university of groningen

faculty of science and engineering

# EXPLORING ARCHITECTURAL KNOWLEDGE ON MICROSERVICE ARCHITECTURE

Siddharth Baskaran

university of groningen

faculty of science and engineering

**University of Groningen**

# Exploring architectural knowledge on microservice architecture

**Master's Thesis**

To fulfill the requirements for the degree of
Master of Science in Computing Science
at University of Groningen under the supervision of
Prof. dr. Mohamed Soliman
and
Prof. dr. Paris Avgeriou
and
Dr. Frank Blaauw

**Siddharth Baskaran (S3922782)**

August 23, 2021

# Contents

# List of Figures

# Abbreviations

**AK** - Architecture Knowledge
**MSA** - Microservices Architecture
**SMS** - Systematic Mapping Study
**SOA** - Service Oriented Architecture

# Acknowledgments

The completion of this thesis would not have been without the support of my supervisors, Prof. dr. Mohamed Soliman, Prof. dr. Paris Avgeriou and Dr. Frank Blaauw for their incredible support during my master thesis.

I wish to acknowledge the support of my family and friends. They kept me going on, and this work would not have been possible without their input.

# Abstract

Software architects utilise a wide variety of resources to design systems. These resources are an ever-growing collection of software architecture designs based on design patterns, implementation strategies and organisation techniques. Selecting an appropriate solution from this pool of resources is a difficult task for any engineer. One approach to relieving the burden of engineers is to have up-to-date architectural knowledge, which helps engineers make informed decisions.

One of the rising architecture trends in recent times is microservice architecture (MSA). Microservice architecture comprises several small services running their process and communicating with other services to form a more extensive and complex system. This architecture has already been adopted a lot by practitioners. Although there has been an increase in MSA research in recent years, there are still many unexplored aspects of this topic, and there is no comprehensive review on this topic.

This study aims to systematically identify, analyse, and classify various sources for MSA knowledge, such as literature, blogs, and forums. We systematically apply a well-defined classification framework for categorising the multiple sources and use it to all the collected sources, including 35 literature studies, 22 blog articles and 25 forum articles. To analyse the collected data further, we annotate the data to obtain annotations and extract key information such as the common problems and solutions in MSA, common architecture design patterns in MSA, and various tools used in MSA. The results obtained can help researchers and practitioners conduct further research on the lesser analysed MSA topics, bridge the gap between research and implementation, and aid the engineers implementing MSA.

# 1 Introduction

Microservice architecture (MSA) has become the latest trend in software development. It advocates the development of software using small and autonomous services that work together. All communications between these services are through network calls, which enforces the services to be independent and avoids tight coupling issues [5]. Many advantages have been justified for MSA in both academia and the industry; for example, they are resilient, can scale well, and are easy to deploy. A large number of world-leading technology companies have been using microservices to implement complex functionalities in their products, such as Amazon, Netflix, Facebook, etc [6].

MSA as a concept arises from a broader area of Service-Oriented-Architecture (SOA). There are, however, many differences between SOA and MSA. One of the philosophies in MSA is to develop services driven by a share-nothing philosophy that supports the agile methodology, promoting quick delivery of features and maintaining isolation and autonomy. SOA supports a more share-as-much-as-you can philosophy to promote a high degree of reuse [7].

Even though there has been a significant increase in the adoption of MSA in practice, and research has identified design principles and architecture patterns for MSA, many aspects of MSA are still unclear or are unexplored. These unclear and unexplored areas of MSA make it difficult for both researchers and practitioners to clearly understand MSA implementation and reduce their potential for broader adoption. One of the important goals of this thesis is to characterize the current state of research in MSA, identify the research gaps, analyze the current industry standards, and identify the gap between research and industry. The overall process for this study is shown in Figure 1. Another important goal is to identify the key concepts being discussed by creating a coding book using annotations.

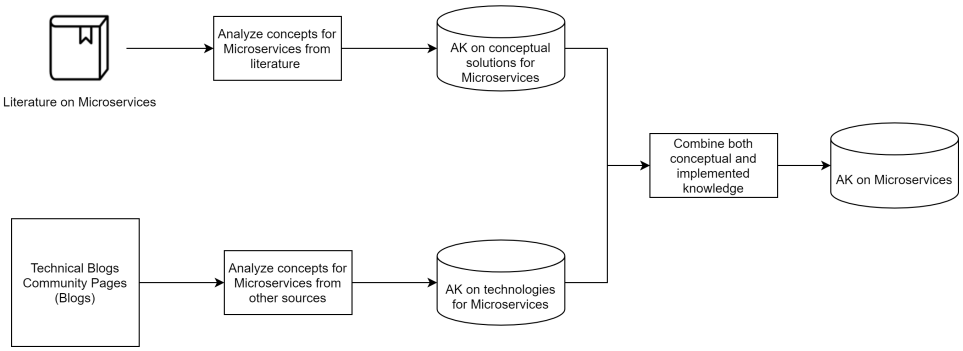


Figure 1: An overview of the study plan

To understand how MSA is being researched upon in academia, we conducted a Systematic Mapping Study (SMS) through a collection of primary studies on MSA. Our study identified, classified, and evaluated the current state of the art on architecting microservices from different perspectives. We identified 35 preliminary studies; then defined a classification framework for identifying and categorizing the research results. Apart from collecting research studies, we also collected studies from the industry in the form of blogs and forums and categorized the studies.

To further analyse the data, we created a coding book as listed in appendix A and annotated all the studies using Atlas ti. Using these annotations, we were able to: (i) Analyse a classification of the problems that practitioners may face when developing systems using MSA and the solutions to solve

these problems, (ii) a list of the most common MSA patterns, (iii) classification of the tools that are commonly used to build systems with MSA.

The rest of the thesis is organized as follows: Section 2 briefly introduces the concepts and the motivation for this study. Section 3 discusses the research method used in this study. Section 4 describes the results obtained. Section 5 discusses the results obtained. Section 6 describes the threats to validity. Section 7 discusses the related work. Section 8 discusses the future work. Section 9 concludes the study.

# 2 Background

In this section, we provide an brief overview of architecture knowledge, comparison between monolith architecture and MSA, and the research questions.

## 2.1 Architecture Knowledge (AK)

As mentioned by Paris et al. in [8] "Architectural Knowledge (AK) is defined as the integrated representation of the software architecture of a software-intensive system or family of systems along with architectural decisions and their rationale external influence and the development environment."

## 2.2 Monolith Architecture

A monolith architecture is one where an application is developed and deployed as a single application which contains all the necessary parts. A typical monolith systems consists of a UI, business logic and a data-access layer which communicates with a database as shown in Figure 2.

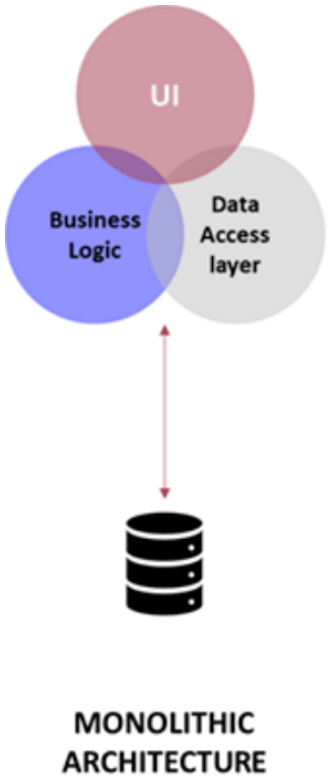


Figure 2: Monolith architecture

A monolith system is a good starting point, but as the codebase grows, the problems of a monolith system increase. Implementing new or existing features becomes difficult as the developer needs to work with a large codebase that is tightly coupled, and making changes could lead to issues in other parts of the system. It also takes a long time for new developers to get accustomed to the codebase as it's vast and often confusing. Monolith codebases are also difficult to refactor as modifying a small piece would require testing the entire system and could add to a significant amount of time, which leads to situations where refactoring is ignored. The codebase is big, and there is a higher possibility

of code duplication as developers could be unaware of similar functionality, making updates difficult as multiple parts of the code need to be changed [9].

In general, a system should be developed using monolith architecture if the application's codebase is small and remains that way in the long term if there is an abundance of developers. It is possible to develop a monolith system that has clean code and modularity; it takes a lot of effort and usually a lot of developers to maintain the codebase.

## 2.3 Microservice Architecture (MSA)

Microservices have been referred to as the solution to overcome the shortcomings of monolith architecture. MSA is an approach to developing a single application using a suite of small services, each running its own process and having its own resources. Microservices are developed based on business requirements, due to which they are often independently deployable. Because of their size, they are easier to maintain and scale vertically. They are also more fault-tolerant as the failure of one service will not cause the entire application to be unavailable. Therefore, MSA allows developers to develop systems that are modular, fault-tolerant, and easy to scale. Figure 3 shows the structure of a microservice architecture.

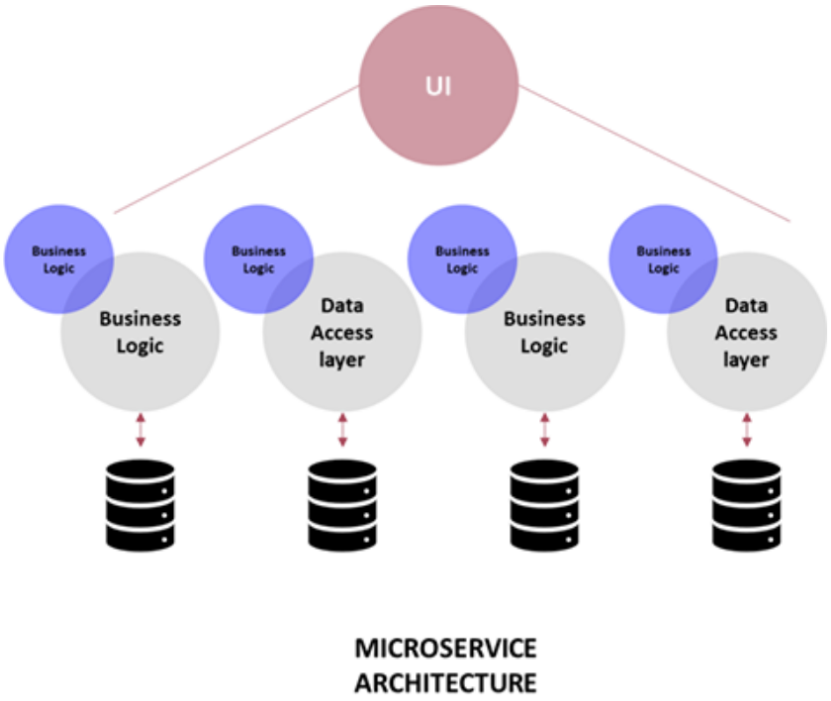


Figure 3: Microservice architecture

## 2.4 Motivation

Due to the fast adoption of MSA, there has been an increasing amount of studies, blogs, and forums published in the past few years. Currently, many of the aspects of MSA are unclear and information scattered across various platforms. Analyzing the existing data will make it easier for practitioners to adopt MSA quickly and help researchers identify further research opportunities. Table 1 details the research questions and the rationale behind selecting these questions.

| Category 1: Classification and mapping of research materials | | |
|---|---|---|
| ID # | Research question | Rationale |
| RQ 1 | What are the publication trends of research studies about MSA? | This RQ is formulated to collect data on the types of research avenues, publication trends on MSA. The answer to this RQ will provide information on the trends in MSA in research and prominent venues and help characterize the intensity of scientific interest in MSA. |
| RQ 2 | What are the primary areas of focus of research on MSA in academia? | The answer to this RQ will help establish the baseline for the systematic analysis of MSA in academia. By answering this RQ, we can set a solid foundation for classifying various MSA concepts from the obtained research articles and analyze the research gaps in MSA in academia. |
| RQ 3 | What are the primary areas of focus of research on MSA in industry? | The answer to this RQ is similar to RQ 2. This RQ helps set a foundation for classifying the various MSA concepts in the industry; we can also analyze the gaps and differences between the industry and academia from the classification. |
| Category 2: Architecture Knowledge identified | | |
| ID # | Research question | Rationale |
| RQ 4 | What are the common MSA AK identified in each of the sources? | As we have analyzed different sources to identify MSA AK, by answering this question, we can understand more clearly how each source contributed towards the various categories of classifications. |
| RQ 5 | What are the common problems and its solution reported in implementing MSA? | Implementing MSA comes with its own set of problems. The answer to this RQ will help us classify the most common problems one faces when using MSA and the solutions one could use to solve said problems. A solution to the problems identified could be a design solution, a pattern, or a tool. |
| RQ 5.1 | What are the common MSA design patterns used? | Several design patterns exist when trying to implement an MSA. The answer to this RQ will help in identifying the most common architecture pattern one uses for MSA. |

| | | |
|---|---|---|
| RQ 5.2 | What are the commonly used technologies when using MSA? | Implementing MSA requires various tools to be used together. The answer to this RQ will help identify the most common technologies one could use while implementing a system using MSA. |

Table 1: Research questions and Rationale

# 3 Study Design

Our study follows a two-phase design. In the first phase, we focused on scientific literature, and in the second phase, we focused on other sources such as blogs and forums. Due to the different nature of the two phases, the study design varied significantly for both phases. Hence, this section has been divided to represent the study design for each of the different sources.

## 3.1 Study design of literature sources

Our literature study set out to identify various trends, problems, patterns, and methods in the context of MSA. The best technique to achieve this was to perform a Systematic Mapping Study (SMS). SMS are designed to analyze a broad research area and to determine if research evidence exists on a topic and provide an indication of the quantity of the evidence [10]. Moreover, SMS offers a decisive method to perform a systematic and objective method to identify and classify what evidence is available in a specific research area [11]. Using SMS makes it possible to perform an extensive literature review and frame the crucial research questions.

Figures 4- 6 give an overview of the search and selection process which are described in more detail below.

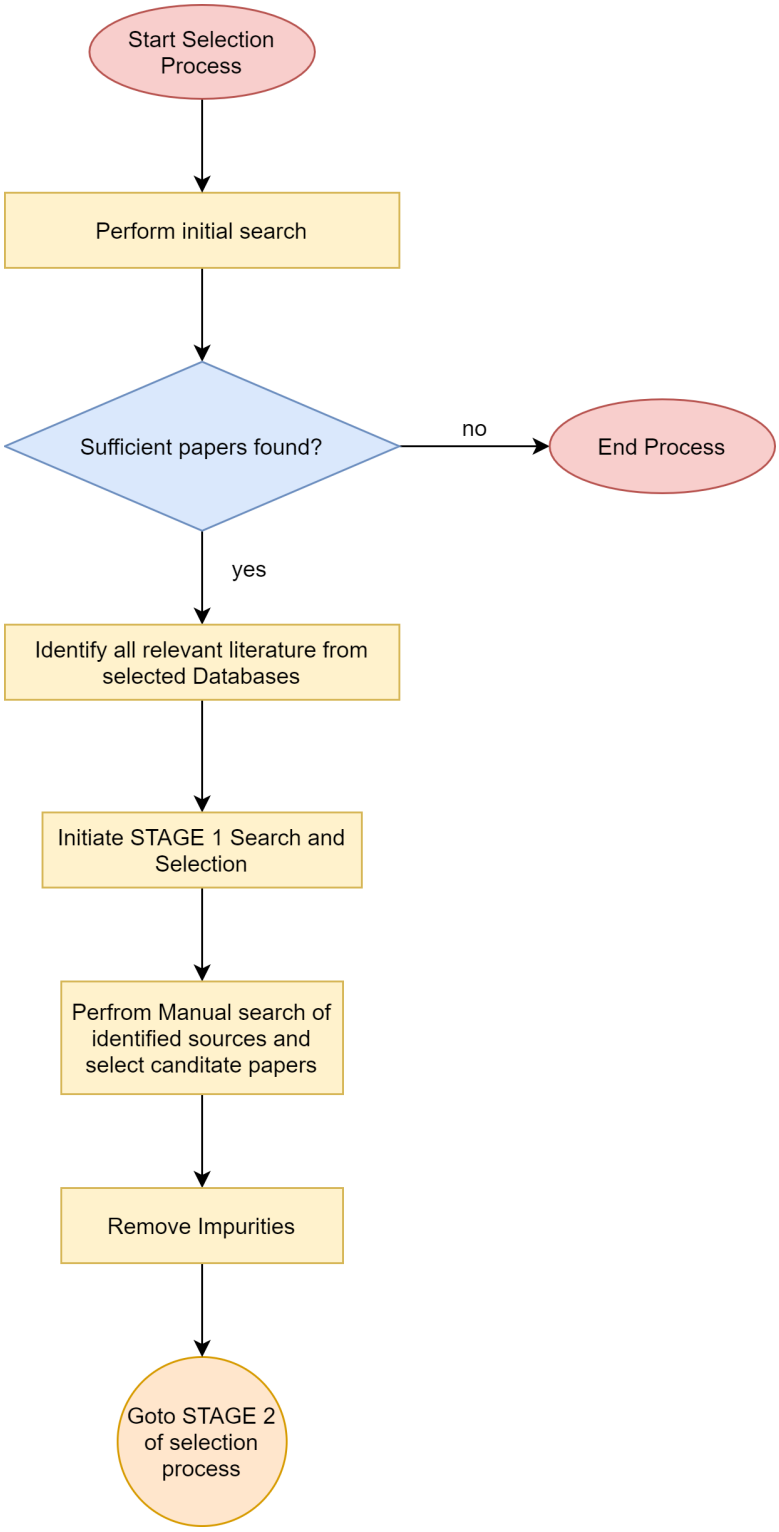


Figure 4: Initial search and stage 1 search process

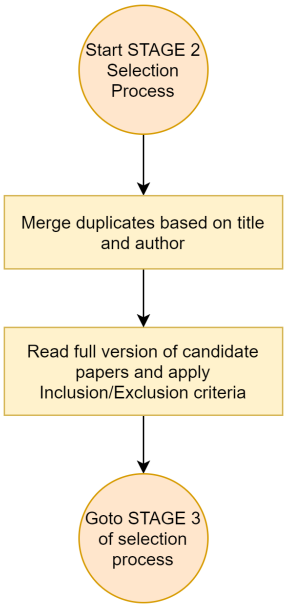


Figure 5: Stage 2: Selection process

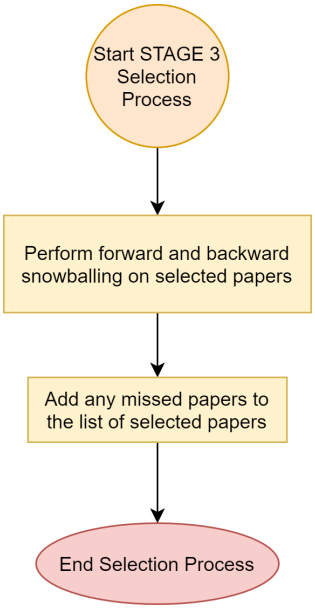


Figure 6: Stage 3: Snowballing and validation process

### 3.1.1 Stage 1: Sources search and selection

B. Kitchenham et al. [12] proposed the concept of PICO (Population, Intervention, Comparison and Outcomes) as a method to identify and formulate research strings for an SMS based study. PICO is defined as:

- *Population:* Population in Software Engineering is defined as the specific software engineering role, category of software engineer, an application area or an industry group [12]. In this particular study, population is micro-service architecture studies.
- *Intervention:* Intervention in Software Engineering is defined as the software methodology, tool, technology, or procedure [12]. In this study, as we concentrate on architecture knowledge on a broad scale, we do not have any intervention.
- *Comparison:* What is the main alternative to compare the obtained results? Comparison is not part of this study as we are not focusing on comparative studies but on architecture knowledge which can contain a wide variety of knowledge.
- *Outcome:* What is the quantitative outcome from this study?. In this particular study, we create a categorization framework and classify studies depending on the category.

Based on the discussion in [12] and [13], the following databases were used to find relevant literature: IEEE Xplore [1], ACM Digital Library [2], Springer [3], and ScienceDirect [4].

The reason why these specific databases were selected are (i) They are recognized as reputable sources for performing SMS on Software Engineering topics as discussed in [12] and [13], (ii) Easy access and a wide selection of literature, and (iii) Easy to export and use the data for analysis.

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://dl.acm.org/
[3] https://www.springer.com/gp
[4] https://www.sciencedirect.com/

One of the key factors that we considered for selection was to collect literature between 2016 and 2021, and the rationale behind this was twofold:

(i) As shown in Figure 7, we can observe that there has been a significant increase in the search for the keyword microservices from 2016, and hence this was selected as the lower boundary.

(ii) The upper boundary was set to 2021 to get up to date information.

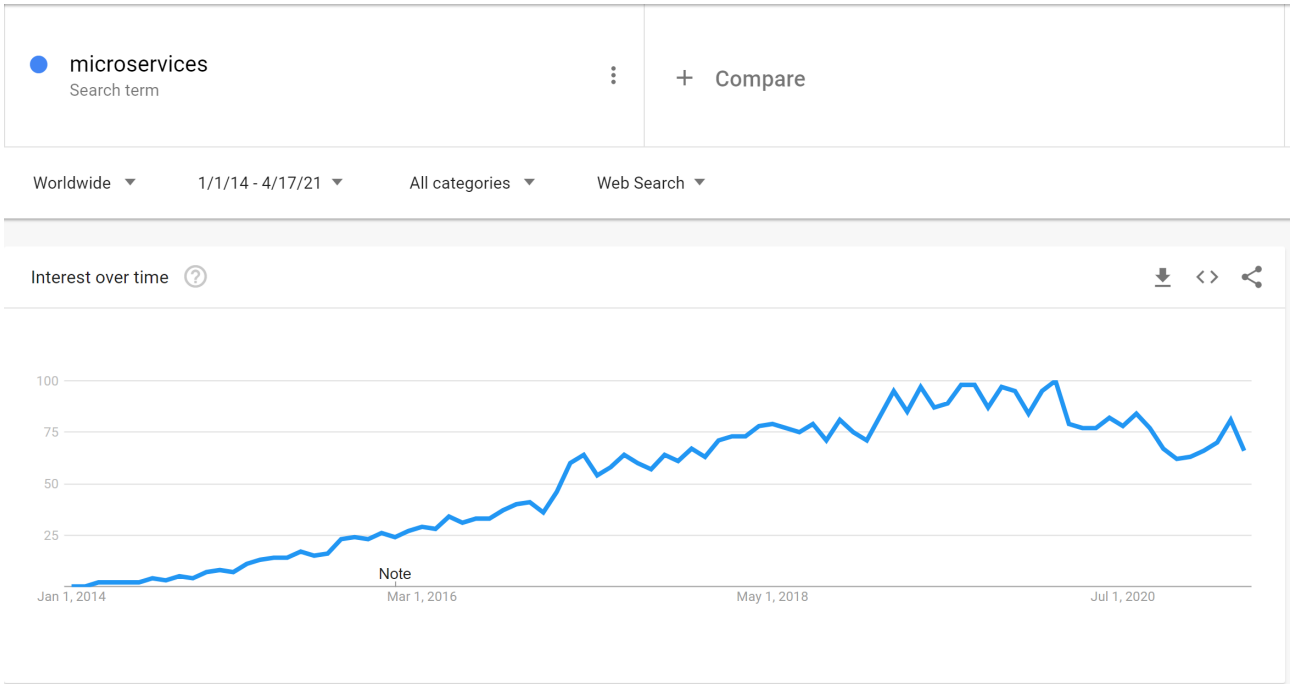


Figure 7: Google search trends

To start with the data collection, as suggested in [12], an automatic search was performed on the selected databases using the following search string:

Original string: `((architect* OR design* OR system OR knowledge) AND (microservices"))`

The search string was applied only to the title, abstract, and list of keywords to maintain consistency and make it easy to filter the results. Table 2shows the search string used on each Database and indicates the search string's area.

While performing a test with the original test string, we observed that some well-known studies were not retrieved. We realised that some of the studies used a different format to identify microservices such as *microservice* or *micro-services*; hence the search string was modified to the one represented as the **final string**.

The following factors were eventually considered when formulating the final search string:

- The research objectives and RQs.

- Different ways to write the terms. (e.g. microservice, micro-service).
- Different combinations of terms. (e.g. architect design, architect system, architect knowledge).
- The limitations of the search engine in the databases.

Final string: `((architect* OR design* OR system OR knowledge) AND (microservi* OR micro-servi*))`

| Strings | | |
|---|---|---|
| String: ((architect* OR design* OR system OR knowledge) AND (microservi* OR micro-servi*")) | | |
| Databases | | |
| Database | Link | Targeted search area |
| IEEE Xplore | https://ieeexplore.ieee.org/Xplore/home.jsp | Title, Abstract and Keywords |
| ACM Digital Library | https://dl.acm.org/ | Title, Abstract and Keywords |
| Springer | https://www.springer.com/gp | Title, Abstract and Keywords |
| ScienceDirect | https://www.sciencedirect.com/ | Title, Abstract and Keywords |

Table 2: Search strings and databases used in this systematic mapping study

### 3.1.2 Stage 2: Selection and filtering process

Once the initial set of papers were collected, an impurity removal was required to remove all data that was not literature, such as posters, short-text, opinions pieces. These were filtered out in the search stage to avoid unnecessary data.

Once the impurities were removed, the next step was to merge and remove duplicates. It was necessary to remove duplicates and merge papers present in multiple databases to clean the dataset further.

Once all the necessary literature was collected using the aforementioned techniques, it was necessary to filter out a lot of the literature collected using very well-defined selection criteria. This filtering was required as we only required relevant literature and since the main idea was to build architecture knowledge on microservices. These criteria were chosen based on [10].

Table 3 represents the list of inclusion criterion's. These inclusion criteria helped narrow down the original data collected and only look for relevant literature that consists of topics relevant to MSA.

| # | Criteria |
|---|---|
| IC-1 | Studies involving architecture solution, methods, or implementation strategies specifically for microservices, case studies and interviews |
| IC-2 | Studies were published between 2016 and 2021 |
| IC-3 | Studies are in English |

Table 3: Inclusion criteria

Table 4 represents the list of exclusion criteria. These exclusion criteria were designed to limit the scope of the research and to disregard studies that do not relate to microservices. All criteria were added to narrow down the search to specific topics that involve topics necessary for MSA.

| # | Criteria |
|---|---|
| EC-1 | Studies involving microservices but ones that concentrated on specific applications, maintenance, brief conceptual ideas and tools used in MSA |
| EC-2 | Studies involving abstracts, posters, short-papers, technical writings |
| EC-3 | Studies that were duplicates of other studies |
| EC-4 | Studies that involved books or grey literature |
| EC-5 | Studies where full-text was not available |

Table 4: Exclusion criteria

### 3.1.3 Stage 3: Snowballing and validation process

To support the manual search, we then performed an automated search based on citation analysis; this technique is known as snowballing.
The search process has various steps that help gather and filter out the literature required for the study. Once the original data set was obtained, the next step was to perform forward/backward snowballing; this was done to enlarge the final dataset with relevant data.

Backward snowballing means using the reference list to identify new papers to include [1]. The following papers were considered for Backward snowballing.

1. P. D. Francesco, I. Malavolta and P. Lago, ”Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption [7].

2. Paolo Di Francesco, Patricia Lago, Ivano Malavolta, Architecting with microservices: A systematic mapping study [14].

3. Schröer, Christoph and Kruse, Felix and Marx Gómez, Jorge, ”A Qualitative Literature Review on Microservices Identification Approaches” [15].

We also performed forward snowballing (i.e., collecting those studies citing the selected studies [1], which was performed using Google Scholar [5]) on the same set of papers.

Figure 8 represents the process of snowballing and the various steps involved in both forward and backwards snowballing.

[5] https://scholar.google.com/

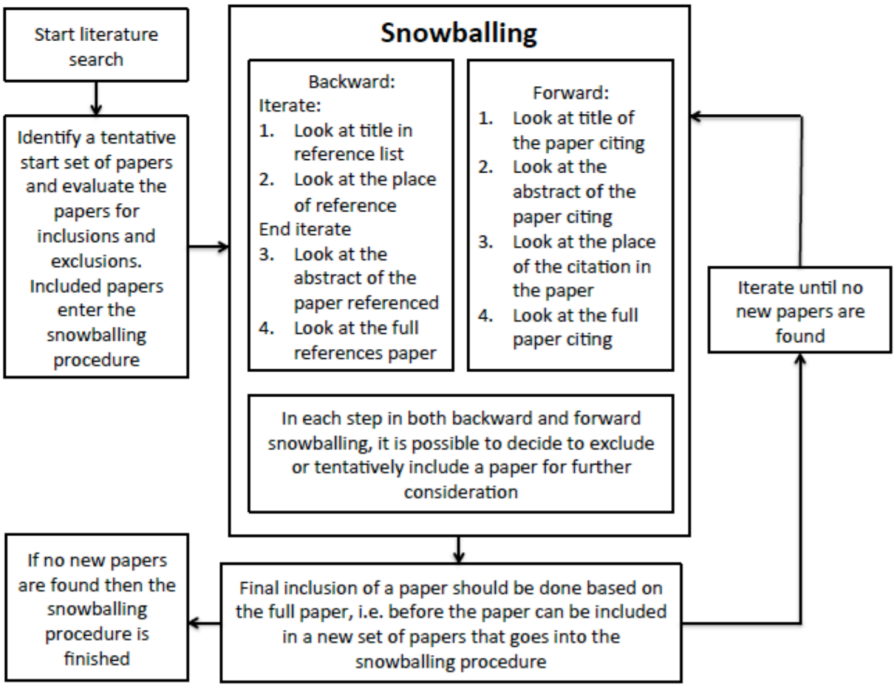


Figure 8: Snowballing process [1]

Figure 9 shows an accurate representation of the number of papers included and excluded in each stage.

While doing the full-text reading of the initially selected 89 documents, it became apparent that some more articles need to be removed as they were not in the scope based on the inclusion and exclusion criteria. The selected 24 articles were used to conduct forward snowballing, and three articles were selected to perform backward snowballing, which led to another ten studies being added.

A couple of papers were marked as borderline papers deemed relevant during the inclusion and exclusion based on title and abstract and were excluded later in the process. On re-reviewing those papers at the end of the process, one study matched the criteria and was added to the included list bringing the total number of studies selected to 35.

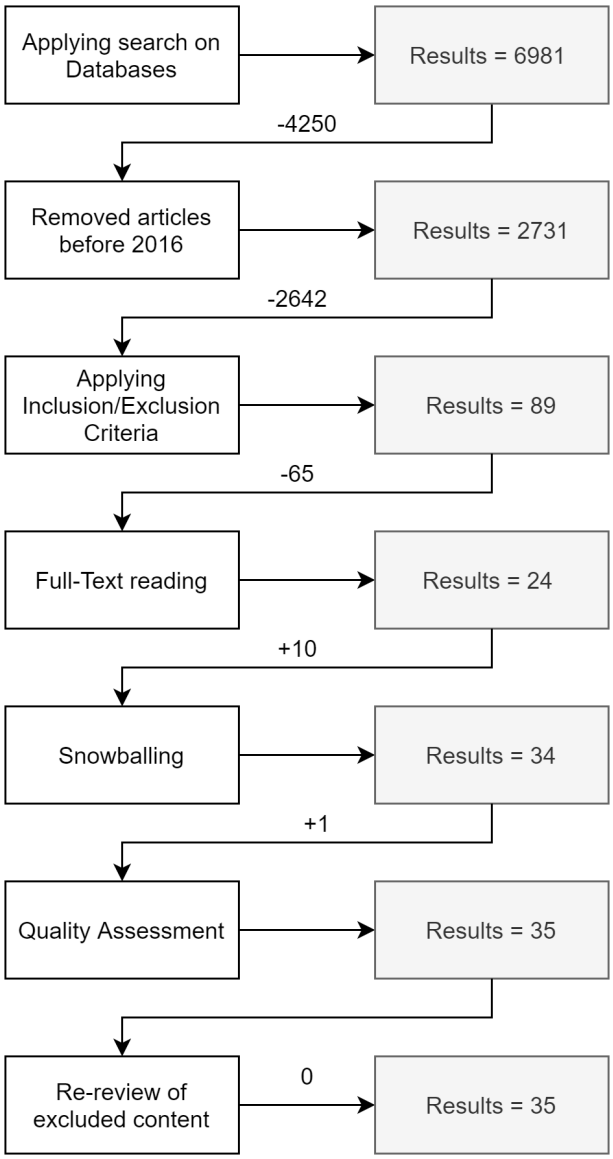


Figure 9: Overview of selection and filtering of literature

### 3.1.4 Data extraction

For answering the research questions, we defined a set of data items. In this activity, we (i) create a classification framework and (ii) collect data for each primary study. In order to answer the research questions and to develop a strong data extraction process, and easily classify the data, we systematically designed a structured classification framework based on [7].

**Publication trends (RQ1)** - We considered the parameters to collect data about publication trends: publication year, publication venue (e.g., conference, journal, and workshops), and publication database. Table 5 represents the data items that are extracted from the studies. Data items (D1–D4) are used to extract the general information of the selected studies, and the rest data items (D4–D) are used to answer the RQs as discussed in 2.4. The relationship between the data items and RQs is also described in Table 5. All the studies were collated and stored in Google Sheets to be further synthesized.

| Code | Data Item | Description | Relevant RQ |
|---|---|---|---|
| D1 | Study ID | The ID of the study (P-Papers, B-Blogs, F-Forums) | N/A |
| D2 | Study Title | The title of the study | N/A |
| D3 | Author(s) List | The authors of the particular study | N/A |
| D4 | Publication Venue | The name of the publishing venue | N/A |
| D5 | Year | The year in which the study was published | RQ 1 |
| D6 | Publication Type | Journal, conference or workshop | RQ 1 |
| D7 | Database | The database which the study is from | RQ 1 |
| D8 | Research strategy | The strategy identified in the study (Literature only) | RQ 2 |
| D9 | Research contribution | The main research contribution of the study (Literature only) | RQ 2 |
| D10 | Research focus | The main research focus of the study | RQ 2 & RQ 3 |
| D11 | Software lifecycle scope | The phase of software lifecycle in the study (Literature only) | RQ 2 |
| D12 | Architecting activities | The main architecting activity of the study (Literature only) | RQ 2 |
| D13 | Quality attributes | The main quality attribute of the study (Literature only) | RQ 2 |

Table 5: Data items extracted

**Focus of research in academia (RQ2)** - To answer this specific question, we performed a systematic process known as *keywording*, as discussed in [7]. Keywording is a technique that is used to develop a classification scheme quickly. The process involves studying the abstracts to highlight the essential keywords of the study, which are required to gain insight into the study. Keywording requires having a clear understanding of the end goal and extensive knowledge of the topic. Using this process, a list of keywords were generated related to MSA and were collated to provide info on the various contributions from each study. However, in some cases, to ensure reliability, some keywords were also generated by examining the studies' introduction and conclusion. Once this list was generated, it was used to ascertain the categories used for the map of the study [11]. Figure 10 represents the steps involved in the keywording process.

The following were the steps involved in the keywording process:

1. *Identify keywords* - The keywords were collected by reading the full text of each study. Once all the keywords were identified, only keywords that identified the emerging context, nature, and research contribution on microservices architecture were retained.

2. *Cluster keywords and form categories* - The collected keywords were clustered according to emerging categories as mentioned in [7]. The output of this stage is the initial version of the classification framework.

3. *Extract data from study and refinement* - Once the initial keywords had been identified, each primary study is analysed to (i) classified into a category identified by the keywording technique and (ii) collect any additional information, and if the identified keyword is relevant, then the classification framework is updated.

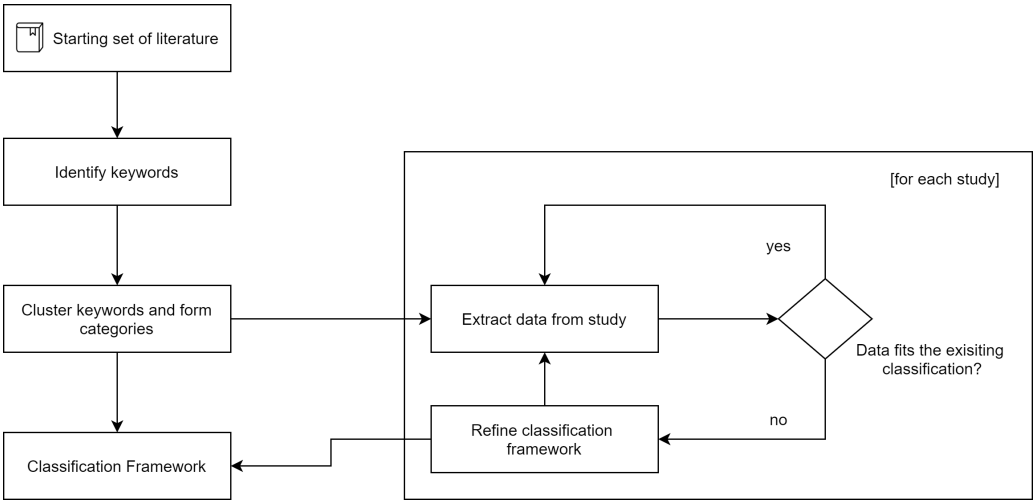


Figure 10: Keywording process

### 3.1.5 Data synthesis

This particular step involves collating and summarising the data that is extracted in the Data Extraction step 3.1.4, where the primary goal is understanding, analysing, and classifying the studies on MSA.

To further explore the data and perform extensive data analysis, `Atlas.ti` was used. The use of Atlas.ti for coding was to perform a cyclic and iterative data analysis which would have been difficult to perform using a spreadsheet, word processing application, or a note-taking tool [16]. The entire process of performing Data synthesis using Atlas.ti has been described in Section 3.3.

## 3.2 Study design of other sources

This subsection describes the study design for the other sources such as blogs and forums.

### 3.2.1 Blogs

To collect blogs for the study, we used the study performed by Mohamed Soliman et al. [17] as the source. The study collected AK using a traditional search engine, Google. Fifty-three developers were tasked to collect data and classify the results depending on the relevance, with 5 having a high relevance where the result discusses a similar problem to the topic and contains useful information and 1 having a low relevance, where the result contains information that is only remotely relevant to the goal.

Based on the search methodology described, we decided to look for studies that contained the keyword microservice in the URL or the title. We restricted the search to a relevance score of 3 to prevent collecting data relevant to the study.

Once the studies were filtered, we removed every link that was not a blog article, performed a short reading of the results and accepted blogs relevant to our research, and discarded the irrelevant blogs. Overall we found 21 blogs that matched our requirements.

Figure 11 shows the overview of the selection and filtering process of the blog articles.

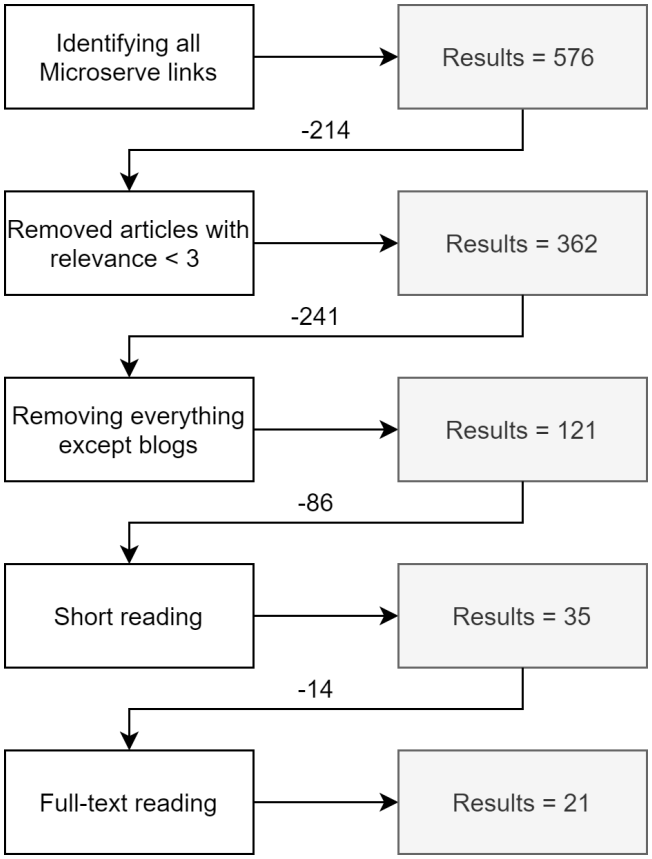


Figure 11: Overview of selection and filtering of blogs

### 3.2.2 Forums

We decided to look into StackOverflow (SO) [6] for collecting data from forums as it is one of the largest forums to discuss software-related topics in a question-answer format. It also supports features as tagging questions based on topics; it also supports users to add details to the questions; the quality of the questions and answers are also ensured by allowing users to vote on the platform. Moreover, as SO is the most used forum for software engineering, it is constantly updated with the latest information from its users. All these features make it an excellent source for collecting AK.

Based on the features offered by SO, we deiced to look at questions tagged '*microservices*' and to filter the results by restricting it to display only questions with more than 35 votes. Once the data was collected, we read all the results and picked only the relevant results, resulting in 25 studies being collected. Figure 12 shows the overview of the selection and filtering process of the forum articles.

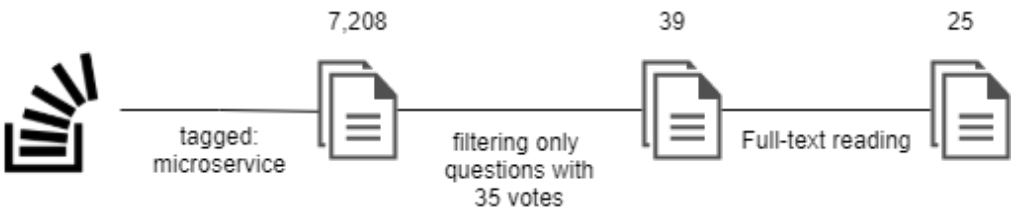


Figure 12: Overview of selection and filtering of forums

[6] https://stackoverflow.com/

## 3.3 Data synthesis using Atlas.ti

To further synthesise the data, Atlas.ti[7] was used, which is a tool that helps researchers systematically analyse complex data. The tool allows users to create a coding book and annotate data using this coding book. For this particular analysis, an inductive approach was used.

Before we could annotate the data, a coding book had to be created. To create this coding book, a technique known as noticing and collecting was used [18]. Using this technique, we were able to develop the codes inductively. Atlas.ti provides a feature, 'apply codes', which lets you select data and apply one of the codes created. We identified 14 different codes through this method and defined these codes to annotate the studies using this as our base. The coding book can be found in Appendix A. Once the coding book was defined, all 81 studies were analysed and we found 872 annotations overall.

[7] https://atlasti.com/

# 4 Results

This section reports the results of the study after analysing and extracting information from the studies. We report the publication distribution in Section 4.1. We discuss the various themes and categorization of literature studies in Section 4.2. Section 4.3 discusses the themes identified in the industry. In Section 4.4 we discuss the various Architecture Knowledge identified. The problems and solutions identified are reported in Section 4.5. We report the various patterns identified in Section 4.6. Section 4.7 discusses the various tools and technologies associated with MSA.

## 4.1 Publication trends (RQ1)

The distribution of publications per year is an integral part of this study as it provides information regarding published studies for each year on MSA. The data obtained through this analysis provides a general overview of the research in MSA.

Figure 13 shows the number of mapping studies identified within the years 2016 and 2021. The interest in MSA has been approximately the same across the period and shows a slight dip in 2020 could be due to the lesser number of conferences taking place in 2020 due to the COVID-19 global pandemic.

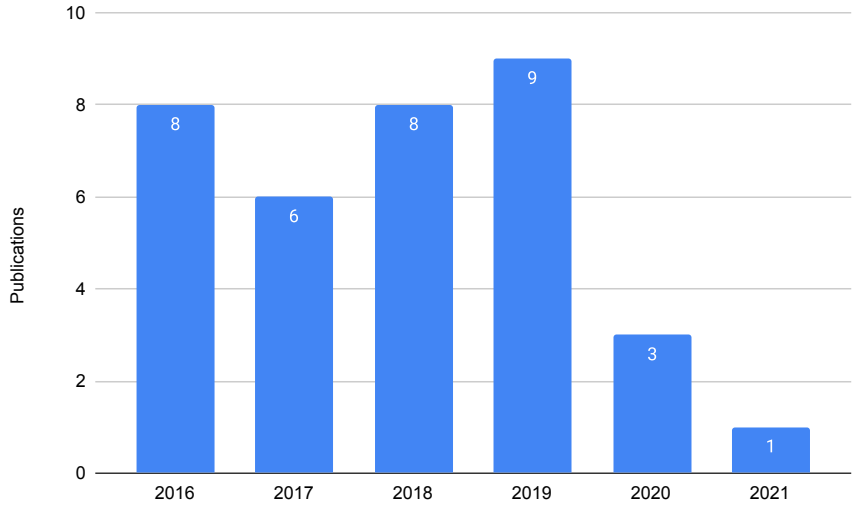


Figure 13: Publications per year

Figure 14 shows the number of publications per year. Each year is represented using three coloured bars, and each coloured bar represents the study type published from 2016 to 2021. As shown in Figure 13, 8 studies were published in 2016, 6 were published in 2017, 8 were published in 2018, 9 were published in 2019, 3 were published in 2020, and finally 1 was published in 2021.

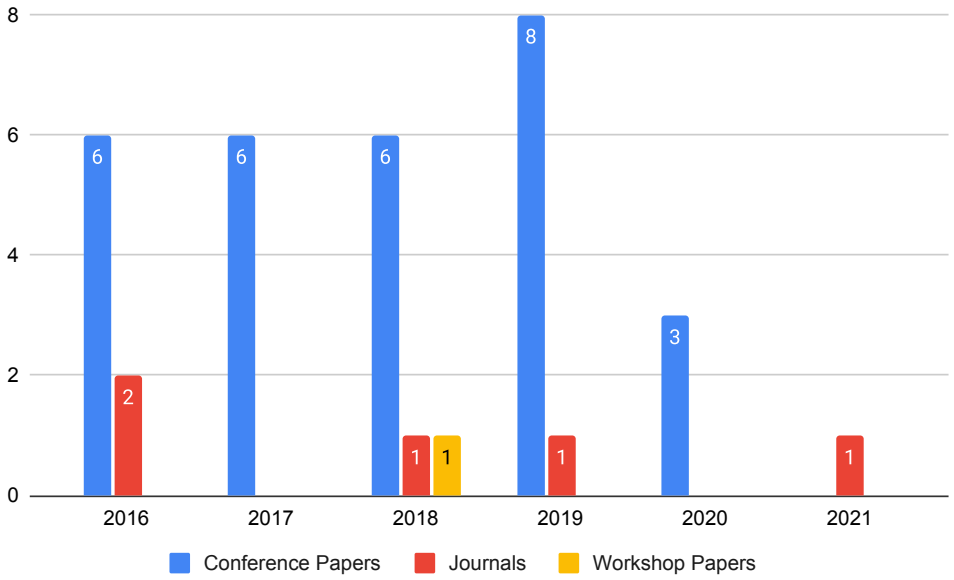


Figure 14: Publication distribution over publications per year

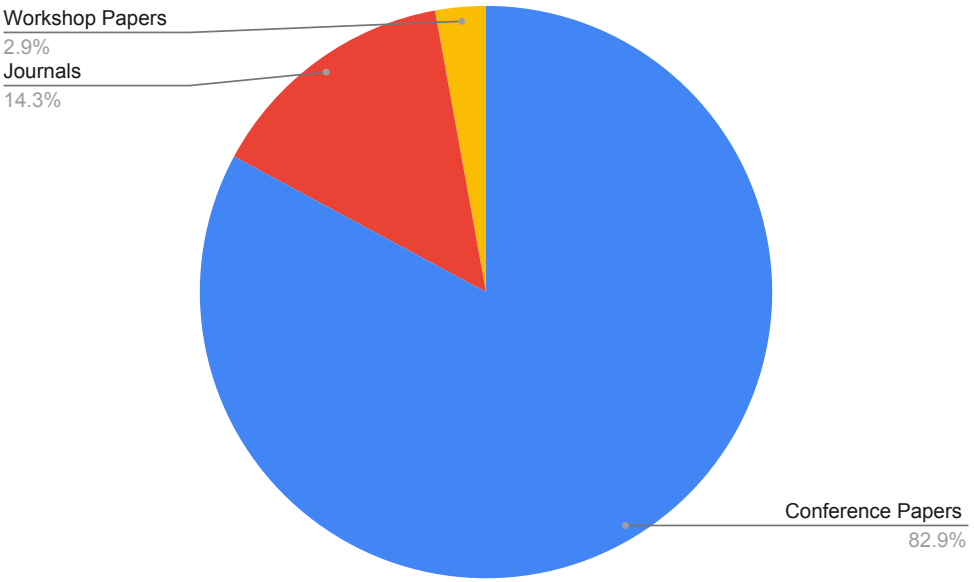


Figure 15: Overview of publication type

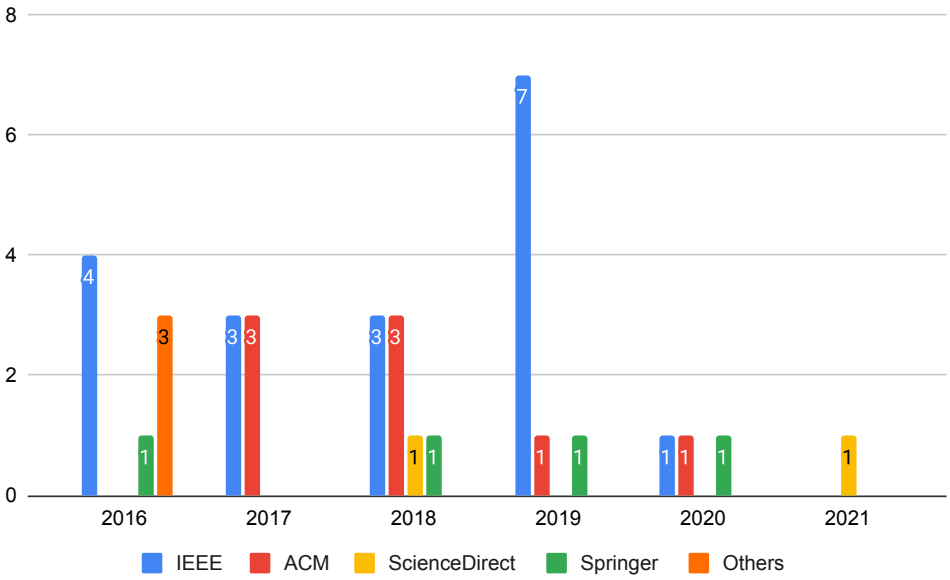


Figure 16: Publications over the years per database

Figures 14, 15 and 16 show the distribution over publications and per database.

## 4.2 Primary focus of research in academia (RQ2)

After collecting all relevant literature, we created two categories to analyse further the focus of research, namely (i) scope of research and (ii) support for architecting based on [7].

### 4.2.1 Scope of research

**Research strategies** - The categories for the research strategies have been obtained from the list provided by Wieringa et al. [19]. The categories in this list is widely used and is also used as the base in various mapping studies [7].

Table 6 represents all the research strategies identified from all the collected literature. From the table it is evident that *evaluation research* is the most common category (17/35). Evaluation research is mainly referred to the investigation of problems in software engineering or an implementation strategy. The knowledge provided in the evaluation research papers primarily evaluate using some form of evaluation strategy; casual properties could be studied empirically using case studies, experiments, etc., and logical properties by mathematics [19]. *Solution proposal* and *opinion studies* are the second most common category (7/35). Solution proposals are those in which the paper proposes a new solution and argues for its relevance. A proof of concept is generally an excellent companion to back a solution proposal [19]. These solutions are usually new strategies researchers propose in tackling MSA, as MSA is a fairly new concept. There are always new ways to solve problems, which gives researchers opportunities to propose new solutions.Opinion papers are generally based on the opinion of the author(s) on how to propose a solution, the merits and demerits of a system, and it usually provokes a discussion [19].

| Research Strategy | No. of studies | Studies |
|---|---|---|
| Evaluation research | 17 | P5, P6, P11, P16, P17, P18, P19, P22, P24, P25, P26, P27, P28, P30, P32, P33, P35 |
| Solution Proposal | 7 | P8, P9, P10, P14, P15, P29, P34 |
| Opinion Study | 7 | P1, P3, P4, P12, P13, P20, P21 |
| Experience Paper | 3 | P2, P7, P23 |
| Conceptual | 1 | P31 |

Table 6: Research strategies distribution

**Research contribution** - From Table 7 we can identify the main research contribution as *Design Patterns* (22/35). The number of studies concentrating on *applications* and *reference architecture* are low compared to design patterns.

| Research contribution | No. of studies | Studies |
|---|---|---|
| Design Patterns | 22 | P1, P2, P3, P6, P7, P9, P10, P11, P12, P13, P14, P16, P17 P20, P22, P24, P25, P26, P27, P28, P30, P35 |
| Application | 7 | P15, P18, P19, P31, P32, P33, P34 |
| Reference Architecture | 6 | P4, P5, P8, P21, P23, P29 |

Table 7: Research contribution distribution

**Research focus** - Research focus represents the main focus area of the study. From table 8, we can observe that the predominant focus is on *cloud* (9/35), *development* (9/35), and *system quality* (8/35). Migration takes up only a tiny part of the research (5/35). It is surprising to note that there are also a few studies where MSA is being considered for *IoT* related topics (4/35).

| Core research area | No. of studies | Studies |
|---|---|---|
| Cloud | 9 | P15, P22, P24, P30, P31, P32, P33, P34, P35 |
| Development | 9 | P1, P3, P4, P18, P19, P20, P21, P23, P26 |
| System quality | 8 | P5, P6, P9, P10, P12, P16, P17, P27 |
| Migration | 5 | P2, P7, P11, P13, P28 |
| IoT | 4 | P8, P14, P25, P29 |

Table 8: Research focus distribution

**Software lifecycle scope** - From Table 9 the most predominant software lifecycle phase is *software design* (23/35). Surprisingly there is a wide gap between software design and implementation phase (12/35).

| Software Lifecycle Scope | No. of studies | Studies |
|---|---|---|
| Software Design | 23 | P2, P3, P5, P6, P7, P10, P11, P13, P15, P16, P17, P19, P20, P21, P22, P23, P26, P27, P28, P29, P30, P31, P35 |
| Implementation | 12 | P1, P4, P8, P9, P12, P14, P18, P24, P25, P32, P33, P34 |

Table 9: Software lifecycle scope distribution

### 4.2.2 Support for architecting

**Architecting activities** - The classifications have been based on the study done in [7]. As shown in Table 10, the most common architecting activities are *architectural analysis* (11/35), *architecture evaluation* (9/35), *architecture understanding* (9/35). This indicates that researchers mostly try to analyse or understand existing MSA rather than implementing new architecture (5/35).

| Architecting activities | No. of studies | Studies |
|---|---|---|
| Architectural Analysis | 11 | P2, P6, P7, P8, P9, P20, P22, P30, P32, P34, P35 |
| Architecture Evaluation | 9 | P4, P5, P10, P11, P18, P19, P26, P27, P28 |
| Architecture Understanding | 9 | P1, P3, P12, P13, P16, P21, P23, P29, P31, P33 |
| Architectural Implementation | 5 | P14, P15, P17, P24, P25 |
| Architectural Description | 1 | P29 |

Table 10: Architecting activities distribution

**Quality attributes** - The study looked at various quality attributes and table 11 shows the distribution of quality attributes. *Performance* is the most discussed quality attribute (14/35). *Functional usability* and *usability* are almost similar with (9/35) and (8/35) respectively and this is primarily due to the fact that MSA is really good with being used for specific use cases and it tends to help developers build systems using different technologies.

| Compatibility | No. of studies | Studies |
|---|---|---|
| Performance | 14 | P4, P5, P6, P9, P10, P11, P14, P15, P22, P26, P27, P32, P34, P35 |
| Functional suitability | 9 | P1, P17, P18, P19, P20, P23, P25, P29, P33 |
| Usability | 8 | P2, P7, P8, P12, P13, P16, P28, P30 |
| Reliability | 3 | P24, P29, P31 |
| Maintainability | 1 | P3 |

Table 11: Quality attributes distribution

## 4.3 Primary focus of research in industry (RQ 3)

After collecting all the studies associated with the industry, such as blogs and forums, we wanted to analyze the studies further; hence we created one category to identify the main focus of the studies. Unlike the categorization of papers, which has multiple classifications, for studies based on the industry, we have one classification as studies from the industry only focus on one topic in their article. We do not have multiple aspects as that of literature.

Table 12 shows the various focus of study of blogs and forums. The most common focus is on patterns (18/45), followed by development (12/45) and system design (11/25). The least discussed

topic is architecture solution (4/45). One crucial observation is that most of the patterns are from blogs, and most of the development studies are forum articles.

| Main focus of study | No. of studies | Studies |
|---|---|---|
| Patterns | 18 | B1, B2, B4, B5, B6, B7, B8, B9, B14, B16, B17, B18, B20, B27, B28, F48, F50, F54 |
| Development | 12 | F21, F24, F30, F31, F38, F41, F42, F44, F45, F46, F47, F53 |
| Architecture Solution | 4 | F22, F23, F25, F32 |
| System Design | 11 | B3, B10, B11, B12, B13, F33, F40, F43, F49, F50, F52 |

Table 12: Main focus of study

## 4.4 Architecture Knowledge identified (RQ4)

Based on the categories defined in Appendix A we annotated all the documents collected and found 872 annotations across all categories, which are represented in Figure 17. Figure 18 shows the occurrences of AK concepts present in each of the AK sources. Based on Figure 18, we can observe the following:

- Architecture Impact is the most discussed AK in the literature, whereas it's discussed a lot lesser in blogs. That is primarily because most of the literature studies are based on comparing various aspects of microservices; hence, they discuss the different impacts of different solutions and systems. In blogs and forums, the discussion is primarily on the implementation of different architectures, and hence there is less discussion of impact but more about how architecture is implemented.

- Architecture Patterns are heavily discussed in blogs, whereas in literature and forums, it is significantly lower as blogs, in general, contain discussions on various implementation techniques. These tend to be more or less based on different patterns available to implement certain features in MSA. When dealing with direct implementation strategies, blogs tend to discuss patterns and how to use them, which is usually not the case in literature, where it is more about discussing using an existing architecture or the impact of certain patterns and alternatives available.

- Architecture Design Rules are discussed a lot in both literature and forums, which is primarily because, in forums, most of the questions regarding MSA are on how to implement certain aspects of the architecture or questions about development using an MSA and the answers to those are more or less in terms of design suggestions or specific strategies in using certain design patterns. Similarly, in literature, when comparison studies are done or when a case study is explored, the results derived are based on different designs to achieve different results.

- Use cases are found almost equally across all 3 sources as studies discuss a concept and an example, making it easier to understand.

- Technology-based solutions are more discussed in literature than blogs and forums, primarily due to the presence of case studies in literature. A lot of literature contains case studies, and these generally contain information on various technologies used to implement MSA in various systems.

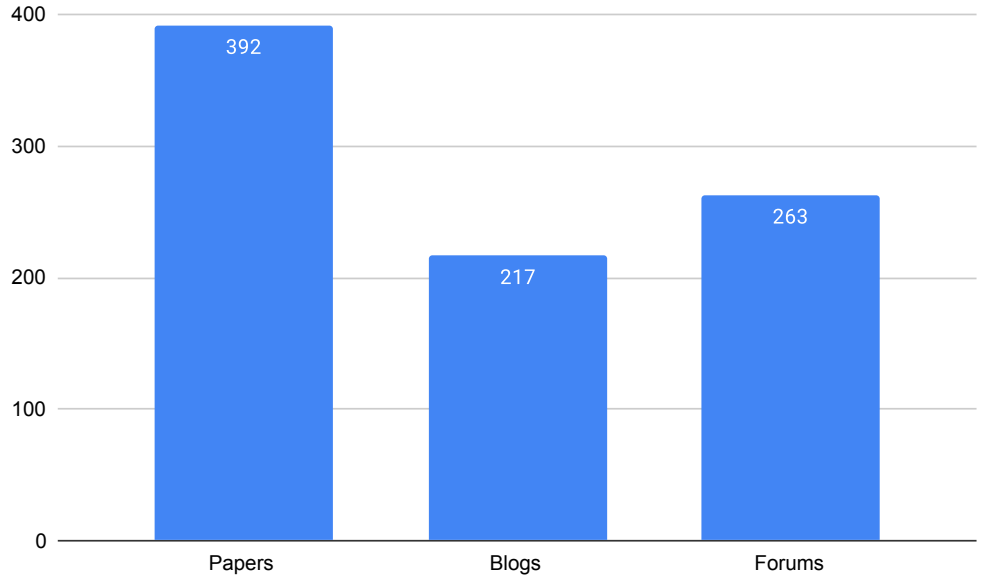


Figure 17: Total annotation per source

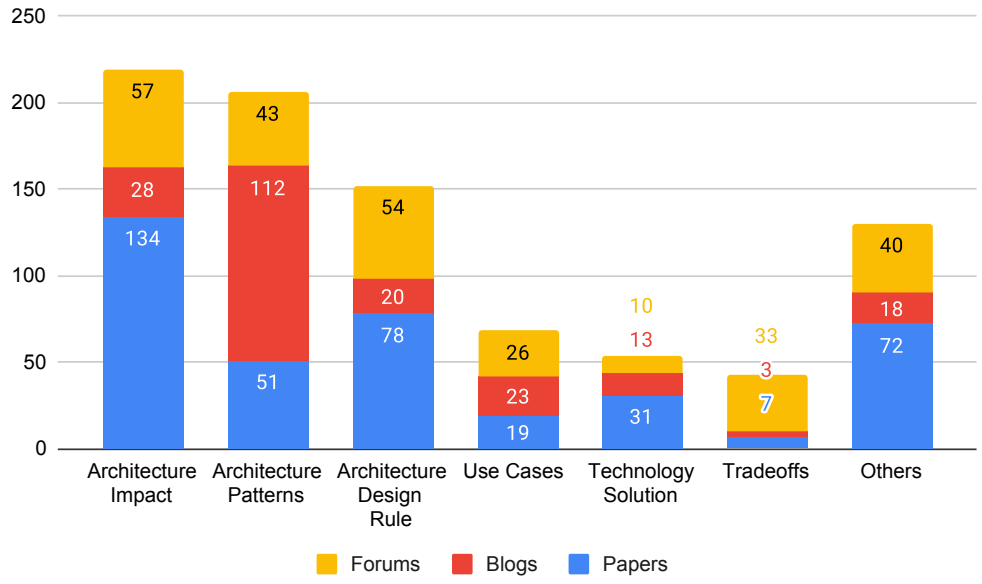


Figure 18: Distribution of annotations

## 4.5 Problems and solutions identified (RQ 5)

This section presents the problems and solutions identified in the selected studies related to MSA. We decided to analyse the annotations tagged Architecture Impact further as it is the most annotated category as discussed in subsection 4.4. To analyse the annotations, we applied thematic analysis [20] on the annotated data and identified six categories to classify the problems and solutions. The six classified themes are (i) Communication in MSA, (ii) Handling data in MSA, (iii) Availability in MSA, (iv) MSA Performance, (v) Design strategies in MSA and (vi) Fault tolerance in MSA.

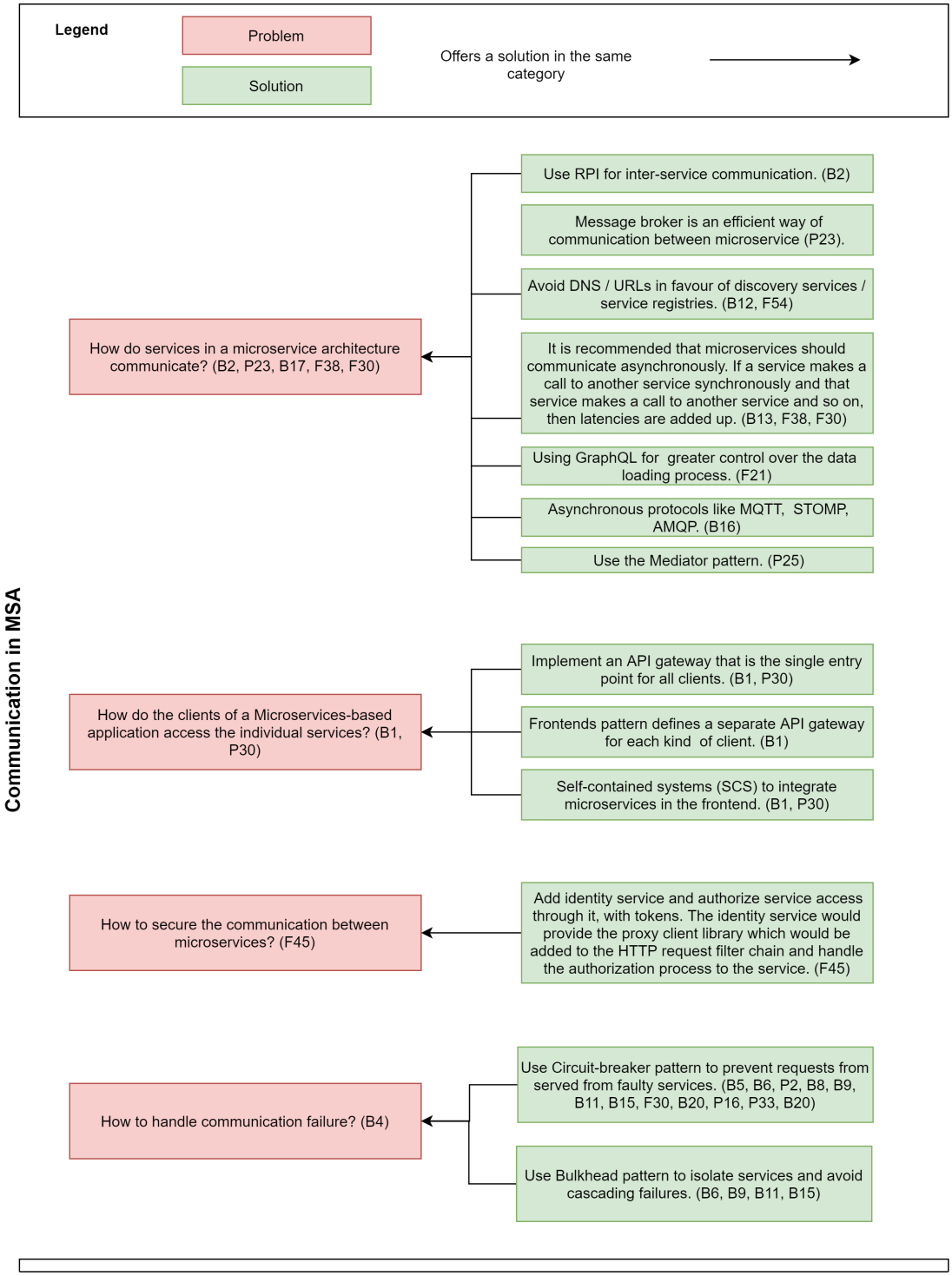


Figure 19: An overview of the identified problems and solution in MSA

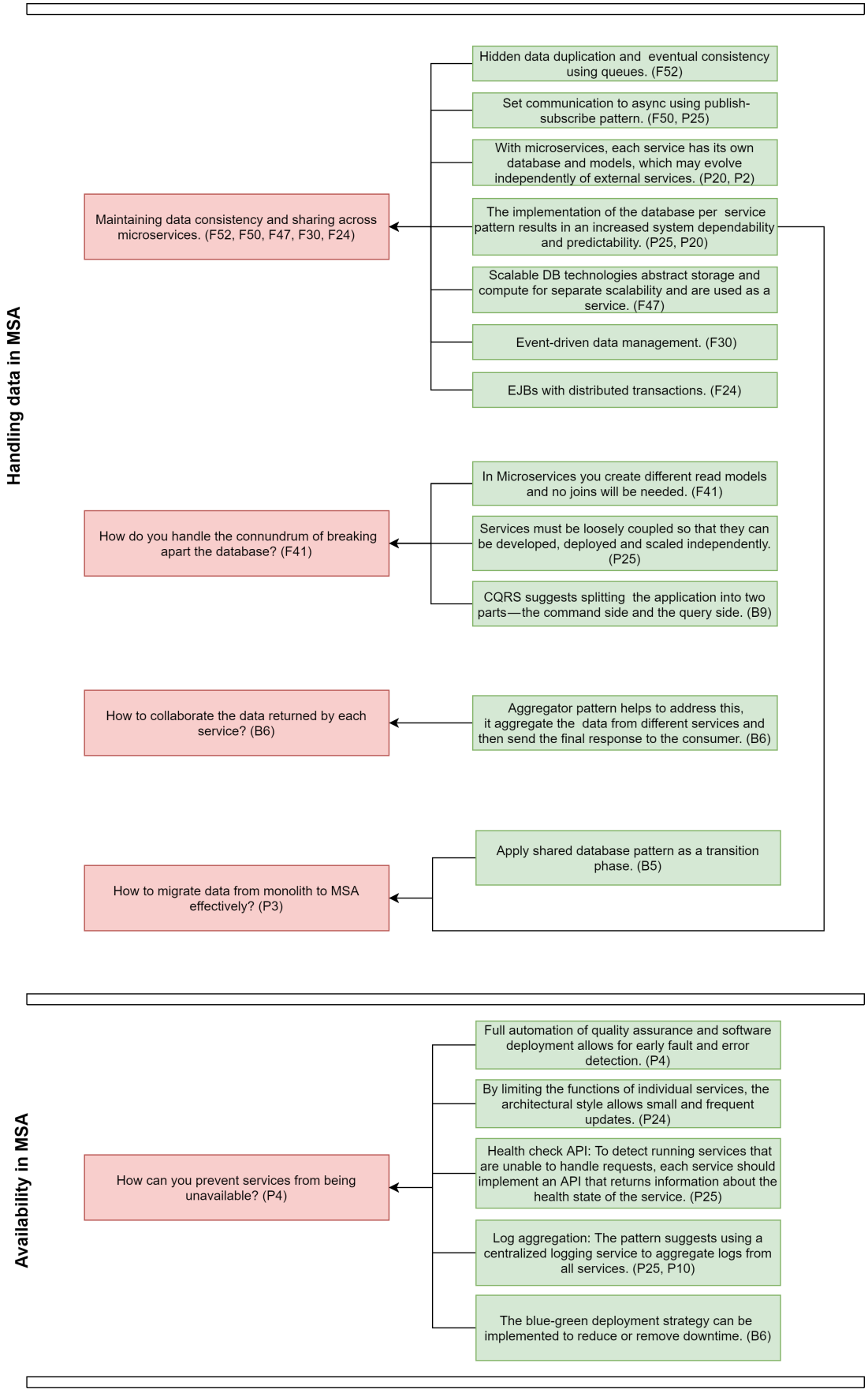


Figure 20: An overview of the identified problems and solution in MSA (contd.)

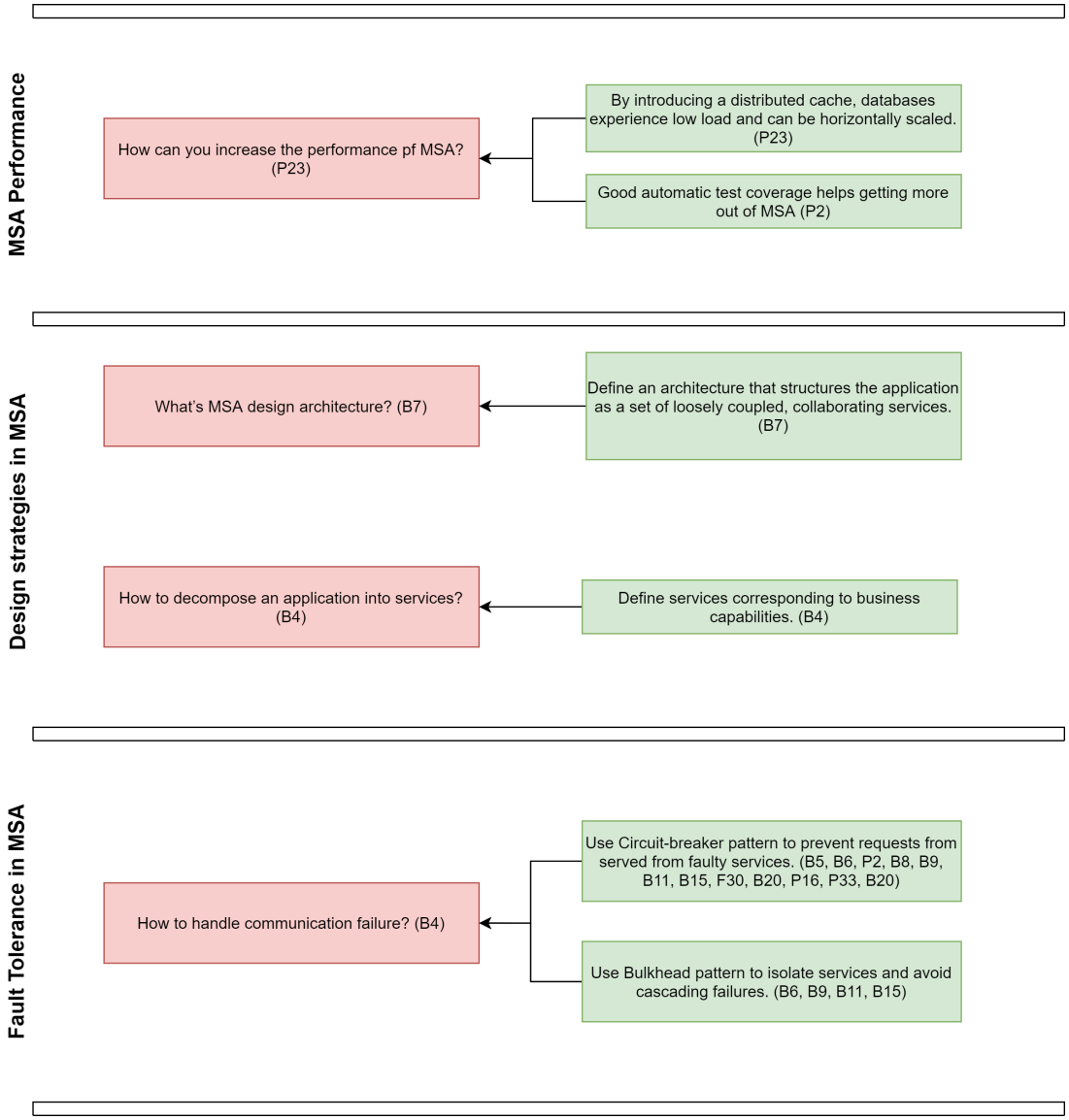


Figure 21: An overview of the identified problems and solution in MSA (contd.)

We were able to identify 13 problems overall and 36 solutions. Figures 19- 21 shows the mapping between the issues identified and their corresponding solutions. The classification indicates that the majority of the problems are in 2 categories, *communication in MSA* and *handling data in MSA*. We briefly discuss these problems and solutions below.

### 4.5.1 Communication in MSA

This category reports the problems and solutions related to communication in MSA. As MSA comprises multiple services, efficient communication is key to have a functioning system. The most common solution from studies (B13, F38, F30) is to implement a sound communication system effectively is to design the system to use asynchronous communication where systems can communicate with multiple services together when compared to synchronous where the calls to each service can add up in terms of latency and slow down the system. Asynchronous protocols such as MQTT, STOMP and AMQP are also suggested in study (B16), and RPI protocol is suggested in study (B2). Study (P23) suggests using a message broker such as Apache Kafka or ActiveMQ to control the flow of

requests; similarly, one could use GraphQL to make requests to control the requests being made as study (F21) suggests.

Another challenge with communication in MSA is how clients will interact with the system as there are multiple services, and a client cannot connect with each of them separately. Studies (B1 and P30) suggest implementing an API gateway pattern that provides a single point of entry to all clients to communicate. Study (B1) also suggests using the Backends For Frontends pattern where each client application has its server-side component. Studies (B1 and P30) also suggest using the Self-contained systems (SCS) pattern, which is alternate to microservices where each service is autonomous.

Another challenge one faces with communication is that, as there are multiple services and a lot of communication happens between them, how does one secure these services? One can solve this by adding an identify service such as JSON web tokens (JWTs) and add it to every request header and perform an authorization based on that as mentioned in study (F45).

### 4.5.2 Handling data in MSA

This category reports the problems and solutions related to data handling in MSA. Similarly, how communication is a challenge in MSA, handle data faces a similar problem as there are multiple services and data needs to be shared across multiple services. The biggest challenge is how to handle the consistency of data and share the data across services. One of the solutions is to implement a database per service pattern where each service has its own database and provides an API for access, as mentioned in studies (P25, P20). Another solution to solve consistency is to use the publish/subscribe pattern as suggested in studies (F50, P25) which makes the communication async where applications publish data to an intermediary broker rather than communicating directly with databases. Study (F52) suggests using queues to push data to multiple databases to achieve consistency; if you are using Java, then EJBs have been suggested as a solution (F24) which makes the system loosely coupled where one can use SOA than POJOs. Study (F30) suggests using event-driven data management where each service publishes an event whenever it updates its data. Other services subscribe to events. When an event is received, a service updates its data.

A second challenge with data management is how one knows where to break down a database when designing a system. One solution is to use a CQRS pattern where the application is split into two parts, the command side, and the query side. It is used instead of CRUD in cases where there are high reads and writes, and by splitting the application, each can be scaled independently as per study (B9). Another solution is to design the system loosely so that each component can be developed, scaled, and deployed independently. Handling data becomes easier when each component can be maintained loosely as per study (P25). Finally, study (F41) suggests developing multiple reads models to prevent the use of joins as joins between multiple tables are cumbersome and use many resources; creating separate models can reduce the system load.

A third challenge is if one application has multiple databases, how does one collaborate the results returned by each service. The solution for this has been proposed in study (B6), which suggests you use the aggregator pattern, which aggregates all the data from different services and then sends the final response to the consumer.

### 4.5.3 Availability in MSA

This category reports the problems and solutions related to availability in MSA. As multiple services are involved in MSA, it is important to keep services available to prevent downtimes. One of the common ways suggested in studies (P25, P10) is to have a log aggregation service like ELK, or Grafana, to keep track of all the logs in a central place and send out alerts when something is wrong so that it can be fixed immediately. Another study (P4) suggests using automatic tests for software development and deployment pipelines to catch faults early and prevent them before deployments. Study (P25) suggests using a health check API, which can be used to detect services that are not running, and the API should be able to return information about the state of the service, which can be shown on a dashboard for ease of use. Study (B6) suggests using a Blue-green deployment strategy where you have two systems, namely blue and green, and when a new service is to be deployed, the deployment is first done. Then the system is switched from blue to green to ensure that we do not have a downtime of the system.

### 4.5.4 MSA Performance

This category reports the problems and solutions related to performance in MSA. Since MSA is considered scalable, one needs to design the system keeping in mind the system's performance. As suggested in study (P23), one way to improve MSA's performance is to implement distributed cache databases that experience low load and can be scaled horizontally easily. As communication between services is one of the most resource-intensive tasks, using a cache database would increase the performance. Another technique suggested in the study (P2) is to have a good automatic testing coverage which helps to catch errors, and run regular tests to stress test the system to improve performance.

### 4.5.5 Design strategies in MSA

This category reports the problems and solutions involved in designing an MSA. One of the ways to design an effective MSA, according to study (B7), is to define an architecture that an application is loosely coupled, the benefits of such a system have been discussed earlier, and to have a system that collaborates so that have small services and a great communication.

Another challenge is how to decompose an application into services effectively. Study (B4) suggests that as a rule of thumb, a service should be decomposed so that most new and changed requirements only affect a single service, which can be achieved by following the decompose by business capacity pattern. One should also develop a service in such a way that only one team can handle the functionality.

### 4.5.6 Fault tolerance in MSA

This category reports the problems and solutions involved in developing a fault-tolerant system using MSA. One of the common reasons why MSA could fail is a failure in communication between services in a distributed setting, leading to system downtime. To prevent such a failure, multiple studies (B5, B6, P2, B8, B9, B11, B15, F30, B20, P16, P33, B20) suggest the use of circuit breaker pattern, which helps the system identify services that are not functioning and return errors instead of wasting resources and waiting for the calls to timeout hence leading to high system load. Another popular solution, as suggested by multiple studies (B6, B9, B11, B15), is to use the bulk-head pattern when designing MSA. When designing a system, one should split the application into multiple components,

and resources should be isolated so that the failure of one component does not affect the other. Using the bulk-head pattern, one can allocate specific resources for certain components so that we do not consume all the application resources unnecessarily, hence improving the system's fault tolerance.

## 4.6 Architecture patterns identified (RQ5.1)

This section presents the architecture patterns identified in the selected studies related to MSA. We decided to analyse the annotations tagged architecture patterns further as it is the second most annotated category as discussed in subsection 4.4. Similar to analysing architecture impact, we applied thematic analysis [20] on the annotated data and identified seven categories to classify the architecture patterns. The 7 classified themes are: (i) Communication, (ii) Fault-tolerant, (iii) Data Management, (iv) System Design, (v) Deployment, (vi) Migration and (vii) State management.

| Category | Identified Patterns | Study ID |
|---|---|---|
| Communication | Circuit Breaker Pattern | B5, B6, P2, B8, B9, B11, B15, F30, B20, P16, P33, B20 |
| | API Gateway Pattern | B5, P25, B8, B9, F23, B20, P17, P20, P6, P33, B20 |
| | Service Discovery Pattern | B5, B6, P25, B9, F54, B20, P17, P20, P33, B20 |
| | Saga Pattern | B5, B6, P25, B9, B16, P10, F48, B28 |
| | Rest HTTP Pattern | P3, P9, B7, B17, F53, B20 |
| | Asynchronous Messaging Design Pattern | B8, B17, B18, P6, B20 |
| | Distributed Tracing | B5, P25, B20, B28 |
| | Chained Microservice Pattern | B6, B8, B9, P6 |
| | Aggregrator Pattern | B5, B6, B9 |
| | Branch Pattern | B6, B8, B9 |
| | Client-Side UI Composition Pattern | B6, P25, B9 |
| | Message Broker Pattern | P23, B17, B20 |
| | Synchronous Messaging Design Pattern | B11, B16, B18 |
| | Proxy Pattern | B9, F23 |
| | Gateway Routing Pattern | B6, B9 |
| | Publisher-subscription Pattern | P25, P34 |
| | Service Mesh Pattern | P20, B20, B28 |
| | Request/Response Pattern | P34, B28 |
| | Domain Events Pattern | F23 |

| | | |
|---|---|---|
| | Adapter Microservice Pattern | P17 |
| | Event Notification Pattern | P17 |
| | Backend for Frontend Pattern | P17 |
| | Service Registry Pattern | P20 |
| | Event Driven Pattern | B20 |
| | Single Receiver | B27 |
| Fault Tolerant | Bulk Head Pattern | B6, B9, B11, B15 |
| | Observability Pattern | B6, P25, B9 |
| | Load Balancing Pattern | P17, P33 |
| | Log Aggregation Pattern | B5 |
| | Timeout Pattern | B11 |
| Data Management | CQRS | B5, B6, B7, B8, F41, B28 |
| | Database Per Service | B5, B6, B9, P20 |
| | Transactional Outbox Pattern | B16, F32, B28 |
| | 2-Phase Commit | F48, F30, B28 |
| System Design | Decomposition Pattern | B6, B8 |
| | Modularity Pattern | P14 |
| | Sidecar Pattern | B6 |
| | Externalized Configuration Pattern | P17 |
| Deployment | Blue-Green Deployment Pattern | B5, B6, B9 |
| Migration | Strangler Pattern | B5, B6, B9 |
| State Management | Event Sourcing Pattern | B6, B9, B16, B20, B28 |

Table 13: Architecture Patterns identified from selected studies

We were able to identify 41 architecture patterns. Table 13 shows the list of all the patterns, and it's corresponding categories. From the table, we can observe that most of the patterns are based on communication, similar to what we observed in problems and solutions where the most common problem was communication. Fault tolerance, system design, and data management also contribute to the list. We will briefly discuss some of the important patterns in detail below.

### 4.6.1 Circuit breaker pattern

The circuit breaker pattern is the most mentioned pattern across all studies with 12 occurrences. When using MSA, services tend to communicate with one another to share data. At times when one service

synchronously invokes another service, there is always a chance that one of the services is unavailable, or it could be taking a long time to respond to requests making it highly unusable [21].

The circuit breaker pattern is designed in such a way that you wrap a protected function call in a circuit breaker object, which monitors failure and when the failure reaches a certain threshold, the circuit trips hence causing all further calls to the circuit breaker to return an error without the internal call being made. In most cases, one should set up a monitor alert if the circuit breaker trips. There is a timeout that is usually set before the request can be retried once again; if the request post timeout is successful, the circuit breaker will resume operations once again [2].

Figure 22 shows the three states of the circuit breaker pattern, which are: (i) closed state, (ii) open state, and (iii) half-open state. The circuit breaker does not allow requests to pass through, counts the number of requests received, and checks for the threshold value in the open state. The circuit breaker allows requests to pass through in the closed state and checks for any requests that fail. In the half-open state, the circuit breaker only allows a few requests to pass through. If these requests are successful, the circuit breaker will go back to the "Closed" state. However, if any request fails again, it goes back to the "Open" state.

This pattern is used to prevent services from accessing remote services or a shared resource if the operation is highly likely to fail. Circuit breakers should be an integral part of systems for monitoring. Any change in breaker state should be logged, and further investigation should be performed as it often depicts a good source of warnings about deeper troubles in the environment. The circuit breaker pattern can be implemented with Netflix Hystrix.

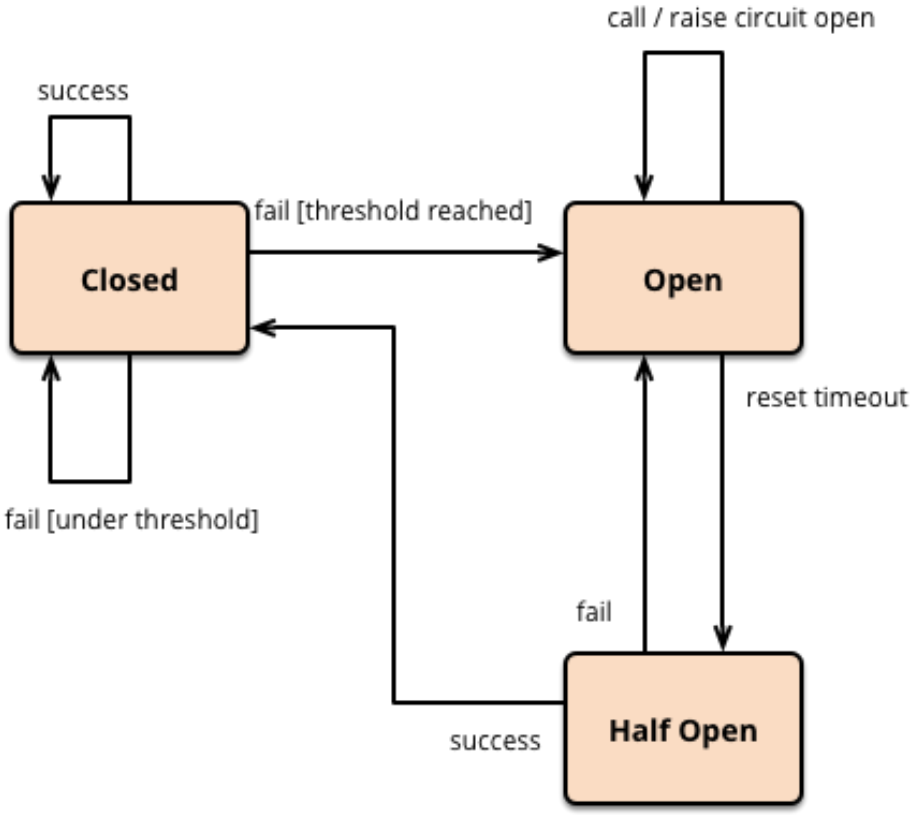


Figure 22: Circuit breaker pattern [2]

### 4.6.2 API Gateway Pattern

The API Gateway pattern is the second most mentioned pattern with 11 occurrences. As microservices are made up of multiple services, it is key to have one service that clients can use to communicate. Since different clients require different data and clients cannot access each service, an API Gateway handles requests in one of two following ways: simply sending the request to the appropriate service or routing to more services [22].

API Gateways can be used for other purposes as well: [23].

- Authentication and Security - API gateways allow developers to encapsulate inner functions and prevent unauthorized access.
- API Monitoring and insights - APIs can be monitored to measure how long API takes to respond, and they can be used to generate reports and identify the behaviour of the system.
- Rate limiting - APIs can be regulated to prevent services from being overwhelmed.
- Load Balancer - API gateway can be used as used a load balancer to provide scalability.

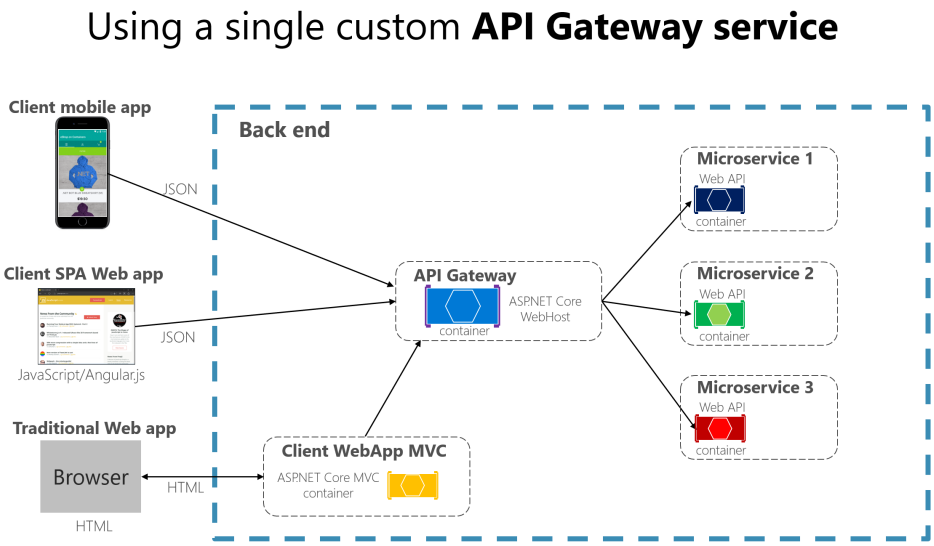


Figure 23: API Gateway pattern [3]

### 4.6.3 Service discovery pattern

The Service discovery pattern is the third most mentioned pattern with ten occurrences and related to communication. In monolith applications, services communicate with one another through language-level methods. In traditional systems, many services run on fixed ports and communicate with REST calls. However, such an implementation is not possible in microservices-based systems as services and ports change constantly depending on the scaling, which can be solved using the service discovery pattern in microservices, which can be used in conjecture with the API gateway pattern. Service discovery help instances of various services to adapt and distribute the load the microservices [24].

Service discovery can be used as a client-side pattern where the client is responsible for determining the network locations of available service instances. The client queries a service registry to find an

available service. Netflix OSS is an example of client-side discovery pattern [4]. Figure 24 shows the client-side service discovery pattern.

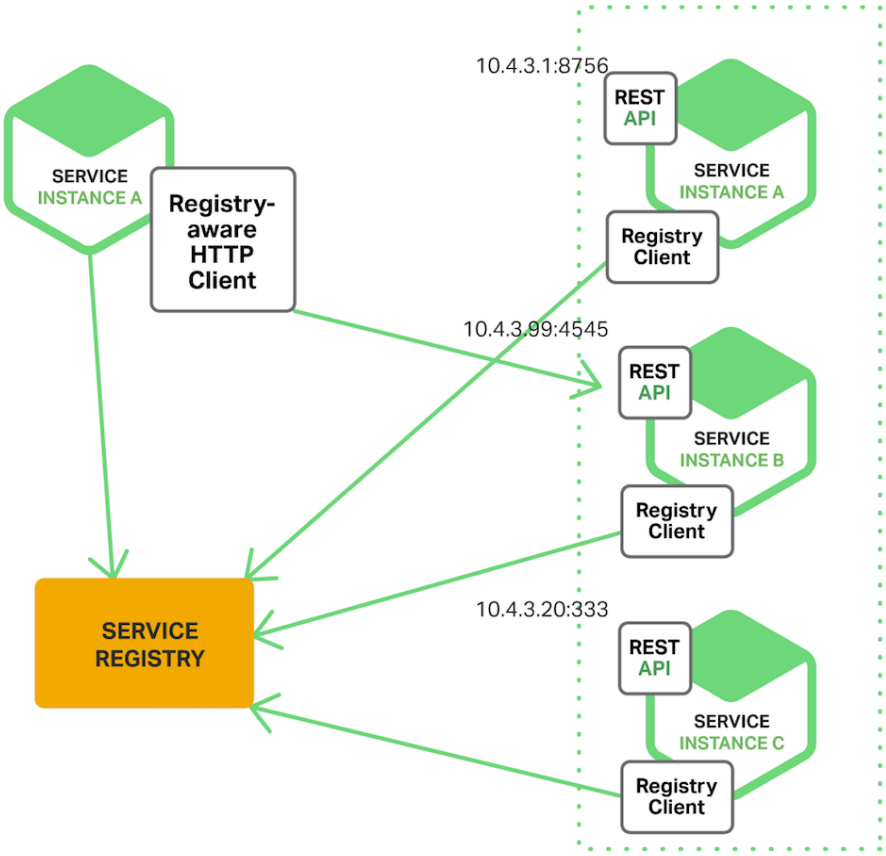


Figure 24: Client Service Discovery pattern [4]

Server-side service discovery happens through the load balancer. The load balancer queries the service registry and routes each request to an available instance. AWS Elastic Load Balancer (ELB) is an example of a server-side discovery router. Figure 25 shows the server-side service discovery pattern.

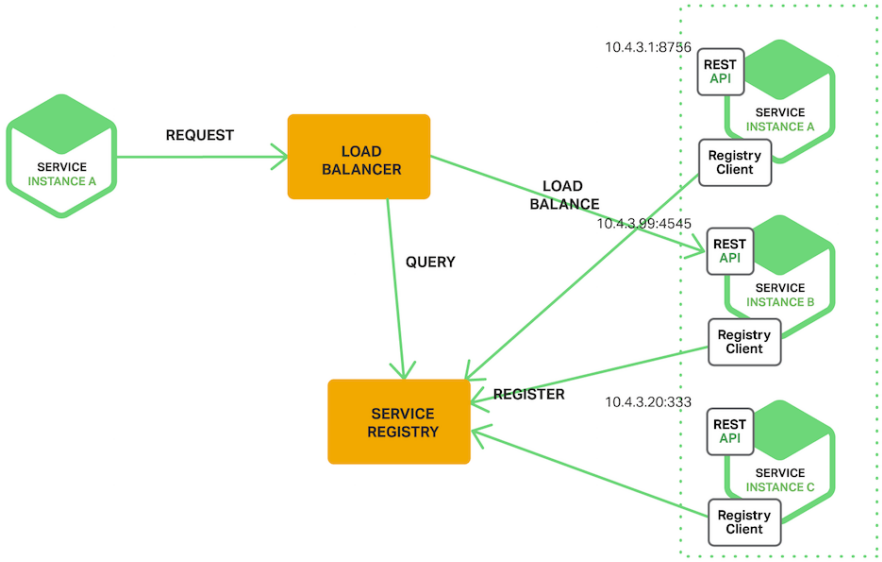


Figure 25: Client Service Discovery pattern [4]

## 4.7 Commonly used technologies in MSA (RQ5.2)

To identify the most commonly used tools in MSA, we first identified all the tools and then classified them into six categories as shown in Figure 26.

We identified a total of 34 tools and found that the tools under development accounted for the majority (11/34). The category with the least number of tools is testing (2/35). The six categories are explained below:

- **Communication** - As MSA is primarily composed of multiple services, robust communication between the services is key to extracting the best performance from the system. We identified 9 studies that mention communication tools. Kafka, a messaging system developed by Apache[8], is the most common tool identified for communication. The primary role of a messaging system is to transfer data from one service to another effectively so that the application can concentrate on the process of the data instead of worrying about how to share the data. As per study (P23), Kafka is a preferred tool for communication even though it is not a traditional messaging queue and is a distributed streaming platform [25]. It scales well as it is a distributed system and can be scaled easily without any downtime. Three studies also reported using ActiveMQ, a traditional message broker that supports various protocols [25].

- **Data** - As MSA primarily comprises multiple services that all need to share data, having a scalable and distributed data store is key. We identified 5 different tools that can be used to store and distribute data between various services. Study (P23) suggests using Debezium[9], a tool that can be used to monitor databases continuously and let any service stream the row-level data in the order it was stored. Another study (F41) suggests using Redis[10]an in-memory data structure tool that can be used as a database, cache, or message broker. It is built for

[8] https://kafka.apache.org/
[9] https://debezium.io/
[10] https://redis.io/

performance as it is in-memory. A couple of other tools mentioned in the study (F47) are Snowflake[11], a cloud-based data warehouse solution, which helps services run in the cloud to store and analyse large amounts of data. Teradata[12] is a solution similar to snowflake. It offers a similar service and Memcached[13], which is a distributed memory object caching system. It is an in-memory key-value store, which promotes quick deployment, ease of development, and solves many problems facing large data caches.

- **Development** - As each service in microservices can be developed using different technologies, choosing the right language and development-related tool is key. Hence, it builds a system that scales and performs well. The most common language and frameworks are Java and Spring. Study (P22) suggests one of the reasons for the use of Spring, which has large community support available, the various integration tools available, and its compatibility with various other microservice tools. Spring also provides various in-built tools such as Spring Cloud, which provides tools for developers to build some of the common patterns in distributed systems quickly and has a plethora of integrations available with popular cloud providers such as AWS, Azure and Google Cloud and Spring Eureka, which is a service discovery service provided by Spring. Other tools used for frontend development are Angular (P3) and React (P3), which develop single-page applications.

- **Testing** - Testing is the category with the least number of tools. Even though it is an integral part of developing MSA-based systems, there is not enough research on w.r.t testing, and the only tools mentioned are from study (P6), which are JMeter and JUnit, which are both Java-based testing tools.

- **Cloud infra/monitoring** - Most microservice-based systems are hosted on the cloud, and given the highly dynamic nature of MSA, they must be constantly monitored to prevent downtimes. We identified 6 tools that are used for cloud infrastructure and monitoring. ELK, which is mentioned in studies (P1, B15), is a great monitoring tool that aggregates logs, metrics, and other information from many sources in one place [26]. Cloud Monkey (P6) and AWS (B20) are used for hosting services, and they both provide a wide variety of tools for the same. Github (P31) is a tool used to host the codebase and provides a wide variety of tools for CI/CD integration.

- **CI/CD Tools** - As microservices are made up of small services, these systems need to be frequently integrated into a shared repository. Under this category, we identified 5 tools, with Docker being the most common tool mentioned in 6 studies. Docker is a tool that allows you to containerize each service and provides individual microservices with their isolated workload environments making them easy to be deployed on their own and scalable. Kubernetes, which is mentioned in studies (P6, P33, B28), is also used for improving scalability and performance by increasing infrastructure utilization through the efficient sharing of computing resources across multiple processes. Jenkins (P1, P33, P16) is an open-source automation service that can manage builds, deployments, and testing.

[11] https://www.snowflake.com/
[12] https://www.teradata.co.uk/
[13] https://memcached.org/

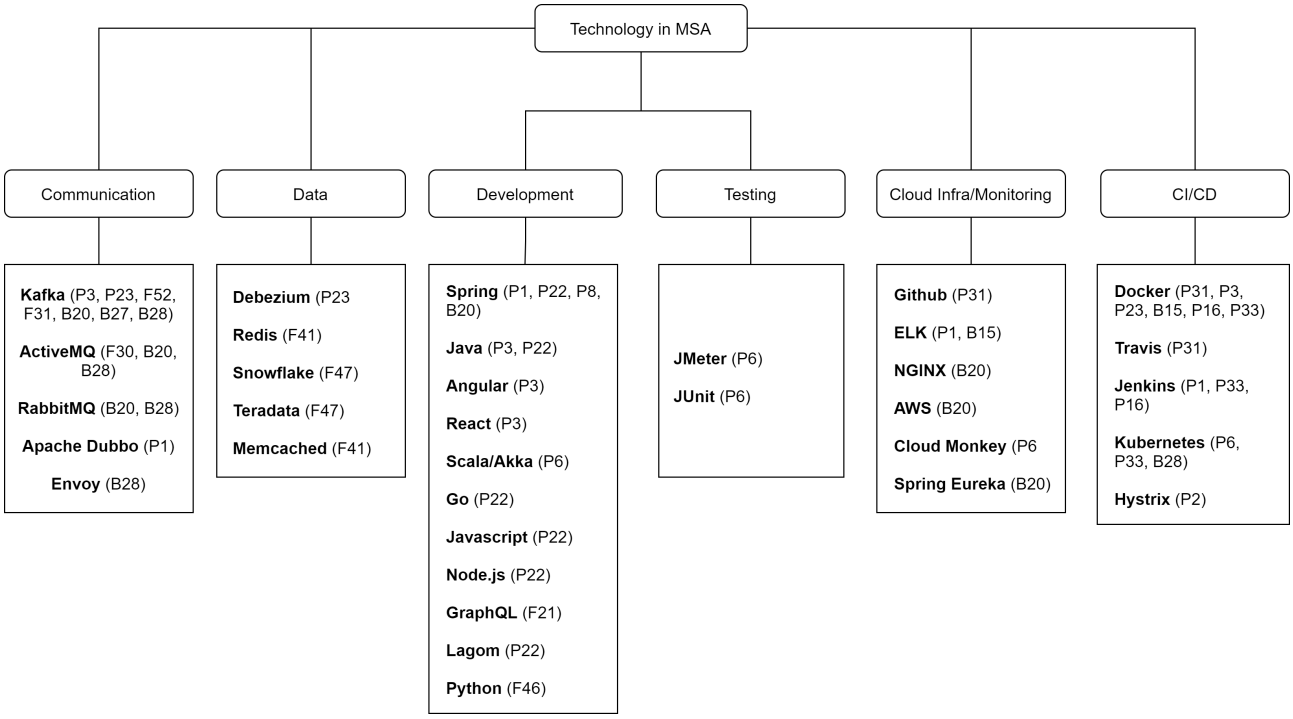


Figure 26: Various tools and technologies used in MSA

# 5 Discussion

In this section, we analyse and discuss the key findings from the study and discuss their implication for research and practice.

## 5.1 Analysis of the results

We further analyse and understand the results of the RQs related to analysing the studies, the AK identified, problems and solutions identified, architecture patterns identified and tools that support MSA.

### 5.1.1 Publication trends

The steady trend in the studies published through the years indicates that MSA as a research topic is considered to be a relevant topic by the software engineering research community. Our findings show that 29 studies (83%) of the studies were published as conference papers. One of the potential reasons for this is that MSA is a quickly evolving area. The work submitted to conferences can be reviewed promptly and published relatively quickly compared to other venues. We further analyse and understand the results of the RQs related to analysing the studies, the AK identified, problems and solutions identified, architecture patterns identified and tools that support MSA.

### 5.1.2 Focus of research in academia

**Research strategies** - The data collected contains a lot of evaluation research because MSA is already quite popular. In the past few years, most of the research has been to compare different implementations and provide a suitable solution based on the results. Researchers have been evaluating monolith systems and MSA. Studies (P5, P7, P11, P18, P28) discuss the main difference between the two systems. Studies P2 and P11 discuss the challenges in moving from a monolith to a microservice and claim that refactoring the code and breaking down the system poses a significant challenge as the services are too dependent on each other. Study P5 claims that it is recommended to implement a monolith for a small scale application as there is no significant gain in performance between a monolith and an MSA for small scale applications.

A few of the studies also focus on the performance aspect of MSA. Studies (P6, P16, P17) primarily focus on the design principles one should follow to develop an MSA application better than its monolith alternative. To achieve this, the system needs to be broken down to keep services independent, avoid tight coupling, keep in mind not to have big teams working on one service, and keep databases connected to only one service.

Since MSA is a new concept, many new solutions are yet to be identified; hence, researchers have identified new techniques to implement MSA. Studies P8 and P14 discuss new solutions with the help of an application and propose using new technologies such as Spring Boot to implement MSA. MSA should also be used with various DevOps strategies to improve the overall usability of the application. In this case, opinions are provided on areas of design where there isn't much information available and opinions are formed by conducting interviews or by performing other studies on MSA.

The low number of *experience papers* (3/28) and *conceptual papers* (1/35) indicate that researchers are not interested in publishing papers that do not have hard evidence to back up their concepts, this

could in a way hinder the research of MSA as researchers should publish more papers based on their experience of implementing MSA or based on a new concept without actually verifying, this way other researchers could use these new concepts and solve problems differently.

**Research contribution** - The high number of studies contributing to design patterns indicates that researchers are more focused on identifying new patterns for implementing different aspects of MSA. As there are multiple parts to MSA, each having its own set of implementation strategies, it is vital to use the correct pattern; hence identifying and proposing new design patterns helps practitioners make informed decisions when designing systems. The most common patterns being discussed are communication, and data handling, as those two categories are the most important factors when developing an application based on MSA.

The number of studies contributing to new applications is much lower because MSA is a reasonably new concept, and using MSA in different applications and proposing new architecture solutions is not easy due to the lack of research. One intersecting fact to note is that most of the studies contributing to applications are from before 2018, which is a bit surprising as research on MSA has constantly been growing, and one would expect new applications to be developed more recently.

**Research focus** - The focus on cloud confirms the close relationship that MSA has with DevOps and the cloud-based nature of applications. It also confirms that MSA is well suited for containerization and virtualization, making sure applications scale easily. Studies (P31, P33) discuss the use of *Docker* and *Kubernetes* with MSA and discuss the key benefits of containerization and suggest ways to design services that are independent and can be run as a standalone service.

The focus on development means that researchers are looking to develop applications using various tools and techniques. Studies (P1, P3, P23) discuss multiple tools that can be used to implement services to communicate effectively between multiple services. *Kafka* has been suggested in numerous studies as a tool that can be well integrated into MSA based systems.

The focus on system quality implies that quality attributes such as scalability, performance and security are essential with the system design of MSA, and researchers are investigating the different solutions to make informed design choices. Studies (P5, P6) suggest using microservices only for large systems as the benefits of implementing an MSA for small scale systems do not have significant performance improvement; instead, it increases the time required to develop the system.

**Software lifecycle scope** - The large gap between software design and implementation confirms that there is still a wide array of challenges and complexities involved in implementing MSA. Researchers are more interested in proposing better design solutions than implementing new systems.

**Architecting activities** - The trend continues here and further confirms that more research is being conducted to understand MSA, and less importance is being given to new implementation, which could lead to a knowledge gap and slow growth for better MSA adoption.

**Quality Attributes (QA)** - MSA based systems have a direct impact on quality attributes. One of the key factors that make MSA so enticing to practitioners is that it can be designed to suit ones need. A significant amount of research is being done on the performance of MSA, which indicates that researchers are concerned about the impact of MSA systems on performance. To further analyse these factors, we looked deeper to identify whether the QAs were positively or negatively impacted. In most cases, they were being impacted positively, and in cases where there were negative impacts,

there were alternate solutions proposed. Multiple studies linked MSA with improvement in scalability and maintainability, which bodes well for the adoption of MSA in practice.

### 5.1.3 Focus of research in industry

As discussed in 4.3, the sources for patterns are primarily blogs, whereas, for development, it is forums. The fact that the majority of the blogs talk about patterns and not about development is a cause for concern. Blogs are a great source of information as it is usually written by developers who have developed a system. The fact that none of the blogs talks about development means the developers who have implemented MSA have not shared their decision decisions or experience with MSA, which could be a great source of information as it will represent a hands-on experience. Developers need to be incentivised to write more tech blogs about their experience of implementing MSA in practice.

Another important observation is that the major difference between the development topics in forums and the literature is that the solutions proposed are very specific and helps the developer understand a particular problem. In contrast, in literature, the development solutions are often generic, cover a wide range of problems, and are not specific to solving a particular problem.

One significant difference between the patterns discussed in blogs versus the ones discussed in literature is that in the blogs, the patterns are discussed more in details, often with an example and use cases whereas in the literature, it is often mentioned as a solution to an issue identified or just as an alternative approach to solve a problem.

### 5.1.4 Problems and solutions identified

**Communication in MSA** - The proposed solutions for the communication problems are patterns or architecture design solutions. For example, the circuit-breaker pattern, bulkhead pattern are offered as solutions to solve communication failure. API Gateway pattern, Frontends pattern are suggested for patterns to facilitate easy client access. Some of the design solutions suggested are (i) designing the system asynchronously, (ii) avoiding DNS/URL routing and favouring service discovery registries, (iii) using message broker services to communicate efficiently. None of the solutions offered are technology-based, and that is quite surprising as implementing the suggested solutions could be challenging without understanding what tools to use. One other interesting fact to note is that there are many solutions than problems. This is one of the reasons to use MSA as there are multiple ways to solve issues and offers a variety of designs and strategies to develop a system that matches the requirements well. Another interesting fact we observed was the difference in the solutions offered by literature and other sources. Forums contributed to most of the solutions, which is not surprising as forums are where developers post questions; the difference was that in the forums, the solutions proposed were direct and offered to solve the problem directly, and in literature, multiple solutions were proposed to solve the same problem.

**Handling data in MSA** - Similar to communication problems, the solutions proposed for handling data were design suggestions and design patterns. For example, publish-subscribe pattern and database per service pattern were suggested to solve data consistency issues. Some of the design solutions offered were (i) Designing loosely coupled systems, (ii) implementing event-driven management design. When there are multiple tools available in the market for MSA for data handling, it is surprising that none of the solutions offered were technology-based.

### 5.1.5 Architecture patterns identified

As mentioned in Section 4.4, architecture patterns are identified more in blogs compared to literature and forum articles, indicating that practitioners are discussing more various patterns available to implement MSA. Most of the patterns discuss communication similar to the questions and solutions identified, which indicates that practitioners are concerned about the implementation of various communication protocols and are looking for ways to implement various strategies to have effective communicative systems.

Another key observation is that the patterns discussed in blogs are more detailed and usually discuss the use cases and the impact of using the pattern in practice. For example in studies (P2, P16) that mention the circuit-breaker pattern only contain information about what a circuit-breaker pattern is and does not provide any information on its architecture impact, which is an important detail that is required to make an informed decision whether to implement the circuit breaker pattern or not. Studies (B5, B6) discuss the circuit breaker pattern w.r.t its use cases and provide a clear explanation of when to use it. This trend is also observed in the API-Gateway pattern where studies (P6, P25, P33) discuss the API gateway pattern as a communication pattern but give no further details about its use cases and the various impact it can have on the system, nor does it provide crucial information on the problem it tries to solve. Whereas in studies (B5, B8), the study discusses the pattern in detail, the benefits and drawbacks, and provides information about another alternative one could consider solving a similar problem.

One other observation is that there are not enough patterns discussing important categories such as *deployment* and *migration*. The lack of research in these categories is concerning as MSA is very cloud-dependent, and having a robust deployment strategy could significantly impact the availability of the system. Despite some literature exploring migration strategies (P2, P7, P11, P13, P28), none of these studies have proposed any migration patterns. These patterns need to be explored more as these provide an effective solution to help practitioners move from a monolithic architecture to MSA.

### 5.1.6 Commonly used technologies in MSA

The most common categories are development and common which one again establishes a similar trend as observerd in earlier discussions on problems and solutions and architectural patterns. Majority of the tools mentioned indicate that MSA are developed using a cloud first solution, Docker and Kubernetes suggest the design of containerized solutions that can easily scale. One of the tools not explored much is testing, and given the fact that MSA systems rely heavily on multiple services, having a good testing framework is key in developing robust systems that are fault-tolerant. Studies (P1, P3, P11) do talk about testing MSA to keep systems available but do not offer any technology solution to solve the same.

# 6 Threats to Validity

Several threats can impact the quality of the results obtained. To mitigate these threats, we followed the guidelines set by Kitchenham and Charters [10]. In this section we discuss the various threats associated with this study.

## 6.1 Internal validation

Internal validation refers to the factors that could impact the analysis of the data collected. The threats could occur during the following stages of the study:

**Study search** - Identifying the suitable studies to analyse is critical, and there is a possibility to ignore studies that could be relevant. To mitigate this issue, apart from collecting studies from popular databases, we performed forward and backward snowballing (Section 3.1.3) and added 10 additional studies. Additionally, we initially ran the search string on a small set as an experiment to get the maximum primary studies, and we realised that we were missing some studies due to the variations in the way microservices was being spelt, which led to us improving the search string to include variations of the term microservices and arrive at the final string as mentioned in Section 3.1.1.

**Study selection** - We defined explicit inclusion and exclusion criteria as described in Section 3.1.2 to select relevant studies and not to exclude critical studies. In most SMS studies, two researchers collect and validate the studies; in this case, one researcher did it, which could have led to some studies being excluded. To prevent that, we defined a strong inclusion and exclusion criteria that help identify whether a study is to be included or discarded.

**Data Extraction** - Researchers can be biased when it comes to extracting data, which could lead to poor extraction quality. We mitigated this issue by creating a well-defined data extraction form, as mentioned in Table 5. The data was initially extracted by the primary researcher and was verified by the secondary researcher. To prevent bias when performing keywording, we followed the guidelines of thematic analysis set by Braun and Clarke [20]. The coding book defined in Appendix A made it possible to annotate the studies clearly and concisely.

**Data Synthesis** - We applied both quantitative and qualitative methods to analyse the extracted data from the studies. The bias in synthesis could have an impact on the final results. To mitigate this, we synthesised the data using a well-defined thematic analysis and followed strategies set by Paolo et al. [7].

## 6.2 External validation

The threats on external validation refer to the impact with which the results of this study can be generalized. The primary outcome of this study provides an overview of the state of research and the adoption of MSA by researchers and practitioners. Hence, the results obtained are only valid for the study topic. To achieve external validation, we only collected studies from 4 of the most popular databases and collected data from the industry from an already verified source.

## 6.3 Construct validation

We avoided the bias in selecting studies by performing an automatic search on the databases and used already verified sources to collect studies. We also ran a sample test with the test string to identify

studies and eventually tweaked the string to include all the studies relevant to the study. We collected data from the best 4 databases to potentially avoid questionable studies.

## 6.4 Conclusion validation

We tried to mitigate this issue by constantly updating the classification framework and adapting the coding book to reduce potential bias when analysing the studies. In doing so, we are confident that the data extraction process aligned with our research questions. In addition, to avoid potential bias, we conducted multiple meetings with all the researchers to discuss the interpretation of the results and conclusion.

# 7 Related Work

We identified three secondary studies by Paod Di Francesco et al. [7], Muhammad Waseem et al.[27] and Alshuqayran et al. [28] that report various aspects of MSA.

A systematic mapping study on MSA was done by Paolo Di Francesco et al. [7] on a primary set of 91 studies between the period of 2012 and 2016. Their study focuses on (i) designing a classification framework and evaluating the studies based on that, (ii) evaluating the results obtained for industrial adoption, (iii) an SMS for the current research, and identify themes for future research topics.

Muhammad Waseem et al. [27] conducted an SMS on MSA in DevOps. They explore the studies between 2009 and 2018. This SMS aims to identify, classify and analyse the use of DevOps in MSA. The results of this study are (i) analyse the publication trends and research themes, (ii) What are the problems and solutions reported with using DevOps techniques in MSA, (iii) Challenges faced when implementing MSA in the cloud, (iv) Various methods, design patterns, the impact of quality attributes and tools used in implementing DevOps strategies in MSA.

Alshuqayran et al. [28] conducted an SMS on MSA based on 33 primary studies published between 2014 and 2016. The major reported findings are (i) MSA challenges, (ii) QAs associated with MSA, (iii) MSA views. Their study also explores existing architectural support for MSA and characterizes a framework for MSA.

# 8 Future Work

The present study is based on only three sources: literature, blogs, and forums. To get more insights into the adoption of MSA and understand the latest MSA techniques, this study can further be extended to include more data sources such as books, technical whitepapers, source code documentation, and user documentation.

Furthermore, the inclusion and exclusion criteria can be relaxed to include more topics such as MSA-based applications and tools to analyse more categories and get a clearer idea of the various applications developed using MSA.

The current study also relies on manually identifying documents and searching for a particular result; this can be expanded by using a Semantic wiki tool such as WikiBase[14]. A semantic Wiki is simply a Wiki that takes advantage of semantics rather than syntax. With the help of this tool, one can develop their own microservice system to automatically export new annotated documents from Atlast.ti to cloud storage such as Amazon S3. From here, we can have an application that is connected to the S3 bucket to observe changes. When a new file is uploaded, it can process the file, create semantic tags, and push it onto the Wikibase database, which can then be accessed through the Wikibase interface. One can also create various semantic queries, which can help extract the necessary information from Wikibase and help researchers find solutions to problems a lot easily.

[14] https://wikiba.se/

# 9 Conclusion

The purpose of this study is to provide an understanding of the various trends in MSA in both research and the industry. We performed a systematic mapping study on 35 literature studies. We annotated all 81 studies collected from various sources and used the extracted data to perform a systematic analysis to identify the publication trends, the various themes of research, identify the common problems and solutions, the common architecture patterns found, and also the most common tools used to implement or support MSA.

The key findings can be summarized as follows:

- The steady research trends in publication over the years indicate good interest in MSA research.
- The main research strategies being explored in research are w.r.t evaluation research, which indicates that researchers are concentrating more on analysing MSA topics and not proposing many new solutions.
- There is a clear indication that MSA and DevOps go hand-in-hand, and multiple studies explore the cloud-based aspect of designing MSA-based systems.
- Researchers are also exploring existing MSA patterns and proposing many new patterns, but they are not implementing a lot of new systems to explore these patterns in practice.
- QAs are given a lot of importance, with performance being the most important factor, and a lot of research has been to improve the various communication gaps between services.
- Migration from monolith to MSA is not being explored much in detail, which could hinder the fast adoption of MSA in practice.
- Industry experts are not publishing a lot of content related to implementation strategies and design decisions. They focus more on design patterns, which could make it difficult for practitioners to adopt MSA as there is a lack of technology-based solutions.
- We identified 13 questions and 36 solutions overall. The majority of the questions and solutions discussed target communication and data handling issues. There have been very few to no questions on other important topics such as migration, security, and various tools available. The solutions proposed also are primarily focused on design strategies or architecture patterns. There is a clear lack of solutions w.r.t technological solutions.
- We identified 41 different architectural patterns. The most common patterns are based on communication, and the circuit breaker pattern is the most common with 12 occurrences.

The findings of this study will benefit researchers and practitioners interested in identifying the current state of research in MSA. The findings of the study also help researchers to identify the areas that are being researched less. It will help practitioners identify the various problems, solutions, patterns, and tools available at their disposal to adopt MSA.

# Bibliography

[1] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, (New York, NY, USA), Association for Computing Machinery, 2014.

[2] M. Fowler, "Bliki: Circuitbreaker," Mar 2014.

[3] Nishanil, "The api gateway pattern versus the direct client-to-microservice communication."

[4] C. R. o. Eventuate, "Service discovery in a microservices architecture," Jul 2019.

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 1st ed., February 2015.

[6] H. Zhang, S. Li, Z. Jia, C. Zhong, and C. Zhang, "Microservice architecture in reality: An industrial inquiry," in *2019 IEEE International Conference on Software Architecture (ICSA)*, pp. 51–60, 2019.

[7] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 21–30, 2017.

[8] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry, "Architectural knowledge and rationale: Issues, trends, challenges," *SIGSOFT Softw. Eng. Notes*, vol. 32, p. 41–46, July 2007.

[9] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," in *Current Trends in Web Engineering* (I. Garrigós and M. Wimmer, eds.), (Cham), pp. 32–47, Springer International Publishing, 2018.

[10] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report, 07 2007.

[11] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pp. 1–10, 2008.

[12] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[13] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011. Special Section: Best papers from the APSEC.

[14] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019.

[15] C. Schröer, F. Kruse, and J. Marx Gómez, "A qualitative literature review on microservices identification approaches," in *Service-Oriented Computing* (S. Dustdar, ed.), (Cham), pp. 151–168, Springer International Publishing, 2020.

[16] J. Lewis, "Using atlas.ti to facilitate data analysis for a systematic review of leadership competencies in the completion of a doctoral dissertation," *SSRN Electronic Journal*, 01 2016.

[17] M. Soliman, M. Wiese, Y. Li, M. Riebisch, and P. Avgeriou, "Exploring web search engines to find architectural knowledge," in *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, pp. 162–172, 2021.

[18] M. Miles, *Qualitative data analysis : a methods sourcebook*. Los Angeles: SAGE, 2019.

[19] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requir. Eng.*, vol. 11, pp. 102–107, 03 2006.

[20] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[21] C. Chris Richardson, "Microservices pattern: Circuit breaker."

[22] A. Messina, R. Rizzo, P. Storniolo, M. Tripiciano, and A. Urso, "The database-is-the-service pattern for microservice architectures," vol. 9832, pp. 223–233, 09 2016.

[23] Systango, "What is api gateway & why i should use it?," Aug 2019.

[24] "Microservices pattern: Server-side service discovery pattern."

[25] C. Lam, "What is the difference between apache kafka vs activemq," Aug 2017.

[26] "Harnessing the power of microservices with the elk stack," Apr 2021.

[27] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in devops," *Journal of Systems and Software*, vol. 170, p. 110798, 2020.

[28] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," pp. 44–51, 11 2016.

[29] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[30] C. Y. C. Y. Baldwin, *Design rules*. Cambridge, Ma.: MIT Press, 2000.

[31] P. Stoll, A. Wall, and C. Norstrom, "Guiding architectural decisions with the influencing factors method," in *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pp. 179–188, 2008.

[32] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. Chichester, UK: Wiley, 2007.

[33] P. Avgeriou and U. Zdun, *Architectural Patterns Revisited - A Pattern*, pp. 1–39. 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, UVK Verlagsgesellschaft, 2005. 10th European Conference on Pattern Languages of Programs : EuroPLoP' 2005 ; Conference date: 06-07-2005 Through 10-07-2005.

[34] "Software architecture - examples, tools, & design. definition & more: Cast."

[35] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pp. 109–120, 2005.

[36] ISO/IEC, "Systems and software engineering – Systems and software quality requirements and evaluation (SQuaRE) – Data quality model," ISO/IEC 25012, International Organization for Standardization, Geneva, Switzerland, 2008.

[37] "What is systems design? definition of systems design, systems design meaning."

[38] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. SEI series in software engineering, Addison-Wesley, 2003.

[39] "What is software implementation in business?," Aug 2021.

# Appendices

## A Coding book for annotations

1. Architecture Configuration (ACF) - Architecture configuration can be defined as a particular way in which a system's components or services and their connectors or bindings are composed using very specific settings that results in the final system design [29].

   Example: Study (F54)

   > "Let's consider the scenario when a service wants to communicate with other service let's say Service-A needs to communicate with Service-B. The Service-A needs to know the IP address and Port number of the Service-B. The easiest solution to the problem is by maintaining a configuration file that holds the IP address and port to Service-B at Service-A."

2. Architecture Design Rule (ADR) - Baldwin and Clark [30] proposed that a software design should adhere to a modular structure framed by design rules. Architectural decisions decouple the rest of the system into modules to evolve independently from each other.

   Example: Study (P25)

   > "The application should also be decomposed in such a manner that most new and changed requirements only affect a single service. This is because changes that affect multiple services require coordination across multiple teams, which slows down development."

3. Architecture Impact (AIM) - Architecture impact can be classified as positive and negative impact. A positive impact means that the decision factor contributes to achieving the goal or the implementation of a software quality attribute. A decision that influences a negative factor implies that the influencing factor inhibits the business goal accomplishment or the software quality attribute implementation [31].

   Example: Study (P17)

   > "The above perks aren't for free, of course. The setup is quite complex, since we need to store the messages in the database of the sending and reveiving services. Also, we need to implement jobs on both sides that poll the database, looking for unprocessed messages and then process them by sending them to the message broker (on the sending side) or calling the business logic that processes the message (on the receiving side)."

4. Architecture Patterns (ATP) - Architectural patterns are defined as universal and reusable solutions to commonly occurring problems in software architecture [32]. Architectural patterns usually describe high-level designs and behaviours of software's and solve a particular type of recurring problem when designing software architecture. They are used to satisfy functional and quality attribute requirements usually satisfy functional and quality attribute requirements [33].

   Example: Study (P17)

> "The API Gateway Design Pattern - This pattern consists of a gateway through which a number of different subservices are accessible. An API gateway acts as a single entry point for requests and based on the nature of a request, it invokes an appropriate subservice. In addition to routing and aggregation, an API gateway performs two important tasks, namely, gateway offloading and circuit breaking."

5. Architecture Planning (ATP) - Architecture planning helps you identify milestones and provide everyone with exact directions to keep the project on track.

   Example: Study (P2)

   > "When splitting up the services, attention should be paid to the fact that the services do not become too fine grained. Microservices can introduce a performance overhead especially if the communication is done over network ."

6. Architectural Solution (ASL) - Architectural Solution is described as a design plan that will describe a system and how different system components come together, how they work together, and whether they meet the system's requirements. This plan acts as a blueprint only during the development phase, which helps during the system's coding, integration, and testing phase. It also helps to establish system requirements [34].

   Example: Study (P27)

   > "Centralize the source code and documentation for all services in a common management system: The solution for this problem goes through keeping a centralized management system whereby all the hisitoric data can be accessed, including documentation, knowledge reports and source code. A change of policy must be in place in the company to ensure the teams will put data in this place. After that, no additional cost should be required, once the teams will only change the place where they will deposit the information."

7. Design Decision (DDC) - As described by Anton [35], software architecture is designed based on various requirements. Requirements define how the particular system should function, whereas software architecture defines how this is achieved. The primary input for software architecture is the requirements document itself. Several tactics are applied to achieve this particular architecture, and the tactics used are generally the decisions one needs to take to develop the architecture.

   Example: Study (P2)

   > " The first thing to do is to define the microservices and their responsibility areas. It is important that the decomposition of services is correct. This is important, because it is expensive to make a lot of changes across the services. Instead it is easy to change functionality inside one service, but when the changes affect multiple services and their interfaces, then the task becomes harder and more time consuming."

8. Quality Attributes (QAT) - The ISO 25012 official spec document defines Quality Attributes as "The Data Quality model represents the grounds where the system for assessing the quality of data products is built on. In a Data Quality model, the main Data Quality characteristics that must be taken into account when assessing the properties of the intended data product are established" [36].

   Example: Study (F33)

> "The whole point of microservices is to update or deploy one service while keeping other services intact. In a loosely coupled system, one service knows little about others. Sot this makes the key idea possible. High cohesion means related logic is kept in one service. Otherwise, different services need to be chatty to each other across service boundary. This will hurt the overall performance."

9. System Design Process (SDP) - The System Design Process defines elements of a system, such as architecture, overall design strategy, components and interfaces based on the requirements. It's also the process of developing and designing a system to meet business requirements [37].

   Example: Study (P31)

   > "The system was setup up so that everytime a developer pushes a set of commits to a shared source code repository which is dedicated to a particular service, notifications will be sent to group of people who are major stakeholders of the system including system owner, other developers, testers, etc and whoever subscribed to the group. This visualization helps to promote the engagement, collaboration between different stakeholders and maximize personal responsibility awareness of the developer in every line of code and somewhat isolate potential bugs."

10. Technical Constraints (TCC) - Technical constraints are fixed technical design decisions that absolutely cannot be changed. Most often, these result from the requirements and constraints provided by the stakeholders or a specific tool. It could be of the form using a particular programming language, operating system or certain libraries of frameworks [38].

    Example: Study (P22)

    > "For example, our results show that Moleculer can achieve a good end-to-end latency performance, but when deploying using Docker, Go Micro can produce much smaller images, which could be beneficial on limited resources."

11. Technical Implementation (TIP) - Technical implementation refers to the process of adopting and integrating a software application into a business workflow using technological tools and solutions [39].

    Example: Study (P22)

    > "In your example, Delivery service can duplicate delivery locations and product information. Product service manage the products and locations. Then the required data is copied to Delivery service's database with async messages (for example you can use rabbit mq or apache kafka). Delivery service does not change the product and location data but it uses the data when it is doing its job. If the part of the product data which is used by Delivery service is changing often, data duplication with async messaging will be very costly. In this case you should make api calls between Product and Delivery service. Delivery service asks Product service to check whether a product is deliverable to a specific location or not. Delivery service asks Products service with an identifier (name, id etc.) of a product and location. These identifiers can be taken from end user or it is shared between microservices. Because the databases of microservices are different here, we cannot define foreign keys between the data of these microservices."

12. Technology Solution (TES) - Technology solution refers to designs that indicate a particular type of technology that can be used to implement a particular system from an architecture.

    Example: Study (P31)

    > "The solution being discussed in the case study is not only leveraging Docker technology, but also strictly following agile methodology in software development and enjoying surrounding tools' benefits like Waffle, Github, Slack and so on."

13. Use case (UCS) - A use case can be described as a way in which users will use the proposed system. It outlines how the system design can be used in a specific scenario running through all the steps.

    Example: Study (P31)

    > "As an example use case, our team had been using the traditional monolithic model for building applications for a long time. We understand the incremental effort we have to spend in development, deployment as well as the overhead in each release of the app where a lot of meetings and coordination were required between dev and ops teams."

14. Tradeoffs (TOF) - Software designing and technologies are solutions to solve specific problems, and their benefits or flaws are never absolute, but always bound to the context, hence a tradeoff is quite common in software engineering.

    Example: Study (P31)

    > " With microservices, each service has its own database and models, which may evolve independently of external services. Decentralized data management and the possibility to use different technologies that best ft each context are relevant advantages. On the other hand, increased operational complexity is the main drawback."

## B List of literature

Table 14 shows all the papers selected for the study along with the study ID, the title, the database, type and year published.

| Study ID | Paper Name | Database | Type | Year |
|---|---|---|---|---|
| P1 | Microservice Architecture in Reality: An Industrial Inquiry | IEEE | Conference Paper | 2019 |
| P2 | Challenges When Moving from Monolith to Microservice Architecture | Springer | Conference Paper | 2018 |
| P3 | Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality | IEEE | Conference Paper | 2019 |
| P4 | Microservice Architectures for Scalability, Agility and Reliability in E-Commerce | IEEE | Conference Paper | 2017 |
| P5 | A Comparative Review of Microservices and Monolithic Architectures | IEEE | Conference Paper | 2018 |
| P6 | Performance Analysis of Microservice Design Patterns | IEEE | Journal | 2019 |
| P7 | An Experience Report from the Migration of Legacy Software Systems to Microservice Based Architecture | Springer | Conference Paper | 2019 |
| P8 | Implementation of the Internet of Things Application Based on Spring Boot Microservices and REST Architecture | Springer | Conference Paper | 2020 |
| P9 | The pains and gains of microservices: A Systematic grey literature review | ScienceDirect | Journal | 2018 |
| P10 | Microservices Architecture: Challenges and Proposed Conceptual Design | IEEE | Conference Paper | 2019 |
| P11 | Migrating from monolithic architecture to microservices: A Rapid Review | IEEE | Conference Paper | 2019 |
| P12 | An Expert Interview Study on Areas of Microservice Design | IEEE | Conference Paper | 2018 |

**Table 14 continued from previous page**

| | | | | |
|---|---|---|---|---|
| P13 | Migrating Towards Microservice Architectures: An Industrial Survey | IEEE | Conference Paper | 2018 |
| P14 | InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities | ACM | Conference Paper | 2017 |
| P15 | A dynamic deployment method of micro service oriented to SLA | Others | Journal | 2016 |
| P16 | Understanding and addressing quality attributes of microservices architecture: A Systematic literature review | ScienceDirect | Journal | 2021 |
| P17 | Quality attributes in patterns related to microservice architecture: a Systematic Literature Review | IEEE | Conference Paper | 2019 |
| P18 | Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report | IEEE | Conference Paper | 2017 |
| P19 | Microservices: architecture, container, and challenges | IEEE | Conference Paper | 2020 |
| P20 | The Database-is-the-Service Pattern for Microservice Architectures | Others | Conference Paper | 2016 |
| P21 | Exploring the Impact of Situational Context — A Case Study of a Software Development Process for a Microservices Architecture | IEEE | Conference Paper | 2016 |
| P22 | Development Frameworks for Microservice-based Applications: Evaluation and Comparison | ACM | Conference Paper | 2020 |
| P23 | Case study on data communication in microservice architecture | ACM | Conference Paper | 2019 |
| P24 | Microservice architecture in industrial software delivery on edge devices | ACM | Conference Paper | 2018 |

**Table 14 continued from previous page**

| | | | | |
|---|---|---|---|---|
| P25 | A Microservice Architecture for the Industrial Internet-Of-Things | ACM | Workshop Paper | 2018 |
| P26 | Towards an Understanding of Microservices | IEEE | Conference Paper | 2017 |
| P27 | Architectural Technical Debt in Microservices | IEEE | Conference Paper | 2019 |
| P28 | Migrating Web Applications from Monolithic Structure to Microservices Architecture | ACM | Conference Paper | 2018 |
| P29 | Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices | ACM | Conference Paper | 2017 |
| P30 | Guidelines for adopting frontend architectures and patterns in microservices-based systems | ACM | Conference Paper | 2017 |
| P31 | Leveraging microservices architecture by using Docker technology | IEEE | Conference Paper | 2016 |
| P32 | Native Cloud Applications: Why Virtual Machines, Images and Containers Miss the Point! | Others | Conference Paper | 2016 |
| P33 | Migrating to Cloud-Native Architectures Using Microservices: An Experience Report | Springer | Conference Paper | 2016 |
| P34 | Design and implementation of a decentralized message bus for microservices | IEEE | Conference Paper | 2016 |
| P35 | Challenges in Delivering Software in the Cloud as Microservices | IEEE | Journal | 2016 |

Table 14: List of selected papers

## C List of blogs

Table 15 shows all the blog articles selected for the study along with the study ID, title, and URL.

| Study ID | Title | URL |
|---|---|---|
| B1 | Pattern: API Gateway / Backends for Frontends | https://microservices.io/patterns/apigateway.html |
| B2 | Pattern: Remote Procedure Invocation (RPI) | https://microservices.io/patterns/communication-style/rpi.html |
| B3 | Functional decomposition for Microservices | http://www.waynecliffordbarker.co.za/functional-decomposition-for-microservices/ |
| B4 | Pattern Decompose by business capability | https://microservices.io/patterns/decomposition/decompose-by-business-capability.html |
| B5 | Design Patterns for Microservices | https://dzone.com/articles/design-patterns-for-microservices |
| B6 | Design Patterns for Microservices | https://dzone.com/articles/design-patterns-for-microservices-1 |
| B7 | Pattern: Microservice Architecture | https://microservices.io/patterns/microservices.html |
| B8 | Everything You Need To Know About Microservices Design Patterns | https://www.edureka.co/blog/microservices-design-patterns |
| B9 | Microservice Architecture and Design Patterns for Microservices | https://dzone.com/articles/microservice-architecture-and-design-patterns-for |
| B10 | Microservices: Decomposing Applications for Deployability and Scalability | https://www.infoq.com/articles/microservices-intro/ |
| B11 | Microservices in Practice - Key Architectural Concepts of an MSA | https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/ |
| B12 | Isolating your microservices through loose coupling | https://medium.com/it-dead-inside/isolating-your-microservices-through-loose-coupling-48b710e28de6 |
| B13 | The importance of loose coupling in microservice architecture | https://info.nl/en/conversation/the-importance-of-loose-coupling-in-microservice-architecture/ |
| B14 | Pattern: Decompose by business capability Context | https://microservices.io/patterns/decomposition/decompose-by-business-capability.html |
| B15 | The What, Why, and How of a Microservices Architecture | https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9 |

**Table 15 continued from previous page**

| | | |
|---|---|---|
| B16 | Microservice Architecture — Communication & Design Patterns | https://medium.com/dev-genius/microservice-architecture-communication-design-patterns-70b37beec294 |
| B17 | Microservice Communication Patterns | https://reflectoring.io/microservice-communication-patterns/ |
| B18 | Pattern: Messaging | https://microservices.io/patterns/communication-style/messaging.html |
| B20 | Design Patterns for Microservice-To-Microservice Communication | https://dzone.com/articles/design-patterns-for-microservice-communication |
| B27 | How are your microservices talking? | https://aiven.io/blog/how-are-your-microservices-talking |
| B28 | Microservice Architecture — Communication & Design Patterns | https://medium.com/dev-genius/microservice-architecture-communication-design-patterns-70b37beec294 |

Table 15: List of selected blogs

## D List of forum articles

Table 16 shows all the forum articles selected for the study along with the study ID, title, and URL.

| Study ID | Title | URL |
|---|---|---|
| F21 | When and How to use GraphQL with Microservice Architecture | https://stackoverflow.com/questions/38071714/when-and-how-to-use-graphql-with-microservice-architecture |
| F22 | Microservice Authentication strategy | https://stackoverflow.com/questions/29644916/microservice-authentication-strategy |
| F23 | orchestrating microservices | https://stackoverflow.com/questions/29117570/orchestrating-microservices |
| F24 | Transactions across REST microservices? | https://stackoverflow.com/questions/30213456/transactions-across-rest-microservices |
| F25 | Microservice Authentication strategy | https://stackoverflow.com/questions/29644916/microservice-authentication-strategy |
| F30 | Microservices: Handling eventual consistency | https://softwareengineering.stackexchange.com/questions/354911/microservices-handling-eventual-consistency |
| F31 | Microservice Decomposition and Inter-service communication | https://softwareengineering.stackexchange.com/questions/395448/microservice-decomposition-and-inter-service-communication |
| F32 | How to handle data inconsistency in microservice architecture? | https://softwareengineering.stackexchange.com/questions/416035/how-to-handle-data-inconsistency-in-microservice-architecture |
| F33 | What is the importance of cohesion and coupling in microservices? | https://www.quora.com/What-is-the-importance-of-cohesion-and-coupling-in-microservices |
| F38 | Microservices Why Use RabbitMQ? | https://stackoverflow.com/questions/45208766/microservices-why-use-rabbitmq |
| F40 | What are microservices and why should you care? | https://inform.tmforum.org/features-and-analysis/2017/02/what-are-microservices-and-why-should-you-care/ |
| F41 | Microservices and database joins | https://stackoverflow.com/questions/29761872/microservices-and-database-joins |

**Table 16 continued from previous page**

| | | |
|---|---|---|
| F42 | Microservices: how to handle foreign key relationships | https://stackoverflow.com/questions/44870461/microservices-how-to-handle-foreign-key-relationships |
| F43 | Microservices vs Monolithic Architecture [closed] | https://stackoverflow.com/questions/33041733/microservices-vs-monolithic-architecture |
| F44 | How does data denormalization work with the Microservice Pattern? | https://stackoverflow.com/questions/27007353/how-does-data-denormalization-work-with-the-microservice-pattern |
| F45 | Single Sign-On in Microservice Architecture | https://stackoverflow.com/questions/25595492/single-sign-on-in-microservice-architecture |
| F46 | Sharing code and schema between microservices | https://stackoverflow.com/questions/25600580/sharing-code-and-schema-between-microservices |
| F47 | Microservices with shared database? using multiple ORM's? [closed] | https://stackoverflow.com/questions/43612866/microservices-with-shared-database-using-multiple-orms |
| F48 | 2PC vs Sagas (distributed transactions) | https://stackoverflow.com/questions/48906817/2pc-vs-sagas-distributed-transactions |
| F49 | What is a microservice? [closed] | https://stackoverflow.com/questions/46575898/what-is-a-microservice |
| F50 | Data Sharing between micro services | https://stackoverflow.com/questions/41640621/data-sharing-between-micro-services |
| F51 | Microservices: what are pros and cons? | https://stackoverflow.com/questions/34903605/microservices-what-are-pros-and-cons |
| F52 | Data Consistency Across Microservices | https://stackoverflow.com/questions/43950808/data-consistency-across-microservices |
| F53 | Kafka Msg VS REST Calls | https://stackoverflow.com/questions/57852689/kafka-msg-vs-rest-calls |
| F54 | What is service discovery, and why do you need it? | https://stackoverflow.com/questions/37148836/what-is-service-discovery-and-why-do-you-need-it |

Table 16: List of select forum studies