Restructuring Software Architecture: Moving From Monoliths to Microservices

Darshana Dadaji Ahire

[1]*darshuahire@gmail.com, Student of D. Y. Patil College of Engineering Akurdi, Pune – 411044, india*
Internal Guide Dr. Dipalee D. Rane

Abstract - The shift from monolithic software architectures to microservices has become a key approach in contemporary software development, offering improvements in scalability, flexibility, and maintainability. This transformation addresses the limitations of tightly integrated systems, such as reduced agility and challenges in scaling individual components. In contrast, microservices advocate for a decentralized model, where independent services communicate via lightweight protocols, such as REST or message queues. This paper explores the key reasons for adopting microservices, including the ability to support rapid deployment cycles, enhance fault isolation, and optimize resource utilization. It delves into the core principles of microservices architecture, such as domain-driven design, bounded contexts, and continuous delivery. The paper also addresses the technical and organizational hurdles of migrating to microservices, including issues like data consistency, greater operational complexity, and the need for comprehensive monitoring and logging. It presents practical approaches for transitioning from a monolithic to a microservices-based system, such as incremental decomposition, implementing API gateways, and utilizing containerization technologies. The conclusion emphasizes the importance of aligning organizational structures with the new architectural approach, as highlighted by Conway's Law, to fully realize the advantages of this transformation.

1. INTRODUCTION

In the fast-evolving software development landscape, organizations are increasingly seeking to update their software architectures to stay ahead in the market. A major trend in software architecture is the transition from monolithic to microservices-based systems. This project investigates the process of migrating from monolithic architecture to microservices, using a practical example that includes User Service, Customer Service, and Product Service.

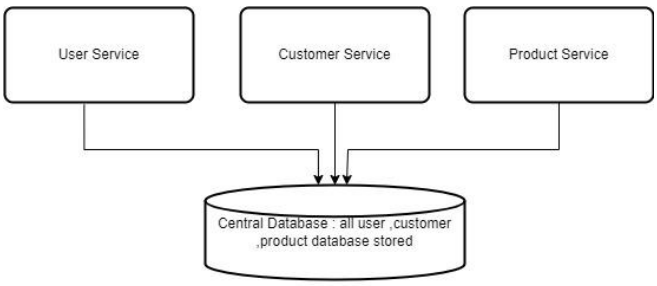
Monolithic Architecture

In a monolithic architecture, all the components of an application are closely integrated and packaged together as a single unit. As a result, the entire system must be redeployed whenever changes are made to any part of the application. Let's take an example of a monolithic e-commerce platform where functionalities related to users, customers, and products are tightly interconnected.

Example:

• User Service: Manages user authentication, registration, and profile management.
• Customer Service: Oversees customer-related data, such as addresses, payment methods, and order history.
• Product Service: Handles the product catalog, including tasks like adding new products, updating product details, and managing inventory.

In this monolithic setup, all three services are tightly coupled, which makes it challenging to scale individual components independently.



Monolithic E-commerce Application

Microservices Architecture

In a microservices architecture, the application is divided into smaller, self-contained services that can be developed, deployed, and scaled independently. Each service focuses on a specific business function and communicates with others through APIs, usually over HTTP [3].

For the e-commerce example, we can split the User Service, Customer Service, and Product Service into separate microservices, each having its own database.

Example Breakdown:

• User Service: Handles user authentication, registration, and profile management, interacting with its dedicated user database.
• Customer Service: Manages customer data, including addresses and payment methods, and operates independently from the User Service. It communicates with the User Service via an API when necessary.
• Product Service: Manages product information, including adding new products, updating product details, and controlling inventory, with its own dedicated database.

With this architecture, each service can be scaled independently to accommodate increased demand in its specific area, without impacting other services.

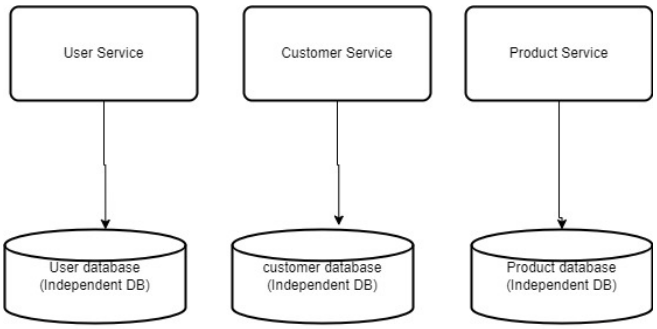


Fig 2 : Microservices-Based E-commerce Application

2. LITERATURE SURVEY

1. A Survey on Microservices Decomposition, Smith et al. 2020

Microservices architecture has become a popular approach for modernizing legacy applications, with many organizations adopting it. However, while there is significant research on the migration process, there remains a gap in the understanding of the principles that should guide the implementation of a microservices architecture. This study offers a comprehensive survey that gathers existing literature exploring the core principles behind object-oriented approaches and their connection to both monolithic and microservices architectures.

Our research includes an examination of both monolithic and microservices architectures, with a focus on the design patterns and principles used within microservices. We contribute by presenting a list of patterns commonly applied in microservices architecture and comparing the principles advocated by experts in microservices decomposition, such as Martin Fowler and Sam Neuman, with the foundational ideas of David Parnas, who introduced the Principle of Information Hiding and discussed modularization as a way to enhance flexibility and understanding of a system [2].
Furthermore, we summarize the advantages and disadvantages of both monolithic and microservices architectures based on our literature review, providing a useful reference for researchers in academia and industry. Lastly, we highlight the current trends in microservices architectures.

2. From Monolith to Microservices: A Systematic Approach, Johnson et al.,2019

Although the Microservices architectural style has gained significant attention in academic literature, there is limited guidance on how to refactor legacy applications. This is an important area of study due to the high costs and efforts involved in the refactoring process, which also affects broader aspects such as organizational processes (e.g., DevOps) and team structures. Software architects facing this challenge must carefully select an appropriate strategy and refactoring technique. One crucial aspect of this decision is determining the appropriate level of service granularity to fully leverage the benefits of a Microservices architecture [7].
This study begins by exploring the concept of architectural refactoring and then compares 10 different refactoring approaches proposed in recent academic literature. These approaches are categorized based on their underlying decomposition techniques and are presented in a visual decision guide for quick reference. The review identifies a variety of strategies for decomposing a monolithic application into independent services. However, with one exception, most of these approaches are only suitable under specific circumstances. Further challenges include the substantial amount of input data required by some methods and the limited or experimental tool support available [6].

3. Benchmarks and performance metrics for assessing the migration to microservice-based architectures, Nichlas Bjørnda

The migration from monolithic systems to microservice-based architectures has gained significant popularity over

the past decade. However, the benefits of such a migration have not been thoroughly explored in the literature, to the best of the authors' knowledge. This paper aims to introduce a methodology and performance indicators that can help assess whether migrating from a monolithic to a microservice-based architecture is advantageous.

A systematic review was conducted to identify the most relevant performance metrics in existing literature, which was then validated through a survey with industry professionals. Subsequently, a set of metrics, including latency, throughput, scalability, CPU usage, memory usage, and network utilization, was used in two experiments to compare monolithic and microservice versions of the same system. The findings presented in this paper contribute to the body of knowledge on benchmarking various software architectures. Additionally, this study demonstrates how the identified metrics can be used to more accurately evaluate both monolithic and microservice-based systems [8].

3. SYSTEM ARCHITECTURE

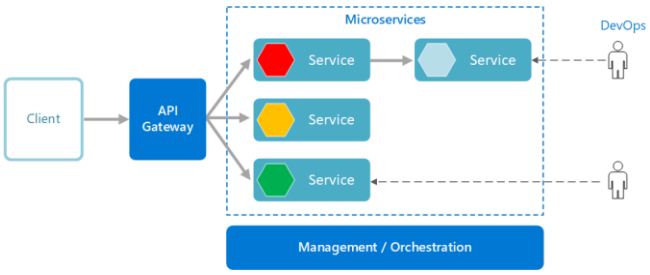


Fig Architecture

A microservices architecture is composed of a collection of small, independent services that operate autonomously. Each service is designed to be self-contained, focusing on delivering a specific business function within a clearly defined bounded context. A bounded context defines a logical boundary within an organization, indicating where a specific domain model is relevant.

Microservices are built to be small, independent, and loosely coupled. A small development team can efficiently create and manage each service, with each service having its own distinct codebase. This design enables services to be deployed independently, allowing updates to a service without requiring a complete rebuild or redeployment of the entire application. Each service manages its own data or external state, unlike traditional architectures that rely on a centralized data layer for persistence. Communication between services happens through well-defined APIs, ensuring that the internal workings of each service remain hidden from others. This architecture also supports polyglot programming, allowing services to use different technologies, libraries, or frameworks [4][,5].

The management and orchestration of these services are handled by a component responsible for placing services on appropriate nodes, detecting failures, and rebalancing services when needed. This management is often facilitated by established technologies, such as Kubernetes, rather than being custom-built.

The API Gateway serves as the primary entry point for clients. Rather than directly interacting with individual services, clients communicate with the API Gateway, which routes the requests to the appropriate backend services [9].

4. ALGORITHMS

Algorithms for three microservices.

1. Customer Service: Manages customer details (registration, login, profile management, etc.)
2. Product Service: Manages product catalog (listing, searching, updating, etc.)
3. User Service: Manages user-related functionalities (user authentication, roles, permissions, etc.)

Each of these services will have its own set of algorithms, usually exposed via REST APIs, and these microservices can interact with each other in a decoupled manner. The focus of each algorithm is to define the high-level logic for handling typical business processes in each service.

4.1. Customer Service Algorithm

Description: This service handles customer data, including customer registration, profile management, and fetching customer information.

Algorithm for Customer Registration

1.Input: Customer information (e.g., name, email, password, address).

2.Check if Email Already Exists:

Query the database to check if the customer email already exists. If an email exists, return an error response (400 Bad Request) with a message saying, "Email already in use".

3.Validate Customer Data:

Validate email format. Ensure the password is strong enough (minimum length, contains uppercase, special characters, etc.). Ensure required fields are not empty.

4.Hash the Password:

Use a hashing algorithm (e.g., bcrypt) to securely hash the password.

5.Store Customer Data:

Insert the customer details (name, email, hashed password, address) into the database. Generate a unique customer ID and store it.
6.Send Welcome Email (Optional):
Send a confirmation/welcome email to the customer.
7.Output: Return a success response with status 201 Created and a message "Customer registered successfully".

4.2 Product Service Algorithm
Description: This service handles product listings, adding, updating, and deleting products from the catalog.
Algorithm for Adding a Product
1.Input: Product details (name, description, price, stock, etc.).
2 Validate Product Data:
Ensure that the product name and price are valid. Ensure that the required fields (name, price, description) are not empty.
3.Add Product to Database:
Insert the product details into the product catalog database.
4.Output: Return a success response (201 Created) with the message "Product added successfully".

Algorithm for Fetching Products
1.Input: Optional search criteria (e.g., product name, category).
2.Search Products:
If search criteria are provided, query the product database for products matching the criteria (e.g., by name or category). If no search criteria are provided, return all products.
3.Output: Return to a list of products.

4.3. User Service Algorithm
Description: This service handles user authentication, authorization, and management of user roles.
Algorithm for User Login
1.Input: Username/email and password.
2.Check User Credentials:
Query the database to find a user with the provided username/email. If the user is found, compare the provided password with the stored hashed password. If credentials are incorrect, return an error response (401 Unauthorized).
3.Generate Authentication Token:
If login is successful, generate an authentication token (JWT, session token) for the user.
4.Output: Return the token in the response (200 OK).

These are the basic algorithms that could be implemented within the respective microservices. In a production environment, these services would be designed to handle security, scalability, and fault tolerance, and could be enhanced with features such as rate limiting, logging, and more sophisticated error handling.

5. CONCLUSION

Defining the feature scope in microservices is crucial for the success of microservices architecture. It ensures that each service remains focused, manageable, and aligned with business goals. By following the principles outlined above, teams can effectively design and implement microservices that provide robust and scalable functionalities. Microservice components collectively support the principles of microservices architecture, promoting independence, scalability, resilience, and agility in application development and deployment

7. REFERENCES

[1] N. Dragoni et al., “Microservices: Yesterday, today, and tomorrow,” in Present and Ulterior Software Engineering, M. Mazzara and B. Meyer, Eds., Cham, Switzerland: Springer, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978--3-319-67425-4_12

[2] M. Kalske, N. Mäkitalo, and T. Mikkonen, “Challenges when moving from monolith to microservice architecture,” in Proc. Int. Conf. Web Eng., I. Garrigós and M. Wimmer, Eds., Cham, Switzerland: Springer, 2018, pp. 32–47. [Online]. Available: https://doi.org/10.1007/978--3-319-74433-9_3

[3] N. C. Mendonça, C. Box, C. Manolache, and L. Ryan, “The monolith strikes back: Why Istio migrated from microservices to a monolithic architecture,” IEEE Softw., vol. 38, no. 5, pp. 17–22, Sep./Oct. 2021. [Online]. Available: https://doi.org/10.1109/MS.2021.3080335

[4] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, “Microservices migration patterns,” Softw.: Pract. Experience, vol. 48, no. 11, pp. 2019–2042, 2018. [Online]. Available: https://doi.org/10.1002/spe.2608

[5] D. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, motivations, and issues for migrating to microservices

architectures: An empirical investigation," IEEE Cloud Comput., vol. 4, no. 5, pp. 22–32, Sep./Oct. 2017. [Online]. Available: https://doi.org/10.1109/MCC.2017.4250931

[6] M. Ahmadvand and A. Ibrahim, "Requirements reconciliation for scalable and secure microservice (de)composition," in Proc. IEEE 24th Int. Requirements Eng. Conf. Workshops, Los Alamitos, CA, USA, 2016, pp. 68–73. [Online]. Available: https://doi.ieeecomputersociety.org/10. 1109/REW.2016.026

[7] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: An assessment framework," Inf. Softw. Technol., vol. 137, 2021, Art. no. 106600. [Online]. Available: https://doi.org/10. 1016/j.infsof.2021.106600

[8] P.Clarke,R.V.O'Connor, and B. Leavy, "A complexity theory viewpoint on the software development process and situational context," in Proc. Int. Conf. Softw. Syst. Process, New York, NY, USA, 2016, pp. 86–90.
[Online]. Available: https://doi.org/10.1145/2904354.2904369

[9] O. Zimmermann, "Microservices tenets," Comput. Sci. Res. Develop., vol. 32, no. 3, pp. 301–310, Jul. 2017. [Online]. Available: https://doi. org/10.1007/s00450--016-0337-0

[10] S. Newman, Building Microservices, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.