From Monolithic Architecture to Microservices Architecture

Lorenzo De Lauretis
Department of Information Engineering,
Computer Science and Mathematics,
University of L'Aquila, Italy
Email: lorenzo.delauretis@graduate.univaq.it

***Abstract*—The purpose of this work is the definition of a strategy, still in early stage, that will be able to support the migration from a Monolithic Architecture to a Microservices Architecture. This strategy aims to be applied to monolith systems, encouraging their evolution into microservices-based systems. Using this migration strategy, the newborn system will take advantages of a number of benefits offered by microservices architecture, such as scalability and maintainability. Companies will be able to migrate their old monolith systems into more flexible microservices-based systems, evolving their software in a more powerful one.**

I. Introduction

In recent years, there has a tendency in the software engineering community towards cloud computing. Cloud platforms are gaining mainstream adoption as the preferred delivery and operating model for modern applications by several companies like Amazon, Microsoft and IBM [1].

The changing infrastructural circumstances lead to architectural styles that take advantage of the opportunities given by cloud infrastructures. An architectural style that has become more relevant in the last years and that allows taking advantage of the benefits obtained with cloud computing is the Microservices Architecture (MSA) [2].

Microservices can be seen as a technique for developing software applications that, inheriting principles and concepts from the Service-Oriented Architecture (SOA) style, permit to structure a service-based application as a collection of very small loosely coupled software services [3]. Microservices architecture can be seen as a new paradigm for programming applications by means of the composition of small services, each running its own processes and communicating via lightweight mechanisms. Microservices are very small (micro) as for their contribution to the application, not because of their lines of code.

Key features of MSA are bounded contexts, flexibility and modularity. In this respect, microservices are a new trend in software architecture, which emphasises the development and design of highly maintainable and scalable software [3], [4], [5].

Monolithic Architecture (MA), instead, was the traditional approach to software development, used in the past by large companies like Amazon and Ebay. In MA, the functions are encapsulated into a single application. Monolith application, if not complicated, have their own strength, for instance, easiness of development, testing and deployment. However, when the application tends to become more complicated, the monolith structure grows in size, becoming a large, hard to manage and scale piece of software [6].

The problem is that, on the current day, a lot of companies have still their software as a monolith, with all the problems related to MA, better described in III. The majority of those software are difficult to manage and evolve, forcing companies to buy new software instead of developing new functionalities in the owned ones. Migrating their MA to MSA can be a solution, and their software will regain evolvability and all the benefits related to microservices architecture.

In the literature, there are recent works describing migrations from monolith to MSA. A number of these works describes migrations from a high level and architectural point of view, lacking specific instructions on how to execute migrations from a low-level point of view. Those works are described in Section II.

Our work discusses a migration strategy that is still in development, seen from a low-level point of view, describing the details around the transformation. When our strategy will be completely developed, it will allow the migration of a monolithic system into a microservices-based one.

Monolithic architectures are mainly composed of a number of functions; a sequence of calls to functions can be defined as a business functionality (BF) [7]. Our strategy will be centred over BFs.

Our methodology is composed of five phases. It begins with the analysis of the functions of the monolith, passes through the analysis of business functionalities, and ends up with the creation of an MSA.

The paper is structured as follows. Section II, discusses related work. Section III motivates the proposal of this work. Section IV describes the proposed migration strategy, and Section V concludes and discusses future work.

II. Related work

In [8], Gysel et al. describe a service decomposition method on the base of sixteen coupling criteria extracted from literature and industry experience. Clustering algorithms are applied to decompose an undirected, weighted graph, which is transformed from Software System Artifacts.

978-1-7281-5138-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ISSREW.2019.00050

In [9], Richardson describes four decomposition strategies, i.e. “decompose by business capability”, “Decompose by domain-driven design sub domain”, “Decompose by verb or use case” and “Decompose by nouns or resources”.

In [10], Abbott et al. state that the decomposition process can be represented by the Y-axis of the scalable cube: split the application into small chunks to achieve higher scalability.

In [11], the authors describe the migration executed by a software editing company on their core business software, transforming it from a monolith into a Web Oriented Architecture using microservices. The paper presents technical lessons learned during this migration process, mainly focusing on how to determine the most suitable granularity of microservices.

In [2], the authors present a formal microservice extraction model to allow algorithmic recommendation of microservices candidates in a refactoring and migration scenario. Their formal model is implemented in a web-based prototype. They made also a performance evaluation demonstrating that their approach has adequate performances.

In [7], the authors describe a real-world case study from the banking domain, demonstrating how reimplementing a monolithic architecture into microservices improves the system scalability. Their migration process is business-driven; the system is designed one business functionality at a time. Within their strategy, they assign to each microservice one or more business functionalities.

A number of these works describes migrations from an architectural point of view, lacking specific instructions on how to execute migrations. Our strategy, when completely defined, will aim to give specific instructions on how to execute a migration, explaining the obtained benefits. Similarly to [7], and differently from the others, our work rotates around the business functionalities concept, a core element in this kind of migrations [7]. Thanks to our strategy, we will be able to obtain interesting advantages, such as scalability and maintainability.

III. Why migrate a MA to a MSA

Monolithic Architecture, was the traditional way for software development, used in the past by large companies. In MA, the functions are encapsulated in a single application. When the monolith is small and has only a few functions, it can have own strengths, for instance, easiness of development, testing, deployment and scaling [6]. If, for example, we need more computation capability, we simply duplicate the whole monolith, and, with the small size of the system, we have just a small overhead. Weaknesses of this system appear when the monolith starts being updated and evolved, increasing in size and number of functionalities. Once this happens, the disadvantages of MA will outweigh its advantages [6]; for example, if more computation capability is needed, the scaling of a large monolith will result in a bigger overhead with respect to a small monolith. Another disadvantage caused by the large size of the monolith is that the development can be slowed down, and, consequently, become an obstacle to the continuous deployment, because of the longer start-up time [6].

Microservices have a lot of reasons that bring us to prefer an MSA instead of an MA, leading to the migration of the existing monolithic architectures into microservices-based architectures.

A key benefit of microservices is the maintainability. Breaking a system into independent and self-deployable services aids developer teams to test their services and make changes independently from other developers, simplifying distributed development. Thus, the small size of the microservices contributes to increasing code understandability, improving also the maintainability of the code [4], [5].

Another primary key benefit of microservices is scalability. Microservices are not automatically scalable, even if they are commonly deployed in stateless architectures. Nevertheless, in an MSA, each microservice can be deployed on different machines, each one with different levels of performances, and can be written in the more appropriate programming language. If, for example, there is a bottleneck on a specific microservice, it can be containerized and executed across multiple hosts that are in parallel, without the need to deploy the system on a new and powerful machine [4], [5], [12].

Another key benefit of microservices is replaceability. Being individual microservices small in size, the cost to replace them with a better implementation, or even their deletion, is easy to manage. Teams using microservices approaches have small or even no difficulties with completely rewriting microservices when required [4].

An important key benefit of microservices is fault tolerance. The failure of a microservice does not commonly impact the whole system. Nevertheless, faulty microservices can also be quickly restarted [5].

Another key benefit of microservices is reliability. As told in [3], to achieve higher reliability, one must find a way to manage complexities of a large system. Building large systems out of simple and small components with clean interfaces (microservices) is a way to achieve higher reliability.

The reasons for migrating to MSA are manifold, and, at the time of writing, MSA may be the only feasible solution for reducing the complexity of the systems [5].

IV. Migration Strategy

In this section we propose a migration strategy, that is still in development, that, when completed, will be used to support the migration from an MA to an MSA.

In MA, a monolith has a certain number of functions, that are used to make actions. Functions can be invoked in ordered and finite sequences; these sequences of calls to functions are called business functionalities. A monolith can have a number of business functionalities, depending on its characteristics and how it is designed. As said in [7], business functionalities have an important role in this kind of migrations.

The strategy that we propose rotates around the business functionalities concept. At the current stage of research, it is composed of five phases, as illustrated in Figure 1.

1) Function analysis
2) Business functionalities identification

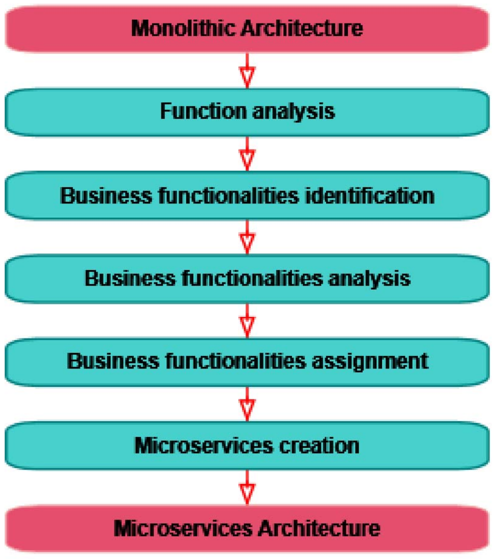
Monolithic Architecture
Function analysis
Business functionalities identification
Business functionalities analysis
Business functionalities assignment
Microservices creation
Microservices Architecture

Fig. 1. Our strategy to split an MA into an MSA

3) Business functionalities analysis
4) Business functionalities assignment
5) Microservices creation

Our strategy, being currently in development, cannot be tested on the current day. Before it can be adopted and recognized as a working strategy, it should pass a strong testing phase, that will comprehend formal demonstrations and empiric tests. Empiric tests will be made on several scenarios, starting from an MA and having an MSA as a result of the migration, verifying and inspecting closely all the phases of the migration. If the strategy will have leaks or missing procedures, it will be studied more or re-engineered if needed. When the strategy will have no more leaks or missing procedures, we will consider it as functionally complete.

When our strategy will be complete, in the first place, all the five phases will be executed manually, without automation, with the possible need of rewriting the legacy software code. In future, when the strategy will be explored more in details, we will put effort into finding algorithms that automatize the migration process, with the aim to create a standard automatic migration procedure, without the need of rewriting the legacy software code.

A. *Function analysis*

In the first part of our strategy, our intention is to extract and analyze all the functions that compose the monolith. Functions are analyzed to extract data such as rate of usage and size (in terms of lines of code). Then, the functions can be also modified: if a function is large, it can be split in a number of subfunctions; if two or more functions are very similar, they can be merged in a single one, and so on.

B. *Business functionalities identification*

In this part of our strategy, business functionalities are identified. Each monolith can have a certain number of business functionalities, depending on its characteristics and how it is designed. The identification of all the business functionalities in a monolith is not an easy task. The easiest way to identify them is to ask directly to the monolith owner or to the monolith developers, but this is not always possible. In a more complex way, they can be identified using a machine learning process. The work described in [13] shows how it is possible to obtain the business functionalities from a service through a machine learning process; with appropriate modifications, this process can be applied to MA.

C. *Business functionalities analysis*

Business functionalities identified in the previous step are analyzed, in order to extract data such as rate of usage and other statistical information. To accomplish this, standard usage of the whole system is simulated through machine learning or human interaction, obtaining statistical usage data. Those data are afterwards analyzed. Thanks to these data we obtained, we are able to accomplish the next step.

D. *Business functionalities assignment*

In this phase, business functionalities are theoretically assigned to microservices. Thanks to the statistical information retrieved on the previous phase, we are able to improve the resulting microservices quality, size and granularity. Following our strategy, if a business functionality is used a lot with respect to the others, it is assigned to a single microservice; if instead, two or more functionalities are used more or less the same number of times, and their scope is similar, they are all inserted into a single microservice. The assignment of the business functionalities to microservices is a complex task, and this section describes only a possible strategy of assignment. We are still working on a strategy that will optimize the number of microservices and their granularity, trying to develop an algorithm that automates the whole process.

E. *Microservices creation*

After assigning theoretically business functionalities to microservices, in this phase we are doing it in a more practical way. Microservices are created, and the functions related to business functionalities are developed inside microservices, following the assignment shown in the previous phase. Now, the MSA is ready to be deployed and tested.

V. Conclusions

The work described in this paper describes a methodology, still in progress, to migrate an MA to an MSA, focusing on the business capabilities concept. Thanks to our strategy, the newborn system can have most of the benefits related to MSA, such as improved scalability, maintainability and evolvability [4], [5]. Companies that need to evolve their systems or to add new functionalities to them, will be able to migrate their existing systems and implement new functionalities on that, instead of developing a whole new system, saving money.

The strategy described in this paper is still in a development phase since it has still things to be defined, such as the business functionalities assignment algorithm, that will automate the whole process of business functionalities assignment. This algorithm will try to optimize the number of microservices and their granularity; it will be developed in future works.

In the near future, we will improve our strategy, defining new rules and defining better the five steps described in Section IV. In a far future, we will test the complete strategy on a real monolith, reporting the data we will obtain for statistical and performance metrics.

References

[1] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The making of cloud applications: An empirical study on software development for the cloud," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 393–403.

[2] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and ulterior software engineering*. Springer, 2017, pp. 195–216.

[4] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[5] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[6] R. Chen, S. Li, and Z. Li, "From monolith to microservices: a dataflow-driven approach," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 466–475.

[7] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From monolithic to microservices: an experience report from the banking domain," *Ieee Software*, vol. 35, no. 3, pp. 50–55, 2018.

[8] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2016, pp. 185–200.

[9] C. Richardson, "Pattern: microservice architecture," *URL: http://microservices. io/patterns/microservices. html*, 2017.

[10] M. L. Abbott and M. T. Fisher, *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

[11] J.-P. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 62–65.

[12] M. Autili, A. Perucci, and L. D. Lauretis, "A hybrid approach to microservices load balancing," in *Microservices: Science and Engineering*. Springer - in press, 2019, pp. 1–21.

[13] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 141–150. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595719