Analysis of Legacy Monolithic Software Decomposition into Microservices

Justas Kazanavičius[1], Dalius Mažeika[1]

[1]Vilnius Gediminas Technical University, Vilnius, Lithuania
justas.kazanavicius@vgtu.lt, Dalius.Mazeika@vgtu.lt

Abstract— Microservice architecture is becoming a standard by default in most of the enterprises because many projects have been implemented using this architecture in the last few years and results have been very positive. Extracting microservices from legacy monolithic software is an extremely complicated task. Each enterprise application is unique. This paper aims to investigate the existing methodologies of monolith decomposition into microservices. The same enterprise application was decomposed into microservices using 3 different methods. Evaluation criteria were proposed that were used to analyze advantages and disadvantages of each.

Keywords—microservice, monolith decomposition, cloud computing, software engineering

1 Introduction

Microservices are entering mainstream, according to NGINX research only 30% of companies not using them at all. NGINX researched 1800 IT professionals and results show that nearly 70% of organizations are using or investigating a microservices architecture, while nearly 1/3 are already using microservices architecture in production. [1]. One of the biggest microservices advantages is that it is a cloud-native application [2, 3]. Because microservices are independent processes each of them could be deployed to a separate container or virtual machine in the cloud.

Migration to microservices from monolithic legacy software cannot be implemented in a fast way. It is important to know that there is a high overall cost associated with decomposing an existing system to microservices [4, 5]. It is not possible to say that only one good way exists to migrate from monolith to microservices because legacy monolith application is a very broad term and can vary in many aspects such as programming languages, database technologies, team size and so on. [6, 7, 8]. Different organizations use different migration patterns, techniques, and methods because microservices are still a relatively new architectural approach and widely approved way of doing it not exist [9, 10]. A key challenge in migration process is the extraction of microservice candidates from existing legacy monolithic code bases [11].

This article's main goal is to analyze existing legacy monolith application decomposition into microservices architecture based application methodologies. During literature review and analysis three main directions how decomposition from monoliths to microservices could be realized were identified [15]: Storage based methods [12], code based methods [13, 17] and business domain based methods [2, 14, 16]. Three methods [12, 13, 14] were chosen to be analyzed because each of them best represents a separate direction of how decomposition from monoliths to microservices could be implemented. Other methods found during literature review and analysis use the same directions or combining them to achieve better results [2, 15].

A comparison between selected methodologies was done by decomposing the same enterprise legacy monolith application, named DataProvider, into microservices three times, using all selected methodologies. Benefits and drawbacks of each methodology were analyzed and compared.

2 Enterprise Monolithic Application Architecture

The primary function of the DataProvider application is to provide important organization data from one place to others information systems in an organization. Different types of data like accounts, books, customers and so on are stored in different mainframe systems within an organization.

The DataProvider application (Figure 1) consist of three main components:

- Business logic – collecting and caching data from old mainframe systems. Business logic writes collected data to the DataProvider local database. Entity framework is used to communicate application and database.
- Database – MS SQL database technology is used to store collected data from mainframe systems.
- Rest API – HTML endpoint for other information systems to access important organization data in DataProvider. Swagger tools are used to provide Rest API functionality.

Application is written with Microsoft .NET framework and C# programing language is used.

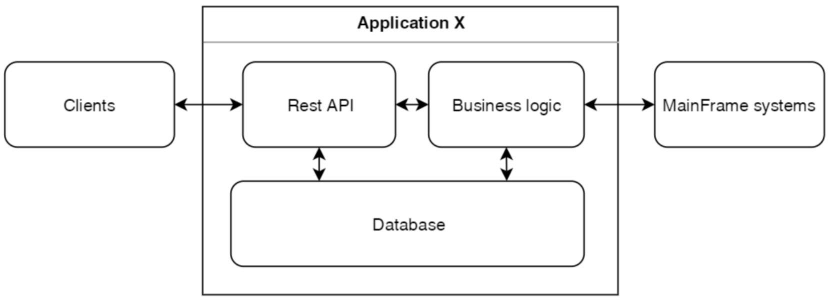

Fig. 1. The DataProvider application architecture

3 Storage-Based Method Evaluation

Alessandra Levcovitz, Ricardo Terra and Marco Tulio Valente describe a technique to identify microservices on monolithic systems [12]. The proposed technique consists of the following four steps.

Database decomposition.
The first step is mapping database tables into subsystems. Each subsystem represents a business area of an organization. DataProvider application has 15 database tables, 9 subsystems, and 8 different business areas. This step of methodology allows identifying a number of tables and business areas. Identify database tables is a task that requires only technical skills. On the other hand, identify business subsystems and assign a table to them require additional effort to understand the business process.

Dependency Graph.
In the second step dependency graph between facades, business functions and database tables were created. It shows business functionality and database dependencies. 5 graphs were pretty straightforward: containing only 1 database table, 1 business functionality layer and 0 dependencies from other database tables and business functionality subsystems. The other 12 database tables were joined into one more complex and complicated dependency graph. Some business functionality contains up to 4 dependencies from other database tables. Mostly 4 business functionality layers identified for full operation from facade to the database table to be accomplished.

Database tables and facades pairs.
Based on the dependency graph unique pairs of facades and database tables were identified and mapped with business subsystems functions. DataProvider application has 68 unique pairs of database tables and facades. 15 facades were in pairs with only 1 database table. Some of the facades were in pair with different database table up to 8 times. More complicated dependency graph more unique pairs exist.

Microservice candidates.
In the last step candidates to be transformed into microservices were identified. For each distinct pair obtained on the prior step, inspect the code of the facade and business functions that are on the path from the facade to the database table in the dependency graph.

37 microservices candidates were found in the DataProvider application after decomposition was done using method I. More functions and tables subsystem had more microservice candidates were identified. It is a possibility that the microservice candidate size could be very small if it contains only one business function. The method requires identifying business subsystems, to do that business knowledge is needed which is why the method couldn't be implemented completely automated.

4 Code-Based Method Evaluation

Genc Mazlami, Jurgen Cito and Philipp Leitner created microservices extraction from the monolithic systems model [13]. There are two transformations between the stages:

Construction step.
The first step is the monolith transformation into the graph representation. In the graph, each vertex represents class from the monolith and undirected edges represent its coupling with other classes in the monolith.

DataProvider application has 273 classes at all. The biggest number of dependencies which one class has is 96. 17 classes have 0 dependencies and are not part of a graph. The average classes coupling is ~10. Unit, integration and manual tests classes were excluded from the graph. More quality code has less coupled classes are, so lower number of edges in the graph indicates a higher quality of code. It is not clear how to treat class inheritance from the article, in this evaluation decision was made to treat class inheritance as not dependency.

Clustering step.
The second and final transformation is to cut the graph in components that are going to represent recommended microservices candidates. The authors proposed three different strategies of how to do it: logical coupling, semantic coupling and contributor coupling. During this comparison, semantic coupling was chosen in evaluation.

8 microservices candidates were found in the DataProvider application. 180 classes were identified for a specific business domain by class name. It was not possible to identify the business domain by class name for 93 classes. ~33% of classes need additional effort to review and assign manually to the specific business domain or refactor and split into more classes. Code quality playing a vital role in how easily a microservice candidate could be identified in the graph. If code is written following clean code standards class should only have one responsibility, few dependencies, and meaningful name. Automation could be used in extraction accurately only if the monolith code has high quality. If a class has a lot of dependencies, not a meaningful name or has too many responsibilities it is not clear to which microservice candidate it belongs. In this case, the additional effort needed to refactor class. Code-Based Method is very formal and requires additional tools to be implemented properly. These tools are note available only algorithm and a mathematical model is provided, so organizations should implement them itself.

5 Business-Domain-Based Method Evaluation

Chen-Yuan Fan and Shang-Pin proposed a migration process based on SDLC, including all of the methods and tools required during design, development, and implementation [14]. Two analysis methods are used in the migration of a legacy monolithic architecture into a microservice architecture.

Domain-Driven Design analysis.
In first step Domain-Driven Design (DDD) was used to find microservice candidates in the original system. The bounded context analysis results are a key tool to identify microservice candidates in DataProvider application. The DDD was used to identify specific domains in solution and identify domain modules in each domain. DDD approach analysis allows extracting of low-coupling microservices.

Database analysis.
The second step involves the analysis of the database structure. It is common practice that each microservice should use a discrete database. This allows to avoid high coupling between services. On the other hand, splitting some data into separate databases could cause data inconsistency. Foreign keys could be used as an indication of the microservice candidate.

New architecture.
After Domain-Driven Design and database analysis, 8 microservices candidates were found in the DataProvider application. 1 additional microservice should be created. To connect all microservices into one solution new microservice was introduced. Sync Service provides data synchronization and an interface for front-end systems. The most important things for a successful migration from monolith to microservices using Business-Domain-Based Method is Strong business knowledge, business process stability in the organization and high-quality database schema.

6 Extraction Methods Results Comparison

This section compares the extraction methods evaluations results in different aspects. Detailed results about evaluations present in Table I.

Table 1. Evaluation results

	Storage-Based Method		Code-Based Method		Business-Domain-Based Method	
Business	Tables/ Functions	Microservice candidates	Classes	Microservice candidates	Tables	Microservice candidates
Accounting	1/2	2	13	1	1	1
Booking	1/5	5	13	1	1	1
Departments	1/3	3	14	1	1	1
Customers	5/9	15	63	1	5	1
Ratings	1/4	4	13	1	1	1
Users	3/3	3	38	1	3	1
Countries	1/2	2	13	1	1	1
Currencies	1/2	3	13	1	1	1

Microservice candidates count.

Extraction Storage-Based Method found most microservice candidates in the DataProvider application. The Storage-Based Method found 37 candidates, Code-Based Method found 8 candidates and Business-Domain-Based Method found 8 candidates also. In Storage-Based Method microservice is extracting as a concrete function in the application while in Business-Domain-Based Method microservice represents the specific business domain. It is obvious that Storage-Based Method will always provide more microservices than Business-Domain-Based Method because the business domain always has at least one function. Code-Based Method is more flexible than other compared methods, it provides optionality to choose the strategy on how microservice should be extracted. Semantic coupling strategy was chosen during this comparison, its key idea, in general, is very similar to Domain-Driven Design so what explains why it found the same number of microservice candidates as Business-Domain-Based Method. Another extraction strategy is logical coupling which focuses on concrete functions. It could be predicted that microservice candidates would be found similar to Method I.

Size of microservice.
The main idea of microservice is that it should have only one responsibility. Responsibility could be business or functional type. Business responsibility is bigger than functional because it contains at least one function and usually it contains much more than one. Split by functions microservices is much smaller and has been named as serverless. If organizations decide to split they monolith application into microservices by business domains when they should choose Business-Domain-Based Method or Code-Based Method with semantic coupling strategy. If the decision is splitting into microservices by functions when Storage-Based Method or Code-Based Method with logical coupling strategy could be used.

Databases.
The most common and popular practice is that each microservice should use its private database. Business-Domain-Based Method perfectly fits this approach. Storage-Based Method splits monolith into microservices by functions and some functions most likely going to use the same table. If the decision was made to use this method database probably going to be shared. Methods authors do not provide any recommendations on how to deal with this challenge. Code-Based Method authors assume that monolith application use repository pattern and each table is represented as a repository class in solution. Methods don’t contain any recommendation of how databases should be adapted to microservice architecture.

Automation.
Code-Based Method with the contributor coupling strategy could be implemented fully automatically. Monolith must be implemented with an object-oriented programming language because the extraction model is based on classes as the atomic unit of computation and the graph. Code-Based Method with a semantic coupling strategy could be implemented semi-automatically. In this case, business domains should be identified manually. How accurate the method will be able to identify class

relation to business domain depends on a naming convention in code. Storage-Based Method and Business-Domain-Based Method can't be implemented automatically. Storage-Based Method requires manually identify business subsystems and assign database tables to one of the subsystems. Business-Domain-Based Method requires two manual analysis to do.

Technological stack.
Storage-Based Method designed to work with backend type applications. It is programming language agnostic. Database storing data in tables must be part of the application, because extraction use tables to generated graph. Code-Based Method is suitable for backend type applications written with an object-oriented programming language. The extraction model is based on classes as the atomic unit. If the application is written with another type of language or with a few different languages when it is not possible to use Code-Based Method for microservices extraction. Business-Domain-Based Method is technologically agnostic and could use with any kind of programming languages and databases.

Implementation and tools.
Business-Domain-Based Method is in the least formal and most universal comparing with other compared methods. On the other hand, it is most uncertain and requires the implementer to have a strong knowledge of application business domain and implementation technical details. Code-Based Method is the most formal and requires an additional tool to generate graph representing dependencies of classes. It is not clear what would be cheaper in time and resources: implement the tool and use it or use other methods to do microservices extraction from the monolith. Storage-Based Method do not require any additional tool to implement, but it requires some business domain knowledge to identify business subsystems.

Code quality.
Code quality has the most impact on Code-Based Method because it creates classes' dependencies graph. Clean and solid code generates a more accurate graph. The more accurate graph allows extracting more accurate microservices. Code quality also has an impact on Storage-Based Method and Business-Domain-Based Method as well. The better code quality is, the easier it is to extract functions from it.

7 Conclusions

Choosing the right microservices extraction from the legacy monolith application method is a very hard task and crucial for a successful migration. Each legacy monolithic application is unique and creates unique challenges. One best methodology on how to extract microservices from monolith does not exist. Each case is different and the organization should choose which method or combination of methods best suits for its migration from monolith to microservices. Selected methodology or combination of methodologies should be: able to extract microservices by selected factors and compatible with technological stack and database technologies used in application.

References

1. NGINX, "The Future of Application Development and Delivery Is Now" [Online]. Available: https://www.nginx.com/resources/library/app-dev-survey/
2. O. Pozdniakova and D. Mažeika, "Systematic Literature Review of the Cloud-ready Software Architecture." Baltic J. Modern Computing, vol. 5, pp .124–135, March 2017
3. O. Pozdniakova and D. Mažeika, "A cloud software isolation and crossplatform portability methods." 2017 Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, 2017, pp. 1–6.
4. M. Fowler and J. Lewis, "Microservices." [Online]. Available: http://martinfowler.com/articles/microservices.html
5. Z. Dehghani, "How to break a Monolith into Microservices." [Online]. Available: https://martinfowler.com/articles/break-monolith-intomicroservices.html
6. D. Linthicum, "From containers to microservices: Modernizing legacy applications." [Online]. Available: https://techbeacon.com/enterpriseit/containers-microservices-modernizing-legacy-applications
7. N. Vennaro, "How to introduce microservices in a legacy environment." [Online]. Available: https://www.infoworld.com/article/3237175/howto-introduce-microservices-in-a-legacy-environment.html
8. S. Koltovich, "How to Modernize Legacy Applications for a Microservices-Based Deployment." [Online]. Available: https://thenewstack.io/modernize-legacy-applications-keepupdate-re-write-needs-re-written/
9. A. Furda, C. Fidge, O. Zimmermann, W. Kelly and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency," in IEEE Software, vol. 35, no. 3, pp. 63–72, May/June 2018.
10. M. Mishra, S. Kunde and M. Nambiar, "Cracking the Monolith: Challenges in Data Transitioning to Cloud Native Architectures." ECSA '18 Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, Madrid, 2018, pp. 1–4.
11. A. Carrasco, B. V. Bladel and S. Demeyer, "Migrating towards Microservices: Migration and Architecture Smells." In Proceedings of the 2nd International Workshop on Refactoring (IwoR '18), September 4,2018, Montpellier, France. ACM, New York, NY, USA, 6 pages. https://doi.org/0.1145/3242163.3242164
12. A. Levcovitz, R. Terra, M. T. Valente, "Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems." 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM), p. 97–104, 2015
13. G. Mazlami, J. Cito and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, 2017, pp. 524–531. Sdf
14. C. Fan and S. Ma, "Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report," 2017 IEEE International Conference on AI & Mobile Services (AIMS), Honolulu, HI, 2017, pp. 109–112.
15. J.Kazanavičius and D. Mažeika, " Migrating Legacy Software to Microservices Architecture2019 Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2019, pp. 1-5.