Tampere University

Noora Rintamäki

IDENTIFYING REQUIREMENTS IN MICROSERVICE ARCHITECTURAL SYSTEMS

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
December 2022

ABSTRACT

Noora Rintamäki: Identifying Requirements in Microservice Architectural Systems
M.Sc. Thesis
Tampere University
Master's Degree Programme in Software Development
December 2022

Microservices and microservice architecture has grown popularity and interest steadily since 2014 but many challenges are still faced in a software project when trying to adopt the concept. This work gathers challenges, possible solutions, and requirements related to the use of microservice architecture and therefore support the work of different stakeholders in a software project using microservice architecture, while also providing more information to the research as well. The study was conducted using systematic literature review (SLR). Overall, 63 scientific publications from four different scientific databases were selected and analysed. As a result, rapid evolution, life cycle management, complexity, performance, and a large number of integrations were identified as the most common challenges of microservice architecture. Solutions such as service orchestration, fog computing, decentralized data, and use of patterns were proposed to tackle these challenges. Regarding requirements, scalability, efficiency, flexibility, loose coupling, performance, and security appeared most frequently in the literature. The key finding of this work was the importance of data. How data acts as a base for functionalities and when inaccurate can cause complex challenges and make functionalities worthless. Based on this, we have a better understanding on what challenges may occur and what to focus on while working with microservice architecture in software development.

Keywords: microservices, microservice architecture, software requirements, challenges, solutions, data, software development.

Contents

1 Introduction

The work of software project begins with the requirements, defining what is actually needed by the system. Requirements can be functional or non-functional requirements, features or attributes how the application should work or be. It is important to have as realistic guidelines and requirements as possible early on that guide the project into the right direction right from the start.

The purpose of this work is to support researchers and practitioners, in this case developers and stakeholders, working with microservice architecture, to know about common challenges in microservice approach, possible solutions to those challenges, and requirements for a microservice architecture. This work maps and analyses the current knowledge and research in microservice architecture and provides information on fundamental requirements and possible obstacles in microservice architecture so that they are known and can be prepared for from the beginning of the project.

This study gained a comprehensive understanding on challenges and possible solutions offered by the literature. In addition to that, 101 requirements were identified, ordered, and categorized based on their frequency in the literature and out of those 12 highlighted as the most appearing ones. Study provides an understanding on what challenges microservice projects are most likely to encounter, what solutions research has identified to solve them, and what requirements are the most relevant ones when operating with microservice architecture.

1.1 Research questions

Research questions were defined with an iterative nature. Research aims to answers to the following questions:

RQ1: What are common challenges in microservice architecture?
This question aims to gather what kind of obstacles are identified through various studies on microservice architecture. What challenges come from implementing this specific architecture and what kind of context may not be functional with microservice architecture.

RQ2: How are common challenges in microservice architecture addressed?
As a follow up question for the RQ1, what possible solution were identified to the challenges by those studies. Were they able to resolve the challenges?

RQ3: What requirements should be set to a microservice project?
Finally, after identifying possible obstacles, challenges, solution options for microservice architecture, what more general information can be drawn in a form of requirements. Requirements that guide software project through not only by avoiding possible obstacles but highlighting the important features and aspects in a software.

1.2 Structure of the Thesis

In this work we identified common requirements regarding the microservice architecture and listed different challenges and solution that appeared in the wide range of studies. This research is conducted by using systematic literature review (SLR). This research approach provides a comprehensive overview on where we are on this research area.

Structure of the work is following. Chapter 2 considers microservice architecture, its' structure, layers, advantages, and disadvantages. Chapter 3 focuses on requirements, quality, requirements engineering, stakeholders, and different levels and classification of the requirements. Chapter 4 summarises related work and Chapter 5 introduces theoretic background of systematic literature review that is used in this study. Chapter 6 examines the research method in practice. Chapters 7 and 8 continue with the results and discussion, Chapters 9 states the possible limitations and threats to validity and finally, Chapter 10 combines the conclusions.

2 Microservice architecture

Microservice architecture (MSA) is based on three ideas [Wolff, 2016]:

- A service should fulfil only one task and do it well.
- Services should be able to work together.
- Service should use a universal interface.

These characteristics aim to support modularization and design for reusable components. Microservice architecture also emphasizes isolation. One of the key things is that each service operates on its own and makes independent deploying, updating, and modifying possible. Each process and user interaction also aims to operate within a scope of a particular service. The primary goal of microservice approach is to enable independent service deployment and evolution across entire system. [Cerny et al., 2017]

2.1 Microservice Structure

The idea behind microservice architecture is to have small-scale services that are independently distributed and loosely coupled and aims to overcome the limitations of monolithic architecture [Li et al., 2021]. Goal is to provide autonomous services that can be deployed, operated, and developed independently time and to some extend also technology-wise. Each microservice is ideally structured around certain business logic. [Söylemez et al. 2022]. According to O'Connor et al. [2017] microservice architecture consists of building blocks such as main business services, discovery mechanisms, communication infrastructure, and infrastructural services. Each block is isolated from each other, and they communicate using a lightweight communication protocol. This isolation also enables evolvability over time when business or technology needs may change [O'Connor et al. 2017]. While considering microservice architecture in a project, it is good to know that by all means it is not the most affordable approach. Microservice architecture should be used when its benefits are greater than its' costs.

2.2 Architectural layers

Regarding architectural layers in microservice architecture, this study follows Fowlers [2016] definition where architecture can be divided into four different layers: microservice, application platform, communication, and hardware.

Hardware layer consist of the actual machines that runs everything. Those can be machines from owned datacentre or virtual machines run by a cloud provider. In addition to machines, databases whether they are dedicated or shared ones, operating systems, configuration management tools, and host level monitoring and logging are also part of the hardware layer. Host level logging and monitoring refers to logging and monitoring that happens on the machines in the hardware layer.

Communication layer holds everything related to communication between systems, services, and applications. It can be networks, DNS, remote procedure calls (RPCs), messaging between microservices, or API endpoints. Also, communication protocols such as HTTP, data formats (e.g., JSON), and traffic routing and distribution are part of this layer. Overall communication layer works together with every other layer between other layers to take care of everything communication related.

Application layer is for internal tools, services, systems, and platforms that microservices run on live. Things that are centralized, system-wide tools and services for developers to use. Standardized development process containing a good version control system, collaboration tool, and stable development environment as well as deployment pipeline, test, build, package, and release tooling are also part of this layer.

Microservice layer is the highest of layers and contains all the microservices and information specific to certain microservice such as configurations. Almost all the development work happens on this layer. In this work, layers are defined based on Fowlers' article [2016] which is summarized as following:

Layer 1: The Hardware Layer

- Configuration management tools
- Databases
- Servers
- Host-level logging and monitoring
- Operating Systems
- Resource isolation
- Resource abstraction

Layer 2: The Communication Layer

- DNS
- Endpoints
- Load balancing
- Messaging
- Network
- Remote procedure calls
- Service discovery
- Service registry

Layer 3: The Application Platform

- Deployment pipeline
- Development environment
- Microservice-level logging and monitoring
- Self-service internal development tools
- Test, package, build, and release tools

Layer 4: The Microservice Layer

- All microservice-specific configurations
- The microservices

2.3 Integrations in microservice architecture

In microservice context, light and heterogeneous protocols are possible to take into use for service interaction. Each service only knows its own data and maintains context for it. Because of this, duplicate data and functionalities are possible across services. In microservice architecture, there is no general context or centralized governance over all services. This means that deployment dependencies are also service specific and on a much lower scale than if they would be on a general level. [Cerny et al., 2017]

2.4 Advantages of microservice architecture

Compared to monolith type of architecture, where there is a one large system that contains all the functionalities and information system needs to operate, microservices and their distributed nature are more efficient in scalability, elasticity, and in automated and continuous deployment. [Kratzke & Quint, 2017] These abilities enable fast demand response, fault tolerance since service failure do not automatically mean system failure and make it more cloud-friendly [Balalaie et al., 2016]. Microservice architecture is also closely linked with DevOps practices, especially continuous delivery [Bass et al., 2015].

2.5 Disadvantages of microservice architecture

The price of this kind of flexibility is that data definitions, and in some cases even business rules need to be restated and redefined across services. This can lead to duplicates in databases, and lack of centralized view on how the overall system processing, its rules and constrains, work. [Cerny et al., 2017]

In microservice architecture, services are connected through point-to-point integrations. There is no integration component that would take responsibility for service orchestration and the business logic is embedded in services. On the one hand, therefore making changes to existing business logic pose a challenge with microservice architecture. On the other hand, since every service is responsible for its own integrations, there is a lot of flexibility to do modifications to single integrations.

2.6 Challenges in microservice architecture

Based on existing studies, data management and consistency, performance prediction, measurement, and optimization, decomposition, service orchestration and discovery, communication and integration, testing and security, monitoring, tracing, and logging were mentioned as identified challenges in microservice architecture by Söylemez et al. [2022].

Soldani et al. [2018] on the other hand divide identified challenges into three category: design, development, and operation. Design ones are Architecture and Security design. Architecture design includes Communication heterogeneity, API Versioning, Service contracts, Service dimensioning, and Size/complexity. Security design focuses on aspects such as Access control, Centralised support, CI/CD, Endpoint proliferation, Human errors, and Size/complexity. For development, Soldani et al. [2018] name Microservice development, Storage development, and Testing development. Microservice development encounters challenges with Microservice separation and overhead, Storage battles with Data consistency, Distributed transactions, Heterogeneity, and Query complexity. Testing sub-category brings up Integration testing, Performance testing, and Size/complexity.

Third and last category from Soldani et al. [2018] study is operation. This refers to Management, Monitoring, and Resource consumption operations. For Management, this means Operational complexity, Cascading failures, Service coordination, and Service location, for Problem location, Monitoring, Logging, and Size/complexity, and for Resource consumption Compute, and Network resource consumption.

3 Requirements in software development

Requirements are attributes that define a collection of needs and how software should perform in order to achieve the end goal and fulfil the original need for the software. That need is identified by a customer, end-user, customer representative, or other stakeholder. Requirements are also one of the key contributors when it comes to the success of a software project. [Aurum & Wohlin, 2005]

IEEE-STD 610.12-1990 standard [1990] definition for requirement is following:

(1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(3) A documented representation of a condition or capability as in (1) or (2).

3.1 Requirements and quality

Requirements are a critical determinant of the quality of the software and defining them is a crucial stage in software design and development. While some studies show that errors related to requirements are the most common ones in the software life-cycle, they are also the most time-consuming and expensive ones to correct. [Aurum & Wohlin, 2005]

3.2 Requirements engineering

There are two perspectives and three levels each in which requirements can be viewed; first one is from technical, and other one from management perspective. Some of the requirements in these two categories may distinct or overlap. Requirements at the technical perspective can be at the project, product, or at organizational level. From the management perspective, requirements can be seen to belong at the operational, tactical, or strategic level. [Aurum & Wohlin, 2005]

Requirements engineering (RE) refers to all life-cycle activities related to requirements. It is the process in which the requirements for the software are gathered, analysed, documented, and managed throughout software’s lifecycle. It may contain interpretation, structuring, verification, negotiation, validation, and tracing of the requirements. [Aurum & Wohlin, 2005]

Purpose of requirements engineering is to identify the goals for a system, transforming those goals into features and constraints, and further on, allocating those resulting requirements to propriate targets. That target can be a human, device, or software. Requirements can change and evolve during software life-cycle, in the development phase and after publishing. This brings out a need for change management over time. To support the evolution, monitoring requirements and managing the project scope, cost and schedule is essential. [Aurum & Wohlin, 2005]

Requirements are specified using System requirements analysis (SRA). SRA is a systematic approach and structured methodology used to identify ways to fulfil a system need and the essential characteristics (requirements) for those solutions [Grady, 2014]. It is a critical task in a software project, and the development team must investigate and study the problem domain in order to better understand the goals, expectations and needs of the projects' stakeholders. [Aurum & Wohlin, 2005]

3.3 Stakeholders

Stakeholders are a primary source of requirements. They can be an individual, group, or even organization and have diverse backgrounds, different needs, goals, expectations, and priorities. Reconciling these requirements from various stakeholders, at the same time as software project is growing to be more and more complex, is a challenge in requirements engineering. Understanding stakeholders needs and describing them as accurately as possible is where quality management begins in software development. [Aurum & Wohlin, 2005]

The purpose of requirements engineering is to perfect stakeholders potentially incomplete, inconsistent, or conflicting requirements into a form that they are unambiguous, high quality, and align with each other. [Aurum & Wohlin, 2005] Stakeholders are an important part of the development process. They represent the original need for the system, product, or project. Stakeholders are the ones that are somehow influences by the system, by its' use or development, directly or indirectly [Pouloudi & Whitley, 1997].

3.4 Different levels of requirements

This study presents two perspectives and three levels each in which requirements can be viewed; first one is from technical, and other one from management perspective. Some of the requirements in these two categories may distinct or overlap. Requirements at the technical perspective can be at the project, product, or at organizational level. From the management perspective, requirements can be seen to belong at the operational, tactical, or strategic level. [Aurum & Wohlin, 2005]

In general, it is important for software project to stay in budget, be ready in time, achieve proper performance and functionalities, and make sure that software requirements and business goals match. In order to succeed in it is essential to have the specifications for requirements properly structured and controlled. A good specification for a requirement is consistent, understandable and comprehensive. [Aurum & Wohlin, 2005]

3.5 Requirements Classification

Requirements can be classified into different categories that disclose what is the requirement related to. A fundamental division is into *functional* and *non-functional requirements*. Functional requirements describe what the system will do. Non-functional requirements are less exact requirement that form a constraints that can be led into more specific, functional requirements. Examples of non-functional requirements are performance, security, and accuracy. [Aurum & Wohlin, 2005]

Requirements can also be classified into *primary* and *derived requirements*. Primary requirements come from stakeholders and derived ones are derived from primary requirements. Other classification ways can be division into product and process requirements, or requirements based on role such as customer requirements, security requirements, user requirements, system requirements, or IT requirements. [Aurum & Wohlin, 2005]

Monitoring and documenting the progress of fulfilling requirements is also important. If the project team is not able to maintain traceability and verification hooks, it is challenging to try to determine whether or not the requirements are fulfilled by the system at the end [Grady, 1994].

4 Related work

This chapter evaluates and provides an overview of the existing work of SLRs on microservice architecture. Table 1 compares eight systematic literature reviews related to microservice architecture, their time span of search, amount of selected primary studies, research questions, and findings. Studies are presented in the order they were found from Google Scholar search made to browse for related work.

Many studies related to microservice architecture focus on common patterns and practices that are used in microservice architecture. Studies also consider architectural applications and principles of architectural patterns in microservice migration practices. [Li et al., 2021]

Study	**Time span**	**Sample size**	**Research questions**	**Finding**
Li et al., 2021.	- 2021	72	RQ1: What are the most concerned QAs for MSA? RQ2: What tactics have been proposed or discussed to improve the most concerned QAs of MSA?	RQ1: Scalability and performance being two most concerned QAs, availability, monitorability, security, and testability gaining the least concern. Out of the identified QAs, a correlation was recognizing between performance and scalability and monitorability and scalability. RQ2: Out of the six QAs mentioned by RQ1, 19 tactics were mentioned. Two of them were scalability tactics, four for performance, four for availability, four for monitorability, three for security, and two were tactics for testability.
Aksakalli et al., 2021.	2014 - 2019	38	RQ1: What are the currently adopted deployment approaches for microservices? RQ2: What are the currently adopted communication patterns for microservices? RQ3: What are the identified issues and obstacles related to the communication and deployment of microservices? RQ4: What are the identified research directions concerning	RQ1: Serverless deployment, Service instance per VM, and Service instance per container. RQ2: Synchronous communication, Publish/subscribe communication, combination of HTTP and Message queue, communication using message-oriented middleware, Asynchronous communication, Point-to-point communication, and communication using binary protocols. RQ3: Deployment challenges: architectural complexity, independently failed microservices (fault tolerance), deployment coordination, distributed logs, deployment cost, container image vulnerability, required specific configurations, and Continuous Integration/Continuous Delivery (CI/CD).

			communication and deployment of micro-services?	Communication challenges: discovery of services, replicated service instances, load-balancing, replicating data, remote calls, and the relation between tables. RQ4: Complexity control, monitoring, services resilience, efficient logging, performance improvement, minimizing service dependencies, fault tolerance, enabling security, and automated deployment system.
Razzaq, 2020.	2014 - 2019	141	RQ1.1: What is the frequency of the publication in software architecture and IoT software over the years? RQ1.2: How many publications per year are found in the research area of software architecture for IoT software? RQ1.3: Which are the foremost venues of the research area for publications? RQ2.1: What are the current challenges reported in the literature of IoT software? RQ2.2: What software architectural solutions have been proposed for IoT software? RQ2.3: What type of MSA based software architecture and design patterns are available for IoT software? RQ3: What are the primary inspirations to adopt the MSA for IoT software in the Ocean?	RQ1.1: Between 2005 and 2020, the most critical launches are in 2017 and 2019. Most of the publications are conference papers. The trend has been rising since 2005 and during the last five years 88% of the studies related to software architecture and IoT in the context of software architecture process were published. RQ1.2: From 2005 to 2011 and 2020, one per year. 2012 and 2013 two per year. 2014 got seven, 2015 eleven, and 2016 sixteen. 2017 and 2019 twenty-six, the peak year been 2018 with 42 publication. RQ1.3: Publication venues for publications published between 2005 and 2020, 76% of them were conferences, 19% journals, and 5% workshops. RQ2.1: Data integration and scalability are the most highlighted ones in the primary studies. Overall, twenty-three challenges were identified. RQ2.2: Thirty-three architectural solutions were identified that help to mitigate identified challenges. RQ2.3: Architectural and design patterns for MSA for IoT identified are listed at Table 2. RQ3: Result for this research question is not stated in the study.
Campeanu, 2018.	2015 - 2018	364	RQ1: How many publications per year are found in the research area usage of micro-service architectures by	RQ1: 2015 there were 33 publications, 2016 123 publications, 2017 197 publications, and by the January 2018 11 publications. This indicates an increasing interest towards the research topic.

			IoT and cloud computing solutions? RQ2: Which are the main venues for the publications of the research area? RQ3: Which are the main publication types in the research area?	RQ2: 5 main publication venues were identifies being SOSE (conference), CLOUD (conference), CCGRID (conference), UCC (conference), and CloudCom (conference). RQ3: A clear majority of publications were published in conferences (349 out of 264) and rest of them (15) in journals.
Söylemez et al., 2022.	2014 - 2022	85	RQ1: What are the identified challenges of microservice architectures? RQ2: What are the proposed solution directions?	RQ1: (1) Service discovery, (2) data management and consistency, (3) testing, (4) performance prediction, measurement, and optimization, (5) communication and integration, (6) service orchestration, (7) security, (8) monitoring, tracing, and logging, and (9) decomposition. RQ2: (1) Using information-centric networking (ICN), (2) multi-agent-based framework to coordinate distributed transactions of the system, (3) automated method of regression tests, (4) adaptive performance simulation approach, (5) event-driven lightweight platform for microservice orchestration, (6) Elasticsearch to solve auto-scalability issues, (7) access control optimization model that is based on role-based access control (RBAC), (8) graph-based microservice analysis and testing (GMAT), and (9) domain-driven design principles.
Guo & Wu, 2021.	2002 - 2021	560	RQ1: What kind of smells related to cloud applications and Microservices are described and detected in scientific literature? RQ2: Are the impact and refactoring approach of such smells discussed in literatures? RQ3: What would the future development of cloud- and service-related smell research be like?	RQ1: Infrastructure and Configuration Smell (IaC) results in increasing size and complexity of the associated code, Microservice Smell create a need for multi-directional communication protocol, technical implementation problems, and high-level structural problems in architecture and endpoints, and automatic refactoring of Code Smells results in increased resource usage. RQ2: Yes, literature works generally consist of smell detection, impact analysis, and smell refactoring.

				RQ3: There's no clear answer to this. From the smell point of view IaC and micro-service smell detection, and different detection approaches.
Santana et al., 2018.	2015 - 2018	18	RQ1: What are the studies published by the scientific community on the adoption of Microservices in the development of IoT applications? RQ2: What are the main contributions of the identified studies? RQ3: What are the perspectives and trends of research in the area of Microservices in the context of IoT?	RQ1: 18 published studies on that topic are listed in the original work. RQ2: Those published studies show that design is predominant phase when architecting microservices for IoT or applying data solutions in cloud computing and fog computing. Regarding gradual maturity, more work is needed to investigate complexity in implementation, maintenance, evaluation, and operation phases. Contributions and problems of each study are also presented. RQ3: In time, research fields associated with microservices, such as machine learning, fog computing, CI/CD, containers, reactive systems, and formal methods for specification of microservices will be explored.
Soldani et al., 2018.	2014 - 2017	51	RQ1: How much evidence of microservices experimentation from industry is available online? RQ2: What are the technical and operational "pains" of microservices? RQ3: What are the technical and operational "gains" of microservices?	RQ1: Microservices are gaining attention steadily over the time period since 2014 thanks to James Lewis and Martin Fowler. Contributions mainly focus on the pros and cons of microservices, and the best practices and patterns to better leverage microservices. IT companies are also increasingly focusing on their consultancy portfolio on microservices. RQ2: Architecture design (API Versioning, Communication heterogeneity, Service contracts, Service dimensioning, Size/complexity), Security design (Access control, Centralised support, CI/CD, Endpoint proliferation, Human errors, Size/complexity), Microservices development (Microservice separation, Overhead), Storage development (Data consistency, Distributed transactions, Heterogeneity, Query complexity), Testing development (Integration testing, Performance testing, Size/complexity), Management operation (Cascading failures, Operational complexity, Service coordination, Service location), Monitoring operation (Logging,

				Problem location, Size/complexity), Resource consumption operation (Compute, Network). RQ3: Architecture design (Bounded contexts, Cloud native, Decentralised governance, Fault tolerance, Flexibility), Design patterns (API gateway, Circuit breaker, Database per service, Message broker, Service discovery), Security design (Automation, Fine-grained policies, Firewalling, Isolation, Layering), Microservices development (CI/CD, Loose coupling, Reusability, Service size, Technology freedom), Storage development (Data persistence, Data isolation, Microservice-orientation), Testing development (Automation, Rollback, Unit testing, Updates), Deployment operation (Containerisation, Independency, Reliability, Speed), Management operation (Fault isolation, Scalability, Updateability).

Table 1: Comparison of eight systematic mapping studies related to microservice architecture.

Li et al. [2021] focused on identifying the most concerned Quality Attributes (QAs) for microservice architecture, and tactics proposed to deal with the most concerned quality attributes of microservice architecture. The results listed scalability, performance, availability, monitorability, security, and testability as the quality attributes and linked tactics regarding each identified attribute. The study also mentions that while being one of the most commonly mentioned benefits of microservice architecture, it is also a necessary concern regarding QAs of microservice architecture. Also, maintainability is highlighted as a QA that would need more research attention for evaluation and effective improvement in the future.

Aksakalli et al. [2021] review deployment and communication in microservice architecture. Different approaches and patterns, issues and obstacles, and possible research direction related to those. The study identifies serverless deployment, service instance per container, and service instance per VM to be the most common deployment approaches for microservices. Regarding communication, synchronous or asynchronous communication, communication using message-oriented middleware or binary protocols, point-to-point or publish/subscribe communication, or combination of HTTP and Message queue are used as communication methods in the microservice context.

Obstacles and challenges related to communication and deployment are the most interesting findings when considering relation to our work. Aksakalli et al. [2021] considers these obstacles and challenges in two category: Deployment challenges and communication challenges. Deployment results highlighted architectural complexity, fault tolerance, deployment coordination, distributed logs, deployment cost, container image vulnerability, required specific configurations, and Continuous Integration/Continuous Delivery (CI/CD). Communication category on the other hand mentioned remote calls, discovery of services, replicating data, replicated service instances, load-balancing, and the relation between tables.

Future research directions needed for deployment and communication in microservice architecture are according to Aksakalli et al. [2021] complexity control, monitoring, services resilience, minimizing service dependencies, performance improvement, efficient logging, enabling security, fault tolerance, and automated deployment system. These research directions were mentioned to be ones that could help to resolve know challenges and obstacles related to communication and deployment of microservices.

Razzaq [2020] is a comprehensive systematic literature review focusing on publications on microservice architecture for IoT software. Study investigates the frequency of the publications over the years and in which venues are they published, current challenges of IoT software and the solution that have been proposed to those challenges, and microservice architecture based software architecture and design patterns that are available for IoT software. From 2017 to 2019 was identified as the peak season of these scientific publications and during 2015 to 2020, 88% of all studies between 2005 to 2020 were published. The most popular publication venues are conferences (76%) followed by journals (19%) and workshops (5%). Data integrity and scalability were highlighted as the most common challenges related to IoT software along with twenty-one other challenges. Thirty-three architectural solutions were presented as possible solutions to those challenges. Architectural and design patterns that traditionally help to mitigate these identified challenges are presented at Table 2.

Pattern type	**Pattern name**
Decomposition	Decomposing application
	Decompose by subdomain
	Self-contained services
Deployment	Service deployment platform
Communication style	Remote Procedure Invocation
	Messaging
	Communication pattern
External API	API gateway
	REST design pattern
Service discovery	Client-side discovery
	Service registry

Table 2: MSA architectural and design patterns for IoT [Razzaq, 2020].

Campeanu [2018] also maps the number of yearly publications in the usage of microservice architectures by IoT and cloud computing solutions, and the main venues and publication types of these studies. The study lists that 2015 there were 33 publications, 2016 123 publications, 2017 197 publications, and by the January 2018 11 publications of the usage of microservice architectures by IoT and cloud computing solutions. This indicates an increasing interest towards the research topic. Top five venues were also named and categorized as conferences. When it comes to publication types, a clear majority of publications were published in conferences (349 out of 264) and rest of them (15) in journals.

Söylemez et al. [2022] gets closer to our interest in this study as well. They view challenges and possible solution directions of microservice architectures. Service discovery, data consistency and management, performance prediction, testing, measurement and optimization, communication and integration, service orchestration, security, monitoring, tracing and logging, and decomposition were identified as challenges in microservice architecture. Following that, each challenges was paired with a possible solution direction. For service discovery it was using information-centric networking (ICN). Solution offered to coordinate systems distributed transactions, data management and consistency problems was multi-agent-based framework. In the microservice context, testing could benefit from automated method of regression tests. Adaptive performance simulation approach was there to help with the performance predictions. Event-driven lightweight platform for microservice service orchestration challenges and Elasticsearch to solve autoscalability issues. Security could be enhanced with access control optimization model that is based on role-based access control (RBAC) and graph-based microservice analysis and testing (GMAT) contributes to monitoring, tracing, and logging. Domain-driven design principles were mentioned to tackle decomposition challenges in microservice architecture.

Guo & Wu [2021] studies what kind of smells are related to cloud applications and microservices, and how are those described and detected in scientific literature? What is the impact and refactoring approach for those smells that is identified in the literature and what the future brings when talking about cloud- and service-related smells? Size and complexity were highlighted regarding infrastructure and configuration smells (IaC), the use of microservice architecture creates a need for multi-directional communication protocol, technical implementation problems, and high-level structural problems in architecture and endpoints. Automatic refactoring on the other hand increases resource usage. When talking about cloud and microservice smells, literature considers detection, impact analysis, and refactoring of the identified smells, but do not provide a clear answer to what is the future development of service- and cloud-related smell research. [Guo & Wu, 2021]

Santana et al. [2018] processes adoption of microservices in the IoT application development. What kind of studies have been published on the topic, what are the main contributions of those studies, and what are the trends and perspectives of that research area? This work listed 18 studies which showed that design is a predominant phase when architecting microservices for IoT or while applying data solutions in cloud or fog computing. More work research is needed regarding complexity in different phases of the development. Santana et al. [2018] lists machine learning, fog computing, CI/CD, containers, reactive systems, and formal methods for specification of microservices as a research are that will be explored in the future.

As the last one of these eight related studies, Soldani et al. [2018] talks about experimentation of microservices, what technical “pains” and “gains” do they have. Results show that microservices have steadily grown more popular over the years, one of major contributions being a blog post written by James Lewis and Martin Fowler [Soldani et al., 2018]. Pains (Table 3) and gains (Table 4) are categorized by design, development, and operation of microservices.

Category	**Area**	**Technology**
Design	Architecture	API versioning
		Communication heterogeneity
		Service contracts
		Service dimensioning
		Size/complexity
	Security	Access control
		Centralized support
		CI/CD
		Endpoint proliferation
		Human errors
		Size/complexity
Development	Microservices	Microservice separation
		Overhead
	Storage	Data consistency
		Distributed transactions
		Heterogeneity
		Query complexity
	Testing	Integration testing
		Performance testing
		Size/complexity
Operations	Management	Cascading failures
		Operational complexity
		Service coordination
		Service location
	Monitoring	Logging
		Problem location
		Size/complexity
	Resource consumption	Compute
		Network

Table 3: Technical pains of microservice architecture [Soldani et al., 2018].

Category	**Area**	**Technology**
Design	Architecture	Bounded contexts
		Cloud native
		Decentralised governance
		Fault tolerance
		Flexibility
	Design patterns	API gateway
		Circuit breaker
		Database per service
		Message broker
		Service discovery
	Security	Automation
		Fine-grained policies
		Firewalling
		Isolation
		Layering
Development	Microservices	CI/CD
		Loose coupling
		Reusability
		Service size
		Technology freedom
	Storage	Data persistence
		Data isolation
		Microservice-orientation
	Testing	Automation
		Rollback
		Unit testing
		Updates
Operation	Deployment	Containerisation
		Independency
		Reliability
		Speed
	Management	Fault isolation
		Scalability
		Updateability

Table 4: Technical gains of microservice architecture [Soldani et al., 2018].

Out of these eight SLRs, Razzaq [2020] focuses on MSA in software architecture and IoT software, Campeanu [2018] talks about the usage of microservice architectures in IoT and cloud computing solutions. Aksakalli et al. [2021] explore currently adopted deployment and communication approaches for microservices, Li et.al. [2021] while been closes to our study, is about the most concerned quality attributes of microservice architecture and how to improve them, Guo & Wu [2021] lists smells related to microservices and cloud applications, and Santana et al. [2018] talk about using microservices in IoT applications. Out of the related studies mentioned in this chapter, Soldani et al. [2018] and Söylemez et al. [2022] were closes ones to our study. When Soldani et al. [2018] talked about the pains and gains of microservice architecture, we are interested if the pains mentioned match with our findings. Söylemez et al. [2022] identified challenges and solutions of microservice architectures. It is interesting to compare solutions from these similar studies, are there result that our work can underline, have a conflict with or provide supporting information for?

5 Research methodology

This research is conducted as a systematic literature review (SLR). Methodology was selected to understand the depth and breadth of the existing knowledge, to identify possible gaps, and new possible research paths related to the challenges and requirements in microservice architecture.

Source material plays a key role in research. In terms of quality, research can at best be as good as the material enables. Quality of the reference research and writers' level of understanding of that research and its findings create a solid base for further research. There are three characteristics that systematic literature review should hold; it should be *valid*, *reliable*, and *repeatable*. [Xiao & Watson, 2017]

Systematic literature review can be used to achieve various outcomes. It can test a new theory or develop one, test an existing hypothesis, evaluate quality and validity of some existing work or try to find inconsistencies, weaknesses, or contradictions from those [Paré et al., 2015]. It can also describe how certain theory has evolved through time [Salminen, 2011] and summarize prior work or extend existing theories [Okoli & Schabram, 2010]. This work aims to summarize prior work and knowledge to help developers and stakeholders to plan and execute microservice projects successfully.

Because systematic literature reviews are contributions of scholarly knowledge, they should follow similar level of quality and discipline in study design as other literature [Templier & Paré, 2015]. This work uses Xiao & Watson [2017] description of systematic literature review as a guideline when designing the study.

At first, systematic literature review is presented as a method. This theoretical baseline guides us at the process. Next chapter gathers how this study was conducted in practice.

5.1 Types of literature reviews

Literature reviews fall into one of three categories: *traditional, meta-analysis, and systematic* [Salminen, 2011].

Traditional review gives a general, comprehensive background understanding of the literature and source material and research questions are broader than in other review types [Salminen, 2011]. It is used to identify new research possibilities or possible gaps or inconsistencies, refining and shaping research questions or develop theoretical and conceptual frameworks [Coughlan et al., 2007].

Meta-analysis approach can be divided into qualitative and quantitative review. It uses statistical methods to summarize the results of different studies and therefore can bring objectivity in evaluating research findings. [Salminen, 2011]

Systematic literature review, the one used in this study, summarizes relevant content of studies on certain topic. It maps the conversation and screens studies based on interesting and important scientific results [Petticrew, 2006]. Bearfield & Eller [2008] instruct to read a majority of material in a summarized format and put discoveries in context in relation to the research field. This way, it is easier to justify why certain source material is selected and included. Petticrew [2001] recommends systematic approach as a good way to test out hypothesis, present search results in a summarized way, and evaluate the consistency of those results. It is important to form a clear answer to the question at hand, reduce unclarity regarding the study selection, evaluate the quality of those included studies, and referencing objectively to the source material. One of the cornerstones in systematic approach is a decision making based on evidence [Metsämuuronen, 2005]. Being systematic creates criteria that brings scientific creditability to the review [Dixon-Woods et al., 2005].

5.2 Process of a systematic literature review

There are three phases in systematic literature review: *planning*, *conducting*, and *reporting* [Kitchenham & Charters, 2007; Breretona et al., 2007]. The planning phase includes identifying the need for review, specifying the research question, and developing protocol that describes how the review is conducted (Figure 1).

Review is conducted using eight steps:

1. Formulating the research problem
2. Developing and validating the review protocol
3. Searching the literature
4. Screening for inclusion
5. Assessing quality
6. Extracting data
7. Analysing and synthesizing data
8. Reporting the findings

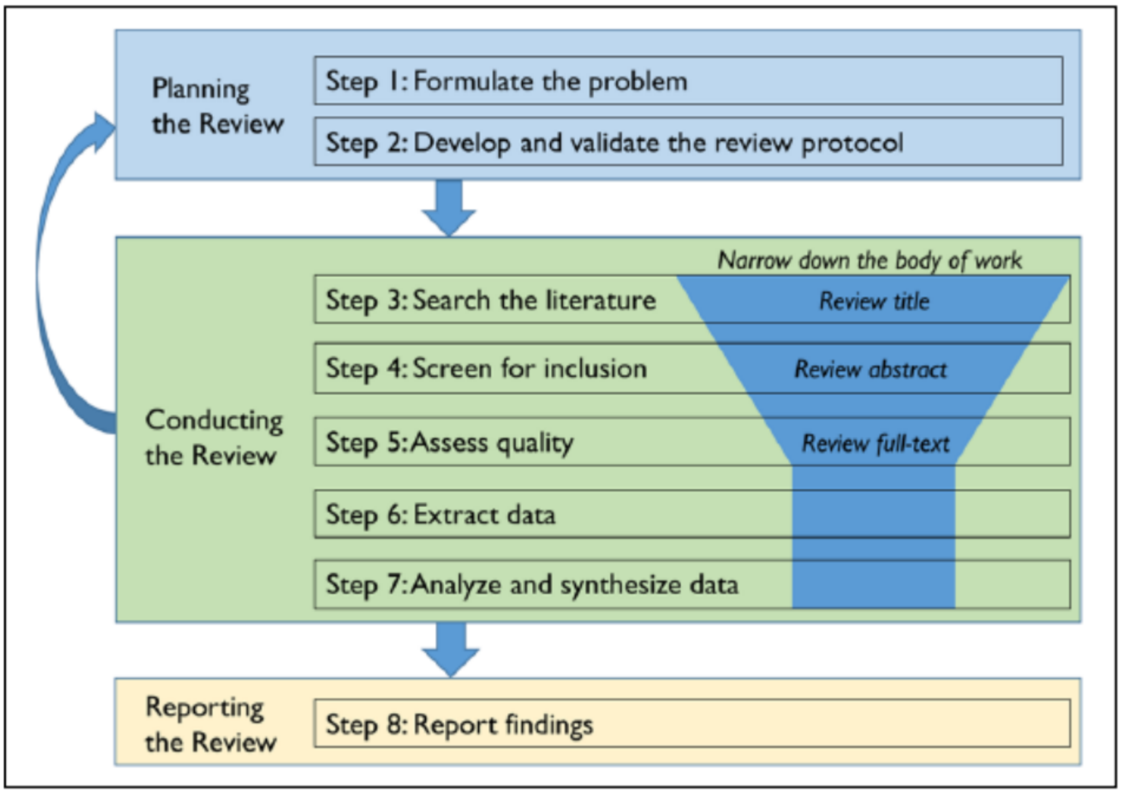
Planning the Review
Step 1: Formulate the problem
Step 2: Develop and validate the review protocol
Narrow down the body of work
Step 3: Search the literature
Review title
Step 4: Screen for inclusion
Review abstract
Conducting the Review
Step 5: Assess quality
Review full-text
Step 6: Extract data
Step 7: Analyze and synthesize data
Reporting the Review
Step 8: Report findings

Figure 1: Process of systematic literature review [Xiao & Watson, 2017].

5.3 Formulating the problem

First step is to formulate the problem. Whole literature review works together and align with the research question [Kitchenham & Charters 2007]. That research question acts as a guideline to the first round of literature search. It is possible for research questions to evolve over the process but getting as well-formed research question at the beginning can save some trouble.

5.4 Develop and validate the review protocol

Second step is developing and validating the review protocol. Producing a review protocol is the first thing that needs to be done. It is a critical part of a strict systematic review process [Okoli & Schabram, 2010; Breretona et al., 2007] in terms of achieving quality since its' role is to minimize the effect of researchers own opinions when selecting and analysing data [Kitchenham & Charters, 2007]. Gates [2002] describes review protocol and its' content as following:

"The review protocol should describe all the elements of the review, including the purpose of the study, research questions, inclusion criteria, search strategies, quality assessment criteria and screening procedures, strategies for data extraction, synthesis, and reporting."

Evaluation of the review protocol gives the study more quality and creditability. One option is to get an external review for the protocol [Xiao & Watson, 2017].

5.5 Search for literature

The third step is the search for literature. Quality literature is in the key role in systematic literature review. Extracted results can only be as good as the source material. Therefore, systematic review can only be systematic if the search for literature is systematic [Xiao & Watson, 2017]. According to Xiao & Watson [2017] the literature search consists of five distinct steps:

1. Channels for literature search
2. Keywords used for the search
3. Sampling strategy
4. Refining results with additional restrictions
5. Stopping rule

5.5.1 Keywords for the search

Essential keywords come from the words in research question [Kitchenham & Charters, 2007] or synonyms or alternatives to those words [Rowley & Slack, 2004; Kitchenham & Charters, 2007]. Cultural differences and context also play a role when thinking about terminology. Another culture may have a different word for a same thing in another culture. [Bayliss & Beyer, 2015] To optimize the search even more, operators such as combinators, word options, and sub-search (new search from the results of the first search) can be used [Fink, 2005]

5.5.2 Sampling strategies

Different kind of sampling logics and search strategies can be used with literature reviews [Suri & Clarke, 2009]. Search for source material can be comprehensive and exhaustive or representative and selective [Bayliss & Beyer, 2015; Paré et al., 2015; Suri & Clarke, 2009]. Different strategies work in different use cases, search styles, approaches, and purposes for the review: in scoping review purpose is to map the entire domain, testing review aims to produce generalizable findings, and some has a goal to extend the existing body of work [Kitchenham & Charters, 2007; Xiao & Watson, 2017].

Scoping review requires comprehensive and an exhaustive search of literature and grey literature (conference proceedings, theses, reports) are included. For example, meta-analysis reviews require a comprehensive search and are more selective in terms of quality than scoping reviews so grey literature might not be included. Experimental reviews can be more selective and purposeful with their source selection. [Kitchenham & Charters, 2007; Xiao & Watson, 2017].

5.5.3 Refining search results

Additional restriction may be added to the filtering of search results. These restrictions are for example based on publications language, publishing date, or source of financial support. [Kitchenham & Charters, 2007; Okoli & Schabram, 2010] As a good rule, only include publications that are written in a language reader can understand, delimit certain publication periods like too old ones, and consider the source of funding and if that might bring some biases to the research [Fink, 2005].

5.5.4 Stopping rule

Stopping rule is that search for additional sources should stop when search results are giving no new results [Levy & Ellis, 2006]. When continuing would not provide any additional information.

There are three approaches used to find additional source material based on selected articles from the database search: a backward search (backward snowballing), a forward search (forward snowballing), or a hybrid of backward and forward search.

In backward search, reference studies of the articles selected from the database search are gathered and evaluated based on the inclusion and exclusion criteria as it was done to the database search results. Included articles continue with included articles from the database search to the next phase. The purpose of this is to identify relevant work that publications and articles are using as their base work either directly or indirectly.

In forward search, instead of taking the reference list of already selected articles, we take articles that reference to those selected articles. Forward search results go through the same operation as did database and backward search results; they are compared against the inclusion and exclusion criteria and included ones continue to the next phase together with other selected studies. The idea is to find authors that have made major contributions on the research field and are the source of the latest or most important discoveries. By going through results of different searches we can find authors that contribute majorly to the research and are the experts in this field. Existing systematic literature reviews are also a good starting point for future reviews. [Kitchenham & Charters, 2007]

According to Mourão et al. [2020], hybrid search strategies provide better performance and the most appropriate balance between precision and recall when acquiring sources for systematic literature review compare with database search or snowballing alone. It also mitigates the risk of missing relevant papers due to a poor selection of search terms. The source also concluded that applying database searches in several digital libraries and following it by an iterative backward and forward snowballing, results in improved changes of identifying relevant studies even though with the risk of adding more effort to the review process.

5.6 Screening for inclusion

Fourth step in the process is to go through search results gathered in an earlier step and screen through what kind of material was acquired. Generally, a good approach is to first go through the material with a rough filtering based on articles abstract and follow that with a more precise one based on full text. At this point, the purpose is to filter out as much unnecessary articles as possible [Okoli & Schabram, 2010]. At the same time, when in doubt whether or not to include a study, it is usually better to include it [Xiao & Watson, 2017].

Filtering criteria, for inclusion/exclusion, should be based on the research question [Kitchenham & Charters, 2007; Okoli & Schabram, 2010]. Table 5 provides guidelines on what inclusion or exclusion criteria can be based on. A list of excluded articles should be created for reproducibility, crosschecking, and record keeping [Kitchenham & Charters, 2007]. Same goes for including duplicate search results into the final documentation [Mourão et al., 2020].

Criteria	**Example**
Data sources	primary vs. secondary data
Duration	long-term vs. short term impacts
Geographic areas	developed vs. developing countries
Measurement	subjective vs. objective
Research design	quantitative vs. qualitative
Sampling methodology	random sample vs. convenience sample
Type of event	hurricanes vs. earthquakes
Type of policy	Euclidean zoning vs. form-based codes
Unit of analyses	individual household vs. the entire community

Table 5: Criteria examples for inclusion and exclusion of research [Xiao & Watson, 2017].

5.7 Assessing quality

In step five, quality of the selected articles is inspected more closely. According to Ludvigsen et al., [2016] purpose of the quality assessment is to understand the chosen articles before continuing to the next step of comparing and integrating findings. There is debate on whether studies should be similar enough in methodological point of view.

Some suggest that studies should be similar enough compared to each other to be able to draw meaningful conclusions later in review methods [Okoli & Schabram, 2010; Gates, 2002]. Others raise a point that excluding a large number of articles just based on poor methodological quality might lead to selection bias and this way diminish the generalizability of review findings [Suri & Clarke, 2009; Pawson et al., 2005].

Quality assessment often refers to reviewing internal validity of the study. Study is internally valid if it has none of the main methodological biases. One approach is to categorize studies qualitatively on high, medium, and low categories. Major arguments and research synthesis should base on high-quality studies before using medium-quality sources. Low-quality sources can be used as a supplement but not in a foundation. [Petticrew & Roberts, 2006]

5.8 Extracting data

In step six, ways to synthesize data are presented. Different literature review typologies and types being synthesized guides researcher towards proper synthesizing method. This includes reporting the knowledge so far, showing there is a need for the research, explaining what is found, and describing the quality of the research. [Fink, 2005] Method in turn, guides towards the data extraction process.

Coding is often involved in a data extraction process. It is important to define if the coding will be based on the data or pre-existing concepts [Suri & Clarke, 2009]. The way the review is coded has a straight impact on the end conclusions. For example, in extending reviews where work focuses on finding new perspective and synthesising a large collection of data instead of collecting new one, conclusions and generalizations are made based on concepts and themes that are coded. If this coding is failed by doing it incorrectly or inconsistently, the review has less validity and reliability. Stock et al. [1996] state that if item is not coded, it cannot be analysed. They also recommend using codebook and having well-designed forms. Well-designed forms increase efficiency and lower the number of judgements that individual reviewer needs to make, and that way reduce possible errors. [Stock et al., 1996] It is important that the researcher evaluates the whole research paper and not just results or main conclusions. It is the only way to provide context to the findings and avoid misunderstandings. [Onwuegbuzie et al., 2012]

5.9 Analysing and synthesizing data

Once the necessary data is extracted, it is time to start organizing it. Organizing is done based on the choosing criteria, the reason why study was originally included. With qualitative studies, analysing is done by finding descriptive themes and filtering them into analytical themes. With quantitative studies, findings are combined based on meta-analyses. [Xiao & Watson, 2017] Contingent designs defining feature is how the research synthesis studies together and combined are able to answer to the raised questions from previous syntheses. Not the grouping of studies or methods as quantitative or qualitative. [Sandelowski et al., 2006] This is a part that people fail most often by synthesizing being too shallow [Baumeister & Leary, 1997].

5.10 Reporting findings

Detailed reporting contributes to reliability and repeatability. To make sure that literature review is independently repeatable and reliable, the process must be reported in sufficient detail. [Okoli & Schabram, 2010] So, by following the same steps anyone should end up into same conclusion. Especially the inclusion and exclusion criteria should be specified in detail and reasoning for each criteria should be explained in the report [Peters et al., 2015; Templier & Paré, 2015]. There should be information about the findings from literature search, screening, and quality assessment [Noordzij et al., 2009]. The literature review should have a clear structure that combines studies together into key themes, subgroups or characteristics [Rowley & Slack, 2004]. All in all, instead of focusing on how strict or loose criteria is used, more important aspect is the transparency of the process, and that conclusions are supported by the data.

It is important to be aware of the risks of adding subjectivity and making assumptions from the data accordingly based on the purpose of the review. All new findings and unexpected results should also be broad up [Okoli & Schabram, 2010] in addition to the opportunities and possible directions for future research [Okoli & Schabram, 2010; Rowley & Slack, 2004].

5.11 Good practices

There are some good practises to follow when conducting a literature review. First, start with a research questions. The entire process (literature search, data extraction, analysis, and reporting) is based and reflected on that questions [Kitchenham & Charters, 2007]. Second, clarify what you want to achieve by this review and choose a review type that serves that purpose. After choosing the purpose, researchers can select a typology to follow and that way the appropriate review methodology. Thirdly, plan before doing. Developing a review protocol is a crucial starting phase for strict systematic reviews [Breretona et al., 2007; Okoli & Schabram, 2010]. The protocol reduces possibility for biases in data selection and analysis [Kitchenham & Charters, 2007] and enables repeatability, cross-checking, and verification later on and that way increases the reliability of the review. As a fourth step, aim to have a comprehensive literature search and mind the quality of literature. The search should be comprehensive and sources up to date, so using multiple databases, backward and forward searches, and expert consultation in the field is advised. It is also important to understand the findings of individual study before comparing and integrating findings together [Ludvigsen et al., 2016]. Fifth is to be cautious, flexible, and open-minded to new concepts and ideas that might appear. Because of reviews' possibly iterative nature, the review can first be piloted, and additional specifications added before making the final decisions concerning the process design. Sixth, document made decisions during the review process. To make review reliable and repeatable, it is required that the process is documented and made transparent. [Xiao & Watson, 2017] Seventh, teamwork is recommended during the review process with a minimum of two reviewers for the literature inclusion [Kitchenham & Charters, 2007]. This way it is not just one person's view on which studies to include, what are the best practices for the review process, and is the quality of the review held according to the set standards. Lastly, software such as RefWorks can provide assistance in a review process. It can be used to manage references, citations, and bibliographies. [Xiao & Watson, 2017]

6 Research methodology execution

As stated in Chapter 5, this study follows the theory of systematic literature review described earlier in this study. This chapter states how the methodology was followed in practise. Figure 2 provides an overview of the process and number of studies filtered, screened, or read in full in each state of the process.

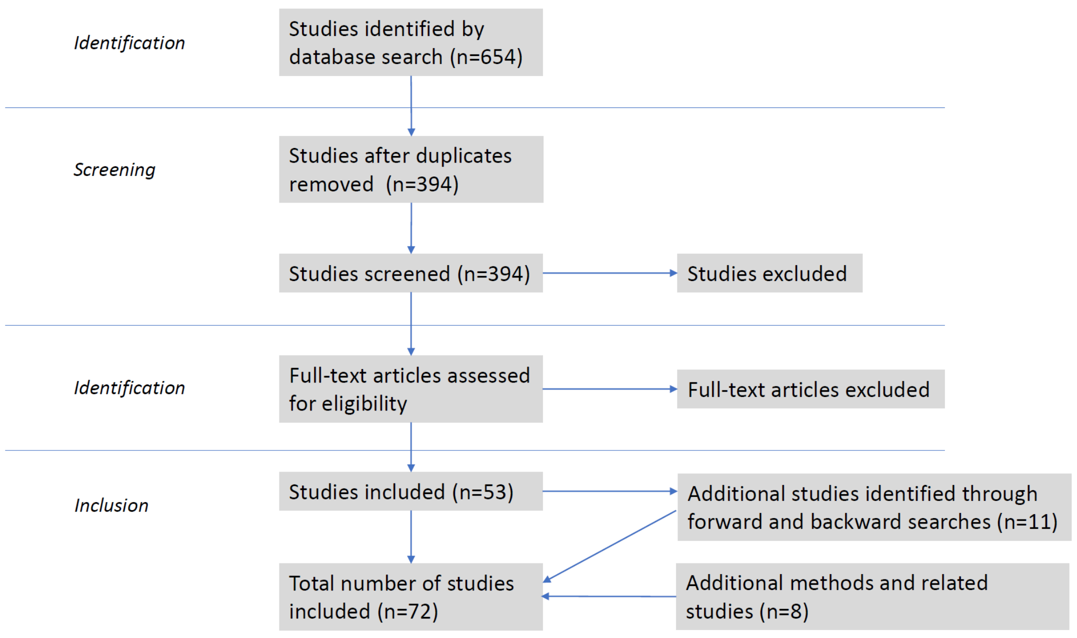
Identification
Studies identified by database search (n=654)
Screening
Studies after duplicates removed (n=394)
Studies screened (n=394)
Studies excluded
Identification
Full-text articles assessed for eligibility
Full-text articles excluded
Inclusion
Studies included (n=53)
Additional studies identified through forward and backward searches (n=11)
Total number of studies included (n=72)
Additional methods and related studies (n=8)

Figure 2: An overview of the research methodology in practise.

Number of studies from the database search was 654. After removing the duplicate article, the number was cut down to 394. Each article was screened for inclusion/exclusion. Studies included for the screening were read in full and eventually 53 studies were included from the original database search. Based on those selected articles, backward and forward searches were conducted in order to find additional sources. Results of backward and forward searches went through the same screening and reading process as the database search studies and at the end, 11 of them were selected. After having a set of 63 selected studies, related work was searched and that resulted in 8 related studies using systematic literature review or systematic mapping.

6.1 Formulating the research problem

Research methodology chapter describes the process of systematic literature review. Research question has a crucial role in that process. Quality and clarity of the research question can have an effect on every step following in the process. From the review protocol to synthesizing results and to the quality of analysis.

1. Clarify the research objectives and identify the research questions,
2. define the search string,
3. pilot the search for articles in a database like Scopus to search, and
4. consider the inclusion/exclusion criteria.

The goal of the research is to recognize system integration requirements in a microservice architecture. To achieve the goal, we may think questions such as:

- What type of systems are reported to adopt a microservice architecture?
- What are the different types of the microservice architecture?
- What are the integration challenges or obstacles of microservices?
- What methods or techniques have been discussed or proposed to tackle the challenges?

Like Figure 1 showed, process can have an iterative nature. If something is realised or learned or problems identified while conducting review, we can go back to planning phase and adjust the process according to the new information. These adjustments can concern some part of the protocol or even change the research question. One of the situations where research question would require changing is when realised that the question is too broad which leads to having too much data and that data being unmanageable [Cronin et al., 2008]. One way to deal with too broad research question is to identify subtopics inside question after the first literature search, count the number of studies in each subtopic and based on that information decide whether or not to narrow down the research question [Breretona et al., 2007].

This study experienced evolution regarding the research questions but search scope remained the same. After the first database search, search results did not provide adequate results, and research questions were delimited. Original questions were:

- What are integration mechanisms for microservice projects?
- What are common integration challenges in microservice architecture?
- How are common integration challenges in microservice architecture addressed?
- What requirements should be set to a microservice integration project?

Challenges and requirements related to integration in microservice context proved to be too restrictive combination, so parameters were iteratively redefined to consider challenges and requirements related to microservices in general. Final research questions are:

RQ1: What are common challenges in microservice architecture?
RQ2: How are common challenges in microservice architecture addressed?
RQ3: What requirements should be set to a microservice project?

The search scope for this study is microservice, microservice architecture, requirements, and challenges.

6.2 Developing and validating the review protocol

Final review protocol used for this work was validated by the Thesis instructor. Developing a review protocol got an iterative nature, but it is important to keep in mind that not everything can be predicted beforehand. More important is to assess and re-evaluate the protocol throughout the process. Protocol selected for this study will be presented in the following chapters.

6.3 Searching the literature

Four scientific databases were eventually used to act as the channels for high quality publications. Scopus was selected to act as the first database. This was based on results published in Mourão et al., [2020] where Scopus delivered the highest overall precision and recall compared to IEEE Xplore, ACM Digital Library, Springer , Web of Science, ScienceDirect, Compendex, Google Scholar, and Wiley online library. Additionally, ACM Digital Library and Web of Science were mentioned to provide good result, although less consistently. Selected databases for this work are **Scopus, ACM Digital Library, IEEE Xplore,** and **Web of Science**.

6.3.1 Keywords for the search

Following search string used for the search in the databases:

microservice* AND integrat* AND (requirement* OR constraint* OR obstacle* OR challenge* OR barrier* OR aspect*)

It is constructed following the instructions mentioned above. Essential keywords such as microservice, integration, requirement, and challenge come from the research questions. Synonyms and alternatives to the word ‘challenge’ such as ‘constraint’, ‘obstacle’, and ‘barrier’ are used to widen the range of results. This was also reason for using Asterix (*) to include variations of certain keyword into the search. AND-combinators are used to maximize the percentage of search results being relevant.

Earlier iterations of the search string included multiple synonyms for ‘microservice’ that were later replaced with an Asterix. Search result including ‘µService’ were experimented in Scopus but did not result in any additional search results, so the term was excluded from the final search string. Also, the emphasis on term ‘requirement’ was softened. Different iterations of the search string are presented below.

Version 1:
system integration AND requirements AND (microservices OR microservice-based application OR microservice architecture OR µService) AND (obstacle OR challenge OR barrier OR critical aspect)

Version 2:
integration AND requirements AND (microservices OR microservice-based application OR microservice architecture OR µService) AND (obstacle OR challenge OR barrier OR critical aspect)

Version 3:
integration AND requirement AND (microservices OR microservice-based application OR microservice architecture OR µService) AND (obstacle OR challenge OR barrier OR critical aspect)

6.3.2 Sampling strategies

Purpose of this study is to identify recurring aspects from different sources and form generalizable findings based on those recurring results. In this case it means finding recurring challenges that projects phase, possible solution to those challenges, and identify requirements that selecting microservice architecture brings to the project.

6.3.3 Refining search results

Since microservice is relatively new as a concept [Lewis & Fowler, 2014], publication date was not limited. Language selection for publications is English and based on title and abstract, they need to be related to the topic. Source of financial support was not considered.

6.3.4 Stopping rule

In practise, search for additional sources ended according to the original plan. Digital library search was conducted to the four selected scientific databases. Out of the search results, essential articles were selected based on defined inclusion (IC) and exclusion criteria (EC). Following that, backward and forward snowballing is conducted to the selected articles to ensure best possible coverage.

Database search resulted in 654 studies (Table 6) and most of them are from Scopus. After screening the results for inclusion and reading the selected one in full, backward and forward snowballing was performed for 53 studies selected from the database search. Backward snowballing provided 1153 search result (Table 7) and forward snowballing 306 search results (Table 8) that were screened for inclusion and selected ones read in full.

Database search	Number of search results
Scopus	338
IEEE	109
Web of Science	171
ACM	36
Total	**654**

Table 6: Number of search results from each scientific database.

Backward snowballing	Number of search results
Scopus	941
IEEE	60
Web of Science	65
ACM	87
Total	**1153**

Table 7: Number of search results from backward snowballing in scientific databases.

Forward snowballing	Number of search results
Scopus	267
Web of Science	20
ACM	19
Total	**306**

Table 8: Number of search results from forward snowballing in scientific databases.

6.4 Screening for inclusion

In this study, articles resulted in the database search were filtered for duplicates, screened for inclusion/exclusion based on title and abstract. Articles resulted from snowballing were also filtered for duplicates, then screened for inclusion/exclusion first based on title, included ones screened again based on abstract, and after that read in full.

Inclusion criteria (Table 9) states that publication should investigate the relevant topic. In addition to that, publications reporting application implementation using micro-service design and development, presents specific application implementation, or paper that are using microservice-based architecture and focuses on designing the application and in the context of microservice architecture.

Exclusion criteria (Table 9) states that papers that are not written in English, focuses on process integration, inter-application integrations, and integrations non-microservice context. Study is not interested in research plans, roadmaps, nor vision papers. Duplicates, non-peer reviewed and out of topic articles, systematic literature reviews, and ones focus-ing on service oriented architecture (SOA) are also excluded.

Criteria type (Inclusion/Exclusion)	Description
Inclusion criteria (IC)	investigating intra-application mechanisms in microservice architecture investigates intra-application integration challenges in microservice architecture investigating intra-application integrations in microservice architecture investigating intra-application integration requirements in microservice architecture reporting application implementation using microservice design and development presenting specific application implementation paper that are using microservice-based architecture and focuses on designing the application
Exclusion criteria (EC)	not written in English focuses on process integration focuses on inter-application integration focuses on integration in other than microservice context duplicate papers research plans, roadmaps, vision papers out of topic and using the terms for other purposes non peer-reviewed papers focuses on service oriented architecture (SOA) systematic literature review

Table 9: Inclusion criteria (IC) and exclusion criteria (EC).

Overall, 386 unique results came from database searches. Search was conducted to four different scientific databases. In addition to that, backward search resulted in 1076 results and forward search in 283 results. Out of all those searches combined 1745 unique search results were found and 63 were selected for this work.

Table 10 show that Scopus provided a great number of results compared to the other databases, but since Scopus was the first database where the search was done, duplicate values that were found from Scopus are removed from other database results. Backward snowballing (Table 11) provided 10 selected results but quality-wise one of the key studies was found from there. Forward snowballing (Table 12) did not provide a significant advantage in this study by providing only one selected study.

Database search	Number of selected results
Scopus	44
IEEE	3
Web of Science	3
ACM	3
Total	**53**

Table 10: Number of selected studies from each scientific database.

Backward snowballing	overall results	unique results	selected
Scopus	941	890	7
IEEE	60	52	2
Web of Science	65	60	0
ACM	87	74	0
Total	**1153**	**1076**	**9**

Table 11: Number of search results, unique search results, and selected articles from backward snowballing in scientific databases.

Forward snowballing	overall results	unique results	selected
Scopus	267	249	0
Web of Science	20	18	0
ACM	19	16	1
Total	**306**	**283**	**1**

Table 12: Number of search results, unique search results, and selected articles from forward snowballing in scientific databases.

6.5 Assessing quality

In this study, publications are not categorized based on quality or excluded based on similarity or lack of it. This is especially because of the lack of publications and data on the subject. Many of the selected articles are case studies or experiments that consider specific case, and amount of data that it contributed to this study is limited and not generalizable. When it comes to methodology and methodological biases, they can be considered minimal. No studies were excluded because of methodological reasons.

6.6 Extracting data

Now that we have literature that satisfies the set criteria and quality requirements, it is time to read the whole text on those selected publications and begin the data collection process. Coding of data extracting is based on research questions and has been iterated according to the research question changes. Data extraction form is used to extract data and facilitate the management of that data [Aksakalli et al., 2021]. Original data extraction coding included questions:

- Goal of the paper
- Method of the paper
- Resource type
- Type of microservice architecture
- Integration mechanism
- Identified challenges/constraints/problems
- How these challenges effect to the overall application
- Solutions to identified challenges
- Integration requirements for microservice integrations
- Deployment method
- Requirements for microservice application

Since research questions were adjusted, so was data extraction coding. Based on the first round of data extraction, we noticed that there are some questions that could not be answered based on the source material. This later on lead to the iteration of research questions as well.

Goal of the paper	Method of the paper	Resource type	Architectural layer	Identified challenges	Solutions to identified challenges	Requirement for microservice

Table 13: Data extraction coding.

Requirements required additional synthesizing during the process. They were extracted from the literature to the data extraction table (Table 13), simplified, and compared with each other. If synonyms were found, they were changed to use the same adjective. After synthesizing the synonyms, frequency evaluation was performed to each requirement (Figures 6 and 7). After having the knowledge on the found requirements, they were categorized to gain better readability and understanding on the areas they are highlighting. Categories were selected based on requirements found in data. More on the categories later on in Section 7.2.

6.7 Analysing and synthesizing data

First, few things about the overall data. Database searches and backward and forward snowballing in combined resulted in 2114 publications. After filtering out the duplicates, total number of results was 1668. Out of those, based on inclusion/exclusion criteria (presented earlier in this study) either from title and abstract or also based on content, 67 publications were selected. Those 67 articles were read in full and used to answer the data extraction questions. Based on extracted data, four more articles were excluded from the selected ones. Data presented in this study is from those 63 selected publications. Out of those 63 publications 37 were conference paper, 19 articles, and rest of them book chapters (Figure 3).

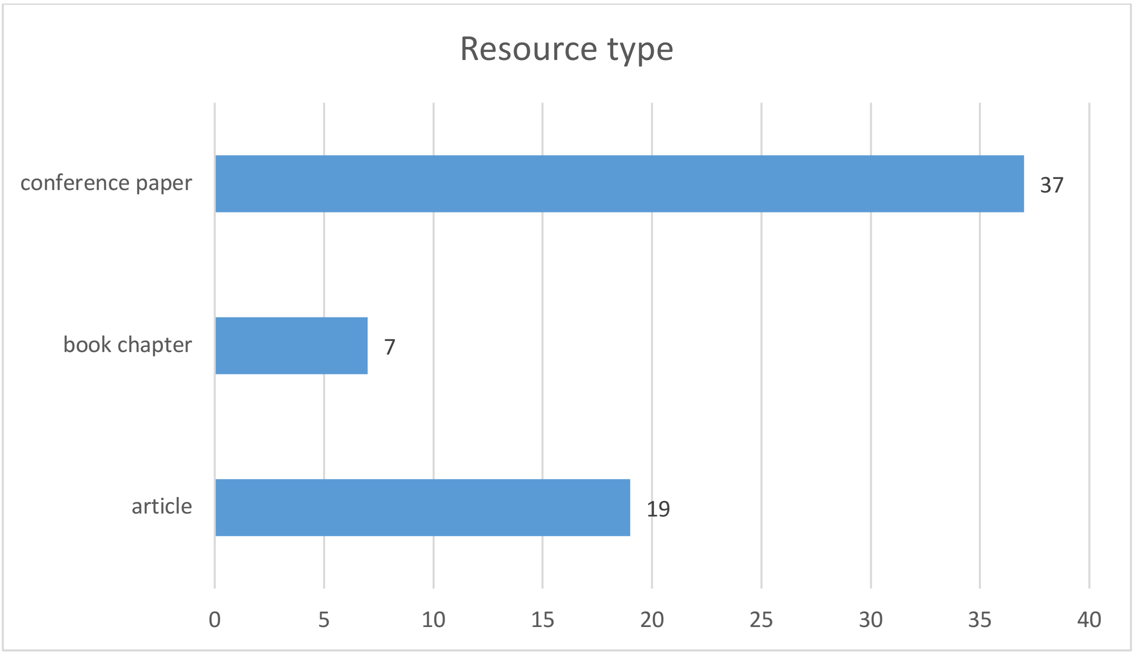

Figure 3: Division of resource types of selected publications.

Some of these publications could not be categorized unambiguously by method and 6 of them were added into two categories. Most of those two category cases were a combination of experiment and case study. Out of the 63 publications, based on method used in the study, there were 37 case studies and 21 experiments (Figure 4). Like often in the software development industry, it is hard to find two completely identical projects. Each project have some differentiating factor such as business context, technology choices, skill level of the developer etc. In addition to case studies and experiments, there were five surveys, four interviews, one observation, and one comparison study which is clear minority.

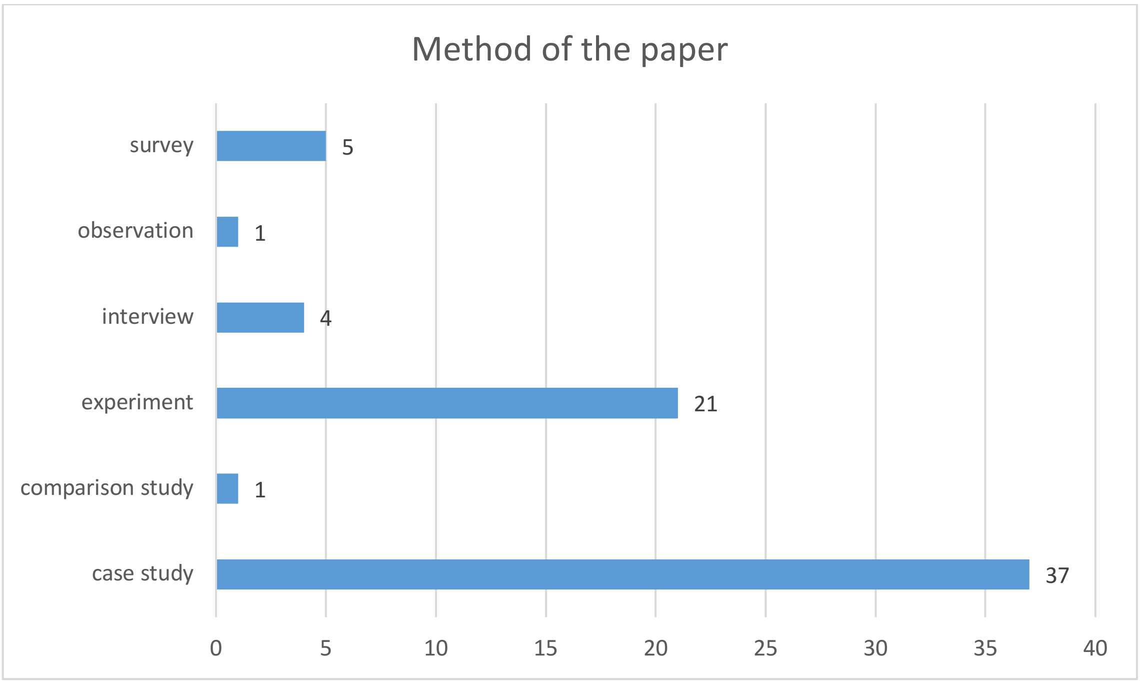

Figure 4: Method of the paper.

An architectural layer that each study covered was also documented. Data (Figure 5) shows that focus is mainly on microservice (44) and communication layers (50). Application layer is studied in 28 papers and hardware layer in 25. Layers are defined based on Fowlers' article [2016].

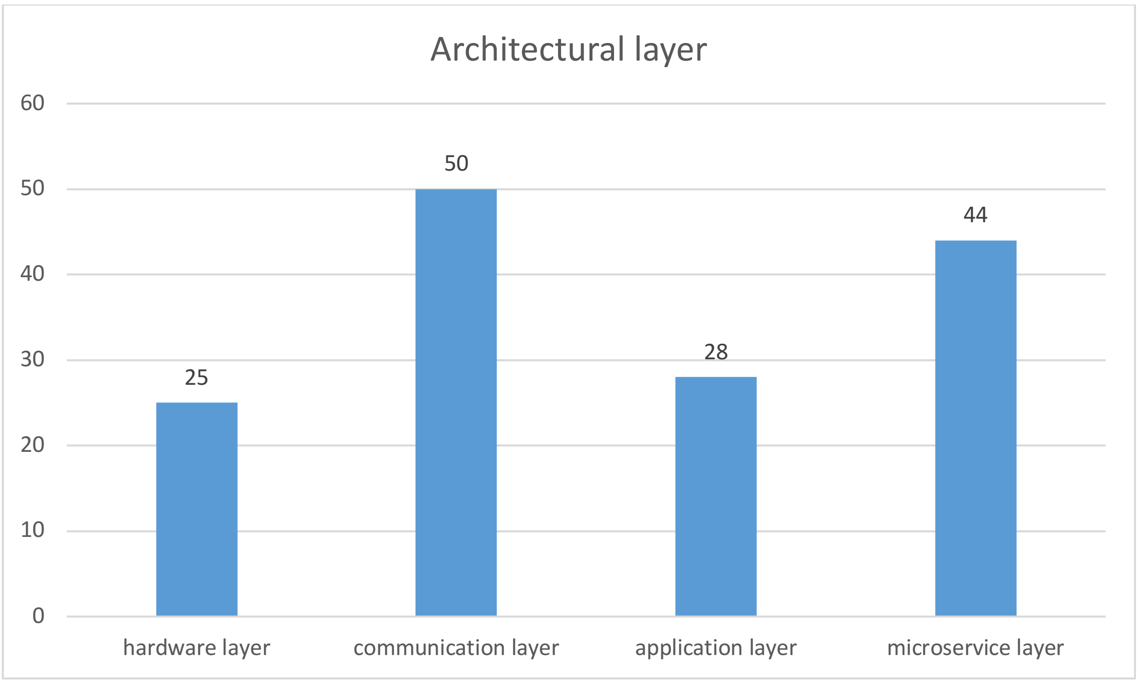

Figure 5: An architectural layer.

6.8 Reporting the findings

Results of this study are presented in Chapter 7. The chapter covers statistic from the scientific database search and presents selected articles, combines requirements for microservice architecture, and presents challenges and possible solutions from the literature.

7 Results

Results conducted from the searches to scientific databases and from the selected studies are gathered and presented in this chapter.

7.1 Search results

Overall, 386 unique results came from database searches. Search was conducted to four different scientific databases. In addition to that, backward search resulted in 1076 results and forward search in 283 results. Out of all those searches combined 1745 unique search results were found and 63 were selected for this work.

Selected studies (appendix, Source literature) for the data in this study is searched from established scientific databases. These high quality publications consist of conference paper, articles, books, and book chapters. Data sources (in this case selected studies) are suitable for the situation and results measured as objectively as possible. When possible, quantitative research was conducted but mainly subject of the study and amount of existing knowledge on the matter made more since to focus on getting qualitative results. Regarding microservice architecture in the selected studies, implementations in different studies were not comparable and most of the studies did not describe the architecture in enough detail. When unsure about the inclusion/exclusion of certain publication, the Thesis instructor was consulted.

7.2 Requirements identified

Requirements results that were got from data extraction acts as a base for identifying and categorizing possible requirements. After original data extraction round, data was refined for couple of rounds and that resulted in collection of requirements options. Based on those requirements, few categories were identified and named to describe the attributes it holds; flexibility, independency, efficiency, stability, maintainability, communication, and security. The ones that did not form a category were added to the Others. Categories were created by finding similarities between requirements found in the literature and did not follow any pre-defined classification.

Flexibility category highlights on of the microservices key characteristics. Its' agility and ability to change. Characteristics that enable scalability, evolvability, adaptability, and features that are crucial throughout the systems life cycle.

Independency of services is typical in microservice architecture. Loose coupling, isolation, autonomy, and decentralization are all requirements that enable flexibility requirements in the system. That changes can be made in one service without having to change all the other services as well. Independency category list requirements that contribute to the service independency.

Efficiency category focuses on requirements related to performance. That information is available in real-time, one operation do not block all the others, and system is not consuming unreasonable amount of resources.

Stability category presents requirements that contribute to the reliability of the software. That it doesn’t crash and provides reliable information and functionalities.

Maintainability category focuses on requirements related to software life cycles management. Monitoring, traceability, and testability

Communication and interaction between microservices is also highly critical part of the microservice architecture. Communication category gathers requirement related to that interaction between services.

Security category, as its’ name implies, gathers security related requirements such as security, privacy, and safety under one category. Encrypted communication could also belong under communication but in this study, it is categorized under security.

Other category combines all the requirements that did not belong to any of the categories mentioned above and did not have enough similarities with other requirements to be combined into their own category. Tables 14-21 presents categorized requirements identified from the selected studies.

Flexibility

accessibility	PS25, PS37
adaptability	PS14, PS19, PS25, PS45, PS55
agility	PS16, PS40, PS48, PS60, PS62
availability	PS1, PS4, PS27, PS33, PS36, PS37, PS38, PS40, PS43, PS44, PS47, PS53, PS59
broken object avoidance	PS57
changeability	PS11, PS50
compatibility	PS11, PS51, PS59
composability	PS16, PS37, PS57
configurability	PS40, PS44
connectivity	PS38
continuous deployment	PS2, PS4, PS57, PS62
continuous integration	PS31
distributed communication mechanism	PS18, PS23, PS26
dynamic development	PS57
easy implementation	PS14, PS20
evolvability	PS2, PS7, PS51, PS53, PS60
extensibility	PS23, PS25, PS32, PS34, PS38, PS51, PS56
fast delivery	PS22, PS28
fast deployment	PS21, PS29
fast integration	PS27, SP31, PS35, PS44
flexibility	PS3, PS9, PS14, PS20, PS23, PS25, PS29, PS35, PS37, PS38, PS39, PS43, PS48, PS49, PS53, PS55, PS60
global information sharing	PS15
integrability	PS2, PS10, PS22, PS23
invokability	PS37
mobility	PS38
multicomponent integration	PS3
portability	PS48
reconfigurability	PS24
refactorability	PS45, PS51, PS61
relocatable	PS61
scalability	PS4, PS8, PS9, PS11, PS14, PS20, PS21, PS22, PS23, PS25, PS26, PS27, PS28, PS29, PS32, PS33, PS34, PS35, PS37, PS38, PS40, PS43, PS44, PS46, PS48, PS49, PS50, PS53, PS54, PS56, PS57, PS59, PS60, PS61, PS62
Software Defined Network (SDN)	PS42

Table 14: Flexibility related requirement identified.

Independency

autonomy	PS28, PS55, PS59, PS60
container management	PS42
containerized integration environment	PS3
composability	PS16, PS37, PS58
decentralization	PS3, PS9, PS11, PS40, PS55, PS57, PS60
distributed architecture	PS18, PS20
independent deployability	PS9, PS24, PS56, PS63
independent development	PS24
independent scalability	PS45, PS63
isolation	PS10, PS11, PS14, PS54, PS58, PS60, PS63
loose coupling	PS9, PS18, PS19, PS26, PS28, PS35, PS37, PS49, PS50, PS51, PS53, PS57, PS58, PS59, PS61, PS62, PS63
modularity	PS9, PS11, PS14, PS23, PS45, PS53, PS57
platform-independency	PS37
self-contained	PS37
single task oriented services	PS57
technological independency	PS23, PS24, PS39

Table 15: Independency related requirements identified.

Efficiency

concurrency	PS40
continuous optimization	PS13, PS18
efficiency	PS14, PS18, PS37, PS38, PS48
load balancing	PS5, PS6, PS33, PS42, PS47, PS57
performance	PS2, PS8, PS11, PS13, PS14, PS17, PS18, PS20, PS27, PS28, PS35, PS37, PS38, PS46, PS47, PS53, PS56, PS58
real-time processing	PS11, PS14, PS15, PS21, PS28, PS34, PS38
remote capabilities	PS50
remote intelligent operation	PS15
resource efficiency	PS2, PS4, PS12, PS13, PS17, PS34, PS47, PS49, PS58, PS62
response time	PS4, PS47, PS58
reusability	PS43, PS46

Table 16: Efficiency related requirements identified.

Stability

cohesion	PS9, PS19, PS53, PS58
data accuracy	PS1
end-to-end connectivity	PS5
fault tolerance	PS6, PS14, PS27, PS33, PS34, PS36, PS38, PS39, PS44, PS58, PS61
Quality of Services (QoS)	PS4, PS16, PS30, PS33, PS37, PS47
reliability	PS1, PS13, PS17, PS19, PS23, PS33, PS36, PS38, PS39, PS40, PS45, PS47, PS48, PS53, PS58
resiliency	PS23, PS29, PS58
robustness	PS14, PS23, PS25, PS27, PS37
semantic coherence of the data flow	PS16
stability	PS6, PS51
statelessness	PS59

Table 17: Stability related requirements identified.

Maintainability

accountability	PS55
anomality detection	PS33
auditability	PS55
facilitability	PS16
fast maintenance	PS14
maintainability	PS1, PS24, PS25, PS26, PS44, PS46, PS49, PS51, PS53, PS59, PS60, PS62
manageability	PS1, PS35, PS42, PS44, PS49
monitorability	PS2, PS37, PS42
observability	PS44
performance monitoring	PS18
real-time monitoring	PS18
sensing/actuation capabilities	PS47
testability	PS53
traceability	PS55
transparency	PS11

Table 18: Maintainability related requirements identified.

Communication

discoverability	PS19, PS37, PS39
interoperability	PS20, PS26, PS27, PS29, PS37, PS46, PS56, PS57, PS59
lightweight communication mechanism	PS10, PS60, PS62

Table 19: Communication related requirements identified.

Security

encrypted communication	PS5
privacy	PS4, PS5, PS17, PS30, PS37, PS39, PS47, PS56
safety	PS38
security	PS1, PS5, PS20, PS23, PS26, PS30, PS33, PS37, PS38, PS39, PS43, PS44, PS46, PS47, PS53, PS56, PS58

Table 20: Security related requirements identified.

Others

context awareness	PS33, PS47
cost efficiency	PS1, PS4, PS13, PS21, PS22, PS23, PS31, PS38, PS44, PS55
heterogeneity	PS23, PS26, PS39, PS40, PS46
proximity	PS33, PS47
simplicity	PS8, PS16, PS23, PS24, PS40, PS57, PS62
small services	PS10
usability	PS8
versatility	PS40
virtualization	PS57

Table 21: Uncategorized requirements identified.

When analyzing the occurrence of each requirement (Figures 6 and 7), scalability (38) is without a doubt most often mentioned requirement in the selected studies. Efficiency (16), flexibility (17), loose coupling (17), performance (19), and security (17) also gained lot of attention. Following that, availability (13), fault tolerance (11), interoperability (10), maintainability (13), reliability (15), and resource efficiency (10) appeared relatively often. Having a large variety of different kinds of studies and context, the things that research highlights also varies a lot. All in all, 101 requirements were extracted from the studies. Many of those requirements had only one occurrence and that is why it is not justified to draw any defined conclusion from those.

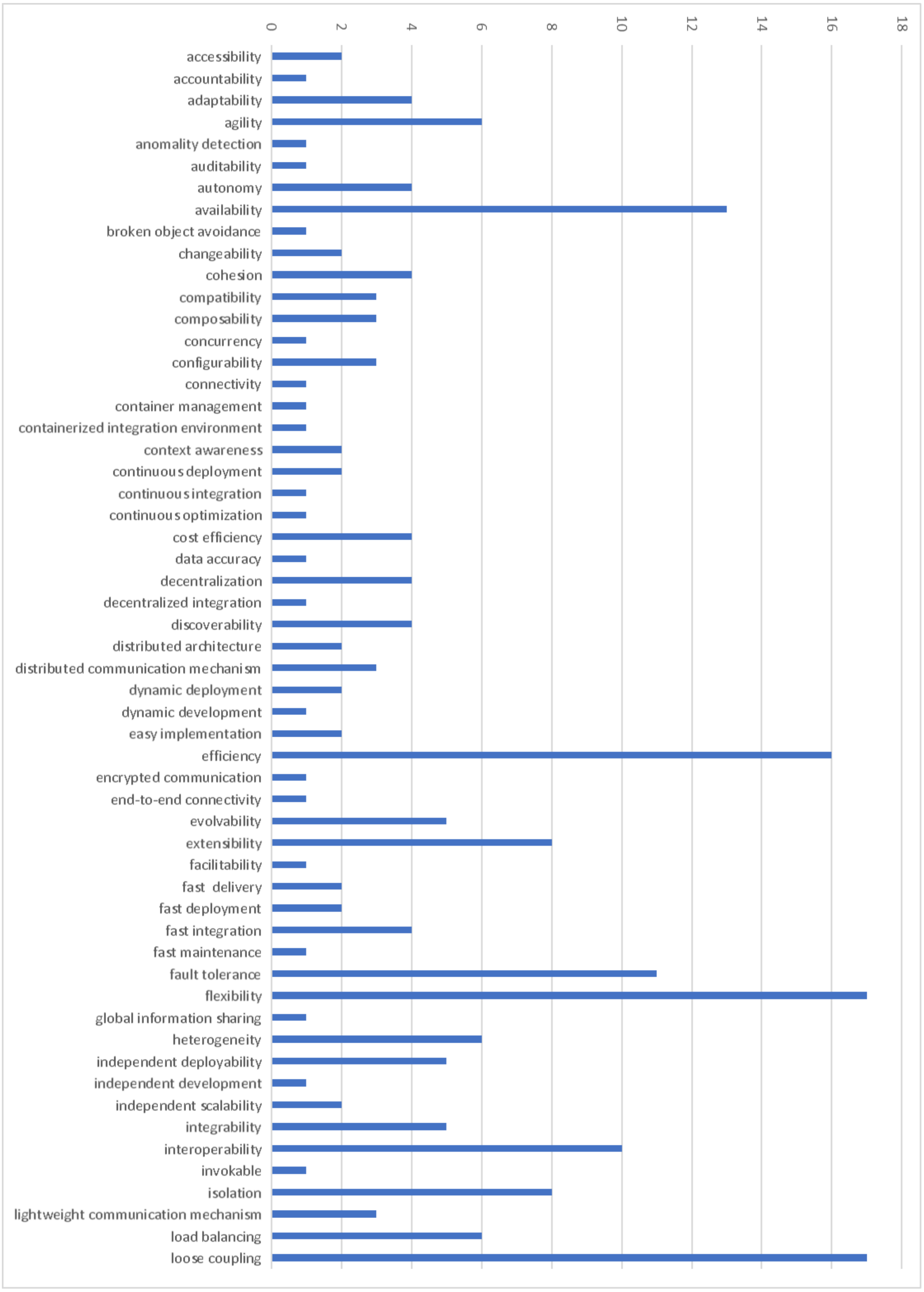

Figure 6: Requirements A-L and their frequency in the data.

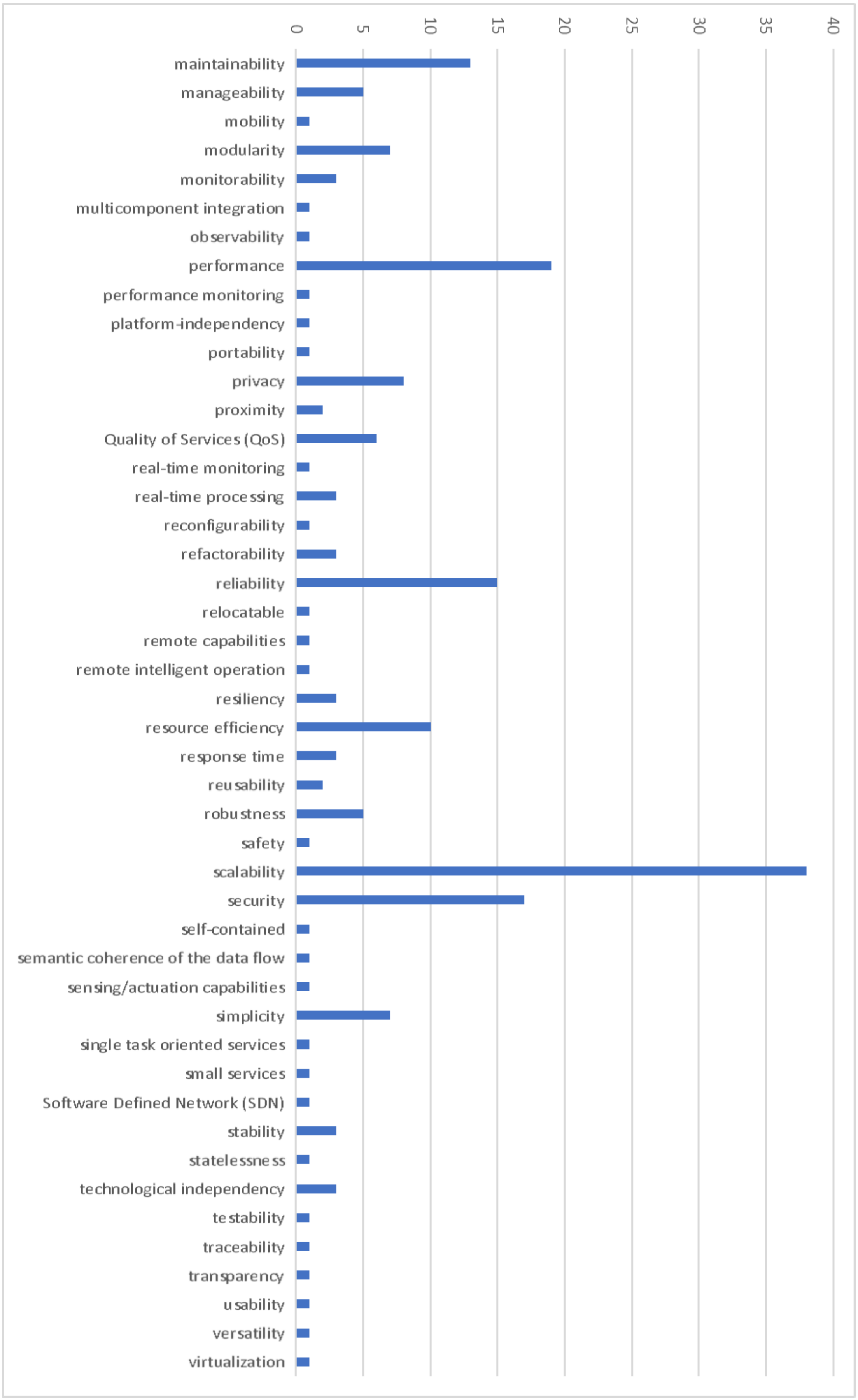

Figure 7: Requirements M-V and their frequency in the data.

7.3 Identified challenges and solutions

In this study we wanted to get a better understanding on what are the common obstacles developer face while developing systems and what solution are already recognised regarding those obstacles. Appendix Data extraction table presents challenges and possible solutions that were identified in the selected studies. Range of challenges and solutions is quite broad and combining those results into numerical format is not reasonable. Therefore, word clouds (Figures 8 and 9) were used to analyse the text formatted data extracted from the selected studies and highlight the most common words appearing in it. Word cloud generator analyses the text, identifies component words from it, and presents the ones that appear most often in the text provided to it. The larger words appear more often in the text than the smaller ones. Generator includes filters automatically non-component words such as and, or, is that do not provide any informative value but appear so often that they would distort the results. In addition to automatically filtered words, words containing microservice or service were manually excluded from the data. This decision was made after analysing the context were these words appeared which was more related to the context of this study and not the actual challenges or solutions.

Word clouds are divided so that Figure 8 is formed out of the challenges data and words in Figure 9 are from the proposed solutions data. In both figures, data is by far the largest and the most often appearing one. Figure 8 focusing on challenges also highlights security, complexity, integration, distributed, performance, API, management, and communication as reoccurring words. For solutions in Figure 9, these are architecture, communication, integration, components, framework, and patterns. Since values are not exact, there's room for different interpretation.

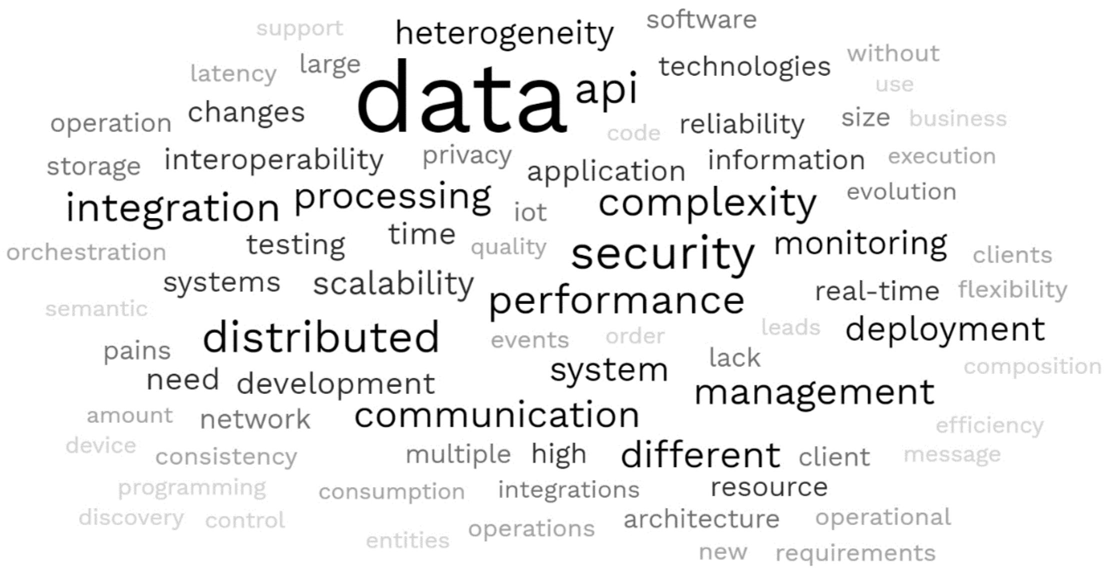

Figure 8: Challenges word cloud.

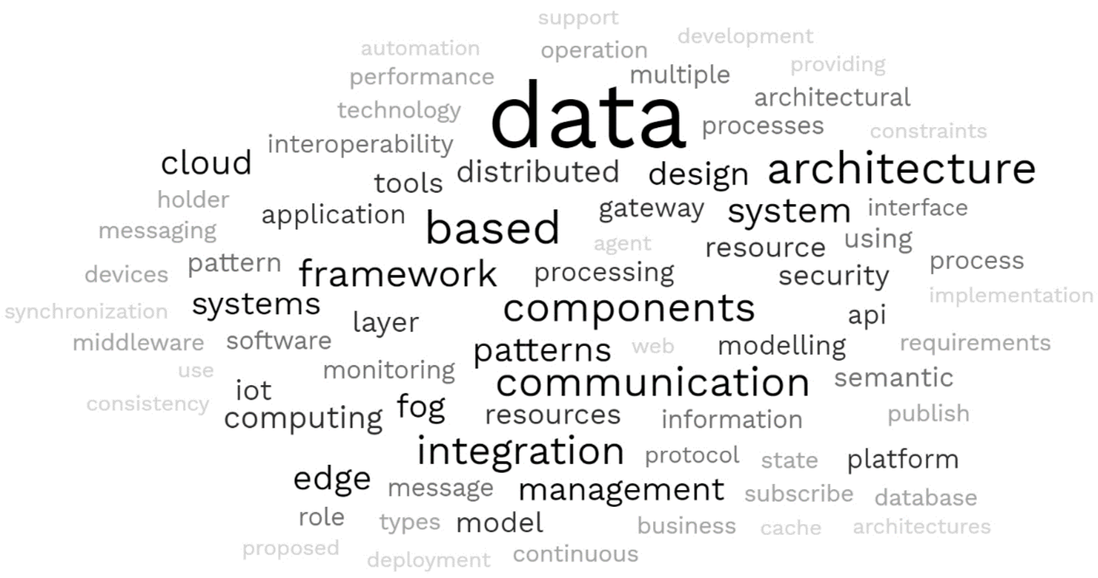

Figure 9: Solutions word cloud.

As challenging as it is to combine and categorize something out of the challenges and solutions data, here are few reoccurring themes that keep coming up throughout the answers. One of the most common reasons to adopt microservice architecture is the number of integrations needed which brings a large number of data and data processing without forgetting real-time processing, low latency, reliability, and security of the system. Solutions suggested to this are service orchestration and use of fog computing where data processing can happen in different parts of the method chain or layer. To avoid blockers that can form around e.g., a message broker, decentralizing data helps to avoid bottlenecks and improve performance. Also, use of microservice architecture together with cloud computing solutions was presented as a good combination since they both aim to provide scalability, agility, and flexibility to the system.

When considering lifecycle, systems are experiencing rapid evolution, and refactoring. The importance of lifecycle management keeps increasing. In terms of integration API management and challenges arising from it are crucial. Ways presented in the literature to solve these possible evolution challenges are use of patterns, traceability models, and versioning strategies. Use of patterns was a reoccurring theme in multiple provided solutions. The use of design patterns, architectural patterns, and different communication patterns and protocols was highly advised. Examples of other challenges mentioned in the literature are complexity, data management, performance, security threats and enlarged attack surface, and monitoring.

7.4 Research questions

Based on result extracted from the literature and presented in Sections 7.2 and 7.3, the research questions for this study are answered.

7.4.1 RQ1: What are common challenges in microservice architecture projects?

Many challenges are reported by the literature and one explaining factor may be that many studies have a unique context that brings its' own challenges. It is also difficult to draw the line between challenges that are characteristic for microservice architecture projects and challenges that appear in the described scenario.

Common challenges identified by the literature are a large number of integrations needed and therefore a large number of data and data processing. Also as shown in Figure 8, data was highlighted as a central factor of many appearing challenges. When combined with the context, it refers to the accurate and secure handling and processing of the data, the large amount of it, data flows and referencing, and integrity, distribution, and quality of the data. Real-time processing, low latency, reliability, and security of the system are also reoccurring themes mentioned in the literature. Performance blockers e.g., a message broker are also mentioned.

As a part of agile practises, principles of continuous integration and continuous deployment highlight a need for rapid evolution and refactoring in software development. Lifecycle management as a whole is an important factor but not an easy task to succeed in. Since integrations are central in microservice architecture, API evolution were brought up as a separate thing. Other challenges mentioned often in the literature are reliability, and security, complexity, data management, performance, security threats, and monitoring.

7.4.2 RQ2: How are common challenges in microservice architecture addressed?

For challenges such as a large number of integrations needed, data and data processing, real-time processing, and low latency, literature suggested service orchestration and use of fog computing where data processing can happen in different parts of the method chain or layers. To provide performance and avoid bottleneck services such as Message broker can be, decentralized data is offered as a recommendation to solve these type of challenges. Figure 9 also shows that solutions around data are the most proposed ones.

Agile practises highlight a need for lifecycle management, rapid evolution, and refactoring in software development. With microservice architecture where integrations and communication between services are present, practical API management stands out. The use of patterns, traceability models, and versioning strategies is recommended to solve system evolution challenges. Use of patterns (design, architectural, communication) was overall a reoccurring theme in multiple provided solutions.

7.4.3 RQ3: What requirements should be set to a microservice project?

Requirements that appeared most often in the literature are scalability (38), performance (19), flexibility (17), efficiency (16), loose coupling (17), and security (17). Following that, reliability (15), availability (13), maintainability (13), fault tolerance (11), interoperability (10), and resource efficiency (10) also appeared relatively often. Even though many more were mentioned, this study presents ones mentioned above as the recommended ones based on data gathered in this study.

8 Discussion

This chapter is about discussing the result of this work, what conclusions can be drawn from them, and what is the information that could, in the light of the result, interest us, support or challenge the results or provide explanation to the results. What could be done differently regarding the topic, possible limitations, is there something more we would like to know that cannot be known based on results, and evaluation of the creditability and reliability regarding the topic and results without forgetting methodological results and limitations.

8.1 Reliability of the data

Like mentioned in Section 4.9, most of the publications were case studies and experiments with their own differentiating factors. This could have an impact on the comparability of the data since each case is a little bit different and so results can be explained by those differences. Unfortunately, this is something we cannot conclude from the study.

It would have been beneficial to get more systematic literature reviews on the topic that could have provided more theoretically stable starting point for our study and helped to understand the scope more universally. Studies did not always explicitly mention the method so there is room for human error when analysing the method of the studies used in this work.

Human error is present when doing evaluations that are not explicitly measurable. In this research, that applies to the inclusion/exclusion of the studies (although minimized by the criteria) and categorization of the requirements. Categories were created by finding similarities between requirements found in the literature and did not follow any pre-defined classification. Some requirements, such as encrypted communication, could belong under multiple categories. In those cases, it was arbitrarily decided under which category that requirement belongs to.

8.2 Keywords for the search

Identifying terms to formulate an appropriate search string is one of the challenges in the database search [Mourão et al., 2020] and one that was faced in this study. It resulted in irrelevant search results and additional effort. Term ‘microservice architecture’ appeared to be a trending word and that was used in many different contexts. Presented architecture might be resemble microservice approach, follow some practices of it, or even have its’ own vision on what counts as a microservice architecture. So even though inclusion/exclusion criteria were fulfilled, and title or abstract contained required keywords, study might still end up being out of scope. Similar happened with the term ‘integration’ since it can refer to integration in the technical or project practices. Since this study focused on technical aspects of microservices, integration of practices was not in the desired scope.

Additional improvement for search term would have been adding ‘communication’ to the search term as a synonym for ‘integration’. After reading the articles, that appeared to be a common term to use when talking about interaction between microservices. Nonoptimal selection of keywords and/or search term also effected the snowballing phase resulting in irrelevant articles among the search results. Articles that are used as a seed for snowballing largely determine the quality and relevance of the results [Mourão et al., 2020]. The database search did not result in many articles that would be deal with the exact research topic, so the snowballing results consisted largely of irrelevant topics. Luckily, snowballing also resulted in few articles that could be considered core articles in this work.

8.3 Screening for inclusion

When analysing the quality of the results, inclusion for papers found in the database search might have been too loose. Xiao & Watson [2017] instruct that at the early stages of the filtering and when in doubt, it is better to be too loose with the inclusion. It is not yet known what kind of results and relevant data will be found.

8.4 Analysing the data

On Section 6.7 Figure 5 shows us that main focus on architectural level was on communication layer and microservice layer. Could this be because granularity and communication are such critical and characteristic functionalities in microservice architecture? Also, could categorizing studies based on domain (cloud, IoT, traditional context) or excluding edge and fog computing provide us with more information and inside on the results?

Since much of the data handing and formatting is done manually, it increases the risk for errors. On the other hand, it helps to identify logic errors and edge cases such as one word containing another word and therefore conflicting the results.

Selected publications were categorized by publication type, used method, and by the architectural level that was consider in the study. The data extracted from those publications were about challenges and obstacles in microservice architecture, possible solutions to the identified challenges, and requirements for microservice architecture. While formatting the data, an error in the method identified. When counting for requirement incidence for ‘usability’ or ‘connectivity’, additional results such as ‘reusability’ and ‘end-to-end connectivity’ were included. This was quickly fixed so that incidence results are correct.

8.5 Analysing the results

Analysing and finding reoccurring challenges and solutions from the extracted data was not an easy nor unambiguous task. Challenges and solutions were identified from the data in two ways, by analysing the results manually and by using a word cloud. Even though these approaches do not provide direct answers, they can highlight certain themes and perspectives to help analyse the data.

8.5.1 Challenges in microservice architecture

Identified challenges were a large number of integrations needed that leads to a large number of data and data processing. Real-time processing, low latency, reliability, and security of the system are also reoccurring themes mentioned in the literature. Performance blockers e.g., a message broker are also mentioned.

Agile practices highlight a need for rapid evolution and refactoring in software development together with lifecycle management. Closely linked with integrations is API evolution that was brought up as a separate thing. Other challenges mentioned often in the literature were reliability, and security, complexity, data management, performance, security threats, and monitoring.

Figure 8 directs analyses towards data that seems to be central factor in challenges. Common challenges identified by the literature are a large number of integrations needed and therefore a large number of data and data processing, accurate and secure handling and processing of the data, the large amount of it, data flows and referencing, and integrity, distribution, and quality of the data. This seems reasonable considering that the amount of data gathered, processed, and stored has grown enormously over the recent years both by companies and users. Figure 8 also highlights security, complexity, integration, distributed, performance, API, management, and communication as reoccurring words which aligns with results from manual analysis.

8.5.2 Solutions for identified challenges

As mentioned in the previous subsection, data was identified as one of the central factors when considering challenges in microservice architecture. Thereby it is only logical that data is considered as a key factor when identifying solutions as well. Data acts as a base for many of the systems functionalities and without accurate quality data, many of the complex functionalities are worthless.

For challenges such as a large number of integrations needed, data and data processing, real-time processing, and low latency, literature suggested service orchestration and use of fog computing where data processing can happen in different parts of the method chain or layers. To provide performance and avoid bottleneck services such as Message broker can be, decentralized data is offered as a recommendation to solve these type of challenges.

Also need for lifecycle management, rapid evolution, and refactoring in software development is highlighted. With microservice architecture where integrations and communication between services are present, practical API management stands out. The use of patterns, traceability models, and versioning strategies is recommended to solve system evolution challenges. Use of patterns was overall mentioned in multiple provided solutions. This is also supported by the word cloud for solution (Figure 9) that brought up data, architecture, communication, integration, components, framework, and patterns as the most common keyword of proposed solutions extracted from the selected studies.

8.5.3 Requirements for microservice project

Requirements appearing most frequently in the results are scalability, performance, flexibility, efficiency, loose coupling, and security followed by reliability, availability, maintainability, fault tolerance, interoperability, and resource efficiency. These requirements are high level, non-functional requirements that cover a lot of things in practise. Nevertheless, these are important requirements that guide system towards working and future proof solution.

8.6 Comparing the results of this study to results of related work

The results of this study are compared to the results of related work introduced in Chapter 2. This to see if the results are aligned with each other and if there are conclusions or discussion that can be invoked by those possible differences.

Quality attributes (QAs) identified by Li et al. [2021] are scalability, availability, monitorability, performance, security, and testability. Compared to the requirements identified in this study, there's a clear consistency with the results since scalability (38) was by far the most often mentioned requirement appeared for microservice architecture and performance (19) being one of the most common ones as well. Monitorability got three results, security 17, and testability one.

Since there are no one set of terms for requirements, results may not contain exactly the same adjectives. This is the case with availability. Availability can be seen similar or part of accessibility, connectivity, discoverability, or integrability that are mentioned in the results.

Aksakalli et al. [2021] analysed deployment and communication approaches for microservices, possible issues related to those, and where future research might be heading regarding those two. For deployment, study mentioned serverless deployment, service instance per container, and service instance per virtual machine -principles. For communication study lists useful communication protocols. Container based solution and single task oriented services are solutions and requirements also identified in this work.

Challenges regarding deployment are identified from architectural complexity, fault tolerance, deployment coordination, distributed logs, deployment cost, container image vulnerability, required specific configurations, and Continuous Integration/Continuous Delivery (CI/CD) [Aksakalli et al., 2021]. Out of the ones mentioned, complexity, fault tolerance, monitoring, cost effectiveness, and CI/CD also appeared in the results of this study.

Razzaq [2020] mapped appearance of studies related to combination of software architecture and IoT software, on which platform, challenges and solution related to IoT software, what kind of microservice based software architecture and design patterns are available for IoT software, and why microservice architecture would work well with IoT software. Data integration and scalability were highlighted as the challenges and a list of architectural patterns and design patterns was provided. Data integration can be equated to integrability or connectivity, and scalability is mentioned as the most appeared one in our study.

Campeanu [2018] is about the amount of interest towards the usage of microservice architectures by IoT and cloud computing solutions. Study shows that the interest has increased over the years and the main source for publications are conferences. This study, however, do not relate to our work.

Söylemez et al. [2022] was the most similar one out of the related work listed in Chapter 2. The study presents service discovery, data management and consistency, testing, performance prediction, measurement, and optimization, service orchestration, integration and communication, security, tracing, logging and monitoring, and decomposition as the challenges of microservice architecture. This aligns very well with the challenges identified by our research.

As solutions, Söylemez et al. [2022] mentioned using information-centric networking (ICN), multi-agent-based framework to coordinate distributed transactions of the system, automated method of regression tests, adaptive performance simulation approach, event-driven lightweight platform for microservice orchestration, Elasticsearch to solve auto-scalability issues, access control optimization model that is based on role-based access control (RBAC), graph-based microservice analysis and testing (GMAT), and domain-driven design principles. On the solutions side, there are less similarities than with challenges but multi-agent-based framework, automated testing, event-driven lightweight platform for service orchestration, role-based access control, and domain-driven design principles are mentioned in both studies.

Guo & Wu [2021] studied smells in cloud and microservice applications and if the related work talks about the impact of those smells and future research related to them. The study identified following ones: Infrastructure and Configuration Smell (IaC) that results in increasing complexity and size of the associated code, Microservice Smell that creates a need for multi-directional communication protocol, technical implementation problems, and high-level structural problems in architecture and endpoints. Study also pointed out that automatic refactoring of Code Smells most likely results in increased resource usage. Themes that were brought up in this study align with challenges that rose from our research.

Santana et al. [2018] discussed on the adoption of microservices in the IoT application development, knowledge on the matter, and on potential future research trends. The study concludes that design is predominant phase when architecting microservices for IoT or applying data solutions in cloud and fog computing. Study also listed contributions and problems identified by each of its' source literature. Challenges and solutions match well with the ones identified by our work. Challenges mentioned are orchestration, development and deployment, self-containment, scalability, versioning, security, monitoring, reuse, development of middleware, connectivity and network overlays, scalability in maintenance, interoperability, and development, collaboration of distributed modules, heterogeneity, data management, and modelling. Mentioned solutions on the other are patterns and best practices, architecture, methodology, middleware, performance and performance metrics, platform, context-based applications, prototype platform, management services, framework, and service modelling.

Soldani et al. [2018] was also contained similar research to our work and talked about the "pains" and "gains" of microservices. Both of them were categorized by following: architecture design, security design, microservices development, storage development, testing development, management operations, monitoring operations, and resource consumption operations. All of the mentioned ones are categories that appear on the results of this study. Architectural design was talked about e.g., in the form of patterns, security and related requirements were represented in the requirements table, microservice development related attributes such as loose coupling, isolation, modularity, scalability, and many more can be found from the requirements results. Gains on the other are not in the scope of this study.

8.7 Threats to validity and limitations

Regarding the selection of source articles, source of funding was not considered in the inclusion/exclusion criteria. It is not known if that had an effect to the used data and final result.

Like all studies, this work has its' limitations. Modifications made iteratively during the process may have effect the work and its' end results. This paper is unable to proclaim that the results and findings of this study are universal and apply in all circumstances. Rather it gathers identified challenges and possible solutions presented in the literature regarding the implementation and adaptation of microservice architecture and list requirements that are recognised as relevant ones. Also, the frequency of different challenges and solutions was not exact when gathering the results. This may leave room for subjectivity when analysing the results. More generalizable results can be investigated in the future research.

9 Conclusion

Popularity and interest towards microservices and microservice architecture, has grown steadily since 2014 when the concept of microservices was first introduced [Campeanu, 2018; Soldani et al., 2018]. Microservice architecture can be utilized together with cloud computing or IoT applications, on its' own, or in the form of edge or fog computing.

The objective of this work was to gather challenges, possible solutions, and based on those, requirements related to the use of microservice architecture and therefore support the work of different stakeholders in a software project using microservice architecture, but also provide information researchers as well. Study followed systematic literature following guidelines presented by Xiao & Watson [2017] and 63 scientific articles were selected.

The study gathered multiple different challenges and possible solutions from 63 scientific publications on microservice architecture. Many of the scenarios discussed in the literature had a very specific context and drawing generalizable conclusions from those would not be right. Rather we highlight which challenges repeatedly appeared in the literature. Challenges identified by the literature are a large number of integrations needed and therefore a large number of data and data processing. Also, real-time processing, low latency, reliability, security of the system, and performance blockers e.g., a message broker are also mentioned.

Rapid evolution and refactoring, and lifecycle management are needed in software development to be able to follow agile principles, such as continuous integration and deployment. Since integrations are central in microservice architecture, API evolution were brought up as a separate thing. Other challenges mentioned are reliability, security, monitoring, complexity, data management, and performance.

For challenges such as a large number of integrations, data and data processing, real-time processing, and low latency, use of fog computing and service orchestration is recommended. Decentralized data on the other hand is suggested to solve challenges related to performance and avoiding bottleneck services.

In addition to the ones mentioned above, use of patterns was one of the most recommended approach withing the suggested solution. Agile practises highlight a need for lifecycle management, rapid evolution, and refactoring in software development and integrations and communication between services are constantly present at a microservice architecture. This emphasizes practical API management. The use of patterns, traceability models, and versioning strategies is recommended to solve system evolution challenges.

Overall, data was identified as a central factor when considering both challenges and solutions in microservice architecture. Data was mentioned in the context of accurate and secure handling and processing of the data, the large amount of it, data flows and referencing, and integrity, distribution, and quality of the data. Based on this, data can be considered as the base and enabler of all functionalities and when inaccurate, a major challenge and a blocker of functionalities.

Regarding requirements, results show that scalability, efficiency, flexibility, loose coupling, performance, and security are requirements that appear most often in the studies our research found when talking about microservice architecture and its' requirements. Following that, availability, fault tolerance, interoperability, maintainability, reliability, and resource efficiency also appeared relatively often. Based on the results, these are the requirements that should be considered in a microservice project.

Researchers and practitioners can both benefit from this study. Nevertheless, more research is needed to better understand what are the most suitable ways to implement microservices and microservice architecture, and where to use them. Future work should consider investigating how facing those challenges in different parts of the development lifecycle effects the amount of work required to solve the problem and how to optimize benefits of using microservice architecture together with cloud computing.

References

Aksakalli, I. K., Çelik, T., Can, A. B., Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. The Journal of systems and software, 180, 111014.

Aurum A., Wohlin C. (2005). Engineering and managing software requirements. Ed. Aybüke. Aurum and Claes. Wohlin. 1st ed. 2005. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. Web.

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. Advances in Service-oriented and Cloud Computing (ESOCC 2015), 567, 201-215.

Bass, L., Weber, I., and Zhu, L. (2015). DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 1st edition, 2015.

Bayliss, H. R., & Beyer, F. R. (2015). Information Retrieval for Ecological Syntheses. Research Synthesis Methods, 6(2), 136–48.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., & Khalil, M. (2007). Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. Journal of Systems and Software, 80(4), 571–83.

Campeanu, G. (2018). A mapping study on microservice architectures of Internet of Things and cloud computing. 2018 7th Mediterranean Conference on Embedded Computing (MECO), 2018, 1-4.

Cerny, T., Donahoo, M. J., & Trnka, M. (2017). Contextual understanding of microservice architecture: current and future directions. ACM SIGAPP Applied Computing Review, 17(4), 29–45.

Cronin, P., Ryan, F., & Coughlan, M. (2008). Undertaking a Literature Review: A Step-by-Step Approach. British Journal of Nursing, 17(1), 38–43.

Dixon-Woods, M., Agarwal, S., Jones, D., Young, B., & Sutton, A. (2005). Synthesising Qualitative and Quantitative Evidence: A Review of Possible Methods. Journal of Health Services Research & Policy, 10(1), 45–53.

Fink, A. (2005). Conducting Research Literature Reviews: From the Internet to Paper, 2nd ed. Thousand Oaks, CA: Sage.

Fowler, S. (2016). The Four Layers Of Microservice Architecture. *https://www.susanjfowler.com/blog/2016/12/18/the-four-layers-of-microservice-architecture*.

Gates, S. (2002). Review of Methodology of Quantitative Reviews Using Meta-analysis in Ecology. Journal of Animal Ecology, 71(4), 547–57.

Grady, Jeffrey O. (1994). System Integration. Boca Raton: CRC Press. Print.

Grady, Jeffrey O. (2014). System Requirements Analysis. 2nd ed. Boston, MA: Elsevier.

Guo, D. & Wu, H. (2021). A Review of Bad Smells in Cloud-based Applications and Microservices. 2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS), 2021, 255-259.

IEEE-STD 610.12-1990, Standard Glossary of Software Engineering Terminology, 1990, Institute of Electrical and Electronics Engineers.

Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. In EBSE Technical Report, Software Engineering Group, School of Computer Science and Mathematics, Keele University, Department of Computer Science, University of Durham.

Kratzke, N., & Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. Journal of Systems and Software, 126, 1–16.

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and Software Technology, 2021, 131, 106449.

Lewis, J., & Fowler, M. (2014). Microservices: A Definition of this New Architectural Term. MartinFowler.com, 2014. https: //www.martinfowler.com/articles/microservices.html.

Levy, Y., & Ellis, T. J. (2006). A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research. Informing Science Journal 9, 182–212.

Ludvigsen, M. S., Hall, E. O. C., Meyer, G., Fegran, L., Aagaard, H., & Uhrenfeldt, L. (2016). Using Sandelowski and Barroso’s Meta-Synthesis Method in Advancing Qualitative Evidence. Qualitative Health Research, 26(3), 320–329.

Mourão, E., Pimentel, J. F., Murta, L., Kalinowski, M., Mendes, E., Wohlin, C. (2020). On the Performance of Hybrid Search Strategies for Systematic Literature Reviews in Software Engineering. Information and software technology, 2020, 123, 106294.

Noordzij, M., Hooft, L., Dekker, F. W., Zoccali, C., & Jager, K. J. (2009). Systematic Reviews and Meta-analyses: When They Are Useful and When to Be Careful. Kidney International, 76(11), 1130–36.

O’Connor, R.V., Elger, P., & Clarke, P.M. (2017). Continuous Software Engineering - A Microservices Architecture Perspective. Journal of software : evolution and process, 2017, 29(11), 1866.

Okoli, C., & Schabram, K. (2010). A Guide to Conducting a Systematic Literature Review of Information Systems Research. Sprouts: Working Papers on Information Systems, 10(26).

Onwuegbuzie, A. J., Leech, N. L., & Collins, K. M. T. (2012). Qualitative Analysis Techniques for the Review of the Literature. Qualitative Report, 2012, 17(28), 1.

Paré, G., Trudel, M., Jaana, M., & Kitsiou, S. (2015). Synthesizing Information Systems Knowledge: A Typology of Literature Reviews. Information & Management, 52, 183–99.

Pawson, R., Greenhalgh, T., Harvey, G., & Walshe, K. (2005). Realist Review: A New Method of Systematic Review Designed for Complex Policy Interventions. Journal of Health Services Research and Policy, 70(1), 21–34.

Peters, M. D. J., Godfrey, C. M., Khalil, H., McInerney, P., Parker, D., & Soares, C. B. (2015). Guidance for Conducting Systematic Scoping Reviews. International Journal of Evidence-Based Healthcare, 13(3), 141–46.

Petticrew, M., & Roberts, H. (2006). Systematic Reviews in the Social Sciences: A Practical Guide. Oxford: Blackwell.

Pouloudi, A. & Whitley, E. A. (1997). Stakeholder identification in inter-organizational systems: Gaining insights for drug use management systems. European journal of information systems, 1997, 6(1), 1-14.

Razzaq, A. (2020). A Systematic Review on Software Architectures for IoT Systems and Future Direction to the Adoption of Microservices Architecture. SN computer science, 2020, 1(6).

Rowley, J., & Slack, F. (2004). Conducting a Literature Review. Management Research News, 27(6), 31–39.

Salminen, A. (2011). Mikä kirjallisuuskatsaus? Johdatus kirjallisuuskatsauksen tyyppeihin ja hallintotieteellisiin sovelluksiin. Teaching publication, Department of Administrative Sciences, Public Management, University of Vaasa. Retrieved from https://www.uwasa.fi/materiaali/pdf/isbn_978-952-476-349-3.pdf. Access date: Jan 27, 2022.

Sandelowski, M., Voils, C. I., & Barroso, J. (2006). Defining and Designing Mixed Research Synthesis Studies. Research in the Schools, 13(1), 29–40.

Santana, C., Alencar, B., & Prazeres, C. (2018). Microservices: A Mapping Study for Internet of Things Solutions. 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), 2018, 1-4.

Soldani, J., Tamburri, D. A., Van Den Heuvel, W-J. (2018). The pains and gains of microservices: A Systematic grey literature review. The Journal of systems and software, 2018, 146, 215-232.

Stock, W. A., Benito, J. G., & Lasa, N. B. (1996). Research Synthesis: Coding and Conjectures. Evaluation and the Health Professions, 19(1), 104–117.

Söylemez, M., Tekinerdogan, B., & Tarhan, A. K. (2022). Challenges and Solution Directions of Microservice Architectures : A Systematic Literature Review. Applied sciences, 2022, 12(11), 5507.

Templier, M., & Paré, G. (2015). A Framework for Guiding and Evaluating Literature Reviews. Communications of the Association for Information Systems, 2015, 37, 112-137.

Wolff, E. (2016). Microservices: Flexible Software Architectures. CreateSpace Independent Publishing Platform.

Xiao, Y., & Watson, M. (2017). Guidance on Conducting a Systematic Literature Review. Journal of planning education and research, 2019, 39(1), 93-112. https://journals.sagepub.com/doi/pdf/10.1177/0739456X17723971. Access date: Dec 6, 2021.

Appendix

Source literature

Paper number	Authors	Year	Title	Source title
PS1	Popović I., Radovanovic I., Vajs I., Drajic D., Gligorić N.	2022	Building low-cost sensing infrastructure for air quality monitoring in urban areas based on Fog Computing	Sensors
PS2	Cortellessa V., Di Pompeo D., Eramo R., Tucci M.	2022	A model-driven approach for continuous performance engineering in microservice-based systems	Journal of Systems and Software
PS3	Kondrashev V.A., Denisov S.A.	2021	System interface of scientific services of a digital platform for multiscale modeling	Russian Microelectronics
PS4	Štefanič P., Kochovski P., Rana O.F., Stankovski V.	2021	Quality of Service-aware matchmaking for adaptive microservice-based applications	Concurrency and Computation: Practice and Experience
PS5	Safran V., Hari D., Arioz U., Mlakar I.	2021	Persist sensing network: A multimodal sensing network architecture for collection of patient-generated health data in the clinical workflow	International Conference on Electrical, Computer, Communications and Mechatronics Engineering, ICECCME 2021
PS6	Cilic I., Zarko I.P., Kusek M.	2021	Towards service orchestration for the cloud-to-thing continuum	2021 6th International Conference on Smart and Sustainable Technologies, SpliTech 2021
PS7	Bogner J., Fritzsch J., Wagner S., Zimmermann A.	2021	Industry practices and challenges for the evolvability assurance of microservices: An interview study and systematic grey literature review	Empirical Software Engineering

PS8	Strljic M.M., Prokop E., Saueressig S., Riedel O.	2021	Resulting artifacts and application scenarios of the communication intermediate layer SFCS with a focus on usability for the automation industry	4th IEEE International Conference on Knowledge Innovation and Invention 2021, ICKII 2021
PS9	Hasselbring W., Wojcieszak M., Dustdar S.	2021	Control flow versus Data flow in distributed systems integration: Revival of flow-based programming for the industrial Internet of Things	IEEE Internet Computing
PS10	Lesniak A., Laigner R., Zhou Y.	2021	Enforcing consistency in microservice architectures through event-based constraints	DEBS 2021 - Proceedings of the 15th ACM International Conference on Distributed and Event-Based Systems
PS11	Serrano-Magaña H., González-Potes A., Ibarra-Junquera V., Balbastre P., Martínez-Castro D., Simó J.	2021	Software components for smart industry based on microservices: A case study in pH control process for the beverage industry	Electronics (Switzerland)
PS12	Sun J., Jin M., Wang Y.	2021	A Microservice-based Approach to Facilitate Multi-Services in Smart Space	Proceedings - 2021 7th International Conference on Big Data and Information Analytics, BigDIA 2021
PS13	Deng J., Li B., Wang J., Zhao Y.	2021	Microservice pre-deployment based on mobility prediction and service composition in Edge	Proceedings - 2021 IEEE International Conference on Web Services, ICWS 2021
PS14	Ibarra-Junquera V., Gonzalez-Potes A., Paredes C.M., Martinez-Castro D., Nunez-Vizcaino R.A.	2021	Component-based microservices for flexible and scalable automation of industrial bioprocesses	IEEE Access

PS15	Ji H., Sun S., Xie Y., Liu H., Jiang T.	2020	Research and application of internet of things edge autonomy technology based on Microservice in Power Pipe Gallary	Proceedings - 2020 7th International Conference on Information Science and Control Engineering, ICISCE 2020
PS16	Zouad S., Boufaida M.	2020	Using multi-agent microservices for a better dynamic composition of Semantic Web services	ACM International Conference Proceeding Series
PS17	Nakamura K., Manzoni P., Zennaro M., Cano J.-C., Calafate C.T., Cecilia J.M.	2020	Fudge: A frugal edge node for advanced IoT solutions in contexts with limited resources	FRUGALTHINGS 2020 - Proceedings of the 2020 1st Workshop on Experiences with the Design and Implementation of Frugal Smart Objects
PS18	Wang Z., Xia Y., Sun C., Cheng L.	2020	Research on microservice application performance monitoring framework and elastic scaling mode	Journal of Physics: Conference Series
PS19	Zimmermann O., Pautasso C., Lübke D., Zdun U., Stocker M.	2020	Data-Oriented Interface Responsibility Patterns: Types of information holder resources	ACM International Conference Proceeding Series
PS20	Pontarolli R.P., Bigheti J.A., Fernandes M.M., Domingues F.O., Risso S.L., Godoy E.P.	2020	Microservice orchestration for process control in Industry 4.0	2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2020 - Proceedings
PS21	Kaneko Y., Yokoyama Y., Monma N., Terashima Y., Teramoto K., Kishimoto T., Saito T.	2020	A microservice-based industrial control system architecture using cloud and MEC	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)

PS22	Grogan J., Mulready C., McDermott J., Urbanavicius M., Yilmaz M., Abgaz Y., McCarren A., MacMahon S.T., Garousi V., Elger P., Clarke P.	2020	A multivocal literature review of Function-as-a-Service (FaaS) infrastructures and implications for software developers	Communications in Computer and Information Science
PS23	Gaggero M., Busonera G., Pireddu L., Zanetti G.	2020	TDM Edge Gateway: A flexible microservice-based Edge Gateway architecture for heterogeneous sensors	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
PS24	Sun C.-A., Wang J., Guo J., Wang Z., Duan L.	2020	A reconfigurable microservice-based migration technique for IoT systems	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
PS25	Ianculescu M., Alexandru A., Neagu G., Pop F.	2019	Microservice-based approach to enforce an IoHT oriented architecture	2019 7th E-Health and Bioengineering Conference, EHB 2019
PS26	Benayache A., Bilami A., Barkat S., Lorenz P., Taleb H.	2019	MsM: A microservice middleware for smart WSN-based IoT application	Journal of Network and Computer Applications

PS27	Emami Khoonsari P., Moreno P., Bergmann S., Burman J., Capuccini M., Carone M., Cascante M., De Atauri P., Foguet C., Gonzalez-Beltran A.N., Hankemeier T., Haug K., He S., Herman S., Johnson D., Kale N., Larsson A., Neumann S., Peters K., Pireddu L., Rocca-Serra P., Roger P., Rueedi R., Ruttkies C., Sadawi N., Salek R.M., Sansone S.-A., Schober D., Selivanov V., Thévenot E.A., Van Vliet M., Zanetti G., Steinbeck C., Kultima K., Spjuth O.	2019	Interoperable and scalable data analysis with microservices: Applications in metabolomics	Bioinformatics
PS28	Smid A., Wang R., Cerny T.	2019	Case Study on data communication in microservice architecture	Proceedings of the 2019 Research in Adaptive and Convergent Systems, RACS 2019
PS29	Chapman M., Curcin V.	2019	A Microservice Architecture for the Design of Computer-Interpretable Guideline Processing Tools	EUROCON 2019 - 18th International Conference on Smart Technologies
PS30	Kochovski P., Bajec M., Sakellariou R., Stankovski V.	2019	A smart and safe construction application design for fog computing	Proceedings - 2019 IEEE World Congress on Services, SERVICES 2019

PS31	Guo S., Xu C., Chen S., Xue X., Feng Z., Chen S.	2019	Crossover service fusion approach based on microservice architecture	Proceedings - 2019 IEEE International Conference on Web Services, ICWS 2019 - Part of the 2019 IEEE World Congress on Services
PS32	Zhou J., Li L., Zhou N.	2019	Research and application of battery production data management system based on microservice	ICEIEC 2019 - Proceedings of 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication
PS33	Buzachis A., Galletta A., Celesti A., Carnevale L., Villari M.	2019	Towards Osmotic computing: A blue-green strategy for the fast re-deployment of microservices	Proceedings - IEEE Symposium on Computers and Communications
PS34	Henning S., Hasselbring W., Mobius A.	2019	A scalable architecture for power consumption monitoring in industrial production environments	Proceedings - 2019 IEEE International Conference on Fog Computing, ICFC 2019
PS35	Kwon J., Kim N.L., Kang M., Wonkim J.	2019	Design and prototyping of container-enabled cluster for high performance data analytics	International Conference on Information Networking
PS36	Aquino G., Queiroz R., Merrett G., Al-Hashimi B.	2019	The circuit breaker pattern targeted to future IoT applications	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
PS37	Ahmed A.I.A., Gani A., Hamid S.H.A., Abdelmaboud A., Syed H.J., Habeeb Mohamed R.A.A., Ali I.	2019	Service management for IoT: Requirements, taxonomy, recent advances and open research challenges	IEEE Access

PS38	Dobaj J., Krisper M., Iber J., Kreiner C.	2018	A microservice architecture for the industrial Internet-of-Things	ACM International Conference Proceeding Series
PS39	Schmidt M., Obermaisser R.	2018	Adaptive and technology-independent architecture for fault-tolerant distributed AAL solutions	Computers in Biology and Medicine
PS40	Munari S., Valle S., Vardanega T.	2018	Microservice-based agile architectures: An opportunity for specialized niche technologies	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
PS41	Petrasch R.	2017	Model-based engineering for microservice architectures using Enterprise integration patterns for inter-service communication	Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE 2017
PS42	Peinl R., Holzschuher F., Pfitzer F.	2016	Docker cluster management for the cloud - Survey results and own solution	Journal of Grid Computing
PS43	Tchoubraev D., Wiczynski D.	2015	Swiss TSO integrated operational planning, optimization and ancillary services system	2015 IEEE Eindhoven PowerTech, PowerTech 2015
PS44	Batista, C., Proen, B., Cavalcante, E. Batista, T., Morais, F. & Medeiros, H.	2022	Towards a multi-tenant microservice architecture: An industrial experience	2022 IEEE 46TH ANNUAL COMPUTERS, SOFTWARE, AND APPLICATIONS CONFERENCE (COMPSAC 2022)
PS45	Ma, M., Lin, W., Pan, D. & Wang, P.	2022	Self-adaptive root cause diagnosis for large-scale microservice architecture	IEEE transactions on services computing
PS46	Nisansala, S., Chandrasiri, G. L.,	2022	Microservice based edge computing architecture for Internet of Things	2022 2nd International Conference on Advanced Research

	Prasadika, S. & Jay-asinghe, U.			in Computing (ICARC)
PS47	Villari, M., Fazio, M., Dustdar, S., Rana, O., Chen, L. & Ranjan, R.	2017	Software defined membrane: Policy-driven edge and Internet of Things security	IEEE cloud computing
PS48	Yang, HB., Ong, SK., Nee, AYC., Jiang, GD. & Mei, XS.	2022	Microservices-based cloud-edge collaborative condition monitoring platform for smart manufacturing systems	International journal of production research
PS49	Zeng, RQ., Zhao, Y., Su, H. & Guo, XY.	2020	A novel construction technology of enterprise business deployment architecture based on containerized microservices	2020 5TH International Conference On Communication, Image, And Signal Processing (CCISP 2020)
PS50	Zimmermann, O., Lübke, D., Zdun U., Pautasso, C. & Stocker, M.	2020	Interface responsibility patterns: Processing resources and operation responsibilities	ACM International Conference Proceeding Series
PS51	Lübke, D., Zimmermann, O., Pautasso, C., Zdun, U. & Stocker, M.	2019	Interface evolution patterns: Balancing compatibility and extensibility across service life cycles	ACM International Conference Proceeding Series
PS52	Tsoutsa, P. Fitsilis, P. & Ragos, O.	2017	Role modeling of IoT services in industry domains	ACM International Conference Proceeding Series
PS53	Dragoni N., Giallorenzo S., Lafuente A.L., Mazzara M., Montesi F., Mustafin R., Safina L.	2017	Microservices: Yesterday, today, and tomorrow	Present and Ulterior Software Engineering
PS54	Namiot D., Sneps-Sneppe M.	2014	On micro-services architecture	International Journal of Open Information Technologies
PS55	Hassan S., Bahsoon R.	2016	Microservices and their design trade-offs: A self-adaptive roadmap	2016 IEEE International Conference on

				Services Computing (SCC)
PS56	Vresk T., Cavrak I.	2016	Architecture of an interoperable IoT platform based on microservices	2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)
PS57	Jarwar M.A., Ali S., Kibria M.G., Kumar S., Chong I.	2017	Exploiting interoperable microservices in web objects enabled Internet of Things	2017 Ninth International Conference On Ubiquitous And Future Networks (ICUFN 2017)
PS58	Ghofrani J., Lübke D.	2018	Challenges of microservices architecture: A survey on the state of the practice	CEUR Workshop Proceedings
PS59	Pautasso C., Zimmermann O., Amundsen M., Lewis J., Josuttis N.	2017	Microservices in practice, part 2: Service integration and sustainability	IEEE software
PS60	Baškarada S., Nguyen V., Koronios A.	2020	Architecting microservices: Practical opportunities and challenges	The Journal of computer information systems
PS61	Kalske M., Mäkitalo N., Mikkonen T.	2017	Challenges when moving from monolith to microservice architecture	Current Trends in Web Engineering
PS62	Esposito, Christian and Castiglione, Aniello and Choo, Kim-Kwang Raymond	2016	Challenges in delivering software in the cloud as microservices	IEEE cloud computing
PS63	Singjai A., Zdun U.	2022	Conformance assessment of architectural design decisions on API endpoint designs derived from domain models	2022 IEEE 19TH International Conference On Software Architecture Companion (ICSA-C 2022)

Data extraction table

Source ID	Identified challenges, constraints, or problems	Solutions to the identified challenges
PS1	integrating large number of sensors, information processing, data communication, security, availability, reliability, serviceability, openness, manageability	fog computing
PS2	rapid evolution, continuous deployment, performance, software complexity, efficient integration, power consumption, memory footprint, monitoring and measuring system execution and performance	Model-Driven Engineering (MDE), logging, monitoring, more efficient integration, use of traceability models
PS3	point-to-point integrations, integration bus integrations, modern information services operating with the paradigm of microservices	Organization of the interaction of services registered on the platform, synchronization of the processes of providing services, interaction interface based on the approaches of flexible integration, cloud technologies, and virtualization. Synchronization of the processes of providing services, ensuring the transfer of data between services, and obtaining the final result are also ensured through the implementation of control processes of the digital platform.
PS4	varying and unpredictable data generation rates, performance bottleneck for data processing, latency, computational overhead, Quality of Service (QoS) aware adaptation	P-Match algorithm (i) enabling resource matchmaking of multiple application components on fog and cloud infrastructure; (ii) retrieving the results in less execution time; (iii) requiring fewer iterations to converge, and (iv) choice of optimal deployment options based on QoS constraints.
PS5	privacy, integration, data processing, data sinks, inflexibility, interoperability	HTTPS REST protocol (for synchronous connections), MQTT protocol (for asynchronous connections), JWT token (security), Microservice architecture
PS6	large amount of heterogeneous device integrations, large amount of generated data, real-time processing, latency, reliability, security	fog computing, service orchestration
PS7	service cutting, no system-centric view, mastering technologies, technological heterogeneity, missing/outdated documentation, inter-service dependencies/ripples, architectural/technical complexity, unhealthy metric usage, in-	balance between decentralization and standardization, guidelines to ensure a base consistency for evolvability such as architectural principles, specialized test automation, source code quality with tools and metrics, architectural or service-oriented tools and metrics, patterns

	tegrating legacy code, inadequate testing, communicating the importance of assurance, microservice integration, slow or manual deployment process, tool selection, breaking API changes, code duplication, distributed code repositories, high issues resolve time, slow adding of new functionality, suitable service granularity without harmful dependencies, ripple effects, chatty inter-service communication to fulfil basic operations	such as Event-driven Messaging, API Gateway, Consumer-Driven Contracts, Service Registry, CQRS, Event Sourcing, Backends for Frontends, Strangler, Service Facade, and Service Mesh
PS8	scalability, simplified deployment, broader programming platform support, simplified development, usability	quality assurance process, distributed, deployment management system, IDE integration for configuration and monitoring
PS9	interactions and flows among microservices, the order of operations executed, data flow and control flow, independent scalability, central control via orchestration and independent evolution of microservices	runtime monitoring, having data flow-oriented system design and data-flow oriented modelling and flow-based programming having control-flow as a secondary
PS10	expressing event-based constraints, such as causal consistency and event processing order. 1. Microservice architecture rely on message brokers pushing events to other microservices to use capture/subscribe communication. Events may be delayed, duplicated, or lost. This is a challenge when it comes to distinct events that present a causal dependency. 2. When two distinct events, with no explicit dependencies, arrive at the same time in a microservice and end up being processed concurrently. If those interleaving leads to opposite outcomes in the application state, the order on which the two events are processed may impact on application safety.	Pursuing a model where data is decentralized, each microservice encapsulates its own private state and exposes such internal state via well-defined application programming interfaces (APIs). Microservices communicate and exchange data via message passing mechanisms, such as asynchronous event-based communication. 1. Causal Constraints + Terminal Constraints 2. Window Constraints + System Design
PS11	high costs associated with automation and production processes, coordination between distributed industrial devices, gap between generic architectures and physical realizations, flexibility, scalability, agility, robust framework to cope with disruptions and to be able to react more quickly to continuous market changes, operational efficiency, meet the demand for growth	processing new data subsequently at different levels of the hierarchy of automation processes, development and implementation of processing plants (high costs) plug-and-play software components (coordination between distributed industrial devices) container technologies, the concept of microservices, the decoupling of each microservice with a middleware based

		on the publish/subscribe pattern (gap between generic architectures and physical realizations) solution based on software components, container technology, microservices and the publish/subscribe paradigm (flexible, scalable, and robust framework)"
PS12	different resource requirements and goals between services, the resource contentions and variable service requirements while promoting the multiple service processes, to provide an intelligent, autonomous and convenient environment for collaborating device resources and services, to validate the feasibility and effectiveness of the framework in terms of function and performance	a model and operating mechanism based on user behaviours and scene resources whereby a corresponding microservice-based scheduling architecture is designed, a prototype system to validate the feasibility and effectiveness of the framework, a model/framework for 1. data entity and data storage and microservice layers (resource management service components, process management service components, log service components, resource scheduling service components, process evolution service components) 2. communication methods and protocols 3. API gateway and access terminal (unified access to all API calls, forwarding all client requests to the back-end server, support the cache storage) 4. workflow (service process and effective device resources)
PS13	resource constraint, coverage constraint, user-mobility, service-composition, microservice-selection, success rate, cost-effectiveness, multi-optimization-objective, multi-constraints	predict future trajectories through mobility prediction, multiple edge servers, consider coverage constraint and resources constraint of edge servers when pre-deploying microservices
PS14	ability to react quickly and constantly to market changes, ability to offer more specialized, customized products with high operational efficiency, demand products to be customizable, modular, flexible, and scalable without losing robustness	framework based on software components, container technology, microservice concepts, and the publish/subscribe paradigm 1. decentralized decisions 2) interoperability 3) technical assistance (humans support) 4) information transparency
PS15	insufficient perception, low level of intelligence, no information sharing, no end-to-end online interaction are exposed in the original technical architecture, processing capacity is limited and the business connection between the main station and substation is isolated, multiple systems cannot be interconnected and information cannot be shared	change of enabling format to achieve communication interoperability, model interoperability, and business interoperability. MQTT (Message Queuing Telemetry Transport) (a lightweight communication protocol based on publish/subscribe mode, which provides real-time and reliable message services for connecting remote devices)

PS16	interoperability, service composition, discovery and composition of services requires human intervention	Enterprise modelling, Agent-based architectures, Microservice architecture, Ontologies. We define three types of agents to deliver composite semantic web services: • The interface agent (Microservice 1): Its role is to semantically rewrite the user request based on ontologies of business processes. • The selector agent (Microservice 2): Its role is to assign a task (semantic description of the composition process) to the composer agent based on the ontologies of the business processes. • The composer agent (Microservice 3): its role is to dynamically compose and coordinate Semantic Web services."
PS17	latency, reliability, power consumption, resource usage, privacy, low-latency constraints	edge/fog and microservice architecture, MQTT (publish/subscribe messaging protocol), "aggregator" (a process that coordinates the data flow between the services on the edge nodes with those in the cloud)
PS18	monitoring analysis and quality assurance of system performance	performance monitoring framework
PS19	how many services should be exposed, which service cuts let services and their clients deliver user value jointly but couple them loosely, how often do services and their clients interact to exchange data, how much and which data should be exchanged, conflicting non-functional requirements for service design, exposing API data entities so that API client can access and/or modify these entities concurrently without compromising data integrity and quality, API support to clients that want to CRUD instances of domain entities that are short-lived, change often and have many outgoing relations, CRUD operations to long-living, quite unchangeable data that is referenced by other data, referencing to data that is referenced in many places, lives long, and is immutable for clients be treated in API contracts, data exchange between communication participants that do not know each other and are not available at the same time, message representation referation to API	responsibility patterns to cover two distinct main architectural roles for API endpoints: Processing Resources and Information Holder Resources (Operational Data Holder, Master Data Holder, Reference Data Holder, Data Transfer Resource, Link Lookup Resource)

	endpoints and operations without binding the message recipient to the actual addresses of the endpoints	
PS20	promote integration between technologies, equipment and automation systems allocated in different hierarchical levels of industrial systems, optimizing the efficiency of the production chain	The Molecular framework for microservices was chosen for the development of the MOA architecture, a cloud infrastructure shared by equipment and systems
PS21	unpredictable network latency to get real-time processing, high cost	latency-cost algorithm, fog computing, Multi-access Edge Computing (MEC), having Asset management service, Demand forecast service, Trading service, Control service, and Service management service
PS22	faster software delivery with less impact on operational systems, and at a reduced hardware provisioning cost	Function-as-a-Service (FaaS), adoption of a distributed high-decoupled service-based architecture (such as microservices)
PS23	effective handling of heterogeneous and distributed data sources, data collecting infrastructure from the point of view of security, reliability, device heterogeneity	TDM Edge Gateway architecture
PS24	a large number of integrations to distributed and heterogenous components, continuously running system in a dynamic environment which frequently suffers environmental changes (e.g., traffic situations) or requirement changes (e.g. preferred paths), a large number of hardware and software modules, which are expected to be easily replaced or reconfigured, connectivity between the decomposed microservices, resulting in poor reconfigurability of the resulting system	a microservice-based migration framework for IoT systems, a supporting tool to enable and automate as much as possible of the proposed technique, decomposition principles: Doman analysis, Static analysis, Hierarchy-aware, and Embedded features-aware
PS25	amount and variety of available data, complex with distributed systems involving multiple entities that generate and consume data, adaptability, accessibility	combination of cloud computing and microservices, integration of real-time and accurate IoT data, RO-Smart Ageing architecture structured on layers: Data Acquisition Layer, Communication Layer, Edge/Fog Layer, Cloud Layer, Visualisation and Action Layer
PS26	Integrating wireless sensor networks using different middleware, with challenges such as heterogeneity and interoperability. Excessive use of IoT leads to challenges in device capabilities, user needs, and application requirements.	Microservice middleware (MsM) to allow interactions between devices without big changes in the system architecture. The proposed microservice model uses the ANN (Artificial Neural Network) concept to achieve a lightweight and intelligent microservice network.

		Use of specialized and adapted tools to permit an efficient exploitation and perfect device integration with the internet, that lead to the emergence of other problems like heterogeneity, interoperability, security, etc.
PS27	rapidly generating datasets of massive volume and complexity, large scale data analytics, need for integrations and constant need for change, scalability of the system	a Virtual Research Environment (VRE) which facilitates rapid integration of new tools and developing scalable and interoperable workflows for performing metabolomics data analysis. Cloud computing offers possibility to instantiate and configure on-demand resources such as virtual computers, networks and storage, together with operating systems and software tools. The study provides a framework for rapid and efficient integration of new tools and developing scalable, and interoperable workflows, supporting multiple workflow engines
PS28	data communication management, migrating legacy monolith applications to microservices, managing data communication from the original monolith to the new microservices and between the distinct microservices themselves, performance, implementation of features that need data in the database of a separate module, synchronizing data replicas when data changes in the master database	automated data streaming system between databases, distributed cache, balance between the performance and coupling, microservice architecture with separate databases, a good design for data communication of microservices, Architecture to improve data communication performance of microservices: 1) data synchronization between the legacy monolithic system's database and the microservices' databases. The architecture proposed here uses message queue and streaming platforms to automatically capture and synchronize database changes. 2) improving data communication performance between microservice instances by applying the cache and message broker Redis. Separated the management system from the monolith application as a microservice based on the functional distinction. A database for the microservice distinct from the monolith database. Use a data streaming platform to synchronize master data changes. multiple instances of the same server (high availability). Two architectures: 1) data synchronization between different databases of distinct microservices or between microservice and

		monolith by building a data stream system with technologies. 2) a ’semi microservice’ technique by introducing a distributed cache instead of splitting the data model into multiple bounded contexts.
PS29	support for external interoperability, increased adoption of CIGs in different parts of the system leads to an integration overhead, system redundancy, and a lack of flexibility in how these tools can be combined	a blueprint architecture to be used in the design of guideline processing tools, based on the conceptualisation of key components as RESTful microservices. Define the types of data endpoints that each component should expose, for both the communication between internal components and communication with external components that exist as a part of a DSS. Centres around three types of RESTful microservices, each with a set of well-defined endpoints
PS30	high Quality of Service (QoS) operation, achieving privacy and security, functional orchestration of various microservices during its operation	fog computing through the design of multi-tier, container-based applications
PS31	semantic inconsistencies in business and interface make crossover service fusion difficult and time-consuming, no uniform development standards	A five-part crossover service fusion framework: 1. identifies key components 2. acquires Service Fusion Requirements 3. Business Matching Between Domain Services 4. Interface Matching Between Domain Services 3.-4. these can detect and resolve semantic inconsistencies. 5. Service Fusion Implementation (to help achieve rapid integration of crossover service) focuses on detecting and resolving semantic inconsistencies on business and data during the service integration, thus achieving smooth integration."
PS32	customizability, deployment, data integration, data management, data analysis, data processing, flexibility, central development	Microservice architecture to complete the supplementary extension for battery MES. A scheme is proposed for unified data platform including application layer, service layer and data layer.
PS33	management of microservices, the IoT proliferation, the rapid development of new technologies in Cloud environments	Microservice architecture

PS34	fault tolerance, extensibility, real-time data processing, resource efficiency	Microservice architectural pattern, fog computing paradigm
PS35	Technical constraints around bottle-necked inter-connections for overlay networking and storage access. Process and analyse large-amount of data generated from diversified data-centric applications and performance.	usage of high-performance parallel file system inside the worker nodes
PS36	dependability of the integration and collaboration between IoT systems, resilience of distributed services, preventing the failure propagation to dependent services	Circuit Breaking pattern
PS37	lack of interoperability, flexibility, scalability, security, identity management, authentication, authorization, vulnerability, availability, data accuracy, real-time analytics, data visualization, heterogeneity of hardware and software, multiple network technologies with different communication protocols, different software platforms, service provisioning, service orchestration, service composition, service adaption	Service management, Web of Things (WoT), RESTful API, Middleware, requirements
PS38	deployment, scalability, integration, interoperability, mobility, performance, maintenance development, collaboration of distributed modules, heterogeneity, discovery, integration, different programming languages, continuous deployment, each service running in its own process and communicating with lightweight mechanisms, Self-Containment, Monitoring, Orchestration, Versioning Secure, reuse, Development of IoT middleware Connectivity and Network Overlays	Middleware, Platform, Context-based applications, Prototype platform, Management Service, Architecture, Framework
PS39	interoperability, interconnected and distributed automation systems, communication, dependability, scalability, performance, real time operation, predictability, data consistency, flexibility, extensibility	"Design patterns. Architecture that combines the IoT world, industrial automation systems, modern information technology (IT) and cloud architectures. Is lightweight and flexible design, along with the support of state-of-the-art development approaches (containerization, continuous integration (CI), continuous deployment (CD)) make the architecture equally suitable for the deployment on cloud, fog and edge devices."

PS40	flawless sensor integration and time synchronization, distributed application scenario management, fault-tolerance, service discovery and registration, heterogeneity of the underlying technologies, multiple integration levels, open-world assumption, communication models, data security, privacy	fault-tolerance concept, a fault-hypothesis for software, partitioning management OS (Hypervisor)
PS41	a downtime in their infrastructure regarding costs and criticality, the massive growth in code base size, distribution, deployment, liveness	architectural principles: agility, versatility, scalability, simplicity technology requirements: high-level abstractions, modern testing framework, beware dependencies, economy matters patterns: layering, pipe, filter configuration management and automation, container-based solutions
PS42	API interfaces, security, exception handling, using a message broker, calling other services synchronously or asynchronously, selection of a self-optimizing and resilient run-time environment	microservice architecture patterns (microservice chassis, database per service), communication style (messaging, domain-specific protocol, remote procedure invocation (RPI)), Enterprise Integration Patterns (EIP), message-oriented middleware (MoM), Domain modelling, bounded contexts, determination of microservice candidates, formalization and modelling of microservices, UML profiles, specified messaging for microservices, "database-per-service" pattern, event-driven architecture, publish-subscriber-channel
PS43	gateway got too much logic, different client types	SCSs, API gateway, plugin approach, UI monolith, BFF pattern, dividing into gateway into multiple gateways, each UI service or client type gets its own gateway, backend is closely linked to the corresponding frontend
PS44	A full guest OS image for each VM, high RAM and disk storage requirements, slow startup	Containers
PS45	API evolution, data values, data structure representation, data input validation and constraint, business rules, context-awareness, providing data structure separately requires meta programming or use of map data structure that leads to loose type safety and lack of consistency	providing data structure info through separate information channel, providing context awareness, context-awareness security, Data Transfer Object (DTO) and map structure, following service-expected data structure, Generative Programming (GP), design proposition (traditional base and connector), Model-Driven Development (MDD), code-inspection tool

PS46	network latency, reliability, dependability, complexity in orchestrating microservices, data consistency, transaction management, load balancing, centralized configuration management	
PS47	reliability operation, maintenance, identifying the root cause of anomaly in large-scale microservice architecture	MS-Rank (a multi-metric and self-adaptive root cause diagnosis framework for microservice architecture)
PS48	delivering real-time IoT services, concurrent and real-time application execution, platform-independent deployment	a novel three-layer architecture (ROOF, Fog, Cloud) that facilitates the service requirements. Then a novel platform and relevant modules are developed with integrated AI processing and edge computer paradigms considering issues related to scalability, heterogeneity, security, and interoperability of IoT services. Further, each component is designed to handle the control signals, data flows, microservice orchestration, and resource composition to match with the IoT application requirements.
PS49	amount of IoT devices, volume of generated data and computing/storage, security, privacy, implementing many new features without considering security, considering security heterogeneity (different types, data formats, and firmware) of devices, Device-centric attacks, Network-centric attacks	IoT applications and services deployed in an OSMOSIS system can be viewed as a graph of MicroElements (MELs), where a MEL can be composed of two types of software components: 1) Microservices, that implement specific functionalities and which can be deployed and migrated across different virtualized infrastructures, and 2) MicroData, that represents a piece of information flowing from and to IoT sensor and actuator devices, and which may occur in a variety of domain-specific data formats. The integration of security strategy to enable optimal migration of microservices and their data (MELs), between Edge and Cloud resources. MELs is based on microservice architecture, and that data processing and handling is done at the edge so that compromising a microservice would have as little effect as possible. It uses isolated networks among MELs, based on communication tunnels and can improve overall IoT application performance and reduce core network load. The SDMem is configured based on a security policy informed by an “attacker model" (evaluate security risk based on possible attack types).

PS50	computing and storage capabilities of the cloud layer, the real-time nature of the edge layer, the lack of a systematic and comprehensive microservices-based condition monitoring (CM) framework, the lack of cloud-edge collaborative mechanism, there are few microservice-based architectures involved in industrial scenarios	Cloud-edge collaborative computing that encapsulates distributed resources into manufacturing services, cloud-edge collaboration mechanism
PS51	low-coupling, scalability, module independency, data adaptation, security, monitoring, authentication, authorization, performance, configuration management, high concurrency	
PS52	role of API endpoint, responsibility of each API operation, service granularity, coupling, learnability, manageability, semantic interoperability, response time, API security, request/response data privacy, compatibility, evolvability, allowing remote client to trigger API actions, client invoking side-effect-free remote processing on the provider side to have a result be calculated from its input, API provider allowing a client to report that something new has happened that is worth capturing for later processing, information owned be retrieved to satisfy an information need of an end user or to allow further client-side processing, client initiating a processing action that causes the server-side application state to advance, API clients and API providers sharing responsibilities required to perform and control the execution of business processes and their activities	The Microservice API Patterns (MAP), endpoint role patterns (Processing Resource, Information Holder Resource), four operation responsibility patterns (Computation Function, State Creation Operation, Retrieval Operation, State Transition Operation), read-access-only operation
PS53	need for semantic approach to model services to enable automated collaboration between microservices, services to interact with other services in an autonomous way, services to easily crawl their domain to discover trusted relevant services, services to delegate their assignments to other, similar trustworthy services, services to advertise their offered services in their domain	the role modelling approach/theory (instead of having a static plan of interactions for services, we model them to play a role, where roles can have a set of behaviours defined in a way that will make them capable to react differently under the circumstances hold at the specific state)
PS54	programming distributed systems is harder than monoliths, managing changes to a service that may have side	

	effects on other services that it communicates with, preventing attacks that exploit network communications, dependability (regarding interfaces, behavioural specifications and choreographies), trust and security (greater surface attack area, network complexity, trust, heterogeneity)	
PS55	additional complexity as a distributed system, testing of distributed system, need for inter-service communication mechanism, need for distributed transactions, increasing deployment complexity, deciding how to split the system into microservices, interface changes in an individual service, Remote Procedure Calls are more expensive than in-process calls	Dividing systems based on responsibilities Single Responsible Principle (SRP) pattern, Remote Procedure Calls (RPCs) for communication, using appropriate communication patterns (Direct calls, A gateway, Message bus)
PS56	intrinsic complexity, development time, operation time, architecture, security, storage, testing, monitoring, resource consumption, management, architectural pains (size/complexity, granularity, service dimensioning, API versioning , service contracts, communication heterogeneity), security related pains (access control, endpoint proliferation, size/complexity, centralized support, CI/CD, human errors), storage pains (data consistency, distributed transactions, query complexity, heterogeneity), testing pains (performance testing, size/complexity, integration testing), management pains (operational complexity, service location, cascading failures, service coordination), monitoring pains (size/complexity, logging, problem location, resource consumption (network, compute), need for methodologies and techniques easing the dimensioning and versioning of microservices, and simplifying the execution of transactions/queries on distributed and heterogeneous data stores	architectural modelling technique MAPE-K loop (Monitoring, Analysis, Planning, Execution, Knowledge)
PS57	need for different technologies, fast implementations and fast improvement and replacement, considering the communication of billions of objects from different domains and handling the complexities of semantic cooperation among	The Web of Objects (WoO), virtualization of objects in a decentralized manner and by using semantic ontology, using concepts and patterns of microservices

	them, understanding the data and information of connected objects, supporting functionalities of every connected object, rapidly implemented in best fit and lightweight technologies and deployed independently, and less centralized management in order to rapid scalability, recovery, and resiliency	
PS58	security, response time, performance, resilience, reliability, fault tolerance, memory usage	consider security, licence, memory usage, last release date and last release when using third-party artifacts in microservice architecture
PS59	service composition, microservice lifecycle management, identifying thresholds for alerts to let developers know that something is wrong	considering following things in certain categories: communication (caching, asynchronous messaging between components, base info comes in one round trip and additional info asynchronously), End-to-end data integrity (data should have a single master, a single source of truth for that data, often third-party data, make each component validate the data it receives and returns on the basis of that component’s own local models, Domain-Driven Design (DDD), consistency is a key approach in large distributed systems, investing in consistency result in sacrificing availability), sustainable service evolution (versioning, backward compatibility, semantic interface versioning), organizational scaling strategies (organization is part of any system you design, team size, team got capability to produce end-to-end functionality), success factors (continuous delivery, ability to create infrastructure on demand, understand the nature of distributed systems, only constant is change)
PS60	distributed system complexity, decomposing of the data layer, lack of relevant development skills, reliance on Software as a Service (SaaS) and commercial off the self (COTS) applications limits on the potential uptake of microservice architecture, governance, organizational structure, master data management, orchestration, choreography, testing, performance, communication/integration, service discovery, fault tolerance, security, tracing and logging, application	continuous delivery, continuous monitoring, systems of differentiation, systems of innovation-experimental system developed, no microservice for large corporate systems of record (ERP, CRM), skilled cross-functional implementation teams, distributed coordination responsibilities

	performance monitoring, deployment operations, development/deployment agility, operational scalability, the fallacies of distributed computing	
PS61	the size of the codebase, automatic testing coverage, decomposition of services, performance overhead, integration between different microservices, logging and monitoring, managing multiple different databases	microservice architecture, having similar architecture as the organization is, not tying integration to a specific technology, having simple to use and backwards compatible interface, to have tools to automatically to deploy, scale and manage services, Circuit breaker pattern, agree on eventual consistency of data between microservice transactions
PS62	enlarged attack surface, the challenge of debugging, monitoring and auditing the application, the use of off the-shelf (OTS) components represent a security threat, trustworthiness, data sharing security threats	using microservice architecture, using Virtual Machine (VM), partitioning monolithic applications into small pieces of computation that allows for the segmentation of application data
PS63	API evolution, microservice model evolution, architecture conformance assessment	a traceable mapping, automated assessment of architectural conformance checking, modelling framework and traceability support, using ground truth and detector results