Department of Computer Science

Distributed and Self-organizing Systems Group

<TODO: Art der Arbeit>

<TODO: Titel der Arbeit>

<TODO: Vorname Autor> mpc

Chemnitz, 20 November 2018

**Examiner:** <TODO: Prüfer>

**Supervisor:** <TODO: Betreuer>

**mpc, <TODO: Vorname Autor>**

<TODO: Titel der Arbeit>

<TODO: Art der Arbeit>, Department of Computer Science

Chemnitz University of Technology, 20 November 2018

Sperrvermerk

Diese <TODO: Art der Arbeit> enthält vertrauliche Daten der <TODO: Name des externen Institutes>. Eine Veröffentlichung dieser Arbeit, auch auszugsweise, ist ohne ausdrückliche Genehmigung der <TODO: Name des externen Institutes> nicht zulässig. Diese Arbeit darf nur den Korrektoren und dem Prüfungsausschuss zugänglich gemacht werden.

Aufgabenstellung

TODO: bei Abschlussarbeiten (Bachelor-, Diplom- oder Masterarbeiten) hier Bitte die ausführliche Aufgabenstellung einfügen.

Table of Contents

[List of Figures vii](#_Toc530495494)

[List of Tables ix](#_Toc530495495)

[List of Listings xi](#_Toc530495496)

[List of Abbreviations xiii](#_Toc530495497)

[1 Introduction 1](#_Toc530495498)

[1.1 Current situation 2](#_Toc530495499)

[1.2 Motivation 3](#_Toc530495500)

[1.3 Problem 6](#_Toc530495501)

[1.4 Objective 10](#_Toc530495502)

[1.5 Outline 12](#_Toc530495503)

[2 State of The Art 15](#_Toc530495504)

[2.1 Requirements 15](#_Toc530495505)

[2.1.1 Requirements of Microservices and Micro frontends 16](#_Toc530495506)

[2.1.2 Requirements of Content trust 16](#_Toc530495507)

[Bibliography 25](#_Toc530495508)

[Appendix A Bezeichner für Anhang A 27](#_Toc530495509)

[A.1 Bezeichner für Anhang A.1 27](#_Toc530495510)

[Glossary XXIX](#_Toc530495511)

[Index XXXI](#_Toc530495512)

# List of Figures

[Figure 2.1 Beispielabbildung **Error! Bookmark not defined.**](#_Toc332899467)

# List of Tables

[Table 2.1 Übersicht vordefinierte Quickstyles (inkl. Tastenkombinationen) **Error! Bookmark not defined.**](#_Toc332899479)

[Table 2.2 Beispieltabelle **Error! Bookmark not defined.**](#_Toc332899480)

# List of Listings

[Listing 2.1 einfaches XML-Beispiel **Error! Bookmark not defined.**](#_Toc332900301)

[Listing A.1 Mit Alt+ F9 bearbeiten (hängt von Heading 4 statt von 1 ab) 27](#_Toc332900302)

# List of Abbreviations

|  |  |
| --- | --- |
| **ABC** | Alphabet |
| **123** | Zahlenreihe |
| **HTML** | Hypertext Markup Language |
|  |  |

# Introduction

Traditional monolithic web application is a software that comprises many parts, all those parts come together to form the final product. But at the end this product will look like as it is made out of one and only one big unit. This unit is composed of few different layers on top of each other but each layer is tightly coupled with the other layers.

Once the application is ready to be deployed, developers will have now to approach it as if it is one coherent part that can’t be divided into smaller parts. Such division can be very useful when there’s a failure and the application is not running as it is supposed to. In this case, developers could isolate the malfunctioned parts. Hence the system will keep providing some of its services to the clients while also being maintained. Moreover, such possibility of parts isolation gives developers more flexibility and features when trying to find solutions for the problems.

Another point, when this application needs to be updated, then developers have to go through the whole source code of the application to introduce the new features. The reason for this is that, the application is very much interconnected and overlapping its parts. Thus it will be very hard to make modifications such as adding new features without having to take into account how such additions and modifications will affect the whole system.

Microservices architecture was developed to overcome the challenges that are imposed when the application is created based on the monolithic architecture. The idea of microservices is to have the system composed of many independent small parts that work together to form the final web application. This concept can also be projected into the frontend part of the application resulting into the micro frontends architecture. In essence, having the frontend as a combination of many small independent micro frontend apps.

## Current situation

Microservices is still a new concept, although some companies have already migrated to the microservices architecture such as Amazon and Netflix [2]. There are companies facing many issues that could be solved by this new architecture. Yet those companies are hesitant in moving forward and migrating their application to the microservices architecture.

The idea of refactoring an existing application into a new one built using another architecture is not easy and brings with it many challenges [1]. Some of those challenges are still waiting to be addressed. Moreover, the process of migration itself is still not clear.

There are still many questions to be asked when thinking about the microservices architecture, for example: when thinking about the functional requirements of the application, developers should think about how their application can be divided into smaller parts. What norms and standards should be followed when making a decision about such division? There can be other questions regarding the nature of the microservice itself. Such as the size of each task and hence the size of each microservice. Moreover the communication between microservices is another challenge that deserves thoughtful consideration. Furthermore, developers should also consider the final product and how can those microservices be merged together.

On the other hand, some concerns related to the frontend part of the application can also be thought about: for example, What if the Frontend becomes also complex and needs to be divided into micro Frontends, How can micro Frontends and microservices work together? Developers should find ways that help Micro frontends to exchange data between each other without violating the isolation and indepence of each micro frontend. Moreover, there’s still an ambiguity regarding the interaction between microservices and micro frontends.

Furthermore, security is a big concern in microservices architecture, a system must be in place to help microservices turst each other. Trust here doesn’t just mean believing each microservice to what it claims it is. But having a system that helps microservices to trust the behaviour of each other.

Questions are many and numerous when it comes to migrating an existing application into microservices architecture. The architecture itself is new and a great deal of research is needed to help those who want to move their application into microservice architecture. Or even if developers want to build their applications from beginning based on the microservices architecture.

There is still no standard definition of microservice architecture and there’s no clear guideline of how an application based on microservices should be built [x]. Over the last few years some characteristics for a microservices-based application have been developed and some general basic outlines are now commonly used.

* A microservices-based application should consist of more than one component: Unlike monolithic-based applications, a system built using microservices architecture should be composed of multiple components, each component is self-contained. This way the application can be changed, updated and modified whenever is needed. In this case, each change will be applied to only the concerned component itself and not the entire application.
* Simple Routing: Components in a microservice-based application will have a simple workflow. They will take an input, process it and then forwards the result. But one should not forget that there are complications and challenges regarding interaction and securities between microservices.
* Decentralization: An application based on microservice architecture is built out of many different components and each has its role. Yet, the application in total is not one unit and there’s no on big unit moderating the communications between the microservices.
* Different technology stack: The development cycle of a microservices-based application involves having different teams working on different microservices. Since each microservice is a self-contained component, each team can then choose the most suitable development technologies and tools that are most suitable for their own microservice.

## Motivation

The current architecture that’s used heavily in building web applications is composed of layers built on top of each other. Each layer is responsible for a field or a big part of the application. How many layers an application can have is still dynamic and up to the developers to decide. This architecture is called monolithic architecture. It usually consists of three layers on top of each other’s, the User Interface, then underneath comes the Logic Layer and finally the Backend [2]. Moreover some applications could end up having their logic layers divided further into more layers.

Although a monolithic web application is divided into layers, yet the application is still very tightly coupled [16]. There’s a great amount of connections between the layers.

As a consequence, the system, of course, will be hard to maintain and update. Each problem in the Backend will make the system useless, where no information could be processed or stored in the database. On the other hand, each problem in the User Interface will make the application unusable, and data can’t flow from the frontend to the backend.

Finding the problem when the system fails is a big challenge. Each part is one big unit and in order to find why the system fails many tests should be run and sometimes thousands of lines of codes should be reviewed. This situation could lead to even a worse one. In some cases fixing the issue is not possible until some modification is performed to suit the new changes. This leads to the problems of updating the system.

Updating a monolithic-based application is another big problem faces developers and business owners [x]. When needing new features or updates to fix the newly discovered issues, developers could find themselves trapped between tow situations: Not being able to achieve the wanted updates and fixes as much as they’re supposed to. And on the other hand, having to perform huge system modifications and changes to accommodate the new desired features.

This kind of situations happens frequently and usually costs money, time and lots of efforts. Microservices architecture was developed to make applications more flexible. With microservice-based application, the system is now more accepting for changes. Developers don’t need to make great modification for the system to adapt a new feature. When a system failure happens or when a problem is discovered, developers have the ability to isolate the problem and fix it quickly.

Under microservices, the application is not just one big unit, nor is it a three-layer architecture. The system is a groups of smaller systems. Each small system is a self-contained unit. Those units can be developed and deployed independently. Moreover, each unit can be developed by a different development team. Hence creating a team of small number of developers becomes a more possible choice.

The concept of microservices is further projected into the other side of the web application. Microservices is not just concerned with the backend side of the system, but it can be also applied to the frontend as well. When microservices concept is applied to the frontend it is called micro frontends [3].

With micro frontends, the frontend of the application is composed of many different small self-contained applications. Instead of having the frontend as a one unit written in one framework such as Angular or React JS. The frontend can be written and developed as a sum of smaller frontends. Each small frontend performs a very specific task and can be called or rendered whenever it is needed.

The point of creating the frontend based on the micro frontends architecture is to make the application more flexible and more adaptable. A frontend could start having problems once the application requires more features. In such case, developers might need to apply many alternations to help the system be able to accept the newly added features. When a problem is discovered, sometimes big revisions and improvements are needed to get the system to the desired working state.

In micro frontends architecture, each part of the application is developed independently, as a standalone application. This means that developers could choose Angular for one micro frontend and then choose React JS or Vue.js for another part. Each development team can focus on their micro frontend. This division also helps with separation of concerns. Designers of the system will be able to divide it into small tasks. Then assign the development of those small tasks to different teams. At this point, each team will not have to bother with interacting with other teams. They could focus on their task, its functionality, problems, and its deadline.

Microservices and micro frontends architecture helps to solve many pressing problems for the development of web application. At the same time such architecture brings its own challenges and problems.

Putting all the small parts together is not an easy task. Developers should think of the best way of having microservices interact with each other. Not just in the sense of sending and receiving information but also microservices should be able to exchange sensitive data. A system of trust between microservices could help make the interaction more secure.

Such trust is very important when there is a need for using third-party microservices. In this case microservices maight end up exchanging sensitive data such as user logins and passwords or maybe even bank details. In such case microservices should not start exchanging such data without veryfiying and knowing more details about the microservices on the receiving end. Here a system of content trust can play a great role. It can insure that exchanging data between microservices only happens after the behaviour of each microservice has been verified. Once it is known that no microservices have any harmful code, or bad intentions then data exchange should take place smoothly between microservices.

## Problem

Having microservices architecture in its early days means that not many resources are available. Moreover, not enough research is done yet to help developers find answers for their problems [x]. When dealing with microservices architecture two types of scenarios could be happening:

One case would be that, the Web application already exists using monolithic architecture. But there is a need to migrate it into microservices architecture for many reasons including:

* The application is not scalable anymore: New features are needed but scaling the application to fit those new features in is not possible without making big alternations to the application itself.
* The application has some serious problems: Developers are not able to find suitable solutions for those problems without creating many additional parts or performing huge changes to the base code.
* Some parts of the application fails constantly causing the whole system to fail: In this case developers are not able to isolate the failing parts. Thus whenever a failure happens the system does not continue to run while the problem is being fixed.

The other case would be that, developers wants to develop the required system from the beginning based on the microservices architecture. One reason for this can be because the application is expected to grow. With microservices architecture, it is easier to scale the system as much as needed compared to monolithic architecture [x]. Another reason would be that the application has a complex nature and requires different technology stack for its various parts. Hence building it using microservices architecture will help developers to use different tools and technologies for different parts. This situation also applies for the frontend. In this case, the frontend could be complex and many special requirements are needed thus developers can divide it into smaller apps. With each app they can use different tools, frameworks and languages instead of having to use one toolset or one framework for developing the whole frontend.

Microservices architecture is basically one variant of Service Oriented Architecture (SOA) [x], but operations conditions are not quite the same as with traditional SOA. With microservices architecture there are many small independent parts operating. Each part is providing or performing one small task. Sometimes microservices need to exchange data with each other and of course with the frontend.

Developers have to decide what kind of communication methods and protocols should be used among microservices. Representational State Transfer (REST) is one architecture that helps developers when creating web services. Another option is Simple Object Access Protocol (SOAP) which is a protocol for exchanging messages in a web services environment. There are many technologies that could be used in a SOA Including:

* Web Processing Service (WPS)
* WSCL - Web Services Conversation Language
* XML-RPC - XML Remote Procedure Call
* JSON-RPC

The most famous ones and most used is REST and then SOAP [x]. The decision for using a specific protocol depends on the developer and how they want data to flow between the different parts of the application.

On the other hand when developers decide to use micro frontends to render the frontend of the application. They should also think of how data is going to transfer between the different micro frontends and how communication between the many frontends and the microservices is happening.

In essence, micro frontends are autonomous independents parts [x]. These parts are not divided according to how they’re going to render on the screen. They’re divided according to their functionality or even the business they support. For example, in an online store, the distribution of micro frontends could be as follows: One micro frontend for the displaying of the products, another would be for the product details, a third micro frontend could be for adding items to the cart. And maybe a fourth for the check out and payment. Other micro frontend can be focused on other functionalities of the web site such as handling user data. For example, registration and creating a new account. This task can be assigned to a micro frontend, or updating user information. One micro frontend can also be created to handle security concerns such as when a user forgets his passport.

Obviously, the division here is concerned with the functionality of each part of the website not how those parts are outlined and organized in the website.

At the end once each micro frontend is developed, the final frontend should be able to contain all the micro frontends when they are needed. There are different technologies for stitching micro frontends together. Two approaches are available for putting all the parts together, either client-side or server-side. For each method there are different tools and technologies available, each has its own good points and negative ones. On the server side, such technologies include:

* NGINX as reverse proxy
* Varnish with ESI module enabled

While on the client-side, developers could use one of the following technologies:

* Single SPA library
* Web Components
* iFrame

For both methods there are more technologies and tools than the previously mentioned ones, but those are the most famous so far. Choosing the right technology depends on the application being developed and if there’s a need for communication between the micro frontends or not.

To give an example, iFrame which stands for inline frame works in a way that enables developers to include an HTML document inside another one. iFrame is not a new technology and it is known since the early days of HTML [x]. When developers decide to use iFrame to bring together their micro frontends, they will face big challenges. In particular, when there’s a need of communication among their micro frontends. iFrame isolates each part and developers should find a way to overcome such isolation.

The problem in the case is which technologies to use? And whether or not such technologies will need additional development to satisfy the needs of the application in hands.

Moreover, security concerns are a big challenge under microservices architecture. How would microservices exchange particular sensitive information such as passwords or bank details? This concern would be stronger if microservices were not all developed inside the same company. Such scenario could rise when small companies want to build their applications using microservices architecture. In this case, when having a small team of available developers, one could think of using ready-made solutions. Developers could use already written microservices to save time and money. But doing this imposes more security concerns. Hence microservices need to establish a trust mechanism to trust each other before exchanging data. Moreover, when adding new microservices to an already running system. The existing microservices need to be able to trust the newly added ready-made solutions? The kind of trust discussed here is related to the behaviour of each microservice. After all, malicious or harmful microservices could hide their true intentions by expressing different behaviour while a harmful one is practiced behind the scenes.

This is not the only concern here, since microservices have to authenticate themselves to each other. Each microservice must make sure that it is handing over the data to the right microservice not another one. Once authentication is performed then the actual behaviour of each microservice is still in question. There should be a way to help microservices trust each other without having a human intervention. When the application is getting bigger and embracing hundreds of microservices. Which in turn might also be depending on other microservices to run, then checking the behaviour for each one of them by the developers could end up being an endless task. Developers could start checking the microservices they adapted into their applications. But then shortly find themselves checking microservices that are used by the microservices they used. Hence keep moving backwards in the string of microservices.

Such situation is not what the solution should be, on the contrary, each microservice should be doing its own task of verifying its behavior with other microservices. When one microservice fails in doing so, then it is the decision of the developer to either cancel using it or to intervene and check what is causing the problem.

## Objective

This thesis will try to fill the gap that still exists in the literature regarding some of the mentioned concerns. The focus will be on building a system out of microservices and micro frontends while providing a solution for the security concerns, in particular content-trust among microservices.

Microservice architecture is still growing in the sense that it has not yet a well-known agreed upon definition. There is still no common understanding of how small each microservice should be. Some resources consider looking at the source code and making sure that it doesn’t exceed hundreds of lines. Some researchers think that the development time for one microservice should not exceed two weeks of work. Another view point is that developers should keep dividing the tasks they have until no further division can be applied. This view point argues that each microservice should be concerned with one task, and one task only. Once the task can’t be divided anymore, developers should stop and assign the task to a microservice [1].

Not everyone agrees with the above suggestions. For example, having the microservices as small as possible means that the application will end up having many small apps working together. The more moving parts an application will have the more overhead there can be for developers to put everything together. On the other hand, the more small apps an application have, the easier it is to solve a problem.

When the application have many small apps forming the final product, developers will have better circumstances when problems and failures happen. Once a problem is detected, developers could then easily isolate the broken part of the application, in this case, one or more small apps. Being able to isolate the dysfunctional parts will give a better chance for the whole system to keep running and offering its services to clients while the problem is being fixed.

Another arguable concept is the method of communication between microservices. Microservices are supposed to be self-contained autonomous small apps [1] but at the same time they should offer an interface for their clients to communicate with them. Their clients are often other microservices trying to send data or request data from them. This scenario contradicts with having microservices as independent small apps. Should developers strive to make each microservice as independent as possible? Or should they try to separate microservices from each other while offering a good mechanism for intercommunication?

Some microservices will, of course, need data from other sources. Maybe some of those microservices are only responsible of receiving data, processing it and passing it forward. For such microservices they should be able to communicate with other microservices to achieve their tasks. Yet, they should not need to communicate with other microservices to help them with the processing of the data. The point of argue here is whether or not processing of the data should be done completely inside on microservice. If one microservice can’t handle the processing by itself, should it be divided into more microservices where the intercommunication will help with the processing? Or should it be combined with other microservices to handle the task internally completely?

The above mentioned points appear on the surface of the discussion because microservices and micro frontends as well are both still in their early days and not many researches have been done to try to find answers or suggest solutions for such problems. Most of what developers could find in resources would be high level suggestions that don’t go deep into each problem.

Of course, there are many more problems associated with microservices architecture. And even many more will face developers if the decide that their frontend should also follow the same architecture and be divided into small parts. The application itself could also be error-prone and vulnerable for security threats. Security concerns should be given adequate attention when designing a microservices-based architecture. When the application has many different parts, each part should be well known to the other parts. There should be a mechanism that prevents small apps from pretending to be something while acting in a different way behind the scene. Such behaviour is risky and could lead to lose of sensitive information. A system for behaviour check could help microservices to verify the behaviour of other microservices. Hence make sure that any microservices with hidden harmful intentions could be detected and isolated.

This thesis will try to find answers to some of the questions mentioned above. While there are many questions and uncertainty to explore and research, this thesis will not try to find answers for every possible problem could result from building microservices-based web applications. The workflow will be the building of a Blog based on the microservices architecture, the development will involve using the latest technologies and tools to build the Blog. Solutions for faced challenges will be documented which then can be used by researchers, students or anyone interested in the microservices architecture. It could also be used by developers who are building a microservices-based web application to help them overcome some of the challenges that could face them.

Moreover, this research will also focus on providing a solution for security concerns, more specifically, the problems of content trust among microservices. A method will be created to help microservices trust each other context-wise. This trust is not about verifying each microservice its identity to the other microservices, but it is about having a mean or way of verifying the behaviour of each microservice. Such method could be very useful when developers have to add different microservices from other sources and have to check their behaviour and trust it.

On the other hand, this thesis will not try to provide a full workflow and complete guidelines for building microservices-based web applications. Such attempts requires years of research and will most likely be a never ending one. Since tools, frameworks and technologies are ever changing and developing and so are the ways of developing web applications.

## Outline

The following chapters will talk more in depth about the microservices architecture and micro front ends. The current situation and state-of-the art will be discussed regarding the currently used approach, technologies, tools and theories. The researched topics will talk about dividing an already existing monolithic application into a microservices-based application but the focus will be more on building microservice-based application from the ground up.

The next chapter of this thesis will be state-of-the-art, in this part, the research will focus on exploring and presenting the used practices when building microservice-based applications. The concept of dividing requirements into small independent tasks will also be examined. Used tools and technologies will be explored, and communications among microservices will be studied. The used methods of data exchange among microservices as well as the communications with the frontend will be researched.

Just as with microservices, the topic of micro frontends will also be researched. The focus will be on how to divide the frontend into smaller parts and how those parts can be brought together, and how can they exchange data among each other.

Furthermore, content-trust will be studied. The research will study the state-of-the-art of the web content trust and will explore how such principles and concepts can be projected into the microservices architecture to help microservices trust each other.

Third chapter will focus on the concept of building microservices and securing their behavior with content trust. This part of the thesis will try to weight the benefits as well as the negative sides of building applications based on the microservices architecture. It’ll investigate how a system with many moving parts can work and offer a stable and seamless experience to its users. At the same time have very clear division and separation of functionalities into small autonomous collaborating tasks. How content trust can play a role in the microservices architecture will be provided. A method of content trust among microservices will be discussed and inspired from the content trust of the web. The discussion will also pay attention to the principles of micro frontends and the different methods, server-side as well as client-side, of combining micro frontends will be presented.

Chapter four will focus more on the practical side of the research. This chapter will discuss the development and building of a Blog based on the microservices architecture. The workflow will be presented and mistakes that have been done during the development will be discussed to help make other researches aware of them. The used tools will be explored and the reasons behind using such tools will be made clear.

The developed method of content trust among microservices and micro frontends will be presented, how this method is developed will also be outlined and discussed.

Last chapter is number five, in this part of the thesis evaluation of the development and carried out research will be performed. This chapter will show the negative as well as the positive sides of the research and the implementation. It’ll also discuss the difference in the productivity when using specific tools or technologies. It’ll show the similarities and dissimilarities when selecting specific tools over others.

# State of The Art

This chapter will be composed of three parts:

1. Requirements Analysis
2. Literature/State of the Art Review
3. State of the Art Analysi

The first part will discuss and analys the requirements for a web application based on the microservices architecture. It’ll also discuss the requirements of content trust between microservices. Hence it’ll be composed of two sub sections. One for microservices and the other for content trust.

The second part will layout the literature review for microservices and also for content trust. As with the first section, this one will be mainly divided into two sections as well. One for microerservices and the other for trust in general and content trust.

Last part of this chapter will be connecting the first two parts together. It’ll check the literature review against the requirements analysis.

## Requirements

This part of the thesis will present requirements analysis for a microservices-based architecture that relies on content trust to verify the behavior of its different components. The discussion will be split into two parts:

1. Microservices and micro frontends
2. Content trust between microservices

The reason for such division is that, microservices and micro frontends are architecture. Hence they have their own set of rules and requirements when it comes to building an application based on this new architecture.

On the other hand, content trust is one of the security concerns, and with just like any application, security concerns will be addressed throughout the development of the proposed Blog. Yet the focus here is on the trust among different microservices, especially those coming from different sources/developers. Thus, this chapter will allocate a subsection in the requirements analysis to discuss the requirements analysis of the content trust of the application.

### Requirements of Microservices and Micro frontends

### Requirements of Content trust

This section will discuss the requirements for a content trust system that will be implemented to help microservices verify the behavior of other microservices. The following pages will try to give more details about what it should be included in content trust system.

Microservices themselves need to have clear rules about how to communicate with other microservices especially ones coming from different developers. This communication should allow microservices to make a judgment of whether to trust the other microservices or not.

On the other sides, there are no requirements for the user, the one who is going to use the overall application. Because, content trust is something to be handled between microservices, or a protocol and set of rules to be read and understood by the microservices themselves. And so the user has no role in how the content trust should be and what it should include.

A system based on microservices architecture consists of different small pieces of code. Each small piece is an application that can be deployed independently. It can also be updated and modified while no need for any modifications of the other small apps because such an update took place for one or more microservices. Such architecture, in theory, makes the system loosely coupled. Thus different system parts and components are easy to change, update, modify or even replace. As long as the interface of the new introduced microservices respects the old interface or keeps using the same communications protocols, then the system will continue to function.

Just like any other system, applications based on the microservices architecture are vulnerable to security threats. This vulnerability comes from the traditional security challenges that face any other system, but also from the nature of a microservices-based application.

When developing microservice-based applications, security concerns should be addressed. Each microservice should be able to verify that the other microservice that’s trying to connect with it is what it claims it is. Then once this established each microservice should be able to move on and do other checks.

The reason for this verification is because microservices are written in a way makes them easily deployed and modified. The whole idea of the microservices architecture is built on creating a very flexible system. Such system will continue to run when one or more of its microservices are not functioning. It will also continue to run when developers decide to make changes. Such changes could be that one or more microservices need update, some alternations or even a complete replacement.

Here at this point, when introducing new microservices to the system, the other older microservices should be able to have a good security system. Such system will allow older microservices to check the identity as well as the behavior of the newly added microservices.

This mentioned scenario is the normal case. It is ok and even expected that developers of a system will make changes once the system is deployed. Especially in the testing phases when developers are running beta versions of their newly developed application. In this stage developers could continue to make changes. Hence microservices are not supposed to exchange data blindly with other microservices.

The fear of hacking into the system is another situation that comes also from the nature of microservices. Such hacking could happen by replacing one microservice with another. The newly added microservice could successfully have a similar interface, allowing the other microservices to exchange data with it. In such case if unwanted party get to access and replace one or more microservices, they could also be able to access sensitive information exchanged with the new implanted microservices. Where those added microservices could send any information exchanged with them outside the application and thus putting the security of the clients at risk.

Content trust has more dynamic nature than other types of trust such as policy-based trust. While when performing an identity check the outcome could be one of two. Either the identity has been proven, or the identity failed to prove itself. But with content trust several characteristics, measures and aspects should be taken into account to come to a decision of trust or distrust. Those measures are inspired from web content trust [17].

Those measures differ in how hard it is to obtain the needed information. It all depends on the kind of information is being retrieved. In order for microservices to be able to check the content trust of one another each microservice should be able check the following:

* The identity of a microservice
* The type of service a microservice provide
* How sensitive the service provided by the microservice
* How many other microservices interact and use the services of a certain microservice
* The evaluation of content trust of a certain microservice by other microservices
* Age of the microservice in concern in relation to other microservices
* The relation of the microservice in concern with other trusted microservices
* The interface and the nature of the connection with the microservice

**The identity of a microservice**

When an application starts using third-party microservices then there should be a need for verifying the identity of microservices.

A system must be in place to help microservices verify each other and make sure at first that each microservice is what it claims. Verification is not an essential step for content trust but if verification was available. Then before proceeding to the contextual check, each microservice will have the opportunity to declare itself to others.

If a verification mechanism is implemented as a part of the content trust system, then the evaluation process will be handled sequentially. The system at first must allow for microservices to verify themselves to other microservices. If the verification process was successful, then microservices move on to the next step. In case the verification step failed then the involved microservices can’t move further in the process and any planned communication will not take place. Moreover such scenario could actually lead to further steps. The system could include a notification mechanism. Those notification will signal the ID of the involved microservices and which one failed in authorizing itself. Hence developers can take notes of what happened and make sure that all the microservices of the system are secure.

It is not mandatory that microservices stop any communication when identity check fails. Microservices can always proceed to the rest of content trust check without going through any identity verifications. But having an identity verification system implemented will help to make the system more secure. Hence a discussion of the different possible verification methods is presented.

Verifications of microservices can be implemented in different ways, the following methods will be discussed:

* One-to-one verification
* Centralized verification

**One-to-one verification**

When one-to-one verification takes place, each microservice will have the opportunity to exchange identity information with another microservice. This exchange of identity information could happen only once. The very first time two microservices are trying to connect with each other.

When a microservice is sending a request to another microservice, if both microservices never connected before then a protocol of identification should be followed. Such protocol could allow each microservice to check the identity of the other microservice and make sure that it is exchanging data with an authorized microservice.

**Centralized verification**

Such approach can take many forms. One way of doing a centralized verification is by having a central microservice responsible for the verifications of different microservices. This approach is simple, the authentication microservice will have a database that contains certain secure information about each microservice. Once a microservice wants to connect with another microservice, it should first contact the authorization microservice.

The authorizing microservice will accept any contact request from any microservice, then will try to check its log in information against the data it has already stored. If there’s a match then this microservice identity has been successfully identified. Hence such identification could result in a better evaluation when other microservices are trying to make a decision of content trust about this microservice.

This verification could actually play a role in the content trust and the decision of trusting the behavior of one microservice or not. In this case if a verification system is implemented and the verification step is passed successfully. Then an extra value could be added to the microservices which successfully authorize themselves to other microservices. Of course such consideration depends on the final system that’s implemented for the content trust.

**The type of service a microservice provide**

Microservices should be able to check the service that’s provided by any microservices trying to connect with them for the first time. The more sensitive the provided service is the more important such check. There can be different ways that allow one microservice to check the service of the requesting microservice. For example, there could be a record that lists the services provided by each microservice with the identity of each microservice. Once a request has been made, the requested microservice can check the request information of the requestor microservice against this record. It will then be able to verify that the service of the requestor microservice is what it claims to be.

Another method could be done by running a simple test with the requestor microservice using dummy data to first check that it’ll run as it claims to be. This might need some form of artificial intelligence. It will enable each microservice to truly verify if the requestor behavior is as it is supposed to be or not.

**How sensitive the service provided by the microservice**

The services provided by the different microservice will vary in nature, some will offer simple service, while others will offer sensitive services. For example, some services could offer routing to help the user navigate from one page to another. On the other hand, another microservice will offer a log in service to the user. Such service has a higher sensitivity than the other.

**How many other microservices interact and use the services of a certain microservice**

Each microservice should be able to know if there are other microservices that used the service of the requestor microservice or replied positively to its requests. Also each microservice should be able to know if the requests of the requestor microservices have been denied or not. And which microservices denied the requests of the requestor microservice. In case of failing or negative replies to the requestor microservice, then it’ll also be helpful to know which microservices responded negatively to the requestor. And what kind of relationship exist between the requested microservice and those other microservices that responded negatively to the requestor.

**The evaluation of content trust of a certain microservice by other microservices**

When a microservice receives a request from another microservice, it should be able to see what evaluation the requestor microservice received from other microservices. This evaluation will be the degree of trust each microservice gave to the requestor. By seeing what other microservice thinks of the behavior of the requestor microservice it’ll then help the requested microservice to increase its own positive or negative evaluation of the requestor.

**Age of the microservice in concern in relation to other microservices**

Microservice in the system might have different operation age, this difference come from the nature of the architecture of microservices itself. Microservices are added gradually to the system. Hence some will be added in the early stages while other will be added at a later stages. Moreover, some microservices will be replaced by new microservices. And some new microservices will also be added to fulfill new requirements or fix a newly discovered bug. When making requests to other microservices, the requested microservice should be able to check the age of the requestor. The older the requestor the ore trust the requested microservice could have about the requestor. Of course, there should be a very trusted method of checking the age of each microservice by the other microservices.

**The relation of the microservice in concern with other trusted microservices**

The relationships between microservices could play an important role when deciding which microservices to trust or not. A transitive system could be in place to help transit the trust of a microservice from one to another. For example, three microservices A, B and C. the content trust relationship between them is as follows: A trust B, and B trusts C. the two microservices A and C have never interacted before. When C makes a first request to A. the microservice A should be able to check the microservices that it already trusts and their trust decision regarding C. Since A already trusts B, and B already trusts C, then A could come to a positive conclusion and increase its evaluation of trust to C based on its trust to B which in turn trusts C.

**The interface and the nature of the connection with the microservice**

Microservices have interfaces that help them interact with each other and exchange information, this can be in a form of an API calls over a protocol. Such interface could be helpful when deciding to trust a microservice or not. When microservices have similar methods for API calls and use the same data type and containers for exchanging data such as JSON or XML then this could be considered a positive point when deciding to trust the behavior or a microservice or not. The reason for this is because if the microservice was installed by the same developers then it is more likely that they will try to unite the way microservices interact with each other inside their system.

When microservices have a system that helps them in identifying other microservices and also helps them in making a decision about the behavior of other microservices. Then the system will exhibit an intelligent behavior that will help in detecting and eliminating security threats. In order to obtain a decision about the behavior of other microservices, each microservice must rely on different set of factors and guidelines. These factors and norms will help microservices come to a decision of whether they should trust another microservice making a request and trying to exchange information with them.

As can be seen, content trust requirements of microservices are a combination or policy-based trust and reputation-based trust. Microservices have to make a decision of trust the content and behavior or other microservices based on a combination trust systems.

The other involved parties in a microservices-based applications are developers and users. Developers are the person or the group of people who are creating the application. From their point of view when trying to handle content trust between microservices, there can be two cases:

* All microservices are developed in-house
* Some microservices are developed by a third-part

When having all the microservices as an internal product, something developed by the same company then trusting the behavior of different microservice is not as important. The reason for this is that when developers develop a microservice they’ll be able to trust its behavior directly. They will be sure that no hidden intentions are implemented or any harmful behavior intentionally in place. Furthermore, some concerns should be taken into consideration if in the future some microservices will be introduced from a third-party.

On the other hand when some microservices are developed by a third-party, then developers must make sure that microservices of both sides will be able to communicate with each other to verify their behavior and build their trust. Developers should make sure that any third-party microservices should be able to provide the requested information. This information such as identity verification, age of operation, the type of service provided, and so on. Those information will help microservices to make a decision of whether they should trust the behavior of a certain microservice or not. Failing to do so while having a content trust system implemented could create many problems.

On the other hand users of the system don’t have any content trust requirements. The reason for this is because content trust is something related to the interaction between microservices themselves. It is all internal behavior.

# Bibliography

[1] H. Gebhardt, “Dezentrale Autorisierung in,” 2010.

#### Bezeichner für Anhang A

##### Bezeichner für Anhang A.1

<xml>

<element id=”guid”>example</element>

</xml>

Listing ‎A.1 Mit Alt+ F9 bearbeiten (hängt von Heading 4 statt von 1 ab)

# Glossary

Glossarbegriff

Im Glossar können ausgewählte Begriffe genauer definiert werden…

HTML

Bei HTML (Hypertext Markup Language) handelt es sich um eine Auszeichnungssprache …

# Index

**No index entries found.**

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 20. November 2018

<TODO: Vorname Autor> mpc

TODO: Es wird empfohlen die offizielle Selbständigkeitserklärung des ZPAs zu verwenden: [http://www.tu-chemnitz.de/verwaltung/studentenamt/zpa/formulare/ Allgemein/allgemein/selbststaendigkeitserklaerung.pdf](http://www.tu-chemnitz.de/verwaltung/studentenamt/zpa/formulare/%20Allgemein/allgemein/selbststaendigkeitserklaerung.pdf)

Für weitere Hinweise siehe Abschnitt **Error! Reference source not found.** „**Error! Reference source not found.**“