Patterns in Microservices-based Development: A Grey Literature Review

Fabio Gomes Rocha[1], **Michel S. Soares**[1], **Guillermo Rodriguez**[2]

[1]Universidade Federal de Sergipe
Sergipe, Brazil

[2]ISISTAN (UNICEN-CONICET) Research Institute
Tandil, Buenos Aires, Argentina

fabio.gomes@souunit.com.br, michel@dcomp.ufs.br

guillermo.rodriguez@isistan.unicen.edu.ar

Abstract. *Microservices emerged due to the massive adoption of cloud computing and the need to integrate legacy systems. However, there still needs to be a greater understanding of adopting a microservice-based architectural style. Besides, there is a need for guidelines to operationalize those microservices. We conducted a grey literature review to identify commonly used architectural patterns and how they are implemented following design patterns. We present two key contributions. Firstly, we identified four architectural patterns and 23 design patterns. Secondly, we identified a catalog of tools for implementing the main patterns adopted when using the microservices style. The Proxy and the SAGA patterns are the most used in communicating and linking data for services. Additionally, tools such as Kubernetes, Docker, and Amazon WS are the most used for implementing microservices and deploying them into containers.*

1. Introduction

Software Architecture deals with the properties a system has, incorporating the elements and their relationships, design and evolution [ISO 2011]. In this sense, designing the architecture of the software is a complex activity, requiring analysis beyond the modular structure of the software, and it is necessary to consider the required technology and the dependencies between technology and modules [Sievi-Korte et al. 2019]. Thus, to design a high-level system, it is necessary to make decisions related to the various quality requirements. However, without this harming the system as a whole [Yang et al. 2016], [Ribeiro et al. 2018]. In this sense, microservices emerge, a software architecture pattern that emerges from systems based on service-oriented architecture (SOA), with characteristics arising from the adoption of the cloud in a distributed and independent way [Balalaie et al. 2015].

Microservices emerged as a result of the massive adoption of cloud computing and the need to integrate legacy systems [Balalaie et al. 2015]. Thus, the adoption of microservices architecture results in flexibility related to scalability and availability. However, according to Christoforou et al. [Christoforou et al. 2017], such a model brings with it new complexities, requiring analysis for adoption. In this sense, the authors point out the redistribution, as a high operational cost point, in addition to

the difficulty of maintaining the various resources and addresses [Balalaie et al. 2015], [Christoforou et al. 2017]. In addition to the problems pointed out, there is a need for new patterns for how these services' communications will be performed and how they will be implemented and operationalized. Thus, this article seeks to characterize the standards adopted by the industry through a systematic review of grey literature. We selected 125 articles between the years 2014 to 2021, divided into six bases. We identified four architectural patterns and 23 design patterns. In addition, we identified ten tools with five or more occurrences. We developed a catalogue of the main patterns adopted when using the microservices style based on the data. The rest of the paper is divided as follows: in section II, we present the background of the research, followed by the related work in section III. Section IV offers the grey literature methodology adopted in the study. Section V presents the results, answering the research questions, discussions about the results in section VI, and finally, the paper's conclusions.

2. Background

Using Microservices [Balalaie et al. 2016] as a solution for distributed software architecture has grown rapidly in the business environment in recent years [Balalaie et al. 2016]. Companies such as Netflix, eBay and Uber have adopted Microservices for the architecture of their systems to replace the monolithic architecture design. However, the lack of consensus in defining what a microservice is and what methodology to adopt when migrating from traditional services to this new paradigm has created several challenges for IT teams. Microservices require differentiated forms of infrastructure, here called agile infrastructure. Agile infrastructure consists of three layers: Technical, Project and Operations. Technical relates to hardware and software used in the environment. Project is about the process that introduces the changes into the environment. Operations is the process of keeping the environment working [Debois 2008]. Within this paradigm arise new technologies such as containerization environments, automation of deliveries, among others.

The microservices architecture paradigm can be considered an approach for developing a single application as a set of small services, each working in an isolated process and communicating through mechanisms [Kitchenham and Charters 2007]. Along this line, microservices have their growth linked to the platform of container [Pahl et al. 2020]. Containerization is a technology for virtualizing applications in a light way that resulted in a significant absorption in the management of cloud applications. How to orchestrate the construction and deployment of containers individually and in clusters has become a central problem [Pahl et al. 2017].

3. Related work

Systematic mapping studies and systematic literature reviews are used to find and assess relevant data for a research issue or topic. However, for this type of secondary research, only academic materials, often known as white literature, are used. We claim that grey literature is crucial for supplying useful insights to researchers from the industrial community. In this Section, we present the grey reviews already conducted on this field.

Based on an online survey (with three questions answered by 25 interviewed practitioners), Ghofrani and Lübke gave a preliminary analysis of state-of-practice on

microservices. Ghofrani and Lübke (2018)'s findings provide a high-level summary of industry-oriented microservices' difficulties without going into the specifics of their actual pains and rewards. Instead, the research attempts to provide a more in-depth examination of the technical/operational challenges and benefits of microservices as identified by industrial researchers and practitioners who work with microservices [Ghofrani and Lübke 2018].

Based on interviews with developers who have worked with microservice-based systems, Taibi and Lenarduzzi identified 11 microservice-specific bad practices. Some of the problems that we discover in our study, whose goal is distinct from Taibi and Lenarduzzi, reflect such faulty behaviors. Rather than conducting interviews to uncover problematic practices, the goal is to carefully analyze the grey literature on the topic to elicit microservices' primary technical/operational challenges and gains [Taibi et al. 2018].

The authors of [Taibi et al. 2020] wanted to help practitioners understand the various patterns by classifying them and reporting potential benefits and concerns. They used a multivocal literature review process, sifting through peer-reviewed and grey literature and categorizing patterns (standard solutions to common problems) as well as benefits and drawbacks. They found 32 patterns in 24 works, organizing them as orchestration, aggregation, event management, availability, communication, and authorization.

In [Soldani et al. 2018], the authors chose 51 industrial studies and analyzed them to distill the hardships and rewards of designing, implementing, and managing microservices due to their review. It also revealed that the industry's grasp of the benefits and drawbacks of microservices is reasonably advanced, implying that academia has a lot to learn from the industry on the subject. Valdivia et al. sought to expand knowledge on the design of microservices-based systems by presenting a multivocal systematic literature evaluation for microservices-related patterns, linking them together with quality attributes and metrics identified in academia and industrial research [Valdivia et al. 2020].

In summary, our research differs in that it examines architectural patterns to characterize them in terms of microservices from the perspective of professionals, development businesses, or independent researchers who have worked on the subject in the context of real projects.

4. Methodology

Grey literature refers to informal literature commonly published in blogs and web sites. Normally, grey literature is collected by using regular search engines, such as Google. Moreover, grey literature comprises of unpublished studies or doctoral theses, conference sessions, book chapters, government and agency reports, as well as blog entries, white papers and video presentations [Calderón et al. 2018]. Garousi, Felderer, and Mäntylä [Garousi et al. 2019] emphasize that software development professionals produce relevant and scaled Grey Literature, usually disregarded by academic research and that it is essential to read this literature in order to gather practical knowledge. This section presents how this review was conducted.

4.1. Research Question

The research objective is defined using part of the model GQM (Goal-Question-Metric) [Mashiko and Basili 1997, Van Solingen et al. 2002]: **Analyze** architectural patterns; **with the purpose of** characterising, **with respect** to microservices; **from the point of view** of professionals, development companies or independent researchers who have experience on the subject **in the context of** real projects. For this study, four research questions are established, as depicted in Table 1.

Table 1. Research Questions

Research Question	Description
RQ1	What is the profile of the publications on standards related to Microservices Architecture? This research question intends to find out (i) the number of publications on Microservices Architecture per year, (ii) the distribution by subjects of the papers, and (iii) the distribution of papers on scientific bases.
RQ2	What types of results were obtained, and what types of validation were adopted in the article? The intention is to identify, using Shaw [Shaw 2003] as a basis, and explain the articles' contributions to the knowledge of Software Engineering.
RQ3	What are the software architecture patterns/styles adopted on microservices projects? The intention is to identify which architectural patterns are adopted for the development of microservices.
RQ4	Which methods are adopted and used for operationalizing the architecture of microservices? The aim is to identify the methods and technologies used for the continuous delivery and operation of microservices.

4.2. Search sources

According to Garousi, Felderer, and Mäntylä [Garousi et al. 2019], the search sources of *grey literature* are generally classified into three levels:

- **Level 1**: Self-level of decision control and credibility - books, magazines, reports from government and reputable organisations in the field.
- **Level 2**: Medium level of decision control and credibility - annual reports, new articles, presentations, videos, and *sites* such as *StackOverflow*.
- **Level 3**: Low level of decision control and credibility - *blogs*, *tweets*, pages from social networks.

We formulated the string "*(architecture OR design) AND (microservice OR "micro-service" OR microservices OR "micro-services") AND (pattern OR style)*" to identify and select search sources as listed below, regarding only Level 2 sources:

- Website of Martin Fowler[1] (level 2). Selected by the author's credibility, who was one of the disseminators of the concept of microservices.

[1] https://martinfowler.com/

- DZone[2] website articles (level 2): created in 1997, the site has diverse content in the form of articles to discuss software development, considered one of the largest communities on the subject, and the submission of works performed by professionals and quality reviewed by the DZone team.
- Medium.com[3] Articles (level 2): appeared in 2012, and described as a blogging platform, or as described, a platform of things that matter, maintained by the company Obvious. Medium.com has diverse content, being highly accessed, mainly because it has a recommendation system.
- NewRelic[4] Blog (level 2) created in 2008 brings articles about what is new on software development, usually linked to services offered by the company.
- OpenSource[5] (level 2) is a portal maintained by RedHat created in 2010 with articles on various topics, with more than two million readers in 2020.
- InfoQ[6] (level 2) is a portal that focuses on helping progressive software development teams with diverse materials such as articles, interviews, and books, and content in several languages including English, Japanese, Chinese, Portuguese and French.

4.3. Criteria for Selecting Publications and Stopping the Search

For the selection of items in the search sources, Garousi, Felderer, and Mäntylä [Garousi et al. 2019] suggest that criteria for inclusion and exclusion of these items have to be applied, safeguarding the systematisation of choice. Thus, we defined the criteria presented in Table 2.

Table 2. Selection Criteria

ID	Inclusion Criteria
IC01	The article proposes guidelines or patterns according to developers' general experience
ID	**Exclusion Criteria**
EC01	Duplicated material
EC02	Material presenting only a brief summary
EC03	Material is not written in English
EC04	Material that does not deal with microservices standards

4.4. Process of Work Selection and Data Extraction

The material selection process was systematized in three stages:

1. execution of the search;
2. first filter; and
3. second filter. Then, the search was performed in the sources that were selected.

[2]https://dzone.com
[3]https://medium.com/
[4]https://blog.newrelic.com/
[5]https://opensource.com/
[6]https://www.infoq.com/

In the first filter, the titles and abstracts of the works found were read. For the videos, the title and descriptions were read, and the inclusion and exclusion criteria were applied. In the second filter, the studies were read in total, and the videos included in the first filter were watched in full. Again, the inclusion and exclusion criteria were applied. Thus, the items whose content did not meet the selection criteria were excluded, justifying the decision.

The execution of the search took place in May 2021, and there was no restriction on the time interval of publication. After execution of the search and application of the first filter, 156 items were returned. Next, following the application of the second filter, 125 items were selected, originating from six sources, as described in Table 3. The papers are distributed between years 2014 and 2021, as depicted in Figure 1. Furthermore, the trend of articles is appreciated in Figure 2.

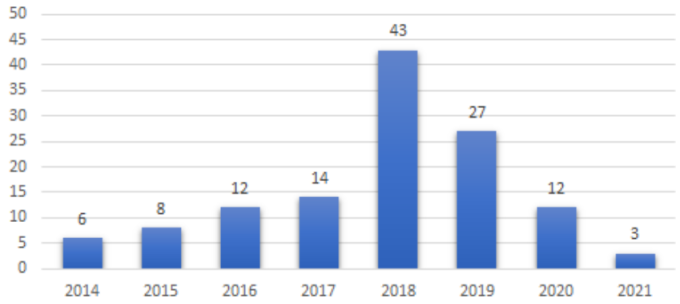

Figure 1. Distribution of works per year

Following the completion of selection of the second filter, the extraction of data was performed. The study was conducted by eight researchers (P1 to P8). The search strategy was developed as follows: Researcher P1 and P2 did a test search, identifying the number of papers and preparing a spreadsheet with the total volume of one hundred and fifty-six papers. Then, it was distributed to researchers P1 to P5 to read and identify the papers that would be part of the study. Then it was rotated; that is, P1 validated the results of P2, P2 of P3 and so on. Finally, P6 made a general validation. For data extraction, the papers were divided among researchers P1 to P5, and the extraction is validated by P6. Finally, researchers P7 and P8 performed the validation together, and all researchers performed the analysis of results.

Table 3. Distribution of works per year

Source	Total of works	Selected	Disclosures
Medium	18	15	3
Martin Fowler	9	9	0
DZone	44	44	0
New Relic	3	1	2
Open Source	32	10	22
InfoQ	50	46	4
Total	156	125	31

Data extraction is one of the main phases of conducting a literature review, including a grey literature review. It is from the data collected that one can answer the

research questions. Thus, for this purpose, a detailed reading of each collected item was conducted to identify, categorise and analyze the following items to answer the research questions.

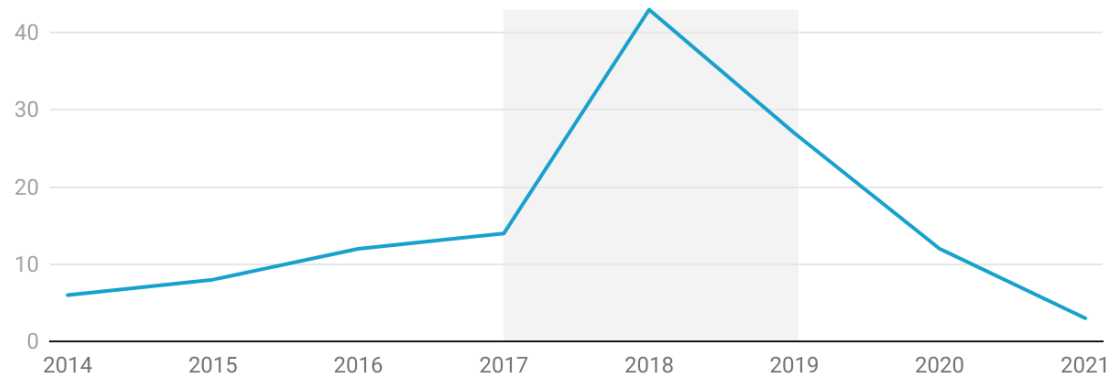

Figure 2. Distribution of works per year

5. Results

In response to **RQ1**, we have a total of 125 selected papers. The year 2018 was the peak of published papers, as depicted in Figure 2. One point to be monitored in the coming years is that in 2020 there was a drop. However, given that it was a year with drastic changes due to the COVID-19 pandemic, it will be necessary to monitor in the coming years if the theme stabilises, if it will still expand or if it is decreasing.

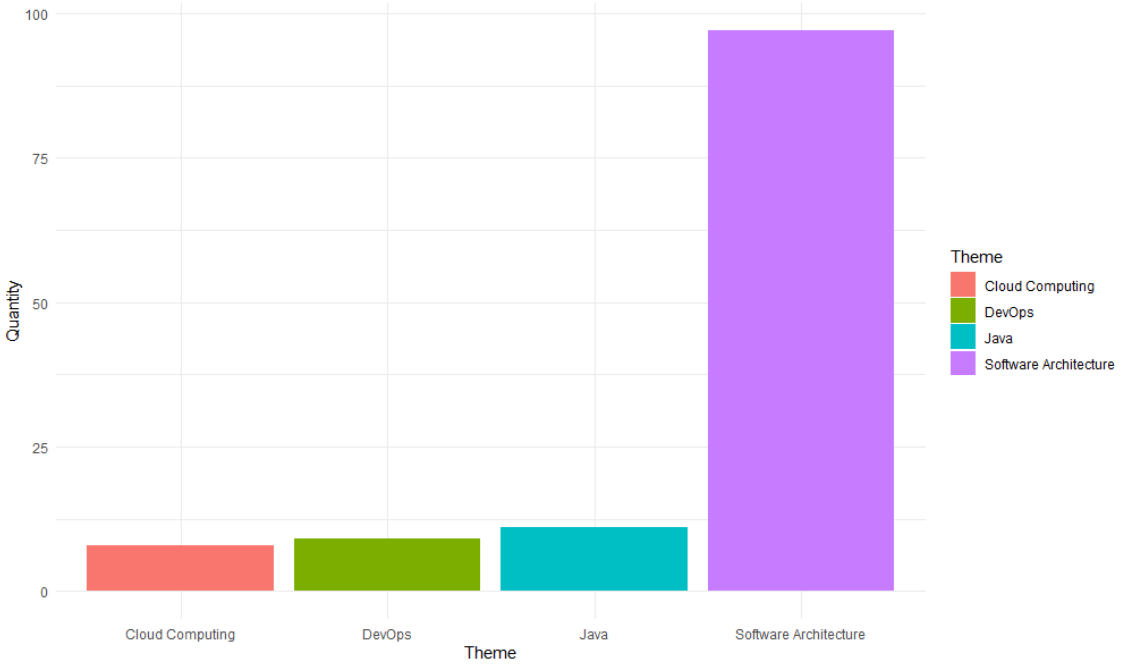

Figure 3. Distribution of works per Theme

Regarding the distribution concerning the work themes, four areas were identified: Architecture, DevOps, Cloud and Java, highlighting the theme of software architecture with 78% of the works, as depicted in Figure 3. About the scientific bases, out of the six bases, all had selected papers, highlighting the bases InfoQ and DZone (Figure 4).

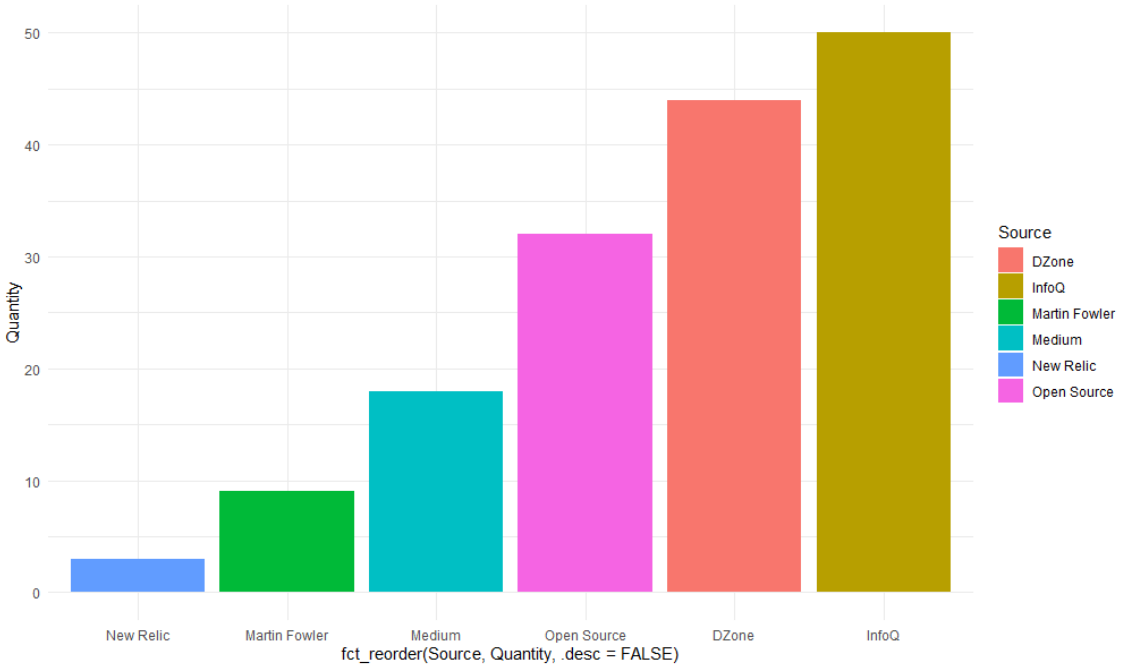

Figure 4. Distribution of works per Source

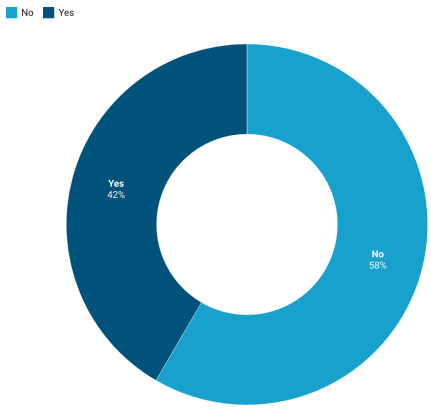

Figure 5. Type of work result

Regarding **RQ2**, first, we sought to understand if the papers published as grey literature had results. At this point, 41.6% had results. That is, they were not only theoretical papers, as depicted in Figure 5. Also, based on [Shaw 2002], of the papers that present any result, the results are validated, being that the same result can be validated in more than one way. The type of validation that stood out the most was the example, followed by statement - Figure 6.

In response to **RQ3**, an article may contain more than one pattern or style of architecture. The highlight is for the Pattern style with 40 works. However, 49 do not

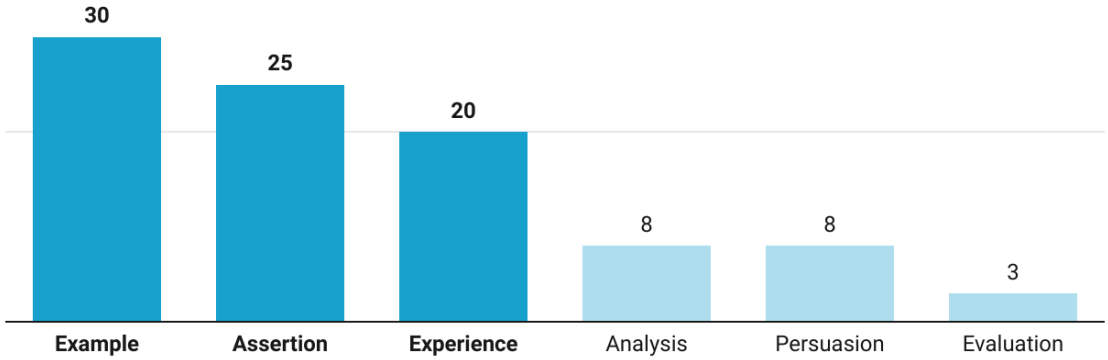

Figure 6. Type of validation

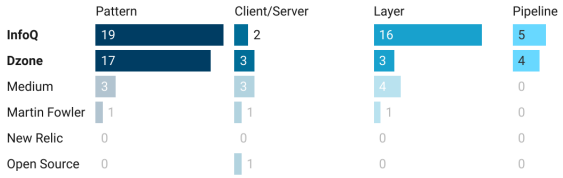

Figure 7. Distribution of Pattern per Source

present styles or patterns of architecture - Figure 7. As for the design patterns employed in microservices works, 23 patterns were identified - Figure 8 - but we can highlight the Proxy pattern with ten works.

InfoQ, Dzone and Medium are the bases that aggregate the most in Design Patterns - Figure 9. As a result, we obtained a set of patterns adopted in microservices architecture and presented them in Table 4 and Table 5.

In response to **RQ4**, of the one hundred and twenty-five papers selected for this article, nine deal with DevOps as a way to operationalize microservices. In addition, another eight deal with this operationalization in the cloud. It was also identified that three methodologies appear only once in the works: BDD, Pragmatic Microservices and Twelve-Factor app. The latter focuses on the delivery of software as services.

To support the operationalization, we identified one hundred and fifty tools. However, we described in Figure 10 those that were mentioned in at least two works. One highlight is Kubernetes with twenty-two occurrences, Docker and AWS with fourteen and Spring with 12. At this point, we can notice that the two items with the most significant highlights are related to the application infrastructure.

6. Discussion

The bibliography that deals with microservices bring some patterns starting in 2014, and since then added new works, highlighting the year 2018 with 48 new works. In total, we obtained 125 works dealing with DevOps, Architecture, Cloud and the Java programming language.

Another critical point is that, despite being grey literature, 42% of the papers presented some results. That is, it was not only an explanation article. In addition, from

Table 4. Patterns Catalogue.

Pattern	Description
Proxy	A proxy is a Structural design pattern intended to provide a substitute or placeholder for another object. Thus, the proxy controls access to the original object, allowing one to execute something before or after the request arrives at the original object. It has been adopted to control access to external microservices in this context.
SAGA	Thus, the proxy controls access to the original object, allowing one to execute something before or after the request arrives at the original object.
Low Coupling	It has been adopted to control access to external microservices in this context.
Strangler	The Strangler pattern focuses on application migration and consists of two types of services, where one implements already existed as monolithic, without new features. Subsequently, a second service implements new resources, thus generating value for the business by adopting the microservice without stopping the existing service.
Bulkhead	The Bulkhead pattern aims at fault tolerance, so the elements must be isolated in containers so that, if one fails, the others keep working.
Sidecar	Sidecar Pattern works together with the main container, with the actual application in a symbiotic way, acting as a single component, used as a reverse proxy, monitoring service, and observability and ambassador.
API Gateway	The API Gateway pattern creates a single entry point for all clients handling the requests.
Aggregator	Microservices should be independent, autonomous, and small. The Aggregator Pattern focuses on receiving customer requests xecutes requests to several microservices and combines the results to answer the initial request.
Gateway Routing	The Routing Gateway acts as a request router, i.e., a single point for clients to make their requests, and this route to the microservices.
Chained Microservice	The Chained Microservice Pattern produces a single, consolidated response to a request, even if it depends on several microservices.

Table 5. Patterns Catalogue (II).

Pattern	Description
Branch	The Branch Pattern is a set of other Aggregator and Chain design patterns that allows the simultaneous processing of several requests and generation of responses from two or more microservices
Client-Side Up Composition	The Client-Side Up Composition allows the independent development of user interface components responsible for a region of the page/screen for a specific service. In this way, it is possible to generate independence in developing screens and working with the composition of user interface components.
Service Discovery	The API Gateway consulted the Service Discovery pattern, which is responsible for discovering the available services. It usually works with the Service Registry pattern, which is responsible for registering the available services.
Circuit Breaker	The Circuit Breaker Pattern avoids cascading catastrophes when a service generates an error. The idea is that the call is packaged with a Circuit Breaker function that monitors failure. Once it reaches a certain threshold, the function stops, and new calls start to return errors without the protected call being performed, thus avoiding a problem.
Blu-Green Deployment	The Blu-Green Deployment pattern is a model for application deployment. The goal is to gradually transfer user traffic from an existing version to a new version; when both are running in the production environment, the old version is blue, while the new one is green. Thus, traffic is transferred from blue to green, and when completed, it can be removed from production or kept for reversion.
High Cohesion	High Cohesion is a Grasp Pattern and is usually linked to low coupling, aiming to keep objects focused, manageable and understandable. The term cohesion is used, in this context, to indicate the degree to which a microservice or class has single and well-focused responsibility.
State	The State is a GOF behavioral pattern that allows an object to change behavior when its internal state changes.
Singleton	Singleton is a creative GoF pattern that aims to ensure that a class has only one instance while providing a global access point.

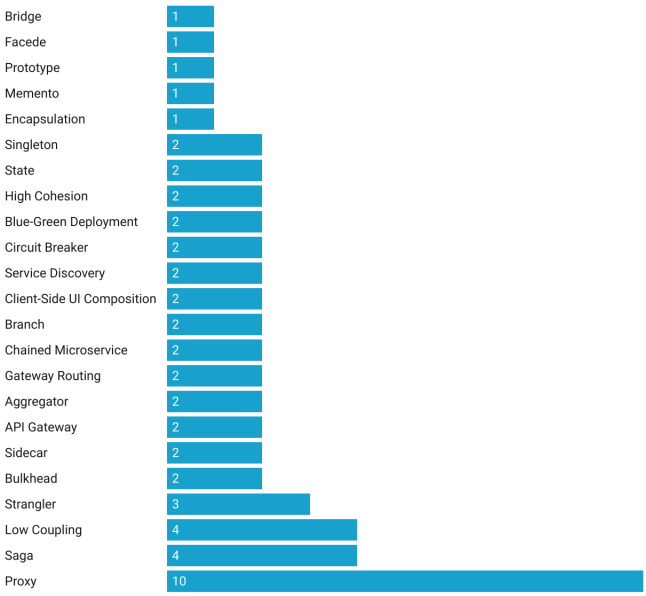

Figure 8. Distribution of Pattern

Figure 9. Distribution of Design Pattern per Source

these articles, we obtained some validations about the result, especially the validation by example, assertion and experience. In terms of patterns or styles of architecture, the highlight is the pattern of layers, and InfoQ and Dzone have more works that present these patterns. When it comes to Design Patterns, we obtained 23 patterns with at least one reference. Based on this, we prepared a table with the patterns refer.

6.1. Threats to Validity

According to Zhou, Xin et al. zhou2016map, we should address several aspects regarding the validity of a systematic review, the main ones being constructed, internal, external and conclusion validity.

6.1.1. Construct Validity

The results may not address the research questions related to the topic. To mitigate this threat, a pilot was conducted to validate the protocol and search chain to minimise this threat. All selection was conducted by one researcher and validated by another researcher, making it as broad as possible. Thus, the researchers searched independently, checking whether the results were related to the research questions and exchanging their results for validation.

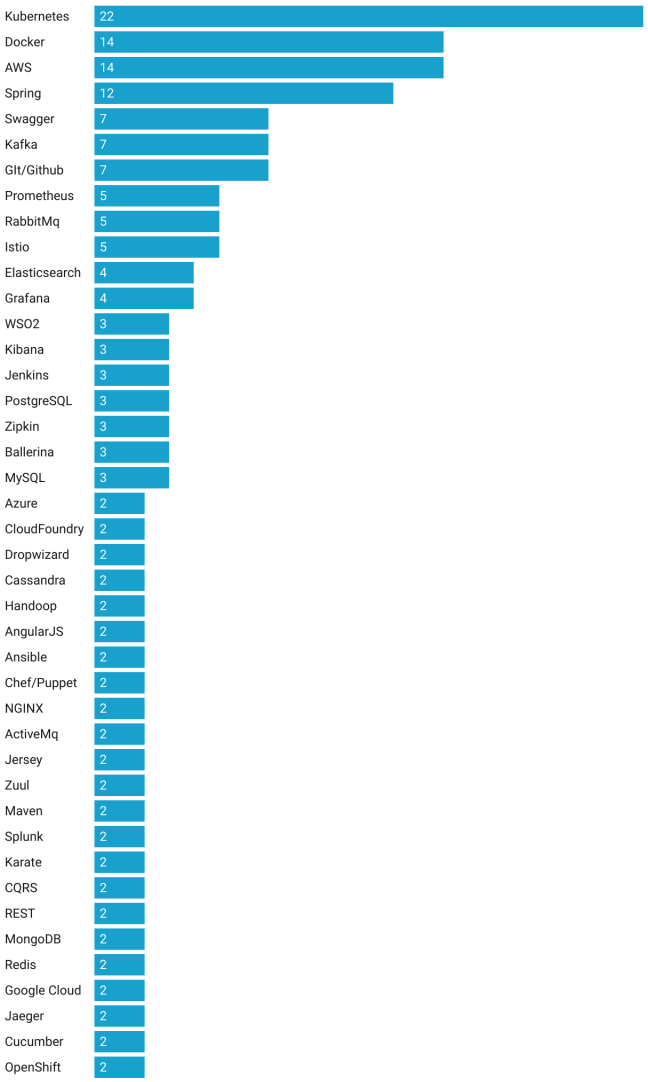

Figure 10. Tools for operationalizing microservices

6.1.2. Internal Validity

Threats to validity include those related to the protocol and the process of selecting papers. Thus, for both tasks, a double validation was performed. After the protocol was defined, a series of searches were tested before the search string was defined. In case of doubts about whether a paper should be selected, all researchers discussed the paper's suitability to the topic. The same occurred with the selection.

6.1.3. External Validity

This information cannot be generalised because no significant number of papers address architecture patterns applied to microservices. Thus, it was possible to identify other papers suitable for the context of patterns and/or microservices, but no references were made to them. The search chain was designed to reach as many documents as possible to

mitigate this threat, allowing the work to become a catalogue of patterns. However, it is well known that the software industry uses microservices in essential projects and that all this knowledge may not be published as papers. Therefore, although the grey literature brings a vast body of knowledge about microservices, many practices are not published.

7. Conclusions

It is noted that, beyond the academy, microservices have been receiving attention from professionals, with the main focus being: independence and lightness. This study sought to understand how the gaps left by the academy have been treated, in terms of standards for microservices, by the industry. We observed that the selected studies started publishing in 2014, with an apse in 2018 and a considerable reduction of works on the subject between 2020 and 2021. This can be attributed to the problems arising from COVID19; however, we would need more data and time for such a statement. It is, at this point, only an empirical finding, requiring a follow-up in the coming years on the subject.

Several mappings and systematic reviews have been published in this field, as presented in the related papers; however, unlike the studies presented, this paper sought to characterize architectural patterns and relationships of results and methods, but from an industry perspective, through grey literature. Thus, for this study, we followed a systematic and rigorous protocol, based on Garousi [Garousi et al. 2019], obtaining 125 studies and sought, based on these, to produce a view of the state of the practice on patterns/architecture styles adopted in practice with microservices.

As a result, it was possible to present a catalog of patterns currently adopted by professionals who develop using microservices architecture, highlighting the Proxy pattern, part of GOF, and the SAGA pattern used in the communication and linking of data for services. Consequently, we identified the main tools that allow the operationalization of microservices, highlighting Kubernetes, followed by Docker and AWS, demonstrating that the operationalization in containers with cloud load distribution has focused on the works presented.

In future work, we plan to extend our work as a multivocal literature review, considering formal literature. Furthermore, as part of our ongoing research, we want to apply a survey to gather data on the technologies used by practitioners to develop microservices from scratch or by migrating a legacy system. We want to learn more about the most popular tools, how people use them, and how they are integrated. With this knowledge, we may describe an implementation technique that supports academic or professional projects that call for a quick, secure, and dependable agile infrastructure.

References

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pages 201–215. Springer.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In *Communications in Computer and Information Science*, pages 201–215. Springer International Publishing.

Calderón, A., Ruiz, M., and O'Connor, R. V. (2018). A multivocal literature review on serious games for software process standards education. *Computer Standards & Interfaces*, 57:36–48.

Christoforou, A., Garriga, M., Andreou, A. S., and Baresi, L. (2017). Supporting the Decision of Migrating to Microservices Through Multi-Layer Fuzzy Cognitive Maps. In *Service-Oriented Computing*, pages 471–480. Springer International Publishing.

Debois, P. (2008). Agile infrastructure and operations: how infra-gile are you? In *Agile 2008 Conference*, pages 202–207. IEEE.

Garousi, V., Felderer, M., and Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106:101–121.

Ghofrani, J. and Lübke, D. (2018). Challenges of microservices architecture: A survey on the state of the practice. *ZEUS*, 2018:1–8.

ISO (2011). ISO/IEC/IEEE 42010:2011 Systems and Software Engineering — Architecture Description.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering.

Mashiko, Y. and Basili, V. R. (1997). Using the gqm paradigm to investigate influential factors for software process improvement. *Journal of Systems and Software*, 36(1):17–32.

Pahl, C., Brogi, A., Soldani, J., and Jamshidi, P. (2017). Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3):677–692.

Pahl, C., Jamshidi, P., and Zimmermann, O. (2020). Microservices and containers. *Software Engineering 2020*.

Ribeiro, F. G. C., Rettberg, A., Pereira, C. E., Steinmetz, C., and Soares, M. S. (2018). An Approach to Formalization of Architectural Viewpoints Design in Real-Time and Embedded Domain. In *21st IEEE International Symposium on Real-Time Distributed Computing, ISORC 2018, Singapore, Singapore, May 29-31, 2018*, pages 59–66. IEEE Computer Society.

Shaw, M. (2002). What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7.

Shaw, M. (2003). Writing good software engineering research papers. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 726–736. IEEE.

Sievi-Korte, O., Richardson, I., and Beecham, S. (2019). Software Architecture Design in Global Software Development: An Empirical Study. *Journal of Systems and Software*, 158:110400.

Soldani, J., Tamburri, D. A., and Van Den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232.

Taibi, D., El Ioini, N., Pahl, C., and Niederkofler, J. R. S. (2020). Patterns for serverless functions (function-as-a-service): A multivocal literature review.

Taibi, D., Lenarduzzi, V., and Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. In *CLOSER*, pages 221–232.

Valdivia, J. A., Lora-González, A., Limón, X., Cortes-Verdin, K., and Ocharán-Hernández, J. O. (2020). Patterns related to microservice architecture: a multivocal literature review. *Programming and Computer Software*, 46(8):594–608.

Van Solingen, R., Basili, V., Caldiera, G., and Rombach, H. D. (2002). Goal question metric (gqm) approach. *Encyclopedia of software engineering*.

Yang, C., Liang, P., and Avgeriou, P. (2016). A Systematic Mapping Study on the Combination of Software Architecture and Agile Development. *Journal of Systems and Software*, 111:157–184.