Migrating towards Microservices: Migration and Architecture Smells

Andrés Carrasco
University of Antwerp
Antwerp, Belgium
andres.carrasco@student.uantwerpen.be

Brent van Bladel
University of Antwerp
Antwerp, Belgium
brent.vanbladel@uantwerpen.be

Serge Demeyer
University of Antwerp
Antwerp, Belgium
serge.demeyer@uantwerpen.be

ABSTRACT

Migrating to microservices is an error-prone process with deep pitfalls resulting in high costs for mistakes. Microservices is a relatively new architectural style, resulting in the lack of general guidelines for migrating monoliths towards microservices. We present 9 common pitfalls in terms of bad smells with their potential solutions. Using these bad smells, pitfalls can be identified and corrected in the migration process.

CCS CONCEPTS

• **Software and its engineering → Extra-functional properties**; *Software architectures*; *Software creation and management*;

KEYWORDS

Architecture Smells, Migration Smells, MicroServices, Literature Study

ACM Reference Format:
Andrés Carrasco, Brent van Bladel, and Serge Demeyer. 2018. Migrating towards Microservices: Migration and Architecture Smells. In *Proceedings of the 2nd International Workshop on Refactoring (IwoR '18), September 4, 2018, Montpellier, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3242163.3242164

1 INTRODUCTION

Microservices is an architectural style for developing an application in independently and automatically deployable services communicating with lightweight mechanisms [23]. The term microservices has become a buzzword nowadays. While some debate is still ongoing on whether microservices is an architectural style itself, or simply a way of doing Service-Oriented Architectures (SOA), there is a concrete distinction on its realization [61].

There is not a one-size-fits-all strategy for microservices, i.e., each solution has a different strategy in place. This plethora of strategies makes the outlining of its characteristics difficult. However, some characteristics are common among microservices, such as the componentization via services, smart endpoints with dumb pipes, and decentralization [23].

IwoR '18, September 4, 2018, Montpellier, France

ACM ISBN 978-1-4503-5974-0/18/09…$15.00
https://doi.org/10.1145/3242163.3242164

The microservices architectural style has grown in popularity for the last few years, due to its potential benefits, such as technology heterogeneity, resilience, scalability, eased deployment, productivity, reusability, and repleaceability among others [51]. Moreover, some research has reported reduced complexity, lower coupling, higher cohesion, simpler integration, better reusability, and performance increase after migrating to a microservices architecture [9, 27]. However, the benefits of adopting a microservices architecture come with the complexities of distributed systems, such as the need for resilience, scaling, and data consistency [43]. Many new technologies have emerged in recent years for dealing with these complexities, such as containerization, automated deployment, and scaling of applications; these technologies are considered enablers for the growth of microservices. Moreover, rapid provisioning, basic monitoring and rapid application deployment are prerequisites for any microservices application [21]. Such requirements are inherently available in the cloud, thus becoming the default home for microservices.

Regardless of the complexities inherent in microservices, a trend on migrating monolithic applications towards microservices architectures has become apparent. Multiple development teams have published their experience migrating to a microservices architecture, including some success stories. However, due to the nature of microservices, following such advices may not be suitable for every strategy. Therefore, publicly available knowledge in this migration trend, such as best practices, success stories, and pitfalls should be collected. The subsequent consolidation of this knowledge in form of migration and architecture smells can provide useful information for teams looking to migrate their applications into microservices.

In this paper, we present 5 new architecture and 4 new migration bad smells found by digesting 58 different sources from the academia and grey literature. The rest of this paper is structured as follows. Section 2 provides an overview of related work. Section 3 presents the research questions, and Section 4 explains our methodology. Section 5 presents the 5 new architecture bad smells, whereas the 4 new migration bad smells are presented in Section 6. Section 7 discusses the threats to validity, and Section 8 concludes.

2 RELATED WORK

Refactoring is part of the Software Engineering Body of Knowledge (SWEBOK). Initially refactoring was intended for restructuring code. However, Stal extended the concept of refactoring to include software architecture refactoring [53]. When refactoring an architecture, the software is changed in a holistic manner for addressing architecture smells.

The microservices architecture is a new architectural style for developing an application in independently and automatically deploying services [23]. As such, development teams wishing to adopt this new architecture must migrate their existing systems. In order to investigate the migration process towards microservices, Taibi et al. surveyed 21 practitioners who adopted the microservices architecture [56]. This survey had the goal of determining the reason for the migration and the issues presented in the process. The reason reported for the migration was driven by technical motivations, such as reduced maintenance, and scalability. Whereas the main issues included: the requirement of experienced developers for the inherent complexity in the architectural style, economical burden due to the required effort for adopting microservices, and the resistance of older developers and architects to new technologies. This showed experts the need for migration patterns with the goal of aiding the migration towards microservices. Taibi et al. subsequently conducted a systematic mapping study for gathering a catalog of principles and patterns for microservices [57]. However, their focus was on gathering architectural patterns for composing a catalog, rather than identifying the potential pitfalls when migrating towards microservices.

To aid practitioners in identifying the potential pitfalls, Taibi and Lenarduzzi defined 11 bad smells related to the microservices architecture [55]. These smells were identified by analyzing 265 bad practices experienced by 72 practitioners. However, their investigation does not include the advice from the state-of-the-art, only from practitioners. Garousi et al. raised the need for including both state-of-the-art and -practice sources when performing research in Software Engineering. When focusing exclusively on research contributions an established body of knowledge is removed [25]. Therefore, a literature survey of best practices for microservices, including both academic and grey literature is needed for identifying more potential pitfalls and their solutions. In this work, we perform such a literature survey.

3 RESEARCH QUESTIONS

RQ1) What architecture and migration bad smells are present in microservices?
- *Motivation* - Identifying the bad smells in the microservices architecture and migration processes would allow individuals to make better decisions when taking apart their monolithic applications, aiding their refactoring efforts.
- *Approach* - Gathering publications from the Academia and grey literature explaining their lessons learned while refactoring monoliths into microservices, allows us to extract and extrapolate common pitfalls. Moreover, the inclusion or exclusion of literature is decided by its contribution to the general proper architectural and migration practices for microservices. Finally, this knowledge is consolidated in the form of architectural and migration bad smells.

RQ2) How can architecture and migration bad smells in microservices be avoided?
- *Motivation* - Identifying potential solutions to the bad smells in the microservices architecture and migration processes could help individuals avoid or fix bad smells in their design.
- *Approach* - From the gathered publications, common best practices and pitfall solutions could be extracted and extrapolated. These findings are presented as potential solutions for the architectural and migration bad smells detected in RQ1.

Table 1: Search Results

	Academic Literature	Grey Literature
refactoring microservices	24	16
migrating microservices	10	8
Total	34	24

4 EXPERIMENTAL SETUP

The sources are gathered using two search engines, namely Google Scholar and Google. Google Scholar allows the searching of academic literature by search terms, whereas the Google engine allows the searching of grey literature on the Internet. All activities regarding the search are performed in a portable version of Firefox. This ensures the traceability of our findings through its browsing history. Moreover, all found Internet sources are downloaded and persisted in HTML and PNG formats, alongside an entry in a Bibtex database. Each source is tagged with the search terms utilized to find the source, as well as the type of literature.

In each of the search engines, the following search terms are utilized for finding relevant literature: *refactoring microservices* and *migrating microservices*. In Google Scholar, the results of both search terms are investigated incrementally by time of publication starting from 2014 after the term was defined in the grey literature by Fowler and Lewis [23]. Each of the search results were investigated and then discriminated by their relevancy through the abstract. Next, for searching grey literature in the Google engine, relevant entries are typically within the first few result pages. Therefore, we stop our search whenever the results fail to include lessons learned or best practices. Lastly, our search was performed at the end of April 2018, any literature published after that date has not been considered.

Our search yielded in total 58 relevant sources; 34 $(58, 62\%)$ academic publications and 24 $(41, 37\%)$ grey literature sources. When searching for *refactoring microservices* we found 24 academic sources and 16 from the grey literature. Moreover, when searching for *migrating microservices* we found 10 academic sources and 8 from the grey literature. In Table 1 our findings are summarized by search term and source. Finally, the smells in the following section were extrapolated from all the relevant sources.

5 ARCHITECTURE SMELLS

5.1 Single Layer Teams

Description: Conway's Law states that a system's design structure is a mirror of the organization communication's structure that designs the system [15]. If a team is divided by layers, e.g. a team for UI, Application Logic, and Database, simple changes can require time and effort to approve between the teams. Therefore, a team may introduce logic in whatever layer the have access to. Although this smell is not unique to microservices, its existence may be

more pronounced in them. Each service ought to be completely independent from the others, however, teams might be tempted to cram their responsibilities on a larger service and make other services dependent to simplify their work. This goes against the independence of microservices and therefore should be avoided.

Solution: Split up cross-functional teams by service. Each team should be composed of the different skills required for the full development of the service. This ensures the independence of each team and consequently the service. However, coordination between the development teams cannot be ignored.

Academic Sources: [1, 5, 9–11, 19, 36, 37, 43, 47]
Grey Literature Sources: [2, 23, 26, 32, 35]

5.2 Greedy Service Container

Description: DevOps practices align with the goals of microservices [5]. For instance, Continuous Delivery allows the automated deployment of a software system on-demand. This automation is considered to be essential for microservices, especially when considering a highly-scalable system. Containerization reduces the time to deployment, therefore, it is considered as one of the main enablers for the microservices architectural style. Not using them adds a whole new level of complexity to the management and orchestration of the application. However, they still require time for their setup. A team looking to cut corners might define a container for all the application to simplify the setup. However, this makes the container heavier and removes the independence of the services.

Solution: Use containers per service. This allows the reduction of the time to deployment, as well as the automation of resource allocation. Moreover, having a container per service aids the scalability of the system, as they can be easily automated for replication. Even though other approaches exist, such as full virtualization with an image, they are generally slow to create and time consuming. Therefore, the general consensus is to utilize containers in the microservices architecture.

Academic Sources: [4–6, 8, 10–12, 18–20, 27, 28, 34, 36, 38, 43, 46, 47, 51, 56–58, 61]
Grey Literature Sources: [7, 14, 16, 30–32, 35, 39, 41, 54]

5.3 Single DevOps Toolchain

Description: Microservices architecture require DevOps toolchains, such as continuous delivery, continuous integration, monitoring, and orchestration. Typically a monolith is split up into multiple services, for which DevOps toolchains must be independently created. However, a time-pressured team might try cut corners and define the same pipeline for all microservices. This would remove the autonomy of the service and remove most if not all of the benefits of the architectural style.

Solution: Have a separate autonomous pipeline for each of the microservices. This means having the complete DevOps toolchain per service, allowing it to be autonomously scaled, monitored, developed, etc., from other services.

Academic Sources: [4–6, 8, 10, 11, 18, 19, 34, 36, 43, 47, 57, 58, 61, 62]
Grey Literature Sources: [7, 13, 21, 35, 39, 45, 49]

5.4 Dismiss Documentation

Description: Documenting any application is generally a good idea. However, this task is often overlooked or not maintained at all, leading to inconsistent and outdated documentation. This smell is valid for any software system, however, in microservices documenting the exposed API is particularly important. With the increased number of independent services, the overview of the system can be easily lost. Moreover, when implementing microservices with multiple teams the cooperation could be hindered without proper documentation.

Solution: Utilize API in-code documentation frameworks, such as Swagger, when possible. Otherwise ensure each team is responsible in maintaining a published API for their service.

Academic Sources: [4, 19, 20, 43]
Grey Literature Sources: [7, 35, 52, 61]

5.5 Grinding Dusty or Coarse Services

Description: A monolith must be broken down into services, for this the proper granularity must be determined. This depends on what the application actually is doing and the wrong granularity can have profound side-effects. If you have a too coarse granularity the benefits of microservices might actually not be worth it, while having a too fine granularity might introduce performance issues due to network latency or other problems.

Solution: The services to be extracted from the monolith can be determined with Domain-Driven Design techniques, such as Bounded Context [43, 59]. Bounded Context presents a good initial approach for defining service granularity. However, the resulting granularity might not be the appropriate for all applications throughout their evolution. Therefore, microservices must be continuously monitored and potentially merging or splitting them if it would aid the overall performance of a system.

Academic Sources: [27, 33, 40, 42, 43]
Grey Literature Sources: [17, 22, 23, 29, 45]

6 MIGRATION SMELLS

6.1 Thinking Microservices Are a Silver Bullet

Description: However promising microservices seem to be, their advantages come with many inherent complexities and challenges at possibly a high cost. Therefore, this architectural style might not be worthwhile for many applications.

Solution: The potential benefits and downfalls of adopting microservices must be thoroughly analyzed. Moreover, alternative solutions without microservices should also be drafted. All potential solutions are then to be compared. The decision of proceeding with the migration, can be taken if after the analysis the microservices

architecture still seems to be the best solution. However, many issues and pitfalls may arise in the process. Therefore, the migration process must be planned with great care. Otherwise, the benefits of microservices can be outweighed by the unexpected costs of mistakes.

Academic Sources: [5, 6, 11, 20, 36, 43, 56]
Grey Literature Sources: [30, 60]

6.2 Rewrite All Services into Microservices at Once

Description: Rewriting any application is not trivial and often not a good idea. Many new faults can be inadvertently introduced, leading to unnecessary risks. Thus, the best approach for migrating monoliths towards microservices is incrementally, i.e., service by service.

Solution: The services of the monolith must be identified by techniques from Domain-Driven Design (DDD), such as the Bounded Context pattern [43, 59]. Through Bounded Context large systems can be modeled according to domains, these domains are considered to be a microservice each. This ensures that a change or addition of functionality is isolated to a domain or in this case a microservice. Moreover, having multiple bounded contexts in a microservice should be avoided. Once the services have been identified, they should be gradually removed from the monolith and transformed into microservices. The migration should be started with a simple service for allowing the development team to gain experience building their first microservice. Even though the extracted service is simple, the monolith has to a certain degree been reduced in complexity. By continuously following this process iteratively, the monolith will gradually be transformed into microservices.

Academic Sources: [5, 12, 43, 56]
Grey Literature Sources: [2, 3, 22, 26, 39, 41, 45, 48, 50, 50]

6.3 Learn as You Go

Description: With the complexities inherent in distributed systems, putting together a team of inexperienced developers in distributed systems is generally not a good idea. Their inexperience may result in higher cost and development time, as they battle with the pitfalls of distributed systems. This smell is not unique to microservices, it can be present on the development of any software. However, the learning curve inherent to microservices might be too steep for inexperienced developers, as it requires knowledge in many different aspects of the development.

Solution: Ensure there are at least a couple of experienced developers with distributed systems in the team. If this is not possible and adopting the microservices strategy is a must, start with a small team developing one simple microservice. Once this team has enough experience, separate them into newly created teams with other developers. In this way, some developers would have acquired some experience and can lead the new teams.

Academic Sources: [5, 6, 43, 57]
Grey Literature Sources: [22, 49]

6.4 Forgetting about the CAP Theorem

Description: Brewer's CAP Theorem states that a distributed data store cannot simultaneously provide more than two of the following guarantees: Consistency, Availability, and Partition tolerance. Due to the nature of microservices, the trade-off scale is tipped towards Availability and Partition tolerance, i.e. we have eventual consistency.

Solution: While there is no solution for ensuring consistency at all times, being aware of it and planning for it is critical in microservices. For instance, working with inconsistent states and implementing ways to detect and correct such inconsistencies is a possible solution. Utilizing Domain-Driven Design practices or Command Query Responsibility Segregation (CQRS) can help mitigate data inconsistency by limiting the shared data between services. However, if consistency at all times is of big importance, the microservices architecture might not be the proper strategy.

Academic Sources: [8, 10, 11, 18, 24, 34, 36, 43, 57, 61]
Grey Literature Sources: [23, 35, 44, 50]

7 THREATS TO VALIDITY

7.1 Conclusion Validity

To ensure the validity of our conclusions to a reasonable extent, we included all the relevant sources that lead to our conclusions for each of the presented bad smells. Moreover, both academic and grey literature is included in the research to ensure a proper heterogeneity.

7.2 Internal Validity

Our search is in no way exhaustive, relevant sources for the research may have been excluded, as the inclusion and exclusion of sources were based on our judgment and experience. However, in hopes for repeatability of results, search engines alongside search terms were reported.

7.3 Construct Validity

Our general inexperience conducting academic investigations is a threat to the validity of our research. However, the RQs and methodology was discussed with experienced researchers in the field. This grant us a reasonable degree of confidence in the validity of our research.

7.4 External Validity

Our research is limited to 58 sources. Although gathering a proper amount of sources was attempted, we cannot assume our findings are a digestion of all best practices, success stories and pitfalls of migrating monoliths towards microservices. However, we included literature from both state-of-the-art and -practice to a reasonable extent. As future work, the usefulness and understandability of the presented bad smells should be evaluated and our conclusions reevaluated.

8 CONCLUSIONS

We presented a brief overview of the apparent trend in both the academia and industry of migrating monolithic architectures toward microservices. Next, we presented 5 new architectural and 4 new migration bad smells by digesting 58 different sources from the academia and grey literature, extending the list of 11 smells proposed by Taibi and Lenarduzzi [55]. Moreover, we presented a possible solution for each of the 9 smells as discussed in the sources. Our contribution provides a foundation for actionable information on the successful refactoring of monolithic applications towards microservices.

REFERENCES

[1] Mohsen Ahmadvand and Amjad Ibrahim. 2016. Requirements Reconciliation for Scalable and Secure Microservice (De)composition. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 68–73. https://doi.org/10.1109/REW.2016.026

[2] Saba Anees. 2016. How to Migrate to Microservices. https://blog.appdynamics.com/product/how-to-migrate-to-microservices. https://blog.appdynamics.com/product/how-to-migrate-to-microservices Accessed 7. May 2018.

[3] Hendry Anwar. 2017. Refactoring Monolith to Microservices. http://us.arvato.com/blog/refactoring-monolith-to-microservices. http://us.arvato.com/blog/refactoring-monolith-to-microservices Accessed 7. May 2018.

[4] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2015. Microservices migration patterns. *Technical report, Tech. Rep. TR-SUTCE-ASE-2015-01, Automated Software Engineering Group* (2015).

[5] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices Architecture Enables DevOps: An Experience Report on Migration to a Cloud-Native Architecture. *IEEE Software* 33, 3 (2016), 42–52.

[6] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In *Advances in Service-Oriented and Cloud Computing*, Antonio Celesti and Philipp Leitner (Eds.). Springer International Publishing, Cham, 201–215. https://link.springer.com/chapter/10.1007%2F978-3-319-33313-7_15

[7] Kyle Brown. 2016. Refactoring for microservices. https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/index.html. https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/index.html Accessed 27. Apr. 2018.

[8] Kyle Brown and Bobby Woolf. 2016. Implementation Patterns for Microservices Architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*. The Hillside Group, 7.

[9] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Stephan T Larsen, and Manuel Mazzara. 2017. *From Monolithic to Microservices: An experience report.* Technical Report. Technical University of Denmark, Örebro University, TU Wien, Danske Bank, and Innopolis University.

[10] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. 2018. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* 17, 4 (Jan. 2018), 29–45. https://doi.org/10.1145/3183628.3183631

[11] Lianping Chen. 2018. Microservices: Architecting for Continuous Delivery and DevOps. *In Proceedings of the IEEE International Conference on Software Architecture (ICSA 2018)* (03 2018).

[12] Fan Chen-Yuan and Ma Shang-Pin. 2017. Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. In *2017 IEEE International Conference on AI Mobile Services (AIMS)*. 109–112. https://doi.org/10.1109/AIMS.2017.23

[13] Matt Chotin. 2018. Top 3 Challenges of Adopting Microservices as Part of Your Cloud Migration. https://devops.com/top-3-challenges-of-adopting-microservices-as-part-of-your-cloud-migration. https://devops.com/top-3-challenges-of-adopting-microservices-as-part-of-your-cloud-migration Accessed 7. May 2018.

[14] Michael Churchman. 2017. Refactoring Your App with Microservices - Rancher Labs. https://rancher.com/refactoring-app-microservices. https://rancher.com/refactoring-app-microservices Accessed 27. Apr. 2018.

[15] Melvin E Conway. 1968. How do committees invent. *Datamation* 14, 4 (1968), 28–31.

[16] Pedro Costa. 2018. Migrating to Microservices and Event-Sourcing: the Dos and Dont's. https://hackernoon.com/migrating-to-microservices-and-event-sourcing-the-dos-and-donts-195153c7487d. https://hackernoon.com/migrating-to-microservices-and-event-sourcing-the-dos-and-donts-195153c7487d Accessed 7. May 2018.

[17] Neil Crawford. 2016. Extracting a Microservice from a Monolith. https://tech.findmypast.com/extracting-a-microservice-from-a-monolith. https://tech.findmypast.com/extracting-a-microservice-from-a-monolith Accessed 7. May 2018.

[18] Nicola Dragoni, Schahram Dustdar, Stephan Thordal Larsen, and Manuel Mazzara. 2017. Microservices: Migration of a Mission Critical System. *CoRR* abs/1704.04173 (2017). arXiv:1704.04173 http://arxiv.org/abs/1704.04173

[19] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*. Springer, 195–216.

[20] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. 2018. Microservices: How To Make Your Application Scale. In *Perspectives of System Informatics*, Alexander K. Petrenko and Andrei Voronkov (Eds.). Springer International Publishing, Cham, 95–104.

[21] Martin Fowler. 2014. MicroservicePrerequisites. https://martinfowler.com/bliki/MicroservicePrerequisites.html. https://martinfowler.com/bliki/MicroservicePrerequisites.html Accessed 16. May 2018.

[22] Martin Fowler. 2015. MonolithFirst. https://martinfowler.com/bliki/MonolithFirst.html. https://martinfowler.com/bliki/MonolithFirst.html Accessed 7. May 2018.

[23] Martin Fowler and James Lewis. 2014. Microservices, a definition of this new architectural term. http://martinfowler.com/articles/microservices.html. Accessed 05.04.2018.

[24] Andrei Furda, Colin Fidge, Olaf Zimmermann, Wayne Kelly, and Alistair Barros. 2017. Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness and Data Consistency. *IEEE Software* PP, 99 (2017), 1–1. https://doi.org/10.1109/MS.2017.440134612

[25] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. 2016. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 26.

[26] Johannes Giani. 2018. 6 Things to Consider for Microservice Migration. https://www.inovex.de/blog/microservice-migration. https://www.inovex.de/blog/microservice-migration Accessed 7. May 2018.

[27] Jean-Philippe Gouigoux and Dalila Tamzalit. 2017. From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 62–65. https://doi.org/10.1109/ICSAW.2017.35

[28] Claudio Guidi, Ivan Lanese, Manuel Mazzara, and Fabrizio Montesi. 2017. *Microservices: A Language-Based Approach*. Springer International Publishing, Cham, 217–225. https://doi.org/10.1007/978-3-319-67425-4_13

[29] Arun Gupta. 2015. Microservice Design Patterns. http://blog.arungupta.me/microservice-design-patterns/. http://blog.arungupta.me/microservice-design-patterns/ Accessed 7. May 2018.

[30] Arun Gupta. 2015. Monolithic to Microservices Refactoring for Java EE Applications. http://blog.arungupta.me/monolithic-microservices-refactoring-javaee-applications. http://blog.arungupta.me/monolithic-microservices-refactoring-javaee-applications Accessed 7. May 2018.

[31] Arun Gupta. 2015. ZooKeeper for Microservice Registration and Discovery. http://blog.arungupta.me/zookeeper-microservice-registration-discovery. http://blog.arungupta.me/zookeeper-microservice-registration-discovery/ Accessed 7. May 2018.

[32] Grzegorz Gwoździ. 2018. How can you refactor a monolithic application into microservices? https://medium.com/@NeotericEU/how-can-you-refactor-a-monolithic-application-into-microservices-2eef8e323840. https://medium.com/@NeotericEU/how-can-you-refactor-a-monolithic-application-into-microservices-2eef8e323840 Accessed 7. May 2018.

[33] Sara Hassan and Rami Bahsoon. 2016. Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap. In *2016 IEEE International Conference on Services Computing (SCC)*. 813–818. https://doi.org/10.1109/SCC.2016.113

[34] Munezero Immaculee Joselyne, Benjamin Kanagwa, and Joseph Balikuddembe. 2017. A framework to Modernize SME Application in Emerging Economies: Microservice Architecture Pattern Approach. In *Microservices 2017*.

[35] Vivek Juneja. 2016. From Monoliths to Microservices: An Architectural Strategy. https://thenewstack.io/from-monolith-to-microservices. https://thenewstack.io/from-monolith-to-microservices Accessed 7. May 2018.

[36] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. 2018. Challenges When Moving from Monolith to Microservice Architecture. In *Current Trends in Web Engineering*, Irene Garrigós and Manuel Wimmer (Eds.). Springer International Publishing, Cham, 32–47.

[37] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. 2016. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. *CoRR* abs/1605.03175 (2016). arXiv:1605.03175 http://arxiv.org/abs/1605.03175

[38] David S. Linthicum. 2016. Practical Use of Microservices in Moving Workloads to the Cloud. *IEEE Cloud Computing* 3, 5 (Sept 2016), 6–9. https://doi.org/10.1109/MCC.2016.114

[39] Abhijit Mandal. 2018. Migrating to Microservices. https://dzone.com/articles/migrating-to-microservices. https://dzone.com/articles/migrating-to-microservices Accessed 7. May 2018.

[40] Genc Mazlami, Jürgen Cito, and Philipp Leitner. 2017. Extraction of Microservices from Monolithic Software Architectures. In *Web Services (ICWS), 2017 IEEE International Conference on*. IEEE, 524–531.

[41] Ben Morris. 2015. Refactoring large monoliths to microservices: strategies, risks and pragmatic reality. http://www.ben-morris.com/refactoring-large-monoliths-to-microservices-strategies-risks-and-pragmatic-reality. http://www.ben-morris.com/refactoring-large-monoliths-to-microservices-strategies-risks-and-pragmatic-reality Accessed 7. May 2018.

[42] Ola Mustafa, Jorge Marx Gómez, Mohamad Hamed, and Hergen Pargmann. 2018. GranMicro: A Black-Box Based Approach for Optimizing Microservices Based Applications. In *From Science to Society*, Benoît Otjacques, Patrik Hitzelberger, Stefan Naumann, and Volker Wohlgemuth (Eds.). Springer International Publishing, Cham, 283–294.

[43] Sam Newman. 2015. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

[44] Pramod Nikam. 2017. How will you design, break and migrate database matching to microservices? http://techprehension.com/2017/12/31/how-will-you-design-break-and-migrate-database-matching-to-microservices. http://techprehension.com/2017/12/31/how-will-you-design-break-and-migrate-database-matching-to-microservices Accessed 7. May 2018.

[45] Brice Nkengsa. 2016. Migrating from a Monolith to a Microservices Architecture. https://medium.com/@briceicle/migrating-from-a-monolith-to-a-microservices-architecture-99cecf8af366. https://medium.com/@briceicle/migrating-from-a-monolith-to-a-microservices-architecture-99cecf8af366 Accessed 7. May 2018.

[46] Claus Pahl and Pooyan Jamshidi. 2016. Microservices: A Systematic Mapping Study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science*. 137–146.

[47] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2017. Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 38–45. https://doi.org/10.1109/ICSAW.2017.32

[48] Chris Richardson. 2016. Refactoring a Monolith into Microservices. https://www.nginx.com/blog/refactoring-a-monolith-into-microservices. https://www.nginx.com/blog/refactoring-a-monolith-into-microservices Accessed 27. Apr. 2018.

[49] Curtis Rissi. 2015. Thoughts on Migrating to a Microservices Architecture. http://www.boringgeek.com/thoughts-on-migrating-to-a-microservices-architecture. http://www.boringgeek.com/thoughts-on-migrating-to-a-microservices-architecture Accessed 7. May 2018.

[50] Rafael Romão. 2017. Microservices Architecture As A Large-Scale Refactoring Tool. http://blog.avenuecode.com/microservices-architecture-as-a-large-scale-refactoring-tool. http://blog.avenuecode.com/microservices-architecture-as-a-large-scale-refactoring-tool Accessed 7. May 2018.

[51] Tasneem Salah, Mohamed Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. 2016. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. 318–325. https://doi.org/10.1109/ICITST.2016.7856721

[52] Ryan Scheuermann. 2017. 9 Tips for a Painless Microservices Migration. http://engineering.invisionapp.com/post/painless-microservices-migration. http://engineering.invisionapp.com/post/painless-microservices-migration Accessed 7. May 2018.

[53] Michael Stal. 2007. Software Architecture Refactoring. In *Tutorial in The International Conference on Object Oriented Programming, Systems, Languages and Applications*.

[54] Chris Stetson. 2017. How To 'Refactor' Monolithic Applications into Microservices. http://www.datacenterknowledge.com/archives/2017/06/05/refactor-monolithic-applications-microservices. http://www.datacenterknowledge.com/archives/2017/06/05/refactor-monolithic-applications-microservices Accessed 7. May 2018.

[55] Davide Taibi and Valentina Lenarduzzi. 2018. On the Definition of Microservices Bad Smells. *IEEE Software* vol 35 (05 2018).

[56] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing* 4, 5 (September 2017), 22–32. https://doi.org/10.1109/MCC.2017.4250931

[57] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2018. Architectural Patterns for Microservices: A Systematic Mapping Study. In *8th International Conference on Cloud*.

[58] MVLN Venugopal. 2017. Containerized Microservices architecture. *International Journal Of Engineering And Computer Science* 6, 11 (2017), 23199–23208.

[59] Vaughn Vernon. 2013. *Implementing domain-driven design*. Addison-Wesley.

[60] James Ward. 2015. Refactoring to Microservices. https://www.jamesward.com/2015/05/26/refactoring-to-microservices. https://www.jamesward.com/2015/05/26/refactoring-to-microservices Accessed 7. May 2018.

[61] Olaf Zimmermann. 2017. Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development* 32, 3 (01 Jul 2017), 301–310. https://doi.org/10.1007/s00450-016-0337-0

[62] Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. 2018. On the Modernization of ExplorViz towards a Microservice Architecture. In *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*, Vol. Online Proceedings for Scientific Conferences and Workshops. CEUR Workshop Proceedings, Ulm, Germany. http://eprints.uni-kiel.de/42119/