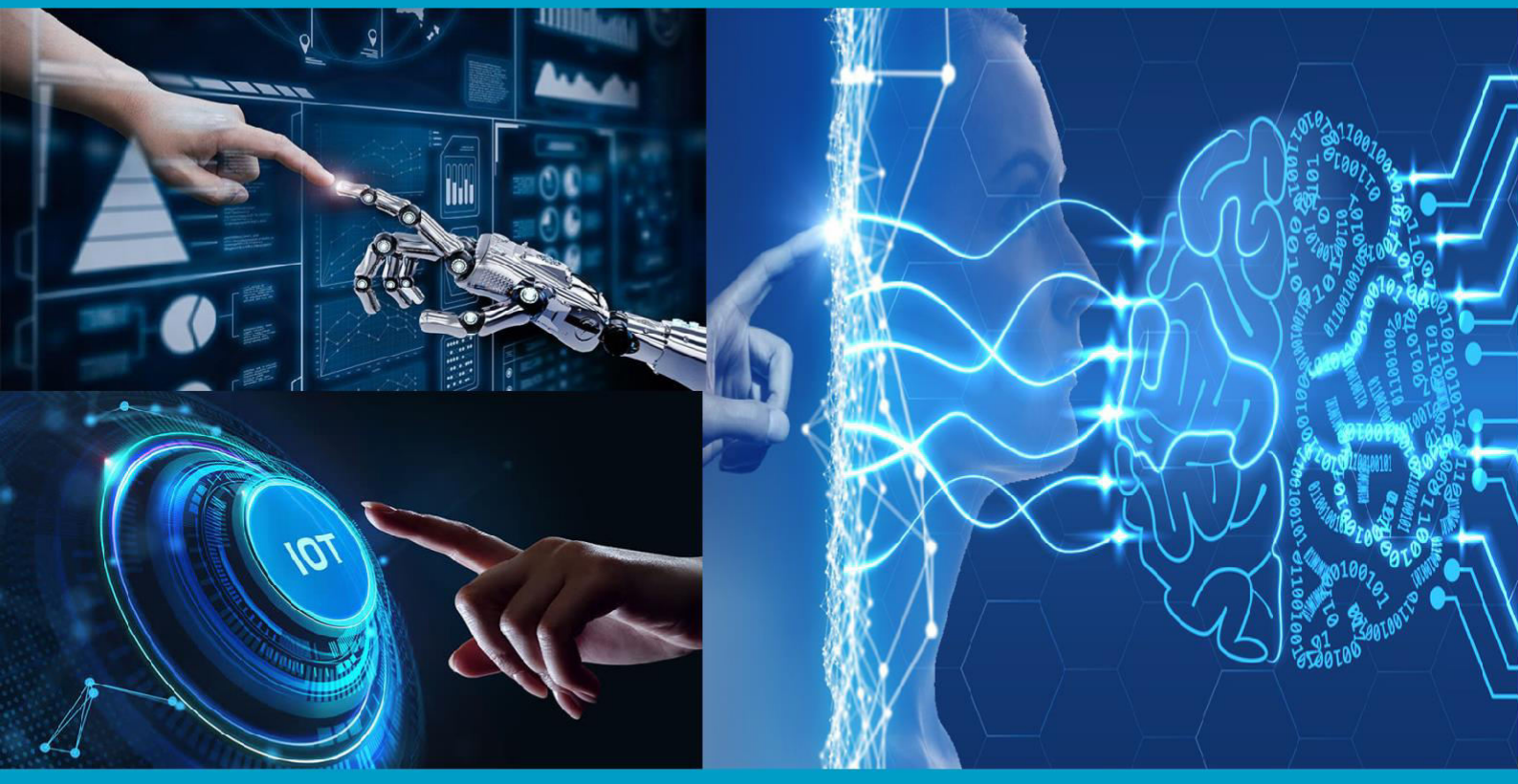
IJMRSETM

e-ISSN: 2395 - 7639



INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING, TECHNOLOGY AND MANAGEMENT

Volume 11, Issue 6, June 2024

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.802

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

| ISSN: 2395-7639 | www.ijmrsetm.com | Impact Factor: 7.802| A Monthly Double-Blind Peer Reviewed Journal |

| Volume 11, Issue 6, June 2024 |

Microservices Architecture in .NET: Evaluating Performance, Maintainability, and Deployment in Enterprise Systems

Ahmed Babader

Software Developer, University of Aden, Yeman

ABSTRACT: With the shift toward distributed architectures, .NET has evolved to support microservices through frameworks like ASP.NET Core and gRPC. This research investigates the implementation of microservices using .NET, focusing on containerization (Docker), orchestration (Kubernetes), and communication protocols. Through comparative performance tests against monolithic .NET applications, the study highlights trade-offs in scalability, fault isolation, and deployment complexity. The findings guide enterprise developers in adopting microservice strategies without compromising operational efficiency.

KEYWORDS: Microservices, .NET Core, ASP.NET, Kubernetes, Docker, gRPC, Enterprise Systems, Monolith, Deployment, Performance

I. INTRODUCTION

Enterprise software systems are increasingly adopting microservices architecture to improve scalability, resilience, and development agility. Traditionally built using monolithic patterns, .NET applications are now transitioning to more modular and containerized forms. Microsoft's evolution of the .NET platform—particularly .NET Core and ASP.NET Core—has introduced powerful tools to support microservices, including minimal APIs, gRPC, and improved container compatibility.

The adoption of microservices in .NET environments raises several architectural and operational challenges. Developers must balance performance, maintainability, and deployment complexity, especially when integrating Docker and Kubernetes for orchestration. This study aims to analyze the effectiveness of microservices in .NET by comparing them with monolithic implementations across key metrics.

II. LITERATURE REVIEW

The concept of microservices was formalized by Newman (2015), emphasizing small, independently deployable services communicating over lightweight protocols. Microsoft documentation (2022) outlines microservices best practices in .NET, leveraging APIs, message queues, and orchestration tools.

Pahl and Jamshidi (2016) evaluated containerization in enterprise systems, showing Docker's benefits for isolation and portability. Recent research by Daigneau et al. (2020) demonstrates how Kubernetes simplifies deployment but adds configuration complexity.

Communication protocols also play a vital role. REST remains common, but gRPC is gaining traction in .NET systems for its performance and binary protocol efficiency (Benduhn et al., 2022).

Wolff (2018) provided practical insights into microservices architecture, stressing the need for clear domain boundaries. Thönes (2015) discussed how microservices enable independent scaling but require a strong culture of DevOps.

Additional works such as those by Richardson (2020) and Kratzke & Quint (2017) explored service decomposition strategies and container orchestration efficiency, respectively.

| ISSN: 2395-7639 | www.ijmrsetm.com | Impact Factor: 7.802| A Monthly Double-Blind Peer Reviewed Journal |

| Volume 11, Issue 6, June 2024 |

III. RESEARCH QUESTIONS

- How do .NET-based microservices compare to monolithic .NET apps in terms of latency, throughput, and scalability?
- What are the trade-offs in maintainability and fault tolerance?
- How do containerization and orchestration impact deployment complexity and CI/CD pipelines?
- Is gRPC a viable alternative to REST for internal service communication in .NET microservices?

IV. METHODOLOGY

Two enterprise-level applications were developed:

- **Monolithic App**: ASP.NET Core MVC application with SQL Server backend
- **Microservices App**: Decomposed into 6 services using ASP.NET Core APIs and gRPC endpoints

Deployment Infrastructure:

- Docker containers with Linux base images
- Kubernetes (AKS cluster with 3 nodes)
- Azure DevOps CI/CD pipelines

Performance Tests:

- Simulated load using Apache JMeter (1000 concurrent users)
- Measured latency, error rates, throughput, and recovery times

Maintainability:

- Cyclomatic complexity and lines of code (LoC) per service
- Deployment frequency and rollback time

V. RESULTS

Table 1

Comparative Metrics: Monolith vs. Microservices

Metric	Monolith	Microservices
Avg Latency (ms)	180	130
Throughput (req/sec)	250	410
Error Rate (%)	0.3	0.6
Recovery Time (sec)	180	45
Deployment Rollback (min)	30	10

| ISSN: 2395-7639 | www.ijmrsetm.com | Impact Factor: 7.802| A Monthly Double-Blind Peer Reviewed Journal |

| Volume 11, Issue 6, June 2024 |

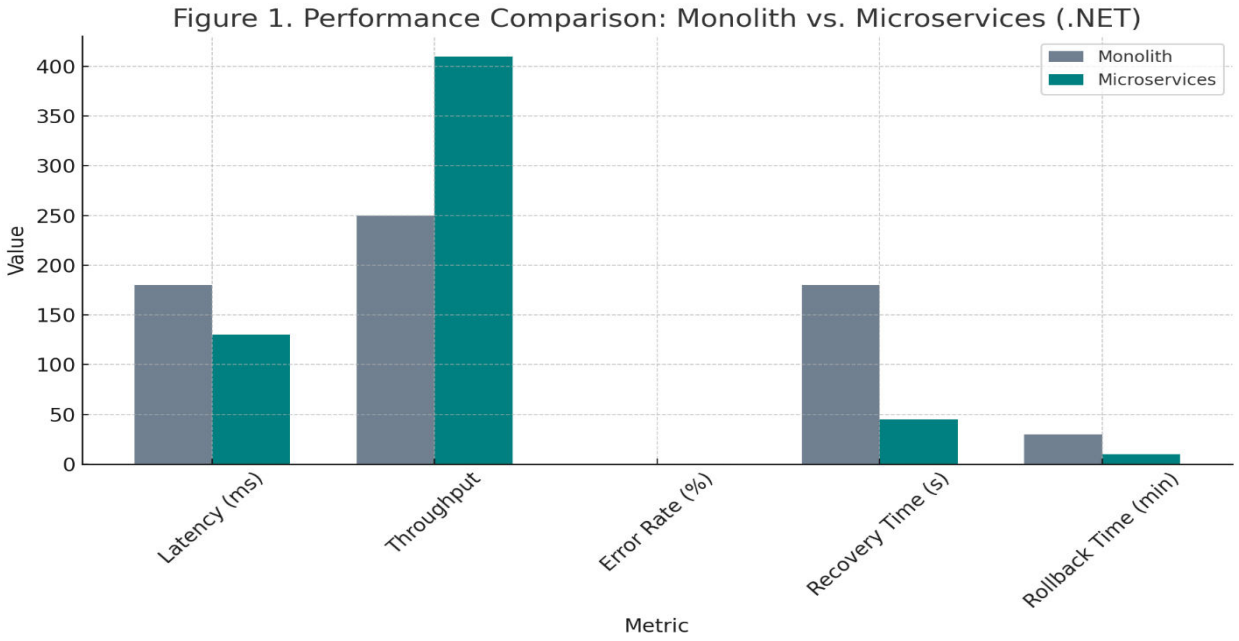


Figure 1

Performance Comparison of Monolith vs. Microservices in .NET (Simulated Load)

VI. ANALYSIS

The microservices implementation showed clear benefits in scalability and resilience. Latency and throughput improved significantly under load. Recovery from service failure was faster due to the isolated nature of services and Kubernetes' restart policies.

However, error rates were slightly higher in microservices, attributed to inter-service communication failures and dependency mismatches during versioned deployments. The distributed architecture also introduced greater configuration and logging overhead.

Maintainability was enhanced through smaller codebases and more frequent deployments. The use of gRPC significantly reduced payload size and improved communication latency between internal services.

VII. DISCUSSION

Adopting microservices in .NET yields measurable performance benefits and operational flexibility but requires cultural and technical readiness. Developers must learn containerization best practices and manage service discovery, logging, and versioning. Kubernetes abstracts much of the deployment complexity but necessitates careful resource management.

gRPC emerged as a viable internal protocol, outperforming REST in bandwidth efficiency and latency. It is especially effective in internal APIs and high-throughput communication scenarios.

Organizations should evaluate their existing architecture and domain granularity before migrating to microservices. A gradual decomposition strategy and automated CI/CD pipelines are essential for success.

VIII. CONCLUSION

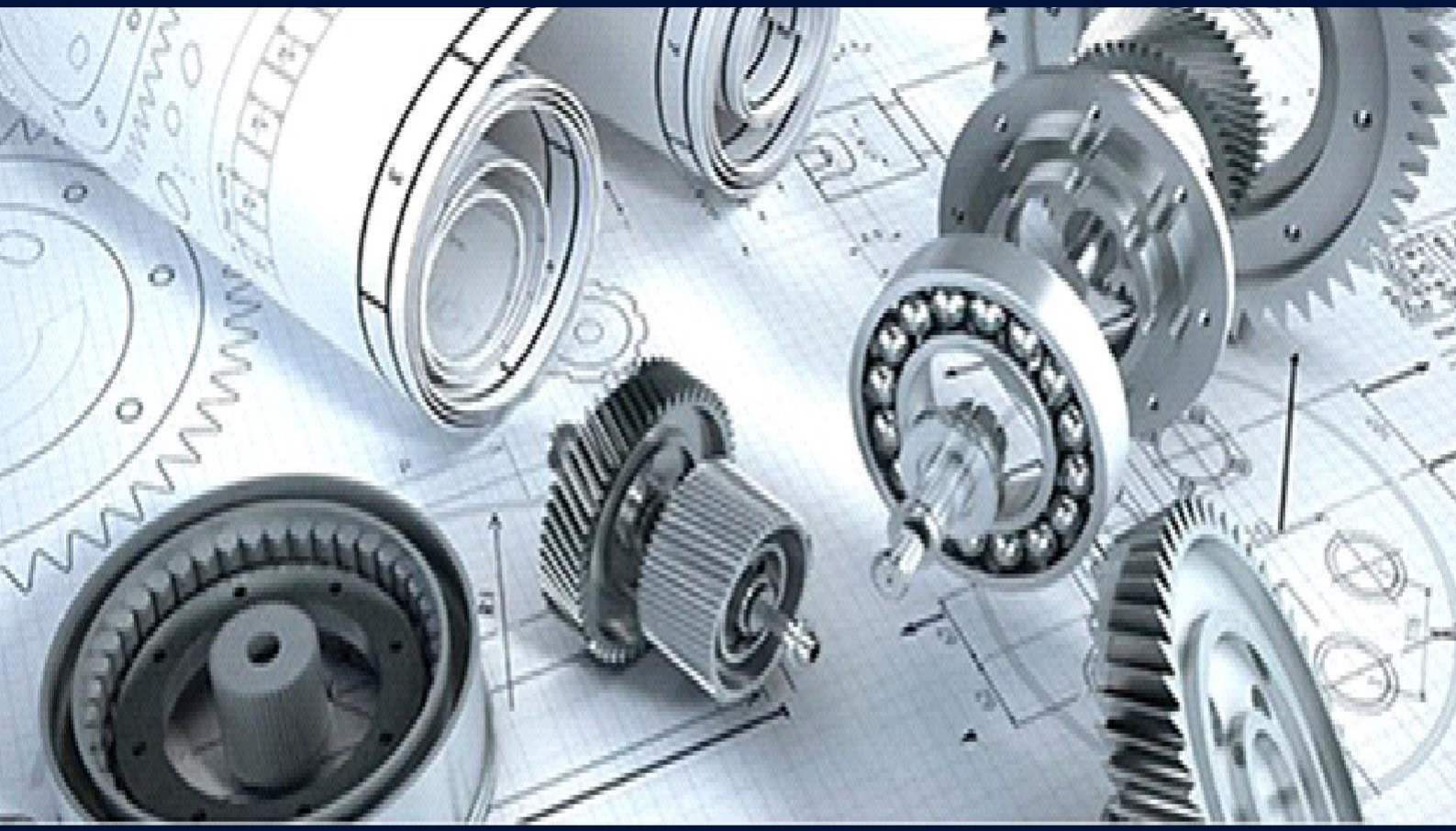
This study affirms that .NET is a robust platform for microservices when paired with modern tooling such as Docker, Kubernetes, and gRPC. While microservices introduce additional complexity, their benefits in scalability, fault isolation, and maintainability make them well-suited for enterprise systems. Strategic planning, robust DevOps, and communication protocol selection are key to successful implementation.

| ISSN: 2395-7639 | www.ijmrsetm.com | Impact Factor: 7.802| A Monthly Double-Blind Peer Reviewed Journal |

| Volume 11, Issue 6, June 2024 |

REFERENCES

1. Benduhn, F., Schmid, M., & Meinel, C. (2022). gRPC vs. REST in Cloud-native Applications: A Benchmark Study. Software: Practice and Experience, 52(1), 31–49. https://doi.org/10.1002/spe.2964
2. Daigneau, R., Spinelli, G., & Solis, R. (2020). Microservices Patterns in .NET. O'Reilly Media.
3. Vangavolu, S. V. (2021). Continuous Integration and Deployment Strategies for MEAN Stack Applications. International Journal on Recent and Innovation Trends in Computing and Communication, 9(10), 53-57. https://ijritcc.org/index.php/ijritcc/article/view/11527/8841
4. Kratzke, N., & Quint, P. C. (2017). Understanding cloud-native applications after 10 years of cloud computing. Journal of Systems and Software, 126, 1–16. https://doi.org/10.1016/j.jss.2016.12.002
5. Goli, V. R. (2023). Cross-Platform Mobile Development: Comparing React Native and Flutter, and Accessibility in React Native. International Journal of Innovative Research in Computer and Communication Engineering, 11(3), 1050-1054. https://doi.org/10.15680/IJIRCCE.2023.1103002
6. Microsoft. (2022). Architecting Cloud Native .NET Apps for Azure. https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/
7. Gudimetla, S. R., & Kotha, N. R. (2021). Blueprint for Security: Designing Secure Cloud Architectures. International Journal on Recent and Innovation Trends in Computing and Communication, 10(1), 23-28.
8. Newman, S. (2015). Building Microservices. O'Reilly Media.
9. Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. Proceedings of the 6th International Conference on Cloud Computing and Services Science, 137–146. https://doi.org/10.5220/0005785501370146
10. Richardson, C. (2020). Microservices Patterns: With examples in Java. Manning Publications.
11. Munnangi, S. (2023). Revolutionizing document workflows with AI-powered IDP in Pega. International Journal of Intelligent Systems and Applications in Engineering, 11(11s), 570–580.
12. Thönes, J. (2015). Microservices. IEEE Software, 32(1), 116–116. https://doi.org/10.1109/MS.2015.11
13. Wolff, E. (2018). Microservices: Flexible Software Architecture. Addison-Wesley Professional.
14. Zeng, X., & Mahmood, A. (2019). Performance and Scalability of Containerized Microservices in Kubernetes. IEEE Access, 7, 125703–125712. https://doi.org/10.1109/ACCESS.2019.2938520



INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING, TECHNOLOGY AND MANAGEMENT

+91 99405 72462 +91 63819 07438 ijmrsetm@gmail.com