Microservices in a DevOps context

A review of "A Systematic Mapping Study on Microservices Architecture in DevOps"

Christoph Bühler
University of Applied Science of Eastern Switzerland Rapperswil OST
MSE Seminar "Design Science and Empirical Software Engineering"
Supervisor: Olaf Zimmermann
Semester: Fall 2020

Abstract

A systematic mapping study conducts a broad search for publications in a research topic and maps the results into a condensed form. It gives an overview over the topic, the problems and corresponding solutions [PFMM08, FD15]. This paper is a review of such a study on the topic of "Microservice in a DevOps context". First, the reader is introduced into the various techniques and terms used in design science and empirical software engineering, then we provide a summary of the paper and a critical review at the end. The study did an excellent job in finding results from the academia but the categorical exclusion of gray literature lead to missed solutions for given problems.

Keywords: Design Science, Empirical Software Engineering, Microservices, DevOps, Systematic Mapping Study, Review

1 Introduction

This paper is a review of the systematic mapping study "A Systematic Mapping Study on Microservices Architecture in DevOps" [WLS20]. The goal is to introduce the reader to the topic, explain the used methods in empirical software engineering and give a critical review of the conducted study and the results.

The study used a broad search over several well-known publication-databases and searched for research material on the topic of microservice architectures in the DevOps context. After the first search additional material was searched for with a technique called "snowballing". All this material was then screened and analyzed and after an initial selection was made, the studies were fully read and information extracted. The results are then mapped and categorized according to guidelines.

The study shows various problems and their corresponding solutions along with some challenges (problems without any purposed solution). Some of the problems do not have a solution because gray literature is not allowed in the search.

The remainder of the paper will introduce terms and principles that are used in the reviewed study, as well as topics that a reader needs for the understanding of the study and this review. Furthermore, this paper creates an objective summary, a critical review and a derived conclusion of the reviewed paper [WLS20].

1.1 Design Science

Design Science has the main purpose of achieving knowledge and a general understanding about a domain. Design Science contains several guidelines according to Alan R. Hevner (among other authors). Those guidelines are [HMPR04]:

- *Design as an artifact*: Design Science must produce an artifact
- *Problem relevance*: The objective is to develop solutions to relevant business problems
- *Design evaluation*: The usefulness of an artifact must be demonstrated with evaluation methods
- *Research contributions*: It must provide clear and verifiable contributions to the topic
- *Research rigor*: The research relies on rigorous methods in construction and evaluation of the model
- *Design as a search process*: The search for artifacts requires satisfying laws to be in place
- *Communication of research*: The targeted audience should be technology based as well as management based

With this guidelines, it is possible to model a domain of interest and acquire specific knowledge about it.

1.2 Empirical Software Engineering

Empirical Software Engineering (ESE) provides a base for discussion and methods for empirical research regarding software engineering topics. *Empirical* means in the context of software engineering that various results are taken into consideration. The results of the methods are proven by existing publications.

1.2.1 Systematic Mapping Study (SMS).

A SMS is a defined method to gather, analyze, classify and structure a field of interest. The analysis focuses on frequency and topics for a field. It is a defined process in which the following steps take place [PFMM08]:

1. Define research questions (RQs) and topic
2. Define search query and parameter
3. Search for articles and publications in given databases
4. Analyze and screen the results (i.e. quality assessment and data analysing)
5. Classify and map the given articles

The result of an SMS allows readers and researchers to determine the coverage of the given field of interest [PFMM08]. In the reviewed paper, the search yielded 47 publications as "relevant" for the SMS.

1.2.2 Systematic Literature Review (SLR). An SLR is a method of ESE to systematically analyze and review a given topic. It uses methods to collect secondary data and critically reviews the given research study. The search for additional data can involve published as well as unpublished work on the subject [SWH19]. SLR and SMS both enable researcher to acquire knowledge about a topic. Whereas the SMS does this on a wide scale, the SLR goes into depth of a domain of interest. Babak Farshchian and Yngve Dahl described the difference as: "An SMS uses the same basic methodology for searching and analyzing literature as in a SLR. An SMS, on the other hand, aims at creating a map of a wide research field. ... The knowledge created by an SMS can be used as the basis for further research (Kitchenham et al 2011), for instance, as a pre-study for one or several SLRs in specific areas." [FD15].

1.2.3 Systematic Gray Literature Review (SGLR). A SGLR is essentially the same as an SLR with a very important difference: It does not only consider published and unpublished *peer reviewed* work, but also "Gray Literature". Gray literature is evidence and material that is not published in commercial and peer reviewed publications [Pae17]. In the context of computer science, gray literature can provide important statements and evidence towards topics that are more driven by businesses than by the academia. Commercial companies drive the innovation around computer science nowadays. Those companies need solutions for acute problems and therefore do not wait on papers and evidence to be peer-reviewed until they move on.

1.3 Microservices and DevOps

Since the topic of the reviewed paper does conduct an SMS over "Microservices in DevOps", it is utterly important to define those terms so that any reader of this paper understands the base of the terms on which the conclusions are built upon.

1.3.1 Microservices. Microservices (sometimes referred to as "Microservice Architecture") is an application structural style. The style focuses on building several small services that cooperate together to create an application. Those services are often deployed on distributed systems. Microservices adhere to the following tenets [Zim17b]:

- *Fine-grained interfaces*: The work unit encapsulates logic around a single topic of work and exposes the interface remotely
- *Domain-Drive Design (DDD)*: The services are conceptional created around business-driven development patterns
- *IDEAL*: **I**solated State, **D**istribution, **E**lasticity, **A**utomated Management and **L**oose Coupling
- *Polyglot Persistence*: Multiple programming paradigms (e.g. object-oriented and functional) and storage paradigms (e.g. NoSQL and relational database systems) are used in a polyglot programming and persistence strategy
- *Lightweight Containers*: The work units are containerized and deployed via corresponding channels (e.g. Docker)
- *Automated Continuous Delivery*: During service development, a high degree of automation is used to deploy the work unit
- *Lean Holistic Management (DevOps)*: Largely automated practices are used to tackle configuration, performance, monitoring and fault management

These tenets define the work units of a microservice architecture. This makes a MSA a highly flexible and dynamic approach to develop software [Zim17b].

1.3.2 DevOps. The term "DevOps" is a mash-up between "Development" and "Operations" which are two strong and important terms in modern software engineering. DevOps provides a set of practices with the intend to reduce the time of a change to the code to the production environment while maintaining a high quality of the software [BWZ15]. As an example, when a change is committed to the version control system (VCS), like GitHub, the build pipeline will test and build the container automatically. Afterwards, the container is deployed to a test environment and on acknowledgement, the container is then shipped to production.

2 The reviewed mapping study

The general topic of the reviewed paper is to conduct an SMS over the topic of "Microservices in DevOps". We should take into consideration, that the context is specifically set to "DevOps". Recent years and recent developments raised the attention to the topic significantly. Furthermore, at the time of writing of the paper, no comprehensive review of the research was available. The study asks several research questions (RQs) and conducts a systematic mapping study (SMS) upon two defined search queries. Then, after screening and analyzing the results, the authors created a mapping to certain categories and several problems and their solutions. After a presentation of the results, a wide discussion of the found results shows some differences between the theory and effective results [WLS20].

2.1 Motivation

The purpose of the reviewed paper is to create an SMS to understand how Microservice Architecture (MSA) is used in conjunction with DevOps. Furthermore the objective is to identify, analyze and categorize the existing literature and research around the given topic. In addition to that, problems and their corresponding solutions - if any - should

be identified. The contribution of the paper to the academia is a classification of the research, a classification of problems and their solution, a list of identified research challenges, a classification of used tools and a list of formal or informal description tools for MSA [WLS20].

2.2 Methodology

With the goal to not limit the results of such an empirical software engineering method to one specific research question, the authors conducted an SMS instead of an SLR, which provides insight and secondary data for a particular question. The SMS contained three essential steps [WLS20]:

1. Planning the mapping study
2. Collection and analyzing the data
3. Mapping and documenting the results

The authors built the conducted SMS upon the guidelines purposed by Kai Petersen [PFMM08].

2.3 Research Questions

Ten different RQs in four categories were derived according to the goal of the paper [WLS20]:

Category 1: Demography, classification, and mapping of research	
RQ1.1	What is the frequency and type of published research on MSA in DevOps?
RQ1.2	What are the existing research themes on MSA in DevOps and how can they be classified and mapped?
Category 2: Problems, solutions, and challenges	
RQ2.1	What problems have been reported when implementing MSA in DevOps?
RQ2.2	What solutions have been employed to address the problems?
RQ2.3	What challenges have been reported when implementing MSA in DevOps?
Category 3: MSA description methods, patterns, and quality attributes	
RQ3.1	What methods are used to describe MSA in DevOps?
RQ3.2	What MSA design patterns are used in DevOps?
RQ3.3	What quality attributes are affected when employing MSA in DevOps?
Category 4: Tool support and application domains	
RQ4.1	What tools are available to support MSA in DevOps?
RQ4.2	What are the application domains that employ MSA in DevOps?

Table 1. Given research questions

The table above (Table 1) shows the research questions that the authors tried to solve with the systematic mapping study.

2.4 Search

To collect data, Waseem et al. used a two-phase-search. The first search was applied to the selected databases with the created search-query. The secondary search used a technique called "snowballing". "Forward snowballing" includes studies that cite the found studies, while "backward snowballing" follows the references of the found research material [Woh14].

The study limited the search to peer-reviewed studies from January 2009 until July 2018. They chose this particular starting point because the term "DevOps" was introduced in the year 2009 [WLS20].

Since MSA has multiple synonyms and can be in context with DevOps or without the said context, the following two search-strings were compiled [WLS20]:

1. `((microservi* OR micro-servi*) AND (architect* OR design OR structur*) AND DevOps)`
2. `(microservice AND DevOps)`

It is worth noting that the contextual part "DevOps" is not optional nor is it a construct like "microservi*". This limits the results to publications that must contain the term "DevOps" in their title.

The search was executed on the following seven databases [WLS20]:

- ACM Digital Library (https://dl.acm.org)
- IEEE Explore (https://ieeexplore.ieee.org)
- Springer Link (https://link.springer.com)
- Science Direct (https://www.sciencedirect.com)
- Wiley InterScience (https://onlinelibrary.wiley.com)
- EI Compendex (https://www.engineeringvillage.com)
- ISI Web of Science (https://webofknowledge.com)

The execution of the provided search queries on the given databases yielded a total of 494 studies. After the initial search, the authors screened and categorized the found studies into relevant and irrelevant publications. Of the original 494 studies, only 285 were flagged as "relevant". Waseem et al. then analyzed the studies according to six generic screening and one specific screening aspect. After the screening, 117 remained flagged as relevant. Those 117 studies were fully read and scored by the authors according to inclusion and exclusion criteria. At the point when the authors read and assessed all 117 studies, only 45 studies remained in the relevant category.

In addition to the conducted search, the snowballing technique yielded additional two studies that were included in the study. The authors compared the results of the snowball with the found results of the initial search and then the same practices were applied to those publications found with the snowball search.

After the search, a total count of 47 different studies remained relevant for the SMS [WLS20].

2.5 Results

After the conducted search and the thorough analytical readings and classification of the 47 studies, the collected results were separated in multiple sections to answer the RQs given in Table 1:

- Demography and classification
- Problems, solutions and challenges
- Description methods, patterns and quality attributes
- Tools and application domain

The following sections will summarize the found results to the specific research questions.

2.5.1 Demography and Classification. This subsection shall tackle RQ1.1 and RQ1.2 of Table 1.

The conducted search had a year span from 2009 to 2018 [WLS20]. Despite the broad search parameters of nearly considering a decade worth of publications, all of the relevant publications were published between 2015 and 2018. The following graph (Figure 1) should give some insight into the yearly publication rate and the type of publication:

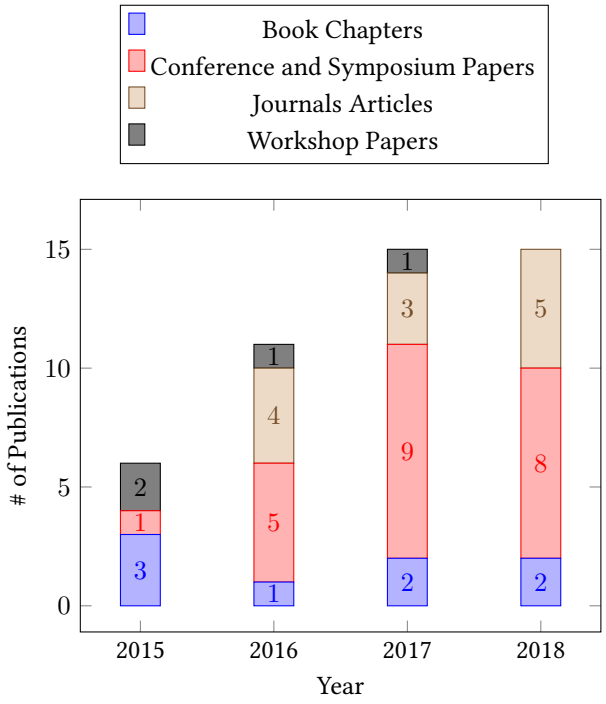

Figure 1. Publication activity

The distribution of publications over the years gives a detailed intuition about the general interest of researchers in the specified topic. As seen in the diagram above, the topic grew more important over the years 2017 and 2018. Since the first publications in 2015 there is an upward trend to the number of publications [WLS20].

As for the publication types, over those four years, 23 studies were published in conference proceedings, twelve in journals, eight in book chapters and merely four as workshop papers [WLS20].

The 47 considered papers were all published in 41 venues. The venues themselves can be divided into four different categories. 17 of the 41 venues count towards the topic **Internet, Cloud, and Services Computing**. Another 16 are categorized as **Software Engineering** venues. The remaining eight are split evenly between **Telecommunications and Networks** with four venues and **Multi-Disciplinary computing** also with four venues [WLS20].

As for RQ1.2, the result shows, that top two subtopics of the publications are *Approaches* (13 studies) and *Tools* (twelve studies). The least discussed topic, with only four studies, is *Monitoring* of microservices [WLS20].

2.5.2 Problems, Solutions, and Challenges. During the SMS, with the aid of a thematic analysis on the extracted data, a total of 24 problems could be identified. There exist several problems that could not be mapped with a documented solution so they represent challenges that need further investigation to provide the community with possibilities to resolve the problem.

The following lists should give a brief overview of the identified problems with an associated problem category. Inside the problem domain, the problems are not specifically ordered [WLS20].

Requirements of MSA-based Systems in DevOps

- Performance Issue due to Lack of Dedicated Access to the Host's Hardware
- Empowering Developers through Intelligent Software
- Performance Overhead due to Fine Grain Decomposition
- Scaling MSA-based Systems

Design of MSA-based Systems in DevOps

- Security and Privacy Across Cloud-Native Applications
- Providing Flexible Authentication to Each DevOps Team
- Application Decomposition into Microservices
- Reducing the Uncertainty in MSA

Implementation of MSA-based Systems in DevOps

- Managing and Migration Legacy Databases
- Modification and Integration of New Functionality in Existing Microservices
- Operational and Configuration Complexity

Testing of MSA-based Systems in DevOps

- Testing of MSA-based Systems in DevOps

Deployment of MSA-based Systems in DevOps

- Frequent Deployment in Different Environments
- Complexity in the Dynamic Deployment
- Deployment of MSA-based SaaS at Fine Granular Level
- Automatic Optimal Deployment of MSA-based Systems

Monitoring of MSA-based Systems in DevOps

- Logging and Post-Deployment Monitoring
- Monitoring and Execution of the Adaptive Actions
- Establishing and Maintaining Monitoring Infrastructure
- Monitoring Microservices at Run Time

Organizational Problems

- Introducing DevOps and MSA Culture
- People Resistance to Adopting DevOps and Microservices
- Less Familiarity with Implementing DevOps

Resource Management Problems

- Resource management for Development, Deployment, and Maintenance of the Cloud-Native Systems

For each of the problems in the shown lists, the considered publications provide at least one solution which can be viewed in "Figure 6" of the reviewed study. In general, a purposed idea to tackle multiple of the problems is to try not to decompose microservice applications too fine-grain [WLS20]. As for the general problem category "designing MSA based systems", multiple solutions are provided. Many different architectures are promoted and evaluated [WLS20].

To implement MSA based systems in a DevOps context, many studies suggest automated pipelines as well as automatic testing libraries. For communicating with other services, agnostic (i.e. independent and non-proprietary) technologies should be used (like REST over HTTP) to negate the need of knowledge of a specific programming language [WLS20].

Testing MSA based applications and systems should, according to the considered papers, be tackled with the given testing strategies that are in place right now. Which means "unit testing", "integration testing", "regression testing", among others [WLS20].

The topic of deploying MSA based systems is covered mostly by containerization and tools like "Docker Compose" and "Kubernetes" [WLS20].

Monitoring is purposed to be addressed with frameworks like "Unicorn"[1] and patter-based approaches like Omnia[2]. The general goal should be, that each team that owns the microservice should be enabled to monitor their responsibilities [WLS20].

Problems that relate to culture, people, cost, and other organizational topics are purposed to be dealt with guidelines for adopting to new structures in organizations. Cross-functional teams should be introduced and they should receive training to spread the acceptance of the new technology [WLS20].

As for the topic of resource management problems, some considered studies proclaim to use virtualized or containerized approaches and well established platforms to share the workspace among the developers [WLS20].

On the contrary, three challenges were identified which remained unresolved by the papers that were accounted for [WLS20]:

- Performance issues due to frequent communication
- Providing security at runtime
- Generating runtime architectural models

Performance issues can emerge when using too fine-grain microservices or when using synchronous communication channels to other services.

Studies found that security tends to be neglected in general which leads to severe vulnerabilities at runtime for microservice based architectures.

Generating models for MSA systems at runtime seems to be an unresearched topic but could be needed to help with decision-making processes in adaptive system development.

2.5.3 MSA descriptions methods, patterns, and quality attributes. The topic of the third category of RQs orbits around descriptive methods of MSA systems. What methods and patterns are used and which quality attributes they should suffice. The regarded studies provided different patterns and methodologies to describe their architectures and systems. To summarize the found description methods, five categories emerged [WLS20]:

1. Boxes and Lines (without any "framework")
2. Unified Modeling Language (UML)
3. Formal method (e.g. π-Calculus or equivalent)
4. Architecture Description Language (ADL)
5. Entity Relationship Diagrams (ERD) or Business Process Modeling Notations (BPMN)

Out of the 47 studies, 46 used some kind of description. The distribution is shown in Figure 2.

To achieve microservice architecture in complex systems, a multitude of patterns are purposed over all studies. The SMS identified 38 different design patterns across all regarded papers. Waseem et al. organized the mentioned patterns in the following three categories [WLS20]:

1. Circuit Breaker (five studies)
2. "Migration pattern" (four studies) [WLS20] [3]
3. Observer pattern (two studies)

[1] https://www.technative.io/unicorn-framework-the-rise-of-devops-as-a-service/

[2] Elaborated in [MT17]

[3] The reviewed SMS does not disclose which specific patterns or languages refers to here

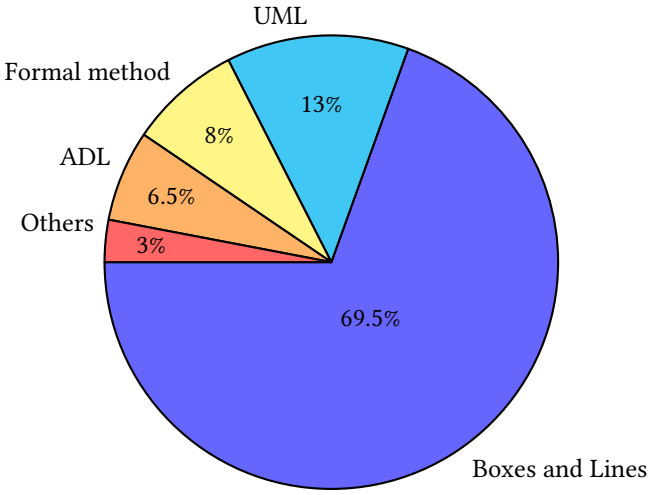

Figure 2. Distribution of descriptive methods

The circuit breaker pattern enables an MSA-based system to "fail fast". The pattern aims to prevent parts of a microservice architecture to cascade a failure beyond its own boundaries, which in turn could lead to a total system failure. Instead of waiting for unresponsive services, after some time the circuit breaker assumes the worst and deals with the fact that the work unit has become unavailable [MW16].

The observer pattern is a software design pattern in which some object (i.e. the "subject") maintains a list of observers. Whenever the subject changes its state, it automatically notifies all observers [GHJ+95].

Regarding affected quality attributes when using MSA in the DevOps context, the presence of said quality attributes (QA) was confirmed by the SMS. The QAs were split into two sections, one for positively influenced QAs when using microservices and one for the negative influenced ones [WLS20].

Positive The studies listed the following QAs as being positively influenced:

- Deployability (42 studies)
- Scalability (32 studies)
- Performance (26 studies)
- Maintainability (27 studies)
- Monitoring (23 studies)
- Testability (22 studies)
- Flexibility (20 studies)
- Availability (19 studies)
- Efficiency (19 studies)
- Security (ten studies)
- Portability (six studies)
- Compatibility (five studies)
- Modifiability (five studies)
- Usability (one study)

Negative Some studies listed the following QAs as being negatively influenced:

- Security (eleven studies)
- Performance (nine studies)
- Scalability (two studies)
- Reliability (two studies)
- Availability (one study)
- Compatibility (one study)
- Maintainability (one study)
- Modifiability (one study)
- Usability (one study)

To have a view from positively mentioned against negatively mentioned, consider the following chart:

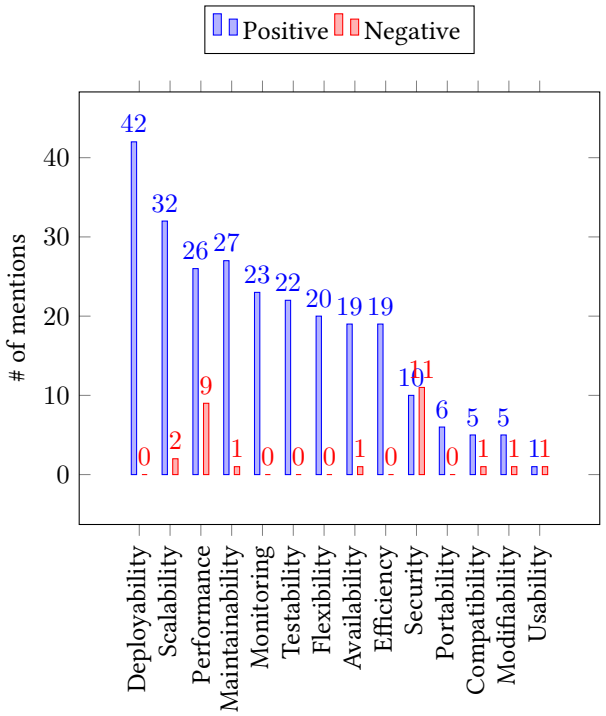

Figure 3. Influenced Quality Attributes

2.5.4 Tool support and application domains. The fourth category of research questions regards tooling support and given application domains. The SMS identified 50 different tools and 11 application domains.

The various tools can be seen in Fig. 7 in the SMS [WLS20]. The authors categorized them in the following list:

- Security Services and Tools (14 tools)
- Monitoring Tools (eleven tools)
- Continuous Integration Tools (seven tools)
- Testing Tools (six tools)
- Configuration Management Tools (five tools)
- Build Tools (five tools)
- Version Control Tools (two tools)

The last RQ that is to be answered, is which application domains exploit the combination of MSA with DevOps. The SMS pinpointed nine application domains with analyzation of the systems and topics in the regarded studies. Waseem et al. identified the following application domains in their study [WLS20]:

- Not Mentioned (15 studies) [4]
- Software Development Tools and Framework (eleven studies)
- Telecommunication (six studies)
- Mobile Software (four studies)
- E-Commerce system (three studies)
- Embedded system (three studies)
- Financial software (three studies)
- Healthcare software (one study)
- Webserver (one study)
- Distributed system (one study)
- Autonomic Management System (one study)
- Betting and Gaming (one study)
- Web Blog (one study)
- eServices Developments (one study)
- Container Management System (one study)
- Content Management (one study)
- Software for non-profit (one study)

As this list shows, beside studies that did not mention their concrete application domain, "Software Development Tools and Framework" has gained the most attention of all identified application domains in the study [WLS20].

2.6 Discussion

The following sections summarizes the "Discussion" of the SMS. The study analyzed the found results and explained certain trends.

2.6.1 Research status and themes. The limitation of the search to peer-reviewed literature from January 2009 to July 2018 is based on the "creation" of the terms MSA and DevOps. But the rise of papers and studies followed seven years later, around January 2016. The study noticed, that 41 papers were published from January 2016 until July 2018 [WLS20].

As seen in the systematic classification of the research themes, the most recurring topics are "Tools" with 13 studies, "Approaches" with twelve studies and "Development and Deployment" with twelve studies. This indicates, that the research is not only centered around new tools, but also regards development life-cycles as well. On the other hand, there were no publications found that focus on the topic of "Requirements Engineering", be it practices or any other activities [WLS20].

2.6.2 Problems and solutions. The given solutions in the regarded publications consist of design patterns, guidelines, frameworks, etc. For example, Domain Driven Design (DDD) and Model View Controller (MVC) patterns are recommended for decomposing an application into a microservice oriented system. The SMS also states that there are no studies found that address testing strategies for MSA based systems. As for optimal deployment of MSA, a very popular solution is the usage of containerization and Kubernetes [WLS20].

2.6.3 Challenges. A big concern in several papers is performance of such MSA based systems. The impact can be due to frequent communication between microservices. Also, poorly engineered architectures can lead to wide spread requests across the whole system. Also, when containers are used, the hardware underneath has a high impact on performance. The study shows that Amazon EC2 containers are worse than applications deployed on Amazon EC2 VMs [WLS20].

The second topic that gets addressed with high frequency, is security. When just "translating" applications to MSA, most of the time, security concerns arise. MSA based systems create complex access control scenarios without any matured patterns to harden the systems against attackers [WLS20].

2.6.4 Description methods and MSA design patterns. Most studies use just plain, informal boxes and lines as well as UML to describe microservice architectures. Other methods, like formal π-Calculator among others, are used rarely. The SMS argues, that this could be adressed with the creation of a standard description method for describing MSA [WLS20].

The used design patterns are shown in the corresponding table in the SMS. The most observed pattern is the "Circuit Breaker" [MW16] pattern which indicates, that cascading failures are a major concern. Next in line is the "Migration Pattern" [WLS20] that recommends various best practices for the transition from a monolytic application to a MSA based system. A not so well covered topic are patterns and recommendations that support CI/CD in MSA [WLS20].

2.6.5 Application domains. The SMS observed, that a third of the studies did not provide a specific application domain, nor any information to which domain they may count. The rest of the publications could be categorized into different application domains. The most referenced domain is "Software Development Tools and Framework". This results indicate, that MSA in the DevOps context is not bound to a specific application domain but rather is an improvement to a broad range of application domains such as healthcare, finance sector and embedded systems [WLS20].

[4] Those 15 studies did not mention any specific information regarding application domains

3 Critical review of the paper

The following section is a critical review of the conducted study. The review contains statements about the used empirical engineering methods as well as the found results and the followed discussion about the results.

As a general critique, the layout of the study should help the reader through the paper smoothly. There are many tables and figures which explain the results quite efficient and to the point, but the layout of the study forces the reader to jump around when going through it. It could have been a better approach to move all figures and tables to an appendix and have the text reference them.

3.1 Definitions

Some abbreviation (like "SLR") are introduced in the later stage of the study and leave the reader without knowledge about their meaning in the first few sections.

The general definition of microservice architecture (MSA) and DevOps are accurate to the literature and are well described. There are many references which allow the reader to further read into the topic of MSA and DevOps.

3.2 Empirical Research Methods

The study used the purposed guidelines of peer-reviewed publications to conduct the SMS. The authors did adjust and combine some of the guidelines according to the peculiarities of the topic.

3.2.1 Research Questions. The research questions are well structured into categories. However, the RQs in categories two to four appear too broad in my opinion. Some of the questions should be split up into finer definitions or should be more specific.

RQ2.1 *"What problems have been reported when implementing MSA in DevOps?"* [WLS20]: The problems that have been reported should be split into categories. Are the problems centered around general understanding of MSA, is it troubling to deploy and maintain a MSA based system or is the development (migration or new solution) process problematic?

RQ3.2 *"What MSA design patterns are used in DevOps?"* [WLS20]: The stated question about design patterns does cover *all possible pattern-topics*. So the question addresses patterns that are used for developing microservice applications and other patterns that recommend a way to migrate a monolith to a MSA. A categorization between the patterns would help the reader to grasp the results in the correct context. As an example this could be done with a *4+1 viewpoint* model (*Logical view*, *Development view*, *Process view*, *Physical view*, and *Scenarios*) from P.B. Kruchten [Kru95].

RQ4.1 *"What tools are available to support MSA in DevOps?"* [WLS20]: A similar categorization should be used for the research question about available tooling. Currently, this question ranges from planning tools (e.g. Jira) to tools for version control (e.g. GitHub).

3.2.2 Search strategy and Snowballing. The search for publications is conducted thoroughly. Many results are found which is a good sign for the search itself. It is an excellent idea to use two search queries to search for "microservices" and "architecture" in conjunction with the DevOps context.

On the topic of the search boundaries, it can be said that "January 2009" does include all the results that are found. Since the term "microservice" was defined by Martin Fowler in 2014 [FL14] the lower bound of the search could be set to "January 2014". According to Fowler, the term "MSA" was not precisely defined until then. The upper bound on the other hand is set to "July 2018" which is most likely the date when the search was executed. While the SMS does explain why January 2009 was selected as the lower bound, it is not stated at all why July 2018 is the upper bound. The reviewed study was published in the year 2020, but since the searched publications were limited to the year 2018, many new techniques and publications were ignored.

The limitation of the search to only allow peer-reviewed publications to be relevant does filter out some solutions to given problems and even challenges. In computer science, one big driving force of innovation are companies that need solutions and tools for the current problems at hand. Companies tend to have shorter innovation-cycles than the academia. With this matter in mind, many solutions are only described in blog posts or as showcases. It would have been a good approach to search for the problems along the peer-reviewed papers and allow some gray literature in the discussion to give advice to certain challenges and/or problems.

The search queries "`((microservi* OR micro-servi*) AND (architect* OR design OR structur*) AND DevOps)`" and "`(microservice AND DevOps)`" [WLS20] also exclude a relatively large portion of potential publications since the term "DevOps" is mandatory and without variants. While different writing styles and variants were used for the term "Microservice", the term "DevOps" is just plainly searched. There could also be material, that does not contain the term in the title at all. Those papers are excluded as well.

3.2.3 Quality assessment. The given screening criteria are well defined and create an exact baseline for the found papers such as the language of the publication or if it is peer-reviewed.

The stated qualitative criteria are nicely balanced and provide a good assessment of the found publications. It does make sense to search for clearly stated motivations, problems, and solutions in the found studies. Furthermore, it is eminent to know the limitations of the studies. The increased weight of the specific criteria versus the generic ones does help to search for the targeted publications that are relevant for the topic of the SMS.

3.3 Results and Discussion

Waseem et al. extensively describe the derived results and the subsequent discussion. Some findings and argumentation spark questions however.

3.3.1 Problems and Solutions. One of the stated problems in the SMS is that it did not find any studies about testing of MSA based systems in DevOps [WLS20]. Since gray literature was excluded from the search, some relevant publications from the industry were ignored. In the specific topic of testing MSA based and cloud-ready systems, Netflix provided a detailed description about an approach in their blog[5].

Among other established testing techniques (Unit testing, Integration testing, etc.) Netflix uses something they call "Chaos testing". The goal of this so called "Chaos Monkey"[6] is to test whole systems for resiliency by shutting down services, nodes, and entire clusters at random times. This is a specific testing use-case for MSA and cloud-ready systems. Since no system can guarantee 100% up-time, the chaos monkey introduces interruptions that the system must handle. The system should depend on some services not being available. With that in mind, a resilient system can be created [IT11, BBd+16].

3.3.2 Research Challenges. The first research challenge *Performance issues due to frequent communication* does state that too fine-grain MSA introduce complexity. I would like to argue, that nearly all fine-grain systems introduce complexity. The biggest issue, in terms of performance, with fine-grain MSA is the direction of the call-flow. Each level in this "service depth" does add to the total amount of time needed to compute the request of the user.

The problem with "service depth" is not the performance impact of computation itself in a work unit, but the general impact on the overall system when chaining subsequent units together. Figure 4 shows such a scenario. For the sake of simplicity, let us assume that each call in Figure 4 (i.e. Client to Api, Api to IAM[7], Service to Service, and Service to Datastore) takes `100ms`.

The assumed time contains computation, transformation from and to the transport protocol and the round trip. The basic idea is to show the performance impact when chaining services into a "deep system". With the given description, let us consider the following two scenarios:

1. *Client sends a SMS*: Client → Api → SMS Service. Api and SMS Service make a credentials check at the IAM[7]. The successful response to the client is delivered after: $4 * 100ms = 400ms$
2. *Client requests a document*: Client → Api → Document Service → Reader Service → Data Service → Data Store. All services check the credentials with the IAM[7]. The successful response to the client is delivered after: $100ms(\text{Client}) + 4 * 100ms(\text{IAM}) + 3 * 100ms(\text{Services}) + 100ms(\text{DB}) = 900ms$

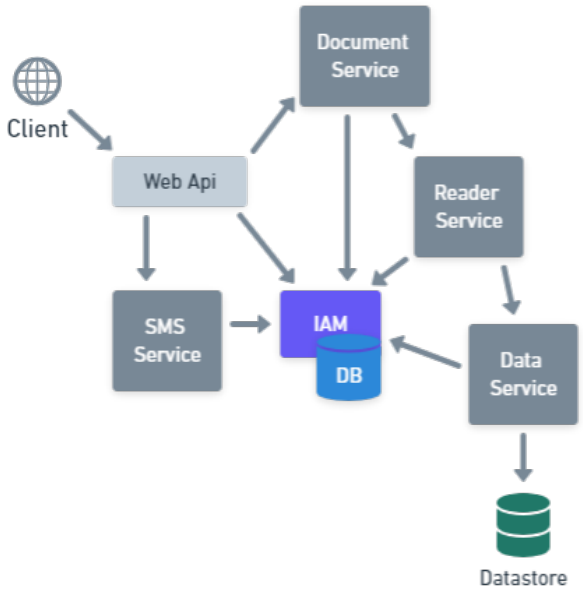

Figure 4. Deep MSA problem

A possible solution to this particular problem is to plan the microservices and the corresponding architecture accordingly. Frequent review of the current architecture and refactoring [Zim17a] if needed are key to prevent too deep services. This can happen in a very agile way since one of the basic goals of MSA is to have fast deployable artifacts.

The usage of synchronous HTTP calls will definitely result in a performance penalty. Systems like "Google PubSub" could help mitigate this bottleneck and create the possibility to scale instances of microservices on the horizontal scale. Furthermore, Kubernetes, among other orchestration platforms, provide developers with battle-proof tools to scale services with load-balancing features such as "services" which map to instances of the given services.

The statement about hardware is accurate. Depending on the hardware, the performance of the whole MSA based system will vary. For example, let us consider a microservice that was developed with PHP[8]. Depending on the used framework, PHP loads the `*.php` script files for each and every call that is made to the application. Since those script files are not cached in memory or any compiled form, when the underlying hardware contains storage with normal hard disk drives instead of solid state disks, the performance impact is huge. On the other hand, a compiled application like

[5] https://netflixtechblog.com/
[6] https://github.com/Netflix/chaosmonkey
[7] Identity and Access Management
[8] https://www.php.net/

a C# Web-Application does load itself into the memory of the container and does not have the need to load some files for each request.

The second challenge *Providing security at runtime* does describe some of the current problems. Security is a hard topic on itself and in conjunction with MSA, it does not get any easier. Several patterns exist that can provide some mechanism of security, but as always there is no silver bullet. A good strategy to authenticate and authorize calls made from a source would be the usage of OpenID Connect (OIDC) [Sir20]. OIDC enables a system to have a centralized user and access management. The calls are authenticated with a token and each microservice can check if the call is authenticated and valid or not. As for authorization, the central identity server can provide endpoints to fetch roles or other means of authorization logic to check if the call is allowed in the given microservice with the provided token.

3.3.3 Quality attributes. The SMS does a good job at showing the positive and negative statements of the relevant studies. The comparison and categorization is well done and gives an overview which topics contain solutions for non-MSA-problems and which problems are newly introduced by MSA and DevOps itself. The most important statement here is the security concern. Security is and will be a part of any public application and therefore is a central topic.

3.3.4 Tool support. The validity of this part of the results is questionable. The listed and found tools over all the regarded publications mix use-cases of various tools. As an example, "Jira" is listed as monitoring tool, but I would argue, that "Jira" is definitely not a tool for monitoring, regardless of the topic or the context. The results show the provide tools in the stated topics, but the discussion could have critically analyzed the tools and their positions in the categories. Another example is "Filebeat", which is listed under "Security Services and Tools". But Filebeat should be listed under monitoring or logging tools since it reads files produced by a service and sends them to an ELK[9] stack. Filebeat itself does nothing in terms of security, it is a log analyzer.

Furthermore, the vast majority of the listed tools are "Enterprise Tools". This does not necessarily mean they are not worth a try, but they tend to be aged and not created according to the newest and latest patterns and practices.

There are many open-source industry driven tools missing. Many modern companies use newer and more state-of-the-art tools. A few examples:

- **Clair** *Security*: A security scanner for containerized software that can statically analyze built containers for issues (https://blogs.vmware.com/opensource/2019/10/31/clair-container-security/)
- **Jaeger** *Monitoring*: Open-Source tracing tool that enables the developers to trace calls trough a complex system from end to end (https://www.jaegertracing.io/)
- **GitHub Actions** *CI*: The automated continuous integration platform from github
- **Gitlab CI** *CI*: The automated continuous intregration platform from gitlab
- **ArgoCD** *Configuration Management*: ArgoCD is an application that manages applications declaratively (GitOps pattern). Instead of using a CI pipeline to deploy an application to a cluster, the CI pipeline only creates the artifact (e.g. a Docker image) and publishes it somewhere. Then the pipeline can update the declarative description of the application with the new version of the image and Argo will be aware of the change and therefore will deploy the new image (https://argoproj.github.io/argo-cd/)

The industry has produced many more tools that solve some of the stated problems during the past years. Most of them were only mentioned or described in gray literature.

4 Conclusion

With this paper, we created a critical review of the systematic mapping study "A Systematic Mapping Study on Microservices Architecture in DevOps" by Waseem et al. [WLS20].

In section 1 the reader was introduced into the topic and various prerequisites were specified. We introduced terms like "Microservice", "DevOps" and methodologies from design science and empirical software engineering like "SMS", "SLR", and "SGLR".

Within section 2 the paper gave a brief objective summary for the reviewed SMS. The methods used are described with the conducted search and the found results. Afterwards the reader got an overview of the held discussion in the paper and the derived research challenges.

The critical review in section 3 then gives the reader a subjective review with critics to certain topics. In general the study covers a broad area of research and did sum up its results in a clean way.

The reviewed SMS yielded good results in the specific context of "DevOps". However the search terms that were used did limit the results to publications what contain the word "DevOps" in their title. This excluded results which may have opinions and solutions to found problems. Furthermore, the categorical exclusion of gray literature did limit the result to problems, solutions and tools from the academia without the results of the industry. The industry is a key driver for this topic in computer science and should be included in such a study. One possible outcome could have been that more research should be conducted based on statements and findings from the industry.

As a result of this review, a conclusion could be to conduct another study with regard to the problems stated in [WLS20] and map solutions from gray literature to their problems.

[9] https://www.elastic.co/elastic-stack

Another possibility is to create further research to topics presented as possible solutions in section 3 and create peer-reviewed publications that include gray literature.

References

[BBd+16] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.

[BWZ15] L. Bass, I.M. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. Always learning. Addison-Wesley, 2015.

[FD15] Babak Farshchian and Yngve Dahl. The role of ict in addressing the challenges of age-related falls: A research agenda based on a systematic mapping of the literature. *Personal and Ubiquitous Computing*, 19, 06 2015.

[FL14] Martin Fowler and James Lewis. Microservices, a definition of this new architectural term. https://martinfowler.com/articles/microservices.html, 2014. Accessed: 2020-11-21.

[GHJ+95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 1995.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[IT11] Yury Izrailevsky and Ariel Tseitlin. The netflix simian army. https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116, 2011. Accessed: 2020-11-22.

[Kru95] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[MT17] Marco Miglierina and Damian A. Tamburri. Towards omnia: A monitoring factory for quality-aware devops. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, page 145–150, New York, NY, USA, 2017. Association for Computing Machinery.

[MW16] Fabrizio Montesi and Janine Weber. Circuit breakers, discovery, and API gateways in microservices. *CoRR*, abs/1609.05830, 2016.

[Pae17] Arsenio Paez. Gray literature: An important resource in systematic reviews. *Journal of Evidence-Based Medicine*, 10(3):233–240, 2017.

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 12:1–10, 2008.

[Sir20] Prabath Siriwardena. *OpenID Connect (OIDC)*, pages 129–155. Apress, Berkeley, CA, 2020.

[SWH19] Andy P. Siddaway, Alex M. Wood, and Larry V. Hedges. How to do a systematic review: A best practice guide for conducting and reporting narrative reviews, meta-analyses, and meta-syntheses. *Annual Review of Psychology*, 70(1):747–770, 2019. PMID: 30089228.

[WLS20] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. A systematic mapping study on microservices architecture in devops. *Journal of Systems and Software*, 170:110798, 08 2020.

[Woh14] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[Zim17a] Olaf Zimmermann. Architectural refactoring for the cloud: a decision-centric view on cloud migration. *Computing*, 99(2):129–145, Feb 2017.

[Zim17b] Olaf Zimmermann. Microservices tenets. *Computer Science - Research and Development*, 32(3):301–310, Jul 2017.