Aalto University
School of Science
Bachelor's Programme in Science and Technology

Security in Microservice Architecture

- Impact of a Switch from Monolith to Microservices

Bachelor's Thesis

xx. xxxxxkuuta 2020

Tommi Jäske

Aalto-yliopisto
Perustieteiden korkeakoulu
Teknistieteellinen kandidaattiohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Tommi Jäske
Työn nimi:	Turvallisuus mikropalveluarkkitehtuurissa - Monoliitisesta arkkitehtuurista siirtyminen mikropalveluarkkitehtuuriin ja sen vaikutukset.
Päiväys:	xx. xxxxxkuuta 2020
Sivumäärä:	?
Pääaine:	Computer Science
Koodi:	SCI3027
Vastuuopettaja:	Professori Eero Hyvönen
Työn ohjaaja(t):	Professori Tuomas Aura (Tietotekniikan laitos)
Kirjoitetaan myöhemmin.	
Avainsanat:	avain, sanoja, niitäkin, tähän, vielä, useampi, vaikkei, niitä, niin, montaa, oikeasti, tarvitse
Kieli:	Suomi

Aalto University
School of Science
Bachelor's Programme in Science and Technology

ABSTRACT OF
BACHELOR'S THESIS

Author:	Tommi Jäske
Title of thesis:	Security in Microservice Architecture - Impact of a Switch from Monolith to Microservices
Date:	MonthName 31, 2020
Pages:	?
Major:	Computer Science
Code:	SCI3027
Supervisor:	Professor Eero Hyvönen
Instructor:	Professor Tuomas Aura (Department of Computer Science)
Will be written.	
Keywords:	key, words, the same as in FIN/SWE
Language:	English

Contents

1 Introduction

In recent years, mobile applications and web services which cater to them have revolutionized our daily lives by infiltrating social life, shopping and almost every aspect of our existence. The rapid expansion and, at times, even faster decline of these web services need a matching architecture to meet these very specific needs.

There are many web services already in use which were designed and implemented before the onslaught of microservices. Some of these services have already made the switch such as Netflix but this is not the case for the whole industry. Also, when a new service is being created it does not make sense to start with microservice architecture. This is due to the fact that microservice architecture is more suitable once the domain has been established.

There are many web services already in use which have been designed and implemented before the onslaugh of microservices. Some of these services need to evolve to be of use in the future. In many cases the monolith services have already started to use certain aspects from the microservice world, such as access tokens and REST API:s. The pressure from new competitors adopting new technologies right from the start and the fact that the industry and its developer base are extremely young dictates that the old and established services have to address the situation someway or the other. Monoliths have served us well but the time has come to evolve with the customer needs.

When new development is carried out by a startup the initial architecture might still be a monolith one. Newman (2019) states that due to limited resources a monolith might be a better fit to these companies trying to navigate to the actual product they are to offer. In the case of success the need to rapidly scale the offering emerges. Newman (2019) refers to these companies as ”scale-ups”. Newman (2019) also states that it is much easier to refactor an existing service than to create a new one and thus the need to split monoliths to microservices is and probably will be relevant to the near future.

Kalske et al. (2018) finds that as the codebase becomes large the MA leads to slower development. This is due to the possible complexity inherint in the entwined monolith. The number of placess to refactor is much larger than in a small microservice. Microservice should do one thing and as such it should be more understandable.

The Stack Overflow annual survey (Stack Overflow) conducted on developers found that half of the respondents identified as full-stack or backend developers. Of the respondents 40% had less than five years of professional experience.

New developers entering the workforce have a very different mindset than the older more seasoned professionals. Thus, it is very clear that the ways of working and paradigms to be used are constantly changing.

Microservices are not the proper choice for all web services (Newman, 2019). Microservices offer multiple benefits such as easier scalability and more modular structure for the application. When the architecture needs to be changed the process needs to happen in an orderly and safe way. Often overlooked security aspects need to be addressed and identified as early as possible.

Microservice Architecture (MSA) differs in many ways from the more traditional Monolithic Architecture (MA). This shift entails very specific security issues.

In this thesis the MSA and security literature is evaluated and the main differences between MA and MSA on security aspects are found.

The first chapter discusses the ... The last chapter in the thesis contains the conclusions and presents further research topics.

2 Architectural Comparison

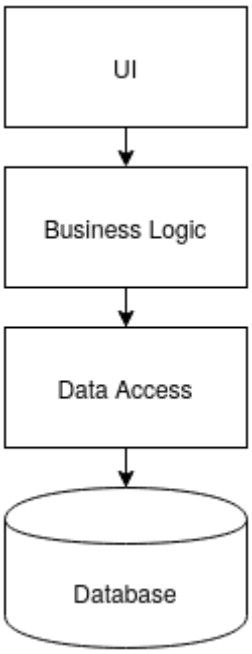

Figure 1: Traditional Monolithic Architecture (Kalske et al., 2018)

MA can be visually bresented as in figure 1. The web service is a layered structure in which all of the different layers have a specific task to perform. This follows the Model View Controller design pattern (Reenskaug, 2018). The UI is the View, Business Logic is the Controller, and Database is the Model.

The MSA presented in figure 2 has many problem areas of which one is the challenging security implementation. This is due to the fact that every microservice accessible to the client can also be accessed or contacted by other more malicious parties in the same

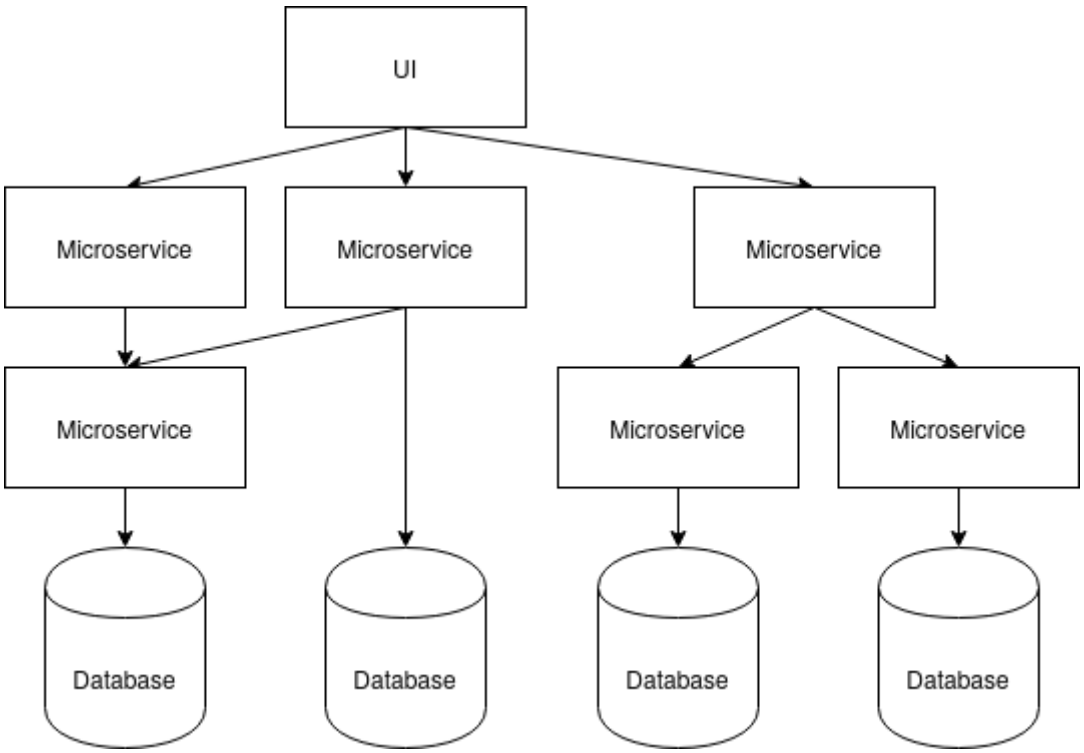

Figure 2: Microservice Architecture (Kalske et al., 2018)

network. The network in the case of web services is the internet. The attack surface available for the malicious party is the antirety of the offered API:s by the microservices.

One solution to limit the attack surface is the addition of API Gateway to the architecture as in figure 3. Montesi and Weber (2016) present an API Gateway design pattern. In this pattern there exists only one web service accessible to clients. The API Gateway allows for a natural place for an Policy Enforcement Point (PEP) and other more MSA specific features such as service discovery. The API Gateway is a critical component and the security features can to some extent be implemented atleast initially only there. Since all communication is to either flow through or atleast be sanctioned by the API Gateway the performance and accessability are critical.

3 Changing the architecture

To change the architecture of an already deployed service from MA to MSA should be a gradual process. This ensures a smoother transition and minimizes outages to the customers using the service. In every case though this is not possible.

The MA is or at least should be split in to modules with separation of concerns (Yarygina, 2018). The actual splitting of the monolith can be carried out in various ways. One of which is Domain Driven Design (Newman, 2019).

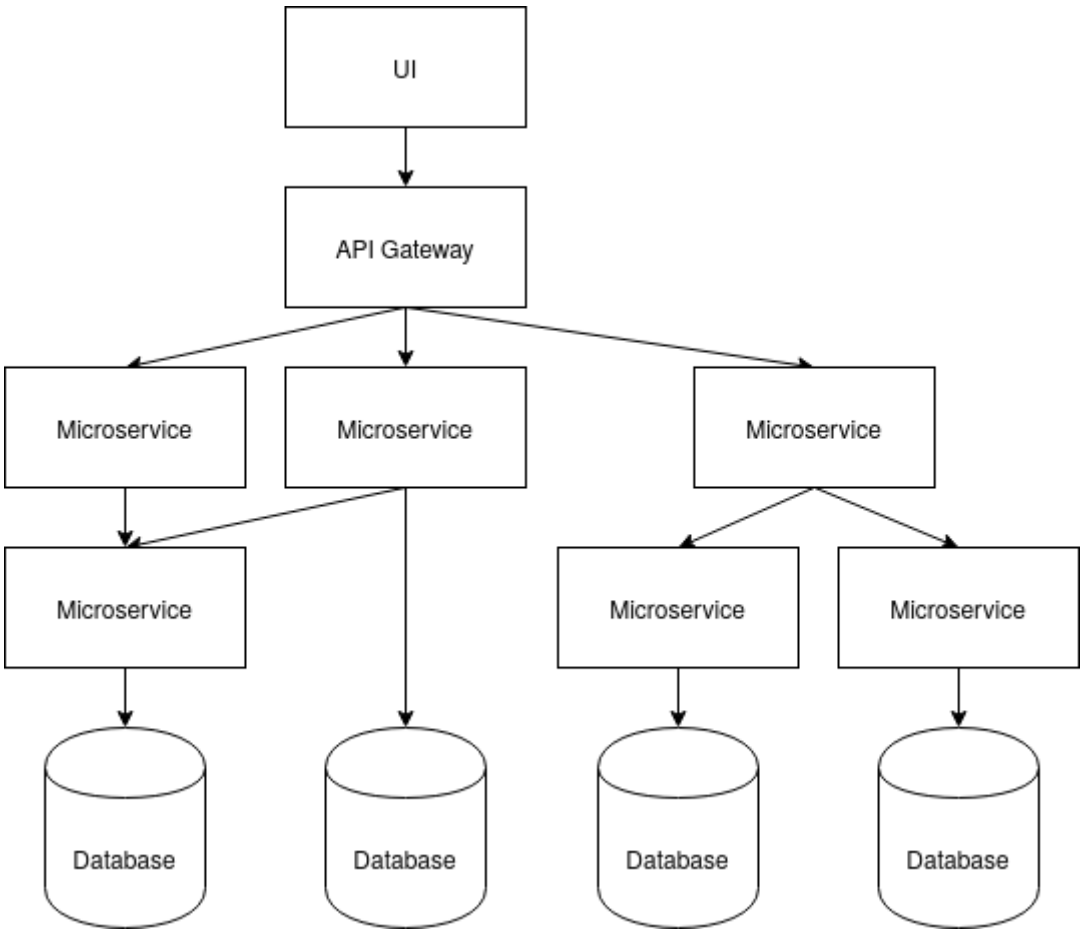

Figure 3: Microservice Architecture With API Gateway

The MSA differs from a MA in fundamental ways. According to Fowler and Lewis (2014) one main difference is the communication between their components. In a monolith application the processes can send function calls or method invocations amongst themselves. In MSA the messaging is based on sending messages or HTTP requests. Function calls entail a stackframe creation in the call stack, execution of the function code, and finally popping the stackframe and returning the result. Compilers can optimize the code further and inline the function calls to eliminate the stackframe creation and following procedures to be carried out. Communication using the network is extremely slow.

Zari et al. (2001) studied the response times of web sites offered to the public. The websites response times where measured in seconds. The requests sent to other microservices through the network are much slower than operation within one computer as the function calls. Therefore, the communication patterns should be changed to take into account the change in communication path. If the architecture is changed in such a way that the previous communication model amongst the components is preserved, there would be an excessive amount of communication and the resulting system would not be

as performant (Fowler and Lewis, 2014).

4 Security

Richter et al. (2018) implemented a test system mimicking the Deutsche Bahn seat reservation system using MSA. The technologies used in the study were: Amazon Web services as the deployment platform, Elastic Compute Clouds running on Kubernetes nodes, and Docker for containers in which the actual services were run. They found out that the cloud-based infrastructure when used in MSA resulted in a more complex solution than in MA. The complexity in modern software systems is inherent. Implementing security is very difficult and resource intensive. The rewards from a good security are invisible. When microservices are implemented or even planned the security should be taken into account as early as possible. Implementing security later on the project or as an after thought is can be more expensive and very difficult.

The added layers all have to be configured correctly and an error in one could potentially compromise the whole system.

5 Authentication

In these cases where the user has to be authenticated the web service needs a way to do this securily. Usually authentication is done using a tuple containing user credentials i.e. a username and a password for the user. The user is authenticated and a key or token is transmitted to the user via the network. This communication should in both MA and MSA be encrypted in a way that none of the actors in the transfer path can intercept the message and be able to use the credentials.

The credential counterparts i.e. shared secret by the server and the user have to be available for the web service for verification. When using MSA the service should own it's own data. When ever such information is available it is a target for thieves and hackers. The services in MSA are to be individually deployable and the service scalable. Authentication service implementation has to take this into account. The service has to adhere to practices that minimize the risks of data breaches.

5.1 Java Script Object Notation Web Token (JWT)

JWT is a format to represent claims. It is base64 encoded point separated strings which concatenated can easily be carried in the HTTP request or response. The contents is key value pairs and the token may or may not be signed and encrypted (Jones et al., 2015b).

The token may contain expiration time. If the token is used to validate requests without a server side implementation that can revoke a token it will be valid until this time.

The JWT token is issued by an authority trusted by the service or services. The issuer has to sign the token for there to exist any real authoritative weight on it.

The signing of JWT can be carried out in various ways. These are presented in the Jones et al. (2015a). The signature is computated using the algorithm and keys or certificates specified in the header values. When the token is signed using PKI private key it can be verified by all parties in possession of the public key.

The choices for signing algorithm for signing the JWT algorithm contain ”none” as one of the choices. This was found to be troublesome by McLean (2015). He found that many libraries did not operate in desired way. The receiving party could be fooled to validate a mutated token without any signature with the ”none” as it's algorithm. In addition to this vulnerability McLean (2015) found that the verification suffered from another fatal flaw. When a token was created by using a symmetric algorithm the servers could be fooled to believe that a token signed by just the public key and not the secret HMAC -key was a valid one.

5.2 Attacks

Authentication can be attacked by a multitude of methods.

- Cracking
- Impersonation attacks
- Hacking the system
- Malware
- Social engineering
- Cracking the encryption on the communication channel exchanging credentials and keys or tokens.

From 2013 onwards malware and data breaches performed by hackers have increased. The motivation to steal user data for hackers is the value of them in the black market. The damage of a data breach to the service provider can be substantial. For example the Yahoo data breach damages have been estimated to have reached $440 billion. The attacks in general have been targeted to entities with valuable data and insufficient security infrastructure. The least likely target to be hacked where non profit organisations and the most likely were medical related organizations (Hammouchi et al., 2019). The hacked

account credentials have to some extent been available for download from the web. Hunt (2020) created a service where everyone can verify whether any of their accounts are amongst the ones added to the service. The service named as "';– have i been pwned ?" allows users to enter their username or password to the site and see a result.

6 Authorization

Authorization of the user rights can be implemented in various ways. One of which is an authorization service which can contain the access control matrix. Services being accessed verify from the authorization service that a particular user or the role that the user has can access the requested service or functionality. In a MA the access rights to a functionality can be implemented using annotations within the source code. The authorization is verified in memory and without any communication over the network.

In MSA accessing the access control matrix or matrices isn't as easy as it is in MA. In order to verify that a specific right exists the service would have communication with the authorization service. This communication would need to happen every time a user tries to access a functionality with access restrictions. This could potentially lead to an extremely lively communication from all the services forming a bottleneck to the service.

6.1 Authorization in Monolith Architecture

In MA it is possible to implement features in such ways that a session can carry user information. This information can consist of granted roles and rights for the user. This session can be queried when e.g. access control is needed to execute an action or operation.

7 Communication

As already discussed in an MA the service components can communicate using events, procedure calls or other methods available within a single server machine. Usually all this communication stays within a single computer and thus does not necessarily compromise confidentiality.

7.1 Representational State Transfer (REST)

Fielding (2000) presented REST in 2000. REST has become a very successful architectural style. The style was derived using various constraints one of which is the demand of stateless communication. This entails that a request must contain all

information needed to fulfill the request because the server does not keep track of the client. All session state is stored in the client of which the server has no prior knowledge before a request. In her doctoral thesis Yarygina (2018) critiques the REST paradigm from the security perspective. She states that the design of the architecture does not meet the security requirements for web applications. She also states that REST does not allow for any server side sessions and thus token repudiation is impossible. Tokens can be validated only for correct issuer by signature and for expiration. As such tokens are more compatible with REST but there still has to be the private keys in the server for signature verification.

7.2 Coping With Failure in Communication

Montesi and Weber (2016) present widely used design pattern for MSA. The Circuit Breaker can be used to mitigate the very likely case that a microservice operates slower than the other services calling it and runs out ot resources to fulfill the requests in time. The circuit breaker is either implemented in the microservices or as a proxy between the client and the microservice. When the microservice does not service requests as intended the circuit breaker is to trip and send a failure message to the clients immediately when requests are received thus allowing the microservice time to service the prior responses.

The circuit breakers can prevent an application becoming completely unresponsive and crashing when a denial of service attack is carried out on the service.

8 Defence in depth

Jander et al. (2018) propose a solution for secure communication in MSA even in multicloud solutions. perimeter defence -> neglect security of individual microservices.

8.1 Deployment and Operation

When software is developed using MA it is usually deployed as a whole and the program code can be compiled, tested and used as a single unit or multiple modules. In contrast to this a service implemented by using a MSA can be deployed in single microservice units and thus a single service can be worked upon individually and deployed once ready.

The immediacy in the deployment of the microservices entail a very specific security risk. Ahmadvand et al. (2018) present threats from malicious insiders working on the services as developers or other positions with access to sensitive information. In microservice development the finished implementations are to be immediately released to production.

There are steps in the CD pipeline prior to this but once tests pass in the test environments the pipeline is supposed to publish the changes to the actual production environment. The paper presents four specific threats. The first one is that the knowledge of sensitive information is spread among the developers more widely than in MA. This is due to access needs by developers. The second threat is that the insiders monitoring and operating the running system intentionally harm the system by making malicious changes. The third threat is the developers knowing the configurations and their ability to make almost instantaneous changes to them or the microservices themselves. The last presented threat in the paper is the non-repudiation. The system is not able to disallow malicious requests when the developers have had access to the keys and other configurations. They can effectively implement services or requests that emit malicious requests or responds. Malicious attempts in a MA are more easily screened by performing security audits and by peer reviewing the code. In a MSA the knowledge of a single service and it's inner workings are shared by a more limited number of people. Finding the compromised actions from the interoperability of the distinct microservices is a daunting task.

Malicious attempts in a MA are more easily screened by performing security audits and by peer reviewing the code. In a MSA the knowledge of a single service and it's inner workings are shared by a more limited number of people. Finding the compromised actions from the interoperability of the distinct microservices is a daunting task.

8.2 Service discovery

Service discovery as presented in Montesi and Weber (2016) is a design pattern in which a registry is kept on currently running microservices. The microservices register them selves to the service discovery registry. This registry is used by either a router to route client service calls to running microservices or by the client directly.

8.3 Externalized configuration

To allow for easy configuration change management there should exist a configuration orchestration service. This service should have an API from which services in their startup can load their appropriate configuration. The configuration of the whole system can be easily maintained through the API.

The contents of the configuration is highly sensitive information. It consists of addressess, credentials and other information that alter the behaviour of the system. Therefore, the content must be stored safely and not allowed to be read or altered by unauthorized users.

9 Conclusion

This paper discussed the security aspects of changing the architecture from MA to MSA.

In MSA there are more things to go wrong than in MA. The deployment necetitates installing: virtualization, monitoring, and a plethora of other tools. In some cases these tools might not even exist and they have to implemented by inhouse developers and thus more costs are incurred upfront and also in the upkeep of the system. In addition to being more costly own development has higher risks involved.

MSA has higher complexity due to more tools needed and having more potentially exposed attack surface. Security can be though of as being as good as it's weakest link. In general a MSA deployment has multiple layers which all have to be consistent and correct. One example being the configurations of a system from the operating system on the server running the virtualization environment. All of the layers from the server hardware to the handling of errors in the actual code have to be of ample quality to mitigate a failure in security.

The communication that was in monolith a simple inprocess call might not be possible as such in a MSA web service. The individual services communicate via network with very high overhead incomparison to a simple function call. Furthermore the identity and authorization of the entity requesting action or data can usually be trusted on an inprocess call. The mechanisms to allow for proper authentication and authorization amount to even higher overhead for the MSA. There exists a very real risk for the development team to implement an insufficient security scheme.

In MSA the security should be implemented in depth. There must be a healthy mistrust on all requests and security should be built in to the system.

Security has to be taken into account right from the beginning of the project in which the architecture is to be changed. The choices made in the development of the web service when following a MA do not carry to the MSA as such.

Future research...

The research carried out...

References

Mohsen Ahmadvand, Alexander Pretschner, Keith Ball and Daniel Eyring. Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework. *Software Technologies: Applications and Foundations*, 2018.

Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine, 2000.

M. Fowler and J. Lewis. Microservices - a definition of this new architectural term, 2014. Available `https://martinfowler.com/articles/microservices.html`. Viewed 15.2.2020.

H. Hammouchi, O. Cherqi, G. Mezzour, M. Ghogho and ME. Koutbi. Digging deeper into data breaches: An exploratory data analysis of hacking breaches over time. *Procedia Computer Science*, 151(99):1004–1009, December 2019.

T. Hunt. ';– have i been pwned ?, 2020. Available `https://haveibeenpwned.com`. Viewed 8.2.2020.

K. Jander, L. Braubach and A. Pokahr. Defense-in-depth and role authentication for microservice systems. *Procedia Computer Science*, 130(1):456–463, December 2018.

M. Jones, J. Bradley and N. Sakimura. Json web signature (jws). RFC 7515, RFC Editor, May 2015a. URL `http://www.rfc-editor.org/rfc/rfc7515.txt`. `http://www.rfc-editor.org/rfc/rfc7515.txt`.

M. Jones, J. Bradley and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015b. URL `http://www.rfc-editor.org/rfc/rfc7519.txt`. `http://www.rfc-editor.org/rfc/rfc7519.txt`.

Miika Kalske, Niko Mäkitalo and Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. *Current Trends in Web Engineering*, Irene Garrigós and Manuel Wimmer, editors, pages 32–47, Cham, 2018. Springer International Publishing. ISBN 978-3-319-74433-9.

Tim McLean. Critical vulnerabilities in JSON Web Token libraries, 2015. Available `https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/`. Viewed 15.3.2020.

Fabrizio Montesi and Janine Weber. Circuit breakers, discovery, and API gateways in microservices. *CoRR*, abs/1609.05830, 2016. URL `http://arxiv.org/abs/1609.05830`.

S. Newman. *Monolith to Microservices. Evolutionary patterns to transform your monolith.* O'Reilly Media, Inc., 2019. ISBN 9781492047841. 1st edition.

Trygve Reenskaug. MVC XEROX PARC 1978-79, 2018. Available `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`. Viewed 29.3.2020.

Daniel Richter, Tim Neumann and Andreas Polze. Security considerations for microservice architectures. pages 608–615, 01 2018. doi: 10.5220/0006791006080615.

Stack Overflow. Stack Overflow Developer Survey Results 2019. Available `https://insights.stackoverflow.com/survey/2019`. Viewed 1.2.2020.

T. Yarygina. Overcoming security challenges in microservice architectures. *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*, 2018.

M. Zari, H. Saiedian and M. Naeems. Understanding and reducing web delays. *in Computer*, 34(12):30–37, December 2001.