Collection and Identification Of Microservices Patterns And Antipatterns

Rafik Tighilt[1], Manel Abdellatif[1,2], Nader Abu Saad[3], Naouel Moha[1], and Yann-Gaël Guéhéneuc[4]

[1] Université du Québec à Montréal, Montréal, Québec, Canada
tighilt.rafik@courrier.uqam.ca, naouel.moha@uqam.ca
[2] Polytechnique Montréal, Montréal, Québec, Canada.
manel.abdellatif@polymtl.ca
[3] Bethlehem University, Bethlehem, Palestine
nabusaad@bethlehem.edu
[4] Concordia University, Montréal, Québec, Canada
yann-gael.gueheneuc@concordia.ca

Abstract

Microservices architectural style is becoming more and more popular in academia and industry. However, a lack of understanding of its core concepts and the absence of ground-truth leads to a lot of misconceptions and development mistakes. In our research work, we aim to clarify the academic knowledge on microservices through the collection and the automatic identification of microservices patterns and anti-patterns. To this end, we aim to (1) introduce an exhaustive collection of microservices (anti-)patterns, and (2) propose an automatic approach for the identification of (anti-)patterns in microservice-based systems. The continuous integration and continuous delivery for microservices can introduce *anti-patterns* that may affect the maintainability of the system and decrease its quality. Thus, we searched for re-engineering tools used to identify (anti-)patterns in microservice-based systems. The results of our analysis showed that there is no fully-automated identification approach in the literature. This motivates us to propose (anti-)patterns for the identification process as a first step and then investigate how we can automatically identify them from the artifacts of microservice-based systems.

1 Introduction

A microservice is defined as a small service, having a single responsibility, running on its own process and communicating with lightweight mechanisms [5]. Microservices are generally built around business capabilities and independently deployable by a fully automated deployment machinery with a minimum centralized management of these services [8]. An illustration of a microservice-based system could be an e-commerce application. Each microservice of this system fulfils a single business function (Inventory microservice, Shipping microservice, Cart microservice, etc.), and has its own database. These microservices are loosey coupled, and communicate through lightweight REST APIs.

The popularity of microservice architectures has grown in the last few years due to the inability of monolithic applications to handle applications scalability and development cycles [8]. Also, this architectural style has been adopted by several major actors in industry such as Netflix, Amazon and Riot Games, which leads to its increasing popularity.

The dynamic and distributed nature of microservice-based systems make them a good solution to offer greater agility and operational efficiency for enterprise architectures. However, the continuous integration and continuous delivery for microservices can introduce common bad practiced solutions i.e., *anti-patterns* - opposed to *design patterns*, that are good solutions

to recurring problems. These anti-patterns may affect the maintainability of the system and degrade the quality of design and QoS of the system [11]. *Shared database* is a good illustration of microservices anti-patterns. This anti-pattern goes against the principle of loosely coupled microservices [1]. It also prevents each microservice to use the right type of database that suits its needs (relational, graph, document, etc.). It also leads to making the whole system go down if some parts of the persistence are down. On the other hand, *External configuration* [4], which consists in externalizing all the configurations, including endpoint URLs and credentials, which can be loaded on microservice startup or on the fly, is a good practice to allow changes in configuration properties without rebuilding and redeploying the entire microservice.

Context: Assessment on design, maintenance and evolution of microservice-based systems. *Anti-patterns* are "bad" solutions to recurring problems. These solutions often lead to harder maintainability of software systems. Like any other architectural style, microservice-based systems also face challenges with maintainability and evolution due to anti-patterns [15]. Thus, changing and implementing functionalities becomes more and more difficult.

Problem: No automated approach to identify (anti-)patterns in microservice-based systems. The nature of microservice-based systems makes them very dynamic (multi-language systems, different operating environments, etc.). This makes the identification of (anti-)patterns in this environment an interesting problem, especially because there is a lack of automated tools in the literature to identify (anti-)patterns in microservice-based systems.

Motivation: Software maintenance is one of the most requiring fields in software industry, whether in expanses or in resources [3]. The identification of (anti-)patterns within a microservice-based system is also a part of the maintenance process, and it leads to a better assessment of the quality of software. In summary, with our automatic tool-based approach, we aim to contribute to better maintenance and quality of microservice-based systems.

Challenges: Microservices are, by definition, independent. With that in mind, we will need to overcome some challenges during our work process. In fact, microservice-based systems can be built using different programming languages, which makes the identification process more challenging. Microservices are also deployed on multiple providers using different tools and configurations. Also, for our empirical study, we will need to find as much open-source microservice-based systems as we can to analyze them and validate our automatic approach.

Contributions: In our work, we want to (1) introduce an exhaustive collection of microservices (anti-)patterns, and (2) propose an automatic tool-based approach for the identification of (anti-)patterns in microservice-based systems.
Our approach is based on software re-engineering techniques to extract from the application useful information for pattern identification.

The remainder of this paper is structured as follows. In Section 2 we describe our research objectives. In Section 3 we outline our methodology for (anti-)patterns identification from microservice-based systems. In Section 4 we describe some related works to microservices (anti-)patterns, as well as (anti-)patterns identification techniques. Finally, Section 5 concludes this paper.

2 Research objectives

In this section we will describe our research objectives and the related research questions.

2.1 Collecting microservices (anti-)patterns

The first goal of our study is to define a catalog of (anti-)patterns of microservice-based systems. Indeed, a lot of these (anti-)patterns are scattered in the literature, but it is still very difficult to synthesise and fully understand them when working on a real-world project. For this purpose, we will try to answer the following research question.

RQ 1: What are the (anti-)patterns for developing microservice-based systems ?

Nonetheless we will also study real-world applications to see whether these best practices are applicable to large scale projects or to industry driven systems. This will allow us to study the gap between academia and industry regarding the way microservice-based systems are built.

2.2 Automatic (anti-)patterns identification

The second goal of our work is to identify (anti-)patterns directly from the analysis of microservice-based systems artifacts (source code, deployment scripts, etc.). To this end, we will answer the following research question.

RQ 2: How can we automatically identify (anti-)patterns by analysing microservice-based systems artifacts ?

3 Methodology

Our research process, described in Figure 1 is divided in two steps, each step leading to the answer of one research question. The following subsections details each step and their validation.

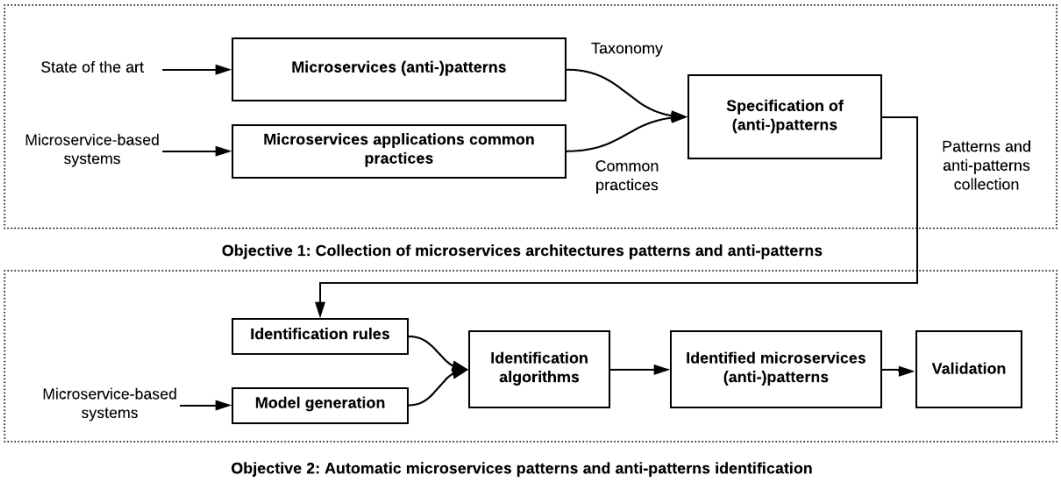

Figure 1: Research methodology

3.1 Collection of microservices (anti-)patterns

For the first step of our work, we will study the literature to extract microservices (anti-)patterns. We will then map the same concepts described with different names into our own taxonomy.

We can take as an example the *Shared database* anti-pattern, for which the corresponding pattern is described in the literature by multiple definitions, as listed in Table 1.

Name	Definition	Source
Decentralised Data Management and Governance	Data is decentralised and distributed between the constituent microservices.	[13]
Database per service	Each microservice manages its own data.	[12]
Isolated Data & State	Each service owns its data and its state.	[7]

Table 1: Shared Database corresponding pattern in the literature

We will also take into account in our work the industrial literature, in order to identify the possible gap between academia and industry in the field of microservices (anti-)patterns. This will allow us to understand the problems occurring when developing microservices in a day-to-day basis, and to identify the solutions implemented by the industry and the emerging patterns we can extract.

In addition to the literature study, we will examine real-world microservice-based systems[1], by analyzing these systems artifacts, such as source code, documentation and deployment scripts to extract common practices used by the industry when building microservices.
We aim to group the (anti-)patterns extracted from the literature and common industry practices into a textual specification of (anti-)patterns. We can then extract a set of automated rules to identify (anti-)patterns with our automatic tool-based approach.

3.2 Automatic (anti-)patterns identification

The second step of our work is to automatically identify (anti-)patterns in microservice-based systems. We aim to do so by specifying a meta-model for microservice-based systems. This meta-model will describe the structure of a microservice-based system, its main components (e.g., list of microservices, deployment scripts, environment variables, etc.) as well as its main dependencies (e.g., databases, configuration files, calls between microservices, etc.). This meta-model will serve as a roadmap for our detection rule engine that will embed the different detection rules of our targeted microservices (anti-)patterns. These detection rules will rely on several metrics such as the number of microservices in the system, the number of endpoints per microservice, the number of microservices per databases, fan in / fan out, etc.

If we take the example of *Shared database*, we can identify this anti-pattern through the analysis of the generated meta-model of the microservice-based system. We focus on the description of the deployment scripts and the source code of the microservices. From these artifacts, we calculate the number of microservices by database. This metric will allow us to identify the shared databases between the microservices.

On the other hand, if we want to identify violations of the *External configuration* pattern, we can analyse the source code to find hard coded configuration-related information such as URLs, Database credentials, configuration constants, as well as the usage of environment variables extracted from our meta-model.

[1] https://github.com/davidetaibi/Microservices_Project_List

3.3 Validation

We will validate the results of our identification process by performing an empirical study on real-world microservice-based systems to obtain outputs from our tool, which we will compare with the output gathered through a manual analysis of a subset of the analysed systems. We already identified microservice-based systems on which we can perform our validation, such as Service Commerce[2] or LakeSide[3].

4 Related work

Several work has been done regarding (anti-)patterns definition and identification for microservice-based systems. For the understanding of the microservice architecture and patterns definition, Pahl and Jamshdi [10] conducted a systematic mapping study on 21 selected works that were published from 2014 to 2016. They classified and compared the selected studies based on a characterisation framework they defined in their work, resulting in a discussion of the agreed and emerging concerns within the microservices architectural style.
Zimmerman [16] proposed to map microservices tenets to service oriented architecture patterns to conclude that microservices constitute one special implementation approach to service oriented architecture. Garriga [2] in his work, proposed a taxonomy of concepts regarding microservices, which includes the whole microservices life-cycle (design, implementation, deployment, runtime and cross-cutting concerns), as well as organizational aspects.
Soldani et al. [14] conducted a study on the industrial *grey* literature. They did so by identifying, taxonomically classifying, and systematically comparing the existing grey literature on pains and gains of microservices, from their design to their development to the state of the art, in order to fill the gap between the academia and the day-to-day industry practices.

For the real-world software study, Marquez and Astudillo[6] extended their previous work with Osses [9] determining whether architectural patterns have been used in the development of microservice-based systems by providing (i) a catalog of microservices architectural patterns reported in academia and industry, (ii) a correlation between quality attributes and microservices architectural patterns, (iii) a list of technologies, deployed as frameworks, used in the construction of microservices-based systems with microservices architectural patterns, and (iv) a comparative analysis of service oriented architecture and microservices architectural patterns. They did so by analysing a set of microservices based open-source projects on GitHub.

Through our literature study, we conclude that very few work has been done in the field of automatic identification of (anti-)patterns in microservice-based systems. Thus, we identified a gap that we aim to fill through our research process.

5 Conclusion

Microservcies are becoming a very popular architectural style, supported by a lot of companies (Netflix, Amazon, Riot Games, etc.). Thus, understanding this architecture, its (anti-)patterns, and the ecosystems around is fundamental. A lot of work has been done in that field, but there is a little that groups common practices together and synthesise the knowledge acquired.
On the other hand, when it comes to (anti-)patterns identification in microservice-based systems, very few work has been done to propose an automatic tool-based approach to analyse

[2] https://github.com/antonio94js/servicecommerce
[3] https://github.com/Microservice-API-Patterns/LakesideMutual

software artifacts and define if a given system respects or breaks a set of microservices patterns. In this paper we presented our plans for creating a collection of (anti-)patterns regarding microservice-based systems, by performing a systematic literature and state of the practice review. We also described our automatic tool-based approach for the identification of (anti-)patterns through the analysis of microservice-based software artifacts. We are still at the beginning of our research work. We will now propose a complete collection of microservices (anti-)patterns and an automatic identification technique to help practitioners and researchers respect patterns when developing microservice-based systems.

References

[1] Brogi, A., Neri, D., Soldani, J., and Zimmermann, O. Design principles, architectural smells and refactorings for microservices: A multivocal review.

[2] Garriga, M. Towards a taxonomy of microservices architectures. In *Software Engineering and Formal Methods*. Springer International Publishing, 2018, pp. 203–218.

[3] Hanna, M. Maintenance burden begging for remedy. *SOFTWARE MAGAZINE-WESTBOROUGH- 13* (1993), 53–53.

[4] Humble, J., and Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.

[5] Lewis, J., and Fowler, M. Microservices a definition of this new architectural term. https://martinfowler.com/articles/microservices.html. Accessed: August 2019.

[6] Marquez, G., and Astudillo, H. Actual use of architectural patterns in microservices-based open source projects. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)* (dec 2018), IEEE.

[7] Microsoft. Microservices: Isolated data and state. https://blogs.msdn.microsoft.com/azuredev/2018/04/11/microservices-isolated-data-state/. Accessed: August 2019.

[8] Newman, S. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc., 2015.

[9] Osses, F., Marquez, G., and Astudillo, H. Exploration of academic and industrial evidence about architectural tactics and patterns in microservices. In *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings - ICSE* (2018), ACM Press.

[10] Pahl, C., and Jamshidi, P. Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science* (2016), SCITEPRESS - Science and and Technology Publications.

[11] Palma, F. Detection of soa antipatterns. In *Service-Oriented Computing - ICSOC 2012 Workshops* (Berlin, Heidelberg, 2013), A. Ghose, H. Zhu, Q. Yu, A. Delis, Q. Z. Sheng, O. Perrin, J. Wang, and Y. Wang, Eds., Springer Berlin Heidelberg, pp. 412–418.

[12] Richardson, C. Microservicespattern: Database per service. http://microservices.io/patterns/data/database-per-service.html. Accessed: August 2019.

[13] Shadija, D., Rezai, M., and Hill, R. Towards an understanding of microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)* (Sep. 2017), pp. 1–6.

[14] Soldani, J., Tamburri, D. A., and Heuvel, W.-J. V. D. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software 146* (dec 2018), 215–232.

[15] Taibi, D., and Lenarduzzi, V. On the definition of microservice bad smells. *IEEE Software 35*, 3 (May 2018), 56–62.

[16] Zimmermann, O. Microservices tenets. *Computer Science - Research and Development 32*, 3-4 (2016), 301–310.