The Comparison of Microservice and Monolithic Architecture

Konrad Gos
Department of Microelectronics and Computer Science
Lodz University of Technology
Lodz, Poland

Wojciech Zabierowski
Department of Microelectronics and Computer Science
Lodz University of Technology
Lodz, Poland
wojciech.zabierowski@p.lodz.pl

***Abstract*—In this day and age, people demand fast efficient and reliable applications. If a client has a high speed internet connection and required client's app, the features such as reliability and efficiency can be provided only by the programmer. It is therefore essential to choose appropriate software architecture, before the implementation of functionalities that have been adopted to the project. Some of the most popular architectures are the Monolithic and Microservice architectures that can achieve the same result, however with different advantages and disadvantages. The purpose of this article is to compare the abilities and performance of the two software architectures – Microservice and Monolithic on the example of a web application in Java.**

Keywords—monolith, microservice, architecture, spring, java

I. Introduction

Software architecture [1] is the fundamental structure of a software, that defines technical and operational requirements. It is responsible for optimization of each attribute of an application, like efficiency, manageability , scalability, reliability, modifiability, deployability and more other aspects. That is why the choice of appropriate architecture is so important in the primary phase of software development.

There are several types of software architectures, however, the Monolithic and Microservice Architectures belong to the most popular ones.

II. Monolithic Architecture

In the past few years software developers winningly used Monolithic Architecture. The Monolithic application is a software in which different components (such as authorization, business logic, notification module, etc.) combined into a single program from a single platform.

The picture below presents an example of a Monolithic application that provides E-commerce business logic.

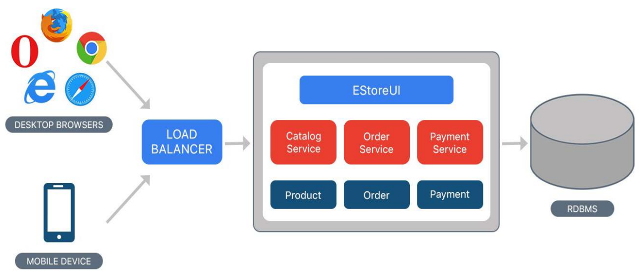


Fig. 1. Monolithic flow on the example of E-commerce application [2]

Despite implementing many parts of the whole software, the application is deployed as one single standalone program.

III. Microservice Architecture

Today, programmers obtain many new possibilities to create an application that will bring development process to the next level. One of the new popular solutions, regarding Google Trends [3], is Microservice Architecture. It is an architectural style that structures an application as a collection of services. Each microservice had to provide one part of the business logic.

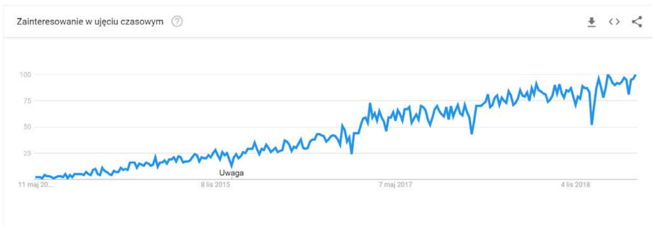


Figure 2. Google Trends statistics for searching word "microservices" between 2014-2019 [3]

Figure 3 presents an example of Microservice application, that provides E-commerce business logic, which is equivalent to the previous monolithic version..

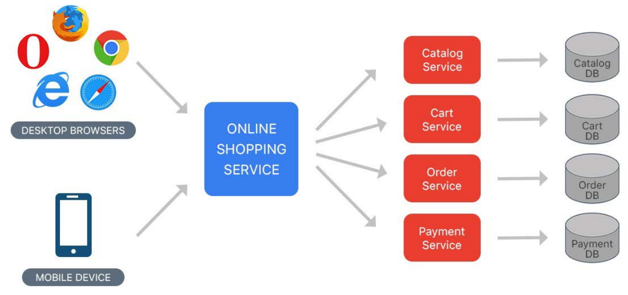


Fig. 2. Microservice flow on the example of E-commerce application [2]

As you can see, there are four core Microservices, that provide business logic and due to one additional service are exposing functionalities as it would be one application.

IV. Java

Java is a programming language, developed by Sun Microsystems in 1995, which was taken over by Oracle in 2010. It is class-based, object-oriented and has a huge amount of users in programmers society. According to source [4] it is in top three of the most popular programming language.

Until today, Oracle released a few versions of Java, that are better optimized and more convenient for a programmer. One of the latest version that has no restriction for commercial usage is Java 8. This implementation provides many features and enhancements [5], such as:

978-1-7281-7180-7/20/$31.00 ©2020 IEEE

- Lambda Expressions – a new feature; enables programmers to treat functionality as a method argument, or code as data and lets them express instances of single-method interfaces more compactly
- Streams – new classes that provide Stream API to support functional-style operations on streams of elements. It enables bulks operations on collections, due to the integration with Collections API

V. Spring Boot and Spring Cloud

Spring Boot [6] is a brand-new framework based on the Spring framework. This tool enables a programmer to build web-based applications faster and in a simpler way. Contrary to Spring framework, Spring Boot almost does not require configuration. There is now a need to provide many XML descriptors. The only need is to choose smartly dependencies that you want to use.

Spring application bases on Multitier Architecture, which looks as follows:

- The web layer – it handles user's requests and handles exceptions thrown by other layers
- The service layer – it implements business logic of application
- The repository layer - - it is responsible for communication with the database

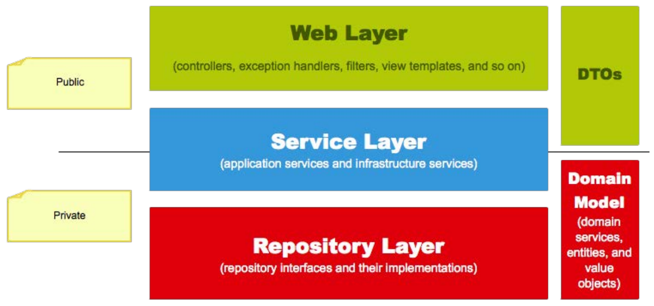

Public
Private
Web Layer
(controllers, exception handlers, filters, view templates, and so on)
DTOs
Service Layer
(application services and infrastructure services)
Repository Layer
(repository interfaces and their implementations)
Domain Model
(domain services, entities, and value objects)

Fig. 4. Representation of Multitier Architecture used by Spring Boot [7]

Spring Boot is a powerful tool, however, to develop microservices there is a need to have some tools that will be responsible for communication and other aspects typical for these architectures. This can be achieved with Spring Cloud, which contains common patterns for distributed systems.

VI. Docker

Deployment of application always needs many of configurations, dependencies and other requirements. To avoid this, the good practice is to use Docker [8] – a platform to develop deploy and run applications with containers. The process of deployment of applications in Linux containers is called containerization and it has a lot of advantages, such as:

- Flexibility
- Lightness
- Interchangeability
- Portability
- Scalability
- Stackability

VII. Gatling

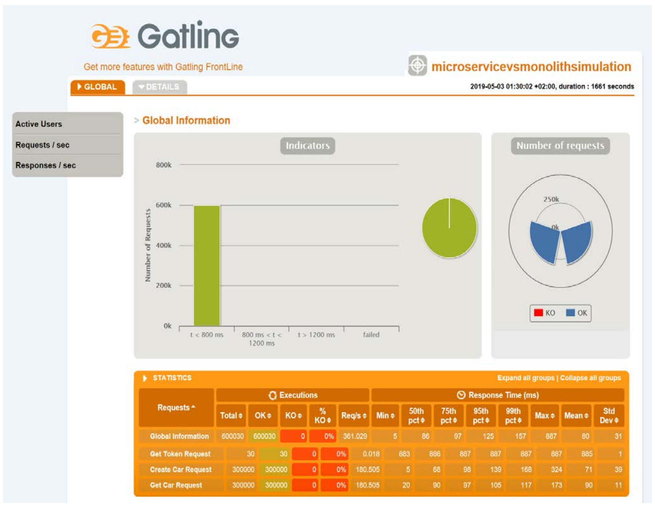
Gatling [9] is a highly capable load testing tool. It is well-known for the ease of use, maintainability and high performance.

Gatling supports HTTP protocol and that is why it can be used for load testing on any HTTP server.

Loading tests had to be written in Scala, however, thanks to Gatling, basic functions are easy to develop, because it supports an expressive DSL (Domain-Specific Language).

If prepared test scenarios are ready, they can be launched and after some time it will generate a report. This includes many statistics and charts in a beautiful form, such as:

- active users during the simulation
- distribution of the response times
- a variety of response time percentiles over time for successful requests
- a number of requests sent per second over time
- a number of responses received per second over time



2019-05-03 01:30:02 +02:00, duration : 1661 seconds

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	600030	600030	0	0%	361.029	5	86	97	125	157	887	80	31
Get Token Request	30	30	0	0%	0.018	883	886	887	887	887	887	885	1
Create Car Request	300000	300000	0	0%	180.505	5	68	98	139	168	324	71	39
Get Car Request	300000	300000	0	0%	180.505	20	90	97	105	117	173	90	11

Fig. 5. Example view of a report generated by Gatling

VIII. Tested Applications

To compare the architectures presented earlier, you need two applications with identical functionalities. The technologies described above have helped to achieve this. Both applications were designed to provide basic car sharing abilities, such as: getting, creating, renting cars and updating the status of rent.

The first implementation was a monolithic version created using Spring Boot. The chosen database was well known PostgreSQL that has an open source licence. The connection between the application and persisting layer was provided by Hibernate – a module used for object-relational mapping in Java. The whole project was containerized with Docker using 'openjdk:8-jre-alpine' image as a base. Picture 6 presents the flow of the used architecture.

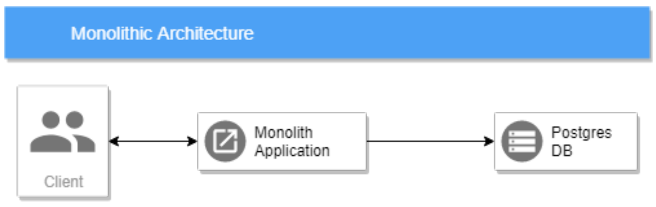


Fig. 6. Monolithic Architecture flow of implemented application

The Microservice application is basically a modularized version of the monolithic variant. It was created using Spring Boot and Spring Cloud technologies. The last one provides prepared and adjusted libraries (some of them bases on Netflix Open Source Software) that were used to implement service discovery (Eureka) and a gateway (Zuul). Eureka helps to discover an available microservice that is already running. Zuul is an implementation of the API Gateway pattern that mainly hides all microservices from a client and shares endpoints like a single application. All microservices should be autonomic and independent, therefore a message broker (Apache Kafka) is used. The database layer is like in the monolithic part (Hibernate with PostgreSQL). To simplify the deployment process, each microservice was containerized with Docker. Below, there is a picture that presents the flow of this architecture.

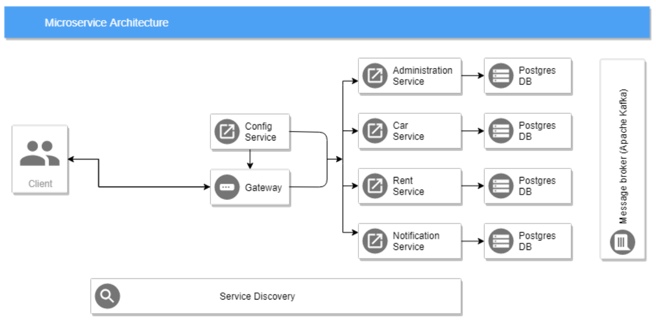


Fig. 7. Microservice Architecture flow of implemented application

IX. Conducted Tests

In order to compare the performance of the two presented architectures, two cases of tests were performed:

- 1 000 requests made at once by 30 users (1 000 * 30 = 30 000 requests)
- 10 000 requests made at once by 30 users (10 000 * 30 = 300 000 requests)

The above cases were performed by prepared tests with Gatling. It includes HTTP GET and POST requests for both architectures. Microservice architecture was tested in three variants – without replication, replicated twice and four times.

The tests were performed on PC with Ubuntu 18.04.2 LTS operating system. The applications were deployed with Docker. In case of Microservice architecture only one database server was available for all running microservices. The work station had the following parameters:

- Processor - Intel i9-9900K 3.6 GHz – 8 cores, 16 threads
- RAM memory - G.SKILL 32 GB DDR4 3200 MHz CL14
- HDD - Samsung SSD 840 EVO 120 GB

The first case handled 30 000 requests. The blue indicator on chart 8 shows that here the most powerful is the monolithic application. It handles ca. 789 requests per second. The Microservice variant was worse. The more microservice was replicated, the worse the results were. The best result of microservice architecture handled ca. 600 requests per second.

The opposite results are for the second case (300 000 requests). Here the best is the Microservice architecture replicated twice with the result of 239 requests per second, were the monolithic application handles only 180 requests per second.

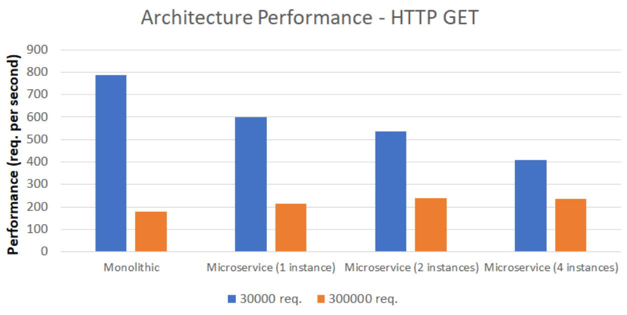


Fig. 8. Architecture performance (HTTP GET) - results

TABLE I. ARCHTECTURE PERFORMANCE – HTTP GET

Architecture	30 000 req.	300 000 req.
Monolithic	789	180
Microservice (1 instance)	600	215
Microservice (2 instances)	535	239
Microservice (4 instances)	410	237

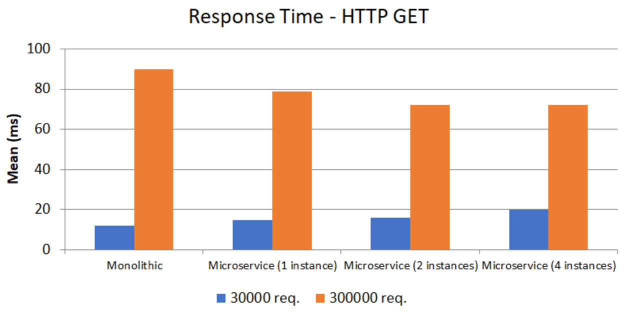


Fig. 9. Response time (HTTP GET) - results

TABLE II. RESPONSE TIME – HTTP GET

Architecture	30 000 req.	300 000 req.
Monolithic	12	90
Microservice (1 instance)	15	79
Microservice (2 instances)	16	72
Microservice (4 instances)	20	72

As you can see, you may think why the four times replicated variant is not the most efficient. The answer is simple, all replicated services were running on the same machine and had one shared database. That is why it could not use its whole potential.

Below there is the second collection of tests for HTTP POST. The data has behaved almost equal to the GET variant, so the conclusions are the same.

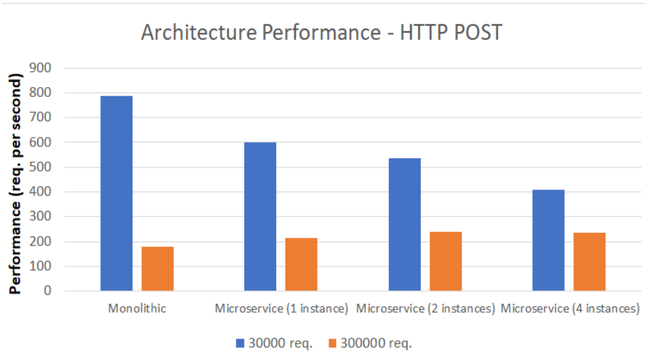


Fig. 10. Architecture performance (HTTP POST) - results

TABLE III. ARCHTECTURE PERFORMANCE – HTTP POST

Architecture	30 000 req.	300 000 req.
Monolithic	789	180
Microservice (1 instance)	600	215
Microservice (2 instances)	535	239
Microservice (4 instances)	410	237

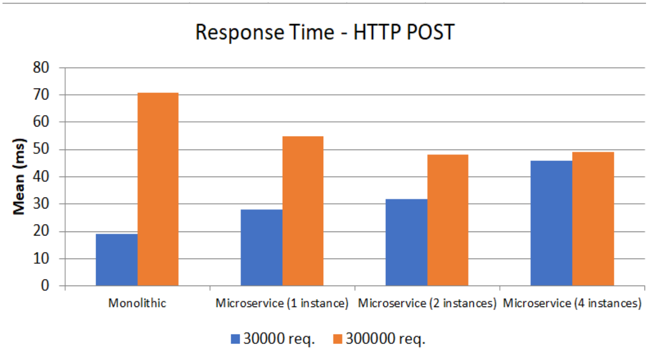


Fig. 11. Response time (HTTP POST) - results

TABLE IV. RESPONSE TIME – HTTP POST

Architecture	30 000 req.	300 000 req.
Monolithic	19	71
Microservice (1 instance)	28	55
Microservice (2 instances)	32	48
Microservice (4 instances)	46	49

X. ARCHITECTURE ADVANTAGES AND DISADVANTAGES

Performance is one of the most important parts of architecture, however, there exist other typical advantages and disadvantages of used architecture. Some of them, which were seen during the implementation process, are presented below.

A. Monlothic Architecture – pros and cons

- + Easy to develop
- + Simple to deploy
- - Complex maintenance
- - Reliability (one fault can bring down the entire application)
- - Availability (redeploy the entire application on each update)
- - Hard to scale

B. Microservice Architecture – pros and cons

- + Easy maintenance (easier to understand by a developer, each core functionality is a separated module)
- + Reliability (microservice fault affects only that microservice alone)
- + Availability (redeploy a new version of a microservice requires little downtime)
- + Easy to scale
- - More complex deployment
- - Autonomy (it is a positive aspect, however to achieve this along with data integration is a huge challenge)

XI. CONCLUSIONS

Both tested architectures have some advantages and disadvantages. All depend on the problem that has to be solved. The conducted load tests have indicated that the Microservice architecture is more efficient if an application has to handle a bigger number of requests. It has a lot of advantages that allow to build a high quality software, which is easy to scale, more reliable and in long term more convenient to maintain. Although it has so many profits, the Monolithic architecture is not bad. It is more efficient during lower load and is easy to develop. There are not so many problems with integration, connection and configuration. The choice of the right architecture should be defined by purposes of business so the investor will get the product which will meet their expectations.

REFERENCES

[1] https://www.techopedia.com/definition/24596/software-architecture

[2] https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

[3] https://trends.google.pl/trends/explore?date=today%205-y&q=microservices

[4] https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10?IR=T

[5] https://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

[6] https://spring.io/projects/

[7] https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/

[8] https://docs.docker.com/

[9] https://gatling.io/docs/current