Aalto University
School of Business

Microservices:
Considerations before implementation

Bachelor's Thesis
Heini Könönen
xx.xx.20xx
Program

Approved in the Department of Information and Service Economy xx.xx.20xx and awarded the grade

Author	Heini Könönen
Title of thesis	Title of thesis
Degree	Bachelor's
Degree programme	Information and Service Management
Thesis advisor(s)	Bragge
Year of approval	yyyy
Number of pages	20
Language	English

Abstract:

Microservices is a relatively recent pattern in software architecture, but it is in wide use already and is used to develop the flagship products of some of the world's most popular services, such as Netflix and Spotify. The pattern is evolving organically from the development practices so there is relatively little formal academic research on it. This thesis explains the benefits and potential risks of moving from a monolithic software architecture to a microservices architecture, in a manner understandable to non-developers.

In a nutshell, microservices architecture breaks up large, monolithic software projects into small, discrete and modular 'services'. The services can be developed separately, sometimes by different teams, and can be deployed independently of each other. Some key benefits are the possibility of using specialized tech stacks for different services, smaller and easier to understand codebases, improved productivity and collaboration between development teams and more robust and flexible systems. Microservices based software is also better suited to cloud computing, which can reduce infrastructure costs.

However, this does not mean that all software projects should be microservices. There are situations in which a monolithic pattern has advantages. First and foremost, applying microservices effectively needs a certain degree of expertise, and since the trend is recent, qualified developers can be hard to find. It is also arguably easier to manage a monolithic architecture with a small team, at least in the beginning. Secondly, microservices architecture is said to move the complexity from the code base to the infrastructure. Microservices can be needlessly complex and expensive if the project isn't meant to scale for many users or is a prototype or proof-of-concept

Table of Contents

1. Introduction

Microservices is a style of software architecture that breaks down large software into smaller, independent and discrete 'services' that are loosely coupled. The services have low or no interdependence, and they communicate with each other, usually through application programming interfaces, APIs. It is different from the monolithic architecture, which is an older way of structuring software. In monolithic systems, the parts of the system are not separated, and the system uses only one or a few servers and databases. Decision makers often have to choose if they want to build their system with monolithic or microservices architecture or if they want to change their system from one to the other.

The pace of development and implementation of new concepts in the IT industry is very fast. Microservices as a term and architectural pattern are relatively new, having emerged within the past decade, but trends evolve quickly and are also adopted quickly in the IT world. Many giant corporations have adopted microservices architecture, and indeed, their efforts are defining the term. For example, Netflix, IBM, Amazon, Google, Spotify, Twitter and many others claim to have had success in implementing microservices architecture and tout its benefits (SaM Solutions, 2017).

There are several case studies and other works touting its benefits but there are not many scientific case studies or articles on the subject. However, there are still some big companies, such as Etsy, that have remained in monolithic architecture and they claim that it's still working for them (SaM Solutions, 2017).

Microservices was not 'invented' in the traditional sense. It is an evolving collection of patterns, principles and rules of thumb that several companies use for software development, especially in solutions meant to eventually scale for very large user bases. As such there is no definitive book of knowledge or methodology (Newman, 2015). As I will elaborate on in later sections, it is also not a binary choice, i.e. adopting all microservices techniques or none. Different problems and types of software projects can expect vastly different results from the microservices architecture. In later sections, I explore perspectives from experts on how to decide whether the microservices approach is wise for a project and try to debunk the notion

that the monolithic architecture is always bad, and wrong for projects. Now is a good time to start to research the medium and longer-term effects of adopting this technology, and what types of risks to prepare for.

1.1 Research objectives and research questions

Modularity and microservices architecture is currently one of the biggest trends in modern software architecture. Many big companies, including Netflix and Spotify have switched to using microservices and the number of studies has grown exponentially in the past few years (Scopus 2018, SaM Solutions, 2017). However, there is still relatively little scientific research and literature on the topic, especially on its impact in the long term. Thus, managerial and non-technical professionals often find it hard to understand whether it makes sense for their software project to be refactored from a monolithic architecture into a microservices architecture. This is also a problem when planning new software projects.

There is a common fallacy that all monolithic software should be converted into the microservices approach, and this can often confuse inexperienced decision makers into wasting time and resources. When combined with the fact that there is no definitive set of guidelines on the subject, this can cause paralysis and confusion when trying to make the decision. As several experts point out, executing microservices architecture takes detailed know-how and the desire to implement it is often not matched with sufficient technical expertise on the subject to do it well. As I will elaborate on later, in certain cases, monolithic architecture actually could be a better solution, at least to start with.

The central research question of my thesis is therefore: 'What is microservices architecture in software, and what should businesses consider before and during implementation?'. This thesis is targeted towards people looking to gain a non-technical understanding of the approach, the benefits it may bring and the risks to prepare for.

1.2 Scope of research

I believe that the microservices trend is sufficiently advanced that everyone working in software architecture and the broader IT industry should know about it and make a conscious decision whether or not to implement it. I will present contrasting opinions on the benefits and pitfalls of microservices and try to arrive at a consensus viewpoint.

This thesis focuses on the aspects that influence the decision to move to microservices architecture, not on the ways how it would be executed. Thus, I will not be focusing on the finer details of technical implementation, or techniques for implementation. Instead I will focus on the managerial and business aspects of the implementation and the overall consensus definition of the term.

I have also included non-academic sources in my work, because I wanted to consider the new points of professionals who have experience with the system and how it works in reality. In addition, the majority of business people considering the pros and cons use these sources to make their decision, and not the academic literature. However, I have also taken into account that these articles are often written to sell a product, especially if they're from a company that offers help with microsystems application, so they might exaggerate the positives and omit the negatives.

1.3 Structure of the research

In Section 2 I will present necessary terms and concepts that are related to microservices architecture. Section 3 contains an explanation of microservices and their differences from the monolithic and service-oriented architectures. In Section 4 I present some positive and negative aspects of microservices that should be considered before adopting the technique. Section 5 concludes the thesis.

2. Key Concepts Related to Microservices

In this section, I will present the key concepts that are related to the microservices architecture. These concepts either help to understand the microservices or are closely in contact with them.

2.1 Modularity

Key benefits of Microservices architectures come from its modular philosophy, so first I'll introduce the concept of modularity and common benefits. According to Bask, Lipponen, Rajahonka & Tinnilä (2010), modularity does not have a universal definition, and there is not much research on modularity in services overall. Traditionally, modularity has been used in production of physical products more than services. However, research on modularity in services is now growing (Brax, Bask, Hsuan, & Voss, 2017).

One popular definition of modularity is given by Baldwin & Clark (1997) Modularity means breaking something into smaller pieces (modules) that bring flexibility. In modular systems, every module is a clearly separate or discrete unit that is complete on its own. The modules then interact, communicate and connect through an interface (Baldwin & Clark, 1997). Modules can also be nested inside one another. Different entities, teams or companies can be made responsible for separate modules of a larger system. In a truly modular system, this works reliably and efficiently, with several quality checks and tests in place to ensure smooth operation. (Sturtevant, 2018)

Benefits of modularity include flexibility, simplification, cost savings and product variety amongst other things (Bask et al., 2010). Modularity makes supply networks simpler (Arnheiter & Harren, 2005 in Bask et al., 2010), and makes it possible for firms to "outsource" functions outside from the firm or make them reusable (Hyötyläinen & Möller, 2007 in Bask et al., 2010)

According to Ernst (2005, in Bask et al., 2010), if an industry becomes too standardized, it can get caught in a "modularity trap" where new innovations aren't feasible because the industry

has invested too much within a certain standard. An example of this is the bicycle market, where the parts are highly standardized.

2.2 Service-Oriented Architecture (SOA)

Microservices is related to Service-Oriented Architecture (SOA), which was also an attempt to make software more modular. In this chapter, I will explain what SOA is and what the differences between SOA and microservices are.

SOA involves redefining applications to be comprised of services, similar to microserices. Services can be thought to be components which can independently complete a task. There are different types of services, such as:

- **Business services** that handle tasks at the business level such as calculating an insurance risk assessment or filtering customer contact details.
- **Technical or infrastructure services** that are for tasks like fetching information from databases, or user authentication and application security. They are the technical functionalities necessary for the business services to do their job
- **Application services** that are called from the user interface and are limited to a specific application scope
- **Integration services** that combine data from different sources and analyze it.

In SOA, the modules are hidden and only the interface is shown to the users (Janssen and Joha, 2008). This way the elements and modules can be changed and replaced without affecting the interface or end user experience (Bask et al., 2010). According to Newman (2015), the goal is to make the software reusable, so applications can use the same services. This would make it easier to maintain or rewrite the software.

2.2.1 Implementation problems in SOA

Saarelainen (2016) tells that SOA is not very popular solution currently, because many companies invested a lot in the idea but did not get good results when trying to implement it.

Newman (2015) believes that SOA services got a bad reputation, because they were not practical enough and the resources did not focus on implementation.

According to Villamizar et al. (2015) SOA solutions, such as enterprise service bust (ESB) can be expensive, time consuming and complex to use and maintain. For example, ESB was one SOA solution created for different parts of the software to communicate with each other. The ESB products were designed to be used by hundreds or thousands of users but are not well suited for cloud services that can have millions of users.

Journalist Ari Saarelainen interviewed Arto Santala from Solita, an IT company. According to Santala, SOA is an old-fashioned term that is used for solutions that are expensive and don't often work for companies. However, the idea of modularity and interface has influenced the microservices approach. (Saarelainen, 2016)

2.2.2 Relationship between Microservices and SOA

Vural, Koyuncu, & Guney (2017) note that SOA is an older term than microservices and that microservices take their best features and ideas from SOA. Savchenko et al. (2015) say that the concept of separating services into functions has been in use for a long time and that the microservices are a specific implementation of SOA.

Some say that microservices is just a SOA pattern or a light version of them, but at the moment most believe that it is a completely new architectural style (Pahl & Jamshidi, 2016; Savchenko et al., 2015; Villamizar et al., 2015). However, there are some key differences, which are presented in the table 1.

	SERVICE-ORIENTED ARCHITECTURE	***MICROSERVICES ARCHITECTURE***
Governance	Common governance and standards. More rigid guidelines to follow.	Relaxed governance, with greater focus on independent teams, collaboration and freedom of choice
Data Storage	SOA services share the data storage	Each microservice can have an independent data storage
Reusability	Maximizes application service reusability	Focused on decoupling
Systematic Change	A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps & Continuous Delivery	DevOps and Continuous Delivery are becoming popular, but are not mainstream	Strong focus on DevOps and Continuous Delivery
Focus	Focused on business functionality reuse	More importance on the concept of "bounded context"
Communication	For communication it uses Enterprise Service Bus (ESB)	For communication uses less elaborate and simple messaging systems
Message Protocols	Supports multiple message protocols	Uses lightweight protocols such as HTTP, REST or Thrift APIs
Platform	Use of a common platform for all services deployed to it	Application Servers are not really used, it's common to use cloud platforms
Containers	Use of containers (such as Docker) is less popular	Containers work very well with microservices

Table 1: Differences between SOA and Microservices. (Adapted from: Respodovski, 2017)

Microservices have the same goals as SOA. With microservices, systems are divided to components like in SOA. However, unlike microservices, SOA uses heavier and less agile ESB solutions, which has many implementation problems. According to Santala, microservices are like SOA but have more agility and scalability. SOA solutions have an end goal but microservices continue changing and adapting. ESB is also more expensive. (Saarelainen, 2016; Sill, 2016; Villamizar et al., 2015)

2.3 Cloud Services and SaaS

Cloud computing is closely connected to microservices, because microservices make cloud and software-as-a-service (SaaS) more accessible and easier to implement. Therefore, I'll present shortly what these terms mean.

There is a lot of hype literature and debate surrounding cloud services. But for the purposes of this thesis, a simple definition will serve: Cloud computing involves storing data in centralized data centers and/or using the processing power of servers that are outside of the companies that can used through internet. (Kandukuri, V., & Rakshit, 2009)

According to Choudhary (2007) cloud services bring a lot of concern regarding data security. Because the data is stored outside of the business, there are more security risks. Businesses have to trust the supplier with the security. Villamizar et al. (2015) however tells about many possibilities cloud services bring, because they enable better scalability, efficiency and dynamics. Computing power can be bought very flexibly according to use.

Software-as-a-Service (SaaS) means buying software to be used through internet on subscription basis. Compared to perpetual license, where business pays for services per-use, SaaS uses periodical payments, often monthly subscriptions. SaaS solutions are not possible with non-modular, monolithic software architecture. However, with microservices it is easy to incorporate SaaS solutions. (Choudhary, 2007; Singleton, 2016; Thönes, 2015)

The University of Florida found in a study that SaaS can bring many advantages to businesses including updating software as soon as the new versions are available, economies of scale, predictable costs and ability to switch providers. Also, SaaS does not necessarily need big upfront investments in infrastructure (Choudhary, 2007). Advantages include also the ability to scale the services on-demand, which is especially helpful during peak periods (Villamizar et al., 2015). Previously, before cloud services, getting more computing power meant investing in hardware that would be unused most of the time.

A quote from David Strauss, CTO of Pantheon sums it up: "[Previously], you would want to start with a monolith because you wanted to deploy one database server. The idea of having to set up a database server for every single microservice and then scale out was a mammoth task. Only a huge, tech-savvy organization could do that. Whereas today with services like Google Cloud and Amazon AWS, you have many options for deploying tiny things without needing to own the persistence layer for each one." (Lumetta, 2018)

3. Microservices in Detail

In this section I'll explain what microservices are and how they have developed.

3.1 History of the term "microservices"

There is no clear, established definition for the term "microservices". The first time the term appeared in academic papers was 2014, when Fernandez-Villamor, J.I., Iglesias, C. and Garijo, M published an article "MICROSERVICES lightweight service descriptions for rest architectural style".

Newman (2015) notes that while SOA was created theoretically, microservices were born with practice. According to Despodovski (2017), the term "microservices" was agreed upon around 2012, when some software architects recognized that they had been working on a similar style of architecture. However, the style had already been in use for almost a decade before that.

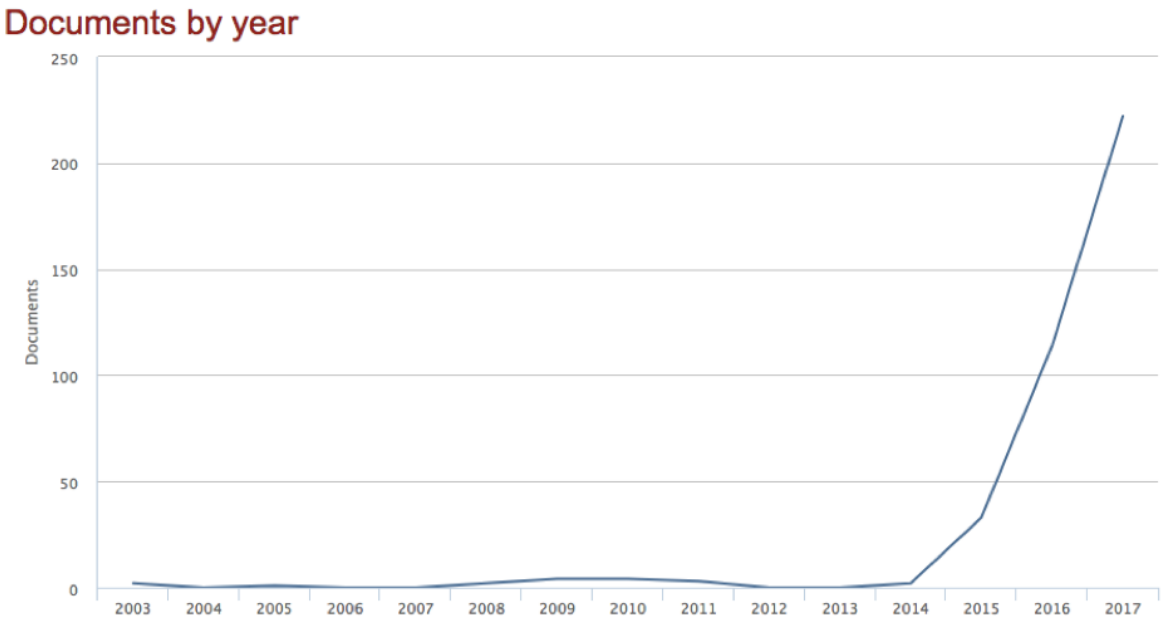

Figure 1: Search results with "microservices" without exclusion criteria (Source: Scopus 6.3.2018)

From Figure 1 we can see that the subject of microservices is very recent in academic research. There are altogether 419 results with the word "microservices" in title, abstract, name or keywords (6.3.2018). The research on the subject has been growing fast since 2014 and is still growing in popularity. This might mean that there aren't many long-term studies on the subject yet.

3.2 Monolithic systems

Microservices are often built over monolithic systems so in this section, I will first explain what they are. Monolithic system architecture means having the application in a single unit or code base. (Savchenko, Radchenko, & Taipale, 2015; Villamizar et al., 2015). This includes the front end or client-side logic, server side back end logic, hidden processes and everything else the whole system needs.

When the needed database is small, it is a good idea to have the application in a monolithic architecture, because it can handle all of the parts of the application. (Singleton, 2016) Also, some people believe that when an application is new, it is best to do it first in a monolithic way, so it does not get too complicated too fast. There are examples of big companies that

have successfully stayed in monolithic architecture, so it's not a requirement for a system to be turned into microservices (SaM Solutions, 2017). However, this view is not shared by everyone.

Savchenko (2015) notes that monolithic systems have many negative sides especially when the system gets bigger and more complicated. Singleton (2016) adds that when a change needs to be made, in monolithic system you have to test and release the whole system at once, which takes a lot of time. This makes the process slow and not efficient. Also, systems are usually first made in monolithic architecture but when the system grows, it becomes complex and a lot less agile. There is a limit, how much one system can handle efficiently (Singleton, 2016). Sturtevant (2018) writes about a study, where they compared a complicated, monolithic style system to other, better organized systems and analyzed the cost of complexity. The developers' productivity was 60 percent less than in other teams and 70 percent of the time they had to fix problems in the code instead of value-creating activities. According to their analysis, this brings significant competitive disadvantage.

Today, it's relatively easy to scale the systems according to use with cloud computing, but it's not as easy with monolithic systems. In monolithic systems, having just in-house servers leads to wasting server resources because they have to be available even when they're not in use (Villamizar et al., 2015). According to Newman (2015), monolithic systems could technically have modular architecture but in practice, the modules become too integrated. To get the benefits of modularisation, the modules should be independent and clearly separated, but in a monolithic architecture, this rarely works.

3.3 Microservices definition

Microservices architecture is a pattern for software development. It involves splitting complex or large software projects into small, modular services that are developed separately, sometimes by different teams, which can be deployed independently of each other (Villamizar et al., 2015; Vural et al., 2017; Savchenko et al., 2015). There is no concrete definition of microservices architecture that is agreed upon by the software industry. However, there are quite many concepts that are central to the concept and can be presented here.

Firstly, individual services should be responsible for separate, focused tasks and not concerned with completing the operations of other modules. This is the reason the term 'loosely coupled modules' is often associated with microservices. However, the services may sometimes need to communicate with each other to do their tasks. For instance, a payment focused service may need to acquire a user's email or personal information from the user profile module to generate receipts. On the whole, however, services are independent, work separately and can be scaled, deployed and tested independently. Each service works by themselves and interacts through an overarching interface. (Sill, 2016; Villamizar et al., 2015)

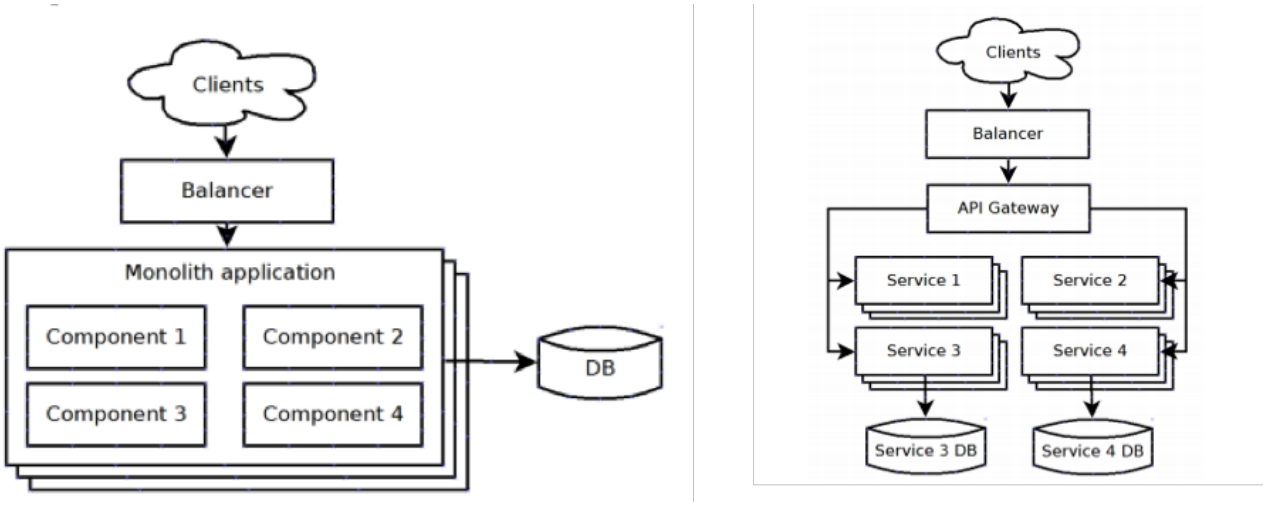

Figure 2: Monolithic vs. microservice system architecture. (Source: Savchenko et al., 2015)

In Figure 2, Savchenko et al. (2015) visualize how the microservice and monolithic system architectures differ from each other. In the monolithic system, all the components are together, but in microservices, the services are separate and work as separate entities. This is shown even clearer in the example in the following Figures 3 and 4.

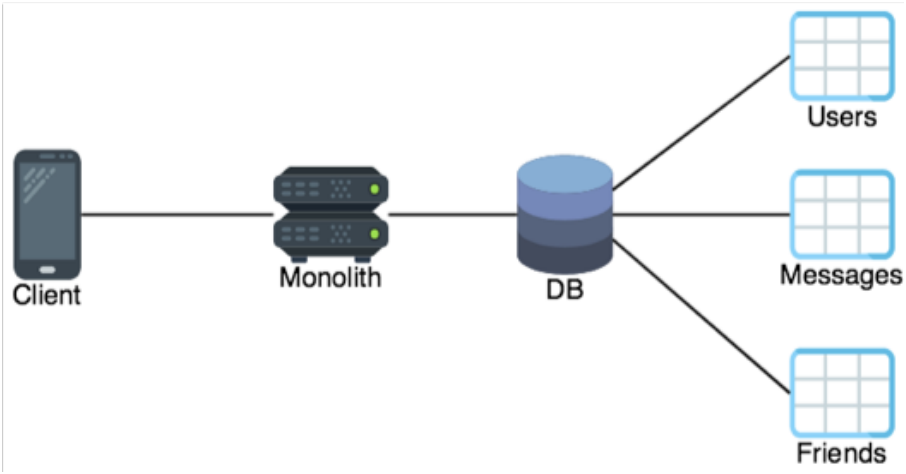

Figure 3: Example of a monolithic architecture (Source: Peck, 2018)

In the third figure, Nathan Peck has illustrated a clearer example of monolithic architecture in a social app. Everything works in the same system and all data is in the same storage.

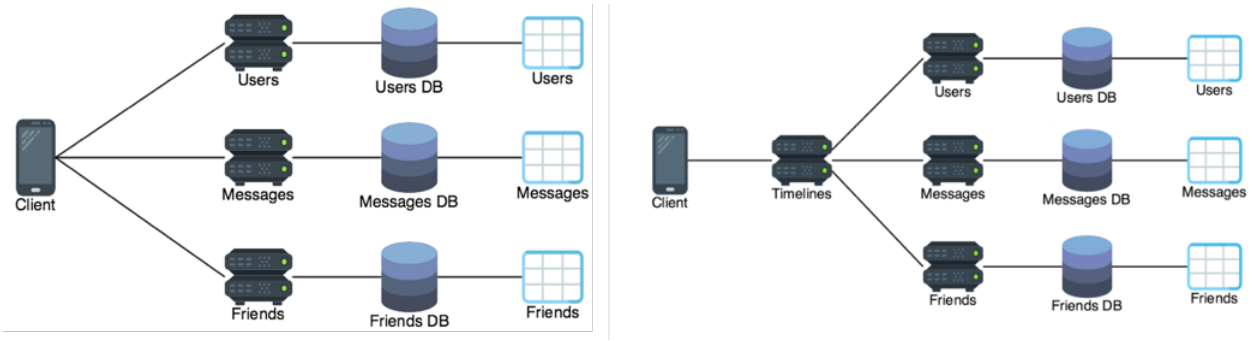

Figure 4: Examples of the microservices architecture. (Source: Peck, 2018)

In the fourth figure, Peck (2018) shows a specific example of how the microservices architecture can work. Different parts of the systems are in separate, independent servers and data is in three different storages. The figure shows how the system can be used by getting the information from three different sources or through timelines, that combine the information and making the system simpler to the client without making it slower.

As an example, consider an application designed for buying tickets for events. The system could be built out of different modules like

- Data Input modules for gathering details of events from vendors
- Login and authentication systems for different types of users
- Payment system module
- Modules to fetch event information from a database and display it to end users
- Search module for text strings

Figures 3 and 4 shows clearly how a system's architecture becomes more complicated when it is refactored into microservices architecture. However, the pieces of the system are now easier to manage, and update as needed. Also, the internal details can be hidden from the client or customer, which makes the system simpler for consumers (Newman, 2015).

There are many opinions on how big a microservice should be. According to Lari Hotari (Saarelainen, 2016), the name is misleading, because the goal for microservices is not to be as small as possible. The size of a microservice should be determined on how small a service can be while still being self-contained. For a mental rule of thumb, consider Amazon's informal "two pizza rule", which references the ideal size of a microservice team, i.e. one which can be fed with two pizzas. (Saarelainen, 2016) Thus, more important than size or scope of the service, is the rule that the microservices are loosely coupled, independent from each other, and that they have high cohesion, so that the places where the code or data needs to be updated is minimised (Newman, 2015).

4. Pros and Cons of Microservices

In this chapter, I will explore the characteristics of microservices that affect the decision of moving to microservices. There are many examples of implementing microservices in big corporations, e.g. Netflix and Spotify, but there are also some large firms like Etsy, that still work with monolith architecture (SaM Solutions, 2017). This shows that it is good to explore the positive and negative aspects of microservices instead of following a trend without thinking about the consequences.

4.1 Positives of microservices

According to Newman (2015), the benefits that microservices bring could be accomplished in theory with any system, but in practice, microservices are the best way. The benefits come from distribution and modularization and with microservices, the functions and systems can be very well distributed. As Figures 2 to 4 above show, while the system architecture can become more complicated compared to monolithic systems, the individual parts are usually easier to understand and develop further. The services have clear boundaries that help understand where the different services operate and what they should accomplish (Villamizar et al., 2015).

With microservice architecture, companies are able to change and update the services continuously and use agile methodologies and cycles (Singleton, 2016; Sturtevant, 2018; Villamizar et al., 2015). In last years, continuous delivery and agile methods have become very popular, especially in internet companies, startups and SaaS providers (Villamizar et al., 2015).

According to Singleton (2016), microsystems architecture is proven to work in many occasions and they fit together like lego blocks. This is how the system can be quickly developed and scaled. Sturtevant (2018) warns that having just agile processes without agile architecture, because agile processes can't alone make the system agile.

According to Julien Lemoine, CTO of Algolia, one of the key benefits is the increased freedom in choosing the right tools and teams for different problems: "Our search API is highly optimized at the lowest level and C++ is the perfect language for that. That said, using C++ for everything is a waste of productivity, especially to build a dashboard! We want the best talents and using only one technology would limit our options" (Lumetta, 2018).

Unlike monolithic systems, microservices can control the capacity of servers more easily and flexibly (Singleton, 2016; Villamizar et al., 2015). Server capacity can be moved from a less used to a busy server to avoid bottlenecks (Singleton, 2016). This scalability and flexible capacity make microservices in many cases cheaper compared to monolithic systems, because smaller pieces can be scaled instead of the whole big system (Newman, 2015).

Microservice architecture also reduces redundancy, because the same data does not have to be stored in many places. Different systems can use the same data storages so there is no need to update multiple places. This way also mistakes, and old data can be reduced (Singleton, 2016; Villamizar et al., 2015). Singleton (2018) also notes that microservices are useful when a company has multiple products or services because the microservices can be reused in more than one product. Newman (2015) notes that the whole system can be thought as a holistic concept; for example, mobile and desktop applications do not have to be thought as separate from each other.

4.2 Negatives of microservices

Savchenko et al. (2015) remind that microservices is not an easy solution for every problem. Instead of one code base, there are multiple services and databases that can make the system complicated, as was shown in Figure 4 in the previous chapter. (SaM Solutions, 2017; Sturtevant, 2018). Essentially, this makes the system more complex by moving the complexity to the infrastructure.

Singleton (2016) warns that microservices are not suitable for small systems. If there's not a lot of data, it can take more time to build the system than is necessary and time and resources get wasted. According to Sturtevant (2018), microservices are the most beneficial for big, complex systems. In the words of Steven Czerwinski, Head of Engineering at Scaylr and former Google employee, "Even though we had had these positive experiences of using microservices at Google, we [at Scaylr] went [for a monolith] route because having one monolithic server means less work for us as two engineers." (Lumetta, 2018)

Microservices architecture also means that team management is different. The way of thinking is very different from a monolithic one, so employees have to unlearn some things and learn new (Newman, 2015). The teams are more independent and responsible for one clear part of the system. This brings challenges in the transition period but in the long run can be beneficial. (Villamizar et al., 2015) This is where the management of change is very

important, and the lack of proper management can make the transition difficult and costly. There is also need for more management because the system is in smaller pieces that need more overview (SaM Solutions, 2017).

Moving to microservices may bring more costs especially in the beginning. Singleton (2016) notes that the move needs extra machinery, which can increase the costs significantly. Microservices also need skilled employees that have high level of expertise and they take more time to develop (SaM Solutions, 2017). This can increase for example the wage costs. Also, even though the costs of build and maintenance are reduced, the operational costs may rise because of increased complexity. (Singleton, 2016)

Overall, there are in most cases more positive than negative aspects in microservices, at least for smaller systems. Singleton (2016) reminds that especially for small systems, moving to microservices is not necessarily the right choice. Microservices can add unnecessary complexity and slow the development down too much in the beginning (SaM Solutions, 2017).

To conclude, here are the pros and cons of first monolithic and then the microservices architecture in two tables.

Monolith Pros and Cons

Positives
Good for systems with small databases
Easier to find qualified experts
More out of the box solutions available for most industries
System architecture is less complicated
Testing can be easier
Easier to manage with a small team
Negatives
If something fails, the whole system might go down because everything is connected
New trend and pattern, so finding qualified architects and developers can be hard
In practice, cannot be modular, so e.g. using cloud services is more difficult, lowers productivity
Not easy to scale
Making changes is difficult as everything is dependent of each other

Table 2: Pros and cons of monolithic systems

Microservices Pros and Cons

Positives
Well suited to Agile methodology and teams – service teams can work independently
Easier to understand and develop each service module separately
Continuous development/ deployment much easier
Ability to use different tech stacks for different modules
Proven to work for large scale needs e.g. Netflix
Reduces redundancy in code
Flexibility - switch modules out at will as needs evolve
Failure in one module doesn't compromise whole system and critical errors can often be traced more easily
Cloud services can be cheaper and easier than buying and maintaining infrastructure
Negatives
New ways of thinking and operation in company needed
Complex infrastructure requirements, relative to simpler monolithic systems
Costly in the beginning
Needs specialized experience in developers
Not recommended for small systems and proofs-of-concept prototypes
Testing/deploying is harder than with monolith
Reliance on cloud services if own infrastructure is too costly or difficult. Can defy security policies

Table 3: Pros and cons of microservices

5. Summary and conclusions

My research question that I presented in the beginning has two parts: 'What is microservices architecture in software' and 'what businesses should consider before and during implementation?'

In short, microservices architecture is a way to organize system architecture so that it's divided into small, modular services that are developed separately and can be deployed independently of each other. The services have low or no interdependence, and they communicate with each other, usually through application programming interfaces, APIs. In monolithic systems, the parts of the system are not separated, and the system uses only one or a few servers and databases.

The second part of the question focuses on weather business should choose monolith or microservices architecture for their system or should they change from monolith to microservices. In short, monolithic architecture can still have certain advantages. Most often, these boil down to lower overheads when starting, ease of finding competent developers and technical advisors and feasibility for small teams, especially those with no knowledge of cloud development or microservices architecture.

If a key requirement is quick delivery of an independent service or module for a large service, then it makes sense to use microservices. For example, when Reaktor was implementing YLE Areena and Uutisvahti, they built the systems in small pieces that they could test and deploy immediately. The previous model was not suitable for growing number of users, so they created a new system based on microservices. To avoid complexity, they built the smallest possible pieces that worked (minimum viable product). This made the installation easy, because they didn't have to implement it all at once. This also means that if some part was built wrong, it could be changed easily, and it didn't take a lot of resources to start over. Although in the beginning microservices architecture demands some high investments and planning, the maintenance costs have been lower because the pieces of the system are simpler and independent (Karemo, 2017).

Another situation where microservices are indispensable is if certain parts of the platform need to be very optimized or need specialized technology or tooling that would not benefit other parts of the system or would actively harm their development. For instance, if large volumes of data need to be processed quickly such as in a financial backend system, a specialized language may be necessary while the user interface can be put together with more common tools like JavaScript. (Lumetta, 2018)

A third case is when the development team is distributed far geographically or split into many different teams. Microservices architecture is built for these situations, especially for rapidly growing teams. Teams get to use the tooling they're comfortable with and deploy highly optimized, independent solutions without worrying how they affect the rest of the codebase, with communication between modules done over simple lightweight protocols and simple messaging systems. (Lumetta, 2018)

In the table 4 below, I have compared the different pros and cons of microservices and monolith architectures. This concludes that monolithic architecture is still a good choice for some systems, but microservices can, when well implemented, bring significant benefits.

Microservices Positives	Monolith Negatives
Easier to understand each service module separately	Lowers productivity because of complexity in large code bases
Continuous development/ deployment much easier	Changes have to be done periodically all at once
Ability to use different tech stacks for different modules	Have to use the same tech stack for the whole system
Reduces redundancy in code	Making changes is difficult as everything is dependent of each other
Failure in one doesn't compromise whole system	If something fails, the whole system might go down because everything is connected
Flexibility - switch modules out at will	Not easy to scale
Cloud services can be cheaper and easier than buying and maintaining infrastructure	In practice, cannot be modular, so e.g. using cloud services is more difficult
Well suited to Agile methodology and teams– service teams can work independently	
Proven to work for very large scale needs e.g. Netflix	

Microservices Negatives	Monolith Positives
Complex infrastructure	System architecture is less complicated
Not for small systems	Good for systems with small databases
Costly in the beginning	More out of the box solutions available for most industries
Needs specialized experience in developers	Has been around for longer so there's more knowledge about it
Testing/deploying is harder than with monolith	Easier to manage with a small team
New ways of thinking and operation in company needed	Easier to find qualified experts

Table 4: Combined table: Comparing microservices and monolithic architecture

Limitations and future research

One limitation that I encountered was that there is surprisingly little scientific literature on the subject considering how widely microservices are used. This is why I used also sources that are not peer-reviewed even though their scientific aspect can be questionable. However, people who work with microservices are able to provide valuable practical knowledge on the subject.

This thesis is not meant to go into the technical side of microservices so interested readers should find some other sources to find more about the different ways of implementing microservices. There are many ready-made solutions for adopting microservices and it's possible to do that in different methods, but they are left out of the scope of this thesis. The differences can affect the costs and other parts significantly, so before making decisions about microservices, it is good to look into them.

In the future research, the long-term implications and especially the negative effects could be researched more. It is also interesting to see how modularity and maybe even microservices could be used in services and user experience management.

References

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, *33*(3), 42–52. https://doi.org/10.1109/MS.2016.64

Baldwin, C. Y., & Clark, K. B. (1997). MANAGING IN AN AGE OF MODULARITY. *Harvard Business Review*, *75*(5), 84–93. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=9709112720&site=ehost-live&authtype=sso&custid=ns192260

Bask, A., Lipponen, M., Rajahonka, M., & Tinnilä, M. (2010). The concept of modularity: diffusion from manufacturing to service production. *Journal of Manufacturing Technology Management*, *21*(3), 355–375. https://doi.org/10.1108/17410381011024331

Brax, S. A., Bask, A., Hsuan, J., & Voss, C. (2017). Service modularity and architecture – an overview and research agenda. *International Journal of Operations and Production Management*, *37*(6), 686–702. https://doi.org/10.1108/IJOPM-03-2017-0191

Chen, L. (2015). Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, *32*(2), 50–54. https://doi.org/10.1109/MS.2015.27

Chen, L. (2015). Towards Architecting for Continuous Delivery. In *2015 12th Working IEEE/IFIP Conference on Software Architecture* (pp. 131–134). https://doi.org/10.1109/WICSA.2015.23

Choudhary, V. (2007). Software as a Service: Implications for Investment in Software Development. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on* (p. 209a–209a). https://doi.org/10.1109/HICSS.2007.493

Despodovski, R. (2017). Microservices vs. SOA – Is There Any Difference at All? Retrieved April 29, 2018, from https://dzone.com/articles/microservices-vs-soa-is-there-any-difference-at-al

Dittrich, K., Duysters, G., & de Man, A.-P. (2007). Strategic repositioning by means of alliance networks: The case of IBM. *Research Policy*, *36*(10), 1496–1511. https://doi.org/https://doi.org/10.1016/j.respol.2007.07.002

Kandukuri, B. R., V., R. P., & Rakshit, A. (2009). Cloud Security Issues. In *2009 IEEE International Conference on Services Computing* (pp. 517–520). https://doi.org/10.1109/SCC.2009.84

Haapaniemi, T., Karemo, J., Valkonen, P. (2017). A Story of a Microservice: Lessons from the Trenches. Retrieved April 22, 2018, from https://www.reaktor.com/blog/a-story-of-a-microservice/?ads_cmpid=736961476&ads_adid=46928756148&ads_matchtype=b&ads_network=g&ads_creative=198322395068&utm_term=microservice&ads_targetid=kwd-388427564&utm_campaign=&utm_source=adwords&utm_medium=ppc&ttv=2

Kratzke, N. (2017). About microservices, containers and their underestimated impact on network performance. *arXiv Preprint arXiv:1710.04049*.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, *36*(6), 47–56. https://doi.org/10.1109/MC.2003.1204375

Lumetta, J. (2018). Microservices for Startups: An Interview with Julien Lemoine of Algolia. Retrieved April 29, 2018, from https://buttercms.com/blog/microservices-for-startups-an-interview-with-julien-lemoine-of-algolia

Lumetta, J. (2018). Should You Start With A Monolith or Microservices? Retrieved April 29, 2018, from https://nordicapis.com/should-you-start-with-a-monolith-or-microservices/

Miller, G. G. (2001). The characteristics of agile software processes. In *tools* (p. 385). IEEE.

Newman, S. (2015). *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc."

Ouertani, S. (2015). „From Microservices to SOA ". *Von T. Erl. Issue XCI. Arcitura Education Inc*, 4–9.

Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. In *CLOSER (1)* (pp. 137–146).

Peck, Nathan. (2018). Microservice Principles: Decentralized Data Management. Retrieved 29.4.2018, from https://medium.com/@nathankpeck/microservice-principles-decentralized-data-management-4adaceea173f

Saarelainen, A. (2016). Mikropalvelut korvaavat it-möhkäleet. Retrieved April 22, 2018, from https://www.tivi.fi/Kaikki_uutiset/mikropalvelut-korvaavat-it-mohkaleet-6588283

SaM Solutions. (2017). Microservices vs. Monolithic: Real Business Examples. Retrieved April 22, 2018, from https://www.sam-solutions.com/blog/microservices-v

Savchenko, D. I., Radchenko, G. I., & Taipale, O. (2015). Microservices validation: Mjolnirr platform case study. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on* (pp. 235–240). IEEE.

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

Singh, M., Chandhoke, K., Verma, A., & Singh, J. (2017). Microservices Design & Development explained (Case Study). Retrieved from https://www.sourcefuse.com/microservices-design-development-explained-case-study/

Singleton, A. (2016). The Economics of Microservices. *IEEE Cloud Computing*, *3*(5), 16–20. https://doi.org/10.1109/MCC.2016.109

Sturtevant, D. (2018). Modular Architectures Make You Agile in the Long Run. *IEEE Software*, *35*(1), 104–108. https://doi.org/10.1109/MS.2017.4541034

Thönes, J. (2015). Microservices. *IEEE Software*, *32*(1), 116. https://doi.org/10.1109/MS.2015.11

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)* (pp. 583–590). https://doi.org/10.1109/ColumbianCC.2015.7333476

Vural, H., Koyuncu, M., & Guney, S. (2017). A Systematic Literature Review on Microservices BT - Computational Science and Its Applications – ICCSA 2017. In O. Gervasi, B. Murgante, S. Misra, G. Borruso, C. M. Torre, A. M. A. C. Rocha, … A. Cuzzocrea (Eds.) (pp. 203–217). Cham: Springer International Publishing.

Watts, S. (2017). Microservices vs SOA: What's the Difference? Retrieved April 28, 2018, from https://www.bmc.com/blogs/microservices-vs-soa-whats-difference/