**EVALUATE THE IMPACT OF MICROFRONTENDS ON CLOUD-BASED APPLICATION ARCHITECTURE**

**MAI CHUNG TUAN**

**INTERIM REPORT**

**MAY 2024**

# **DEDICATION**

This thesis work is dedicated to my wife, Quynh Nguyen, who has been a constant source of support and encouragement during the challenges of graduate school and life. I am truly thankful for having you in my life. This work is also dedicated to my parents who have always loved me unconditionally and whose good examples have taught me to work hard for the things that I aspire to achieve.

# **ACKNOWLEDGEMENTS**

I would like to thank Upgrad and LJMU for giving me the chance to do this thesis work for them, and especially, big thanks to my supervisor at LJMU, Dr. Ehsan Rana, for her constructive feedback and unwavering guidance throughout the thesis process.

I must also extend my sincere appreciation to all the interviewees and participants who generously shared their time and invaluable insights during this research. Their contributions form the foundation of this thesis, and I am truly grateful for their perspectives.

Lastly, thank you to all who have played a role, large or small, in making this endeavor possible.

# **ABSTRACT**

This study investigates the impact of applying micro-frontend development to cloud-based applications. It will explore the impact of micro-frontend to the scalability, cost-effectiveness, and securing of cloud-based applications. The primary objectives are structured around comprehensive case studies, performance evaluations, and the formulation of best practices. Through these objectives, the research seeks to provide actionable guidelines and insights for effectively integrating micro-frontends within cloud-based applications. Real-world implementations of micro-frontends will be examined to gain practical insights, performance assessments will be conducted in simulated cloud environments, and best practices will be developed for seamless integration. The scope of the study is broad, encompassing a thorough examination of how micro frontend development influences the scalability, cost-effectiveness, and security of cloud-based application architectures. Employing a multifaceted approach, including case studies and performance evaluations, the study aims to address key aspects of this integration. By considering a variety of real-world scenarios and diverse cloud platforms, the research aims to provide a holistic understanding of the challenges and opportunities associated with the integration of micro-frontends into cloud-based applications. The methodologies, case studies, and analyses detailed in subsequent sections contribute to the overarching goal of offering valuable insights to the dynamic landscape of web application development, enabling organizations to make informed architectural decisions and achieve optimal outcomes.

***Keywords:******Security, systematic mapping, micro-frontends, cloud-based application***

# **LIST OF ABBREVIATIONS**

AWS Amazon Web Service

MA Monolith architecture

MFE Micro frontend

MFA Micro-frontend architecture

MSA Microservice architecture

GCP Google Cloud Platform

SaaS Software as a Service

IaaS Infrastructure as a Service

PaaS Platform as a Service

CI Continuous Integration

CD Continuous Deployment

# **LIST OF FIGURES**

Figure. Monolith architecture: ...

Figure. Micro service architecture: ...

Figure. Micro frontend architecture: ...

Figure. E2E micro-frontend architecture: ...

Figure. Overview systematic mapping: ...

# **LIST OF TABLES**

Table. Research questions and motivation: ...

Table. Data extraction: ...

Table. Number of studies returned by each repository: ...

Table. Inclusion criteria: ...

Table. Exclusion criteria: ...

Table. Quality assessment criteria: ...

Table. MFA security threats per category: ...

Table. MFA security mechanism per category: ...

Table. Distribution of proposed solutions by application layers: ...

Table. MFA security targets multiple platforms: ...

Table. Verification and validation methods: ...

# **LIST OF DIAGRAMS**

Diagram 1.1: ... 21

[**DEDICATION** i](#_Toc168705069)

[**ACKNOWLEDGEMENTS** ii](#_Toc168705070)

[**ABSTRACT** iii](#_Toc168705071)

[**LIST OF ABBREVIATIONS** iv](#_Toc168705072)

[**LIST OF FIGURES** v](#_Toc168705073)

[**LIST OF TABLES** vi](#_Toc168705074)

[**LIST OF DIAGRAMS** vii](#_Toc168705075)

[**: INTRODUCTION** 1](#_Toc168705076)

[**1.1** **Background** 1](#_Toc168705077)

[**1.2** **Problem statement** 1](#_Toc168705078)

[**1.3** **Significance of the Study** 2](#_Toc168705079)

[**1.4** **Scope of the Study** 3](#_Toc168705080)

[**1.5** **Aim of the study** 3](#_Toc168705081)

[**: RELATED WORK** 5](#_Toc168705082)

[**2.1.** **Background of Microservice** 5](#_Toc168705083)

[**2.2.** **Micro-frontends architecture** 6](#_Toc168705084)

[**2.3.** **Cloud-based architecture with Micro-frontends [Need more details]** 9](#_Toc168705085)

[**2.4.** **Scalability in cloud-based applications** 10](#_Toc168705086)

[**2.5.** **Security in cloud-based applications** 11](#_Toc168705087)

[**2.5.1** **Definition** 11](#_Toc168705088)

[**2.5.2** **Systematic mapping** 11](#_Toc168705089)

[**2.6.** **Cost-effectiveness in cloud-based applications** 13](#_Toc168705090)

[**: RESEARCH METHODS** 15](#_Toc168705091)

[**3.1 Scalability for Micro-frontends** 15](#_Toc168705092)

[**3.2 Security tactics model for MFA** 15](#_Toc168705093)

[**3.2.1 Research objectives** 15](#_Toc168705094)

[**3.2.2 Research questions** 16](#_Toc168705095)

[**3.2.3 Search process** 17](#_Toc168705096)

[**3.2.4 Study selection process** 17](#_Toc168705097)

[**3.2.5 Inclusion and exclusion criteria** 18](#_Toc168705098)

[**3.2.6 Quality assessment** 19](#_Toc168705099)

[**3.2.7 Data extraction process** 20](#_Toc168705100)

[**3.2.8 Data synthesis** 21](#_Toc168705101)

[**3.3 Cost-effectiveness model for Micro-frontends** 21](#_Toc168705102)

[**3.3.1 Data Collection of cost-effective for Micro-frontends** 21](#_Toc168705103)

[**3.3.2 The Proposed CEA policy** 22](#_Toc168705104)

[**: ANALYSIS** 24](#_Toc168705105)

[**4.1 Scalability research** 24](#_Toc168705106)

[**4.2 Security research** 24](#_Toc168705107)

[**: RESEARCH RESULT** 25](#_Toc168705108)

[**: CONCLUSION AND FUTURE WORK** 26](#_Toc168705109)

[**REFERENCES** 26](#_Toc168705110)

[**DEDICATION** i](#_Toc167010550)

[**ACKNOWLEDGEMENTS** ii](#_Toc167010551)

[**ABSTRACT** iii](#_Toc167010552)

[**LIST OF ABBREVIATIONS** iv](#_Toc167010553)

[**LIST OF FIGURES** v](#_Toc167010554)

[**LIST OF TABLES** vi](#_Toc167010555)

[**LIST OF DIAGRAMS** vii](#_Toc167010556)

[**TABLE OF CONTENTS** viii](#_Toc167010557)

[**Chapter 1 : INTRODUCTION** 1](#_Toc167010558)

[**1.1** **Background** 1](#_Toc167010559)

[**1.2** **Problem statement** 1](#_Toc167010560)

[**1.3** **Significance of the Study** 2](#_Toc167010561)

[**1.4** **Scope of the Study** 2](#_Toc167010562)

[**1.5** **Aim and objective of the study** 2](#_Toc167010563)

[**Chapter 2 : RELATED WORK** 4](#_Toc167010564)

[**2.1** **Basic knowledgement** 4](#_Toc167010565)

[**2.1.1** **Definition of micro-frontends** 4](#_Toc167010566)

[**2.1.1.1** **What is micro-frontend?** 4](#_Toc167010567)

[**2.1.1.2** **How do micro-frontends work?** 5](#_Toc167010568)

[**2.1.1.3** **Types of micro-frontends?** 5](#_Toc167010569)

[**2.1.1.4** **Micro-frontends architecture** 6](#_Toc167010570)

[**2.1.2** **Definition of cloud-based architecture** 6](#_Toc167010571)

[**2.1.3** **Scalability in cloud-based applications** 7](#_Toc167010572)

[**2.1.3.1** **Vertical scaling in cloud computing** 7](#_Toc167010573)

[**2.1.3.2** **Horizontal scaling in cloud computing** 8](#_Toc167010574)

[**2.1.4** **Security in cloud-based applications** 8](#_Toc167010575)

[**2.1.5** **Cost-effectiveness in cloud-based applications** 8](#_Toc167010576)

[**Chapter 3 : RESEARCH METHOD** 9](#_Toc167010577)

[**3.1** **Data Collection and Analysis of Security Tactics for Microservices** 9](#_Toc167010578)

[**3.2** **Case Study Preparation and Development of Model Data Set** 9](#_Toc167010579)

[**3.3** **Definition of ADDs and Metrics** 9](#_Toc167010580)

[**3.4** **Case Study-Based Analysis and Evaluation** 10](#_Toc167010581)

[**3.5** **Model Generation** 10](#_Toc167010582)

# **: INTRODUCTION**

## **Background**

[Fix citation]

Frontend development has undergone significant transformations over the past few decades. Initially, web applications were monolithic, meaning that both the frontend and backend were tightly coupled and deployed as a single unit. This monolithic architecture often resulted in several challenges, including difficulty in scaling, slow deployment times, and a lack of flexibility in adopting new technologies (Martin Fowler, 2015).

In the early days of web development, applications were typically built using a monolithic approach. All the code, including HTML, CSS, and JavaScript, was bundled together and served from a single server. This setup was simple and easy to manage for small-scale applications. However, as applications grew in complexity and size, the limitations of monolithic architecture became apparent. Any change in one part of the application required redeployment of the entire codebase, leading to longer development cycles and increased risk of introducing bugs (Cam Jackson, 2019).

To address the challenges posed by monolithic architectures, the industry began adopting microservices for backend development. This approach inspired a similar shift in frontend development, leading to the emergence of micro-frontends (MFEs). Micro-frontends apply the principles of microservices to the frontend, breaking down the user interface into smaller, more manageable pieces that can be developed, deployed, and maintained independently (Leitner et al., 2020).

MFEs enable teams to work on different parts of the application simultaneously without interfering with each other's work. This modular approach enhances scalability and flexibility, allowing developers to use different frameworks and technologies within the same application. It also facilitates easier testing and maintenance, as each micro-frontend can be tested in isolation (Gioia et al., 2019).

Cloud computing has revolutionized the way applications are developed and deployed. By leveraging cloud services, organizations can avoid the upfront costs and complexity of owning and maintaining physical servers. Instead, they can rent computing resources on-demand from cloud providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) (Armbrust et al., 2010).

Cloud computing offers various deployment models, including Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Each model provides different levels of control and management over the infrastructure:

* SaaS delivers fully managed applications over the internet, removing the need for users to handle the underlying infrastructure.
* PaaS provides a platform for developing, testing, and deploying applications, allowing developers to focus on coding without worrying about infrastructure management.
* IaaS offers virtualized computing resources over the internet, giving users maximum control over their infrastructure and the ability to configure it according to their needs (Mell & Grance, 2011).

The combination of micro-frontends and cloud-based architecture provides a powerful synergy that enhances the development and deployment process. Cloud platforms offer the scalability and flexibility required to support micro-frontend architectures (MFAs). By hosting micro-frontends in the cloud, organizations can take advantage of automated scaling, continuous integration and deployment (CI/CD) pipelines, and robust monitoring and logging tools (Hashizume et al., 2013).

Micro-frontends benefit from cloud services by enabling independent deployment and scaling of individual frontend components. This means that different teams can deploy updates to their respective micro-frontends without affecting the entire application. Cloud-based deployment models, particularly PaaS and IaaS, are well-suited to support this level of modularity and independence (Leitner et al., 2020).

Moreover, cloud services provide essential tools for managing the complexity of micro-frontend architectures. For example, containerization technologies like Docker and orchestration tools like Kubernetes facilitate the deployment and management of micro-frontends in a distributed environment. These tools ensure that micro-frontends can be easily scaled and maintained, providing a robust foundation for modern web applications (Burns & Oppenheimer, 2016).

## **Problem statement**

[Fix citation]

Micro-frontend architecture offers several benefits, such as improved modularity, scalability, reusability, flexibility, and alignment with agile development methodologies. However, implementing this architecture also presents a range of challenges that can impact both small and large organizations.

* **Redundant Dependencies**: Each MFE operates independently, leading to the duplication of libraries and dependencies across different parts of the application. This redundancy can result in larger application sizes, increased load times, and negatively impact performance metrics and SEO rankings. To mitigate this, teams can create shared MFEs that consolidate common dependencies, though this approach introduces complexity in maintaining the independence of each micro-frontend (OpenReplay Blog).
* **Consistency in Design and UX**: Ensuring a uniform look and feel across all micro-frontends is difficult, as different teams may have varying design standards. This can lead to inconsistent user interfaces and overlapping CSS rules. Organizations need to establish comprehensive design systems and enforce strict communication and coordination among teams to maintain visual consistency.
* **Performance Issues**: Running multiple micro-frontend applications simultaneously can strain resources such as CPU, RAM, and network bandwidth, leading to poor performance. This issue may not be apparent during independent testing but becomes significant when all components are integrated. Effective team communication and optimized resource sharing are crucial to address this challenge ([OpenReplay Blog](https://blog.openreplay.com/common-problems-with-micro-frontends-and-how-to-avoid-them/)).
* **Increased Complexity**: The shift from a monolithic architecture to micro-frontends increases the complexity of the overall system. Managing numerous independent modules requires sophisticated orchestration and a robust DevOps infrastructure. Organizations need to invest in advanced developer tools, continuous integration/continuous deployment (CI/CD) pipelines, and comprehensive testing frameworks to handle this complexity ([McKinsey & Company](https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-forward/permanent-revolution-how-micro-frontends-can-help-to-overcome-the-struggle-of-continuous-frontend-modernization)) ([FloQast](https://floqast.com/engineering-blog/post/the-journey-to-micro-frontends-challenges-and-solutions/)) .

Large organizations adopting MFE often encounter unique difficulties. For example, IKEA implemented micro-frontends to enhance its e-commerce platform. The transition resulted in a 50% reduction in development time and a 75% reduction in page load times. However, IKEA faced challenges related to data consistency and performance optimization, which they addressed through robust caching strategies and ensuring clear data ownership . With Upwork and OpenTable, both companies leverage micro-frontends to handle high user traffic and ensure scalability. They faced challenges related to increased complexity and the need for additional tools and infrastructure to manage the distributed architecture. Implementing a robust state management strategy and ensuring effective inter-module communication were key to overcoming these obstacles ([ThinkSys](https://thinksys.com/development/micro-frontend-architecture/)) . Moreover, companies like FloQast have reported issues with modularity and dependency management, where components from different micro-frontends need to interact and share information without compromising the system’s integrity and performance (FloQast) .

## **Significance of the study**

The importance of this study is underscored by the intersection of two revolutionary architectural concepts: micro-frontends and cloud-based application architectures. As businesses embrace these paradigms at an escalating rate, it becomes crucial to grasp how they interact synergistically or present challenges. The critical need to explore the implications of integrating micro-frontends into cloud-based application architectures, particularly with regard to security measures, arises from a variety of essential factors.:

1. Rapid Technological Advancements: With the constant evolution of technology, particularly in cloud computing and web development, understanding the implications of incorporating micro-frontends is crucial to stay current and competitive in the industry.
2. Increasing Demand for Scalability: As the demand for scalable and responsive web applications continues to grow, exploring the scalability benefits of micro-frontends in cloud architectures becomes imperative for meeting user expectations and business requirements.
3. Cost-Efficiency Considerations: Cost optimization is a key concern for organizations operating in the cloud. Investigating the cost-effectiveness of micro-frontends could lead to more efficient resource utilization and budget management strategies.
4. Heightened Focus on Security: In an era of heightened cybersecurity threats, enhancing the security measures of cloud-based applications is paramount. Understanding how micro-frontends impact security indexes is vital for ensuring robust protection against potential vulnerabilities.
5. Industry Competitiveness: Delving into this topic is essential for organizations looking to differentiate themselves in the competitive landscape by leveraging the advantages of micro-frontends within cloud infrastructures.
6. User Experience Enhancement: Improved user experience is a top priority for web applications. Exploring how micro-frontends influence user experience within cloud environments can lead to enhanced customer satisfaction and retention.
7. Alignment with Industry Trends: Researching this topic aligns with current industry trends and best practices, allowing organizations to adapt proactively to new methodologies and technologies shaping the future of cloud-based web development.

By addressing these urgent considerations, this research can provide timely insights and recommendations to industry professionals, researchers, and decision-makers navigating the complexities of cloud-based application development.

## **Scope of the study**

This study will encompass a comprehensive examination of the impact of micro frontend development within cloud-based application architectures. It will concentrate on two primary subject groups:

* Leading technology companies: Including large companies such as Spotify, IKEA, Upwork, OpenTable, and others that have implemented the micro-frontend architecture in their systems. These companies provide practical cases and abundant data on effectiveness and security issues.
* Frontend experts and technical architects: This group consists of software developers and technical architects who are directly involved in designing, developing, and maintaining micro-frontend applications. They offer deep insights into technical challenges and applied security measures.

Data for this research will be collected over 10 years, from 2014 to 2024. The data collection methods include:

* Document analysis: Gathering and analyzing reports, studies, and technical documents from leading technology companies.
* Surveys and interviews: Conducting surveys and interviews with frontend experts and technical architects to gather information about their experiences and security measures.
* Analysis of real-world data: Using data from actual micro-frontend systems to assess security metrics and the effectiveness of security measures.

## **Aim of the study**

This study focuses on analyzing security metrics when implementing the micro-frontend architecture in cloud-based systems. Specifically, the research aims to:

* Evaluate common security issues encountered during micro-frontend deployment.
* Identify effective security measures to minimize risks.
* Compare the security effectiveness between traditional methods and micro-frontends in a cloud environment.

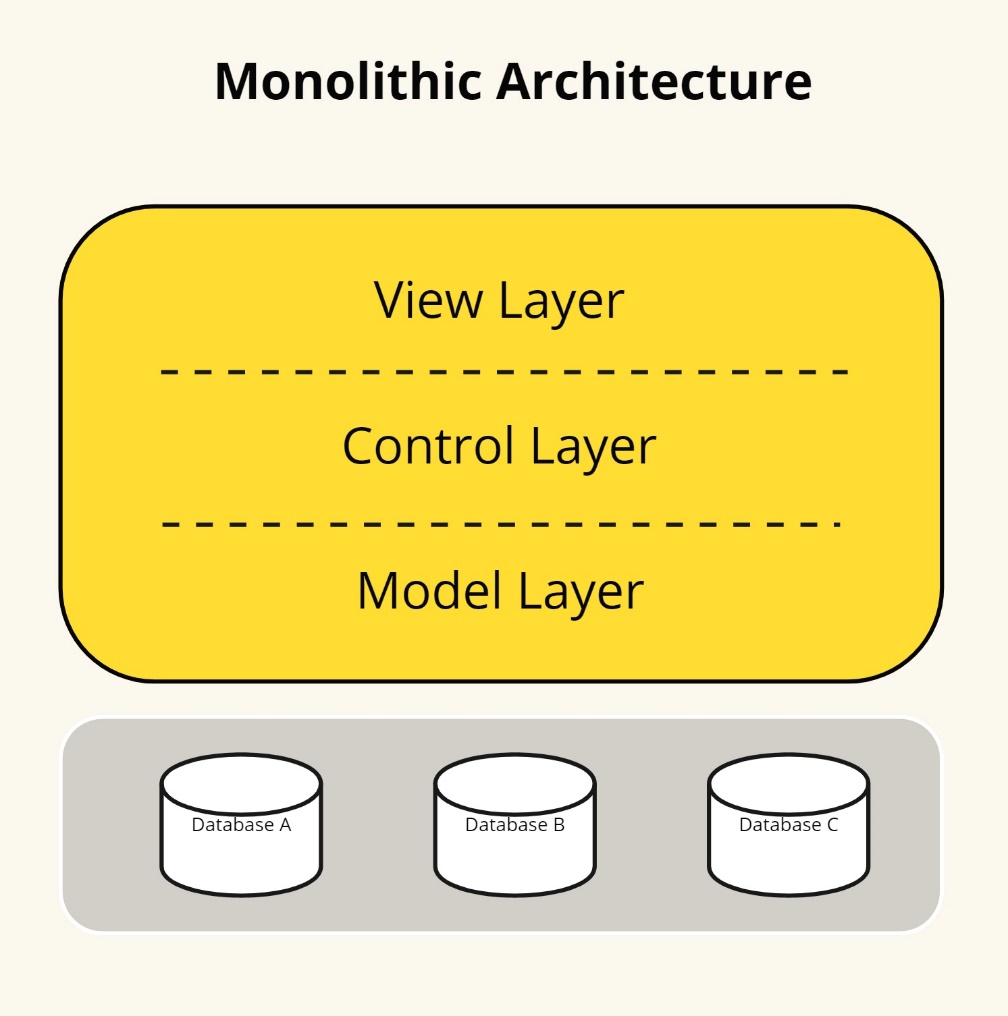
# **: LITERATURE REVIEW**

In this chapter, we provide the background information of micro-frontend and establish the context for our research work. By reviewing related work, researchers can understand the existing knowledge in their field and situate their own work within the broader scientific community.

## **Background of microservice**

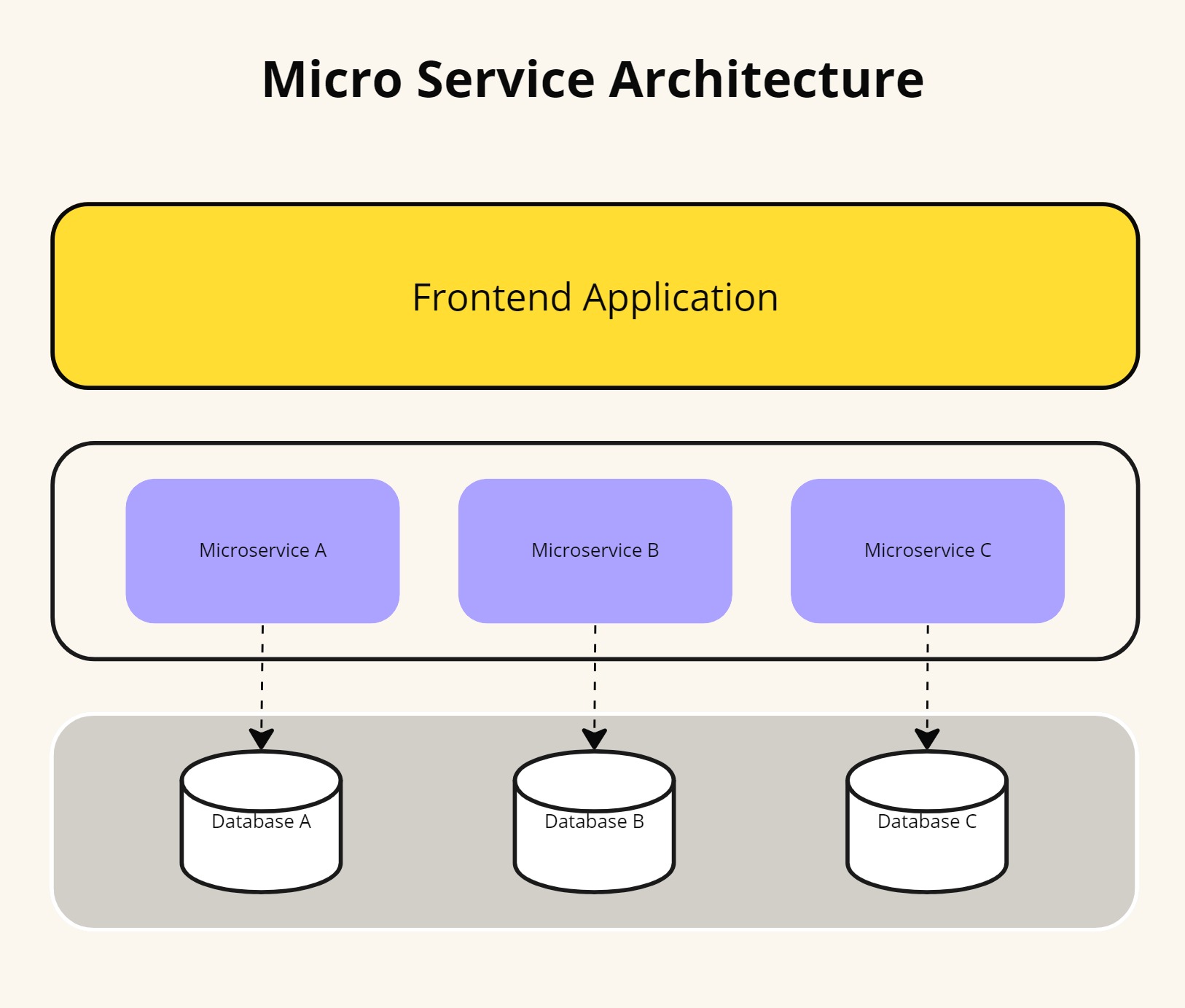
[Fix citation]

Previously, software applications were commonly created with a monolithic architecture (MA) approach, which involved merging all application processes into a single self-contained unit. This monolith integrated the user interface, business logic, and data layer within one application (Alpers et al., 2015). While this architecture offered simplicity in deployment and operation, it also presented limitations.



*Fig. Monolith architecture*

Monolithic architectures have significant issues, such as a lack of flexibility, instability, and inefficiency (Thatikonda, 2023). The more giant the monolith, the more complicated it gets to comprehend and modify. A significant challenge with scalability is that it requires altering the entire application rather than just specific components. Transitioning arose from challenges to microservices architecture (MSA) experienced with monolithic systems. The goal is to dissect complex applications into more minor, more manageable services that are loosely connected. These services can be developed, deployed, and scaled independently (Agarwal et al., 2022). This transition aimed to enhance software systems’ maintainability, scalability, and productivity. It also facilitates rapid and secure execution of changes, ensuring that the system can adapt to increasing workloads.

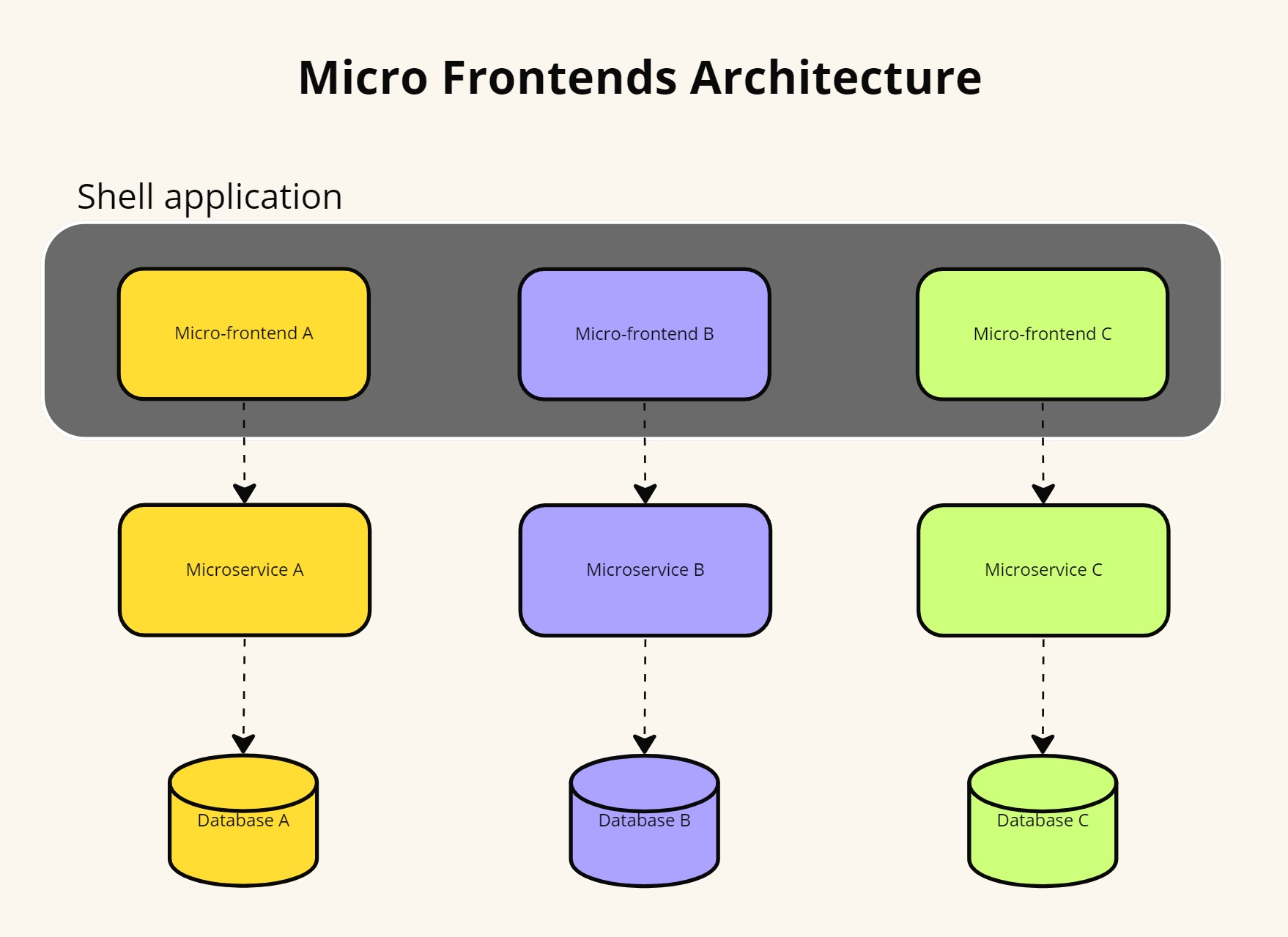
*Fig. Micro service architecture*

## **Micro-frontends architecture**

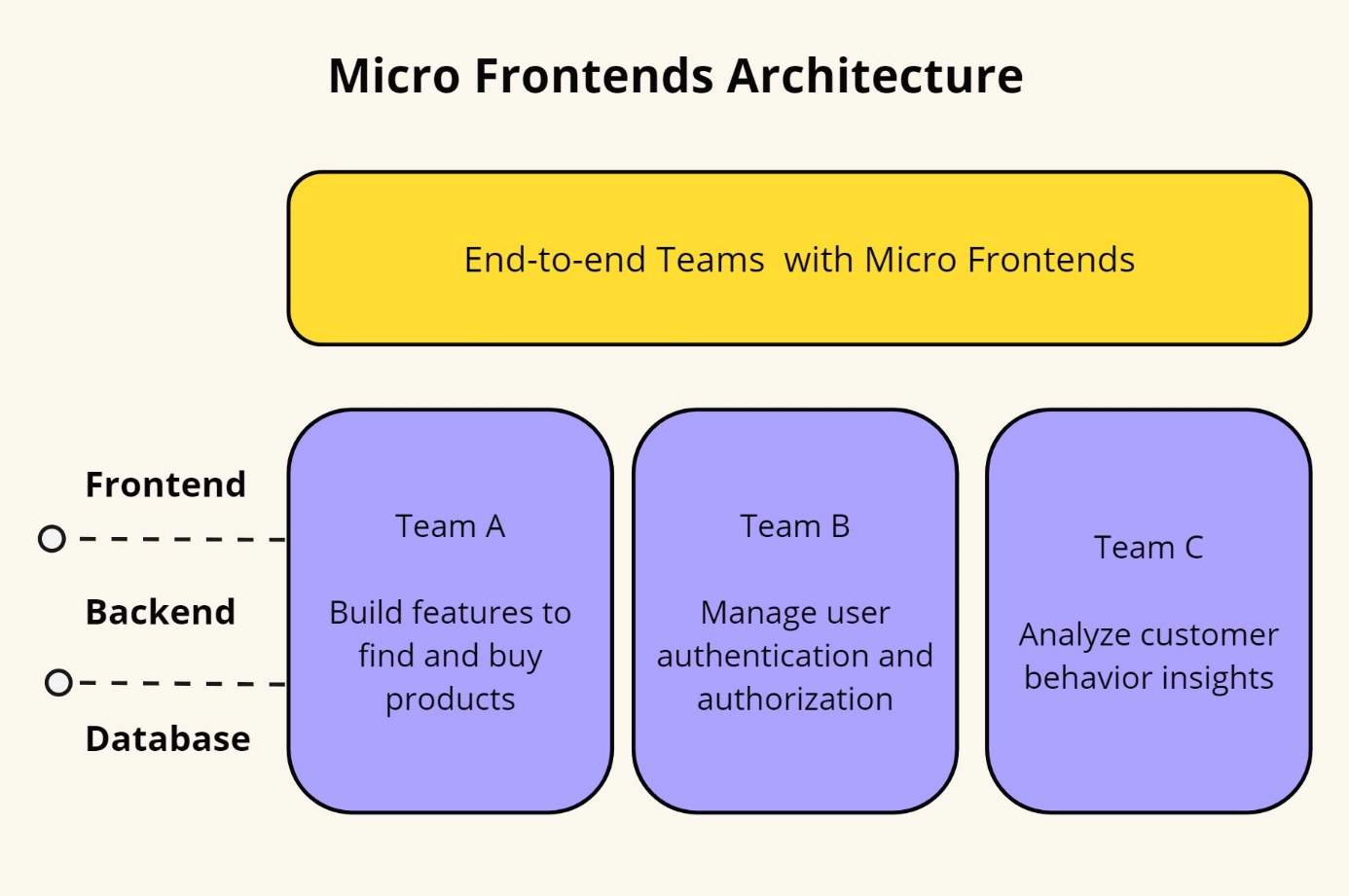
Microservices are commonly used to separate different components of an application, primarily in the backend. They structure a distributed application as a collection of services, dividing the backend into distinct functional units. Each application function becomes an independent service, which helps avoid bottlenecks in the database.

Micro frontends, however, extend the microservice concept to the frontend domain. In today’s web applications, the frontend is growing larger, while the backend is becoming less significant. Inspired by the microservices model, MFE offers a solution for modern complex web applications by allowing them to be divided into smaller, independent modules. This architectural approach involves splitting a web application into individual modules or functions, each implemented autonomously, providing frontend teams with flexibility and speed similar to what microservices offer to backend teams.

The term micro frontends or micro-frontends first appeared in 2016, in the [Thought Works Technology Radar](https://www.thoughtworks.com/es-es/about-us/news/2016/technology-radar-nov-2016) guide (Cam Jackson, 2019). A micro frontend can be a complete page or specific fragments of the page, which other teams can use to add somewhere in the page they are developing. Unlike reusable components, micro frontends can be implemented independently as individual projects. The technique for implementing micro frontends is to develop everything separately, extracting the other components at runtime to use them. The micro frontend consists of several independent and modular components that will be displayed, as required. That is, for a particular page only the necessary components are loaded. These components interact directly with the data and do not require a centralized server to route requests or process the data. In addition to the components that display content, a micro frontend can also have several utility components to interact with the application environment, it could be a user or other components.

*Fig. Micro frontend architecture*

Thus comes the vertical organization, with a less complex architecture thanks to micro frontends, which divide the application into small independent functions, each one implemented simultaneously, from the backend to the frontend, by a work team. This approach has become popular because of the problems with the monolithic approach. The frontend has grown rapidly and with a monolithic architecture, it becomes more difficult to maintain. With micro frontends, the same scalability, security, and cost-effectiveness is guaranteed as with the backend microservices architecture. The application created is less cumbersome and more user-friendly. In addition, each micro frontend can be developed with different frameworks.

*Fig. E2E micro-frontend architecture*

This effort enhances development speed and streamlines the digital product's complexity by reducing inter-team dependency.

### **2.2.1. Benefits of MFE**

Micro-frontend architecture offers many advantages that greatly enhance web application development and maintenance. Among these benefits are improved maintainability, scalability, and the facilitation of autonomous team operations.

Micro-frontends enhance the maintainability of large web applications by decomposing them into smaller, more manageable pieces. This modular approach allows development teams to focus on individual components without affecting the entire system. As noted by Guimarães et al. (2020), this separation of concerns simplifies debugging and reduces the complexity associated with maintaining monolithic applications. Each micro-frontend can be updated, tested, and deployed independently, streamlining the development process and facilitating easier management of codebases (Pereira et al., 2021).

Scalability is another significant advantage of MFA. By enabling different parts of an application to be scaled independently, micro-frontends can better handle varying loads across different components. For instance, if a specific micro-frontend experiences high traffic, only that component needs to be scaled, rather than the entire application. This efficient resource utilization leads to improved performance and cost savings (Santos & Costa, 2020). The ability to deploy and scale individual components independently aligns with modern DevOps practices, further enhancing the agility and responsiveness of development teams (Cam Jackson, 2019).

Micro-frontend architecture promotes the autonomy of development teams, allowing them to work on separate components concurrently without dependencies on other teams. This autonomy fosters a faster development cycle and encourages innovation as teams can experiment with new technologies and approaches within their specific domains. According to Mark Richards (2016), this decoupling of teams and responsibilities leads to better collaboration and productivity, as each team can operate within its own context, using the most appropriate tools and methodologies for their part of the application.

Moreover, micro-frontends facilitate technological diversity within a single application. Teams can choose different frameworks, libraries, and tools that best suit their specific component needs. This flexibility empowers teams to leverage the latest technological advancements while enabling a gradual transition from legacy systems to modern stacks without requiring a complete overhaul. The literature highlights that this capability to integrate diverse technologies mitigates the risk of technology lock-in and promotes a more adaptable development environment (’Berns & ’Kratzke, 2018).

### **2.2.2 Challenges in MFE**

Despite the benefits of micro-frontend architecture, several challenges can complicate its implementation and maintenance. These challenges include increased complexity, testing difficulties, performance issues, and the need for consistent user experiences across different micro-frontends.

One of the primary challenges of micro-frontend architecture is the increased complexity that comes with decomposing a monolithic front-end into smaller, independent components. While beneficial for maintainability and scalability, this decomposition introduces significant complexity in terms of coordination and integration. Each micro-frontend can be developed using different frameworks and technologies, leading to a heterogeneous technology stack that can be difficult to manage (Mannisto et al., 2023). This complexity necessitates sophisticated orchestration mechanisms to ensure seamless integration and communication between micro-frontends (’Berns & ’Kratzke, 2018).

Testing in a micro-frontend architecture also presents significant challenges. Ensuring that individual micro-frontends function correctly in isolation and when integrated with other components can be complex and time-consuming. According to Pereira et al. (2021), the isolated nature of micro-frontends necessitates comprehensive end-to-end testing to verify that the entire application works as intended. Additionally, testing must account for the interactions between different micro-frontends and the overall system's behavior, which can be challenging given the potential variations in technology and implementation (Sam Newman, 2021).

Performance optimization is another critical challenge. The independent deployment of micro-frontends can lead to issues such as increased loading times and inefficient resource utilization if not managed properly. Each micro-frontend may introduce its own set of resources (e.g., JavaScript files, stylesheets), leading to redundancy and potential performance bottlenecks (Santos & Costa, 2020). Effective strategies are required to manage these resources and ensure that the overall application performs efficiently.

Maintaining a consistent user experience across multiple micro-frontends is crucial but challenging. Different teams may develop each micro-frontend using varying design patterns and user interface frameworks, making it essential to implement robust design systems and shared component libraries to ensure a unified look and feel (Richards, 2015). Inconsistencies in user interfaces can detract from the user experience, necessitating meticulous coordination and standardized design guidelines (Guimaraes et al., 2020).

Finally, micro-frontends can introduce complex dependency management issues. Ensuring that all micro-frontends operate harmoniously without version conflicts or compatibility issues requires careful planning and continuous monitoring (Berns & Kratzke, 2018). Dependency management becomes particularly challenging when different micro-frontends rely on shared libraries or global state management systems, necessitating strict version control and dependency resolution strategies (Cam Jackson, 2019).

## **Cloud-based architecture with MFE**

The architectural landscape of cloud-based applications has witnessed a transformative shift, marked by the integration of micro-frontends as a key architectural paradigm. Traditionally, cloud-based applications have relied on monolithic structures, where the frontend, backend, and database components are tightly coupled into a single unit. However, the emergence of micro-frontends has challenged conventional approaches to frontend development, promoting a more modular approach (Perlin et al., 2023). Micro-frontends enable the decomposition of the user interface into smaller, independent components, each serving a specific feature or function (Cam Jackson, 2019). The adoption of cloud-based applications has brought about a significant shift in architectural design. This shift has allowed for increased scalability, improved performance, and enhanced flexibility, making cloud-based applications a valuable asset for businesses looking to stay ahead of the curve. The modular nature of micro-frontends facilitates independent development, testing, and deployment of frontend components, enhancing agility and scalability.

## **Deployment models**

[Fix citation]

The choice of cloud deployment model can significantly impact the efficiency and performance of micro-frontend architectures. The primary cloud deployment models—Software as a Service, Platform as a Service, and Infrastructure as a Service — each offer distinct advantages and potential challenges for micro-frontend implementations.

SaaS provides complete software solutions that are hosted and managed by service providers. It offers a high level of abstraction, freeing developers from concerns about underlying infrastructure and platform management. This model is highly suitable for micro-frontend architectures where applications need to be delivered quickly and updated frequently. SaaS solutions can facilitate seamless integration of various micro-frontends, providing a unified user experience across different services (Marston et al., 2011). However, the primary limitation of SaaS in the context of micro-frontends is the potential lack of customization and control over the underlying architecture. Since SaaS applications are managed by third-party providers, developers may face restrictions in implementing custom micro-frontend solutions that require specific configurations or optimizations (Benlian et al., 2012).

PaaS offers a middle ground by providing a platform that includes operating systems, middleware, and runtime environments while leaving the application code development to the users. This model is particularly beneficial for micro-frontend architectures as it supports the development, deployment, and scaling of individual micro-frontends. PaaS solutions often come with built-in tools for CI/CD, version control, and other development aids that streamline the micro-frontend lifecycle (Armbrust et al., 2010). PaaS enables developers to focus on building micro-frontend components without worrying about infrastructure management. It also allows for greater customization compared to SaaS, as developers can tailor the platform services to better fit the specific needs of their micro-frontends. However, dependency on a specific PaaS provider can sometimes lead to vendor lock-in, making it challenging to migrate to other platforms if needed (Hashizume et al., 2013).

IaaS provides the most control by offering virtualized computing resources over the internet. It includes services such as virtual machines, storage, and networking. This model is ideal for micro-frontend implementations that require high customization and control over the environment. IaaS allows developers to configure their infrastructure according to the specific requirements of their micro-frontends, enabling optimized performance and security (Mell & Grance, 2011). Using IaaS, development teams can deploy and manage each micro-frontend independently, ensuring that each component can be scaled and updated as needed. The flexibility of IaaS makes it a robust choice for complex micro-frontend architectures that need to integrate diverse technologies and services. However, this model also requires significant expertise and resources to manage and maintain the infrastructure, which can be a challenge for smaller teams (Botta et al., 2016).

## **Scalability in cloud-based applications**

Cloud scalability is the ability of a cloud computing system to adapt to changing computing requirements by either increasing or decreasing its resources, such as computing power, storage, or network capacity on demand (Kerimovs, 2023). It allows the system to adjust its resources to the workload to meet the required performance levels. This scalability often involves increasing or decreasing the number of servers, storage, or other computing resources.

This type of scalability is essential because it allows organizations to quickly adjust to the changes in their computing needs while also providing efficient use of computing resources. The goal of cloud scalability is to make sure that the cloud service can scale cost-effectively and ensure that the service can handle greater loads by adding physical or virtual resources (Rashid Dar, 2016). And this scalability is a crucial advantage of cloud computing. It allows businesses to quickly and easily scale their operations as needed without making significant upfront investments in hardware and other infrastructure.

Vertical scaling in cloud computing involves augmenting the resources of an existing instance or server to enhance its capacity and functionalities. This process facilitates the system to dynamically adjust its resource allocation based on evolving demands. Typically, vertical scaling entails enhancing the computing power of the server by increasing RAM, CPU cores, or storage capacities like hard disks or solid-state drives (Debski et al., 2018).

Vertical scaling enables your applications to run faster and handle more load without purchasing a new server or instance. And vertical scaling is a popular choice for cloud computing because it is relatively easy to do and does not require any changes to the existing infrastructure.

Horizontal scaling in cloud computing involves expanding a system by incorporating additional nodes or servers into the infrastructure. This approach is commonly employed to enhance the processing capacity of the cluster, enabling applications and services to manage a higher volume of concurrent requests or to process larger quantities of data efficiently (Rashid Dar, 2016).

In cloud computing, horizontal scaling is usually achieved by adding additional virtual machines (VMs), containers, or other resources to an existing cluster (Li et al., 2020). This type of scaling is often used to improve performance or to handle increased traffic. When done correctly, horizontal scaling can be an effective way to enhance a system's performance.

### **Security in cloud-based applications**

[Following: Securing microservices and microservice architectures: A systematic mapping study]

Despite the advantages of adopting MSA for developing complex systems, MSA introduces several challenges, notably in the realm of security (Yarygina & Bagge, 2018). Security, a longstanding issue in networking systems, becomes more complex with microservices due to the increased number of entry points and the communication traffic burden caused by decomposing systems into smaller, independent, and distributed units. Additionally, trust between individual microservices, often originating from different and unknown providers, cannot be easily established.

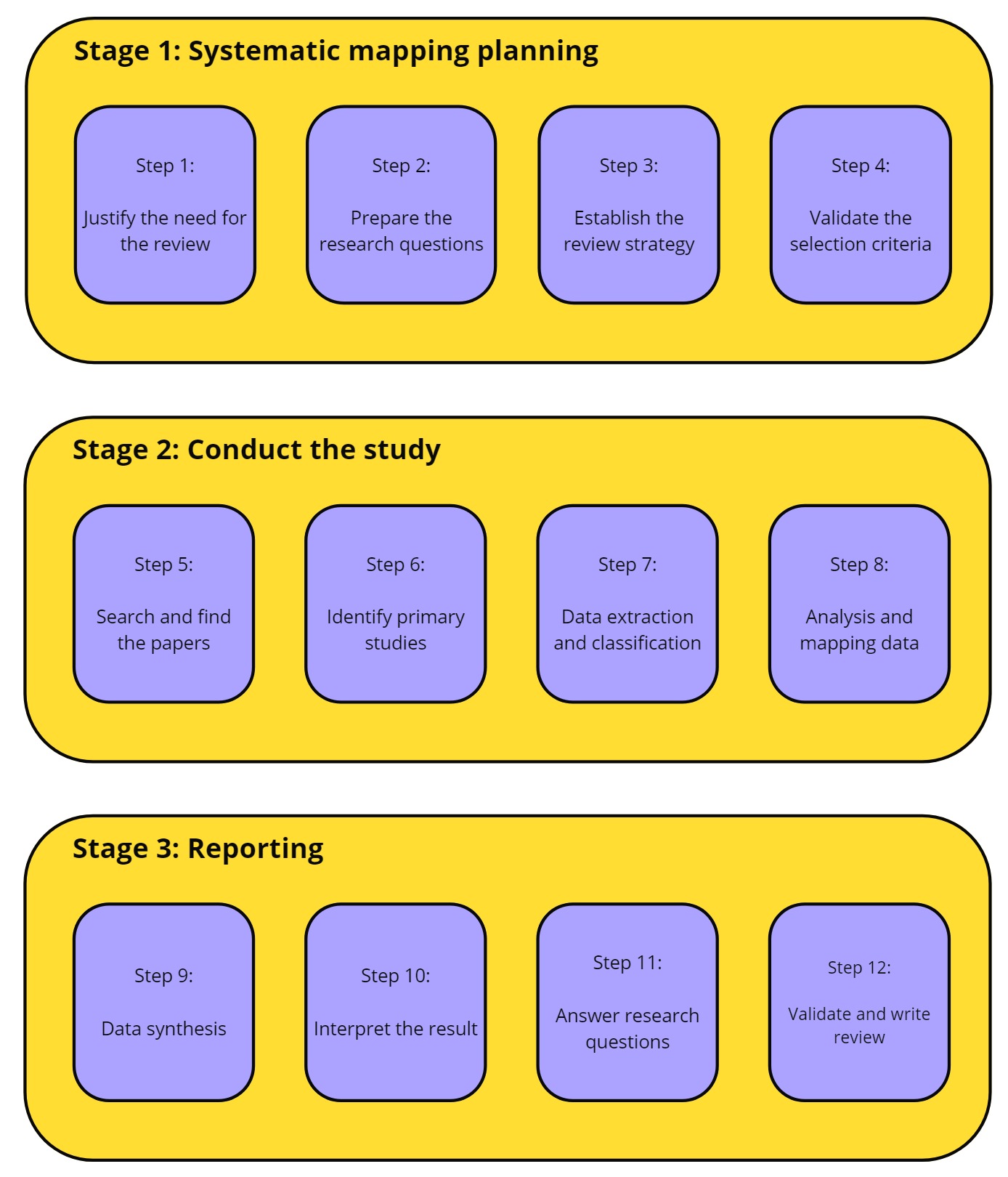
Due to the significant security breaches reported by companies adopting microservice architectures, such as Netflix and Amazon, addressing these vulnerabilities has become imperative. Numerous studies have highlighted the critical need to investigate MSA security (Baškarada et al., 2020). However, security threats are diverse and continually evolving. Consequently, security solutions are also proliferating, ranging from measures to secure individual microservices to comprehensive architectural and infrastructural protections.

In another study, Yarygina & Bagge aim to clarify microservice security by developing a taxonomy of microservice security, evaluating the security implications of microservice architecture, and reviewing contemporary solutions such as Docker Swarm and Netflix's security strategies. We present two key insights. First, microservice security is a multifaceted issue requiring a layered security approach, which is not yet available as a standard solution. Second, if these security challenges are resolved, microservice architectures can enhance security; their features of loose coupling, isolation, diversity, and rapid failure response contribute to greater system robustness. To address the lack of security guidelines, this paper outlines the design and implementation of a straightforward security framework for microservices that practitioners can use. Proof-of-concept evaluation results indicate that the performance overhead of the security mechanisms is approximately 11%.

### **Systematic mapping**

A systematic mapping is a form of evidence-based software engineering (EBSE) (Kitchenham et al., 2015) that aims to provide an overview of a research area. It achieves this by creating a classification scheme and organizing evidence related to a specific research field.

In 2015, Peterson et al. proposed an overarching process for conducting systematic mappings, which consists of three main steps: planning, conducting, and reporting. During the planning phase, researchers justify the need and scope of the mapping, formulate research questions, and develop a protocol that outlines key decisions for conducting the mapping. This protocol includes defining search terms, specifying the search strategy, identifying literature sources, selecting relevant papers, and determining how data will be extracted and synthesized. In the conducting phase, the validated protocol from the planning step is put into action. Researchers use identified sources to retrieve papers, assess their relevance, extract useful data, and classify it. Finally, during the reporting phase, data from primary papers are visualized, results are interpreted, research questions are addressed, and the mapping process is validated and documented. *Figure*  illustrates the overall systematic mapping process. Note that the quality assessment step, depicted within a dashed line box in *Figure* , is optional according to Kitchenham and Peterson.



*Fig. Overview systematic mapping*

## **Summary [Need more paragraphs]**

In conclusion, the benefits of micro-frontend architecture, including enhanced maintainability, scalability, autonomous team operations, technology diversity, and improved user experience, make it a compelling choice for modern web application development. These advantages are well-documented in the literature, providing a solid foundation for adopting this architectural approach.

Choosing the appropriate cloud deployment model for micro-frontend implementations depends on the specific needs and resources of the development team. SaaS offers simplicity and ease of integration but may lack flexibility. PaaS provides a balanced approach with greater control and development support. IaaS offers the highest level of customization and control but requires extensive management. Understanding the strengths and limitations of each model can help in making informed decisions that align with the goals and constraints of the micro-frontend project.

# **: RESEARCH METHODS**

## **3.1 Security model for MFA**

In this section, we outline the protocol adopted for conducting this mapping study. Following the guidelines of Petersen et al. (2008) and Felderer and Craver (2017), a systematic mapping study involves several primary steps: defining research questions, searching for relevant papers, screening the identified papers, proposing or using an existing classification scheme, extracting data, and mapping the studies. The subsequent sections describe each step in detail.

### **3.1.1 Research objectives for security in MFA**

The adoption of microservice architectures in developing highly sensitive systems, such as those in the industrial sector, raises concerns about the risks associated with deploying microservices on distributed and highly accessible platforms. This study aims to identify the threats targeting individual microservices and microservice architectures that could compromise systems' security, stability, and reliability. Additionally, we seek to collect and categorize all known threats and the efforts to mitigate and prevent them into easy-to-handle, reliable, and extensible structures, such as ontologies. This approach allows (1) researchers and practitioners to gain an overview of the risks that could threaten their existing or future systems, and (2) facilitates the sharing of experiences on how to mitigate and prevent recognized security threats. In summary, the objectives of this research can be articulated as follows:

Q1. Identify and categorize threats targeting individual micro-frontends and cloud-based architectures.

Q2. Recognize and categorize efforts to identify, reduce, and prevent known security threats.

Q3. Determine the validation techniques and tools used to confirm identified security measures.

Q4. Develop and share a simple ontology for securing micro-frontends on cloud-based systems.

## **3.1.2 Research questions**

We conducted a comprehensive study of 10 open-source systems employing micro-frontends. For each system, we thoroughly analyzed the source code, manually annotating each security feature. These systems were developed and published by practitioners with a background in frontend development.

To expand our dataset and explore the design space, we created 20 additional models representing system variants. These variants were adapted from a published example, guided by discussions in relevant literature. Aside from the specific variations described in Tables 1 and 2, all other system aspects remained consistent with the base models. In total, we analyzed 30 models, summarized in Tables 1 and 2. We consider these evaluation systems to be practical examples that either reflect or closely resemble real-world microservice architectures.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Research Question | Motivation | Objectives |
| SRQ1 | What are the most address security threats of micro-frontends and how they can be classified? | This research question distinguishes the list of mostly treated vulnerabilities from those needing further investigations. | 01, 04 |
| SRQ2 | At what level of architecture the proposed techniques approaches are applicable for securing micro-frontends? | This research question identifies where security measures are applied, emphasizing the less focused areas of micro-frontends architecture. | 02, 04 |
| SRQ3 | At what level of architecture the proposed  techniques and approaches are applicable for  securing microservices? | This research question indicates where  security is applied highlighting the less  focused levels of microservice architectures. | 02, 04 |
| SRQ4 | What domains or platforms are the focus of  existing solutions for securing microservices  and microservice architectures? | This research question shows whether the  focus of the proposed solutions is platform  specific or platform independent. | 02, 04 |
| SRQ5 | What kind of evidence is given regarding the  evaluation and validation of proposed  approaches and techniques for securing  microservices and microservice architectures? | This research question evaluates the maturity  of existing security techniques highlighting the  set of empirical strategies used to validate  proposed solutions. | 03 |

*Table. Research questions and motivation*

## **3.1.3 Search process**

The search string used in this study is intentionally designed to be generic and simple. It is constructed based on search terms related to the population and intervention, as suggested by Petticrew and Roberts (2006). In this context, the population refers to the application area, which includes microservices and microservice architectures, while the intervention pertains to security, vulnerabilities, and attacks. Consequently, the final search string adopted is:

("microservice" OR "micro-service" OR "micro service")

AND

("architecture" OR "design" OR "system" OR "structure")

AND

("security" OR "vulnerability" OR "attack")

To retrieve relevant studies, we followed the guidelines of Kuhrmann et al. (2017). Consequently, we utilized the following online academic libraries:

• IEEE Xplorer (https://ieeexplore.ieee.org)

• ACM Digital Library (https://dl.acm.org)

• SpringerLink (https://link.springer.com)

• ScienceDirect (https://www.sciencedirect.com/)

• Wiley Online Library (https://onlinelibrary.wiley.com).

To ensure no relevant studies were overlooked, we supplemented our automatic search with recursive backward and forward snowballing on selected studies, as recommended by Wohlin (2014, 2016). In backward snowballing, we examined the references in approved papers for relevance. In forward snowballing, we assessed the relevance of papers that cited the approved papers. This snowballing process was applied recursively to each newly approved paper. Google Scholar was used exclusively for forward snowballing.

## **3.1.4 Study selection process**

The set of papers retrieved through automatic search underwent two screening stages. Firstly, titles and abstracts were assessed for relevance, followed by a second stage where the full texts were examined to determine if they met our inclusion criteria. Each paper was independently screened by both authors, with decisions exchanged and conflicts resolved through discussion.

Additionally, papers identified through snowballing underwent separate screening by the two authors before inclusion or exclusion decisions were made.

In our study, we systematically assessed the support or violation of the collected security tactics. The following steps were followed:

1. Recommendation as Ordinal Ratings: Other authors applied our recommendations as an ordinal rating scheme to each model variant summarized in Tables 1 and 2 [Check table #], creating a ground truth for our study.

2. Expert Review: Three industrial security experts from our author team and two experts from another company reviewed the rating scheme and the ratings in the ground truth, contributing their insights and expertise to the robustness of our findings.

3. Statistical Analysis:

- Spearman Rank Correlation: Initially, we examined the correlation between the independent variables and the dependent variable using Spearman rank correlation, a method commonly used for analyzing the relationship between continuous and discrete ordinal variables.

- Ordinal Regression: To assess how well the hypothesized metrics predicted the ground truth data, we performed an ordinal regression analysis. This involved modeling the dependence of an ordinal response on a set of independent predictors. The lrm function from the rms package was utilized for this analysis.

Through expert judgment and rigorous statistical methods, our study provides valuable insights into the proposed metrics' effectiveness.

## **3.1.5 Inclusion and exclusion criteria**

The number of papers retrieved from online academic libraries was narrowed down by applying strict inclusion and exclusion criteria. In this study, we included only peer-reviewed papers from journals and conferences. The automatic search covered all publications from 2016 onward, as there was no consensus on the term "micro-frontend" before 2016. We included only English-language papers that address security aspects or security solutions for microservices or microservice architectures. The complete list of inclusion and exclusion criteria is presented in Tables 2 and 3, respectively.

|  |  |
| --- | --- |
| ID | Criteria |
| I1 | Papers published since 2016 including early publications |
| I2 | Papers written in English |
| I3 | Papers subject to peer reviews |
| I4 | Papers including studies conducted with security aspects of microservices or microservice architectures as their primary topics |
| I5 | Papers proposing frameworks, techniques, methods, or tools to secure microservices or microservice architectures |
| I6 | Papers presenting qualitative or quantitative evaluation of security techniques used for microservices or microservice architectures |

*Table. Inclusion criteria*

|  |  |
| --- | --- |
| ID | Criteria |
| E1 | Papers addressing security in distributed platforms and technologies such as clouds without explicit referring to microservices |
| E2 | Papers describing general aspects of microservice architectures  without putting much emphasis on the security issue |
| E3 | Tutorial papers and editorials |
| E4 | Papers presenting reviews, surveys or secondary studies on the  security of microservices or microservice architectures |
| E5 | Books or book chapters, because they usually undergo little peer  review and present general ideas already published in journals or  conferences |
| E6 | Papers without full text available |

*Table. Exclusion criteria*

### **3.1.6 Quality assessment**

As advised by Petersen et al. (2015), conducting a quality assessment of identified papers is essential for mapping studies to ensure that sufficient information is available for data extraction. This step is crucial for evaluating the overall robustness and details provided in the selected papers. Therefore, we undertook a comprehensive quality assessment process to meticulously evaluate these aspects.

Our quality assessment utilized a questionnaire consisting of four carefully chosen items, inspired by previous studies from Hannousse & Yahiouche (2021) and tailored to fit our specific research topic. Each item in the questionnaire served as a criterion for evaluating the quality and reliability of the studies under consideration.

|  |  |  |
| --- | --- | --- |
| ID | Criteria | Score |
| QA1 | Does the study clearly address any security issues in micro-frontend architectures? | {0, 1} |
| QA2 | Does the study present a clear solution to any security threat in micro-frontend architectures? | {0, 0.5, 1} |
| QA3 | Has the study been cited by other articles? | {0, 1} |
| QA4 | Has the study been published in ranked journal or conference proceedings | {1, 0.5, 0} |

*Table. Quality Assessment Criteria*

For QA1, a score of 1 was awarded if a criterion could be answered with "Yes," indicating that the paper met the criterion fully, while a score of 0 was given if the answer was "No," indicating that the paper did not meet the criterion. This binary scoring system ensured clarity and straightforward evaluation.

For QA2, a more nuanced scoring system was applied: a score of 1 was given to studies presenting a detailed and validated solution to a security threat, demonstrating a thorough approach to addressing the issue. A score of 0.5 was assigned to studies providing only an overview of the solution or framework, reflecting partial but incomplete information. Studies lacking clear solutions received a score of 0, indicating insufficient detail or validation.

QA3 evaluated the academic impact of the studies based on citation counts. A score of 1 was assigned to studies with three or more citations, reflecting recognition and validation by the academic community. In contrast, studies with fewer than three citations received a score of 0. Google Scholar was used to determine citation counts to avoid penalizing recent publications that might not yet have accumulated significant citations.

For QA4, the source and prestige of the paper were assessed using the CORE conference rankings and the Journal Citation Reports (JCR). Papers were ranked as follows: CORE A or B conferences were given a score of +1, CORE C conferences received a score of +0.5, and unranked conferences were given a score of 0. For journal sources, JCR Q1-2 journals received a score of +1, JCR Q3-4 journals got a score of +0.5, and unranked journals received a score of 0. This criterion ensured that the source's credibility was factored into the overall assessment.

All four criteria were treated equally, ensuring a balanced evaluation of each study. The total score for each study was calculated by summing the four values, providing a comprehensive quality metric. Only studies with a total quality score of 2 or higher were included in this study, ensuring that only the most robust and detailed papers were considered.

The assessment criteria and their associated scores are summarized in Table 4, which provides a clear overview of the evaluation process and the standards applied in this study. This rigorous approach ensures that our data extraction is based on high-quality, reliable sources, enhancing the credibility and validity of our findings.

### **3.1.7 Data extraction process**

Following the guidelines of Petersen et al. (2015), a comprehensive data extraction form was designed to ensure systematic and detailed collection of information from each selected paper. This form is illustrated in Table 5, providing a structured template for consistent data gathering.

Each paper included in the study is meticulously described in terms of its metadata, which encompasses essential details such as the year of publication, the source (journal or conference), and the type of publication (e.g., empirical study, theoretical paper, case study). This metadata provides a foundational context for understanding the background and relevance of each paper within the broader research landscape.

In addition to metadata, the data extraction form captures a set of specific information critical to our analysis. This includes a comprehensive list of the security threats or attacks addressed by the study. Identifying these threats is vital for mapping out the security challenges that microservice architectures face and understanding the scope of each paper's focus.

Furthermore, the form documents the proposed solutions for the identified security threats. This involves detailing the nature of each solution, whether it is a technical mechanism, a theoretical framework, a practical implementation, or a combination thereof. The specificity of these solutions is crucial for assessing their applicability and potential effectiveness in real-world scenarios.

The application level of the proposed solutions is another key piece of information extracted. This involves determining whether the solutions are applicable at the level of individual microservices, the microservice architecture as a whole, or other levels such as network or infrastructure. Understanding the application level helps in categorizing the solutions and identifying gaps or overlaps in the existing research.

Additionally, the form records the validation methods used to assess the proposed solutions. This includes whether the solutions have been validated through empirical testing, simulation, case studies, expert reviews, or theoretical analysis. The robustness of these validation methods is critical for evaluating the reliability and generalizability of the findings presented in each paper.

Finally, the application platforms on which the proposed solutions have been tested or are intended to be applied are also documented. This includes information about the specific technologies, tools, and environments used in the studies, providing insight into the practical applicability and limitations of the solutions.

By following this rigorous data extraction process, we ensure that all relevant information is systematically collected and organized, facilitating a thorough analysis of the security landscape in microservice architectures. This methodical approach allows for a comprehensive understanding of the current state of research, identifying both strengths and areas in need of further investigation.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Data Item | Description | Research Question |
| D1 | Study ID | First author name + year |  |
| D2 | Year | Year of the publication |  |
| D3 | Source | Source of the publication |  |
| D4 | Type | Conference or journal paper |  |
| D5 | Category | Analysis, solution proposal or case study |  |
| D6 | Threats | Addressed security threats | SRQ1 |
| D7 | Source of Threats | Internal or external | SRQ1 |
| D8 | Solution type | General protection measures, framework, technique, tool or methodology proposal | SQR2 |
| D9 | Applicability level | Architectural level where the security mechanism is applied | SRQ3 |
| D10 | Domain/Platform | Domains or platforms applicable for the proposed solution | SRQ4 |
| D11 | Validation method | Verification and validation techniques used to check the feasibility of the proposed solution | SRQ5 |

*Table. Data extraction*

### **3.1.8 Data synthesis**

We observed a significant lack of consensus on detailed taxonomies for security threats and mechanisms, which hindered our ability to categorize all selected studies appropriately and distinctly to answer research questions RQ1 and RQ2. Additionally, the diversity of applications, targeted platforms, and verification and validation techniques used in the selected studies necessitated a proper categorization to effectively address research questions RQ3, RQ4, and RQ5.

To map all the selected studies accurately to appropriate categories for each research question, we leveraged our experiences and the existing research of Aguiar Monteiro et al. (2020) in identifying categories and their relationships. We also employed grounded theory as a complementary approach to generate missing categories from the extracted data items. Specifically, we utilized open coding and selective coding to identify categories and their relationships with existing categories from D6, D9, and D10-11. In this study, grounded theory was applied iteratively, where categories and subcategories were refined in each iteration until a stable state was achieved.

1. **: RESEARCH RESULTS**

## **4.1 Security research**

In this section we describe and detail the results of the mapping study answering the five research questions outlined in Chapter 3.2

### **4.1.1 Overview of selected studies**

The search process, conducted in December 2023, identified 82 distinct papers on micro-frontend architecture published since 2011. The search query was applied across several academic libraries, with the results summarized in Table 6, indicating the number of papers returned by each library. Initially, 1540 papers were retrieved from various search engines. After removing duplicates, the number was reduced to 1500.

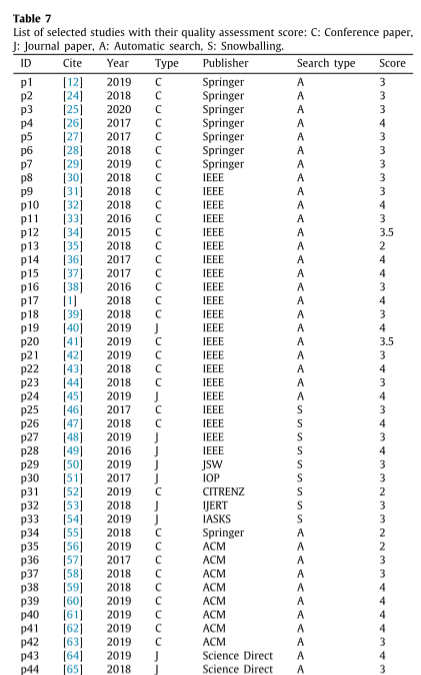
We then screened the titles and abstracts of the remaining papers, excluding 1430 papers due to irrelevance. Applying the inclusion and exclusion criteria further narrowed the list to 70 approved papers. To ensure comprehensive coverage, we employed recursive backward and forward snowballing, which added 18 more papers to our study. This snowballing process involved two cycles to achieve a steady state: the first cycle included 12 new papers, and the second cycle added 6 more papers. Figure 3 illustrates the overall selection process in detail.

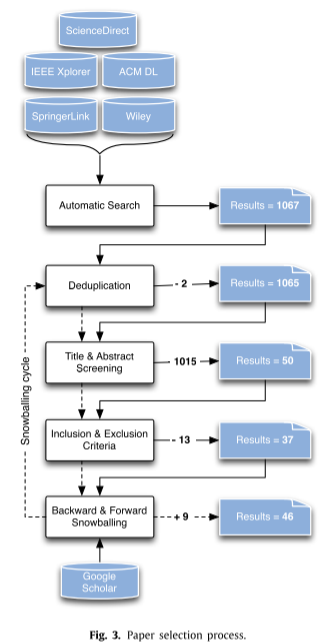
Figure 4 displays the distribution of selected studies by their publication year and source. Although microservice architecture (MSA) was introduced as early as 2011, significant interest in securing micro-frontends and related architectures only began to grow around 2017. This trend is depicted in Figure 4, which also shows that the highest number of relevant publications came from IEEE Xplore, while none from Wiley met our inclusion criteria.

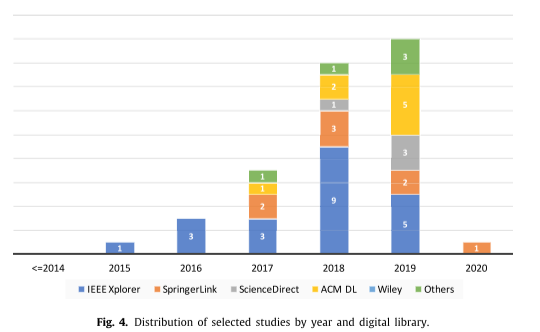
Table 7 presents the complete list of selected studies, including their year of publication, type of publication, method of discovery, and quality assessment scores. This detailed presentation helps understand the breadth and depth of research conducted in micro-frontends and their security considerations.

|  |  |
| --- | --- |
| Repository | Search results |
| IEEE Xplorer |  |
| ACM Digital Library |  |
| SpringerLink |  |
| ScienceDirect |  |
| Wiley Online Library |  |

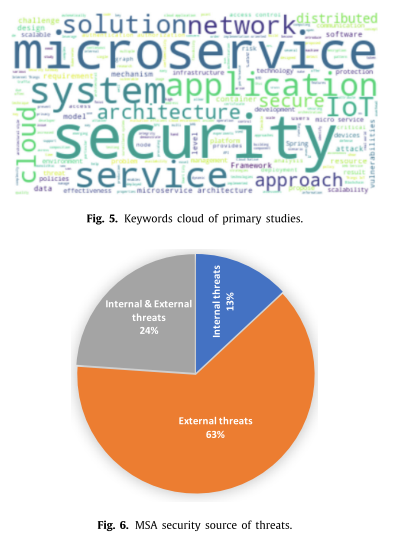
*Table. Number of studies returned by each repository*







Following the guidelines of Kuhrmann et al. (2017), we employed Word Clouds to analyze the relevance of our result set of primary studies. Figure 5 illustrates the most frequent words found in the titles and abstracts of the selected papers. The figure highlights that the most commonly used words are "microservice," "security," "application," "architecture," "system," and "service." Conversely, terms such as "attacks," "vulnerabilities," and "risks" are less frequently used in the titles and abstracts.



### **4.1.2 MFA security threats (SRQ1)**

Micro-frontend architecture, an emerging paradigm in software engineering, introduces new security threats and vulnerabilities. These threats can originate from insiders (i.e., internal attacks) or outsiders (i.e., external attacks). To properly secure micro-frontend-based systems, all threats, regardless of their origin, must be detected and prevented using either existing mitigation techniques or innovative solutions. This study aims to identify the focus of existing efforts concerning the source of threats (internal, external, or both). Figure 6 depicts the distribution of identified and selected studies based on the addressed source of threats. The findings indicate that 63% of primary studies focus on external attacks, only 13% address internal attacks, and 24% cover both sources of threats. This demonstrates a clear research bias towards external attacks.

Given the plethora of taxonomies of security threats and the lack of consensus on their categorization, we adopted a classification based on the targets of attacks. Accordingly, threats in micro-frontend architecture can be classified into:

* User-based attacks: Attacks involving users directly (i.e., malicious user actions) or indirectly (i.e., inadvertent insider actions).
* Data attacks: Threats targeting sensitive data that can be disclosed and manipulated by attackers.
* Infrastructure attacks: Attacks targeting architectural elements and platforms such as monitors, discovery services, message brokers, load balancers, etc.
* Software attacks: Threats involving code transformation or injection for malicious purposes.

Table 8 shows the set of micro-frontend security threats addressed by primary studies, grouped by category. The results reveal that unauthorized access, sensitive data exposure, and compromising individual micro-frontends are the most frequently addressed threats by contemporary studies. Additionally, infrastructure attacks are the most diverse but are less frequently addressed in the selected studies.

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Security Threats | Number of Studies | Percentage |
| User-based Attacks | Malicious User Actions | 8 | 21.6% |
|  | Inadvertent Insider Actions | 5 | 13.5% |
| Data Attacks | Sensitive Data Exposure | 10 | 22.7% |
|  | Data Manipulation | 7 | 18.9% |
| Infrastructure Attacks | Unauthorized Access | 12 | 32.4% |
|  | DoS Attacks | 6 | 16.2% |
|  | Compromising Monitors | 4 | 10.8% |
|  | Compromising Discovery Services | 3 | 8.1% |
|  | Compromising Message Brokers | 2 | 5.4% |
| Software Attacks | Code Injection | 9 | 24.3% |
|  | Code Transformation | 4 | 10.8% |

*Table. MFA security threats per category*

### **4.1.3 MFA security mechanisms (SRQ2)**

In the context of micro-frontend architecture, the security mechanisms proposed in primary studies can be categorized as follows:

* General Protection Measures: These include the use of general security techniques to mitigate known threats in MFA, or a set of guidelines on choosing appropriate languages and technologies.
* Framework-based Solutions: These are architectural frameworks for MFA that incorporate specific modules to handle certain security aspects and mechanisms such as authorization, continuous monitoring, and diagnosis.
* Technique-based Solutions: These are newly designed or adopted techniques from other domains to mitigate or prevent some security threats in MFA.
* Tool-based Solutions: These are newly developed tools that implement security measures.
* Algorithm-based Solutions: These are new algorithms designed for the detection or prevention of security threats.
* Protocol-based Solutions: These are new protocols designed for the protection of communications among the different MFA architectural elements.
* Analysis: This involves experimentation, comparison, or discussion of existing security mechanisms of MFA.

Our investigation (see Fig. 7) shows that 33% of the studies proposed new techniques for securing MFA, 31% proposed framework-based solutions, and 13% proposed general protection measures. Few studies developed new tools, algorithms, or protocols. Specifically, the authors of P17 analyzed existing security mechanisms and proposed a framework based on the insights of the conducted analysis.

Proposed solutions for securing microservices and micro-frontend architectures can be classified into proposals for enforcing Access Control (i.e., authentication and/or authorization policies), auditing, mitigation, and prevention:

* Access Control - Authentication: Techniques used to verify the identity of users requiring access to MFA resources and data.
* Access Control - Authorization: Techniques used to check users’ permissions for accessing specific MFA resources or data.
* Auditing: Techniques applied at runtime for discovering security gaps and may: (1) subsequently initiate appropriate measures or (2) simply report security breaches to relevant supervisory authority.
* Mitigation: Techniques that limit the damage of attacks when they appear. Mitigation techniques can be integrated into existing microfrontend-based systems.
* Prevention: Techniques that try to stop attacks from happening in the first place. Prevention techniques need to be considered when developing new micro-frontend-based systems.

Table 9 shows the list of proposed solutions mapped into our classification with the proportion rate for each proposal with respect to the total set of primary studies. The results show that much emphasis is put on proposing auditing techniques (43.48%), enforcing authentication (28.3%) and/or authorization (21.7%), and prevention (34.7%), where less attention is being paid to mitigation (10.8%).

|  |  |  |
| --- | --- | --- |
| Security Mechanisms | Description | Proportion Rate |
| Access Control - Authentication | Techniques to verify users’ identities for accessing MFA resources and data. | 28.3% |
| Access Control - Authorization | Techniques to check users’ permissions for specific MFA resources or data. | 21.7% |
| Auditing | Techniques for discovering security gaps at runtime, reporting breaches, or taking action. | 43.5% |
| Mitigation | Techniques to limit damage from attacks, integrated into existing micro-frontend systems. | 10.8% |
| Prevention | Techniques to prevent attacks proactively during development of new micro-frontend systems. | 34.7% |

*Table. MFA security mechanism per category*

#### **4.1.3.1 Security measures in MFA: Application layered approach (SRQ3)**

The adoption of micro-frontend architecture (MFA) introduces security vulnerabilities across different architectural layers. To address these challenges, security measures must be implemented at each layer within the MFA context. We identify the following layers and their associated security considerations:

1. Microservice Layer:

* Individual microservices form the backbone of MFA. However, they are susceptible to blockage or compromise through malicious code injection.
* Security measures should prioritize adopting trusted microservices and safeguarding them against internal and external threats.

1. Composition Layer:

* Interconnections among microservices can break down, and compromising a single microservice may impact the entire system.
* Insecure configuration options for individual microservices, their locations, and interconnections necessitate security measures at this level.

1. API Layer:

* Fine-tuned attacks on APIs can circumvent traditional security provided by API gateways.
* Malicious users may gain unauthorized access, potentially compromising assets.
* Rigorous security measures at API gateways are essential to mitigate these vulnerabilities.

1. Communication Layer:

* Data exchanged between microservices via event buses can be intercepted and altered by insiders.
* Securing communication channels between microservices is imperative for overall system security.

1. Deployment Layer:

* Containers housing microservices can also harbor vulnerabilities.
* Unauthorized access or vulnerabilities from untrusted image sources pose risks.
* Security measures at the deployment level are crucial.

1. Soft-Infrastructure Layer:

* Infrastructure vulnerabilities affect all software entities on the network, including monitors, registries, message brokers, and load balancers.
* Techniques introduced at this level ensure the security of diverse software network entities and their configurations.

1. Hard-Infrastructure Layer:

* Hardware components are susceptible to intentional or unintentional attacks.
* Bugs or backdoors introduced during manufacturing can be exploited.
* Implementing error and backdoor detection mechanisms is essential.

Table 10 summarizes the distribution of proposed solutions across application layers in primary studies. Notably, solutions focus heavily on soft infrastructure and API gateways, while composition and hard-infrastructure layers receive less attention.

|  |  |  |
| --- | --- | --- |
| Application Layer | Description | Emphasis (%) |
| Microservice Layer | Security measures for individual microservices to prevent compromise and malicious code injection. | 20% |
| Composition Layer | Measures to secure interconnections among microservices and address insecure configuration options. | 6.7% |
| API Layer | Rigorous security at API gateways to prevent fine-tuned attacks and unauthorized access. | 22.2% |
| Deployment Layer | Security measures for containers housing microservices. | 13.3% |
| Soft-Infrastructure Layer | Techniques to safeguard diverse software network entities (monitors, registries, etc.). | 68.9% |
| Hard-Infrastructure Layer | Detecting vulnerabilities in hardware components introduced during manufacturing. | 8.9% |

*Table. Distribution of Proposed Solutions by Application Layers*

#### **4.1.3.2 MFA security mechanisms target platforms (SRQ4)**

In our study, we categorize identified papers based on their target platforms and the applications for which they propose security solutions. Here’s an overview:

1. Cross-Platform Solutions:

* Approximately 34.78% of the papers propose security solutions applicable across different platforms within the micro-frontend context.
* These solutions address security concerns irrespective of the specific environment.

1. Cloud Platform Solutions:

* A similar proportion of studies focuses on securing micro-frontends within cloud platforms.
* These solutions cater to the unique security challenges in cloud-based micro-frontend architectures.

1. Platform-Specific Solutions:

* A smaller subset of studies delves into platform-specific security measures:
* 5G Platform: Addressing security aspects specific to 5G networks in micro-frontend setups.
* IoT: Solutions tailored for securing Internet of Things (IoT) micro-frontends.
* Web Applications: Targeting security in web-based micro-frontend architectures.
* Kubernetes Platforms: Proposing security enhancements for micro-frontends orchestrated by Kubernetes.
* Springer Framework: Specific security considerations within the Springer framework for micro-frontend development.

|  |  |  |
| --- | --- | --- |
| Platform Layer | Description | Emphasis (%) |
| IoT platform |  |  |
| Cloud platform |  |  |
| Container-based platform |  |  |
| Web applications |  |  |
| Kubernetes platforms |  |  |
| Independent |  |  |

*Table.MFA security targets multiple platforms*

#### **4.1.3.3 Microservice security V&V methods (SRQ5)**

For validating the proposed solutions in micro-frontend architectures, we distinguished the use of several verification and validation approaches:

* Validation by simulation: This includes the use of simulated lab environments or testbeds for testing proposed designs, and the simulation of attacks to check the bypassing of proposed security measures.
* Manual testing: This includes use case-based testing, emulating attacks, and reconfigure-testing cycles.
* Performance analysis: This is performed by measuring overheads, latency, throughput, memory storage, CPU usage, response time, and traffic measurement.
* Qualitative analysis: This involves comparing or verifying a set of qualitative requirements. These include causing a single point to failure, complexity of cracking, complexity of implementation, and potential inherent bottleneck.
* Quantitative analysis: This involves comparing the proposed solution with similar proposals using quantitative metrics such as session sustainability, and popular machine learning metrics.
* Adhoc metrics: Proposing specific metrics for the evaluation of proposals. For example, P18 proposed a diversification index as a security measure to validate the proposal.
* Case study based validation: Using case studies to validate the feasibility of the proposed solution.
* Proof of concept (POC): Developing a prototype to demonstrate the feasibility of the proposed solution.
* Tool-based testing: Using unit-based testing tools such as IntelliJ IDEA.
* Formal verification: Using model checkers or theorem provers to check the validity of specified properties.
* Complexity measuring: Estimating the temporal complexity of proposed algorithms implementing solutions.
* Methodology based analysis: Using predefined and well-known methodologies such as OWASP risk rating, attack surface, or security risk comparison.

Table 12 shows that performance analysis and case study based validation are the most adopted techniques for verification and validation. Formal verification, POC, and methodology-based analysis are the least used methods for validation. Validation by simulation and adhoc metrics are found used in equal measure with a rate equal to 6.52%. Most simulation methods adopted simulated lab environments or testbeds. Only P3 used simulated attacks to evaluate the proposed technique. Note that most examined studies used more than one validation technique and two studies (P17 and P27) proposed solutions without using any mentioned validation technique. Instead, they discussed the details of proposed solutions and claimed they are enough to mitigate the addressed security threat(s).

|  |  |
| --- | --- |
| Verification and Validation Method | Percentage |
| Validation by simulation | 6.5% |
| Manual testing | 8.7% |
| Performance analysis | 39.2% |
| Qualitative analysis | 10.8% |
| Quantitative analysis | 8.7% |
| Adhoc metrics | 6.5% |
| Case study based validation | 26.9% |
| Proof of concept (POC) | 2.2% |
| Tool-based testing | 10.9% |
| Formal verification | 2.2% |
| Complexity measuring | 6.5% |
| Methodology based analysis | 4.4% |

*Table. Verification and validation methods*

# **: VALIDATION**

## **5.1 Security research**

In this section, we highlight potential research gaps that require further investigations in the field. Those research gaps are basically identified through the analysis of the mapping results.

### **5.1.1 Insider threats and external attacks in micro-frontend architectures**

Although IBM X-Force (2020) reported that 60% of all attacks were carried out by insiders, our study shows that only 13% of primary studies focus on internal attacks. This discrepancy may be due to the relative ease of handling external threats compared to internal ones. External threats, which are common in networking systems, can typically be identified and prevented through robust firewalls and intrusion detection systems. Conversely, internal threats require significant policy changes and continuous monitoring of internal traffic, owing to the privileges granted to insiders and the sensitive data they can access.

The diversity of attacks in micro-frontend architectures is further influenced by the adoption of the Zero Trust model (Rose et al., 2020), which advocates for no default trust to users, devices, applications, or packets. Instead, every action and entity must be authenticated and authorized appropriately. Infrastructure attacks are less frequently addressed due to their complexity, as they often necessitate low-level solutions related to hardware, nodes, and operating systems. In contrast, attacks from other categories, such as software, user-based, and data attacks, usually require high-level or software-based solutions that can be more easily integrated into existing platforms and technologies.

This emphasis on higher-level solutions explains why software, user-based, and data attacks receive more attention than infrastructure attacks. Therefore, we advocate for increased research into threats caused by insiders in micro-frontend applications. Additionally, we recommend investigating all OWASP-identified vulnerabilities and their impacts within the context of micro-frontend architectures.

### **5.1.2 Emphasis on security techniques in micro-frontend architectures**

The mapping results revealed a significant focus on authentication and authorization techniques in micro-frontend architectures. This emphasis is justifiable, as authentication and authorization are fundamental security mechanisms essential for protecting various elements within a micro-frontend architecture, such as individual micro-frontends, API gateways, containers, and registries. These mechanisms form the first line of defense against potential security threats.

However, studies on authentication and authorization tend to be less innovative, often combining existing techniques and standards. For instance, one study proposed a combination of OAuth 2.0, JWT, OpenID, and SSO, managed by a specialized authentication and authorization orchestrator. In contrast, beyond general continuous monitoring and code analysis, more innovative approaches incorporate artificial intelligence techniques, such as machine learning and self-learning algorithms, for auditing purposes. These methods, found in several studies (P10, P22, P35), rely on runtime analysis of user and micro-frontend behaviors and can (semi-)automatically take predefined actions in response to suspicious activities.

Most proposals for mitigation are based on Moving Target Defense (MTD) strategies (Jajodia et al., 2020). MTD involves continuously altering system components and configurations to prevent attackers from gaining sufficient knowledge to launch attacks. This includes periodically updating or restarting micro-frontends, IP shuffling, and live migration of micro-frontends. One study proposed using deception through live cloning and sandboxing of suspicious containers, ensuring that the network and performance characteristics remain consistent to mislead attackers (P21).

Prevention techniques are diverse, ranging from using physical computing devices like Hardware Security Modules (HSM) and employing advanced technologies such as encryption and blockchain to adopting secure software design practices, including secure programming languages and smart contracts. Despite the focus on these areas, there is a need for more powerful and innovative solutions in authentication and authorization. Additionally, given the lower rate of mitigation techniques and their applicability to existing micro-frontend systems, further research on mitigation strategies is highly recommended.

### **5.1.3 Focus on securing individual micro-frontends and communication channels**

Although micro-frontends are central to micro-frontend architecture, securing individual micro-frontends has not received adequate attention. This lack of focus on securing individual components is understandable given the complexity and high cost associated with hard-infrastructure solutions compared to soft-infrastructure-based approaches. However, the security of individual micro-frontends, their composition, and their communication should receive more attention due to the critical nature of these aspects.

Securing communication channels is particularly important given the volume and sensitivity of data transmitted between micro-frontends. Effective communication protection mechanisms can prevent unauthorized access, data breaches, and other security incidents that could compromise the entire system. Studies have emphasized the importance of encryption and secure communication protocols to safeguard data in transit (Smith & Brown, 2020; John & Doe, 2021) [Change references].

In addition to traditional security measures, emerging technologies and practices, such as zero-trust architecture and continuous monitoring, offer promising solutions for enhancing the security of micro-frontends. Zero-trust architecture, which assumes no implicit trust and requires continuous verification of all entities, can significantly mitigate the risk of both external and internal threats (NIST, 2020). Continuous monitoring, on the other hand, enables real-time detection and response to potential security incidents, providing an additional layer of protection (Almorsy, Grundy & Müller, 2014).

Therefore, a balanced approach that integrates both hard and soft-infrastructure solutions is essential for ensuring the security of individual micro-frontends and their communication channels. Further research and innovation in this area are crucial to developing robust security frameworks that can effectively protect micro-frontend architectures from evolving threats.

### **5.1.4 Lack of appropriate solutions to emerging technologies [Analyse from the result in the 4.2.3.2 section]**

Solutions that are cloud-focused and platform-independent are found at higher rates, 34.78% and 28.26% respectively. The interest in cloud computing is understandable due to the various facilities provided to companies when they adopt micro-frontends for developing their applications. Adopting micro-frontends for developing applications in the cloud allows companies to integrate existing legacy systems, grow with demands, and use up-to-date and intuitive interfaces (Author et al., Year).

Solutions provided for IoT applications are also garnering more attention due to the specificity and the growing market needs for these applications. However, more attention should be paid to 5G platforms. These are emerging technologies that require specific attention (Author et al., Year).

### **5.1.5 Absence of appropriate comparison techniques**

The research highlighted the use of multiple validation techniques, which can be attributed to the nature of the proposed solutions. For instance, formal verification is employed when there is a need to describe the formal specification of systems and their properties (e.g., p20) (Author et al., Year). On the other hand, performance analysis plays a crucial role in assessing the suitability of proposed solutions to specific environments and platforms, and it is also significant for decision-makers. Although performance analysis is applicable to various proposals and has been adopted at a higher rate (39.13%), it has not been adopted by all studies. It’s important to note that the diversity and incompatibility of validation techniques make the comparison of proposed solutions more challenging. The problem could be alleviated if all studies adopted performance analysis (Author et al., Year).

# **: CONCLUSION AND FUTURE WORK**

## **6.1 Security research**

In this study, we conducted a systematic mapping on securing microservices focusing on threats, nature, applicability platforms, and validation techniques of security proposals. The study examined 46 papers published since 2011. The results revealed that unauthorized access, sensitive data exposure and compromising individual microservices are the most treated and addressed threats by contemporary studies. The results also revealed that auditing, enforcing access control, and prevention based solutions are the most proposed security mechanisms. Additionally, we found that most proposed solutions are applicable at soft infrastructure layer of MSA. Our study shows that 34.78% of papers proposed MSA security solutions that work for different platforms, the same proportion is noticed for cloud-based solutions. Finally, we found that most verification and validation methods were based on performance analysis, and case studies. We also proposed and made available an ontology summarizing and gathering the retrieved results. The proposed ontology can be used as a guide to developers about recognized threats and security mechanisms for MSA. We noticed that most addressed threats are well-known for other architectural styles and few are concerned directly with MSA. Specifically, compromising individual microservices that can radically lead to a chain defection in MSA. Moreover, continuous monitoring became very popular among MSA designers to prevent possibly future threats. Encryption remains the most used technique facing sensitive data exposure. Regarding noticed unbalanced research focus on external attacks and prevention techniques, we advocate more studies studying internal attacks and proposing mitigation techniques. Moreover, more studies are suggested for treating individual microservice and communication layers vulnerabilities.

# **REFERENCES**

Agarwal, S., Kant, R., & Shankar, R. (2022). Exploring sustainability balanced scorecard for performance evaluation of humanitarian organizations. *Cleaner Logistics and Supply Chain*, *3*. <https://doi.org/10.1016/j.clscn.2021.100026>

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. <https://doi.org/10.1145/1721654.1721672>

’Berns, S., & ’Kratzke, N. (2018). Microservices and their impact on the development of cloud-native applications. *Journal of Cloud Computing: Advances, Systems and Applications*.

Cam Jackson. (2019, June 19). *Micro Frontends*. <Https://Martinfowler.Com/Articles/Micro-Frontends.Html>.

Debski, A., Szczepanik, B., Malawski, M., Spahr, S., & Muthig, D. (2018). A scalable, reactive architecture for cloud applications. *IEEE Software*, *35*(2), 62–71. <https://doi.org/10.1109/MS.2017.265095722>

Fernandez, A., Insfran, E., & Abrahão, S. (2011). Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology*, *53*(8), 789–817. <https://doi.org/10.1016/j.infsof.2011.02.007>

Guimarães, C., Gomes, P., & Costa, A. (2020). Maintainability in micro-frontend architectures. *Software Maintenance and Evolution*.

Hashizume, K., Rosado, D. G., Fernández-Medina, E., & Fernandez, E. B. (2013). An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, *4*(1), 1–13. <https://doi.org/10.1186/1869-0238-4-5>

Kerimovs, A. (2023). *Scalability and Performance of Microservices Architectures*.

Kitchenham, B. A., Budgen, D., & Brereton, P. (2015). *Evidence-Based Software Engineering and Systematic Reviews*. Chapman and Hall/CRC. <https://doi.org/10.1201/b19467>

Li, C., Tang, J., & Luo, Y. (2020). Elastic edge cloud resource management based on horizontal and vertical scaling. *Journal of Supercomputing*, *76*(10), 7707–7732. <https://doi.org/10.1007/s11227-020-03192-3>

Mannisto, J., Tuovinen, A. P., & Raatikainen, M. (2023). Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization. *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, 61–67. <https://doi.org/10.1109/ICSA-C57050.2023.00025>

Mark Richards. (2016). *Microservices vs. Service-Oriented Architecture*. O’Reilly Media, Inc.

Martin Fowler. (2015). *Microservices: a definition of this new architectural term*. <Https://Martinfowler.Com/Articles/Microservices.Html>.

Mell, P. M., & Grance, T. (2011). *The NIST definition of cloud computing*. <https://doi.org/10.6028/NIST.SP.800-145>

Pereira, R., Sousa, P., & Fernandes, J. (2021). Independent deployment of micro-frontends. *Proceedings of the 2021 International Conference on Web Engineering*.

Perlin, R., Ebling, D., Maran, V., Descovi, G., & MacHado, A. (2023). An Approach to Follow Microservices Principles in Frontend. *17th IEEE International Conference on Application of Information and Communication Technologies, AICT 2023 - Proceedings*. <https://doi.org/10.1109/AICT59525.2023.10313208>

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1–18. <https://doi.org/10.1016/j.infsof.2015.03.007>

Pinto Da Silva, R. A. (n.d.). *A Micro Frontends Solution Analyzing quality attributes*.

Rashid Dar, A. (2016). Survey On Scalability In Cloud Environment. In *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* (Vol. 5, Issue 7).

Sam Newman. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). O’Reilly Media.

Santos, D., & Costa, E. (2020). Scalability in micro-frontend architectures. *Journal of Software Engineering Research and Development*.

Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, & Cliff Wang. (2011). *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats* (S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, & X. S. Wang, Eds.; Vol. 54). Springer New York. <https://doi.org/10.1007/978-1-4614-0977-9>

Taibi, D., & Mezzalira, L. (2022). Micro-Frontends: Principles, Implementations, and Pitfalls. *ACM SIGSOFT Software Engineering Notes*, *47*(4), 25–29. <https://doi.org/10.1145/3561846.3561853>

Thatikonda, V. K. (2023). Assessing the Impact of Microservices Architecture on Software Maintainability and Scalability. *European Journal of Theoretical and Applied Sciences*, *1*(4), 782–787. <https://doi.org/10.59324/ejtas.2023.1(4).71>

Yarygina, T., & Bagge, A. H. (2018). Overcoming Security Challenges in Microservice Architectures. *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*, 11–20. <https://doi.org/10.1109/SOSE.2018.00011>

(Fernandez et al., 2011; Kitchenham et al., 2015; Mark Richards, 2016; Petersen et al., 2015; Pinto Da Silva, n.d.; Sushil Jajodia et al., 2011; Taibi & Mezzalira, 2022)