Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 2, April 2024, pp. 357-365

DOI: https://doi.org/10.52436/1.jutif.2024.5.2.1338
p-ISSN: 2723-3863
e-ISSN: 2723-3871

DOCKER-BASED MONOLITHIC AND MICROSERVICES ARCHITECTURE PERFORMANCE COMPARISON

Deni Panji Dirgantara*[1], Dana Sulistyo Kusumo[2], Rio Guntur Utomo[3]

[1,3]Information Technology Department, School of Computing, Telkom University, Indonesia
[2]Software Engineering Department, School of Computing, Telkom University, Indonesia
Email: [1]denipanjid@student.telkomuniversity.ac.id, [2]danakusumo@telkomuniversity.ac.id, [3]riogunturutomo@telkomuniversity.ac.id

(Article received: August 19, 2023; Revision: September 14, 2023; published: April 04, 2024)

Abstract

Most developers still use the monolithic architecture, where all components of an application are combined into one integrated system, so each part depends on other components. The monolithic architecture has weaknesses, such as when a failure occurs in one component, all parts cannot be executed because each component relies on one other component. Microservices can be a solution to this, considering that in the microservices architecture, each element or service is created and put separately, so when a failure occurs in one component, other components will not be affected and can still run normally. This research aims to determine the implementation and performance comparison between monolithic architecture and microservices Architecture in the Agreeculture Market web app. Agreeculture Market is a web application that aims to facilitate the transaction process of agricultural commodities and make it easier for agricultural commodity producers to market their products. The measurement method used to measure the performance of both architectures is load testing using JMeter and performance tools from task manager and comparing the response time, throughput, disk usage, CPU usage, and memory usage of both used architectures. With two measurement schemes with Docker and without Docker, the result of this research is a performance comparison between the two architectures, where the backend application Agreeculture Market, which uses microservices architecture with Docker and API gateway, performs better than the monolithic architecture version. Conversely, the monolithic architecture performs better than the microservices architecture in the scheme without Docker and API gateway.

Keywords: *docker, microservices architecture, monolithic architecture, performance comparison.*

1. INTRODUCTION

Monolithic Architecture is still popular among developers, especially beginners, because of its easier implementation [1], [2]. However, microservices architecture is widely used today because it has several advantages, such as easy service organisation, the ability to use different technologies, and the ability to make updates without the need to redeploy all services. On the other hand, Microservice Architecture also has disadvantages, such as increasing complexity as services boost and requiring advanced skills from developers [1], [3].

Microservices architecture is an application architecture where each component or can be called services is placed separately from other services [2]. It differs from a monolithic architecture combining all services in one codebase. According to [6], microservices architecture has advantages in terms of scalability and also fast development cycles. The scalability of microservices architecture can be said to be better than monolithic architecture because it does not need to change many components to develop the entire application so that it can be easier and faster [6], [7]. in addition, microservices architecture also has easier maintenance compared to monolithic architecture [8], [9].

Docker container is necessary for developing and running microservices applications. Because, with docker container, the authors can run all services simultaneously [4]. In [5], the authors measured the performance of microservices applications using docker container and monolithic applications without using docker container, so a further approach is needed to measure microservices and monolithic performance with better scenarios.

This research aims to compare the implementation and performance between microservice architecture and monolithic architecture on the agreeculture market web app, hoping to show the advantages of using microservice architecture in terms of application efficiency and performance. Agreeculture Market is a web application that aims to facilitate the transaction process of agricultural commodities and make it easier for agricultural commodity producers to market their products.

This research is divided into five sections as follows: section one is the introduction of this paper, section two discusses related previous research, section three discusses the development method and structure of the application, section four discusses the results of the performance comparison analysis between microservices architecture and monolithic architecture, and section five contains conclusions and suggestions from the research conducted.

2. METHOD

The authors use a conceptual model to facilitate the flow of development and performance testing of the Agreeculture Market backend web application. The flow of development and performance testing of the Agreeculture Market backend web application using Microservices Architecture technology is shown in the following figure 1.

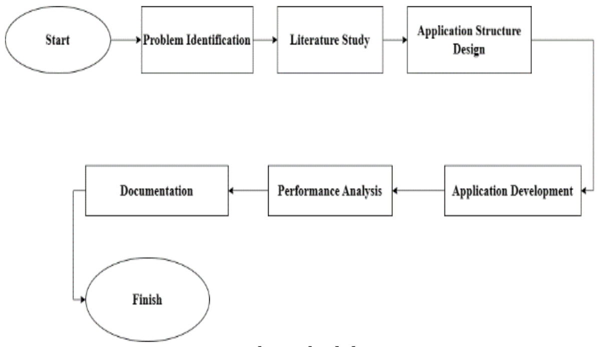

Figure 1. Research Methodology Diagram

1. The problem identification stage is the stage where the authors identify existing problems.
2. At the literature study stage, the authors conducted a literature study of previous research to obtain a theoretical basis related to the research being conducted.
3. The authors started developing the Agreeculture Market backend web application at the application development stage with the NodeJS framework.
4. The performance analysis stage is where the authors take measurements and compares the measurement results between the applications built with the two architectures used, microservices and monolithic architectures.
5. After the performance analysis stage, the authors documented the entire research and conducted the documentation stage.

2.1. Structure of Agreeculture Market Web Application with Microservices Architecture

Six services have different business objectives in the Agreeculture Market web application with Microservice Architecture shown in figure 2.

1. Wishlist Service aims to display and process data on items buyers want. Wishlist Service is run on port 8080 for API Gateway and port 9025 if without API Gateway.
2. Cart Service has the purpose of displaying and processing shopping list data from unpaid buyers. Cart Service is run on port 8080 for API Gateway and port 9010 without API Gateway.
3. User Service has the purpose of processing user data with an account on the Agreeculture Market and is divided into two roles: buyers and sellers. User Service is run on port 8080 for API Gateway and port 9000 without API Gateway.
4. Offer Service aims to provide offer data related to products in the Agreeculture Market web application to buyers. Offer Service is run on port 8080 for API Gateway and port 9015 without API Gateway.
5. Transaction Service aims to process data related to buyer transactions, such as transaction status and terms. Transaction Service is run on port 8080 for API Gateway and port 9005 without API Gateway.
6. Product Service aims to display and process product data in the Agreeculture Market Web Application. Product Service is run on port 8080 for API Gateway and port 9020 without API Gateway.

2.2. Structure of Agreeculture Market Web Application with Monolithic Architecture

The web application Agreeculture Market application structure with monolithic architecture has the same service structure as the microservice mrchitecture but is differentiated based on figure 3. Agreeculture Market Web Application with Monolithic Architecture runs on port 9001.

2.3. Performance Measurement

In this study, the authors measured performance with the load testing method, and the tools used were JMeter [16]. There are two measurement schemes, the measurement scheme with the use of docker container and without docker container. Both architectures will be measured in each scheme, and the measurement results using the two architectures will be compared.

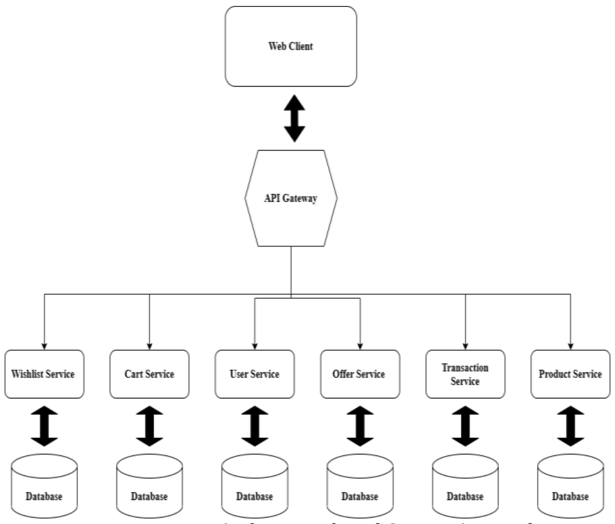

Figure 2. Microservices Architecture-based Service Agreeculture Market Structure Diagram

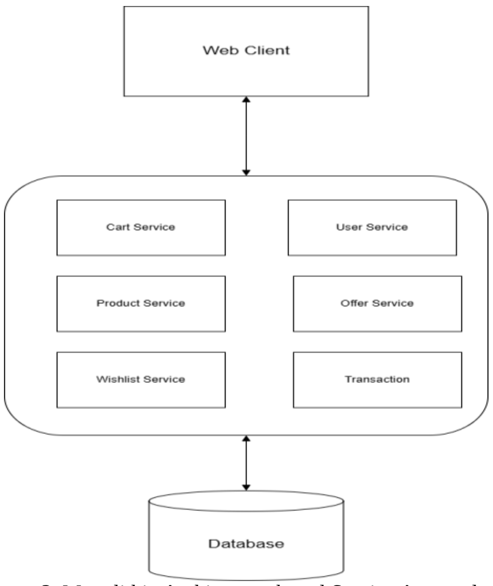

Figure 3. Monolithic Architecture-based Service Agreeculture Market Structure Diagram

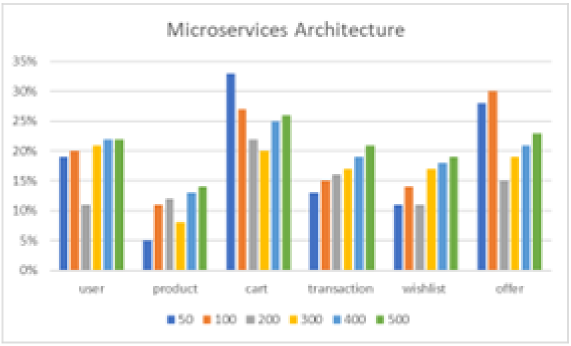

Figure 4a. CPU usage on microservices application on schemes with docker

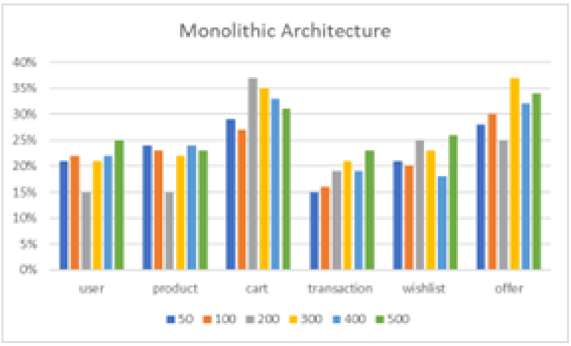

Figure 4b. CPU usage on monolithic application on schemes with docker

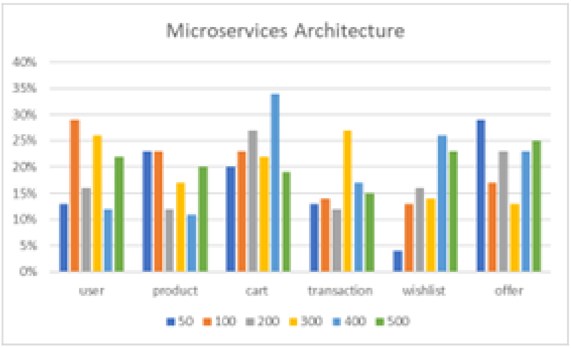

Figure 4c. CPU usage on microservices application on schemes without docker

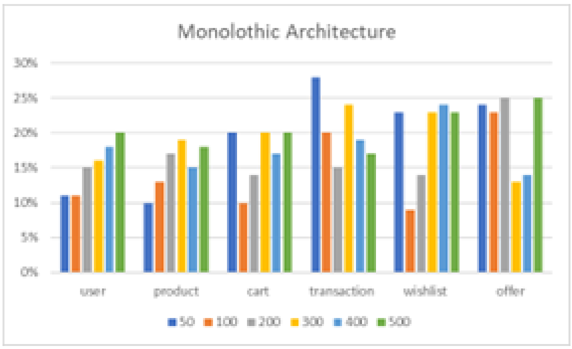

Figure 4d. CPU usage on monolithic application on schemes without docker

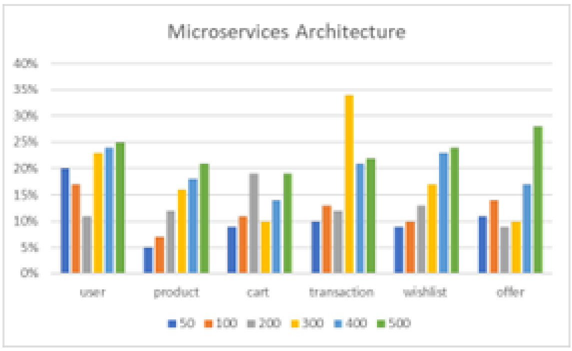

Figure 4e. Disk usage on microservices application on schemes with docker

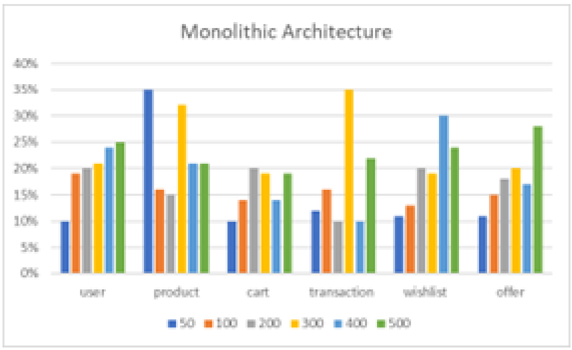

Figure 4f. Disk usage on monolithic application on schemes with docker

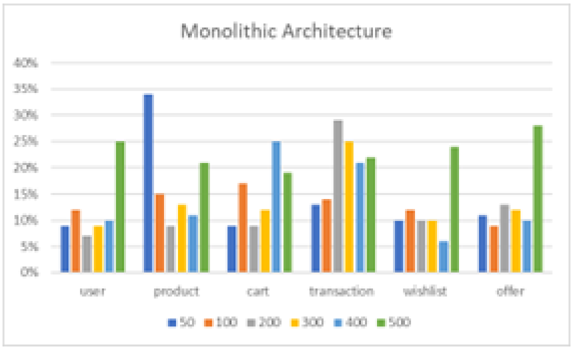

Figure 4g. Disk usage on microservices application on schemes without docker

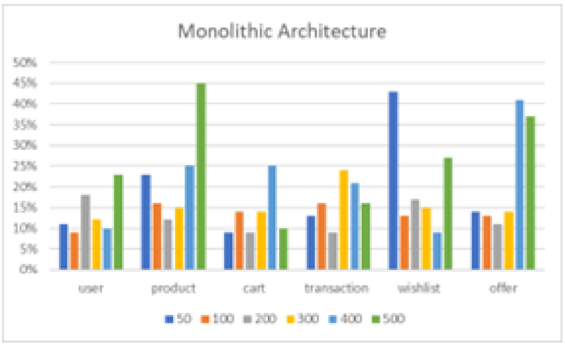

Figure 4h. Disk usage on monolithic application on schemes without docker

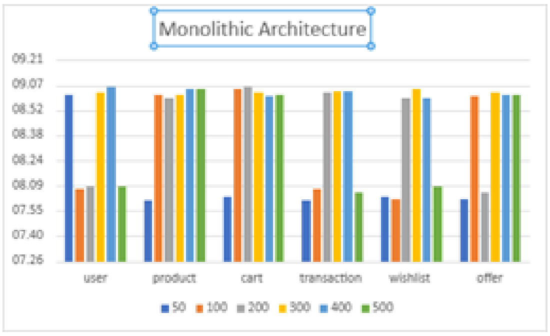

Figure 4j. Memory usage on monolithic application on schemes without docker

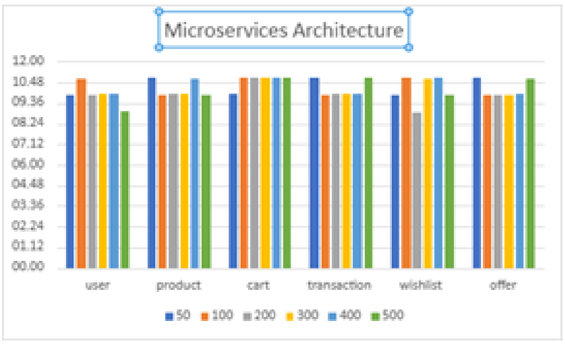

Figure 4i. Memory usage on microservices application on schemes with docker

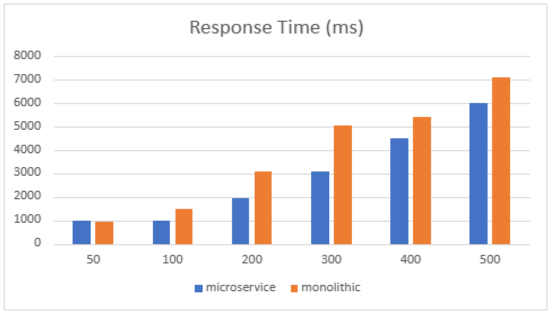

Figure 5a. User service response time on scheme with docker

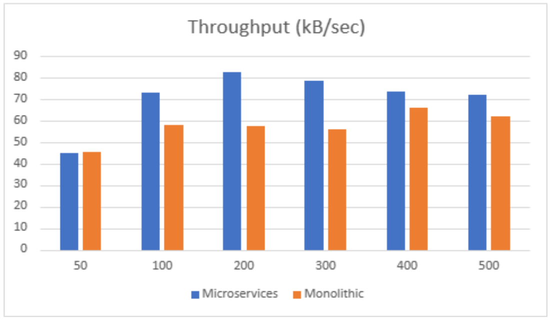

Figure 5b. User service throughput on scheme with docker

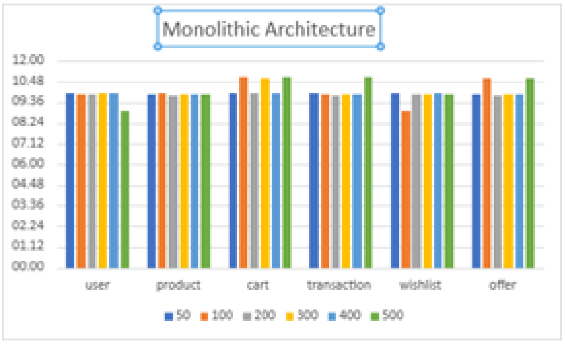

Figure 4j. Memory usage on monolithic application on schemes with docker

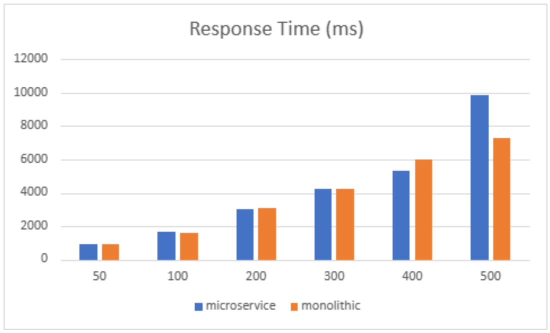

Figure 5c. User service response time on scheme without docker

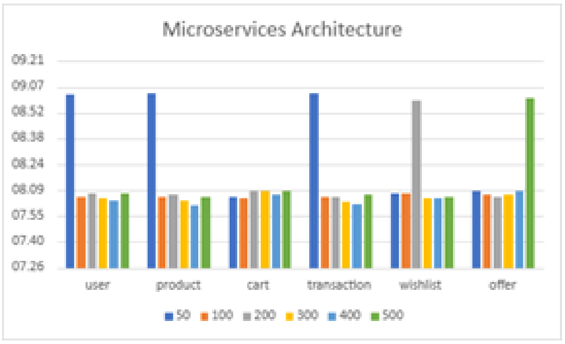

Figure 4j. Memory usage on microservices application on schemes without docker

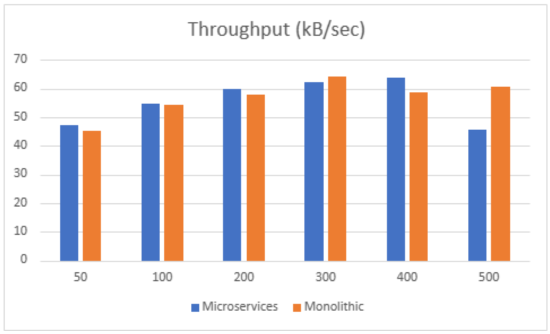

Figure 5d. User service throughput on scheme without docker

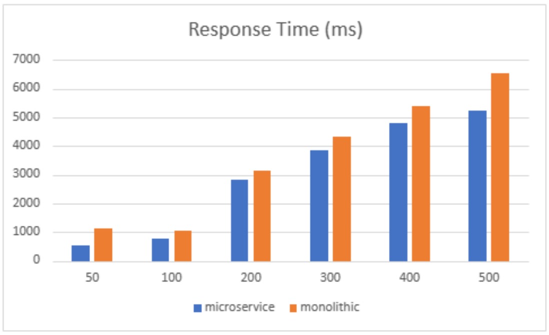

Figure 6a. Product service response time on scheme with docker

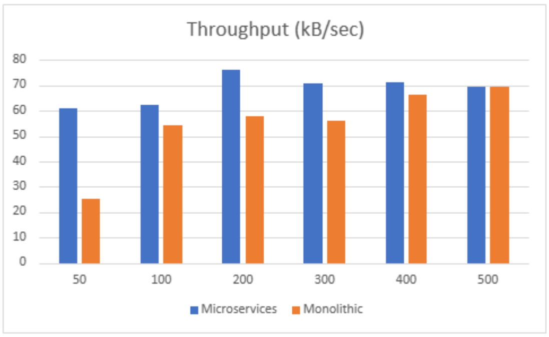

Figure 7b. Cart service throughput on scheme with docker

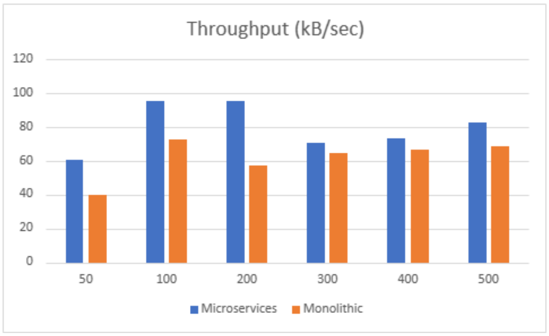

Figure 6b. Product service throughput on scheme with docker

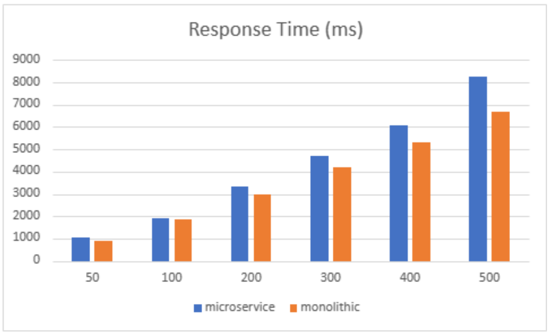

Figure 7c. Cart service response time on scheme without docker

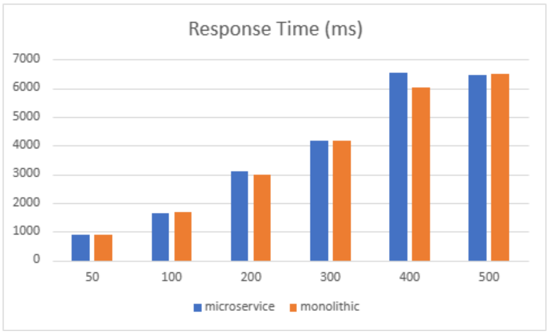

Figure 6c. Product service response time on scheme without docker

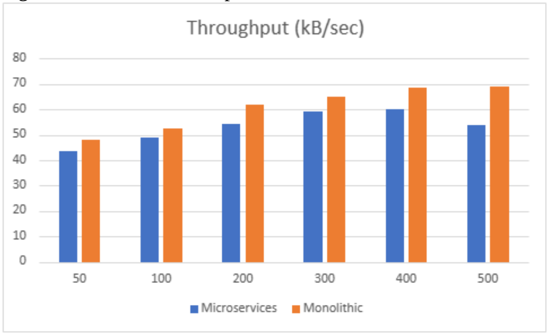

Figure 7d. Cart service throughput on scheme without docker

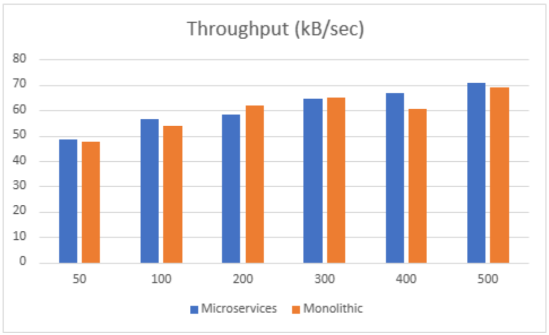

Figure 6d. Product service throughput on scheme without docker

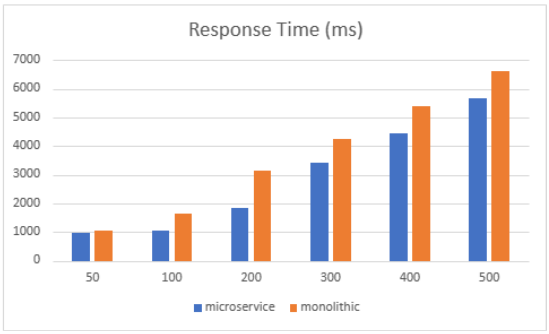

Figure 8a. Transaction service response time on scheme with docker

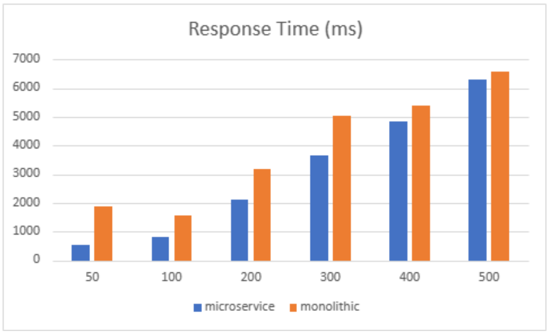

Figure 7a. Cart service response time on scheme with docker

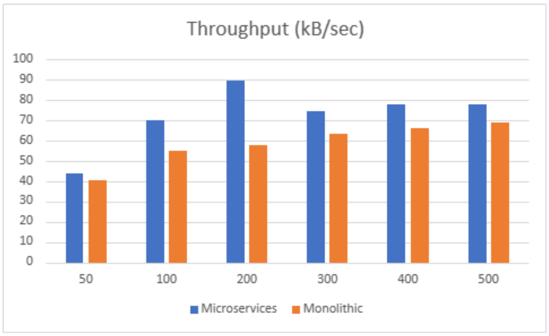

Figure 8b. Transaction service throughput on scheme with docker

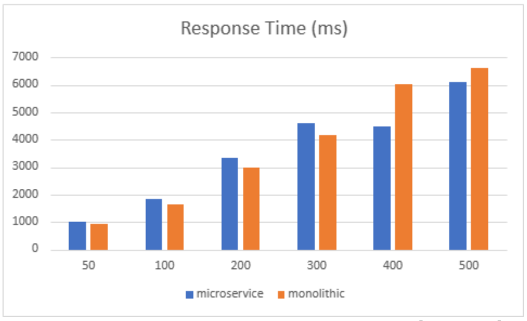

Figure 8c. Transaction service response time on scheme without docker

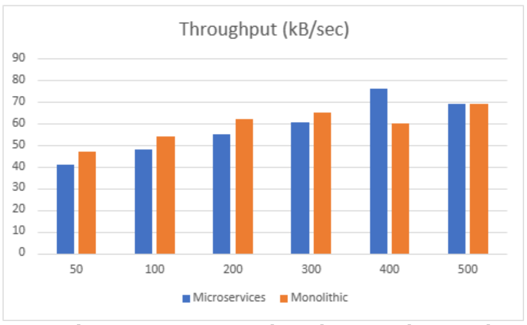

Figure 8d. Transaction service throughput on scheme without docker

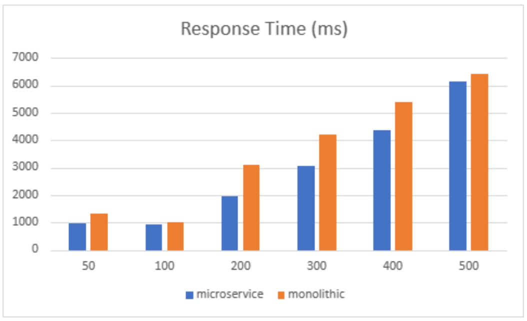

Figure 9a. Wishlist service response time on scheme with docker

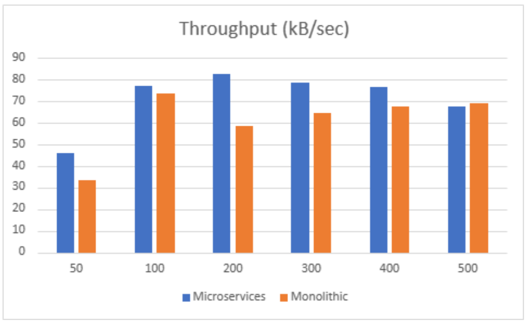

Figure 9b. Wishlist service throughput on scheme with docker

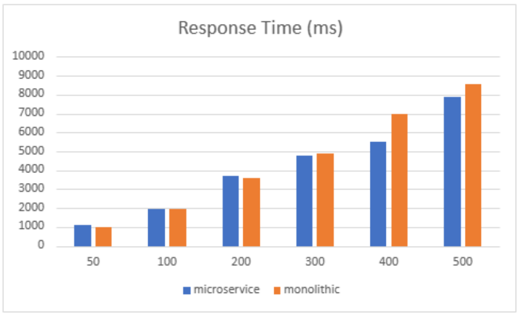

Figure 9c. Wishlist service response time on scheme without docker

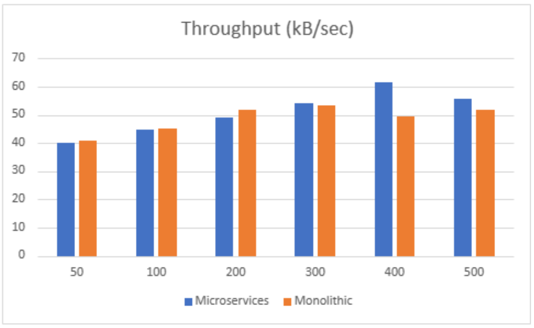

Figure 9d. Wishlist service throughput on scheme without docker

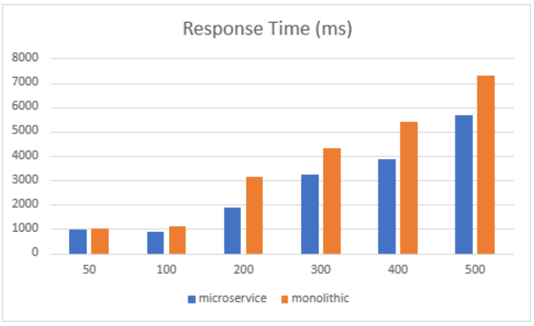

Figure 10a. Offer service response time on scheme with docker

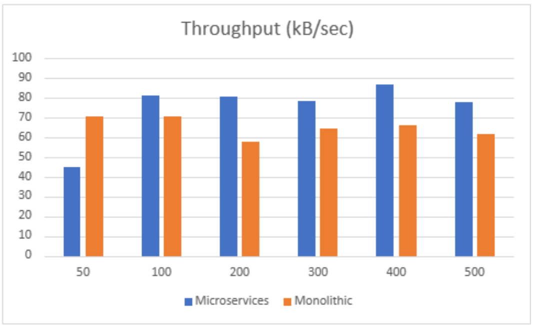

Figure 10b. Offer service throughput on scheme with docker

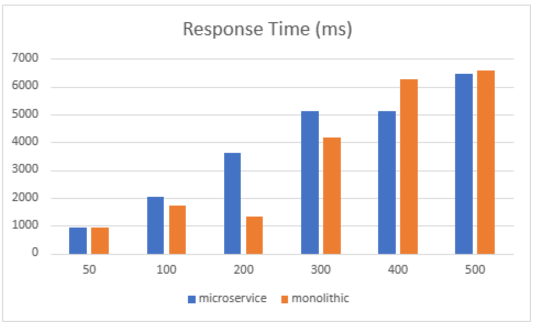

Figure 10c. Offer service response time on scheme without docker

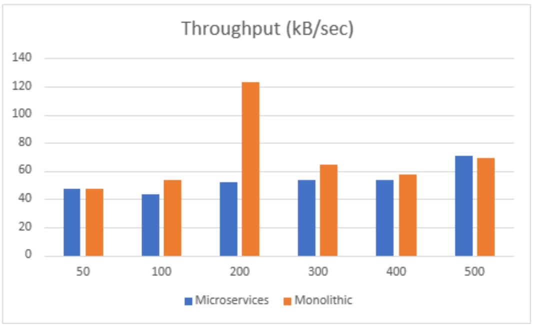

Figure 10d. Offer service throughput on scheme without docker

3. RESULT

3.1. Test Result and Comparison

Evaluation of the Agreeculture Market web application is carried out by measuring the performance of the Agreeculture Market web application using monolithic architecture and microservices architecture. The parameters used during measurement are response time and throughput, which will be measured using JMeter, and CPU usage, disk usage, and memory usage, which will be measured using performance tools from the task manager. These parameters are sufficient to measure performance because the developed application is run on localhost [16]. The measurement schemes used is when the application is run using Docker and API Gateway and when the application is run without Docker and API Gateway. The measurement was carried out using the load testing method with the number of virtual users of 50, 100, 200, 300, 400, and 500. After the measurement, the performance measurement results will be compared to determine which architecture is better in each test scheme. Tests were carried out on equipment with the characteristics presented below :

1. CPU : Intel Core i7-9750H @ 2.6 GHz 4.5GHz
2. OS: Windows 10 1909 with WSL (Windows Subsystem for Linux) Ubuntu 20.04.4 LTS installed
3. 16 GB RAM
4. 128 GB SSD ROM.

3.2. Result

1. CPU, Disk, and Memory Usage

The comparative analysis of CPU, disk, and memory usage shows differences in the results of the two test schemes. In the docker usage scheme on figures 4a – 4c, microservices applications' average memory and disk usage has lower results than monolithic applications. This is due to API gateway and load balancer in the microservices application run on docker to connect each existing service. On the other hand, in the scheme without the use of docker, the average CPU and disk usage of monolithic applications has lower results in most services in figures 4d – 4f. The absence of API gateway and load balancer is why this can happen. Because in the scheme without docker, microservices applications run on different ports.

2. User Service

User service based on figures 5a and 5b, the microservice applications yield better performance than monolithic applications on docker deployment schemes. This is due to API gateway and load balancer use in microservice applications that connect each service. But on the other hand, based on figure 5b, monolithic applications perform better on most services than microservice applications on schemes without docker. In the scheme without docker, microservice applications do not use API gateway and load balancer and run on separate ports.

3. Product Service

Based on figures 6a and 6b, product service on microservices architecture and docker performs better than applications that use monolithic architecture in response time and throughput. This is due to the use of API Gateway and Load Balancer, where both are used to connect each service in the microservices application. But in the scheme without Docker in figures 6c and 6d, monolithic applications have better performance mostly on all services than microservices applications. This can happen because services in microservice applications run on separate ports, and there is no use of API gateway and load balancer in microservices applications.

4. Cart Service

Based on figure 7a and 7b, the performance of applications that use microservices architecture with Docker have better response time and throughput than applications that use monolithic architecture. API gateway and load balancer are why microservices applications perform better than monolithic applications in the Docker deployment scheme. On the other hand, in the scheme without the use of docker on figure 7c and 7d, the monolithic application performs better in all services due to the non-use of API gateway and load balancer. In the scheme without ocker, microservices applications run on a separate port for each service so that they perform worse than monolithic applications.

5. Transaction Service

Based on figure 8a and 8b, microservice applications yield better performance in transaction service than monolithic applications on docker deployment schemes. This is due to API gateway and load balancer use in microservice applications that connect each service. On the other hand, on figure 8c and 8d, monolithic applications perform better on most services than microservice applications on schemes without docker. In the scheme without Docker, microservice applications do not use API gateway and load balancer and run on separate ports.

6. Wishlist Service

Based on figures 9a and 9b, wishlist service on microservices architecture and Docker performs better than applications that use monolithic architecture in response time and throughput. This is due to the use of API gateway and load balancer, where both are used to connect each service in the microservices application. But monolithic applications perform better in almost all services than the scheme's microservices applications without docker on figure 9c and 9d. This can happen because services in microservice applications run on separate ports, and there is no use of API gateway and load Balancer in microservices applications.

7. Offer Service

Based on figure 10a and 10b, the offer service in microservices applications performs better than monolithic applications. API gateway and load balancer are why microservices application perform better than monolithic application in the docker deployment scheme. On the other hand, in the scheme without Docker on figure 10c and 10d, the monolithic application has better performance almost in all services due to the non-use of API gateway and load balancer. In the scheme without docker, microservices application run on separate port for each services so they perform worse than monolithic application.

3.3. Discussion

Of the six services that have been measured and compared, all applications that use microservices architecture with docker perform better than those that use monolithic architecture with docker's help in response time and throughput. CPU and disk usage of microservices architecture has a lower average when compared to monolithic architecture. However, the memory usage of microservices architecture and monolithic architecture applications have the same average memory consumption due to docker, which consumes more than when the application runs without docker. Conversely, the performance of monolithic architecture without the use of docker has a higher value when compared to microservices architecture, even though, in some test cases, microservices architecture still performs better. This is because when the microservices architecture application is run without docker, each service is on a different port. In contrast, in the use of docker, each service is connected using API gateway and runs only on one port and the application load balancer from the API gateway used, namely NGINX, which also affects the performance improvement of microservices architecture. In this research, the authors compare the performance between microservices and monolithic architectures in contrast to previous research. The study that is the reference of this research [3], [5], [7]–[10], Some of these studies use different development technologies and performance measurement methods and tools. In addition, some of the referenced studies only produce applications without performance measurement. To complement the research study that is the reference of this research [11], [13]–[15], The authors of this research complements previous research by performing performance comparisons between microservices and monolithic architectures running in Docker Container and on-premises environments.

4. CONCLUSION

Based on the comparison of the results of performance tests that have been carried out using the load testing method, it can be concluded that the Agreeculture Market web backend application that uses microservices architecture has better performance than applications that use monolithic architecture in terms of response time and application throughput in the docker usage scheme. However, when the application is run without API gateway and docker, monolithic architecture performs better in almost every service, even though microservice architecture performs better in some test cases. In addition, the use of API gateway also affects the performance improvement of microservices architecture, where the API gateway already has a load balancer. In this research [5], microservices applications with docker container perform better than monolithic applications that run on a local environment. However, in research related to Agreeculture Market, the two architectures have different results between those run on docker container and the local environment. With the docker container, microservices applications produce better performance than monolithic applications. Conversely, monolithic applications performed better than microservices applications when both architectures were run on a local environment. The impact of this research is to know the efficiency and performance of microservices and monolithic architectures when using docker dontainer or without docker container. Remember that this research still uses the Docker environment and NoSQL database. This research can be expanded and developed by deploying applications and combining NoSQL and SQL databases in a cloud-based environment. Further research and more sophisticated technology are needed to get more valid results.

REFERENCES

[1] M. E. Gortney *et al.*, "Visualizing Microservice Architecture in the Dynamic Perspective: A Systematic Mapping Study," *IEEE Access*, vol. 10. Institute of Electrical and Electronics Engineers Inc., pp. 119999–120012, 2022. doi: 10.1109/ACCESS.2022.3221130.

[2] C. V. Dave, "Microservices Software Architecture: A Review," *Int J Res Appl Sci Eng Technol*, vol. 9, no. 11, pp. 1494–1496, Nov. 2021, doi: 10.22214/ijraset.2021.39036.

[3] J. A. Rasheedh and S. Saradha, "Design and Development of Resilient Microservices Architecture for Cloud Based Applications using Hybrid Design Patterns," *Indian Journal of Computer Science and Engineering*, vol. 13, no. 2, pp. 365–378, Mar. 2022, doi: 10.21817/indjcse/2022/v13i2/221302067.

[4] N. Singh *et al.*, "Load balancing and service discovery using Docker Swarm for microservice based big data applications," *Journal of Cloud Computing*, vol. 12, no. 1, Dec. 2023, doi: 10.1186/s13677-022-00358-7.

[5] K. Gos and W. Zabierowski, “The Comparison of Microservice and Monolithic Architecture,” in *International Conference on Perspective Technologies and Methods in MEMS Design*, 2020, pp. 150–153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[6] A. Bucchiarone *et al.*, “From Monolithic to Microservices An Experience Report from the Banking Domain,” 2018.

[7] J. Xiang, “Microservices-based Dating Platform,” 2021. [Online]. Available: https://martinfowler.com/articles/

[8] J. Ferdinand, A. Syahrina, and A. Musnansyah, “PERANCANGAN ARSITEKTUR PERANGKAT LUNAK MICROSERVICES PADA APLIKASI OPEN LIBRARY UNIVERSITAS TELKOM MENGGUNAKAN gRPC,” *TELKATIKA*, vol. 1, no. 2, 2022.

[9] G. Blinowski, A. Ojdowska, and A. Przybylek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[10] F. Ponce Mella, G. Márquez, H. Astudillo, and F. Ponce, “Migrating from monolithic architecture to microservices: A Rapid Review BaaS-SE Blockchain as a Service for Stock Exchange View project TacPat4SS-Empirical evaluation of tactics and patterns for building secure systems View project Migrating from monolithic architecture to microservices: A Rapid Review,” 2019. [Online]. Available: https://www.researchgate.net/publication/335716451

[11] I. Braun, M. Hoffmann, and R. Mörseburg, *IMPLEMENTATION OF A WEB-BASED AUDIENCE RESPONSE SYSTEM AS MICROSERVICE APPLICATION VS. MONOLITHIC APPLICATION*. 2019.

[12] Óbudai Egyetem, IEEE Hungary Section, M. IEEE Systems, Hungarian Fuzzy Association, and Institute of Electrical and Electronics Engineers, *18th IEEE International Symposium on Computational Intelligence and Informatics : proceedings : 2018 November 21-22, Budapest.*

[13] S. Jhingran and N. Rakesh, “PERFORMANCE ANALYSIS OF MICROSERVICES BEHAVIOR IN CLOUD VS CONTAINERIZED DOMAIN BASED ON CPU UTILIZATION,” *Journal of Data Acquisition and Processing*, vol. 38, no. 2, p. 3221, doi: 10.5281/zenodo.777159.

[14] P. Sharad Salunkhe, “Microservices vs Monolithic Architecture: Load Testing in AWS on ReactJS Web Application for Performance MSc Research Project Programme Name.”

[15] N. Goncalves, D. Faustino, A. R. Silva, and M. Portela, “Monolith Modularization towards Microservices: Refactoring and Performance Trade-offs,” in *Proceedings - 2021 IEEE 18th International Conference on Software Architecture Companion, ICSA-C 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 54–61. doi: 10.1109/ICSA-C52384.2021.00015.

[16] D. Demashov, “Efficiency Evaluation of Node.js Web-Server Frameworks,” 2019.