International Journal of Advances in
Soft Computing and Intelligent Systems (IJASCIS)
2023, Vol 02, Issue 02, 08-21

Science Transactions © 2023

Original Paper

ANALYSIS OF MONOLITHIC AND MICROSERVICES SYSTEM ARCHITECTURES FOR AN E-COMMERCE WEB APPLICATION

Allur Shivprasad Rao[a], Ajay Kotalwar[b]
Sanket Mendhe[c], Dr. Meenakshi Thalor[d]

[a,b,c,d] AISSMS Institute of Information Technology, Pune, 411001,
shivarao34@gmail.com, kotalwarajay27@gmail.com,
sanketmendhe33@gmail.com, meenakshi.thalor@aissmsioit.org

ABSTRACT

This research paper explores the relative merits of monolithic and microservices architecture for E-Commerce web applications, using Express JS and Node JS as the primary technologies. The study provides a comprehensive examination of the two architecture patterns and employs a practical approach to demonstrate the differences. The architecture is compared based on metrics such as latency, throughput, response-time, error percentage and cost. The findings indicate that when it comes to large and complex applications, microservices architecture outperforms monolithic architecture in terms of scalability and reliability. On the other hand, monolithic architecture offers a simpler and more straightforward approach for small-scale applications. Moreover, monolithic architecture also provides better results for a small-scale approach whereas microservices architecture would be an expensive approach. In the experiment, we found that monolithic architecture gives satisfactory results compared to microservices architecture while having low traffic. However, the error percentage of monolithic architecture is extremely high while having heavy traffic whereas microservices architecture handles heavy traffic with a very low error percentage. In the paper we conclude that the appropriate choice of architecture pattern should be determined by the unique needs of the project. The objective of this research is to evaluate the monolithic and microservices architectures for an ecommerce use case, and to propose guidelines for small and large scale enterprises on which architecture to implement. This is a generic use case that does not account for any specific conditions or constraints.

KEYWORDS

Microservices, Monolithic, Software Architecture, Comparison, Architectural Metrics, Performance Testing

1. INTRODUCTION

Modern businesses have used microservices architecture, including Amazon, Netflix, Uber, and Spotify. Financial, e-commerce, and travel service providers are switching from monolithic to microservices architecture.

The term "monolithic architecture" refers to a single-tiered software application in which various components are combined into a single programme from a single platform. Despite the fact that the application contains multiple components/modules/services, it is built and deployed as a single application for all platforms (desktop, mobile and tablet). [4] Advantages of monolithic architecture include:

- Easy to develop and deploy
- Simple to understand and maintain
- Good for small-scale applications
- Cost-effective compared to microservices architecture

However, monolithic architecture can become unwieldy and difficult to maintain as applications grow in size and complexity, leading to a greater risk of codebase degradation and long development cycles. This is one of the reasons why microservices architecture has gained popularity.

Microservices architecture is a software design approach where an application is composed of small, independently deployable services. Each service is focused on performing a specific business function and can be developed, deployed, and scaled independently of the other services. Advantages of microservices architecture include:

- Improved scalability
- Better fault tolerance
- Enhanced maintainability
- Increased deployment velocity
- Facilitation of a polyglot development environment
- Ability to evolve systems incrementally
- Better support for continuous delivery and deployment
- Easier integration with diverse systems and technologies

However, microservices architecture can be more complex to set up and manage compared to monolithic architecture, and can also lead to increased operational overhead and communication overhead between services. It also requires a high level of technical expertise.

The paper compares the monolithic and microservices architecture specifically for E-Commerce web applications. Based on the research conducted, the architectures are compared on the metrics such as latency, throughput, error percentage, response-time and cost. Because of the ever-increasing popularity of media sites such as Instagram, YouTube, Facebook, etc. e-commerce businesses are also gaining popularity as it has become easier to market their products on a huge scale. With this increasing popularity, consequently, network traffic is bound to increase. This can result in various problems for small businesses such as low tolerance and inability to scale. Whereas, small scale businesses should not adopt microservices architecture as it would be expensive and the utilization would be insufficient, which in turn would make the business harder to sustain. Hence, the research paper focuses on selecting the appropriate architecture for an e-commerce web application based on the necessities, requirements and future scope. Moreover, we are not focusing on any particular domain, platform, or functionality that may be unique to a certain application or service. Instead, we are using a common and generic scenario that can be applied to any application or service that follows the monolithic or microservices architecture. This allows us to compare the architectures in a fair and objective way, without introducing any bias or confounding factors that may affect the results.

The rest of the paper is organised as follows. Section 2 describes the related work. The research method and the experimental design are explained in Section 3. Section 4 presents the implementation details of the proposed approach. This is followed by Section 5, which contains the experimental results. Finally, Section 6 summarizes and concludes the paper by drawing some conclusions.

2. LITERATURE REVIEW

The authors of the work by Raj V. et al.[1] offered a comparison of a web application created utilizing both Service Oriented Architecture (SOA) and microservices architectures. Two separate parameters are used in the comparison: 1) Architectural metrics for complexity; 2) load testing for performance. The results demonstrated that even though the microservices design is sophisticated, it responds to HTTP requests far more quickly than SOA services.

Grzegorz Blinowski et al.[2], compares the Monolithic and Microservice Architecture. Microservices-based architecture has gained widespread popularity due to its advantages, such as improved availability, fault tolerance, and horizontal scalability, as well as greater software development agility. The key lesson is on a single machine, a monolithic performs better than its microservice-based counterpart.

Microservices and monolithic architectures are compared in terms of performance in Omar Al-study Debagy's [3], in order to ascertain how these architectures perform in various scenarios using various testing setups. This paper comes to the conclusion that monolithic applications and microservices can perform similarly when the application is under normal load. A monolithic application may perform marginally better than a microservices application under a light load of fewer than 100 users.

The authors of the work by Konrad Gos et al.[4] compared the Monolithic and Microservices architecture using the Gatling load testing tool. The tests were performed on PC with Ubuntu 18.04.2 LTS operating system. The applications were deployed with Docker. This paper compares the monolithic and microservices architecture on different parameters like Architecture performance, response time by sending a number of HTTP GET and POST requests. This paper also describes the pros and cons of Monolithic and Microservices architecture.

Table 1. Literature Review.

Ref. No	**Paper Reference**	**Methodology**	**Conclusion**
[1]	Performance and complexity comparison of Service Oriented Architecture and Microservices Architecture	In this paper, to compare the Service Oriented Architecture and Microservices Architecture Complexity and Performance of both the architectures is analyzed. The authors have built a standard web-based application (Vehicle Management System) to perform the comparison of two architectures by sending various number of requests.	The authors of this paper used JMeter to compare the performance of both the Service Oriented Architecture (SAO) and the microservices architecture. Response time is the amount of time it takes to complete a specific business request (BR) from start to finish. Despite the fact that the chosen architectures are service-based, the implementation style and deployment environment are completely different, and the impact of cloud can be assessed through load testing. Response time for processing the request is very fast in case of Microservices architecture compared to that of Service Oriented architecture. Whereas, the complexity of Microservices architecture is higher than Service Oriented Architecture.

[2]	Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation	In this paper, the application was implemented in four different versions, covering two different architectural styles (monolithic vs. microservices). The authors conducted a series of controlled experiments in three different deployment environments (local, Azure Spring Cloud, and Azure App Service).	This paper concludes that for a single machine the performance of monolithic is better than its microservice counterpart. Microservice architecture is not best suited for every context. For basic, lightweight systems that don't need to serve a lot of concurrent users, monolithic design appears to be a preferable choice.
[3]	A Comparative Review of Microservices and Monolithic Architectures	This article discusses the outcomes of a development environment called JHipster, which was used to create online apps using the Spring Boot and Angular JS frameworks. In order to compare the performance of monolithic and microservice architecture in various situations and testing setups, this study compares the performance of these architectures based on response time and throughput.	Under normal application traffic, microservices and monolithic applications might perform similarly. Because the monolithic application can handle requests more quickly, it can be employed when the developer specifically wants the application to accept requests more quickly. In concurrency testing, monolithic architecture outperformed microservices design by 6% in terms of throughput.
[4]	The Comparison of Microservice and Monolithic Architecture	In this paper, by sending a number of HTTP GET and POST queries, this study analyzes the monolithic and microservices architectures on several aspects such as architecture performance and response time. This study also discusses the advantages and disadvantages of monolithic and microservices architectures.	Monolithic and microservices architectures have advantages and disadvantages. Load testing has shown that the microservice architecture is more efficient when the application has to process more requests. It has many advantages that allow you to create high-quality software that is easy to scale, more reliable, and cheaper to maintain in the long term. A monolithic architecture is more efficient, has less overhead, and is easy to extend.

From the above research it is clear that processing requests using a microservices architecture is quicker than using a service-oriented design and monolithic architecture.. However, not every situation calls for it. For simple, light-weight systems that don't need to support many concurrent users, monolithic design is preferable. Monolithic architecture outperformed microservices design in concurrency testing by 6% in terms of throughput. The microservice architecture is more effective when the programme has to handle more requests, according to load testing. There are benefits and drawbacks to both monolithic and microservices systems, with monolithic architecture being more effective, having less overhead, and being simple to extend.

3. METHODOLOGY

To begin with the experiment, we first start with developing the e-commerce web application using the monolithic architecture.

We developed three models: Customers, Products and Shopping. The Customers model contains the schema of the customers, Products holds the schema of the products and Shopping holds the schema of orders and the status of the orders. All the three models store their data in the same database as various collections. After successful development, testing of the routes and database is done using Postman by sending HTTP Requests to the various routes.

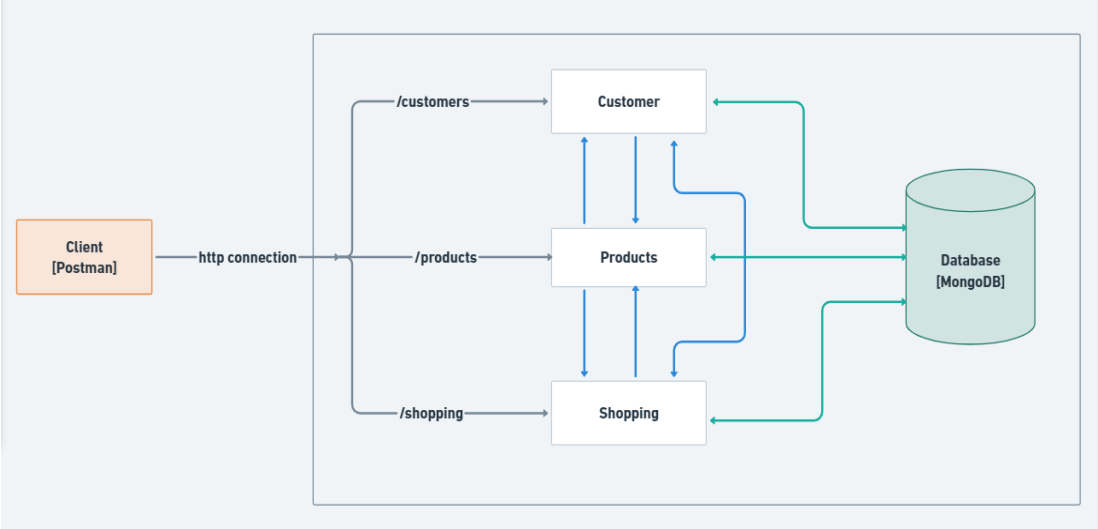

Figure 1. Monolithic Architecture

After developing the monolithic architecture, we migrate the web application to microservices architecture.

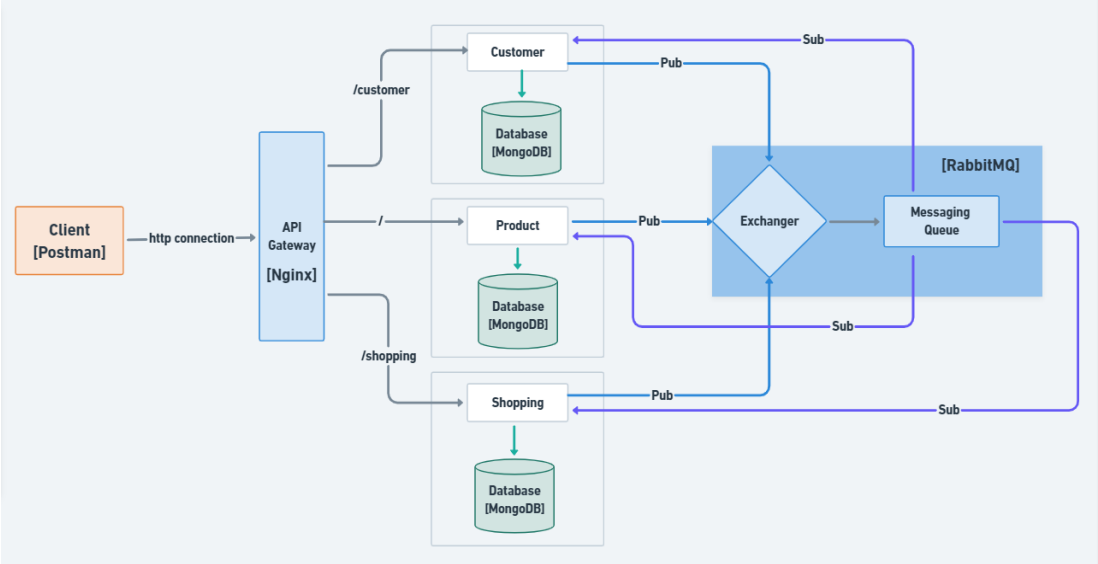

Figure 2. Microservices Architecture

Here, the 3 models are converted into independent services, i.e. the Customer Service, Products Service and Shopping Service. Each service has its own independent database to which it is connected. One of the features of microservices architecture is the ability to communicate with each other while also being independent, this was achieved by using RabbitMQ.

4. IMPLEMENTATION

To compare the two architectures, two similar web applications have to be developed using the same technologies. The two technologies used for developing the web application are NodeJS and ExpressJS. For the database, we are using MongoDB Atlas as the NoSQL cloud database. For testing, we are using Postman and for load testing and generating experimental results we are using Apache JMeter.

The rationale for choosing this technology stack was based on the factors such as speed, efficiency, scalability and flexibility. Additionally, the compatibility and supportability of this technology stack with other tools used in the experiment were considered.

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code on the server-side. It allows developers to build fast, scalable, and efficient server-side applications using JavaScript.

Express.js is a popular Node.js framework that makes it easier to build server-side web applications. It provides a robust set of features for web and mobile applications, and simplifies the process of building RESTful APIs and web applications. Furthermore, it provides features like routing, middleware, templates, and more, and makes it easier for developers to build scalable, robust, and maintainable server-side applications.

MongoDB Atlas is a fully-managed cloud database service developed by MongoDB Inc. It provides a document-based database system with a flexible schema that can be easily integrated into modern applications. It offers various features like automatic scaling, multi-cloud deployment, 24/7 support, and backup and recovery. With MongoDB Atlas, users can host their databases on cloud platforms such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure.

Postman is a popular API development and testing tool. It provides a platform for API developers to send, test, and document API requests and responses. Postman allows users to design and test APIs with ease, enabling efficient and seamless collaboration across teams.

Apache JMeter is a free and open source load testing tool that can be used to measure the performance of various services, including web applications, databases, and APIs. It allows users to simulate a heavy load on a server, network, or object to test its strength or to analyze overall performance under different load types. JMeter can generate a large number of concurrent users and send requests to the target system. The results can be analyzed and visualized to identify performance bottlenecks and provide recommendations for improvement.

RabbitMQ is open-source message broker software that implements the Advanced Message Queuing Protocol (AMQP). It allows for decoupled communication between applications by storing messages in a queue and allowing consuming applications to process them as they become available. RabbitMQ can handle high volume and high throughput of messages, and can support multiple messaging patterns including publish/subscribe, request/reply, and message routing.

All the services are connected to the Message Queue by the publish/subscribe messaging pattern. Each service has its own queue and also listens to another queue. When one service publishes its payload, the payload contains a binding key, according to the binding key the exchanger sends the payload to the specific queue where the targeted service is listening. The target service executes an event depending on the payload received. Also if the targeted service goes offline for any reason, as soon as it comes back online, it receives the all payloads from the Message Queue and quickly synchronizes itself with the other services. In this way, all the independent services are able to communicate with each other.

As shown in the architecture above (Fig. 2.2), the microservices architecture requires an API Gateway. The API Gateway forwards the HTTP Requests to the target service according to the

path of the HTTP Request. For the API Gateway, we are using Nginx to harness its ability of functioning as a reverse-proxy and its load balancing features.

Nginx is web server software that is widely used for web serving, reverse proxying, caching, and load balancing. Nginx is known for its high performance, stability, and low resource utilization, making it a popular choice for deploying high-traffic websites and applications. It can handle a large number of concurrent connections and can serve static and dynamic content efficiently. Nginx can also act as a reverse proxy, forwarding requests from clients to backend servers, and as a load balancer, distributing incoming requests across multiple servers to optimize resource utilization and increase availability.

Once the development is concluded, we can test the microservices architecture by using Postman similar to any backend system. Another important characteristic of microservices architecture is that each service runs on its own system or virtual machine. To maintain this characteristic for testing purposes we are using Docker.

Docker is an open-source platform that automates the deployment of applications inside containers. A container is a standalone executable package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers provide a consistent, reproducible, and isolated environment for applications, making it easier to develop, test, and deploy software across different environments and platforms.

For the load testing results, we are using Apache JMeter. In Apache JMeter, we are creating a thread group. In the thread group, there are three threads which are given the different paths to send a high number of HTTP Requests.

In monolithic architecture, we are sending requests to one database and accessing all the three collections i.e. Customers, Products and Shopping.

In microservices architecture, we are sending all the HTTP Requests through the API Gateway, the API Gateway routes the requests to all the three services depending on the path specified. As each service has its own database, the requests are sent to all the three databases.

5. RESULTS

For better understanding of the architectures, we are conducting load testing experiments on various amounts of HTTP Request samples starting from low to high. Load Testing was performed on the system with following specifications:

- Processor - 11th Gen Intel(R) Core(TM) i7 @ 3.40GHz
- RAM - 16.0 GB

We selected commodity or standard computing resources in our test case, to mimic the application's performance under practical industrial settings. This may not correspond to the precise outcomes, but it can offer significant insights for our analysis. The real outcomes after employing high-end computing resources could exhibit improved response time, reduced CPU utilization and enhanced results compared to our analysis.

5.1. Response Time Graph

Response time is the time taken by the request to reach the server and get a response from the server. In the graph, the Y-axis represents the response time of each request in milliseconds and the X-axis represents the HTTP Requests. In the experiment, we have configured 1000 virtual users to transmit the HTTP requests.

5.1.1. 1st Run (10,000 samples)

5.1.1.1. Monolithic

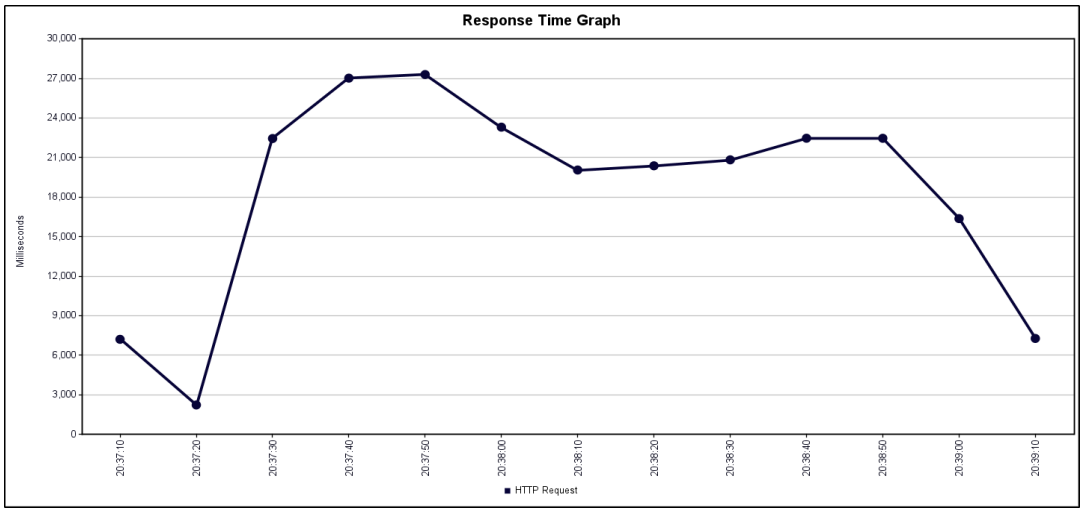

Figure 3. Monolithic Response Time Graph (1st Run)

5.1.1.2. Microservices

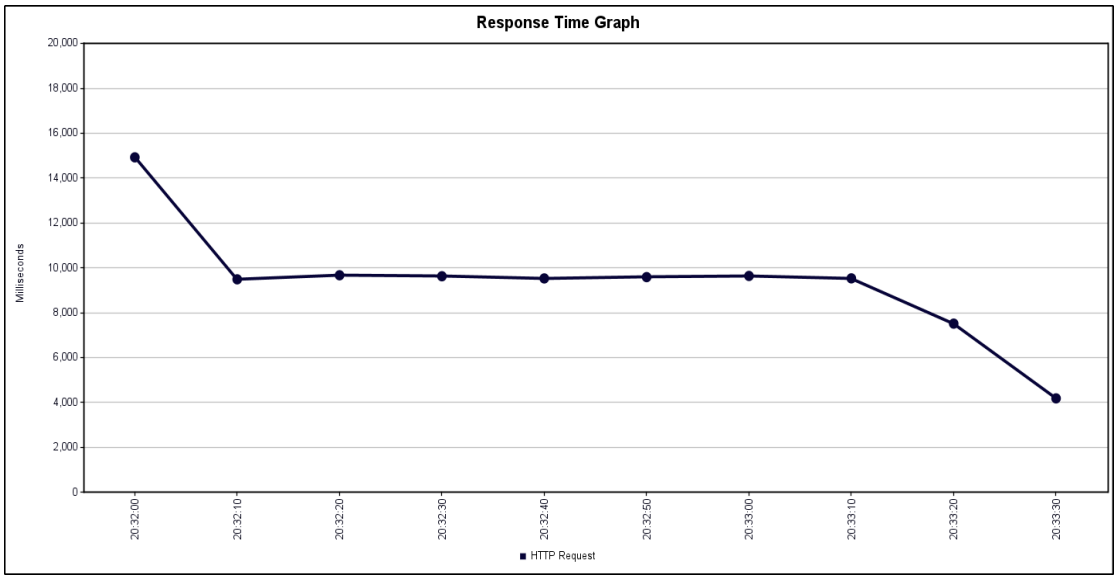

Figure 4. Microservice Response Time Graph (1st Run)

The 1st run shows the response-time graph for 10,000 samples. As it is evident in the graph, the monolithic response-time graph is non-uniform compared to the microservices architecture. However, the monolithic performs better compared to the later test runs with a higher number of samples.

From the graph analysis, we infer that monolithic architecture suffers from a high response time due to a bottleneck. To investigate this further, we decreased the traffic load on the monolithic architecture and reduced the virtual users to 100. Consequently, we observed a significant improvement in its performance compared to the previous graph.

5.1.2. 2nd Run (20,000 samples)

5.1.2.1. Monolithic

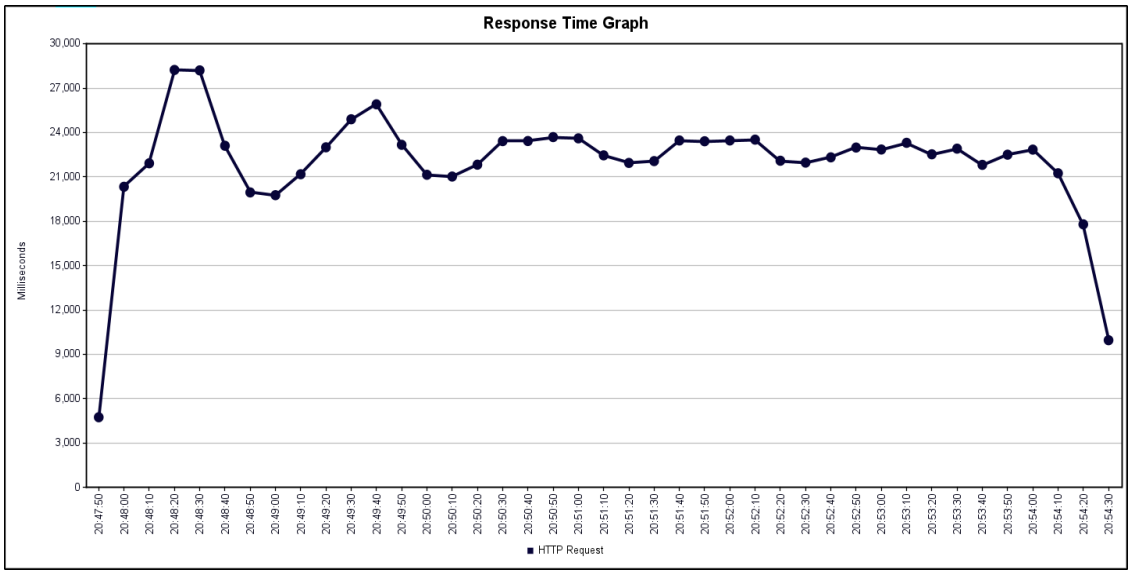

Figure 5. Monolithic Response Time Graph (2nd Run)

5.1.2.2. Microservices

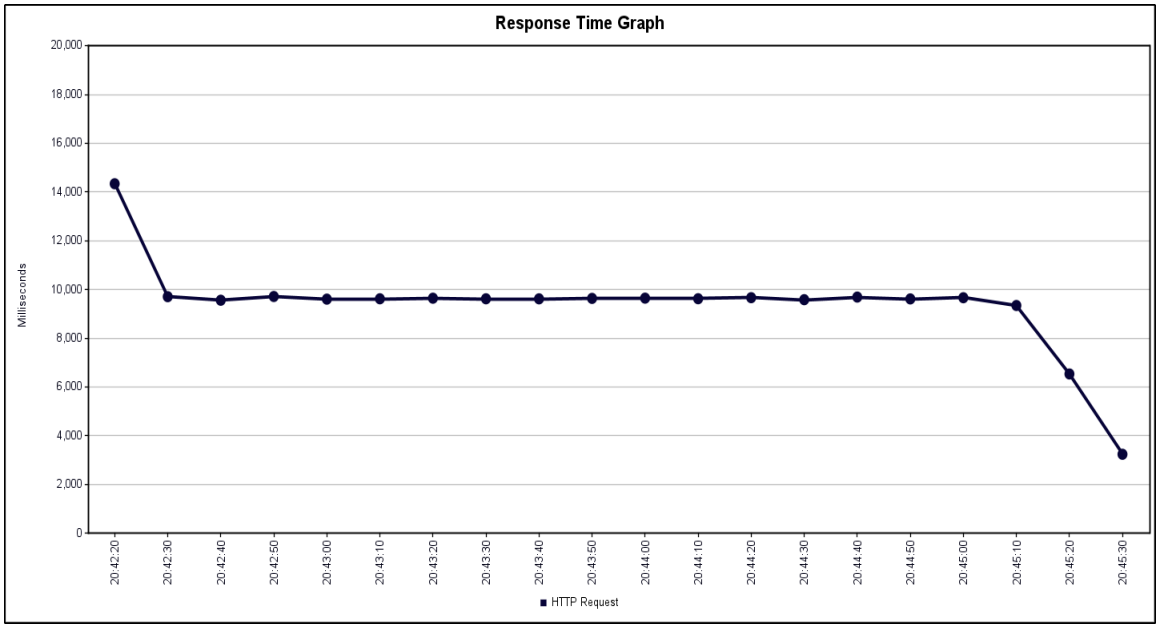

Figure 6. Microservice Response Time Graph (2nd Run)

The 2nd run shows the response-time graph of 20,000 samples. Compared to the previous monolithic graph, the monolithic graph of the 2nd run has higher deviations. Whereas, the microservice architecture graph is similar to the graph of the 1st run in spite of higher number samples. Similar to the previous graph of the monolithic architecture, the 2nd run also exhibits a bottleneck in the architecture resulting in high response time.

5.1.3. 3rd Run (30,000 samples)

5.1.3.1. Monolithic

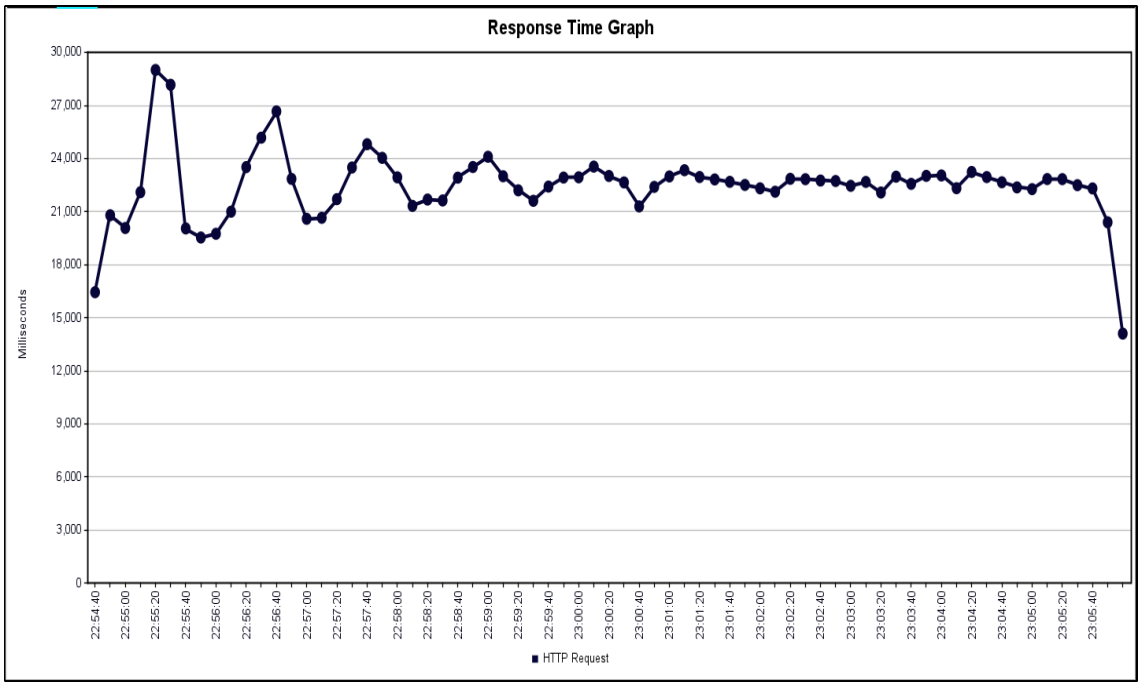

Figure 7. Monolithic Response Time Graph (3rd Run)

5.1.3.2. Microservices

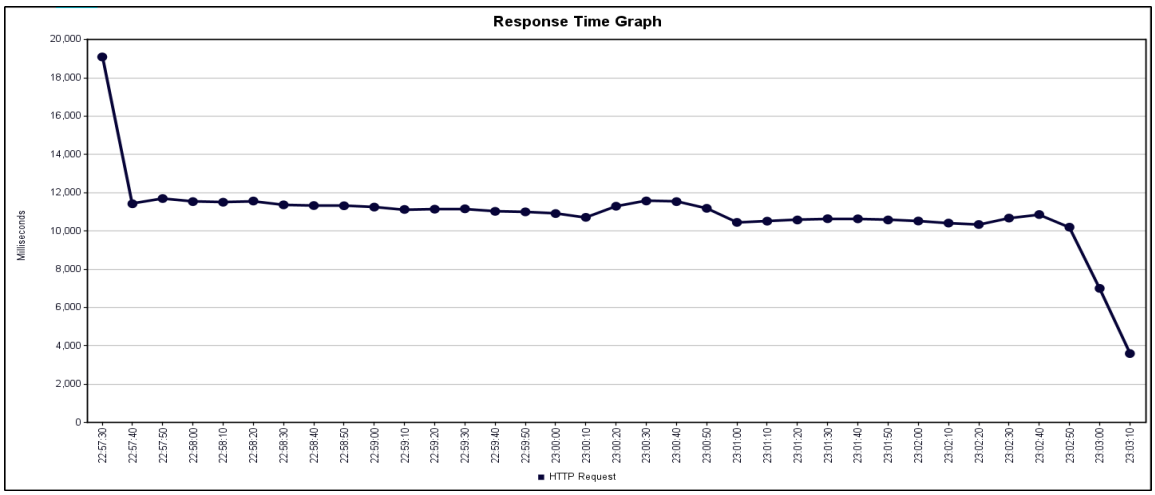

Figure 8. Microservice Response Time Graph (3rd Run)

The 3rd run shows the response-time graph for 30,000 samples. Similar to the previous runs, the monolithic architecture response-time graph continues to have a non-uniform pattern with increasing number of samples which shows its inability to handle heavy traffic and the microservices architecture's response-time graph outperforms the monolithic architecture's response-time graph by depicting a uniform graph indicating its ability to handle heavy traffic.

To conclude, the monolithic architecture's response-time graph provides unideal results as per increasing number of samples. Hence, it is logical that for the lower number of samples the monolithic architecture can give good results and can be sufficient for a web application having a few hundred users simultaneously. For the microservices architecture, the graph appears to be stable for all the tests, which indicates that microservices architecture is robust and can handle high amounts of traffic and provide good results.

5.2. Summary Report

Summary report provides insightful data for the performance testing. This report displays average, minimum and maximum response time taken by the requests in milliseconds. “Std. Dev.” row represents the deviation from the average value of response time. “Error %” represents the percentage of failed requests throughout the test duration. It is calculated as (No. of failed requests) / (Total No. of requests sent). Throughput row represents the number of requests processed by the server per second.

Table 2. Comparative Summary Report (Monolithic vs Microservices)

Label	**HTTP Request (1st Run)**		**HTTP Request (2nd Run)**		**HTTP Request (3rd Run)**	
# Samples	**10000**		**20000**		**30000**	
	Monolithic	**Microservices**	**Monolithic**	**Microservices**	**Monolithic**	**Microservices**
Average (ms)	11370	9137	19400	9247	22186	10877
Min (ms)	1	40	1	40	1	40
Max (ms)	39031	31982	56766	29226	51053	35790
Std. Dev.	11607.23	10897.00	10918.31	12458.36	8377.75	134223.80
Error %	15.03%	9.00%	51.20%	4.73%	62.01%	3.13%
Throughput	86.8/sec	108.4/sec	51.3/sec	107.7/sec	45.0/sec	91.4/sec
Received KB/sec	200.93	119.46	108.64	111.48	92.58	94.54
Sent KB/sec	5.63	12.13	5.81	12.62	5.79	10.89

In the 1st run, it is clear that for 10,000 samples microservices provide better results and the monolithic architecture results to have average performance.

The 2nd run has 20,000 samples. Compared to the previous run, the monolithic architecture began to get worse with an increasing number of samples and the microservices architecture gives approximately similar results.

The 3rd run has 30,000 samples. The monolithic architecture continues to worsen, whereas the microservices architecture provides similar results showing its ability of handling high traffic

Table 3. Average Summary Report (Monolithic vs Microservices)

Label	Average Reading	
	Monolithic	**Microservices**
Average (ms)	17652	9753.66
Min (ms)	1	40
Max (ms)	48950	32332.66
Std. Dev.	10301.09	12268.72
Error %	42.74	5.62
Throughput	61.03	102.5

From the average readings that are calculated and the results of the runs, we can conclude that the monolithic architecture gives better results for lower numbers of samples and keeps deteriorating as the number of samples goes on increasing. However, the microservices architecture keeps on giving similar results for a high number of samples. This indicates that the microservices architecture is robust and can balance heavy load which is ideal for handling thousands of users simultaneously.

The graphs show strange emissions in the monolithic architecture and it was speculated that the bottleneck in the monolithic architecture was caused by MongoDB rather than the monolithic

application itself. However, after connecting all of the services to the same cluster used by the monolithic architecture, the microservices response time graph remained unchanged. This shows that, irrespective of MongoDB clusters or multiple MongoDB clusters used in the microservices architecture, microservices architecture excels the monolithic architecture.

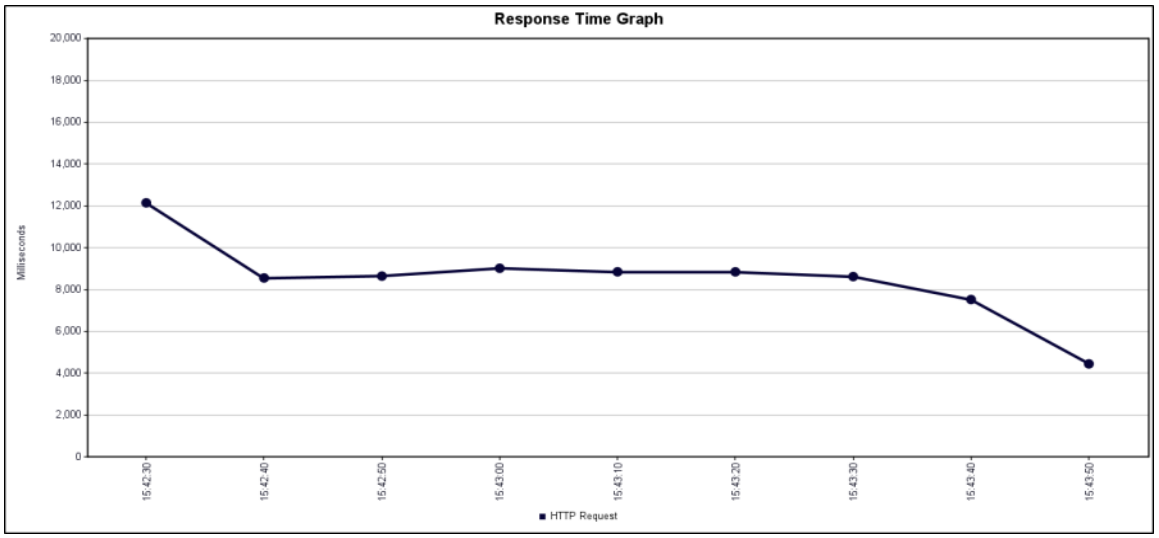

Figure 9. Microservice Response Time Graph using Single MongoDb Instance

Table 4. Microservices Summary Report using Single MongoDb Instance

Label	**HTTP Request**
# Samples	10000
Average (ms)	8051
Min (ms)	28
Max (ms)	26552
Std. Dev.	10397.31
Error %	8.97%
Throughput	123.0/sec
Received KB/sec	253.84
Sent KB/sec	13.78

To conclude, a decision should be made according to the e-commerce business. The test cases are designed for generic ecommerce web applications and do not account for any specific scenario. Moreover, the analysis and test cases may differ depending on the computing resources and hardware used. The cases are as follows:

1. If you have a small business and are having a network traffic of a few hundred users simultaneously, a backend server with monolithic architecture should suffice. This will help you to have low cost and maintenance for the e-commerce business and will also help your business sustain.

2. If you are a small business but are having or expecting to have high network traffic, you need an architecture which will help you scale easily and quickly. In that case, the microservices architecture will help you to scale your systems, allow you to make updates and also help your business to avoid loss of customers and will help your business. Combined with the services provided by various cloud platforms such as AWS, Microsoft Azure, Google Cloud Platform, your business will have a very robust and efficient e-commerce web application. If you are already having an existing system, you might have to consider migrating to the microservices architecture.

3. If you are having big business and consider having a web application which is expected to have low traffic or developing a web application for your organization for a limited geographical region such as a country, you can develop a simple backend system having the monolithic architecture. This will be sufficient for your application and you can also save money because of easy development, low system specification and low maintenance cost.
4. If you are having a big business and also having heavy traffic on your e-commerce web application such as a worldwide business or you are developing a web application for your organization and the employees from all around the world are accessing it, you should develop a system having the microservices architecture. Considering this scenario, coupling the microservices architecture with Kubernetes and other services provided by various cloud platforms such as AWS, Microsoft Azure, Google Cloud Platform, will be a great investment for the business.

6. CONCLUSION

Monolithic architecture and microservices architecture both have their pros and cons. In the research paper, we have explained the architecture, implementation, methodology, as well as the results and findings in detail. Based on these findings, we can say that the project stakeholder has to make an appropriate choice of architecture pattern based on the unique needs of the project. To assist with these choices, we have explained the various cases and the recommendation approach for that case respectively.

As it is evident that the monolithic architecture satisfies the basic requirements of a backend server compared to the microservices architecture that is complicated to develop, however, the microservices services gives better response time and low rate of errors during heavy traffic on the backend server. The Monolithic architecture results in having low minimum latency but also has a very high maximum latency during heavy traffic. The microservices architecture has a higher minimum latency compared to monolithic architecture, but can handle heavy traffic with ease. Considering the cost, the complicated architecture of the microservices architecture along with its communication, configuration and multiple devices/virtual machines makes it more expensive than the monolithic architecture.

Due to the rapid growth of information and technologies, it has led to exponential growth of not only industries, companies and organizations but also a massive number of users. To handle this growth, we need robust and high performance systems to keep up. This brought about the development of various distributed systems such as the microservices architecture in web application development. Despite all the benefits and drawbacks it has brought with it, there is still a tremendous amount of room for improvement. The microservices architecture needs technical innovations to give solutions to problems such as network complexity, security issues, etc.

REFERENCES

[1] Raj, V. and Sadam, R. (2021) ‘Performance and complexity comparison of service oriented architecture and microservices architecture’, Int. J. Communication Networks and Distributed Systems, Vol. 27, No. 1, pp.100–117

[2] G. Blinowski, A. Ojdowska and A. Przybyłek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," in IEEE Access, vol. 10, pp. 20357-20374, 2022, doi: 10.1109/ACCESS.2022.3152803

[3] O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), 2018, pp. 000149-000154, doi: 10.1109/CINTI.2018.8928192.

[4] K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2020, pp. 150-153, doi: 10.1109/MEMSTECH49584.2020.9109514.

[5] D. Kuryazov, D. Jabborov and B. Khujamuratov, "Towards Decomposing Monolithic Applications into Microservices," 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT), 2020, pp. 1-4, doi: 10.1109/AICT50176.2020.9368571.

[6] F. Ponce, G. Márquez and H. Astudillo, "Migrating from monolithic architecture to microservices: A Rapid Review," 2019 38th International Conference of the Chilean Computer Science Society (SCCC), 2019, pp. 1-7, doi: 10.1109/SCCC49216.2019.8966423.

[7] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou and Z. Li, "Microservices: architecture, container, and challenges," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020, pp. 629-635, doi: 10.1109/QRS-C51114.2020.00107
.

[8] Bogner J, Fritzsch J, Wagner S, Zimmermann A (2019) Assuring the evolvability of microservices: insights into industry practices and challenges. In: Proceedings of the 2019 IEEE international conference on software maintenance and evolution (ICSME), pp 546–556

[9] Fritzsch J, Bogner J, Wagner S, Zimmermann A (2019) Microservices migration in industry: intentions, strategies, and challenges. In: Proceedings of the 2019 IEEE international conference on software maintenance and evolution (ICSME), pp 481–490

[10] Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016) 'Microservices architecture enables devops: migration to a cloud-native architecture', IEEE Software, 18 March, Vol. 33, No. 3, pp.42–52.

[11] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," in 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017, pp. 466–475.

[12] L. Carvalho, A. Garcia, W. K. G. Assunç ao, R. de Mello, and M. J. de Lima, "Analysis of the criteria adopted in industry to extract microservices,'' in Proc. Joint 7th Int. Workshop Conducting Empirical Stud. Ind., 2019, pp. 22-29.

[13] J. Jaworski, W. Karwowski, and M. Rusek, "Microservice-based cloud application ported to unikernels: Performance comparison of different technologies," in Proc. 40th Anniversary Int. Conf. Inf. Syst. Archit. Technol., L. Borzemski, J. wi¡tek, and Z. Wilimowska, Eds. Cham, Switzerland: Springer, 2019, pp. 255-264.

[14] M. Jagie[aa]o, M. Rusek, and W. Karwowski, "Performance and resilience to failures of an cloud-based application: Monolithic and microservices based architectures compared," in Computer Information Systems and Industrial Management, K. Saeed, R. Chaki, and V. Janev, Eds. Cham, Switzerland: Springer, 2019, pp. 445-456.

[15] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, "Towards recovering the software architecture of microservice-based systems," in Proc. IEEE Int. Conf. Softw. Archit.Workshops (ICSAW), Apr. 2017, pp. 4653.