JENSEN yrkeshögskola
TRÄNING FÖR VERKLIGHETEN

From Monolith to Microservice

A review of the transition process

Examensarbete

DevOps 22

VT 2024

Espen Lever

Handledare: Ulf Eriksson

Abstract

This essay investigates the process of migrating a monolithic architecture environment to a microservice architecture, and the challenges and benefits associated with it. It further investigates the role that Kubernetes plays in such a migration. A microservice architecture is a codebase that has divided its different features into isolated units, often virtual machines or containers. In order to answer the questions a survey has been conducted, as well as a literary review. The challenges found in literature are divided into technical and organizational challenges, and consist of challenges in refactoring the existing codebase, being able to properly divide the services and databases, and setting clear boundaries. The organizational challenges consist mostly of communication and education challenges, as well as the difficulties associated with restructuring teams. The survey findings support the literary findings, however with a heavier emphasis on the organizational challenges. The conclusion drawn regarding that discrepancy is that Kubernetes solves the technical challenges in microservices to a significant degree, making the migration process easier, despite its steep learning curve.

Table of Contents

1. Introduction

1.1. Background

In the modern day and age of software development, microservices have become increasingly popular as a software architecture. Thanks to cloud computing and cloud deployment becoming almost an industry standard, scalability and an increasingly rapid development cycle started becoming more and more important in development as a whole. This would explain the emergence of microservice architectures becoming more prevalent and companies switching from their monolithic architectures in order to fulfill their needs, something that will be explored more in depth below. But first we must answer, what is a monolith and what are microservices? The following sections will give a short definition and background to what the different terms mean that will be used throughout the essay.

1.1.1. Monolithic architecture

A monolithic architecture refers to a product, project or other type of software or code base that is deployed as a single unit, a monolith. There are generally three types of monoliths; the single process monolith, the distributed monolith and the third-party systems. Single process monoliths refer to a structure where all code is packaged into a single process, often outputting their data into a single database. These are the typical monoliths that cause issues, and are the reason for the term monolith existing in the first place. The distributed monolith is a system that is distributed over multiple services, but still needs to be deployed together. This architecture usually emerges when a transition to microservices is done incorrectly, resulting in an awkward mix between monoliths and microservices that gain all the disadvantages of both architectures. The final monolith, third-party systems, are the products you buy and use along with your own products, often developed by a third party. They act as a black box, where the processes are unknown to the user, and cannot be changed. (Newman, 2019)

Similarly to Newman, when mentioning monoliths henceforth, it will refer to the single process monolith.

1.1.2. Microservice architecture

A microservice architecture refers to the product or business domain being split up into components, or microservices, that can be deployed individually. They communicate with each other via networks, creating a distributed system. Microservices have special attributes that will be explained here. (Newman, 2019)

The first attribute is independent deployability. This means that each service can be changed individually and deployed without affecting the rest of the system. For this to work the services need to have explicit and well-defined contracts and limits between the services, in order to prevent issues due to overlapping. (Newman, 2019)

The second attribute is that all microservices are modeled around a business domain. This means that the architecture that we design is made in such a way to complement the business domain and its processes. This means that we don't divide our services in frontend, backend and database, but instead combine the three layers and utilize them in every service, which can for example be a payment processing service, administration system service etc. (Newman, 2019)

The final attribute is that every microservice should own their own data. Databases are not supposed to be shared across services. This makes it easier to decide what data can be communicated and what cannot. It also reduces the risk of overlapping services, ensuring proper independent deployment in the future. (Newman, 2019)

1.1.3. Kubernetes

One of the most popular tools used nowadays for managing and orchestrating services in a system is called Kubernetes, or K8s for short. Kubernetes was developed by Google and in 2014 it became open-source. The need for a program such as Kubernetes came when we started running our services through containers. Containers are a form of isolated virtual machine that is easily replicable and managed, due to its configurations being stored as images. Kubernetes is the system solution used for a myriad of management tools for your services, such as load balancing, self healing and replicating, version rollouts and rollbacks and many other things. It is commonly used in cloud storage solutions such as Microsoft

Azure or Amazon Web Services, but can be used on local physical servers as well. Its popularity over the years makes it an interesting tool to investigate further in this essay. (*Kubernetes*, 2023)

1.2. Purpose

This article will examine the challenges and benefits of migrating the code base of an organization from a monolithic architecture to a microservice architecture. Additionally it will examine the role of Kubernetes within microservice deployment and management, and the impact that Kubernetes has during the migration process. This will naturally lead into cloud technology as well.

1.3. Research Questions

Given the purpose of this article described, the following three research questions have been defined:

1. **What challenges typically arise when migrating from a monolithic architecture to a microservices-based architecture?**
2. **What are the key benefits associated with transitioning from monolithic to microservices architecture?**
3. **Specifically exploring Kubernetes, how does it contribute to orchestrating and managing microservices during the migration process?**

2. Methodology

In the methodology section I will describe the methods I will use in order to answer the research questions stated above. The purpose is to give the reader an insight into what methods were chosen to gather the data required to support a well founded answer and discussion.

2.1. Literature

The first methodology used to gather data will be literary reviews. Given the rise of microservices that happened a few years ago, the process of migrating from monolithic to microservices has been completed and documented already. This gives me an opportunity to use those previously documented changes in order to lay a foundation upon which I will base my analysis. This literature will consist of a combination of books and scientific articles found related to the subject.

A literature review is the method of summarizing what currently is known about the chosen field of study, and can make up parts or an entire article. The way a literature review is conducted is by gathering different sources of relevant studies, and reviewing their methods and findings in regards to your own study's research question. (Lewis-Beck et al., 2004)

An advantage of using a literary review as part of my methodology is that I do not have to reinvent the wheel, so to say. By using the theories and processes of people that have experienced and documented their findings, I can increase the credibility of my findings. Another advantage is that I can focus my paper on consolidating the findings of multiple papers and create a more nuanced analysis.

Lastly, using literary reviews increases the quality of my second methodology, the survey. By gaining some insight into the process I am reviewing before writing the survey questions, I can make sure that the questions I include in my survey are relevant and provide me with the data I need to answer my research question. (*Literature Review*, n.d.)

There are however some limitations with literary reviews as well. First of all the data found may not be relevant anymore. In an industry so fast-paced such as software development, technologies advance and replace each other within months sometimes. This makes it difficult to make sure that the literature used is based on relevant information and does not mislead the reader from the current technology available.

Another limitation is the fact that all written text is subject to bias from the author. It is difficult to keep a written text purely objective, especially when there is a discussion in the text. This means that all sources will need to be read critically in order to verify that they are credible and present their data and arguments properly. (*What Is a Literature Review?*, 2022)

2.2. Survey

The second methodology I will utilize during this essay is a survey. The reason for choosing the survey method is to gather some primary data sources regarding my research questions and get the opinions of people who have experienced an architecture migration first-hand. The survey will be written and published on surveyplanet.com, a website dedicated to making and publishing surveys, and distributing it across forums focused on DevOps and Kubernetes in order to try to reach the target audience.

Looking at the literature, according to Lewis-Beck et al. (2004), an internet survey is simply a survey distributed over the internet. It can be served to the participant in multiple ways, but the method chosen here is a web survey, served on a webpage that consolidates the results.

There are many advantages of using a web-based survey. First of all is the cost (Lewis-Beck et al., 2004). By using a service such as SurveyPlanet I can create a survey for free, barring from a select few and optional features. The majority of popular forums allow you to sign up for free as well, thus making it easy to access the communities that can provide the insight I need. That also brings us to the second advantage of web-based surveys, namely ease of distribution (Lewis-Beck et al., 2004). Given the immense online community associated with software development, it is not difficult to find a place online where people with the knowledge and experience I am looking for will reside. This way I can get participants even without being a member of the industry and having personal connections yet.

There are however some limitations with a web-based survey as well. First of all is anonymity and credibility. By posting a link in an open forum, virtually anyone can click that link and fill in the survey, regardless of qualifications. According to Lewis-Beck et. al. (2004), this has been tracked to not be a big concern based on data compared to alternative survey forms. Nonetheless, this can hurt the credibility of the survey results, if some precautions are not taken. First of all the survey will require an email address in order to participate, taking away part of the anonymity and deterring potential saboteurs from completing the survey. It might however also deter serious participants, but the increased credibility outweighs that risk in my opinion.

The second limitation is the volume of information uploaded on the internet. By posting a link on a forum it will most likely be visible to the readers for the first 24 hours at maximum. This results in fewer answers unless the survey is posted repeatedly, which could be considered spam and cause a ban or similar restriction to my account. Because of this I will only upload the survey twice on each forum maximum, in order to compromise the risk and reward.

3. Result

This section contains the results gathered when trying to answer my research questions. The section is structured as follows: First a presentation of the results from the literary review, split in three parts to answer the three research questions. Secondly, a presentation of the results of the survey that I conducted.

3.1. Literary Review

The main literature that will be considered is the book *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith* by Sam Newman (2019). This book is the second book Sam Newman has written about microservices, and particularly describes and tackles the process of migration to a microservice architecture. Using this book and supporting scientific articles that observed and analyzed the migration process, I have been able to find a theoretical foundation to build my analysis upon.

3.1.1. The Challenges in Migrating

The challenges of migrating your database can be divided into two categories, namely technical and organizational challenges. Technical challenges look at the difficulties in changing the codebase itself, and its surrounding tools and technologies. Organizational challenges look at the difficulties the company and its employees can face, mostly on a structural or workflow level.

3.1.1.1. Technical Challenges

When looking at a migration from a monolithic architecture to a microservice system, oftentimes the original **code base** will have to be **refactored**. This can be a slow and difficult process, and needs to be planned and executed very carefully. According to Kalske et. al. (2018), the best way to refactor a code base is to do it in small parts, tackling one feature at a time. In addition to that, new features should be designed to fit the new architecture, not append to the old monolith. That way the transition is slowly moving to the codebase being fully transitioned.

There are many factors to consider when refactoring a codebase. First of all, rewriting the code increases the risk of introducing new bugs. When trying to apply old functionality to a new design, problems tend to come up because of the original workflow being disrupted. These problems can be mitigated by having proper test coverage for your codebase. Having a proper testing process when changing code can not only help ensure that the individual components work as they should, but also if they interact with each other properly. (Kalske et al., 2018)

Second of all a monolithic code base might not be designed according to a business domain, but instead be modeled in a more technical categorization. This makes the refactoring process more difficult to resolve, since the functionality might be correct, but behaves improperly in regard to the new system that is being implemented. For example, can the data transformation logic of a component be correct, but the in- and output can be wrong with the new communication system. In order to fix this the new system must be designed properly and clear boundaries must be set for each feature. Solving this problem takes us to the next major challenge in our transition - separating services. (Newman, 2019)

When separating the services, it is important to define **clear boundaries** between the different responsibility areas you want to cover. The goal is to make each service cover a very specific feature in the codebase, and have them all designed according to your business domain, as stated above. It is however also important to not be too specific with the separation, due to added complexity in the new architecture. The more services your new codebase contains, the more connections need to be formed between them and the more intricate your network becomes.

The recommended way to go about it is to start with the easiest and most obvious features that naturally are more separated from the rest of the codebase and work inward from there. The goal is to be able to easily change a feature in the future by changing just one service, instead of multiple connected ones. (Kalske et al., 2018)

The final challenge faced when migrating the codebase is **migrating, or breaking up, the database**. A database that supports a monolithic architecture is usually quite large and contains multiple schemas that connect to each other in many ways, and has most likely evolved over time to become more and more intricate. When switching over to a microservice system, the instinct might be to use the same database for the newly created services. This is however not advised. (Newman, 2019)

Using the same database for multiple services can lead to data being changed by unintended services, causing potential errors for certain processes. It also poses a security risk due to services having access to all data in the database, even potentially sensitive data. Additionally it couples the services together again, basically undoing many of the benefits of having a microservice architecture. (Kalske et al., 2018)

This does not mean that splitting the database will be an easy task. There are many challenges in splitting a database. Firstly the schemas will need to be redesigned to fit not only their intended services, but also the foreign keys and API calls to and from other services. Additionally the integration process will be a challenge, due to databases in production changing continuously. (Newman, 2019)

3.1.1.2. Organizational Challenges

On top of the technical challenges explained above, there are also organizational challenges when performing a migration of this magnitude. First of all is the structure of the organization. When working in a monolithic environment it is common for teams to consist of people with the same role or expertise, for example a frontend team consisting of graphical designers and JavaScript developers, a backend team with mostly system engineers and developers, and a database team and operations or server team. This is not a structure that works cohesively with the microservice architecture. With the new system design, one small change in one service will still require the attention of multiple, if not all teams. Therefore a restructuring is necessary where teams are created based on focus areas or services. This will create a more streamlined workflow for the team, with less barriers for communication and disruption. (Kalske et al., 2018)

Making such a big change in structure does however come with its own challenges. Without a thorough plan with regular checkups and feedback opportunities, a restructuring is doomed to fail. First of all you need to explain to the teams why the changes are happening. A team that is ordered to perform a certain task without motivation will seldomly perform that task well, which is especially true when mixing people together with different experiences, perspectives and languages.

Secondly the teams have to be able to give their own input in the changes. Regular feedback from the teams is really powerful, especially in the early stages of the change. Since the members of the teams most likely have more hands-on experience with the product and task ahead of them, their insight is generally quite useful in changing the processes and workflows for the better.

Lastly it is very important to document the process of the restructuring and migration. Even if the current structure might suit your needs right now, this might, and probably will change in the future. Having the experiences of this change documented will help your company avoid mistakes made earlier and handle the restructuring more efficiently. (Newman, 2019)

3.1.2. The Benefits of Migration

Now that we have gone through the challenges of migrating from a monolithic architecture to microservices, we have to ask ourselves the question of why we would go through all this trouble, or more accurately what are the payoffs of going through such a big change.

The first benefit of the migration is **improved team autonomy**. According to Newman, many organizations have shown that creating smaller organizational teams can create stronger bonds, reduce bureaucracy and increase productivity and quality (Newman, 2019). By increasing the sense of ownership of the product to the team and giving them more power in decision making, a team will generally try to use that to make the best product they can.

The second benefit to switching to a microservice architecture is the **reduced time to market**. By restructuring the teams and the services to focus on one specific feature, and having that feature isolated from the rest of the codebase, the team can focus completely on the functionality of that feature, with minimal consideration needed to the rest of the product. This increases the development time, and along with the attribute of independent deployability, makes the pipeline from idea to deployed product much shorter. (Newman, 2019)

The third benefit from the migration is **more cost-effective scaling**. Because of the independent deployability of the microservices, scaling can be done more easily and accurately than in a monolith. Due to the services often being deployed as isolated virtual machines or containers, the horizontal scaling of each individual service can be done by simply adding more replicas of the relevant services into your production environment. This also helps keeping costs down by being able to quickly respond to the traffic loads for each service and scale accordingly, saving money on deployment costs. (Newman, 2019)

The final benefit to using microservices over a monolith is that it will **reduce downtime** of the production environment. Finding bugs or having unexpected crashes happening is inevitable, and in a monolithic environment will mean that the entire product will have to be taken offline. With a microservice environment however, a bug can be more easily located and fixed, and can often be done without having to take down the entire environment. There is also a possibility to roll back an update much quicker, or with the help of horizontal scaling, start another container to replace a failing one, in case of unexpected crashes. (Newman, 2019)

3.1.3. Kubernetes as a Migration Tool

The last question that will be answered in the literary review portion is what role Kubernetes plays in the migration from monoliths to microservices. In order to answer that I will explain some of the features that Kubernetes provides, to further discuss them in relation to the benefits and challenges in the discussion below.

3.1.3.1. Container Orchestration

First of all is **container orchestration**. Kubernetes provides a lightweight platform where containers can easily be deployed with a high level of granularity. The deployment feature lets you manage the amount of computing power and memory each container is allowed, what versions you want to run, as well as providing labels that can be used in further categorization of your services (*Kubernetes*, 2023). Additionally, the containers can be divided by namespaces, which further isolates the container from the rest of the environment (*Kubernetes*, 2023).

3.1.3.2. Scaling

Secondly is the **scaling** that the deployments provide. You can specify the amount of replicas of each service that you deploy, thereby allowing the service to scale according to the workload demanded of them (*Kubernetes*, 2023). When deploying your environment on a Kubernetes cluster in a cloud solution, like for example Microsoft Azure, that cloud provider usually offers an auto-scaling feature as well, that uses unutilized computing power to scale your service in case of increased traffic (*Microsoft*, 2024).

3.1.3.3. Self-healing

Additionally to scaling, the deployment feature also allows for **container self-healing**. Due to the declaration of the desired state of your service deployment, Kubernetes will periodically inspect the deployment and automatically restart containers if they are down for some reason. (*Kubernetes*, 2023)

3.1.3.4. Networking

Kubernetes uses pod-specific IP-addresses for each container to be able to communicate with each other with the help of services. This allows you to configure the networking of each container with very high granularity and make sure that the pods have the right amount of network exposure. The networking also allows for ingresses to be set up in order to be accessed from the internet. (*Kubernetes*, 2023)

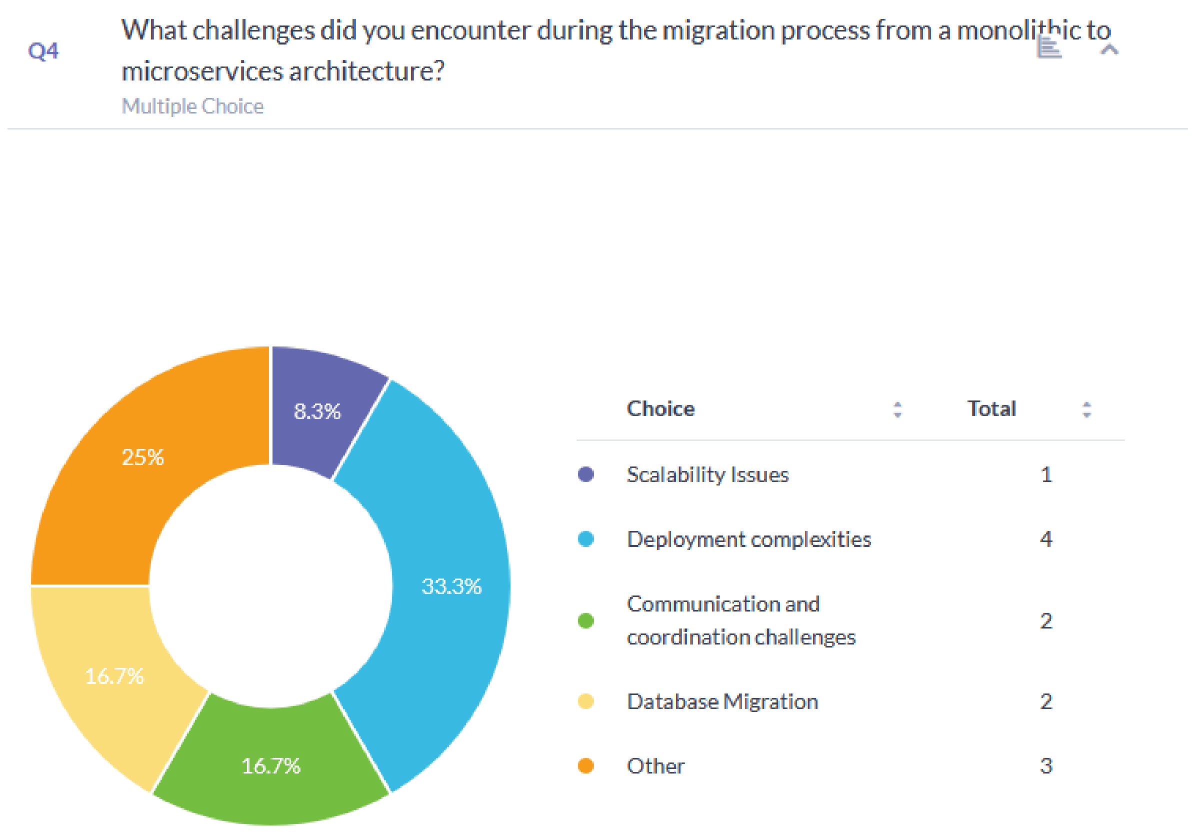
3.1.3.5. Updates and Rollbacks

Finally, deployments allow for rolling updates and rollbacks to be done on a container level as well. This makes error handling much easier and reduces downtime. (*Kubernetes*, 2023) Additionally, with the use of the Kubernetes package manager Helm, you are able to further optimize this update and rollback loop, including keeping a version history log (*Helm*, n.d.).

3.2. Survey

As stated in the methodology, the survey that I have conducted is a web-based survey. The target demographic for this survey was Devops engineers, developers or other IT professionals that have participated in a migration from monolithic to microservice architecture. The survey was hosted on surveyplanet.com, and was distributed to the Kubernetes and Devops subreddits on Reddit, the Kubernetes Community Forum and the Oracle Developer Community Forum. The survey questions are presented in the appendix below. This section will describe the results of the survey. The survey had four participants.

The participants consisted of two developers and two devops engineers, with an experience span of 8 to 20 years in the industry.



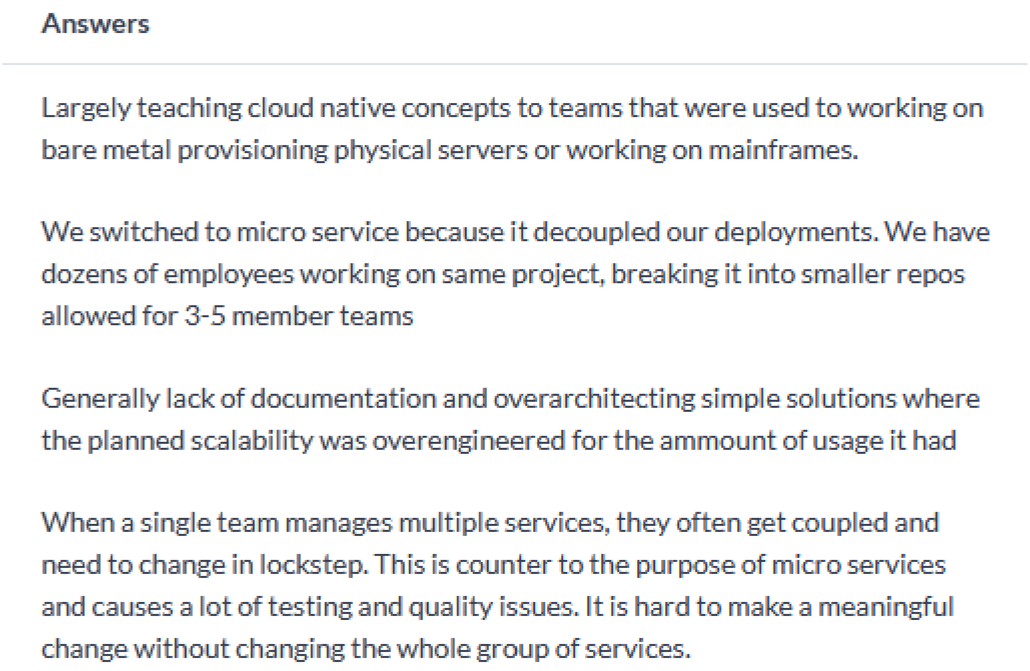
Q4 What challenges did you encounter during the migration process from a monolithic to microservices architecture?

Multiple Choice

	Choice	Total
●	Scalability Issues	1
●	Deployment complexities	4
●	Communication and coordination challenges	2
●	Database Migration	2
●	Other	3

Figure 1.

In figure 1 we can see that all participants encountered issues related to deployment of services, 2 had issues with database migration, 2 with communication and coordination challenges, and 1 with scalability. Additionally, 3 gave other answers that consisted of the following: “Educating the team to support the new platform”, “Bad documentation in dependencies” and “Integration testing between services”. Figure 2 below shows the elaborations given about the challenges.



Answers

Largely teaching cloud native concepts to teams that were used to working on bare metal provisioning physical servers or working on mainframes.

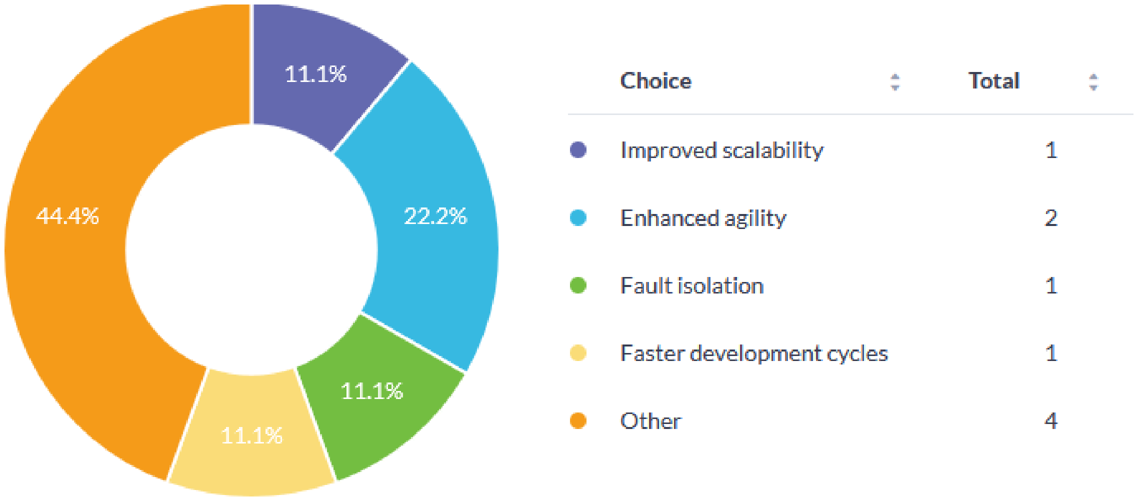
We switched to micro service because it decoupled our deployments. We have dozens of employees working on same project, breaking it into smaller repos allowed for 3-5 member teams

Generally lack of documentation and overarchitecting simple solutions where the planned scalability was overengineered for the ammount of usage it had

When a single team manages multiple services, they often get coupled and need to change in lockstep. This is counter to the purpose of micro services and causes a lot of testing and quality issues. It is hard to make a meaningful change without changing the whole group of services.

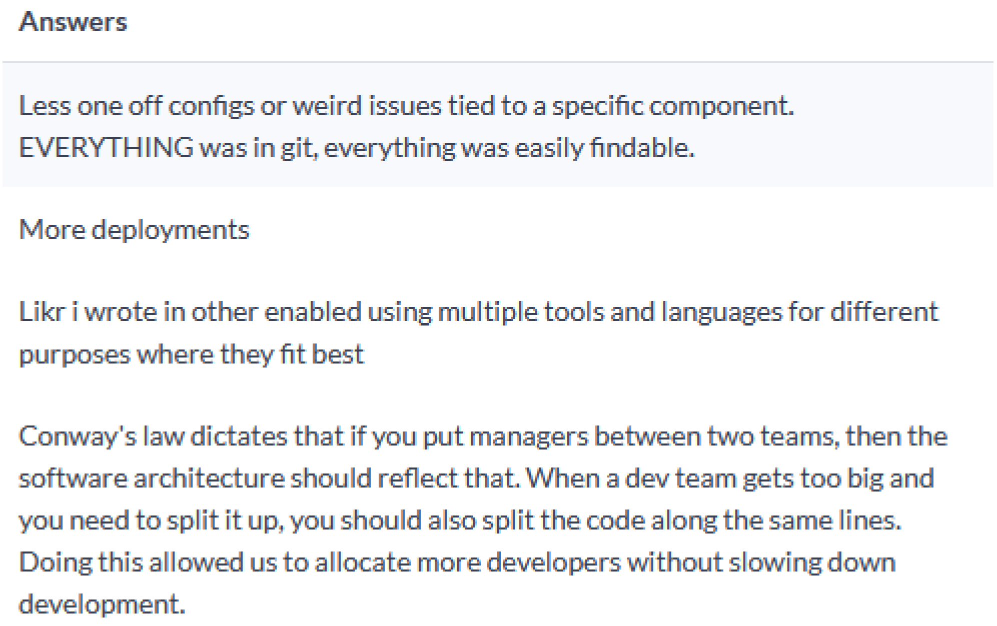
Figure 2.

The next question asked about the benefits of the microservice architecture, and is displayed in figure 3. The answers are more split here, with 1 answer per choice, except for enhanced agility, and all participants writing a write-in option. The write-in options were the following: “Less "pets", easier to support once migrated.”, “Less arguing about deployment schedule. Met deliverables faster without fighting”, “Code and language independence so it enables of using best language for best purpose”, “Decoupling development teams from each other.” The elaborations to their answers are displayed in Figure 4.



	Choice	Total
●	Improved scalability	1
●	Enhanced agility	2
●	Fault isolation	1
●	Faster development cycles	1
●	Other	4

Figure 3.



Answers

Less one off configs or weird issues tied to a specific component. EVERYTHING was in git, everything was easily findable.

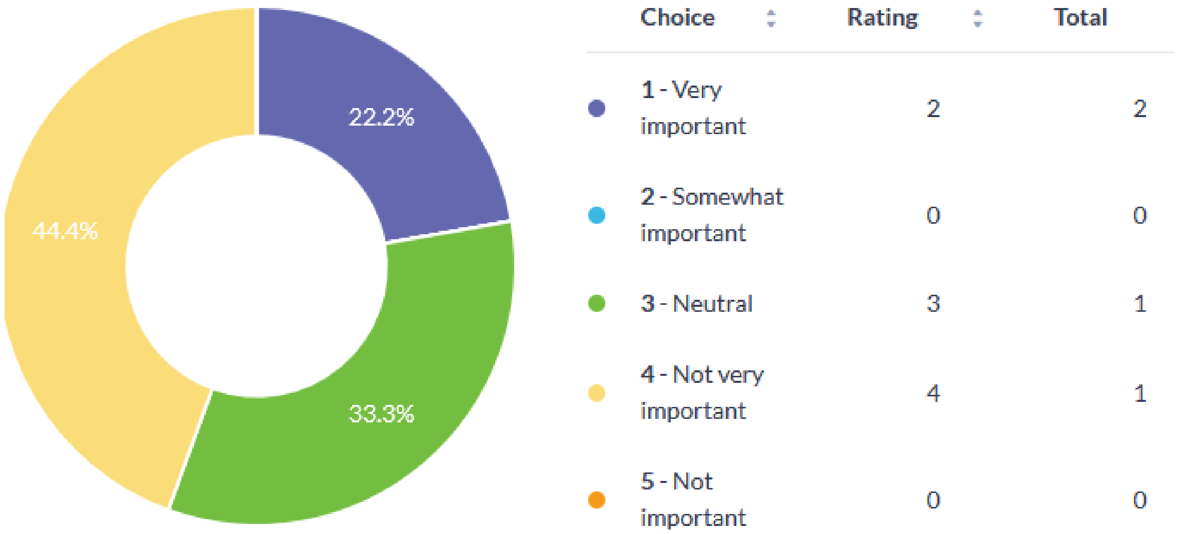
More deployments

Likr i wrote in other enabled using multiple tools and languages for different purposes where they fit best

Conway's law dictates that if you put managers between two teams, then the software architecture should reflect that. When a dev team gets too big and you need to split it up, you should also split the code along the same lines. Doing this allowed us to allocate more developers without slowing down development.

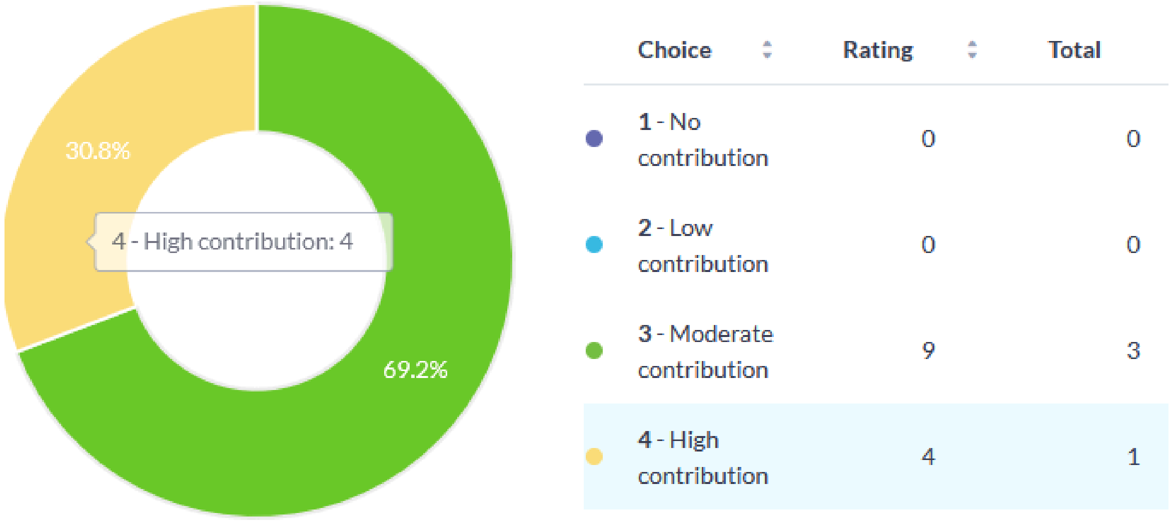
Figure 4.

Three of the participants use Kubernetes when managing their microservices, while one does not. When asked how important they consider Kubernetes for orchestrating and managing their microservices, 2 answered very important, 1 answered neutral and 1 not very important (Figure 5).When asking how much they think Kubernetes contributes to resource optimization in the deployment and scaling of microservices, 3 answered moderate and 1 answered high (Figure 6).



	Choice	Rating	Total
●	1 - Very important	2	2
●	2 - Somewhat important	0	0
●	3 - Neutral	3	1
●	4 - Not very important	4	1
●	5 - Not important	0	0

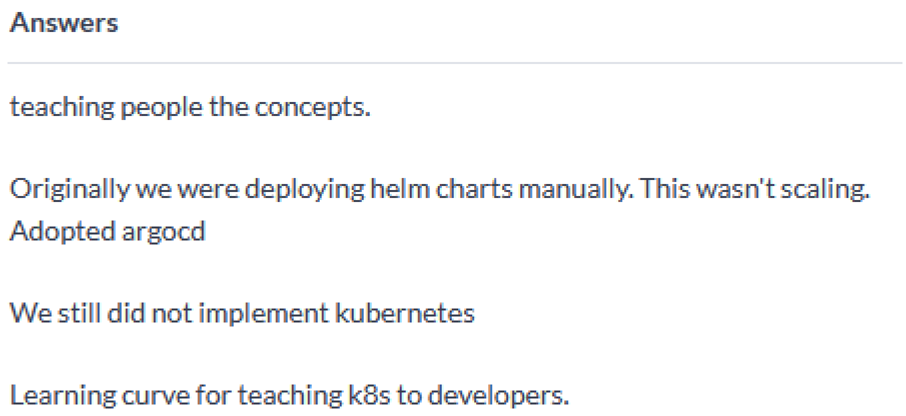
Figure 5.



	Choice	Rating	Total
●	1 - No contribution	0	0
●	2 - Low contribution	0	0
●	3 - Moderate contribution	9	3
●	4 - High contribution	4	1

Figure 6.

The summary that the participants gave about the challenges of implementing Kubernetes were mostly about learning the concepts of Kubernetes (Figure 7), and the main benefits that people have enjoyed are the features, third party support, and community (Figure 8).



Answers

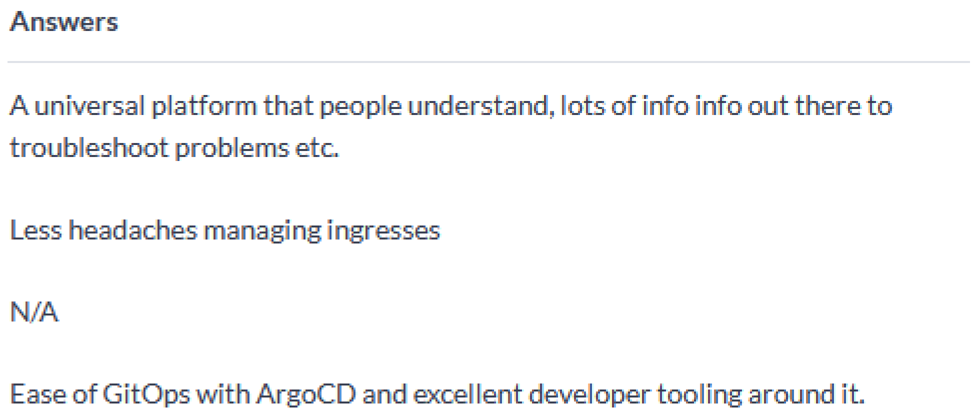
teaching people the concepts.

Originally we were deploying helm charts manually. This wasn't scaling. Adopted argocd

We still did not implement kubernetes

Learning curve for teaching k8s to developers.

Figure 7.



Answers

A universal platform that people understand, lots of info info out there to troubleshoot problems etc.

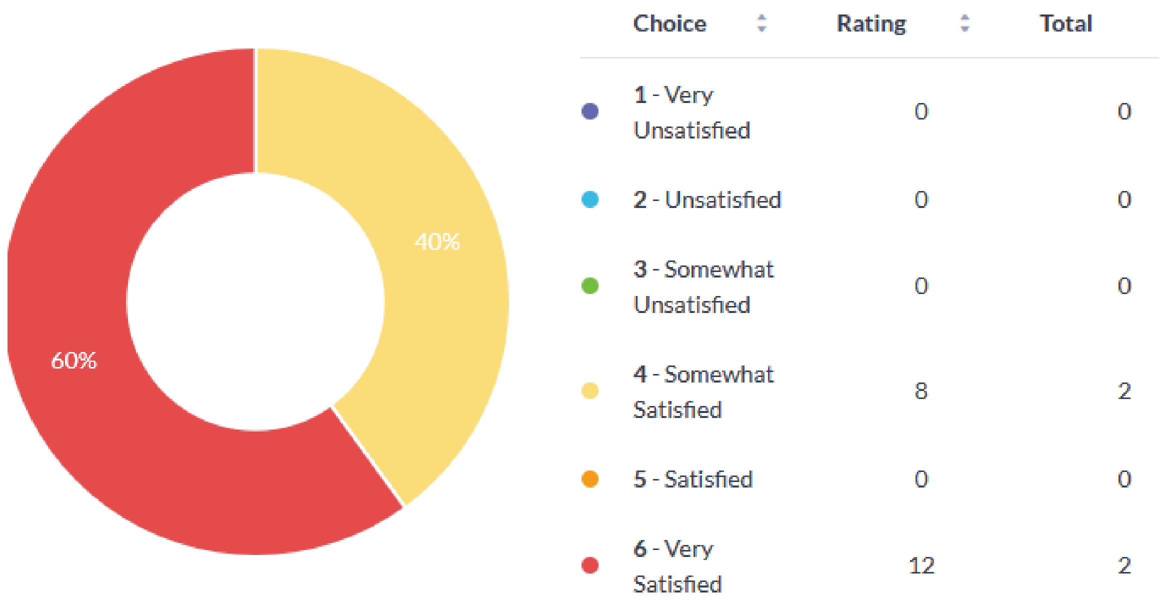
Less headaches managing ingresses

N/A

Ease of GitOps with ArgoCD and excellent developer tooling around it.

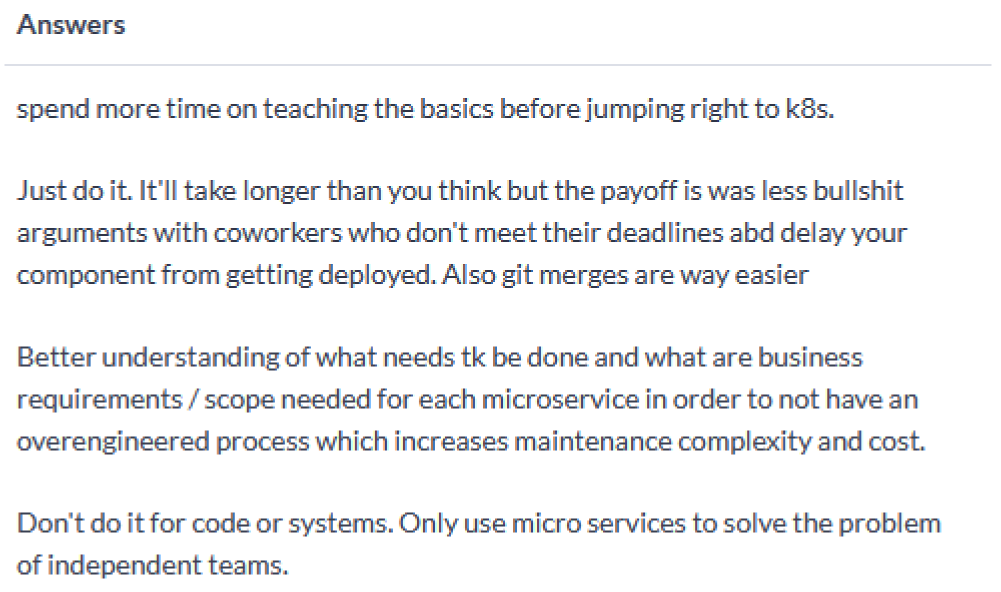
Figure 8.

Lastly, the survey asked about the participants' satisfaction with the outcomes of the migration, of which 2 answered somewhat satisfied and 2 very satisfied (Figure 9). After this they were asked about suggestions for improving the migration process, and most answers focused on preparations before the migration, and mainly on a team/organizational level. (Figure 10.)



Choice	Rating	Total
● 1 - Very Unsatisfied	0	0
● 2 - Unsatisfied	0	0
● 3 - Somewhat Unsatisfied	0	0
● 4 - Somewhat Satisfied	8	2
● 5 - Satisfied	0	0
● 6 - Very Satisfied	12	2

Figure 9.



Answers

spend more time on teaching the basics before jumping right to k8s.

Just do it. It'll take longer than you think but the payoff is was less bullshit arguments with coworkers who don't meet their deadlines abd delay your component from getting deployed. Also git merges are way easier

Better understanding of what needs tk be done and what are business requirements / scope needed for each microservice in order to not have an overengineered process which increases maintenance complexity and cost.

Don't do it for code or systems. Only use micro services to solve the problem of independent teams.

Figure 10.

4. Discussion

With the result being described above with two different methodologies that reflect the theoretical and practical perspective of the migration process, it is now time to compare the two perspectives and examine the similarities and differences in the results. Further this section will discuss whether Kubernetes plays a significant role in migrating to a microservice architecture.

First, let's examine the similarities and differences between the literature and survey. When looking at the elaborated answers from the survey participants, we can see that most of them are focused on organizational challenges. This may be because the survey only had "Communication and coordination challenges" as an answer option, since I focused more on the technical aspect in my survey at first. However none of the elaborated answers tackled the technical side, which leads me to believe that organizational difficulties are more prevalent in a real-world scenario than I had anticipated. This is however not an abnormal outcome, due to many of the technical challenges being solved by technologies such as Kubernetes, which will be elaborated on further below.

The organizational challenges mentioned in the survey consisted of preparation and education. As we found in the literature, in order to change the company in such a big way, a lot of communication needs to happen. I do not think this is a problem that can be eliminated, changes will always bring problems and difficulty in the beginning, but management needs to make sure that they are doing their best to minimize these issues. Documentation was also mentioned as missing during the migration, which reinforces the fact that a well-documented codebase is important at all times. This is partly supported in the literature as well, even though I interpreted the importance of documentation there to refer to the migration for future endeavors, not historical documentation to support the refactoring of the database. Lastly the importance of boundaries between services was mentioned as well, where a participant experienced trouble with coupled services and them creating issues in testing and quality. This has been emphasized by Sam Newman multiple times in his book, and seems to be the primary takeaway from this investigation on the technical side.

On the benefits side we still see many similarities between literature and practice. The first one is more deployments, which coincides with the concept of faster time to market. This was mentioned by two separate participants as a write-in option, and as the elaboration, which makes me believe that this is one of the main benefits from a developers perspective. Another answer was the ability to use multiple tools and languages, allowing the team to choose the tools that work best for the task. This is also reflected in literature and makes sense from personal experience as well. The concept of autonomous teams is also reflected, with one participant mentioning an increase in productivity when decreasing the team size.

So what role does Kubernetes play in all this? As seen by the survey, the majority of teams used Kubernetes as their service management platform, and are satisfied with that choice. The main downside of using Kubernetes seems to be the steep learning curve, which is an aspect I have experienced in my personal life as well. However, it seems that the benefits outweigh the learning curve. I interpret the lack of focus on the technical challenges from the participants to be because of their technology solving a lot of those problems for them. When looking at the risks and benefits of a microservice architecture and comparing them to the main features of Kubernetes, there is a significant overlap. For example is the connection and communication difficulty between services solved thanks to the natively integrated networking of Kubernetes, along with the breadth of customization that can be done with it. Benefits such as less downtime and economic scaling are also features that Kubernetes openly advertises, with its self-healing features and autoscaling technology. This, along with my personal experience of working with Kubernetes in school and in my internship makes me reach the conclusion that Kubernetes plays a significant role in the migration process and beyond when working with microservices.

One important reflection about this essay is the survey and its low participation rate. This can be explained by two factors. Firstly is the target demographic. The demographic I was looking for in this instance was very specific, and the more specific your demographic, the more difficult it is to reach said demographic. So maybe a survey was not the right method to use to gather first hand data and an interview would have been more appropriate, which leads to the second factor, access to the industry. As a student I have not had the chance to talk to a lot of people in the software development industry, much less people who have such specific experiences. By choosing the survey method I was able to cast a wider net and find people I do not know personally, and let them answer the questions in their own time, which made my survey less sensitive to time zones and other geographical challenges.

5. References

Deployments. (2023, September 6). Kubernetes. Retrieved March 11, 2024, from https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Helm Rollback. (n.d.). Helm. Retrieved March 11, 2024, from https://helm.sh/docs/helm/helm_rollback/

Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges When Moving from Monolith to Microservice Architecture. *Current Trends in Web Engineering*, *10544*, 32-47.

Kubernetes Overview. (2023, September 19). Kubernetes. Retrieved March 8, 2024, from https://kubernetes.io/docs/concepts/overview/

Lewis-Beck, M. S., Bryman, A., & Futing Liao, T. (2004). *The SAGE Encyclopedia of Social Science Research Methods* (Vol. 1). Sage.

LibGuides: Literature Reviews: What is a literature review? (2022, October 26). LibGuides at University of Texas at Austin. Retrieved March 6, 2024, from https://guides.lib.utexas.edu/literaturereviews

Literature Review. (n.d.). University of Illinois Springfield. Retrieved March 6, 2024, from https://www.uis.edu/learning-hub/writing-resources/handouts/learning-hub/literature-review

Namespaces. (2023, June 29). Kubernetes. Retrieved March 11, 2024, from https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Newman, S. (2019). *zas*. O'Reilly Media, Incorporated.

Services, Load Balancing, and Networking. (2023, October 19). Kubernetes. Retrieved March 11, 2024, from https://kubernetes.io/docs/concepts/services-networking/

Use the cluster autoscaler in Azure Kubernetes Service (AKS) - Azure Kubernetes Service. (2024, January 11). Microsoft Learn. Retrieved March 11, 2024, from https://learn.microsoft.com/en-us/azure/aks/cluster-autoscaler?tabs=azure-cli

6. Appendix

6.1. Survey Questions:

1. **What is your role in the software development lifecycle?**
    - Developer
    - Operations/DevOps Engineer
    - Architect
    - Manager
    - Other
2. **How many years of experience do you have in software developments or related fields?**
    - Sliding scale from 0 - 50 years.
3. **Have you been involved in a migration from a monolithic to microservices architecture in an organization?**
    - Yes
    - No
4. **What challenges did you encounter during the migration process from a monolithic to microservices architecture?**
    - Scalability issues
    - Deployment complexities
    - Communication and coordination challenges
    - Database migration
    - Other
5. **Can you give a short elaboration about the challenges you experienced?**
    - Free answer.
6. **In your opinion, what are the key benefits of adopting a microservices architecture in a cloud environment?**
    - Improved scalability
    - Enhanced agility
    - Fault isolation
    - Faster development cycles

- Other

7. **Can you give a short elaboration about the benefits you experienced?**
    - Free answer.
8. **Is your organization using Kubernetes for managing their microservices?**
    - Yes
    - No
9. **How important do you consider Kubernetes in orchestrating and managing microservices?**
    - Very important
    - Somewhat important
    - Neutral
    - Not very important
    - Not important
10. **To what extent do you believe Kubernetes contributes to resource optimization in the deployment and scaling of microservices?**
    - No contribution
    - Low contribution
    - Moderate contribution
    - High contribution
11. **Can you give a short summary of the challenges your organization faced when implementing Kubernetes?**
    - Free answer.
12. **Can you give a short summary of the benefits you have enjoyed thanks to Kubernetes?**
    - Free answer.
13. **How satisfied are you with the outcomes of the migration from a monolithic to microservices architecture?**
    - Very unsatisfied
    - Unsatisfied
    - Somewhat unsatisfied
    - Somewhat satisfied

- Satisfied
- Very satisfied

14. **Based on your experience, what suggestions do you have for improving the process of transitioning from a monolithic to microservices architecture, with or without Kubernetes?**
    - Free answer.
15. **I understand and agree that my answers will be used in the writing of this essay.**
    - Yes
    - Yes, but keep my information anonymous.
16. **I would like to receive a copy of the essay to my email address when it is finished.**
    - Yes
    - No