DOI https://doi.org/10.30525/978-9934-588-79-2-1.4

SCALABLE MICROSERVICE ARCHITECTURE FOR WRITERS

Hrushko S. S.
PhD,
Associate Professor at the Department of Computer Systems
Zaporizhzhia Polytechnic National University

Timenko A. V.
Assistant professor at the Department of Computer Systems
Zaporizhzhia Polytechnic National University

Vusu-Hovu Silviia
Master at the Department of Computer Systems
Zaporizhzhia Polytechnic National University
Zaporizhzhia, Ukraine

The term "Microservice Architecture" has gained popularity in the past few years like a way to describe how applications are designed as a set of independent deployed services. There is no precise description of this architectural style, but there is a common set of characteristics: organization of services around business needs, automatic deployment, transfer of logic from the message bus to the sink (endpoints), and decentralized control over languages and data.

Using microservice architecture provides the developer with the following opportunities to expand the project:

– modules can be easily replaced at any time: emphasis on simplicity, independence of deployment and updating of each of the microservices [1, p. 52];

– modules are organized around functions: the microservice, if possible, performs only one fairly basic function [1, p. 50];

– modules can be implemented using different programming languages, frameworks, composite software, executed in different development environments; run in different environments of containerization, virtualization, under different operating systems on different hardware platforms;

– the architecture is symmetric, not hierarchical: the dependencies between microservices are peer-to-peer [4, p. 31].

The microservices philosophy actually mirrors the Unix philosophy, according to which every program should «do one thing and do it well» and interact with other programs with simple gadgets.

To organize a microservice architecture based on a monolithic one, it is proposed to use the technologies and software, which are used to create a client-server architecture on a JavaScript technology stack: the server-side consists of Node.js, Koa, MongoDB, Redis; the client-side was developed using technologies such as React.js, React-Bootstrap, React Redux, React Router, React Router DOM. The open source message broker NATS, an open source messaging system, was used to communicate with the microservice modules. The NATS server is written in the Go programming language. Figure 1 shows the operation and structure of the system.

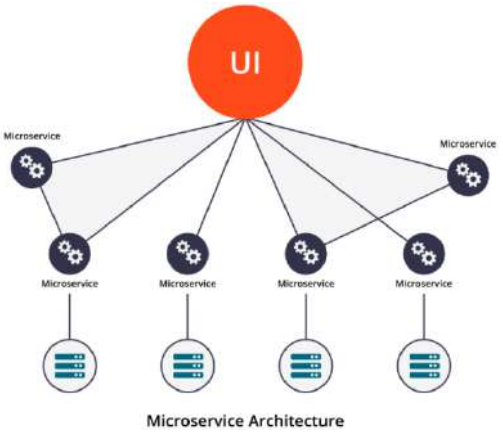

Fig. 1. The operation and structure of the system

The presented system for expanding the monolithic architecture is cross-platform and runs under Windows and Unix / Linux operating systems. The operation of the system on these browsers has been tested in a practical way.

Compared to the older version of the program, the microservice enhancement provides the application with faster startup times and the ability to deploy microservices independently, which is really beneficial for CI / CD [2, p. 220].

Microservices are not tied to technologies that are used in other services. This means that it is possible to use the best fitting techniques. Old services can be quickly rewritten to use new technologies. The ability to quickly adapt to new technologies can keep projects relevant for a long time and expand their potential and capabilities.

Application scalability is the potential for an application to grow over time – being able to efficiently handle more and more requests per minute (RPM) [3, p. 24]. It's not just a simple tweak you can turn on/off**,** it's a long-time process that touches almost every single item in your stack, including both the hardware and software sides of the system.

In case of problems, you can keep adding new CPUs or increase memory limits, but by doing so, you're just increasing the throughput, not the app performance.

It's not the way you should stick to when you see your app is starting to have efficiency problems. Scaling the app is not an easy thing and thus you should know your app very well before starting to think about how and when to scale it.

Scaling can be vertical and horizontal, but in this project the task is to optimize the application without affecting the hardware part of the project, but to use software tools to increase performance. In the course of the project, vertical and horizontal scaling will be used.

Vertical Scaling

The vertically scale means adding resources to our existing server or to replace it with another powerful server. Figure 2 shows how vertical scaling works [6, p. 2].

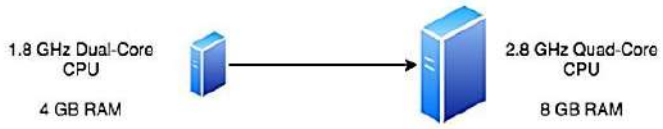

Fig. 2. Vertical scaling

Basically, the architecture remains the same, it's just that the server has been upgraded to accommodate more clients than before. This would solve our issue temporarily, but it's not a permanent fix. As more and more people begin to use web application, the added resources would eventually run out and it would need to keep on vertically scaling the server.

Horizontal Scaling

The horizontally scale means adding additional servers that serve the same purpose [6, p. 3]. As application continues to get popular day by day, the current servers exhaust out of resources by supporting all the clients, thus it needs to add more servers to serve other incoming clients. Figure 3 shows how horizontal scaling works

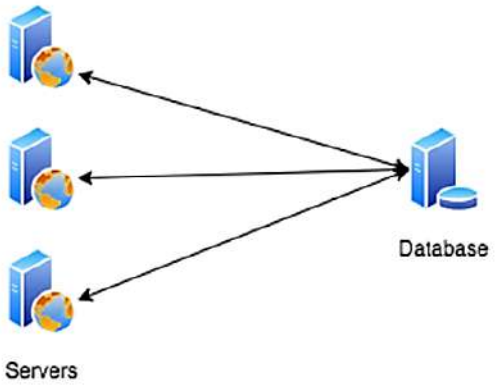

Fig. 3. Horizontal scaling

In the course of the project, these two types of scaling will be compared, and in the course of the results an analysis of the effectiveness of a particular method will be carried out.

Thus, a microservice system for creating and reading works of art, which consists of a server and a client part, which are supplemented with microservice modules has been designed. Application operation and performance are optimized by the selected scaling methods, which reduces the workload on the project. Unlike the old version of the application, the new one is characterized by a faster data processing speed, has many options for improvement and is easy to maintain.

References:

1. Kocher Parminder Singh Microservices and Containers 1st Edition. Addison-Wesley Professional; 1st Edition. 2018. 304 p.

2. Martin Kleppmann Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 590 p.

3. Fernando Doglio Scaling Your Node.js Apps: Progress Your Personal Projects to Production-Ready. Apress; 1st ed. Edition. 2018. 177p.

4. Newman Sam Building Microservices. 2015. Sebastopol: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

5. Eyal Ronel Auto Scaling Real-Time NodeJS Applications on AWS – The last tutorial you'll need!. 2017. URL:

https://medium.com/@eyalronel1984/auto-scaling-real-time-nodejs-applications-on-aws-the-last-tutorial-youll-need-eba1d2c88a4c

6. Harith Javed Bakhrani Scaling Your Web Application. 2020. URL: https://medium.com/@harithjaved/scaling-your-web-application-693657ce333c

7. AWS Deploying Node.js applications to Elastic Beanstalk – Developer Guide. URL: https://docs.aws.amazon.com/elasticbeanstalk/ latest/dg/create_deploy_nodejs.html

DOI https://doi.org/10.30525/978-9934-588-79-2-1.5

МЕТОДИ РІШЕННЯ ЗАВДАНЬ ПЛАНУВАННЯ ПОВЕДІНКИ АГЕНТІВ В ІНТЕЛЕКТУАЛЬНИХ СИСТЕМАХ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ

Демченко Є. Я.
кандидат технічних наук,
начальник науково-дослідного відділу науково-методичного забезпечення розроблення і реалізації програм розвитку озброєння та військової техніки та державного оборонного замовлення Центрального науково-дослідного інституту озброєння та військової техніки Збройних Сил України
м. Київ, Україна

Вступ. Необхідність проектування інтелектуальних систем прийняття/підтримки прийняття рішень (СППР) при управлінні складними об’єктами і процесами різної природи обумовлюється безперервним зростанням їх складності з одночасним скороченням часу, що відводиться людині, яка приймає рішення (ЛПР) на аналіз проблемної ситуації, ідентифікацію виниклого відхилення від нормального (штатного) режиму функціонування об’єкту, пошук можливих коригувальних рішень по впливу на об’єкт (процес), оцінку наслідків прийнятих рішень і, нарешті, видачу команд на відпрацьовування необхідних впливів.

Реалізувати СППР у повному обсязі можна лише з використанням сучасних технологій проектування інтелектуальних систем, заснованих на концепціях розподіленого штучного інтелекту, мультиагентності, динамічних баз знань, нейронних мереж, хмарних обчислень.