Back to the Future: From Microservice to Monolith

Ruoyu Su[1] · Xiaozhou Li[1] · Davide Taibi[1,2]

[1] University of Oulu, Oulu, Finland

[2] Tampere University, Tampere, Finland

✉ ruoyu.su@student.oulu.fi; xiaozhou.li@oulu.fi; davide.taibi@oulu.fi

Abstract. Recently the trend of companies switching from microservice back to monolith has increased, leading to intense debate in the industry. We conduct a multivocal literature review, to investigate reasons for the phenomenon and key aspects to pay attention to during the switching back and analyze the opinions of other practitioners. The results pave the way for further research and provide guidance for industrial companies switching from microservice back to monolith.

Keywords. *Microservice · Monolith · Multivocal literature review · Practitioner · Trend*

1 Introduction

Microservice has become an important style of architecture due to its decomposable and decentralized nature [12]. In recent years, microservice has become increasingly popular, especially in industry [10]. Big companies like Netflix, Amazon, and Spotify have also adopted microservice architecture and more and more companies are following this trend and migrating their systems to microservice [11]. They want to utilize the benefits of microservice, such as independent development, deployment, and scaling to help the system solve the problem at hand, improve the quality of the system, or facilitate software maintenance [6, 13]. Meanwhile, issues with microservices, e.g., architecture degradation, coupling in organization structures, etc. are also noticed in academia when approaches towards mitigation are also proposed [1, 2, 3].

However, while several companies have made significant improvements in velocity and team independence, others did not achieve the benefits expected after migrating to microservices. With the increasing number of companies migrating from monolith to microservice, the drawbacks of microservice architectures are enhanced [7, 9].

Recently, there is a trend towards switching from microservice back to monolith. One example is Amazon Prime Video, which is one of the world's largest streaming services, serving millions of customers worldwide. It claims that the switch from a distributed microservice architecture to a monolithic application helps achieve greater scale, resilience and lower costs [5]. This report caused a heated debate among practitioners, after all, even big companies like Amazon have made rollbacks from microservice.

This paper aims to investigate the cases that switch from microservice back to the monolith with a multivocal literature review. Based on our goals, we define the following research questions: ***RQ***$_1$ *What are the reasons for switching back to monolith?* ***RQ***$_2$ *What are the key aspects to pay attention* *to during the switching back?* ***RQ***$_3$ *What are the opinions of the other practitioners regarding such "switch-back"?.*

The results show that there are four cases that switch from microservice back to monolith of the company: Istio [1], Amazon Prime Video[2], Segment [3] and InVision [4]. The five main reasons that switch back to monolith are: cost, complexity, scalability, performance and organization. During the process, there are six key aspects needed to be aware of: (1) stop developing more services, (2) consolidate and test paths, (3) unify data storage, (4) implement message bus principle, (5) give up diverse techniques and (6) learn to use modular design principles. Opinions of other practitioners are mixed, but most still believe that the decision to switch back to monolith requires careful consideration of the actual system situation and principles.

2 Methodology

Here we aim to understand the state of the arts regarding the methods, techniques, and tools facilitating the shift from microservice back to the monolith, as well as the practitioners' opinions and advice towards such practices. To such an end, we conducted a multivocal literature review (MLR) based on the guidelines defined by [4]. An MLR is a combination of two parts, including 1) a Systematic Literature Review (SLR) on the academic literature (white) published in journals or conferences, and 2) that on the grey literature, e.g., blog posts, social media posts, and videos [4]. Herein, we used the search query *(microservice* OR micro-service* OR "micro service*") AND monolith* AND (back OR return* OR refactor* OR rearchitect* OR migrat* OR revert* OR re-architect*)* in the both white and grey literature search. From the search results, we aimed to select the articles (white and grey) that propose *change back from microservice to monolith* and provide factual evidence and/or practical advice related to real industrial cases. By following the traditional SLR process, we obtained only one academic paper from four sources (Scopus, IEEE, ACM, and Web of Science). Furthermore, by searching on Google, Reddit, Quora, and Stack Overflow, we obtained 19 useful articles and 9 extra from snowballing [5]. We extracted the data to answer the RQs by adapting and merging the categories provided by the selected articles. The complete paper, with the list of selected articles, is available at [8].

3 Results

According to the review, there are four cases that switch from microservice back to monolith: Istio [SA2], Amazon Prime Video [SA5], Segment [SA14] and InVision [SA20]. Among them, the case of Amazon Prime Video is the most discussed with nine articles. These sources were first published in 2018 and did not attract much attention at first. With the case of Segment in 2020, some discussion among practitioners was generated. The case of Amazon Prime Video in 2023 brought the heated discussion of switching back to monolith.

[1]https://istio.io/

[2]https://www.primevideo.com/offers/nonprimehomepage/ref=dv_web_force_root

[3]https://segment.com/

[4]https://www.invisionapp.com/

[5]All selected articles are listed in the Appendix which is saved in Arxiv.org. The link will be shared when the paper is accepted.

3.1 RQ1: What are the reasons for switching back to monolith?

From the review, we identified five main reasons that cases switch back to monolith.

Cost. Cost is the most common reason why companies switch from microservice back to monolith. In Istio, marginal costs and operational costs are high due to the microservice architecture [SA1] [SA2] [SA3] [SA4]. Amazon Prime Video has the most serious cost problem in four cases. It uses of serverless components resulted in the overall cost of all the building blocks not allowing for the large-scale acceptance of the solution, and the way the video frames (images) are passed between the different components is expensive for the large number of Tier-1 calls to S3 buckets. [SA5] [SA6] [SA7] [SA8] [SA9] [SA10] [SA11] [SA12] [SA13]. The cost-benefit of Prime Video's switch back to monolith is also the most significant. According to official reports, moving the service from microservice back to a monolith reduced the infrastructure cost by over 90% [SA5]. Segment also has a cost problem. The operational costs of supporting microservices are unaffordable for it [SA15] [SA18]. Finally, in the case of InVision, it makes the comment that " Microservices Also Have a Dollars-And-Cents Cost" [SA20] [SA21]. The service runs on the server, talks to the database, reports on metrics and generates log entries, all of which have a very real dollar and cent cost. Therefore, it is evident that microservices incur expenses, particularly when accounting for the necessity of maintaining redundancy for establishing a highly available system [SA20].

Complexity. Complexity is one of the most important reasons why companies switch from microservice back to monolith. In the case of Istio, microservices lead to greater complexity. Firstly, different planes in Istio are written in different programming languages [SA2]. Secondly, different teams are responsible for different services separately, but the reality is that this approach makes for increased complexity and has a bad impact on user usability, rather than making it simpler for the development team to manage [SA2] [SA4]. In addition, all components in Istio's control plane are always released in the same version at the same time, while the functionality of the decoupled version of microservices complicates it [SA2]. Finally, Istio has only limited isolation, making full isolation of microservices difficult [SA1] [SA2] [SA3]. However, some other factors lead to greater complexity in the case Segment except the nature of the architecture itself: managing multiple repositories and divergence of shared libraries [SA14] [SA16] [SA19]. Initially, each destination was divided into separate services but the same repository, but this caused frustration and inefficiency. A single broken test affected all destinations, and deploying changes required fixing unrelated tests [SA14] [SA16] [SA19]. Breaking out the code for each destination into separate repositories increased complexity and maintenance effort. To ease the burden of developing and maintaining these codebases, it created shared libraries to make common transforms and functionality [SA14]. The complexity of the problem InVision encountered is very similar to Segment. As time went on, InVision had more repositories, more programming languages, more databases, more monitoring dashboards, etc. that became too much for the development team to bear [SA20] [SA21].

Scalability. The advantage of microservice scalability becomes a disadvantage in these cases. The control plane costs in Istio are mainly determined by the individual feature (XDS) [SA1] [SA2] [SA3]. In contrast, all other functions have marginal costs and the value of isolation is very small [SA2] [SA4]. Amazon Prime Video met a scaling bottleneck. Due to the microservices architecture, it hit a hard scaling limit with around 5% of the expected load, resulting from the orchestration management implemented using AWS Step Functions [SA5] [SA6] [SA7] [SA10] [SA11] [SA13]. Different from Prime Video, Segment scaling challenges are the reason for the automated scaling configuration. As the number of destinations grows, managing and scaling each microservice becomes a significant operational overhead [SA14]. Each service has a specific load pattern that requires manual scaling to cope with unexpected spikes [SA17]. Tuning the auto-scaling configuration becomes more challenging due to the different resource requirements of each service [SA14] [SA18].

Performance. Performance is the most important reason for Segment's switch back to monolith. The most serious is the head of line blocking. The microservices cause head-of-line blocking, causes delays in all destinations [SA14] [SA16]. This affects the timeliness of event delivery and also customer satisfaction [SA17]. In addition, high complexity is also a factor that causes system performance to decline. The high complexity caused by microservice puts development teams in a difficult situation where the benefits of modularity and autonomy become burdensome, slowing them down and reducing productivity, which leads to poorer performance [SA14].

Organization. The suboptimal management of teams is also a headache, especially in Istio and InVision. In the case of Istio, although microservices allow different teams to manage services individually, in practice this creates a mess for development teams who want simpler management [SA2].In contrast, InVision has the most serious people problem. InVision had a legacy team with fewer people but more repositories, databases, programming languages, etc. [SA20]. As time went on, the benefits of Conway's Law became a burden on the legacy team because of this unsuitable 'size', so it was necessary to merge microservices back into a monolith [SA20] [SA21].

3.2 RQ2: What are the key aspects to pay attention to during the switching back?

We analyzed six key aspects during the process that switching from microservice back to monolith.

Stop developing more services. This means new services cannot be introduced. Switching from microservice back to monolith requires an existing microservice to be used as the "center" of the future monolith to host the new functionality [SA22]. All other services will eventually be merged into this center. However, if we still make new services after switching back to the monolith, this could lead to the whole system getting messy.

Consolidate and test paths. Microservices sometimes have a single, coherent flow between multiple systems. Consolidation paths are necessary when systems are merged from microservices to a monolith [SA22]. After the merger, it is also important to test the path to ensure that the new system runs smoothly. This process ensures that the new monolithic architecture works properly and meets all requirements [SA23].

Unify data storage. Shanea proposed there are two main options: move the data to a single database or keep the data separate [SA23]. The former can reduce costs and improve performance while reducing the complexity of the system. The latter can help maintain the autonomy and isolation of separate components, while still moving towards a more homogeneous structure [SA23]. The choice of data storage is critical and development teams need to choose carefully based on actual requirements.

Implement the message bus principle. Implementing a message bus, like Kafka, can be a layer of indirection while transitioning [SA23]. This strategy enables a gradual consolidation of microservice into monolith without any interruptions to the current system. By utilizing a message bus, smooth communication between various components is ensured, facilitating the decomposition and recombination of services as required.

Give up diverse techniques. The feature of microservices is that different services can use different languages, frameworks, etc. And after the system has switched back to the monolith, these diversifications need to be given up. For example, most systems should use no more than two back-end languages at any one time [SA22].

Learn to use modular design principles. We need to maintain a modular design when switching back to monolith. Modular design allows the code to be organized into distinct modules with clear boundaries, which promotes separation of concerns and maintainability [SA22]. The modular design also allows systems to gain the flexibility and modularity benefits of microservices with the simplicity and ease of use of monoliths [SA23].

3.3 RQ3: What are the opinions of the other practitioners regarding such "switch-back"?

Other practitioners have mixed opinions about this 'switch-back' behavior. Some argue that this way is correct. They think microservice is not the "utopian application architecture" [SA3]. David Heinemeier Hansson scoffs microservice is a zombie architecture [SA7]. Monoliths do have an advantage over microservices because they are easier to code, scale, deploy, test, and deal with cross-domain issues [SA24] [SA29]. Some still don't agree with it. They believe that microservices are still one of the most popular architectures. Angel posts monoliths are not the solution, and organizations need to think better and support proactively communication channels that can supply the gaps between teams [SA25]. However, most practitioners still believe that the need to switch back to a monolithic architecture requires consideration of the actual system situation and principles. Such a switch back would require an assessment of whether monolithic is really the best fit for the company's team size, structure, skills, and operational capabilities [SA23] [SA27]. Moreover, most of the disadvantages of microservices are well-known [SA26], so Itiel believes that the recommendation for architecture depends on the type of project [SA28].

4 Conclusions

There are discussions among practitioners regarding switching from microservice back to monolith when, especially, some companies have already taken action. Though it is still too early to claim it as a trend, the practitioners' opinions are certainly worth noticing. In this work, we performed a preliminary investigation on the reasons for companies to decide to switch back to monolith and key aspects to pay attention to during the process. At the same time, we analyzed the opinions of other practitioners regarding this trend. By systematically addressing 29 white and grey literature in the field, our findings reveal cost is the most important reason why companies switch from microservice back to monolith. Furthermore, complexity, scalability, performance, and organization are also the main reasons for this trend. During this process of switching back, we summarized there are six key aspects worth noticing: (1) stop developing more services, (2) consolidate and test paths, (3) unify data storage, (4) implement message bus principle, (5) give up diverse techniques and (6) learn to use modular design principles. The results show that practitioners have started seriously considering the benefits and motivation of switching back. Though academic studies and industrial applications of microservices are obviously in their prime, the pains of microservices and the benefits of adopting monolith can still provide insights into their improvement. In future studies, we shall further investigate the in-depth opinions of the industry on this topic via surveys and interviews. We shall also conduct comparative case studies on the performances of microservice and reversed monolith systems.

References

[1] Amr S Abdelfattah and Tomas Cerny. Roadmap to reasoning in microservice systems: A rapid review. *Applied Sciences*, 13(3):1838, 2023.

[2] Tomas Cerny, Amr S Abdelfattah, Vincent Bushong, Abdullah Al Maruf, and Davide Taibi. Microservice architecture reconstruction and visualization techniques: A review. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 39–48. IEEE, 2022.

[3] Dario Amoroso d'Aragona, Xiaoxhou Li, Tomas Cerny, Andrea Janes, Valentina Lenarduzzi, and Davide Taibi. One microservice per developer: Is this the trend in oss? *arXiv preprint arXiv:2308.02843*, 2023.

[4] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and software technology*, 106:101–121, 2019.

[5] Marcin Kolny. Scaling up the prime video audio/video monitoring service and reducing costs by 90%. https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90, 2023.

[6] Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, and Davide Taibi. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, 169:110710, 2020.

[7] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232, 2018.

[8] Ruoyu Su, Xiaozhou Li, and Davide Taibi. Back to the future: From microservice to monolith. `https://doi.org/10.48550/arXiv.2308.15281`, 2023.

[9] Davide Taibi and Valentina Lenarduzzi. On the definition of microservice bad smells. *IEEE software*, 35(3):56–62, 2018.

[10] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5):22–32, 2017.

[11] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Architectural patterns for microservices: a systematic mapping study. In *CLOSER 2018*. SciTePress, 2018.

[12] Davide Taibi, Valentina Lenarduzzi, Claus Pahl, and Andrea Janes. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, pages 1–5, 2017.

[13] Davide Taibi and Kari Systä. From monolithic systems to microservices: A decomposition framework based on process mining. In *2019 CLOSER*. SciTePress, 2019.