logic of services or on scalability and maintainability issues, concerns that appear to be of major importance for modern organizations. New programming languages based on the new paradigm are emerging in recent years, and allow to describe computation from a data-driven instead of process-driven perspective.

In the last few decades, software architectures have shifted toward distribution, modularization, and loose coupling with the goal of increasing code reuse and robustness, which has become a necessity dictated by the need to improve software quality, not only in safety and financial-critical applications, but also in more common off-the-shelf software packages. Microservice architecture, a style inspired by service-oriented computing that has recently gained popularity and promises to revolutionize the way software is viewed, planned, and created, appears to be the most recent stage in this process. In service-oriented architectures, the focus was primarily on cross-organizational, technology-agnostic communication and business process orchestration. Foundational components of service composition, such as accuracy and verifiability, received a lot of attention and effort from the research community. The nature of the internal logic of services, as well as scalability and maintainability difficulties, which appear to be key challenges for modern businesses, received little attention. In recent years, new programming languages based on the new paradigm have emerged, allowing computation to be described from a data-driven rather of a process-driven perspective. The move to microservices is a touchy subject these days, with some firms undergoing large refactoring's of their back-end systems to fit the new paradigm's ease of use. This is the case, for example, with the system and institution examined in this study, Danske Bank's FX Core. Others, on the other hand, begin their business model by designing software using the microservice paradigm from the beginning. We're in the midst of a huge shift in how software is perceived and intended, as well as how capabilities are structured into components and industrial systems are envisioned. The microservices architecture is built on very simple principles:

- **Bounded Context:** This concept captures one of the key properties of microservice architecture: focus on business capabilities. It combines related functionalities into a single business capability which can be implemented as a service.
- **Size:** This represents a crucial concept for microservices and brings major benefits in terms of service maintainability and extendibility. If a service is too big, it should be split up into two or more services, according to the idiomatic use of microservice architecture, which keeps the granularity and the emphasis on the user's needs providing only a single business capability.
- **Independency:** This concept encourages loose coupling and high cohesion by stating that each service in microservice architectures is operationally independent from others, and the only form of communication between services is through their published interfaces.

II. LITERATURE REVIEW

[1] In this article, they put out a paradigm for evaluation to help businesses decide whether moving to microservices is worthwhile.

We identified a select group of traits and KPIs that businesses should address when they think about converting to microservices. 52 professionals with experience in the development of microservices were interviewed as part of an industry survey to identify these traits.

The answers to a questionnaire that served as the basis for the interviews were asked to indicate which metrics and characteristics had been adopted when they will be transferred to microservices, which of these were useful, and which had not been adopted but ought to have been. Open-ended questions were used to collect the metrics in order to prevent any To prevent any bias in the results caused by a set of prepared answers, the metrics were collected using open-ended questions. Following the open-ended questions, they further questioned the practitioners to see if they had additionally gathered some of the metrics suggested in the research and if they thought it would have been beneficial to do so. The outcome of this effort is an assessment system that will help businesses decide whether or not they need to move. With the aid of this process, they will be able to avoid switching frameworks when it is not necessary, especially if reworking their monolithic system or redesigning their internal structure would yield superior outcomes.

[2] In this essay, we examined the Danske Bank FX Core system as a paradigmatic case study of a mission-critical system. Both the modern microservice architecture and the antiquated monolithic design have been studied. Both architectures' designs, implementations, use of scalability strategies, and degree of scalability have been published. This led to a detailed examination of the implementation of scalable microservice architecture and the potential complexity of the change. The system under discussion in this study was re-engineered, which resulted in decreased complexity, decreased coupling, increased cohesion, and simplified integration. By contrasting the two architectural models, we can observe how microservices improved scalability and provided solutions to the main issues that the monolithic realisation brought about. Although quantitative measurements were not included in the comparison, the adoption of particular methodologies has been cited as evidence in favour of greater scalability.

[3] Practitioners and researchers agree that microservices must be lowly coupled and highly cohesive. The development of microservice-based systems is growing, however at the best of our knowledge; there are no validated metrics to evaluate coupling and cohesion between services. Some researchers (Bogner et al., 2017a) proposed to extend coupling measures adopted for SOA but these measures have never been validated nor used in the microservice domain.
[4] In this paper we have proposed architecture for managing the deployment of microservice-based applications involving several cloud providers. The approach provides a set of modules to manage both application owner requirements and cloud provider monitoring services semi automatically, adopting a MOMILP to identify the best solutions for matching those requirements and monitoring services. This aspect is relevant since monitoring data enable the application owners to analyze the efficiency and effectiveness of the application and to make informed decisions regarding improvements and modifications.
[5] This study used a mixed-method design to gather best practises, lessons learned, and obstacles from practitioners who successfully implemented microservice-based architectures. In-depth interviews with 21 practitioners served as part of the study, and a follow-up survey yielded 37 responses that met our criteria for selection. One of the best practises that contributes to the success of their development processes, according to our participants, is a strong feeling of ownership. Other best practises include rigorous API management, automated processes, and investments in reliable logging and monitoring equipment. They discovered that a variety of linguistic approaches and the recommendation to divide microservices by business functionality are not always successful.
[6] We carried out two qualitative studies to examine industry practises and issues for the assurance of evolvability of microservices. The evolvability assurance for 14 different microservice-based systems was first discussed with 17 software professionals from 10 different firms during our initial interview process. In order to confirm the prevalence of the indicated practises and problems, we next conducted a systematic analysis of the grey literature (GLR) and applied the interview coding method to 295 practitioner online resources. The combined findings indicate that new approaches are needed to address the most important stated difficulties in order to ensure the evolvability of microservices. Individual service quality was not characterised as bad, but the system's architecture served as a focal point for problems.
[7] In this paper, we explore the migrations towards microservices that software engineers and companies go through. In particular, we learn about the various choices made during migrations and how they interact. We learn more about the various decision-making characteristics during migrations from the 16 different migrations toward microservices that we examine in this study. On the one hand, a series of choices are made to demonstrate the viability of a microservices-based design from a technological standpoint. On the other side, choices must be made about what motivates a migration in the first place and how engagement is achieved. This demonstrates how to achieve the necessary "buy-in" to ensure that the organisation as a whole embraces the migration to microservices.

III. OBJECTIVES/SCOPE

An company may use the architectural style known as microservices for creating complicated applications. The use of this architecture makes it possible for the application that is being developed to be loose, independent during deployment, highly maintainable, and tested, as well as firmly organised around business capabilities. The microservice architecture can also be explained fairly simply in another way. It is a method of developing software that enables the organisation of an application. The application under consideration will be organised as a collection of connected services.
The microservice architecture makes it simple to comprehend, create, and test any application. Furthermore, the application will eventually adopt a somewhat rigid stance toward deterioration. This is because of the design concept chosen.
Applications are divided into smaller services, states, etc. as part of the microservice architecture's design to make them more lightweight and fine-grained. As a result, the resulting modularity promotes the adoption of approaches that will enable the division of work. The concurrent design, development, and independent testing of the sub-services will be facilitated by these approaches.
Even though this architectural design is appropriate in some situations, it does have drawbacks like every other architectural design. The research of the architectural platform before adoption is advised by microservice experts. Numerous programming languages might have an impact on how the microservice architecture is implemented.

In the framework of this post, I go over the goal of the microservice architecture as well as look at the benefits and the justifications for learning and using them. I also examine the reasons why specific niches employ microservice architectures and which ones permit their use.
Microservices Architecture, Examples, and Types
The monolithic architectural approach has been significantly decomposed by the microservice architecture. Applications were created using a monolithic architecture, which also allowed the entire development cycle to function as one seamless operation. Contrarily, the microservice architecture functions by facilitating modularization. The division of the main task into smaller services, where each service just reflects a demand, function, or necessity, intensifies and makes possible the requirements that make up the heart of the microservice architecture.
The service scenario for airlines contains examples of microservice designs. If we focus on the part of the process that involves booking flights, we can see that it is divided into the following main tasks: inventory adjustment, seat allocation, fare calculation, timetable lookup, reward management, and customer update.

3.1 Applying Microservices

According to research, the majority of businesses that employ microservices are highly expensive. Organizations like Netflix, eBay, Amazon, PayPal, Twitter, The Guardian, etc. are part of this majority. The United States Government Digital Service is also included in this list.
The use of microservices can be used to a variety of technologies, but according to some studies, the cloud environment and the DevOps environment are also necessary for this architecture to function effectively. These environments and technologies aid in managing the operational difficulties that slow down the architecture's quick functionality.

3.2 Why Do They Use It?

The majority of businesses that employ the microservice architecture have moved beyond the monolithic approach. This is mostly because the strategy enables firms to divide core applications into more manageable components. Then, these components can be individually constructed (which is simpler) and independently maintained. Developed components can converse with one another and cooperate with one another.
The adoption of microservice architecture offers a decentralised method to constructing the software to management of an organisation and everyone participating in its development. This method divides the deployment, rebuild, and redeploy phases of the development process.
Organizations are also given the freedom to use different methods, tools, approaches, etc. for the services involved when choosing a microservice architecture. According to studies, companies who use this architecture are very proficient. This is a result of the architecture's suitability for deployment, replacement, and other independent uses.
The architecture encourages simple problem detection and error debugging. When an issue is detected, it is simple to identify the section where it began, and the mistake is soon discovered. This, as previously said, results from the division of major duties into smaller services. Then, each service can be simplified to a fine state. It's also feasible to divide the service into a smaller collection of services before the basic state is reached. This choice enables the development team to put this architecture's key principle—that a service should only be responsible for one task—into practise. Additionally, the architecture enables the firm to adopt management strategies that are helpful. Options like the change management process, continuous software development procedures, issue and incident management strategies, etc. are included in these management methodologies.

IV. RESEARCH METHODOLOGY

The paper presents the outline of how we understood the technologies used by multiple peoples were we applied a mixed-method approach: first, we conducted a qualitative semi-structured interview study to identify challenges and lessons learned in microservice-based development. Then, utilizing quantitative data gathered from the interviews, we conducted an online survey to further refine and validate (or not) the findings. We synthesized the findings from both the interview and survey studies, forming the final results reported in this paper. We now describe this process in detail, including our selection of subjects and our approach to data collection and analysis, for both the interview study and the follow-up survey.

V. ANALYSIS

We found that practitioners frequently fail to accurately assess their product, process, and cost prior to migrating to Microservices and become aware that critical information is missing only afterwards (during or after the conversion). In addition to assisting in the selection of the most pertinent traits and metrics for migration, our suggested evaluation methodology should also help professionals understand the value of measurement prior to, during, and following the migration to Microservices. Additionally, it has not been made obvious what should be measured before switching to microservices. To close this gap, we propose the assessment approach we have developed. Future study will, however, need to assess and improve the framework in industrial case studies.

VI. FINDINGS

Our analysis identified a clear sense of ownership, strict API management, automated processes, and investment in robust logging and monitoring infrastructure as some of the best practices that contribute to the success of their development processes. They discovered that a variety of linguistic approaches and the recommendation to divide microservices by business functionality are not always successful. Our study found that maintaining shared code between microservices and various product versions are two major issues that practitioners of microservice-based architecture confront. Based on the study, we described different approaches used by practitioners to deal with these problems and noted lessons they had discovered from their experience. We also suggested potential directions the community may go in order to promote effective software engineering techniques in the creation of microservice-based applications.

VII. CONCLUSION

When considering a migration to microservices, businesses should address a select group of traits and parameters that we discovered. This effort yielded an assessment system that will help several businesses decide whether or not they need to move. Their monolithic design can be refactored or their internal organization can be restructured to achieve better results, which will assist them avoid migration if it is not essential.

REFERENCES

[1] From Monolithic Systems to Microservices: An Assessment Framework (Florian Auer - University of Innsbruck, Valentina Lenarduzzi - Austria LUT University, Michael Felderera - Blekinge Institute of Technology, Sweden, Davide Taibid - Tampere University, Finland) - 2021

[2] Microservices: Migration of a Mission Critical System (Nicola Dragoni - Technical University of Denmark and O¨ rebro University, Sweden ndra@dtu.dk, Schahram Dustdary -TU Wien dustdar@dsg.tuwien.ac.at, Stephan T. Larsenz - Danske Bank, Denmark stephantl@gmail.com, Manuel Mazzara - Innopolis University, Russia m.mazzara@innopolis.ru) – 2017

[3] Structural Coupling for Microservices (Sebastiano Panichella - Zurich University of Applied Science (ZHAW), Zurich, Switzerland, Mohammad Imranur Rahman - CLoWEE - Cloud and Web Engineering Group. and Davide Taibi - Tampere University. Tampere. 33720, Finland) – 2019

[4] Monitoring-aware Optimal Deployment for Applications based on Microservices (Edoardo Fadda, Pierluigi Plebani, and Monica Vitali) - 2019

[5] Promises and Challenges of Microservices: an Exploratory Study (Yingying Wang, Harshavardhan Kadiyala, Julia Rubin) - 2020

[6] Industry practices and challenges for the evolvability assurance of microservices (Justus Bogner, Jonas Fritzsch1, Stefan Wagner1, Alfred Zimmermann) - 2021

[7] Facing the Giant: a Grounded Theory Study of Decision-Making in Microservices Migrations (Hamdy Michael Ayas ayas@chalmers.se | CSE Department | Chalmers | University of Gothenburg | Gothenburg, Sweden, Philipp Leitner philipp.leitner@chalmers.se | CSE Department | Chalmers | University of Gothenburg | Gothenburg, Sweden, Regina Hebig hebig@chalmers.se | CSE Department | Chalmers | University of Gothenburg | Gothenburg, Sweden) – 2021