International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 12 | December - 2023 SJIF Rating: 8.176 ISSN: 2582-3930

A COMPREHENSIVE STUDY FOR DEVELOPING A FRAMEWORK FOR MICROSERVICES ARCHITECTURE MIGRATION

Sonal Dhomne[1], Tellaboina Upendar[2] ,Dr. Goldi Soni[3]

[1]*Amity School of Engineering and Technology , Amity University Chhattisgarh , Raipur.*
[2]*Amity School of Engineering and Technology , Amity University Chhattisgarh , Raipur.*

[3] *Amity School of Engineering and Technology , Amity University Chhattisgarh , Raipur.*

Abstract - Building scalable and resilient systems has become a common practice using microservices architecture. Large and complex programs may be split up into manageable components that can be created and deployed separately. Migrating from a monolithic to a microservices architecture is a challenging process that needs careful planning and execution. In this article, we provide a method for migration to a microservices architecture that is based on accepted best practices and industry standards. Three essential phases make up our framework: assessment, design, and execution or Implementation. The assessment stage involves analyzing the existing monolithic application to identify potential microservices candidates. In the design stage, we create a high-level architecture and define microservices boundaries. The implementation stage focuses on building and deploying microservices, along with the necessary infrastructure and tooling. Additionally, we discuss best practices and lessons learned from practical microservices migration projects. Our framework offers a structured approach to organizations seeking to migrate to microservices architecture.

***Key Words*:** Monolithic Architecture, Microservices Architecture, Migration Framework

1.INTRODUCTION

Monolithic applications have dominated the architectural environment for a while. They have intrinsic restrictions even if they are quite simple to design and use. Applications typically have development constraints as they expand, making them harder to scale and manage. By separating programs into smaller, independent components that can be created and deployed individually, a revolutionary method to application development known as microservices architecture seeks to overcome the drawbacks of monolithic design. Microservices design has a number of benefits, including increased scalability, robustness, and agility. But making the switch from a monolithic to a microservices design might be challenging. In this paper, we provide a migration plan to get through this change.

2. BACKGROUND

Microservices architecture is a design pattern that decomposes an application into smaller, independent components, each running in its own process or container. Each microservice handles a specific task or business capability and communicates with other microservices through APIs. Microservices architecture offers benefits such as enhanced scalability, resilience, and agility. It empowers autonomous service development and supports horizontal scaling as needed (Fowler, 2014).

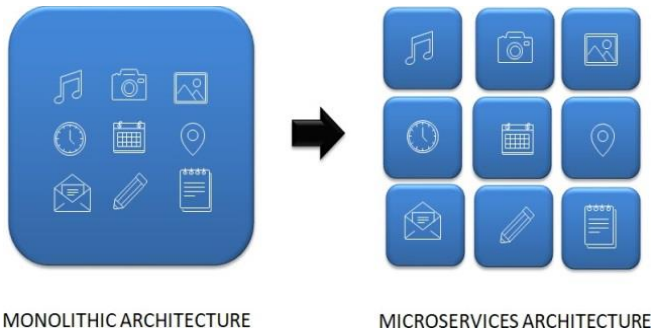

Fig - 1. Monolithic Architecture vs Microservices Architecture

In contrast, monolithic architecture represents an application as a single, standalone component. Although it comprises various modules, they are tightly coupled and share the same codebase. While monolithic applications are straightforward to develop and deploy, they suffer from scalability and maintenance challenges as they grow (Fowler, 2014).

3. PROPOSED FRAMEWORK

Our proposed framework for migrating to a microservices architecture comprises three primary phases: assessment, design, and implementation. These phases are crucial for a successful transition.

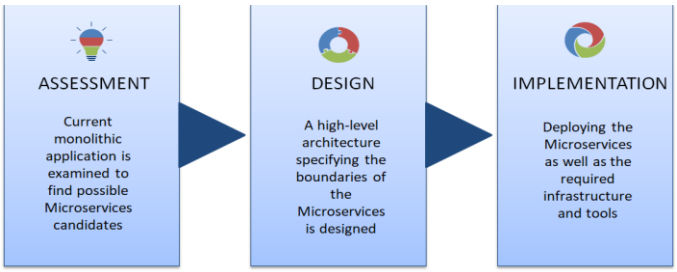

Fig - 2. Migration from Monolith to Microservices phases

3.1 Assessment

In the assessment phase, the existing monolithic application is examined to identify potential microservices candidates. This phase serves as the foundation for the entire migration process. The assessment stage includes the following steps:

Step 1: Determine Business Capabilities - Identify the fundamental business capabilities supported by the application and assess its functionality.

Step 2: Determine the Modules - Examine the codebase to identify modules responsible for each business capability.

Step 3: Assess Module Dependencies - Analyze dependencies between modules to distinguish independent and interdependent modules.

Step 4: Identify Microservices Candidates - Select components suitable for separate microservices based on their independence, minimal dependencies, and distinct business responsibilities.

3.2 Design

The design phase involves creating a high-level architecture and defining microservices boundaries. This phase is critical as it sets the foundation for implementation. The design phase includes the following steps:

Step 1: Define Service Boundaries - Establish boundaries for each microservice, defining the services it offers and the APIs it exposes.

Step 2: Manage Design Data - Develop a data management plan, addressing data sharing and consistency maintenance among microservices.

Step 3: Design Communication Protocols - Determine communication protocols for inter- microservice communication.

Step 4: Determine Infrastructure Needs - Specify the infrastructure needs, such as the platforms for deployment, the containerization tools, and the monitoring programs.

Step 5: Build a high level architecture - The creation of a high-level architecture that incorporates the microservices, their dependencies, and their APIs is the last phase. This design ought to give a clear picture of how the microservices will communicate with one another and deliver the necessary business capabilities.

3.3 Implementation

The implementation phase involves building and deploying microservices, along with the necessary infrastructure and tools. This phase is critical as it paves the way for production deployment. The implementation phase includes the following steps:

Step 1: Build the Microservices - Develop microservices by writing code, conducting testing, and deploying them in a development environment.

Step 2: Containerize the Microservices – Microservices can be containerized by building docker containers and defining their dependencies.

Step 3: Deploy the Microservices –The required infrastructure components needs to be installed, including as load balancers, service discovery, and monitoring tools, before deploying the microservices in the production environment.

Step 4: Monitor and manage the microservices - By putting monitoring and management tools in place to deal with upgrades, failures, and to keep track of their performance and general well-being, Microservices can be improved.

4 OPTIMAL TECHNIQUES AND LESSONS LEARNED

Real-world microservices migration projects have revealed several best practices and lessons:

1. Start with a Pilot Project: Begin with a small pilot project to validate the microservices architecture and identify potential issues (Wooten & Reeves, 2017).

2. Automate Deployment: Implement continuous integration and deployment (CI/CD) pipelines to automate microservices creation and deployment (Balalaie et al., 2016).

3. Ensure Fault Tolerance and Resilience: Incorporate fault tolerance and resilience mechanisms into the microservices architecture to ensure high availability (Balalaie et al., 2016).

4. Use Service Registry and Discovery: Manage inter-service communication and dependencies using a service registry and discovery tool (Pautasso et al., 2016).

5. Implement Security Measures: Establish security measures, including access control, encryption, and authentication, to protect microservices from unauthorized access (Balalaie et al., 2016).

5. CONCLUSIONS

In conclusion, converting to a microservices architecture is a difficult but worthwhile process that may improve the scalability, resilience, and agility of a business. Our methodology offers a methodical strategy for migrating microservices, guided by best practices and insights from actual projects. It allows businesses to take use of this architecture's advantages while minimizing the risks and difficulties of the changeover.

One major benefit of the microservices approach is its capacity to break down large, complex programs into manageable, smaller components. This enables more autonomous development and faster feature rollout, enhancing organizational agility and innovation. However, migrating to microservices requires careful planning and execution. Our framework offers a structured path, guided by best practices and practical insights, to assist organizations on this journey.

REFERENCES

1. 1 Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software, 33(3), 42- 52.
2. Fowler, M. (2014). Microservices: a definition of this new architectural term. Retrieved from https://martinfowler.com/articles/microservices.html
3. Newman, S. (2015). Building microservices: designing fine-grained systems. " O'Reilly Media, Inc.".
4. Pautasso, C., Zimmermann, O., & Leymann, F. (2016). Microservices in practice. In Service Oriented and Cloud Computing (pp. 149-160). Springer, Cham.
5. Richardson, C. (2014). Microservices architecture pattern. Retrieved from https://microservices.io/patterns/microservices.html
6. Vogels, W. (2006). Amazon.com's approach to cloud computing. Communications of the ACM, 51(11), 29-31.
7. Wooten, I., & Reeves, J. (2017). From monolith to microservices: lessons learned on an agile journey. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (pp. 2453-2456).
8. Fowler, M. (2014). Microservices: a definition of this new architectural term. Retrieved from https://martinfowler.com/articles/microservices.html
9. Lewis, J., & Fowler, M. (2014). Microservices: a practitioner's guide. Retrieved from https://www.martinfowler.com/microservices/
10. Newman, S. (2015). Building microservices: designing fine-grained systems. O'Reilly Media, Inc.
11. Wampler, J. (2016). Migrating to microservice databases. Retrieved from https://www.oreilly.com/content/migrating-to-microservice-databases/
12. Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devOps: Migration to a cloud-native architecture. IEEE software, 33(3), 42- 52.
13. Bucchiarone, A., Dragoni, N., Mazzara, M., & Pistore, M. (2018). Microservices: migrating from theory to practice. IEEE software, 35(3), 56-64.
14. Chatterjee, K., Ray, A., & Mandal, T. (2019). A comparative study of monolithic and microservices architecture. Procedia Computer Science, 165, 706-713.
15. Dragoni, N., Mazzara, M., & Pistore, M. (2017). Microservices migration patterns and antipatterns. In Proceedings of the 2nd International Workshop on Software Engineering for Microservices (pp. 1-6).
16. Richardson, C. (2020). Microservices patterns: with examples in Java. Manning Publications Co.
17. Schäfer, T., & Hasselbring, W. (2017). Software engineering for microservices: a systematic mapping study. IEEE Transactions on Software Engineering, 44(11), 1032-1056.