Systematic Literature Review of Microservice Architectures

Thijmen J. Kurk
Vrije Universiteit Amsterdam, The Netherlands
t.j.kurk@student.vu.nl

ABSTRACT

A microservice architecture is an interconnected web of small independent services that each run in its own process while interacting with each other using messages. These architectures are still evolving giving rise to the need for literature reviews. We aim at identifying and classifying definitions and patterns found in microservice architectures described by scientific literature into a concise model. Additionally, we set out to identify challenges found in inter-microservice data management and classify their possible solutions. Towards this goal, we apply well-known systematic literature review methods. Following these methods, we selected 21 relevant studies and iteratively updated our model based on these studies. This work contributes (i) a general model of a modern microservice architectures and (ii) an overview of challenges and solutions within inter-microservice data management.

KEYWORDS

Systematic Literature Review, Microservice, Microservice Architecture, Data Management

1 INTRODUCTION

Nowadays, microservices are the norm when software engineering scalable applications, especially considering that cloud computing and containerization technologies are becoming more prominent [1, 16, 21]. Amazon, Netflix, LinkedIn, Spotify, SoundCloud and other companies [8, 35, 37] are actively adopting and evolving architectures in which microservices are deployed. The first definition of a microservice was introduced by [12] in 2014 as: *'A service that can be automatically and independently be deployed, runs in its own process and communicates using lightweight mechanisms'*. Complementary to this definition, [12] specifies a microservice architecture (MSA) to be a collection of microservices working together towards desired business objectives.

The **goal** of this paper is to analyze published literature on MSAs and classify them. This classification will be done by iteratively creating a model using the definition of, and patterns in MSAs. Additional to this goal we zoom in on inter-microservice data management and classify problems and solutions commonly found therein.

The **audience** of this study are researchers, software engineers and developers who are interested in microservices or are inquisitive about the current state of inter-microservice data management.

The **outline** of this paper is as follows, Section 2 discusses the relevant work we have found and how our research fits into this picture. Subsequently, Section 3 goes into detail on how our systematic literature review is designed. Followed by Section 4 where we define the model and answer our research questions based on the selected literature. Next, in Section 5 we discuss the results. And finally in Section 6 and Section 7 we cover common threats to validity and conclude our study respectively.

To simplify to model used in Section 4, we choose to use the following definition of microservices (Definition 1) and MSAs (Definition 2) both originating from [13], where a process can be either be a process as defined by the operating system (OS) or a collection of threads within an OS process. These definitions generalize better and allow us to define a model that contains less edge cases. For example, using these definitions we can define services used for persistence to be microservices as well, although they do not directly implement logic towards any business objective. Additionally, we use the term service and microservice interchangeably in this paper.

Definition 1 (Microservice, MS). *A microservice is a cohesive independent process interacting via messages.*

Definition 2 (Microservice Architecture, MSA). *A microservice architecture is a distributed application where all its modules are microservices.*

2 RELATED WORK

Numerous secondary studies on MSA have been performed [3, 8, 13, 14, 25, 32, 33]. A majority of these studies aim to provide an overview of what components generally reside in an MSA including but not limited to, how quality attributes [24] can be applied to each of these components [3, 13, 25]. Whereas others concentrate on the patterns in which these components can be utilized most effectively within MSAs [1, 7, 22, 23, 28, 33].

This study distills the work published on MSAs into a model by iterating through studies systematically. In contrast to the work mentioned above, we put the focus on creating a general model of MSAs, instead of enumerating organizational aspects or specific patterns.

3 STUDY DESIGN

This study is designed and carried out by following the guidelines on secondary studies [10, 17, 18]. We provide a replication package, which contains the raw data from each intermediate step executed during the search and selection phase (including any scripts that were used), over at our GitHub[1].

3.1 Research Approach

The goal of this literature study is to provide the reader with a model that contains an overview of components that are generally found inside microservice architectures. Furthermore, we describe the challenges and available solutions that are commonly used for the data management between microservices (i.e., transactions that

[1] https://github.com/ThijmenKurk/literature-study

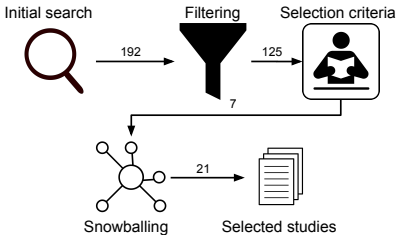

Figure 1: Search and selection process

spawn over multiple microservices). Towards our goal we have drafted the following research questions.

RQ1 How can modern microservice architectures be modelled?
- We drafted this research question to provide a generalizable model for modern microservice architectures derived from scientific literature. This high-level overview will facilitate the next research question by providing context, definitions, and concepts found in microservice architectures.

RQ2 What are the challenges and available solutions used for inter-microservice data management?
- We aim to identify challenges within inter-microservice data management such that we can expose the (potential lack of) available solutions.

3.2 Search and Selection

We now describe the search and selection process of this study. The study design (i.e., the search string and selection criteria) is locked in after the start of the search and selection process to avoid any personal biases. This and other threads to validity are discussed in Section 6.

3.2.1 Initial Search. For this study *Elsevier Scopus*[2] is used as our search engine. This tool has functionality such as the ability to, export search results to different formats (including but limited to CSV and BibTex), apply in-depth filters and search through both citations and references of studies automatically. All of which are used to facilitate the next steps. Our initial selection of 192 studies was a result of the query shown in Figure 3.

3.2.2 Filtering. Using the filter functionality of Scopus we were able to further refine our selection. First, all literature not pertaining to the subject area computer science is removed. Secondly, we filter the literature such that it only contains conference papers, articles and book chapters that are in a finalized state. Thirdly, we exclude any papers that are not written in English. Finally, we omit literature with the subject of microservices in combination with Internet of Things (IoT) and network architectures, since this is out of scope of this review. Refining our selection to a total of 125 studies.

3.2.3 Application of Selection Criteria. The selected studies are manually filtered using the inclusion and exclusion criteria [10, 17, 18] listed below.

I1 Studies focussing on microservices or microservice architectures.
I2 Studies focussing on data management within microservice architectures.
I3 Studies that are peer-reviewed.
I4 Studies that are written in English.
E1 Studies that marginally describe a microservice architecture.
E2 Studies describing the migration process from a monolith to microservice architecture.
E3 Studies that are of bad quality (e.g., many spelling errors, unreadable sentences, etc.)
E4 Studies not available as full-text.

$$(I1 \vee I2) \wedge I3 \wedge I4 \wedge \neg E1 \wedge \neg E2 \wedge \neg E3 \wedge \neg E4 \quad (1)$$

The criteria are applied by following equation 1 during each of the following two steps. First, we read the title and abstract of the study. Last, we read each study full-text. After the manual filtering is done we are left with a selection of 7 studies.

3.2.4 Snowballing. During this phase *forward* and *backward* snowballing [36] is used on the selected studies. On completion, we expanded our selection to 21 studies.

3.2.5 Final Selection. The search and selection process selected 21 studies for this review. An overview of the selected studies can be found in Table 2.

3.3 Data Extraction

During the data extraction phase, every selected study is read full-text and information pertaining to the research questions is stored in a spreadsheet. The goal of this phase it to find patterns within MSAs such that the primary components within these patterns can be described and modelled.

3.4 Data Synthesis

During the data synthesis phase we iteratively go through the extracted data and update our model to fit it. This method is called narrative synthesis and is commonly used when synthesizing literature in the context of systematic literature reviews [26]. The product of this step is a generalized model of MSAs and the challenges and potential solutions within data management, both of which are described in Section 4.

4 RESULTS

In this section we discuss the results of our literature review. The iterative model created from the selected studies is displayed in Figure 2. It defines the main components inside a microservice on an implementation level without detailing any topological specifics. Now the model is described.

4.1 The Model

There are four types of microservices, all of which are interconnected with other microservices through a network. (i) Business microservices implement logic towards a business objective, (ii) utility microservices facilitate business microservices with general functionality (e.g., logging, monitoring, circuit breakers, load balancers, service discovery, etc. [28]), (iii) persistence microservices typically provide database management systems (DBMS) or cache systems to business microservices, and finally (iv) middleware services which facilitate communication between other microservices

[2] https://www.scopus.com/

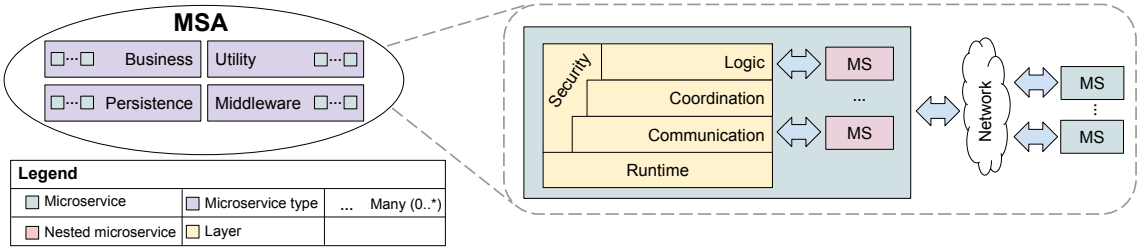

Figure 2: Generalized model of microservice architectures

(e.g., an API gateway, a message broker, a message bus, etc. [13, 28]). An MSA is an interdependent cohesive loosely-coupled composition of these four types of services.

Every microservice can have nested microservices which only the parent microservice is allowed to communicate with. Nested microservices can be colocated on the same node as the parent microservice or hosted on separate nodes [28]. The former is generally the case with the sidecar pattern in which nested utility microservices are deployed to abstract away general functionality from the parent, while the latter is often the case with persistence services [23, 28]. A microservice consists of 3 layers with one overarching layer being security [13, 23, 28, 31] and one foundational layer being runtime. Each layer is now described.

Runtime of a microservice consists of the platform on which it is executed. The industry has evolved such that physical hardware is abstracted away behind many layers of virtualization [28]. There are two levels of virtualization relevant to MSAs, (i) Hardware Abstraction Layer (HAL) virtualization (i.e., virtual machines [28]) and (ii) OS-level virtualization (i.e., containers [16, 28]). Microservices are typically deployed in either, or a combination, of these types of virtualized environment [8].

Communication between microservices consists of three parts [31], (i) the interaction model, (ii) the transportation and finally (iii) the presentation. The interaction model between services can be synchronous or asynchronous. Synchronous communication implies a request/response type pattern, meaning that it is vulnerable for services blocking other services while they are waiting for a response [7, 23, 28]. Asynchronous communication is non-blocking by definition and often comes hand in hand with event-based architectures [34]. However, it requires state management within microservices and typically incurs more communication overhead due to it generally requiring some kind of middleware service, such as a message broker or message bus to facilitate and decouple message delivery [7, 19, 28]. Messages between microservices are transported either via the network or through Inter Process Communication (IPC) depending on performance and scalability requirements. The protocol used for transportation should be selected while considering the desired interaction model, since it can be performed using a variety of protocols [31] (e.g., HTTP, gRPC, XMPP, MQTT, AMQP, etc.). Finally, presentation describes how the data is serialized such that it can be transported between services. Common serializers are JSON, XML and Protobuf [28].

Coordination between services inside an MSA is necessary since it is a distributed system per definition [7, 23, 28]. The coordination within an MSA to perform more complex and elaborate functionalities are either choreography or orchestration based [13, 15, 28]. Orchestration requires composite services that directly coordinates with other services to oversee the process by receiving responses. On the other hand, choreography uses asynchronous events with the publish/subscribe pattern to facilitate collaboration [13, 23, 28].

In essence, the coordination between microservices is about data management [20]. There are various challenges in data management all of which are subject of the next research question.

Logic is the last layer inside of microservices and includes all the programming required to implement the necessary functionality for each particular service. This can be anything due to our wide definition of microservices, ranging from business logic to DBMS logic. One important part of the logic layer is state management. Service can either be stateful or stateless [7]. Statefulness implies session affinity, which in term means less flexibility and scalability [2]. How and if state should be managed within a service can therefore be determined from the scalability requirements of that service.

Security is a layer that overarches all the previously defined layers and is challenging to implement properly. However, it vital within MSAs, due to the large attack surface and complicated connections between services [30]. There are two approaches to securing distributed systems, either a zero-trust-network or a trust-the-network approach [4, 6]. Depending on the security requirements of the MSA, a certain network type should be chosen [4, 6, 13, 30].

4.2 Data Management

The functionality of business microservices is commonly determined by applying domain-driven design (DDD) techniques [11]. In DDD, the problem space of the business is referred to as the domain. This domain can be divided into multiple subdomains each representing a different part of the business. Every subdomain has its own domain model, the scope of which is called a bounded context [9, 28, 29, 34]. From this bounded context, one or more business microservices can be derived [28]. Typically, there is some (minimal) data dependency between the bounded context of these services. Therefore, inter-microservice data management (handled by the coordination layer in the model, Figure 2) is required, given that MSAs are commonly designed with a database per service

pattern [32]. According to the **CAP** theorem, any networked system which shares **P**artitions of data can only pick one of/attain a balance between the following two properties: **A**vailability or **C**onsistency [5]. For example, a highly available MSA cannot be consistent all the time [28]. Therefore, depending on the consistency and availability requirements of the MSA, different solutions to facilitate inter-microservice data management are desirable. We now describe three common data management challenges found within MSAs.

4.2.1 Inter-microservice data relations. Consider two business microservices, one that keeps track of departments and one that keeps track of employees. Every department can have zero-to-many employees. In a relational DBMS, this type of relation is enforced by a foreign key on employee referring to the department of which the employee is a part of. The DBMS enforces policies to keep the database consistent, by for example, cascade deleting employees that are member of a deleted department, or making sure that every employee is member of a valid department. However, this type of enforcement is not trivial if each service has its own (possibly different type of) DBMS [20, 23, 28]. Foreign key emulation is currently resolved using ad-hoc implementations due to the lack of a general solution [20].

4.2.2 Inter-microservice data transactions. A transaction is a unit of work that needs to be completed in its entirety or rolled back. This is challenging to implement in MSAs due to the use of (polyglot) distributed persistence [7, 20, 28]. Consider two business microservices, X_1 that keeps track of inventory and X_2 that is responsible for the placement and tracking of orders. When an order is placed, the stock of the product is updated through the inventory service. A product with only one left in the stock is ordered by two users at the same time. Now X_2 receives two requests to check the stock followed by two requests to update the stock. A naive implementation allows the product to be ordered twice, since X_1 and X_2 do not coordinate the transaction. There are two solutions for this problem, depending on the desired properties of the MSA, displayed in Table 1.

Saga is pattern in which eventual data consistency can be guaranteed by splitting up inter-microservice transactions into a sequence of compensable (potentially retryable) subtransactions each having the scope of only a single microservice [20, 28]. A Saga transaction is successful if each subtransaction has succeeded and can be coordinated using either a choreography or an orchestration based interaction model [28]. If one subtransaction fails, the already succeeded subtransactions are rolled back by executing their respective compensating subtransactions. This makes the MSA eventual consistency and basically available [5].

Two-Phase Commit (2PC) is a pattern in which strong data consistency is guaranteed by orchestrating each operation of the inter-microservice transaction in two phases [20, 28]. The transaction orchestrator starts the first phase by requesting each service to prepare the necessary operations and waiting for each to respond. The second phase executes after each service has successfully prepared and makes the operations permanent. The first phase requires each service to lock resources until the transaction is finalized or aborted since otherwise transactional consistency cannot be guaranteed [20, 27]. This trades availability for strong data consistency [28].

Pattern	Interaction Model	Consistency	Availability
Sagas	Orchestration, Choreography	Eventually consistent	Available
Two-Phase Commit	Orchestration	Consistent	Less available

Table 1: Inter-microservice transaction patterns

4.2.3 Inter-microservice query aggregation and joining. When a microservice requires data from multiple service, it needs to retrieve and aggregate data from each microservice individually and deal with possible inconsistencies by itself [20]. These inconsistencies can form due to, for example, the incorrect implementation of ad-hoc solutions described in Section 4.2.1 or a service failing to return the requested data [28]. Typically, queries that require aggregation or joins are highly optimized inside of DBMSs. Therefore, performing them manually on data from different services implies steep performance penalties. A common solution to this problem is the Command Query Responsibility Segregation (CQRS) pattern. With CRQS, so-called views (read only copies) of tables are prematurely colocated at microservices that require them to speed up aggregation and join operations [28]. Only the microservice owning the table can write to it. Writes are streamed to the views via events using the publish/subscribe pattern, making them eventually consistent [28].

5 DISCUSSION

This literature review aimed to model modern MSAs and the important components within them. This provides researchers with an overview of important aspects and layers to consider while implementing MSAs. We found similar studies within our selection of literature, but these studies considered different aspects of MSAs (e.g., organizational aspects or architectural aspects) instead of implementation aspects.

Additionally, this study derived three challenges and potential solutions for data management problems within MSAs from the selected literature. The lack of solutions for some of these challenges show that MSAs are still evolving and not fully mature yet.

6 THREATS TO VALIDITY

In this section, we discuss the most prominent threats to validity of this study.

Internal validity. The internal validity threat is related to the design and execution of the literature review [17]. This threat is mitigated by defining a detailed research protocol in Section 3 and locking this protocol so that it cannot change after the review is started to prevent personal bias.

External validity. The external validity threat is related to the generalizability of the results [17]. More specifically, for a literature study this would mean that the selected studies not being

representative of the full population (i.e., all the available published literature). To mitigate this form of bias, we used a generic search query (Figure 3) inside of a search engine that indexes scientific literature from multiple sources. Additionally, we utilized both forward and backward snowballing [36] to further expand our selected literature with studies our automatic search might have missed.

Construct validity. The construct validity threat concerns the relation between theory and observation. We mitigated this bias by searching multiple sources using Scopus with only general terms in our search string. Additionally, we rigorously selected relevant studies according to inclusion and exclusion criteria [10, 17, 18].

Conclusion validity. The conclusion validity concerns the degree in which our conclusions are reasonable based on our extracted data. We mitigated this by iteratively defining and updating our model during the data extraction and synthesis phase. Additionally, this threat was mitigated by applying well-known systematic literature review methods [10, 17, 18, 36].

7 CONCLUSION

This study gives an overview of modern microservice architectures in the form of a model and identified three data management challenges and their potential solutions by systematically analyzing the latest studies on microservices using well-known literature review methods [10, 17, 18, 36].

We found that microservices consist of four layers: (i) Runtime, (ii) Communication, (iii) Coordination and (iv) Logic, with one overarching layer being Security. These microservices are connected with each other to form a microservice architecture. There are four types of microservices: (i) Business, (ii) Utility, (iii) Persistence and (iv) Middleware. Additionally, we found three data management challenges: (i) inter-microservice data relations, (ii) inter-microservice data transactions and (iii) inter-microservice query aggregation and joining. We conclude that microservice architectures are evolving but that they are far from mature, given the fact that not all data management challenges have general solutions.

In the future we would like to explore how to solve the challenge described in Section 4.2.1 by proposing a general solution for this problem.

REFERENCES

[1] A. Akbulut and H.G. Perros. 2019. Performance Analysis of Microservice Design Patterns. *IEEE Internet Computing* 23, 6 (2019), 19–27. https://doi.org/10.1109/MIC.2019.2951094 cited By 12.

[2] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2018. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing* 11, 2 (2018), 430–447. https://doi.org/10.1109/TSC.2017.2711009

[3] N. Alshuqayran, N. Ali, and R. Evans. 2016. A systematic mapping study in microservice architecture. *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016* (2016), 44–51. https://doi.org/10.1109/SOCA.2016.15 cited By 184.

[4] Doug Barth and Evan Gilman. 2017. Zero Trust Networks: Building Trusted Systems in Untrusted Networks. (2017).

[5] Eric Brewer. 2012. CAP twelve years later: How the "rules" have changed. *Computer* 45, 2 (2012), 23–29. https://doi.org/10.1109/MC.2012.37

[6] Binildas Christudas. 2019. Microservices Security. In *Practical Microservices Architectural Patterns*. Springer, 733–777.

[7] Shahir Daya. 2015. *Microservices from theory to practice: Creating applications in IBM bluemix using the microservices approach*. IBM Corporation, International Technical Support Organization.

[8] P. Di Francesco, P. Lago, and I. Malavolta. 2019. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software* 150 (2019), 77–97. https://doi.org/10.1016/j.jss.2019.01.001 cited By 68.

[9] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. 2017. *Microservices: Yesterday, today, and tomorrow*. 195–216 pages. https://doi.org/10.1007/978-3-319-67425-4_12 cited By 418.

[10] Tore Dyba, Torgeir Dingsoyr, and Geir K. Hanssen. 2007. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 225–234. https://doi.org/10.1109/ESEM.2007.59

[11] Eric Evans and Eric J Evans. 2004. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

[12] Martin Fowler and James Lewis. 2014. Microservices. (2014). http://martinfowler.com/articles/microservices.html

[13] M. Garriga. 2018. Towards a taxonomy of microservices architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10729 LNCS (2018), 203–218. https://doi.org/10.1007/978-3-319-74781-1_15 cited By 25.

[14] M.S. Hamzehloui, S. Sahibuddin, and K. Salah. 2019. A systematic mapping study on microservices. *Advances in Intelligent Systems and Computing* 843 (2019), 1079–1090. https://doi.org/10.1007/978-3-319-99007-1_100 cited By 3.

[15] Gregor Hohpe. 2007. Let's Have a Conversation. *IEEE Internet Computing* 11, 3 (2007), 78–81. https://doi.org/10.1109/MIC.2007.68

[16] D. Jaramillo, D.V. Nguyen, and R. Smart. 2016. Leveraging microservices architecture by using Docker technology. *Conference Proceedings - IEEE SOUTHEASTCON* 2016-July (2016). https://doi.org/10.1109/SECON.2016.7506647 cited By 74.

[17] Barbara Kitchenham. 2004. Procedures for Performing Systematic Reviews. *Keele, UK, Keele Univ.* 33 (08 2004).

[18] Barbara Kitchenham and Stuart Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2 (01 2007).

[19] Seda Kul and Ahmet Sayar. 2021. A Survey of Publish/Subscribe Middleware Systems for Microservice Communication. In *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 781–785. https://doi.org/10.1109/ISMSIT52890.2021.9604746

[20] R. Laigner, Y. Zhou, M.A.V. Salles, Y. Liu, and M. Kalinowski. 2021. Data management in microservices: State of the practice, challenges, and research directions. *Proceedings of the VLDB Endowment* 14, 13 (2021), 3348. https://doi.org/10.14778/3484224.3484232 cited By 1.

[21] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li. 2020. Microservices: Architecture, container, and challenges. *Proceedings - Companion of the 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS-C 2020* (2020), 629–635. https://doi.org/10.1109/QRS-C51114.2020.00107 cited By 1.

[22] A. Messina, R. Rizzo, P. Storniolo, M. Tripiciano, and A. Urso. 2016. The database-is-the-service pattern for microservice architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9832 LNCS (2016), 223–233. https://doi.org/10.1007/978-3-319-43949-5_18 cited By 21.

[23] Sam Newman. 2021. *Building microservices.* " O'Reilly Media, Inc.".

[24] Liam O'Brien, Paulo Merson, and Len Bass. 2007. Quality attributes for service-oriented architectures. In *International Workshop on Systems Development in SOA Environments (SDSOA'07: ICSE Workshops 2007)*. IEEE, 3–3.

[25] C. Pahl and P. Jamshidi. 2016. Microservices: A systematic mapping study. *CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science* 1 (2016), 137–146. https://doi.org/10.5220/0005785501370146 cited By 153.

[26] Jennie Popay, Helen Roberts, Amanda Sowden, Mark Petticrew, Lisa Arai, Mark Rodgers, Nicky Britten, Katrina Roen, Steven Duffy, et al. 2006. Guidance on the conduct of narrative synthesis in systematic reviews. *A product from the ESRC methods programme Version* 1, 1 (2006), b92.

[27] Dan RK Ports, Irene Zhang, Samuel Madden, and Barbara Liskov. 2010. Transactional consistency and automatic management in an application data cache. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*.

[28] Chris Richardson. 2018. *Microservices patterns: with examples in Java*. Simon and Schuster.

[29] N. Santos, C.E. Salgado, F. Morais, M. Melo, S. Silva, R. Martins, M. Pereira, H. Rodrigues, R.J. Machado, N. Ferreira, and M. Pereira. 2019. A logical architecture design method for microservices architectures. *ACM International Conference Proceeding Series* 2 (2019), 145–151. https://doi.org/10.1145/3344948.3344991 cited By 1.

[30] Ekaterina Shmeleva et al. 2020. How Microservices are Changing the Security Landscape. (2020).

[31] A. Sill. 2016. The Design and Architecture of Microservices. *IEEE Cloud Computing* 3, 5 (2016), 76–80. https://doi.org/10.1109/MCC.2016.111 cited By 74.

[32] J. Soldani, D.A. Tamburri, and W.-J. Van Den Heuvel. 2018. The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software* 146 (2018), 215–232. https://doi.org/10.1016/j.jss.2018.09.082 cited By 133.

[33] D. Taibi, V. Lenarduzzi, and C. Pahl. 2018. Architectural patterns for microservices: A systematic mapping study. *CLOSER 2018 - Proceedings of the 8th International*

```
TITLE("microservice?") AND (TITLE("architecture") OR
    (TITLE("data") AND (TITLE("coordination") OR
  TITLE("management")))) AND (LIMIT-TO(PUBSTAGE,
     "final")) AND (LIMIT-TO(DOCTYPE, "cp") OR
LIMIT-TO(DOCTYPE, "ar") OR LIMIT-TO(DOCTYPE, "ch"))
       AND (LIMIT-TO(SUBJAREA, "COMP")) AND
        (LIMIT-TO(LANGUAGE, "English")) AND
  (EXCLUDE(EXACTKEYWORD, "Internet Of Things") OR
   EXCLUDE(EXACTKEYWORD, "Network Architecture"))
```

Figure 3: The initial search string and the refined search string after the filtering process

Conference on Cloud Computing and Services Science 2018-January (2018), 221–232. https://doi.org/10.5220/0006798302210232 cited By 87.

[34] H. Unlu, S. Tenekeci, A. Yildiz, and O. Demirors. 2021. Event Oriented vs Object Oriented Analysis for Microservice Architecture: An Exploratory Case Study. *Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021* (2021), 244–251. https://doi.org/10.1109/SEAA53835.2021.00038 cited By 0.

[35] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. 2015. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*. 583–590. https://doi.org/10.1109/ColumbianCC.2015.7333476

[36] Claes Wohlin. 2014. 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* (London, England, United Kingdom) *(EASE '14)*. Association for Computing Machinery, New York, NY, USA, Article 38, 10 pages. https://doi.org/10.1145/2601248.2601268

[37] E.B.H. Yahia, L. Réveillère, Y.-D. Bromberg, R. Chevalier, and A. Cadot. 2016. Medley: An event-driven lightweight platform for service composition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9671 (2016), 3–20. https://doi.org/10.1007/978-3-319-38791-8_1 cited By.

	Document Type	Title	Authors	Year
1	Article	Performance Analysis of Choreography and Orchestration in Microservices Architecture	Kristianto, H., Zahra, A.	2021
2	Conference Paper	Data management in microservices: State of the practice, challenges, and research directions	Laigner, R., Zhou, Y., Salles, M.A.V., Liu, Y., Kalinowski, M.	2021
3	Conference Paper	Microservices: Architecture, container, and challenges	Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., Li, Z.	2020
4	Conference Paper	A logical architecture design method for microservices architectures	Santos, N., Salgado, C.E., Morais, F., Melo, M., Silva, S., Martins, R., Pereira, M., Rodrigues,...	2019
5	Conference Paper	A Comparative Review of Microservices and Monolithic Architectures	Al-Debagy, O., Martinek, P.	2018
6	Conference Paper	Towards a taxonomy of microservices architectures	Garriga, M.	2018
7	Conference Paper	Leveraging microservices architecture by using Docker technology	Jaramillo, D., Nguyen, D.V., Smart, R.	2016
8	Conference Paper	A systematic mapping study in microservice architecture	Alshuqayran, N., Ali, N., Evans, R.	2016
9	Conference Paper	Microservices: A systematic mapping study	Pahl, C., Jamshidi, P.	2016
10	Article	The pains and gains of microservices: A Systematic grey literature review	Soldani, J., Tamburri, D.A., Van Den Heuvel, W.-J.	2018
11	Article	Microservices Patterns	Richardson, C.	2018
12	Article	Microservices: Yesterday, today, and tomorrow	Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.	2016
13	Conference Paper	Architectural patterns for microservices: A systematic mapping study	Taibi, D., Lenarduzzi, V., Pahl, C.	2018
14	Article	The Design and Architecture of Microservices	Sill, A.	2016
15	Conference Paper	The database-is-the-service pattern for microservice architectures	Messina, A., Rizzo, R., Storniolo, P., Tripiciano, M., Urso, A.	2016
16	Article	Performance Analysis of Microservice Design Patterns	Akbulut, A., Perros, H.G.	2019
17	Conference Paper	Microservices architecture: Challenges and proposed conceptual design	Munaf, R.M., Ahmed, J., Khakwani, F., Rana, T.	2019
18	Article	Microservices from theory to practices	Daya, S.	2015
19	Conference Paper	Event Oriented vs Object Oriented Analysis for Microservice Architecture: An Exploratory Case Study	Unlu, H., Tenekeci, S., Yildiz, A., Demirors, O.	2021
20	Article	Architecting with microservices: A systematic mapping study	Di Francesco, P., Lago, P., Malavolta, I.	2019
21	Conference Paper	A systematic mapping study on microservices	Hamzehloui, M.S., Sahibuddin, S., Salah, K.	2019

Table 2: Selected studies