Decision criteria between microservice and monolithic architecture

University of Oulu
Department of Information Processing Science
Niklas Menard
03.09.2020

Abstract

In the contemporary software market companies face a challenge of continuously developing and delivering their products quickly. To answer this challenge the correct software architecture must be chosen. The conservative approach is a monolithic architecture, where all the code base is in a single unit. This approach offers simplicity and rapid initial deployment but faces challenges when companies need to scale their software. A more novel approach is a microservice architecture, which was enabled by the growth of cloud infrastructure. This architecture offers higher scalability and autonomy but brings with it a higher level of complexity.

In this study I conducted a literature review to examine both architectures to understand the advantages and disadvantages of both approaches. The intent was to get a clear understanding of the underlying criteria that companies need to consider when making an architectural related decision. The current literature revealed that the advantages and disadvantages of both architectures are quite well known, but there is ambiguity regarding the criteria that is outside of the functional requirements discovered during the design phase. This study offers a baseline to further study decision criteria regarding monolithic and microservice cloud-based applications. Further studies can be done to further examine the criteria on a more detailed and practical level.

Keywords
software architecture, microservice architecture, monolithic architecture, DevOps

Contents

1. Introduction

This literature review focuses on the microservice software architecture and how it compares to the traditional monolithic approach. We will first examine on a general level what software engineering and what a software architecture is and then delve into the details of a microservice architecture as well as a monolithic architecture. We will compare the strengths and weaknesses of both approaches. The goal is to get an understanding of the criteria on which architecture to use in different situations. The research method used is literature review, in which we summarize the previous research on the topic and deduct commonalities and principles.

The topic is current and important as more and more companies are trying to answer the market demand for quick product delivery, something that microservice applications can possibly deliver. As the research shows there are specific situations when to use either software architecture. The microservice architecture is complex to execute and can possibly create more issues than it can solve. The monolithic approach is simpler to implement but will be complex to scale. Choosing the right architecture can save companies a lot of time and resources.

The outcome of this review is a set of criteria that should be taken account when deciding related to the software architecture being chosen. The criteria are presented on a general level. One option for future studies is to delve into the details of the criteria and examine what they mean on a practical level. The goal is to create a set of guidelines that could help with the software architecture decision making process.

The motivation for the study was to get a better understanding of the contemporary software architectures of modern cloud-based applications. The challenge of designing software before implementing them is very interesting to me as it makes the whole process a lot simpler when you have a roadmap to follow. The decision to switch architectures is not a light one for companies so creating a decision framework would be very beneficial for the process.

The structure of the study is the following: examine the relevant terminology, view research methods, explore the prior research, and finally findings, implications, and conclusions.

2. Terminology

In this chapter we will briefly go over the relevant vocabulary that is related to the study. The goal is to give a quick overview of the relevant concepts and the larger concepts will be examined in detail further on in the study. The relevant vocabulary and concepts related to this study are as follows: software architecture, service-oriented architecture, ESB, microservice architecture, monolithic architecture, and DevOps.

2.1 Software and service-oriented architecture

Software architecture can be described as a macro level blueprint of how the different components of the program will be organized within the system, how they interact with each other and how the environment affects their design and evolution (P. Eeles, 2006). Service-oriented architecture is a software architecture where the purpose is to increase usability of software components by separating them into services that offer functionality through commonly used communication standards. The architecture makes us of an ESB or enterprise service bus, which is a centralized component that executes the integration to backend systems and forms service interfaces out of them. (IBM, 2019.)

2.2 Microservice and monolith architecture

Microservice architecture is a cloud native architecture where the application is split up into services that each have their own stack and they communicate with other services through REST APIs. The services are divided according to business capabilities. (IBM Cloud Education, 2019.) The main difference between a microservice architecture and service oriented architecture is the scope of the architecture and use of an ESB. Service oriented architecture is an enterprise level architecture while microservice architecture is an application level architecture. Microservice architectures generally do not utilize ESBs as they restrict the autonomy of the services by creating a bottleneck that all services must adhere to. (IBM, 2019.) Monolith architecture is a software architecture in which the whole application is located in a single codebase. The application can consist of different component that are all deployed and scaled as one unit. (Esposito, Castiglione, & Choo, 2016.)

2.3 DevOps

DevOps is derived from the words “development” and “operations. In practical terms it is a series of operations that aim to make the development and release of software quicker by utilizing automation as well as integration. It builds on previously established lean and agile practices. (Ebert, Gallardo, Hernantes, & Serrano, 2016.)

3. Research methods

The research method used in this study is a literature review. A literature review consists of gathering information related to a specific subject from books, articles, scholarly articles, or any other reliable sources. The goal is to create a description of the issue, summarize it and examine it critically by comparing sources to each other. During the process, each source should be examined according to the research problem that has been formulated. (Grant & Booth, 2009.) The purpose of a literature review is to determine how your own research fits the existing literature, provide an overview of essential concepts, examine possible conflicts between sources, find new interpretations of prior studies, identify possible research gaps and present the theoretical background for what has already been established (A. Fink, 2020).

In this study we gathered literature from several peer-reviewed literature databases including Scopus, EBSCOhost, and Google Scholar. The validity of sources was verified by checking the reliability of the publication from Finnish Publication forum. The search was limited to the architectural level of applications. The search in the databases included phrases like: “Software AND Architecture”, “monolith AND architecture”, “DevOps” and “Service AND Oriented AND Architecture”. The gathered literature covered the topic from a variety of different angles and produced an overview of the relevant information. Some of the definitions for terms used throughout the paper were found from non-academic but otherwise trustworthy sources (e.g. IBM).

An important note is that the software industry is further ahead than the academic world when it comes to microservices. A lot of the literature mentioned that there is still a lot research to be done to establish the theory in an academic sense. The most current information came from studies that were either based on quantitative questionnaires or benchmarking techniques.

4. Prior research

Before we can examine the criteria that should be considered when choosing between an microservice or monolithic approach, we must first explain briefly what software engineering and architecture means. After we have examined the basic concepts of software engineering and architecture, we will delve into the central concepts of the microservice design approach as well as its counterpart the monolithic approach. We will then examine the ways both approaches can be evaluated and then compare the strengths and weaknesses of each approach. The goal is to get a clear picture of when to use either approach as well as list the criteria that this decision should be based on.

To briefly summarise the most relevant literature related to the research are the following:

- Aderaldo et al., (2017) give an overview of the microservice research and examine the benchmark requirements to accurately measure microservice applications for academic purposes (Aderaldo, Mendonça, Pahl, & Jamshidi, 2017).

- Jamshidi et al., (2018) explain the developmental stages microservices have gone through and what the challenges are in the future (Jamshidi, Pahl, Mendonca, Lewis, & Tilkov, 2018).

- Lewis and Fowler (2014) give a theoretical overview of the central concepts related to microservices and how it compares to monolithic architecture (Martin Fowler & James Lewis, 2014).

- Villamizar et al., (2015, 2016, 2017) examines the strengths and weaknesses of the microservice and monolithic architecture through benchmarking cloud based applications (Villamizar et al., 2015; Villamizar et al., 2016; Villamizar et al., 2017).

- Soldani et al., (2018) review industry pain points in their literature research related to microservices (Soldani, Tamburri, & Van Den Heuvel, W. -J., 2018) .

- Taibi, Lenarduzzi and Pahl (2017) conducted an empirical investigation in the motivations, issues and processes related to microservice architecture migrations (Taibi, Lenarduzzi, & Pahl, 2017).

- Balalaie, Heydarnoori and Jamshidi (2016) and Di Francesco, Lago and Malavolta (2018) give an overview of the challenges faced during an application migration process to an microservice architecture (Balalaie A., Heydarnoori A., Jamshidi P., 2016; Di Francesco P., Lago P., Malavolta I., 2018).

Other literature is related to central concepts and their purpose is to give a wider perspective on the topic as well as a varied view.

4.1 Software Engineering

Pressman describes software engineering as a process of implementing a systematic engineering approach to the development of software with an emphasis on quality. The process itself is not a strict ruling of how to execute various software building tasks but rather a general approach that people working on the project agree upon. Processes are a collection of operations that need to be completed to achieve a certain task. On a general level software engineering processes are built on foundational frameworks that establish five activities: communication, planning, modelling, construction, and deployment. (Pressman, 2010.)

Communications activities are crucial for determining the direction that the overall project or the next development step should be taking. Communication activities encompass all the correspondence that occurs between the development team and the customer as well as other stakeholders. Planning activities create a blueprint or a software project plan for the actions that must be executed to reach the functionality that has been plotted. A software project plan imparts the technical tasks, risks, resources required and the work schedule that is related to the developmental tasks of the project. Modelling activities revolve around creating understandable visual models to understand how the requirements of the software are met by the design of the software. Construction activities are the execution of the design or in other words writing code that reflects the design that has been laid out beforehand. Deployment activities are implemented once the software product or partial iteration has been completed and the results can be returned to the customer for further review. (Pressman, 2010.)

Software engineering activities are executed in small or large projects, and often iteratively throughout the project. The goal is to create a small tested iteration that when approved by the customer becomes a small part and progression of the larger software project. In addition to the previously stated activities there are several complementary umbrella activities that are executed to help and control progress, quality, change and risk. These umbrella activities are for example risk management, technical reviews, reusability management and measurements. (Pressman, 2010.)

It is important to note the several external or internal factors that affect the software design process, including social, organizational, economic and political factors. All of these factors create constraints on how the software can be developed and they cannot be overlooked. After all it is a team of people who develop the software and it is crucial that everybody understands the common goal and what is their role in the development process. (Budgen, 2003.)

Some activities are more emphasized at different phases of the project and some are crucial throughout the project. For example, maintaining clear communications is important throughout the project but certain planning related tasks are more crucial at the beginning of the project. One of these planning and modelling related tasks is forming and designing the architecture of the software. (Pressman, 2010.)

4.2 Software architecture

In the 1980s software systems were running on a single computer or mainframe and their architecture was contained within those systems. During this time software architecture was not a developed discipline but it slowly started to take form when the concept of encapsulation and modularization was introduced to structural programming design techniques. The need for large scale software and the complexity that distributed systems introduced helped to evolve the idea of software architecture. (Woods, 2016.)

Software architecture is a way to answer the technical needs of a software application by creating a structured solution and blueprint. To start creating a structured solution it is crucial to understand the underlying requirements of the application, how those requirements shape into software elements, how those elements interact with each other and how the overall design can be implemented. This is not always a simple task as there are a multitude of ways perceive these requirements and designing them into functional solutions. (Wolf & Perry, 1992.) It is important to differentiate software architecture from software design as architecture is concerned more with the higher-level structures and the underlying relationships. Architecture involves the organizational, business and quality attributes, while software design mainly focuses on the technical modules and how to organize the code within them. Broadly speaking architecture involves the entire system and design is centred around the details. (Pillai, 2017.)

Wolf & Perry (1992) explain that after the requirements of the application have been understood, the next step is to create a logical model of the solution. The software architecture model can be imagined containing the following three parts: elements, form, and rationale. The elements can be further deconstructed into three subclasses: processing elements, data elements and connecting elements. Processing elements are the components that execute processing functions, the data elements store the data and connecting elements work as the glue elements of the application. Form of the architecture model dictates the importance of properties and relationships. Properties are constraints put on architectural elements and relationships are used for determining the way different elements interact with each other. Lastly the rationale of the architecture model condenses the thought process and motivation of the whole design. If correctly done the software architecture design is a blueprint that designers can follow and execute. (Wolf & Perry, 1992.)

However, Pillai (2017) gives a more comprehensive explanation and divides the concept of software architecture into four themes: system, structure, environment, and stakeholder. A system is a collection of software components that are organized so the desired functionality can be achieved. A structure is a collection of elements that are organized according to a set rule or principle and the individual elements can be either software of hardware components. The environment is the context that where the system is being developed (e.g. business, entertainment) and stakeholders are the people who have invested into the project. All these themes affect the architecture and they are factors that need to be considered. On a basic level a software architecture defines the structure of the system, it sets the elements that system contains, it determines the early design decisions, it manages stakeholder needs, it influences the impacts the organizational structure and it is influenced by its environment. (Pillai, 2017.)

4.2.1 Software architecture decision making process

To emphasize one of the most important factors to remember is that any architectural decisions must be based on non-functional requirements that have been discovered usually at the beginning stages of the development process. The architecture should reflect these requirements and intend to correspond to them. (Jansen & Bosch, 2005.) On a context design level software architecture is represented using architectural context diagrams (ACD) that highlights the entities that the system interacts with. Once the context has been established the next step is to define architectural archetypes, which are abstractions or patterns that represent critical aspects of the design. The process of designing the architecture is gradual and as the refinement of the architecture proceeds, the essential components start to emerge. Components are formed according to entities and technical infrastructure requirements that were identified previously.

Determining the level of modularity, cohesion and coupling of these components is one of the most important aspects that the architecture must convey. The goal of a software architecture is to have the right amount of modules that have the correct amount of cohesion, they are reusable, easy to understand and maintain. (Pressman, 2010.)

The software decision process is comprised of the following elements: rationale , design rules, constraints, and additional requirements. Rationale is the reasoning behind the architectural decision and design rules, or constraints are the boundaries that future software additions must adhere by. Additional requirements might emerge later in the product life cycle and the software architecture must address these requirements. (Jansen & Bosch, 2005.) An important step in the decision-making process is the documentation of the decision-making process. It is crucial to have an explanation of the rationale behind choosing the current architecture and why other alternatives were not chosen. The lack of documentation can lead to knowledge vaporization related to the systems architecture, which in turn delays development and incurs unnecessary costs. (Harrison, Avgeriou, & Zdun, 2007.)

Lack of documentation and understanding of the underlying architecture can lead several issues in the development of the software such as contradicting design decisions, violating of previously established rules and constraints as well not removing unused design decisions. (Jansen & Bosch, 2005.) Harrison, Avgeriou, & Zdun (2007) note that one answer to the challenge of software architecture documentation is using a pattern solution. Patterns are general solutions to problems that occur often. In the software architecture context, they inherently capture critical information about the decision-making process, they make documentation easier as each pattern conveys essential information in itself and they fit well in the generally used architecting methods. The use of patterns does not solve all documentation related issues, but they make it easier. (Harrison, Avgeriou, & Zdun, 2007.)

4.3 Microservice design approach

Microservice design is a relatively new cloud based software design approach so the academic research has not yet produced enough empirical data on the development, evaluation and design of applications that implement this design (Aderaldo et al., 2017). The previous research has mostly focused on the strengths and weaknesses of microservices as well as the evolution that the design paradigm has went through (Jamshidi et al., 2018). The idea of a microservice approach was first introduced in 2011 and the driving force behind it was a need to further improve the already in use SOA (service-oriented architecture) approach. Microservice design has a firm base on the SOA approach but it was developed further because a lot of designers felt that it did not fully answer the need for scalability and continuous delivery. (Salah, Zemerly, Yeun, Al-Qutayri, & Al-Hammadi, 2017.) Papazoglou (2003) describes that service oriented architecture is a software integration solution where the software has the ability to offer services to other programs through service application components. Microservices has similarities with the service-oriented architecture but the main difference is the scope and the use of an ESB. Service-oriented architecture scope is enterprise level and it utilises an ESB. Microservices do not utilise an ESB and it is an application level architecture. (IBM, 2019.)

According to Lewis and Fowler (2014) each service is designed to execute a certain task and they share the following characteristics: they can communicate through networks using a commonly used lightweight protocol (e.g REST or HTTP), they can be deployed independently, they are designed to implement the company's business functions and they are separate services that can be implemented in any coding

language that fits best. Because every separate service executes a specific business function, it is very easy to perceive the boundaries between services and understand what the scope of the code (Salah et al., 2017). Due to the loose coupling of components, it opens the possibility to execute continuous delivery of updates. Continuous delivery means that the development team can make all types of changes to the software quickly and safely without disrupting other parts of the program. To execute continuous delivery properly and to ensure the integrity of the program, development teams must monitor the software continuously. Continuous monitoring gives the developers feedback related to performance and possible errors in operations. (Lewis & Fowler, 2014.)

4.3.1 Advantages of a microservice design

The advantages of an microservice approach can be divided mainly into three benefits: faster delivery, improved scalability, and greater autonomy. The benefits are a solution to many issues that companies face with their software in today's contemporary software development. (Jamshidi et al., 2018.)

Faster delivery enables companies to make changes and add features to their product as quick as possible. This is very desirable in the fast-moving global markets of today. Microservices use lightweight containers and are uploaded on the cloud to achieve quick delivery. (Zimmermann, 201.7) Lewis and Fowler (2014) add that a another factor that greatly improves the software delivery time is the automation that is implemented within microservice architecture. The concept of continuous delivery and continuous integration are crucial parts of the microservice architecture. Both methods make use of automation to carry out tests to ensure functionality and enable deployment to production. (Lewis & Fowler, 2014.)

In reference to Jamshidi et al. (2018) scalability can either mean the accommodation of additional users without a drop in performance or the amount of developers that can work on the project simultaneously. In a microservice architecture scalability can be fulfilled on each separate service according to their specific needs and without worrying about interfering with other parts of software. This differs from a monolithic approach where the whole program must be scaled. (Jamshidi et al., 2018.) Developers are divided into separate teams that each develop one service. This improves developer scalability as the scope of what they are developing is limited to that service and they do not have learn the intricacies of the whole system. Teams are also expected to be committed to their respective service for the lifetime of the product. This means that support and development does not end when the product is delivered but continues afterwards as well. It is a mind shift where the service being developed is thought as a product that continuously improves rather than a project that has a set end point (Lewis & Fowler, 2014.)

One of the most appealing aspects of the microservice design approach is the greater level of autonomy that each separate microservice has. Greater autonomy gives the development teams of different services the ability to make localized decisions. (Alshuqayran, Ali, & Evans, 2016.) According to Lewis and Fowler (2014) this greater autonomy is achieved by decentralizing data management and governance as well as using the concept of "smart end point and dumb pipes". Decentralizing data management in the microservice architecture context means that each service has its own database that it manages. Other services can access this data through requests. (Lewis & Fowler, 2014.)

Decentralizing governance gives development teams of separate services more freedom to make decisions on how to build that service. Decisions can be related to what coding language or database to use. This differs from a monolithic approach where the tools being used are standardized across the software. The smart end points and dumb pipes principle supports decentralization by making sure that all the complex logic is done inside the components. The communication with other components is done through simple request and response protocols or by using messaging over a lightweight message bus. If executed properly this principle leads to a more decoupled and cohesive application. (Soldani et al., 2018.)

4.3.2 Disadvantages of a microservice design

The issue with microservice design starts when it is thought as an answer for all the software architectural issues that design teams face. The approach should not be used in situations where the inherit complexity brings costs that outweigh the benefits. In some situations, developments teams use the design in an appropriate situation but do not execute it properly. (Jamshidi et al., 2018.) A crucial step to execute microservices correctly is the division of software into separate microservice components and making sure that software fits the component. The division should be done so that future refactoring wont overstep this division. Additionally, if the components are not constructed according to the "smart end point and dumb pipes" principles, it will lead to ambiguity of where the actual logic is being executed. (Lewis & Fowler, 2014.) Zimmerman (2017) adds that another issue that makes division of software into microservice components difficult is specifying the size of each component. There are varying interpretations of what the size of a service should be and the lack of proper patterns to help with specifications make this task even harder (Zimmermann, 2017). Lewis and Fowler (2014) also point that another important aspect to consider is the expertise of the development team. They explain that executing a microservice design requires that the whole team understands the intricacies that the design possesses.

4.4 Monolithic design approach

The traditional way of designing software systems is using a monolithic approach, in which the whole application is stored in a single codebase and all developers share this codebase (Villamizar et al., 2015). From a structural point of view monolithic applications combine several components to form a single program. The components can be related to authorization, presentation, business logic, database layer or application integration (Esposito et al., 2016). The data management and governance are centralized, which is opposite of microservice design. All technical decisions regarding the programming language and database type must be standardized. From organizational point of view the development teams are divided according to the technology. For example, there are teams that focus on the UI side of the application and another team that focuses on the database. (Soldani et al., 2018.)

4.4.1 Advantages of a monolithic design

The strength of a monolithic approach is that it is generally easier to monitor, debug, test, simpler to develop and deploy. Monitoring monolithic applications is easier because everything is contained within one codebase. You do not have to conduct monitoring throughout several service modules. (Villamizar et al., 2015.) Due to the same reason monolithic applications are easier to debug, as its clearer which components are connected. Testing can also be done in an end-to-end fashion as

everything is in a single unit. Deploying is simple as the whole application is contained within one file or directory. As the monolithic approach is the traditional way of developing programs, most developers are familiar with the methodologies and no time must be invested into training your personnel. (Lewis & Fowler, 2014.)

4.4.2 Disadvantages of a monolithic design

The weakness of a monolithic approach usually appears when teams want to grow the application. When teams want to add functionality or remove it, they must be wary of not unintentionally affecting other components. Scaling is also not without issue as when you scale one aspect of monolithic applications, the whole application must be scaled at the same time. (Taibi et al., 2017.) Villamizar (2015) explained that as you add features to the program the complexity increases throughout the application which stifles innovation and slows down product deployment.

4.5 Deciding between a monolithic and microservice approach

In this chapter we will examine the and criteria on which you should make the design decision between a monolithic and microservice approach. We will examine the issue from several perspectives including experiences when migrating to microservice, industry feedback, infrastructure cost and performance.

4.5.1 Migration experiences

In the context of monolithic and microservice architecture the decision to choose between either option is made either when creating a whole new application or when migrating an application to a new architecture. It is common that companies start out with a monolithic architecture and then migrate it into a microservice architecture once it becomes too complex to maintain and scale. (IBM Cloud Education, 2019.)

According to the industrial survey done by Di Francesco et Al., (2018) related to companies switching their software to a microservice model, migration was usually done in small increments and iteratively. Among the respondents, formal models were rarely used for designing the architecture or to portray the architecture. There were no established protocols or templates to adhere by. Agile development approaches were commonly used. The advantages that the migration brought was the implementation of new functionalities that were impeded by the previous monolithic model or improvement of old functionalities. The challenges were related to the complexity of the architecture and initial implementation of services. Companies struggled with migrating their previous data model to the microservice model, which in turn undermined the decoupling benefits that the architecture brings. (Di Francesco P., Lago P., Malavolta I., 2018.)

In reference to the case study conducted by Balalaie, Heydanoori and Jamshidi (2016) they implemented an DevOps approach in addition to agile methods while migrating their application to a microservice architecture. According to Balalaie, Heydarnoori and Jamshidi the motivation to migrate to a new architecture was the need for reusability, decentralized data governance, automated deployment, and scalability. From their perspective they were able to achieve improvements in scalability and adaptability with their application but faced several challenges along the way. The main challenges they faced included complexity in initial development, relationships between services, lack of service development templates and personnel expertise. The answer to these

challenges was using DevOps practices to bridge the gap between development and operational teams. Using these practices created an overall understanding of the architecture and how to implement it within the personnel. After the migration process, they reviewed the design patterns used to mitigate the practical issues when implementing the architecture. (Balalaie A., Heydarnoori A., Jamshidi P., 2016.)

4.5.2 Industry feedback

The general industry pain points of microservice design can be divided into three distinct categories based on the software lifecycle: design, development, and operation. During the design phase the main concerns are related to architecture and security of the application. When developing the concerns are related to the separate services, storage, and testing. Once the application is in operation the concerns are related to application management, monitoring and resource consumption. (Soldani et al., 2018). Gouigoux & Tamzalit (2017) confirm that in the design phase the pain points in architecture are the overall complexity and size of microservice applications as well determining the size of the separate services. Taibi et al. (2017) point out that identifying the business capabilities that the service boundaries are based on is hard to determine. Determining the API structure between services and designing them in a way that they are used only when needed is also a complex issue. The increased exposure that microservice API brings is a pain point from security perspective. (Taibi et al., 2017.)

In the development stage the difficulty with storage is related to the distributed nature of data in microservices and assuring data consistency. The separation of services increases complexity through the development process which especially affects testing and measuring performance. Measuring user experience and testing interfaces is specifically difficult. (Taibi et al., 2017.) Once the application is in operation the difficulties are once again related to the complexity and size of the architecture. If the design and the separation of services is not executed properly it will lead to increase in network consumption as the API calls between services will slow down the system as well as cause difficulties in managing and monitoring the system. Unclear boundaries and failed isolation between services makes it hard understand which parts are connected as well as how to monitor them. Microservices generate vast amount of distributed logs for each service, so if separation and isolation have not been done properly, it will be very hard to find the source of the problem. (Soldani et al., 2018.) In essence the complexity of microservice design brings a lot of issues that need to be taken into account. With applications that are simpler and do not have to be scaled, it is more efficient to choose a monolithic approach. (Jamshidi et al., 2018.)

4.5.3 Cost and performance

In reference to Villamizar et al. (2016) in the context of cloud web applications the microservice design approach is more cost effective and efficient than the monolithic architecture. Web applications in the cloud that are designed with the microservice approach reduce infrastructure costs substantially and offer higher performance. However, there is an initial cost for companies that have not implemented an microservice design before and must spend resources to learn new practices, methodologies, and processes. (Villamizar et al., 2017)

The clearest performance advantages microservices bring compared to a monolithic approach are the speed of delivery and the ability to scale. In the fast-paced market of today there is an increasing need to deliver quickly and microservices have delivered an answer for this need. In practice microservices typically answer this need by using

lightweight packaging technologies, validated DevOp practices and automation. (Lewis & Fowler, 2014.) Combining these three elements enables companies to deliver services quickly in changing schedules and very little centralized management. In a microservice context scaling can refer to performance or the amount of developers working on application simultaneously. Both types of scaling can be achieved better in a microservice approach than in a monolithic design. (Villamizar et al., 2016.) Each service is a separate unit so they can be scaled during runtime according to the need that is related to that service. Services have their own separate development teams that make localized decisions. Resources and personnel can easily be allocated to specific services according to development needs. Features can be developed in parallel as they do not interfere with other parts of the program. All these qualities bring a reduction in cost and an increase in performance as companies can more accurately allocate resources as well as answer market needs quicker. (Jamshidi et al., 2018.)

4.5.4 Decision criteria

Architecture decisions should be based on the non-functional requirements that are established during the early phases of the design process (Pressman, 2010). However, there are other factors that affect the architecture decision. When deciding on which architecture to choose you should consider the following variables: team size, expertise, simplicity of the application, need to scale and the urgency to launch the application (Jamshidi et al., 2018). We will first examine situations when an architectural decision should be made. Then we will examine the criteria and situations where it is preferable to choose a monolithic approach and then consider when a microservice architecture is optimal.

An important factor to consider is when an architectural decision is made. The two situations where an architectural decision must be made is when you are creating a completely new application or migrating an old architecture into a new one. Companies commonly start out by using a monolith architecture and then refactor into a microservice architecture if there is a need to scale. (IBM Cloud Education, 2019.) However, if there is a definite and recognized need to scale a new application from the start of the development, then it can be wise to implement an microservice architecture if the required expertise is met. The added complexity that the microservice architecture brings compels companies to educate personnel on the execution of the design and create developmental templates to adhere by. (Balalaie A., Heydarnoori A., Jamshidi P., 2016; Di Francesco P., Lago P., Malavolta I., 2018.)

If the team size is small, it is not optimal to choose an microservice architecture as the existing personnel will be split into teams that each work on their respective service. Each team should have various skillsets to implement and maintain the service properly. (Balalaie A., Heydarnoori A., Jamshidi P., 2016.) In order to successfully implement a microservice architecture the required level of expertise throughout the personnel has to be acceptable. A misaligned understanding of how the architecture works and how different services should be split up will lead to an inefficient and hard to maintain application. (Taibi et al., 2017.) If the application that you are developing is relatively simple and does not require high business logic, scalability or flexibility, then microservices is not the right architecture. The trade-off that you get with microservices is that you get high scalability, flexibility and a way to split your application according to your company’s business logic, but the it brings with an increase in complexity that in turn can create issues. (Jamshidi et al., 2018.) Monolithic applications are also quicker to launch as you can deploy it in a single unit. Applications using the microservice architecture must launch in several deployments. In situations where the

application is simple and it is critical to launch the application quickly, the monolithic approach is preferable. (Lewis & Fowler, 2014.)

On the opposite side of the spectrum if there is a need to scale the application and your company has enough capable employees, microservices bring a lot of advantages. They are more cost effective and easier to maintain in the long run. It is easier to scale up separate services according to needs instead of scaling the whole application. The separate teams can make optimal decisions that are confined within the service they are developing. (Taibi et al., 2017.) If there is no immediate need to launch your application, microservices offer easier maintainability in the long run. Correctly implemented microservice applications offer distinct components that are divided according to the business logic of the company. This makes it easier to understand the operation of the application and recognize possible needs to scale certain services within the program. (Lewis & Fowler, 2014.)

5. Findings and implications

Software architecture is based on the non-functional requirements that are established in the early phases of the design process. A proper architecture follows a clear rationale and establishes design rules and constraints that must be followed throughout the development project. (Jansen & Bosch, 2005.) A software architecture conveys the level of modularization and cohesion of the software. A higher level of cohesion can lead to increased complexity and difficulty in maintaining the software as components are either too tightly coupled or they affect each other in ways that are hard to detect. The goal of every software architecture is to answer to the requirements discovered during the design process and create a clear and distinct blueprint of the software and the modules it is constructed of. A successful architecture strikes the correct balance between the right level of modularity and cohesion between modules. (Pressman, 2010.) In the context of cloud-based applications the essential question is what level of scalability is required. Microservice architecture is a very loosely coupled architecture which brings with it a high level of scalability if executed properly. (Lewis & Fowler, 2014.)

The prior research indicates that the microservice approach was an evolution of service-oriented architecture. It takes the concept of offering functionality to other applications through an interface and takes it a step further by omitting the ESB and offering functionality through APIs from each separate service. The underlying factors for a microservice approach were the development of cloud infrastructure and a growing need for high scalability as well as quick continuous delivery. The advantages of having a microservice architecture is high scalability, continuous delivery, reduced maintenance costs and greater autonomy. The drawbacks are complexity of the design and a higher initial investment. Most of the industry feedback on the pain points of the microservice architecture is derived from the lack of expertise of the personnel. A flawed design and understanding of the architecture will create numerous problems in the future. A proper implementation of the architecture yields higher performance and reduction in costs. (Gouigoux & Tamzalit, 2017; Jamshidi et al., 2018; Lewis & Fowler, 2014; Taibi et al., 2017; Villamizar et al., 2015; Villamizar et al., 2016; Villamizar et al., 2017; Zimmermann, 2017.)

According to the research the traditional software architecture used is a monolithic architecture, in which the whole application is contained within one codebase. Monolithic applications utilize modularization, but they do not achieve the same level decoupling as microservice applications. The advantage of having a monolithic architecture is that it is easier to deploy initially, the architecture is easier to understand, and it is commonly used so there is no initial investment to train personnel. (Esposito et al., 2016; Lewis & Fowler, 2014, 2014; Villamizar et al., 2015.)

The purpose of the study was to examine the decision criteria in addition to the functional requirements that companies must consider when either creating a microservice architecture or migrating a monolithic model into one. The criteria were not directly presented but it could be deduced from the issues that companied had when trying to implement an microservice architecture. The criteria are the following: teams' size, expertise, simplicity of the application, need to scale and urgency to launch the application. In the table below you can find the criteria and sources where they were deduced from.

Table 1: Summary of decision criteria deduced from sources

Criteria	**Source(s)**
Team size	Balalaie A., Heydarnoori A. (2016), Jamshidi P. (2016), Di Francesco P., Lago P., Malavolta I. (2018), Soldani et al (2018)
Expertise	Soldani et al (2018), Balalaie A., Heydarnoori A. (2016), Jamshidi P. (2016)
Simplicity of the application	Jamshidi et al., (2018), Soldani et al (2018), Taibi et al., (2017), (Villamizar et al., (2015, 2016, 2017)
Need to scale	Jamshidi et al., (2018), Lewis & Fowler, (2014), Soldani et al (2018), Taibi et al., (2017)
Urgency to launch application	Jamshidi et al., (2018), Lewis & Fowler, (2014), Soldani et al (2018), Taibi et al., (2017)

Balalaie & Heydarnoori (2016) explained that changing into a microservice architecture required the personnel to be split into teams according to the specified services. This separation enabled teams to create smaller agile teams that conformed with DevOp practices. The prerequisite was that each team had the required expertise to run and maintain the service. Having the required amount of capable employees is an important thing to consider before implementing a microservice architecture. (Balalaie A., Heydarnoori A., Jamshidi P., 2016.) Di Francesco, Lago and Malavo (2018) as well as Soldani et al., (2018) had similar results in their research.

The complexity of a microservice architecture brings with it a high initial investment, slower launch of the application and a risk of failing to implement the design correctly. The advantages are higher scalability, reduced maintenance costs and greater autonomy. If the application is simple and there is an urgency to reach the market quickly, then a microservice architecture is not the right design. (Jamshidi et al., 2018; Lewis & Fowler, 2014; Soldani et al 2018; Taibi et al., 2017.)

5.1 Contributions

This study is relevant as the need for scaling applications is growing. Making the right architectural decision can save companies substantial time and resources. To assist on making the decision there should be a decision framework to follow. There is yet to be enough empirical data to create and accurate framework, but this study gives the general outline on what the criteria within the framework should be based on.

5.2 Limitations

The academic literature is behind the industry in regards of studying the effectiveness of the microservice architecture. There is still a gap in gathering relevant empirical information on the performance benefits of the design. There are no clear established guidelines on how to execute the architecture, which in turn creates ambiguity. The best source of academic research is industry surveys and case studies where participants explain how they were able to implement the architecture successfully. The lack of empirical data makes it challenging to specify decision making criteria in a detailed level.

5.3 Future research

Future research should focus on further defining the criteria and examining them on a more detailed level. Definition of a team size, some scale of expertise, level of simplicity and measurement of urgency should be defined. Of course, it is impossible to give universal definitions, but some guidelines would be worth establishing. Additionally, the academic world is behind of the private industry in microservice research and more effort should be placed gathering empirical data on the effectiveness of the architecture.

6. Conclusions

The research method used in this study was a literary review, that consisted of 17 separate studies related to software, monolith and microservice architecture. The research problem was examining the decision criteria that should be considered when choosing between a monolith and microservice architecture. The study was limited by the lack of empirical data that has been related to the subject and because of this it was not possible to determine the criteria on a detailed level. However, it was possible to deduce the decision criteria in a general level from the literature. To recap the decision criteria deduced from the literature are the following: team size, expertise, simplicity of the application, need to scale and the urgency to launch the application

The strength and weaknesses of both approaches are well established but reaching the benefits of a microservice architecture is challenging due to complex nature of the architecture. There are no clear established patterns or guidelines on how to execute the design and the best source of information was industrial surveys and case studies where a successful implementation of the architecture was explained. Some common characteristics with microservice implementations was the use of agile and DevOps methodologies. The main obstacle was creating an overall understanding as well as a division of work within the personnel and these methodologies helped to achieve that. In successful implementations of the microservice design there were performance and cost benefits in the long run.

References

A. Fink. (2020). *Conducting research literature reviews: From the internet to paper* (Fifth Edition ed.). London: SAGE.

Aderaldo, C. M., Mendonça, N. C., Pahl, C., & Jamshidi, P. (2017). Benchmark requirements for microservices architecture research. Paper presented at the *Proceedings - 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE 2017,* pp. 8-13

Alshuqayran, N., Ali, N., & Evans, R. (2016). A systematic mapping study in microservice architecture. Paper presented at the *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016,* pp. 44-51.

Balalaie A., Heydarnoori A., Jamshidi P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Software, 33*(3), 42-52. Retrieved from https://www.scopus.com

Budgen, D. (2003). *Software design*. Essex CM20 2JE England: Pearson Education Limited.

Di Francesco P., Lago P., Malavolta I. (2018). Migrating towards microservice architectures: An industrial survey. *Icsa 2018,* pp. 29-38.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). *DevOps* Retrieved from SCOPUS database. Retrieved from https://www.scopus.com

Esposito, C., Castiglione, A., & Choo, K. -. R. (2016). Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing, 3*(5), 10-14.

Gouigoux, J. -., & Tamzalit, D. (2017). *From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture*

Grant, M. J., & Booth, A. (2009). *A typology of reviews: An analysis of 14 review types and associated methodologies* Retrieved from SCOPUS database. Retrieved from https://www.scopus.com

Harrison, N. B., Avgeriou, P., & Zdun, U. (2007). Using patterns to capture architectural decisions. *IEEE Software, 24*(4), 38-45. Retrieved from SCOPUS database. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-34547120204&doi=10.1109%2fMS.2007.124&partnerID=40&md5=779f5209faedf7b48fa316fca5ed728b

IBM. (2019). *SOA (service-oriented architecture).* Retrieved 08.05., 2020, from https://www.ibm.com/cloud/learn/soa

IBM Cloud Education. (2019). *What are microservices?* Retrieved 08.05., 2020, from https://www.ibm.com/cloud/learn/microservices

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24-35.

Jansen, A., & Bosch, J. (2005). Software architecture as a set of architectural design decisions. Paper presented at the *Proceedings - 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005, , 2005.* pp. 109-120. Retrieved from SCOPUS database. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33947154413&doi=10.1109%2fWICSA.2005.61&partnerID=40&md5=1d42167abc72b89205eec7ec97904ed2

Martin Fowler, & James Lewis. (2014). *Microservices.* Retrieved 07.04., 2020, from https://martinfowler.com/articles/microservices.html

P. Eeles. (2006). *What is a software architecture?* Retrieved 08.05., 2020, from https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html

Papazoglou, M. P. (2003). Service -oriented computing: Concepts, characteristics and directions. Paper presented at the *Proceedings - 4th International Conference on Web Information Systems Engineering, WISE 2003,* pp. 3-12.

Pillai, A. B. (2017). *Software architecture with python*. Birmingham, UK: Packt Publishing.

R.Pressman. (2010). *Ls.*

Software engineering: A practitioner's approach (7th ed.). New York: McGraw-Hill.

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2017). *The evolution of distributed systems towards microservices architecture* Retrieved from SCOPUS database. Retrieved from https://www.scopus.com/

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. -J. (2018). *The pains and gains of microservices: A systematic grey literature review*

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). *Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation* Retrieved from SCOPUS database. Retrieved from https://www.scopus.com

Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., et al. (2015). *Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*

Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., et al. (2016). *Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures*

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., et al. (2017). *Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS lambda architectures*

Wolf, A., & Perry, D. (1992). Foundations for the study of software architecture. *Acm Sigsoft, 17*(4), 40-46.

Woods, E. (2016). Software architecture in a changing world. *IEEE Software, 33*(6), 94-97.

Zimmermann, O. (2017). *Microservices tenets: Agile approach to service development and deployment*

Appendix A. Structure for the research plan

Introduction

When creating cloud applications companies must make a crucial decision on the underlying software architecture. The conservative choice has been using a monolithic software architecture approach, where the application is a single unit that usually consists of a user interface, business logic, data interface and a database. As cloud infrastructure has advanced so has the need for software scalability, flexibility, and agility. The answer to this need has been the microservices software architecture approach, which splits the monolithic model into several smaller units. Each unit is an independent module that has its own functionality and they each cover a certain service. The units have their own databases and they communicate to other units through API functionality. Microservices units can be scaled and updated separately.

Research problem and research methods

Essentially the research problem is to analyse both software architecture models separately and then compare them to each other. We want to answer the question "On what criteria should I decide which design approach I should use?". To answer this question, we must analyse the strengths and weaknesses of both software architecture models and when are they used. We will then form the criteria that should be evaluated when choosing one or the other software architecture. The research method is a literature review.

Limitations

This research will be limited to the design level of the software architecture. The focus will not be on how to execute the architecture on a practical level or how to program certain features. The research will stay on a general level and not examine details of each architecture.

Preliminary earlier research

The need for a scalable software architecture that can be easily maintained has risen after the growth of cloud infrastructures (Alshuqayran et al., 2016). The microservice design has answered this need by creating an architecture that is based on dividing the functionality of the program in separate services that communicate through API calls (Jamshidi et al., 2018). Microservices offer several advantages compared to the tradition monolithic approach. These advantages include scalability, flexibility, and agility. The drawback is that the architecture is considerably more complex and it there is a greater risk of making a mistake in the design. Executing the design properly requires expertise withing the personnel. (Jamshidi et al., 2018) The criteria that the design decision should be based on comes down to team size, expertise, simplicity of the program and need for scalability (Taibi et al., 2017).

List of main prior literature in relation to the background theory

Aderaldo, C. M., Mendonça, N. C., Pahl, C., & Jamshidi, P. (2017). Benchmark requirements for microservices architecture research. Paper presented at the *Proceedings - 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE 2017,* pp. 8-13.

Alshuqayran, N., Ali, N., & Evans, R. (2016). A systematic mapping study in microservice architecture. Paper presented at the *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016,* pp. 44-51.

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24-35.

R.Pressman. (2010). *Ls.Software engineering: A practitioner's approach* (7th ed.). New York: McGraw-Hill.

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. -J. (2018). *The pains and gains of microservices: A systematic grey literature review*

M, Fowler., & J, Lewis. (2014). *Microservices.* Retrieved 07.04., 2020, from https://martinfowler.com/articles/microservices.html

Timetable

Work will be completed during the summer

Preliminary structure of contents

1. Introduction
2. Terminology
3. Research methods
4. Previous research
5. Discussion and implications
6. References

Appendix B. Reference chart

Source	Purpose of the study	Research methods	Background theory	Findings	Implications	Future research
Aderaldo, C. M., Mendonça, N. C., Pahl, C., & Jamshidi, P. (2017).	Benchmark requirements for microservices architecture research	Benchmarking	Microservice architecture, DevOps	The applications analyzed in the study are not mature enough to be used in benchmark studies	Empirical software architecture research must develop further	Creating an ideal benchmark for conducting research for microservice studies.
Alshuqayran, N., Ali, N., & Evans, R. (2016)	A systematic mapping study in microservice architecture	Systematic mapping study, qualitative and quantitative synthesis method	Microservice architecture	Challenges microservice systems face, diagrams used to represent the architecture, quality attributes	Research is behind the industry; challenges are related to design and execution of the architecture	Systematic literary review that researches other architectural factors related to microservice architecture
Balalaie A., Heydarnoori A., Jamshidi P. (2016)	Microservices architecture enables DevOps: Migration to a cloud-native architecture	Case study	Microservice architecture, DevOps	Design patterns and design methods used in the migration process	Establishment of policies that supports application migration to a microservice architecture	Further establish patterns and customer to support migration projects
Di Francesco P., Lago P., Malavolta I. (2018).	Migrating towards microservice architectures: An industrial survey	Survey	Microservice architecture	Industry Experiences with migrating to a microservice architecture	Migrations are done incrementally, agile methods used	Establishing customs and patterns
Esposito, C., Castiglione, A., & Choo, K. -. R. (2016)	Challenges in delivering software in the cloud as microservices	Article	Microservice architecture	There must be a balance between security and performance	There must be different options to improve security of microservice design	How to improve security of microservices
Gouigoux, J. -., & Tamzalit, D. (2017).	*From monolith to microservices: Lessons learned on an industrial migration to a web-oriented architecture*	Case study	Microservice architecture	Switching to a microservice architecture has brought numerous advantages and savings	Switching to a microservice architecture has benefits	Development of agile methods to support switching architectures

Source	Purpose of the study	Research methods	Background theory	Findings	Implications	Future research
Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018)	Microservices: The journey so far and challenges ahead.	Article	Microservice and monolithic architecture, development of software architecture	How software architecture has evolved	The evolution of microservices and the advantages/disadvantages	Future research on the advantages and disadvantages of microservices
M, Fowler., & J, Lewis. (2014).	Microservices	Article	Microservice architecture	Basics of microservice architecture	Theoretical framework of microservice architecture	-
Papazoglou, M. P. (2003).	Service - Oriented Computing: Concepts, Characteristics and Directions	Article	Service-Oriented computing	Extended service-oriented architecture	The usage of service-oriented architecture	-
Pillai, A. B. (2017)	*Software architecture with python*.	Book	Software architecture, programming	How to program software architectures with python	-	-
Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. -J. (2018)	*The pains and gains of microservices: A systematic grey literature review*	Book	Literature review	Industry feedback on the microservice architecture	Most pain points are related to the complexity of the architecture	Systematic review for grey literature
Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., et al. (2015)	*Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*	Case study	Microservice architecture, cloud computing	Benefits and issues with the microservice architecture	It is the right architecture in certain situations	Performance tests

Source	Purpose of the study	Research methods	Background theory	Findings	Implications	Future research
Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., et al. (2016)	*Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures*	Case study	Microservice architecture, service-oriented architecture, cloud computing	Microservice architecture offers cost benefits compared to a monolithic architecture in cloud computing context	Microservice architecture is more cost effective in cloud computing applications	Future performance tests regarding technical concerns
Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., et al. (2017)	*Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS lambda architectures*	Case study	Microservice architecture, cloud computing	Microservice architecture offers cost benefits compared to a monolithic architecture in cloud computing context	Microservice architecture is more cost effective in cloud computing applications	Cost of migrating legacy software to an microservice model
Wolf, A., & Perry, D. (1992)	Foundations for the study of software architecture	Article	Software architecture	Basics of software architecture	-	-
Woods, E. (2016)	Software architecture in a changing world	Article	Software architecture	The evolution of software architecture	-	-
Zimmermann, O. (2017).	*Microservices tenets: Agile approach to service development and deployment*	Article	Software architecture, microservice design, service-oriented architecture	Seven tenets of microservice design	-	-