Techniques for migrating monolithic systems to microservices

Final Progress Report

Shihao Hu
Department of Computer Science
Western University

THESIS - CS 4490Z

April 8, 2022

Prof. Konstantinos Kontogiannis
(Project Supervisor)
Department of Computer Science
University of Western Ontario

Prof. Nazim Madhavji
(Course Instructor)
Department of Computer Science
University of Western Ontario

Glossary

POC	Proof of Concept
MDG	Model Dependency Graph
API	Application Programming Interface
REST	REpresentational State Transfer

STRUCTURED ABSTRACT

- **Context/Motivation:** In the era of digital transformation of enterprises, making changes without some formal procedures and strategies to a complex monolithic system is very expensive in terms of time and cost, which is often an obstacle for small and medium sized enterprises. Hence, this paper aims will explore and provide an effective approach in transforming monolithic systems to microservices.
- **Research Question:** What is the modern technique and procedure of migrating monolithic systems to microservices?
- **Principal ideas:** We will extract and analyze the dependencies of a monolithic system and then decompose and group the high coupled classes into several clusters. Then each cluster could be constructed and deployed independently as a microservice module.
- **Research approach (Methodology):** We will use Build Methodology to construct microservice systems and demonstrate the working POC for our research problem.
- **Anticipated results**: The anticipated outcome of this research would be successfully deploying the microservice modules based on the clustering results with compare and analysis of different clustering strategies or algorithms.
- **Anticipated impact of results:** The results from the research work will not only make some of the theories in academia more reliable, but also provide some practical strategies for helping the industry enterprises transform their monolithic systems to microservices.

1 Introduction

Most existing industrial systems are long-term applications with a monolithic architecture. This architecture is closely coupled, which implies that if one of the components is missing, the programme will not be executed or compiled. Typically, legacy applications grow and complexity over time, resulting in gigantic monolithic software after a few years of development, where the disadvantages of monolithic design outweigh the benefits. Fixing issues and adding new features to such an application is a time-consuming and complex process. Scalability is frequently not achievable or requires a significant amount of effort. When this occurs, businesses begin seeking for a new architectural solution and microservices come out into play (Kazanavicius et al., 2019). Microservices is an architectural and organisational approach to software development that consists of small, self-contained services that communicate over well-defined APIs. They can be written in a variety of programming languages, scale independently of other services, and be delivered on whichever hardware serves their needs best. Furthermore, owing of their smaller size, they are easier to manage and more fault-tolerant, as a single service failure will not disturb the entire system, as it would in a monolithic system (Florian et al., 2021). Hence, the transformation from monolithic systems to microservices has become an inevitable trend for facilitating the growth and competition of those enterprises.

For this research, we focus on exploring the techniques in the migration process from monolithic systems to microservices, which thoroughly discussed and addressed the key questions of how different clustering algorithms will influence the process of migrating monolithic systems to microservices and what is the modern technique and procedure of migrating monolithic systems to microservices. Also, we form our key results by implementing our theory on a small open-source monolithic system.

The novelty aspect of our research is we compared and analyzed different clustering algorithm and how they will influence the later migrating procedure. Also, we have integrated some latest techniques of spring framework for constructing the microservices, which is also significant to the industrial perspective.

This paper is composed in the following structure: section 2 introduces the background and related work, section 3 declares our research objectives, section 4 mentions the methodology that we use, section 5 demonstrates all the resulting details, section 6 discusses the threats, implications, limitations, and generalizability related to the results. Finally, section 7 and 8 concludes this paper along with view of future work and some lessons we learnt from this research.

2 Background and Related Work

2.1 Building Up the Industrial Background

- To obtain some industrial background knowledge in the field of microservices migrating, we have reviewed three papers in total, which respectively talked about the challenges of microservices migration, the benefits and drawbacks of some migration methods and the proposed roadmaps for modernizing the legacy systems with microservices.

- The first paper presented by Furda & Fidge & Zimmermann & Kelly & Barros elaborates three challenges of microservice migration where focus on multitenancy, statefulness, and data consistency. And it provides a hint of best-practice solution to develop a microservice iteratively, focusing on eliminating statefulness from the extracted legacy code, implementing multitenancy functionalities, and solving potential new introduced data consistency challenges [2].

- The second paper presented by Kazanavičius & Mažeika includes six migration process methods, which introduces and analyzeds the benefits and drawbacks for each of them. It indicates that each legacy monolithic application is special and there is no best way to migrate monolith to microservices, which results in a lack of general guidelines for migrating monoliths towards microservices [6].

- The third paper presented by IBM researchers Kalia & Krishna & Vukovic & Banerjee developed a tool called Mono2Micro for solving the daunting task of migrating monolithic systems to microservices. In this paper, they talked about the mechanism of how Mono2Micro performs the migration tasks and compare Mono2Micro against other clustering techniques on a set of open-source and proprietary Java applications and using different metrics to assess the quality of decomposition and tool's efficiency [4].

2.2 Building Up the Academic Background

- To build up academic background knowledge in the field of microservices migrating, we have reviewed eight academic papers in total.
- We have selected five papers to build up the theoretical background and enhance our understanding for this research. These papers are written by Kamimura & Yano & Hatano & Matsuo, Mazlami & Cito & Leitner, Ivanov& Tasheva, Li & Ma & Lu, Yugopuspito & Panduwinata & Sutrisno respectively and have been referenced appropriately in this paper.
- Three of these papers, respectively by Kamimura & Yano & Hatano & Matsuo, Mazlami & Cito & Leitner, Ivanov& Tasheva, introduced the technique of identifying and extracting the microservices. The common problem in these efforts is to identify from monolithic systems the candidates of microservices which includes the program files and class files or data (such as database tables, files, and data objects) that can be turned into cohesive, standalone services (Kamimura & Yano & Hatano & Matsuo, 2018).
- Other two papers, repectively by Li & Ma & Lu and Yugopuspito & Panduwinata & Sutrisno, implemented the migration process from monolithic systems to microservices on two different systems, which also demonstrating the working POC for our view.

2.3 Analysis and Research Gap

At the stage of doing literature reviews for consolidating and strengthening our research background and knowledge, we have selected several papers for analysis and research gap. For instance, the paper proposed by Kamimura & Yano & Hatano & Matsuo, developed a method that identifies the candidates of microservices from the source code by using software clustering algorithm SArF, which could also visualize the relationship between the extracted candidates and the system. The paper proposed by Zhao & Zhao proposed an approach that extract microservices candidates from monolithic systems by analyzing the database structure and combining source code relationship at the same time. In software system, domain pattern could be analyzed from the table relationships, then domain division could be achieved based previous analyzed result. We found that many research papers only focus on either the clustering techniques or the use of existing techniques on a monolithic system, and there is lack of systematic review of whole migration process from the comparing and analyzing different clustering strategies to the construction of a microservice system by using modern microservice tech stack. We will also discuss the tech stack we used to build microservices, which meant to provide some industrial perspective for the researchers.

3 Research Objectives

O1: To extract and analyze the dependencies of monolithic systems
O2: Understand the mechanism of different clustering algorithms and group the highly coupled dependencies into several clusters
O3: Decompose the monolithic systems into several independent modules based on the clustering results
O4: Construct decomposed modules as independent microservices, which will be deployed by using container techniques such as Docker
O5: Testing the microservice systems with the comparison of the monolithic one

3.1 Significance

Since the migration from monolithic systems to microservices has always been an issue, the study and explore of the clustering algorithm and the implementation of the migration process will contribute foundational research findings and knowledge into the academia of system migrating and microservices category, opening the way for future research on advanced topics incorporating unique clustering algorithms and migrating process by achieving the objectives.

In practice, it will also assist enterprises and corporations in getting a broad perspective of a formal system migration process from monolithic systems to microservices. Also, it will help enterprises to achieve higher quality microservice system migration with less effort and time. Thus, the objectives we set could help us achieve the final goal in a more consistent and effective way.

4 Methodology

For this research, we have adopted build methodology to validate our microservice migration approach. We have incorporated several third-party technologies, components, and libraries into our work to improve its development efficiency, may include but not limit to followings:

Name of Technology, System, etc.
Java Programming Language
IntelliJ Java IDEA 2021
Bunch 3.4 [8]
REST
Spring Framework
Spring Boot (2.3.12)
Spring Cloud Alibaba (2.2.7)
Spring-petclinic (monolithic system)
Mysql (5.7)

5 Results

5.1 Contextual Diagram

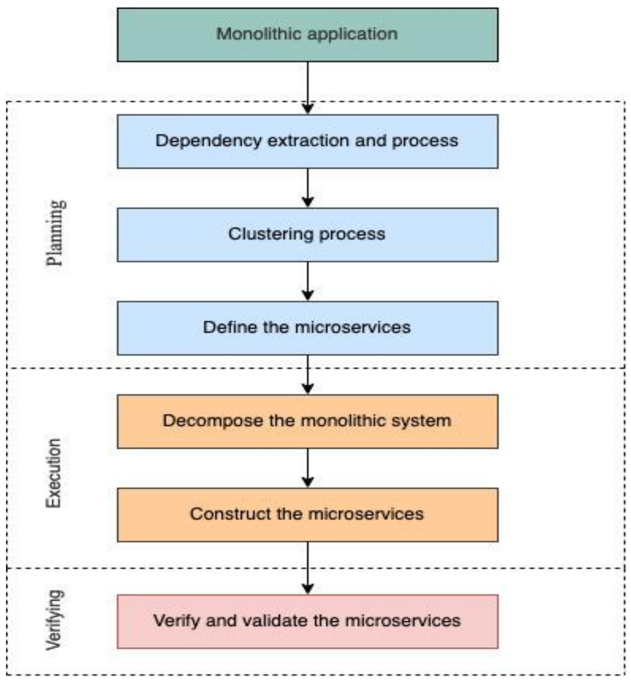

Monolithic application
Planning
Dependency extraction and process
Clustering process
Define the microservices
Execution
Decompose the monolithic system
Construct the microservices
Verifying
Verify and validate the microservices

Figure1: Migration Context Diagram

5.2 Technical Work

5.2.1 Key Requirements

We have successfully achieved all research objectives defined in section 3 except the deployment of each microservice with docker, which will be completed if the time is allowed.

5.2.2 System Design and Architecture

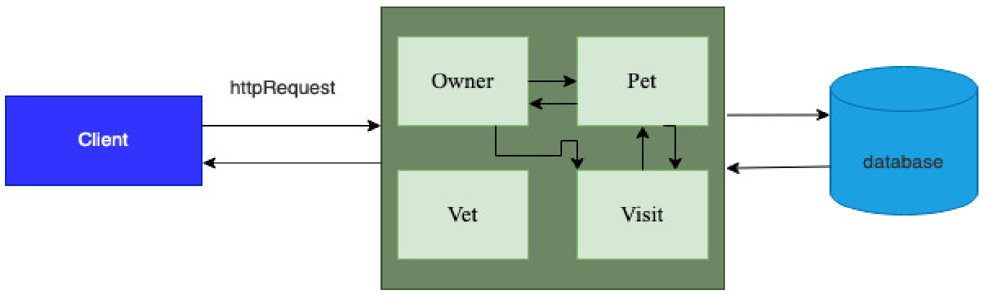


Figure2: Monolithic system

Figure2 shows that the architecture of a monolithic system which is originated from an opensource application called spring-petclinic. Our migration approach as shown in Figure1 was finally implemented on this monolithic system. The first step we do is we extract all dependencies of the system and write a small program to process the extracted information for getting a good representation of MDG. The second step is we use MDG as the input for the clustering software such as Bunch, we implement different algorithm such as basic MQ or hill climbing for getting a better clustering result. Afterwards, we will decompose the original system and construct microservices as shown in figure3, and each service is an independent application.

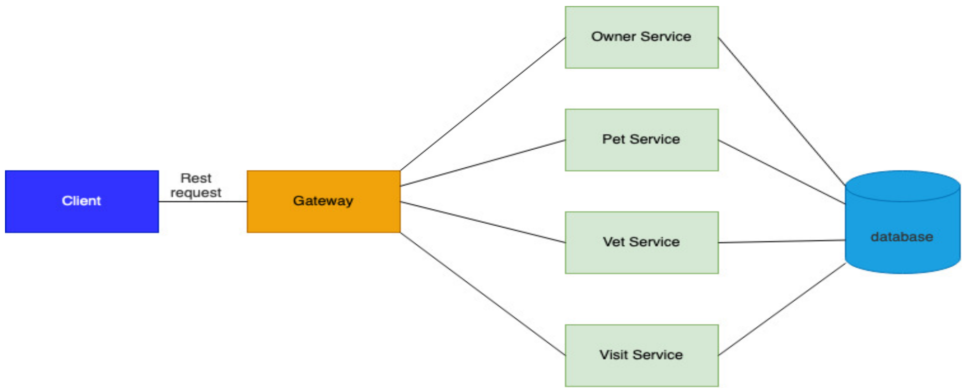


Figure 3. Reconstructed microservices

Definition 1: The API gateway is a service built between the client and the microservice. With it, the client sends the request to the API gateway first, and then the API gateway forwards the request to the microservice instance based on the request's identification information.

Definition2: A REST API is an API that follows the design principle of the REST, which enables an application or service to access a resource within another application or service.

5.2.3 System Implementation and Testing

For this research, we first used a tool called sourceNavigator with some additional process and modification for the system dependencies extraction. We then used some clustering tools such as Bunch or ACDC for the clustering process with the comparison and analysis of different algorithms.

The main language we used for constructing the microservices is Java 8, and we have adopted some Spring Framework such as Spring Boot and Spring Cloud Alibaba for helping us to develop microservice applications faster and easier.

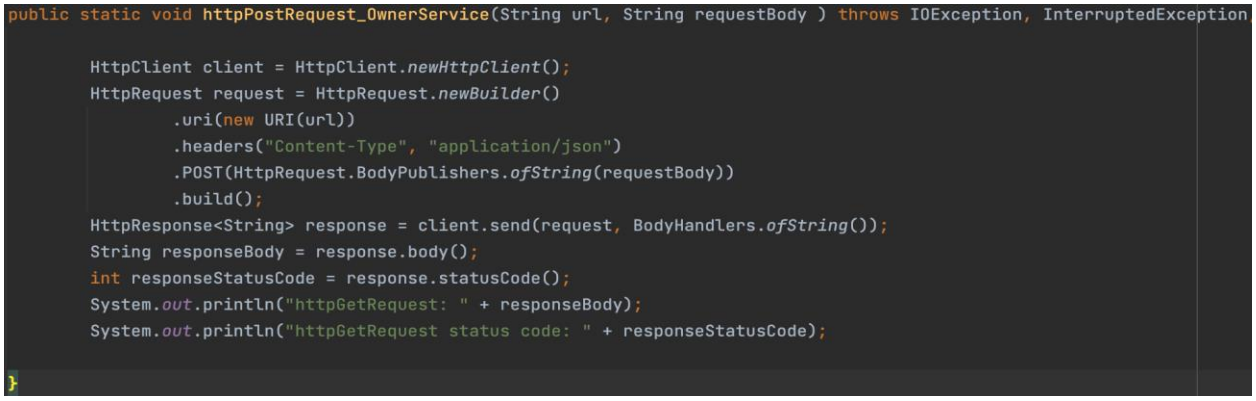
We have used API testing which is part of integration testing to test all the REST APIs we defined in each microservice, and we will take an owner service as our example for demonstrating.

For example, the following REST APIs for owner service are:

POST: http://127.0.0.1:8100/owner-service/owners/new

GET: http://127.0.0.1:8100/owner-service/owners/find/1

We have created a client module for sending http requests. We first define two methods for GET and POST requests by using the HTTP Client library, which is shown as Figure 4.



```
public static void httpPostRequest_OwnerService(String url, String requestBody ) throws IOException, InterruptedException

        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(url))
                .headers("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(requestBody))
                .build();
        HttpResponse<String> response = client.send(request, BodyHandlers.ofString());
        String responseBody = response.body();
        int responseStatusCode = response.statusCode();
        System.out.println("httpGetRequest: " + responseBody);
        System.out.println("httpGetRequest status code: " + responseStatusCode);

}
```

Figure 4. HTTP client demo

Then we test the POST API by calling the function and parse in URL such as http://127.0.0.1:8100/owner-service/owners/new with appropriate request body as we defined. For now, we have implemented above steps and tested all REST APIs for each microservice.

5.2.4 System Validation

For validating the whole system, we first package each microservice module as a jar file, then we deploy every jar file on localhost to see if they can function together well. Instead of using integration testing for each microservice module separately as we described in section 5.2.3, we will have a client-side application for validating the wellness of whole system. Then we will try to dockerize each jar file (each service) and database for simulating the isolated environment.

5.3 Novelty of Results:

We have compared our results with the results presented by Kamimura & Yano & Hatano & Matsuo. They first proposed a method which produces the candidates of microservices as the list of programs and data from the source code of monolithic applications. They use SArF software clustering algorithm with data access and the relationship between "program groups" and "data" as the input. Then they implement their approach on an open-source monolithic system, which is the same one as ours. The result they get is they reconstructed three microservices: the first service contains all java classes related to Owner and Pet modules, the second service contains all java classes related to the Visit module and the third service contains all java classes related to the Vet module. Instead, our technique presents a more granular feature, which the clustering result contains 4 clusters, which could be considered to build microservices.

6 Discussion

- Threats to the validity of the results:

  During the process of conducting this research, we found that the accuracy of model dependency graph will influence the clustering result we get. Also, we are not sure what the maximum amount of concurrency the microservice we are building can carry. Some data consistency issue still could happen since we have not used any advanced approach to handle them except for the lock, which we will resolve in the future.

- Implications of the research results:

  Since system migration has been a prominent problem these decades, the systematic analysis of monolithic systems and implementation of different clustering strategies will have some great impacts on the result of deploying the clusters as independent microservices, which will contribute some foundational research knowledge into the academia of system migrating and microservices. This thesis work will build up some research foundations and methodologies for advanced topics related to the novelty of clustering algorithms and microservice migration. It will help the organizations and enterprises to have an optimized and efficient approach of migrating their monolithic applications to microservices. Some benefits this research bring to enterprises will result in costs saving and less time consumption for system migrating process.

- Limitations of the results:

  First, our migration approach has only implemented under a small open-source application, which is less complex than a real-world industrial application. Thus, this can result in some of our methods may perform well to some larger applications. Also, our approach needs an accurate or good representation of MDG as the input for the clustering algorithm we used in Bunch. For getting more accurate clustering result, we need to pre-process the dependencies extracted from the system, and if some of the dependencies extracted from the system are missing, we may also need to add some important dependencies manually.

- Generalisability of the results:

  Our migration process could be used by enterprises who is planning to reconstruct their software architecture to microservices. Also, it may also be a good guide for some people who want to get into the area of microservices with less experience.

7 Conclusions

For this research, we have answered the problem of what is whole process of migrating monolithic systems to microservices with the compare and analysis of different clustering strategies. The research objectives from O1 to O5 that were defined earlier in section 3 have been successfully completed. The significant contribution in this research is the compare and analysis of different clustering strategies [O2], the decomposition of extracted dependencies into clusters [O3] and the construction of several microservices with modern industrial tech stack [O4].

The key results achieved are:

(I) Constructing the migration approach

(II) Building the Proof of Concept

(III) Implementing our migration theory on an open-source monolithic system

8 Future Work and Lessons Learnt

- First, we would narrow down the research area and only explore one specific area instead of the techniques for whole microservice migration process.
- Second, we would focus on more about clustering techniques and create new clustering algorithms or software.
- Third, we would develop a systematic method for solving the data consistency issue
- Finally, we would implement our migration approach on a more complex real-world system instead of a small system for acquiring more reliable and persuasive result.
- Through this research experience, the lesson I learnt is you should learn to be an independent researcher instead of heavily relying on the supervisor. Sometimes your questions may not be that hard if you would like to spend some time on it.
- For the area you have no previous knowledge, just do enough research by yourself first, then head to your supervisor and have a talk with him. Do not be ashamed to ask the same question multiple times until you figured out what your professor means.

9 Acknowledgements

First, I am honored that I can complete my undergraduate thesis under the guidance of my supervisor, Professor Konstantinos Kontogiannis, and I am very grateful to him for his professional instruction. Professor Kontogiannis has instructed me patiently on how to conduct research work from the beginning since I had no previous research experience. Also, he was willing to take much time to discuss with me the problems encountered in the project even though he was very busy. which I really appreciate that. Here I want to thank you for your professional guidance, which exercise me the ability of solving problems independently and broad my research scope.

Moreover, I would like to express my appreciation to our course instructor, Professor Nazim Madhavji, for his guidance and dedication to the thesis course. Thank you for your advice for my questions related to the course structure and appreciate your carefulness for each student.

Finally, I want to say thank you to my parents for their supported family call every week. Even though they cannot give me any professional advice, but they comfort my stressful heart all the time. Hopefully, we can reunite together when I graduate from Western.

10 References

We have reviewed the following literature so far:

[1] Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From Monolithic Systems to microservices: An assessment framework. *Information and Software Technology*, *137*, 106600. https://doi.org/10.1016/j.infsof.2021.106600

[2] 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, doi: 10.1109/MS.2017.440134612.

[3] N. Ivanov and A. Tasheva, "A Hot Decomposition Procedure: Operational Monolith System to Microservices," 2021 International Conference Automatics and Informatics (ICAI), 2021, pp. 182-187, doi: 10.1109/ICAI52893.2021.9639494.

[4] A. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, “Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices,” *arXiv.org*, 14-Sep-2021. [Online]. Available: https://arxiv.org/abs/2107.09698. [Accessed: 25-Mar-2022].

[5] M. Kamimura, K. Yano, T. Hatano and A. Matsuo, "Extracting Candidates of Microservices from Monolithic Application Code," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), 2018, pp. 571-580, doi: 10.1109/APSEC.2018.00072.

[6] J. Kazanavičius and D. Mažeika, "Migrating Legacy Software to Microservices Architecture," 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream), 2019, pp. 1-5, doi: 10.1109/eStream.2019.8732170.

[7] C. -Y. Li, S. -P. Ma and T. -W. Lu, "Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System," 2020 International Computer Symposium (ICS), 2020, pp. 519-524, doi: 10.1109/ICS51289.2020.00107.

[8] S. Mancoridis, B. S. Mitchell, Y. Chen and E. R. Gansner, "Bunch: a clustering tool for the recovery and maintenance of software system structures," Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360), 1999, pp. 50-59, doi: 10.1109/ICSM.1999.792498.

[9] G. Mazlami, J. Cito and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," 2017 IEEE International Conference on Web Services (ICWS), 2017, pp. 524-531, doi: 10.1109/ICWS.2017.61.

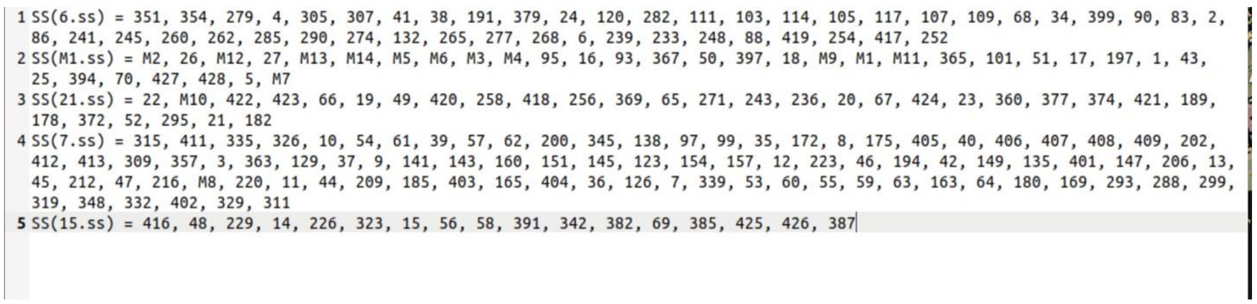
[10] J. Zhao, J. Zhou, H. Yang and G. Liu, "An orthogonal approach to reusable component n discovery in cloud migration," in China Communications, vol. 12, no. 5, pp. 134-151, May 2015, doi: 10.1109/CC.2015.7112036.

[11] P. Yugopuspito, F. Panduwinata and S. Sutrisno, "Microservices architecture: Case on the migration of reservation-based parking system," 2017 IEEE 17th International Conference on Communication Technology (ICCT), 2017, pp. 1827-1831, doi: 10.1109/ICCT.2017.8359946.

[12] S. Sarkar, G. Vashi and P. P. Abdulla, "Towards Transforming an Industrial Automation System from Monolithic to Microservices," *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 1256-1259, doi: 10.1109/ETFA.2018.8502567.

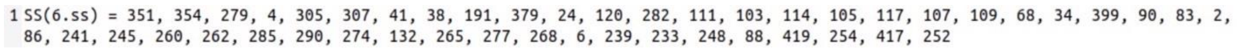
APPENDIX

The following content corresponds to the clustering process as we defined in section 5.1 Context Diagram. Here is the clustering result:



```
SS(6.ss) = 351, 354, 279, 4, 305, 307, 41, 38, 191, 379, 24, 120, 282, 111, 103, 114, 105, 117, 107, 109, 68, 34, 399, 90, 83, 2,
  86, 241, 245, 260, 262, 285, 290, 274, 132, 265, 277, 268, 6, 239, 233, 248, 88, 419, 254, 417, 252
SS(M1.ss) = M2, 26, M12, 27, M13, M14, M5, M6, M3, M4, 95, 16, 93, 367, 50, 397, 18, M9, M1, M11, 365, 101, 51, 17, 197, 1, 43,
  25, 394, 70, 427, 428, 5, M7
SS(21.ss) = 22, M10, 422, 423, 66, 19, 49, 420, 258, 418, 256, 369, 65, 271, 243, 236, 20, 67, 424, 23, 360, 377, 374, 421, 189,
  178, 372, 52, 295, 21, 182
SS(7.ss) = 315, 411, 335, 326, 10, 54, 61, 39, 57, 62, 200, 345, 138, 97, 99, 35, 172, 8, 175, 405, 40, 406, 407, 408, 409, 202,
  412, 413, 309, 357, 3, 363, 129, 37, 9, 141, 143, 160, 151, 145, 123, 154, 157, 12, 223, 46, 194, 42, 149, 135, 401, 147, 206, 13,
  45, 212, 47, 216, M8, 220, 11, 44, 209, 185, 403, 165, 404, 36, 126, 7, 339, 53, 60, 55, 59, 63, 163, 64, 180, 169, 293, 288, 299,
  319, 348, 332, 402, 329, 311
SS(15.ss) = 416, 48, 229, 14, 226, 323, 15, 56, 58, 391, 342, 382, 69, 385, 425, 426, 387
```

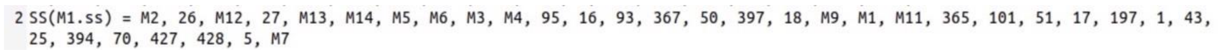
Cluster 1:



```
SS(6.ss) = 351, 354, 279, 4, 305, 307, 41, 38, 191, 379, 24, 120, 282, 111, 103, 114, 105, 117, 107, 109, 68, 34, 399, 90, 83, 2,
  86, 241, 245, 260, 262, 285, 290, 274, 132, 265, 277, 268, 6, 239, 233, 248, 88, 419, 254, 417, 252
```

Most of nodes are related to Visit.java, maybe we could make Visit as an independent microservice.

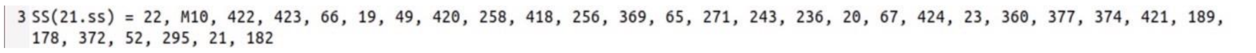
Cluster 2:



```
SS(M1.ss) = M2, 26, M12, 27, M13, M14, M5, M6, M3, M4, 95, 16, 93, 367, 50, 397, 18, M9, M1, M11, 365, 101, 51, 17, 197, 1, 43,
  25, 394, 70, 427, 428, 5, M7
```

Most of nodes are related to the module System, which we could take them into consideration for making an independent microservice.

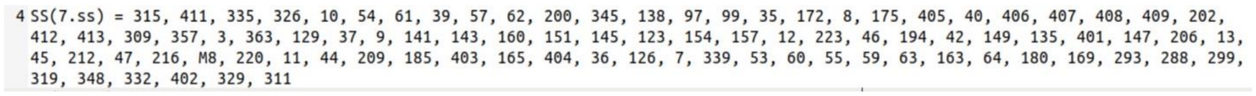
Cluster 3:



```
3 SS(21.ss) = 22, M10, 422, 423, 66, 19, 49, 420, 258, 418, 256, 369, 65, 271, 243, 236, 20, 67, 424, 23, 360, 377, 374, 421, 189,
  178, 372, 52, 295, 21, 182
```

Most of nodes are related to the module Vet and module Specialty, which we could take them into consideration for making an independent microservice. (Already done)

Cluster 4:



```
4 SS(7.ss) = 315, 411, 335, 326, 10, 54, 61, 39, 57, 62, 200, 345, 138, 97, 99, 35, 172, 8, 175, 405, 40, 406, 407, 408, 409, 202,
  412, 413, 309, 357, 3, 363, 129, 37, 9, 141, 143, 160, 151, 145, 123, 154, 157, 12, 223, 46, 194, 42, 149, 135, 401, 147, 206, 13,
  45, 212, 47, 216, M8, 220, 11, 44, 209, 185, 403, 165, 404, 36, 126, 7, 339, 53, 60, 55, 59, 63, 163, 64, 180, 169, 293, 288, 299,
  319, 348, 332, 402, 329, 311
```

Most of nodes are related to OwnerController and PetController.java and Pet.java, which we could take it into consideration for making an independent microservice. But we also could separate into two services if possible.