Modular e-Commerce Data Warehouse using Microservices

1st Adam Žák
Faculty of Informatics and Information Technologies
Slovak University of Technology Bratislava
Bratislava, Slovakia
zak.adamx@gmail.com

2nd Martin Bobák
Institute of Informatics
Slovak Academy of Sciences
Bratislava, Slovakia
martin.bobak@savba.sk

***Abstract*—Data warehouses are systems that are used today by many companies for storing a large amount of structured data. Microservices allow splitting the whole software system into multiple parts, which can be deployed separately and work on their own. These services communicate and manage individual software functionalities.**

The paper describes a proposal of a data warehouse built with a microservice architecture. Also, the microservice data warehouse is compared with a classic monolithic data warehouse.

***Index Terms*—data warehouse, microservices, modularity, scalability**

I. Introduction

Cloud computing pawed a way to better resource utilization. It makes it possible to share infrastructure, platform, and software as a service with users who do not need to maintain the service. The next step was customization of platforms offered as a cloud service [1]. This approach is useful for many industrial as well as research communities [2] which are dealing with exascalability [3]. When lightweight resource sharing was more mature the new concept of software system design emerged. This new approach is microservices.

The paper is focused on microservices effectiveness within data warehouses which are subject-oriented, integrated, time variant, and non-volatile collection of data in support of management's decision making process [4]. Microservices as a concept are attracting a lot of attention these days and are also being implemented by many companies. Various other companies see this interest and want to also migrate to microservices. But this is not as straightforward as it can look. The microservice architecture brings many benefits to programming and development, but also many disadvantages. This study investigates parts of this problem, such as scaling, networking, comparison to the monolith data warehouse, and so on.

Corresponding author: M. Bobák (email: martin.bobak@savba.sk).
This work is supported by APVV grant no. APVV-20-0571 and VEGA grant no. 2/0125/20

978-1-6654-8775-7/22/$31.00 ©2022 IEEE

II. Microservices

Microservices can be understood as basic applications, with a simple task. These basic applications can run independently on different servers, can be scaled individually, according to the current server load, tested autonomously, and so on [5]. The precise definition is: "Microservices are a software development technique, a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services [6]."

Microservices are building on containerization. Splitting software into multiple separately running parts, was not possible in the past or was very costly and hard to maintain. But thanks to virtualization of operating system it is much easier and cheaper. It allows us to run multiple instances of the execution environment. These instances are behaving and looking like their servers, with their dynamic resource management [7].

A. Advantages and disadvantages of microservices

Thanks to a different approach that microservices bring, it is possible to achieve multiple advantages over monolithic architecture. The main advantages are the following [8], [9]: easier maintenance of source code, faster development, scaling of application, simplified testing, and easier bug fixing.

The different approach of microservices brings a lot of advantages, but it also brings disadvantages. A software architect has to take into account [10]–[12]: performance of the system, economical aspects of the solution, management of microservices, communication between services, hard Bug fixing (within the system as the whole), and test integration .

B. Micorservice and monolith approach comparision

Monolithic architecture is well-known approach, that is used regularly today. When developers build software with this monolithic approach, they think of it as a whole program in one piece. It contains everything, necessary for running, like frontend, backend, or management tools. Also in this architecture, developers usually have one database for the whole system [13].

This is one of many comparisons, that is needed to analyze before decision between monolithic and microservice architecture is made. Based on the study [14], it is possible

to identify crucial points such as: maintenance, deployability, team organization, cost, scalability, reusability, size of project, size of team, and price of migration.

On the other hand, microservice architecture is still a very young and evolving approach, and as such, still has many potential problems. Some of these problems are: microservice granularity, language diversity, code management, and managing API changes [15].

III. Related work

There are two types of data warehouses, local and cloud. Normal warehouses are the classic way of using them. Cloud warehouses on other hand are using cloud architecture. They are stored in computers elsewhere, not under your control and you access them by internet. This work is more focused on cloud-based data warehouses, as they are more used today.

TABLE I
Comparison of the state of the art approaches.

Data warehouse	Pros	Cons
Amazon Redshift	Columnar storage Querying language Strong AWS integration	Uniqueness Not Enforced Structured data Not multi-cloud solution
Google BigQuery	Easier management of warehouse Pay per use Multi-cloud solution	Overkill for smaller data sets Flat pricing -
Snowflake	Easy implementation Auto-scaling Structured and semi-structured data	Customer is dependent on infrastructure Big data load No support for unstructured data.

IV. Microservice-oriented architecture for Data Warehouse.

The microservice system is divided into multiple parts, each working with its own related services on its own task. This is building some kind of tree structure. Each rectangle representations one microservice. Figure 1 is showing a simplified architecture diagram.

Proposed microservice flow is shown on invoices example in bigger detail. This example is part of the data warehouse that is working with invoices, loading data, and storing them. We can see a diagram of this flow in figure 2.

The proposed microservice-based data warehouse is composed of the following main parts:

Admin service - this is a microservice, which is responsible for the management of the whole data warehouse and data pipeline. In this case, it is also responsible for uploading of files.

Dispatcher - Microservice, which decides, what is the type of file by its name and type. For example, the GLS delivery service is sending .xls files, so when the .xls file arrives at the dispatcher, together with parameter type = 'gls', it will recognize, that it is an invoice from the GLS company. After this recognition, it chooses where to send this data.

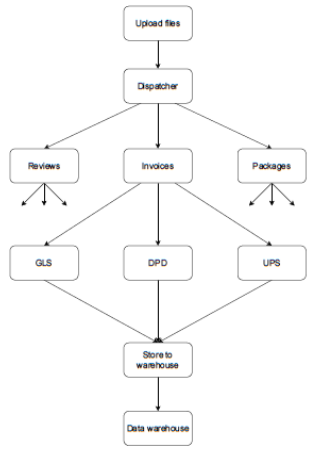


Fig. 1. Tree structure of microservice architecture.

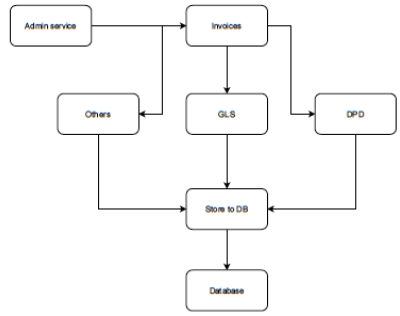


Fig. 2. Map showing services in data flow.

GLS - This microservice is processing GLS invoices. It receives data, which are transformed to the requested format, and then they are forwarded. This microservice is the parser, and each type of file and company has individual parser like this.

Store to DB (DB is abbreviation for word Database) - The responsibility of this microservice is to receive data from parsers (like GLS service) and then store the parsed data correctly in the database. It must pick the right table and insert all additional important data.

Database - The purpose of this service is running a data warehouse database.

Concept sumarization The goal of the microservice concept is to divide each part of ETL into a microservice. Each part of stored data, like invoices, package information, user reviews, and so on, has its own services its own parser (like GLS for example). But they also share some services. They share storage in the data warehouse (Database), share extract (Admin service) and also share load (Store to DB).

V. Experiments

The proposed approach is evaluated by the following hypothesises: 1) Microservice-based data warehouse will be bit faster than a monolithic data warehouse, but the difference will be small, 2) Microservice-based data warehouse will be more capable to handle problems with overloaded modules, and 3) Microservices are more suitable for working with large data.

The evaluation environment has the following configuration: CPU: 1.35GHz Intel i5-4210U, RAM: 12GB GPU: Intel Haswell-ULT, and OS: Linux - Pop!_OS 20.04

A. Monolithic data warehouse comparison

The system chosen as the monolithic data warehouse is qFlow developed by SoftPoint [1]. qFlow is a workflow management application that offers multiple options, one of which is a data warehouse for your chosen files loaded from different places.

An invoice in qFlow takes an average of 1:38 to process. This is caused by many additional logs that are performed in qFlow. In the microservice program, the average time to process an invoice is 13 seconds. This is a major difference, but is not caused by a different architecture, but rather because of a different implementation (such as multiple logs on side of qFlow, but not on in microservices).

To obtain more precise values and a better understanding of the difference between these two options, the execution time has been divided by the average execution time of one invoice. This produces a number that shows how many invoices with average run time could be processed. The smaller number is better because it means that for a shorter period of time, a larger number of invoices could be processed.

This number can be compared with the numbers from experimenters with microservices-based solution. The graph 3 shows that the monolithic architecture performs better without scaling on 10 uploaded invoices. But further scaling of processor containers enables better utilization and higher speed, which reflects on smaller division number.

This graph shows that it performs even better than microservices when they are not scaled. With further scaling, the execution time is reduced and microservices start to perform better than the monolith.

The graph with 20 invoices shows that even in the default configuration of microservices, they perform better than a monolith. Scaling speeds the program, but runtime does not go down below 10,75 which is 2 minutes and 9 seconds.

The last test was performed on 30 invoices in the system. Again, it is visible that even in the default configuration, the microservices performed better. Scaling the reduced time even more to 15.6 division number, which is equal to 3 minutes and 8 seconds.

[1]qFlow: https://softpoint.tech/qFlow/

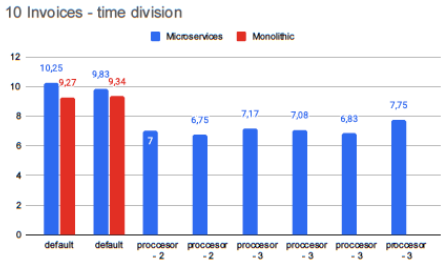


Fig. 3. Division number comparison of 10 invoices uploaded to system

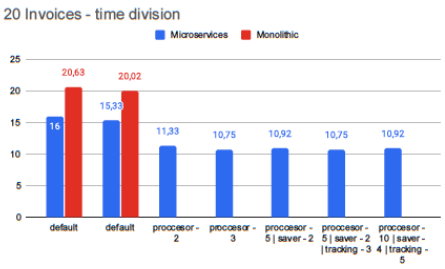


Fig. 4. Division number comparison of 20 invoices uploaded to system.

B. Effectiveness tests

The first type of experiment aimed at a different speed in containers of different scales. With the use of a larger number of invoices or reviews, bottlenecks are more significant and have a larger impact on the program. This is demonstrated, for example, in figure 6. One of consequences is a slower

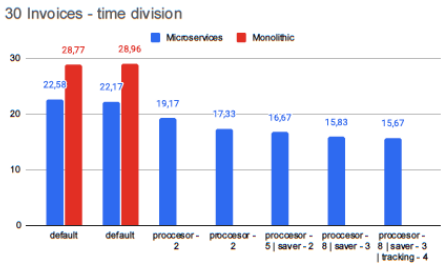


Fig. 5. Division number comparison of 30 invoices uploaded to system

execution time, as shown at the beginning of table II and also in table III.

1) Invoice effectiveness test: The initial test case was in the invoice part of the program (GLS invoice). An invoice is uploaded to the Admin Service, then processed in the processor, and saved in saver. In a case like this, this is enough. But in the case of multiple invoices uploaded at once, this is not enough. Five invoices uploaded at once create this kind of flow diagram as shown in figure 6.

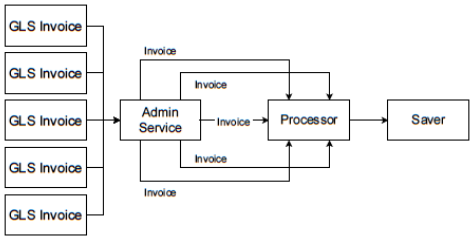


Fig. 6. Flow of 5 invoices in program.

Figure 6 clearly shows that in the case of containers scaled as this, the processor container creates a bottleneck in the program. While one invoice is being processed, the rest are waiting. This slows the program and can be easily solved by scaling the number of containers.

When there is only one container per application module and 10 invoices are being uploaded, their processing takes between 2:02 and 2:07 minutes. An additional processor container saved 36 seconds. This is a big advantage of microservices. A container that creates a bottleneck can be deployed multiple times, and the bottleneck can be removed.

In the case with 10 uploaded invoices, the additional horizontal scaling does not anymore shorten the execution time. This is because this amount of invoices is not enough to better utilize additional containerization.

2) Reviews effectiveness test: In this test scenario, the review processing speed is being tested. Reviews are downloaded once a day using a cron job in the background. To test this part of the program, this background job was not scheduled once a day but 10 times (or more in some cases). One file has around 6000 reviews, so 10×6000 reviews equal to 60000 reviews that need to be parsed and saved.

This part of the program has two options to run. One is classic, the same as invoices; processors receive one XML file, parse it, and then send parsed data to the saver, which stores those into the database. Another option is to go review, by review in the XML file, parse this one, send it to background job, which then calls saver, which then stores this one review into the database.

These measurements are taken in the second-mentioned version, where each line of the file has been stored separately.

In a default configuration, where each container is deployed only once, 10 review files were processed in 11 minutes and 21 seconds. In the version, with two processor containers, the execution time was 9:41 minutes. This means that 1:40 is obtained, thanks to the double processor container. The next experiment was with two single_review containers. With two of these containers, the run-time was 9:37 minutes. This is very much the same as for double processors. Another experiment was a combination of 2 single_reviews and 2 savers. This releases pressure from the saver, as it was one container that received data from 2 containers. This manifests itself also in time, which is 7:10 minutes. That means another 2:27 minutes saved.

The last 3 experiments were with 20 and 30 review files. In the case of 20 files and 20 single_reviews, 20 savers, the execution time was 7 minutes and 9 seconds. In the case of 30 files and 30 single_reviews, 30 savers, the execution time shortened to around 1 minute and 20 seconds. This is the best execution time achieved in review-based testing. 30 files, 30 savers, and 30 single_reviews are the best set-up of microservices in the reviews part of the program.

TABLE II
MEASURED TIMES OF EXECUTION IN DIFFERENT CONFIGURATIONS. TIMES ARE SLIGHTLY HIGHER BECAUSE COMPUTER WAS DIFFERENTLY LOADED (WHICH RESULTS IN 30 ADDITIONAL SECONDS AS IN DIFFERENT TESTS).

Test type	Execution time	Scale
10 GLS invoices	02:05	default
10 GLS invoices	02:03	default
10 GLS invoices	01:58	default
10 GLS invoices	01:38	processor - 2
10 GLS invoices	01:31	processor - 2
10 GLS invoices	01:24	processor - 2
10 GLS invoices	01:21	processor - 2
10 GLS invoices	01:26	processor - 3
10 GLS invoices	01:25	processor - 3
10 GLS invoices	01:22	processor - 3
10 GLS invoices	01:33	processor - 3
10 GLS invoices	02:09	saver - 2

TABLE III
MEASURED TIMES OF EXECUTION IN DIFFERENT CONFIGURATIONS.

Test type	Execution time	Scale
10 Heureka reviews	11:21	default
10 Heureka reviews	09:41	processor - 2
10 Heureka reviews	09:37	single_review - 2
10 Heureka reviews	07:10	single_review - 2 — saver - 2
10 Heureka reviews	05:29	single_review - 5 — saver - 5
10 Heureka reviews	03:31	single_review - 10 — saver - 10
10 Heureka reviews	03:20	single_review - 20 — saver - 20
10 Heureka reviews	03:17	single_review - 20 — saver - 20
30 Heureka reviews	02:05	single_review - 30 — saver - 30
30 Heureka reviews	01:20	single_review - 30 — saver - 30
30 Heureka reviews	01:19	single_review - 30 — saver - 30

C. Network tests

This section focuses on testing the network and its use. Almost all tests are executed on the review processing part of the solution. The measured numbers show directly the network load and how it grows with scaling.

The first test was done using Heureka reviews, also with two processor containers. The network load is roughly the same, around 162 - 165 KiB/s. After adding a saver container, the network load changes. With one saver container, the load is approximately 162 KiB/s, with two savers, the load is approximately 243 KiB/s.

This grows directly further; more containers equal more network load. The case with 20 savers and single_reviews is no exception. A double number of containers means that the network load also doubles. This network load exceeds 1 MiB/s.

Scaling also affects the network load at rest, without any request, as TCP keep-alive messages are sent to more containers. The average load was equal to 23.5 KiB/s.

D. Load size test

The graphs show that with the increasing number of invoices, the gap between microservices and the monolithic solution is increasing. In the case of 3, the difference between the fastest container configuration and the monolithic solution is approximately 3 division numbers; with 20 invoices, the difference increases to about 10 and in the case with 30 invoices, it increases to up to 13. But this also applies to comparison among microservices themselves. The biggest gap between fastest and slowest times in differently scaled containers is increasing.

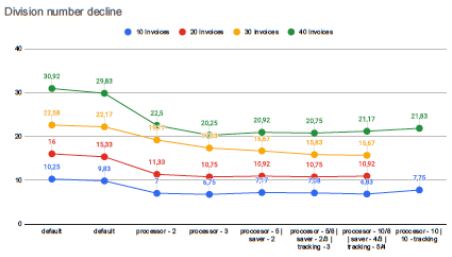


Fig. 7. Comparison of different load and execution times.

This is shown in the graph 7. With an increasing number of uploaded invoices, the execution time automatically increases. The first two points in the graph are always with the default number of containers, which means that there are no additional containers, only 6. The third point is where an additional processor container was added. In all four conditions tested with invoices, the second processor container helped. Except for 30 invoices, where growth was slightly slower, in each larger configuration, this difference was greater than in the previous test.

E. Discussion

In this section results of experiments and related hypotheses are discussed.

1) Hypothesis number 1: : "Microservice-based data warehouse will be bit faster than a monolithic data warehouse, but the difference will be small." was proved to be true in this case. With the right configuration, microservices work faster. This difference is becoming more and more visible with a larger and larger load.

The containers are performing better in the tests, they are faster than the monolith, thanks to scalability. However, these tests also proved that they can be slower in some scenarios.

The question for developers now is: is it better to build a whole program with microservices, or is it better to stay with the classic monolithic approach?

The experiments showed that they also need the right setup for it. Without this setup, they can cause more harm than benefit. This can be seen again in the graphs, which show that, with scaling, there was acceleration in execution time, but without it, for example, with 10 invoices, the program ran slower.

2) Hypothesis number 2: The original hypothesis is as follows: "It will be easier to solve problems with overloaded parts of the data pipeline." This assumption is addressed in V-B, which talks about execution time speed. The running time of the program depends on the load. In the event of overload, bottlenecks arise and slow down the program. The given section examines how the execution time changes with the change in the number of scaled containers.

The monolithic solution lacks options to address these unexpected cases of overload. If such an unexpected problem arises, it means that the program needs to be rewritten or better computer units can be bought to improve performance. Both of these options are costly and time-consuming.

On the other hand, microservices offer a great solution with scaling. It is directly visible in tables II and III. As the number of scaled containers grows, the execution time decreases. This is happening because scaling removes bottlenecks.

All experiments are tested on the same computer with the same configuration. And just one additional step in deployment shortened execution time from circa 11 minutes to somewhere in between 1:30 and 2:30 minutes.

Our experiments confirms that multiple containers make better use of hardware infrastructure and parallelization. The first image shows the utilization of cores without scaling any containers. The second image shows how cores are utilized with the processor container scaled three times. More CPU load is placed on the processor, but the execution time is much faster. All of this shows that the second hypothesis is true.

3) Hypothesis number 3: states: "Microservices are most efficient when working with large data.".

Experiments showed that with larger loads, microservices offer faster speed than monoliths. In cases with a smaller number of uploaded files, qFlow performance was very close to microservices, sometimes even better.

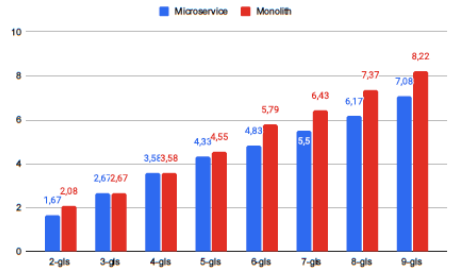


Fig. 8. Different division number from 1 uploaded invoice to 9 invoices.

The graph 8 shows how the gap between the monolithic and microservice solution is increasing with the increasing number of invoices. In cases with 3 and 4 invoices, the division number is even the same for both systems.

As the number of files increased, qFlow started to slow down compared to microservices. The graph 5 shows that in a good microservice configuration, the gap between monolithic and microservices is the 13 division number (28-15). This is a continuation of the trend from 8, where the largest difference is one division number. Note that the containers were not scaled in this configuration, so the difference could be larger. But also if we compare nonscaled version of microservices, in case with 30 invoices, the division number difference is around 6-7.

This hypothesis is also proved in the experiments with reviews. In this case, the program was not compared to qFlow, but to itself. Reviews, as mentioned above, offer two options to run. In one case, they are less containerized, and in another they are more containerized.

The detail is shown in graph 9, where the blue line represents the more divided and containerized part of the program, and the red line represents the simpler and less containerized part.

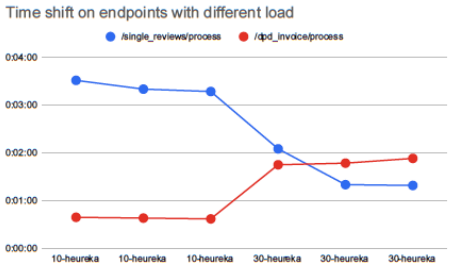


Fig. 9. Graph of time shift on endpoints with different load.

It proves Hypothesis 3 and also proves that microservices are not a good option for smaller systems. Review processing is designed to download one review per day and process it. There is no need for microservices and, in this scenario, they are creating more of an obstacle than helping.

VI. Conclusion

The monolithic qFlow data warehouse is compared to the microservice-based data warehouse. Invoice experiments have shown that microservices are performing faster in most cases, but they have also shown that in many cases this difference is not large. The difference increases with the larger number of invoices. However, to achieve this, the microservice data warehouse must be correctly configured. This is not an easy task and can be quite difficult to find the right setup of containers.

The research in this work proved that microservices are a better and easier option when it comes to scaling applications. Experiments showed that the option to scale different parts of the program as needed can dramatically change the execution time of processes in the system. Monolithic applications, on the other hand, need more work and time when they encounter a situation like this.

References

[1] M. Bobák, L. Hluchý, and V. Tran, "Tailored platforms as cloud service," in *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*, 2015, pp. 43–48.

[2] M. Graziani, I. Eggel, F. Deligand, M. Bobák, V. Andrearczyk, and H. Müller, "Breast histopathology with high-performance computing and deep learning," *COMPUTING AND INFORMATICS*, vol. 39, no. 4, p. 780–807, Jan. 2021. [Online]. Available: https://www.cai.sk/ojs/index.php/cai/article/view/2020_4_780

[3] M. Bobák, A. S. Z. Belloum, P. Nowakowski, J. Meizner, M. Bubak, M. Heikkurinen, O. Habala, and L. Hluchý, "Exascale computing and data architectures for brownfield applications," in *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2018, pp. 450–457.

[4] W. H. Inmon, "What is a data warehouse," *Prism Tech Topic*, vol. 1, no. 1, pp. 1–5, 1995.

[5] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Software*, vol. 35, no. 3, pp. 96–100, 2018.

[6] B. Christudas, *Practical Microservices Architectural Patterns: Event-Based Java Microservices with Spring Boot and Spring Cloud*. Apress, 2019.

[7] K. Kolyshkin, "Virtualization in linux," *White paper, OpenVZ*, vol. 3, no. 39, p. 8, 2006.

[8] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[9] B. Mayer and R. Weinreich, "A dashboard for microservice monitoring and management," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 66–69.

[10] P. Utomo and F. Falahah, "Conceptual model of a dashboard for monitoring microservices," *EAI Endorsed Transactions on Cloud Systems*, vol. 6, no. 18, 2020.

[11] D. Shadija, M. Rezai, and R. Hill, "Microservices: granularity vs. performance," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 215–220.

[12] D. Savchenko, G. Radchenko, and O. Taipale, "Microservices validation: Mjolnirr platform case study," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 235–240.

[13] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," in *International Conference on Web Engineering*. Springer, 2017, pp. 32–47.

[14] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: an assessment framework," *Information and Software Technology*, vol. 137, p. 106600, 2021.

[15] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: an exploratory study," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–44, 2021.

Authors



Bc. Adam Žák
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava, Slovakia
zak.adamx@gmail.com
His study field covers computer science as a whole, from neural networks, and data engineering to computer graphics and games. His main professional activity is oriented toward microservices and their application within data warehouses. Currently, he is focused on the use of semantics in both these areas of computer science.



Mgr. Martin Bobák, PhD.
Institute of Informatics
Slovak Academy of Sciences in Bratislava, Slovakia
martin.bobak@savba.sk
His field of research is multi-cloud computing and architectures of distributed, cloud-based applications. He is the (co-)author of scientific publications and has participated in several European (e.g. H2020, Horizon Europe) and Slovak R&D projects. He is also a university lecturer at the Faculty of Informatics and Information Technologies, Slovak University of Technology.