Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 3 (17)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

UDC 681.3.06
DOI: https://doi.org/10.30837/ITSSI.2021.17.013

D. ZOLOTARIOV

MICROSERVICE ARCHITECTURE FOR BUILDING HIGH-AVAILABILITY DISTRIBUTED AUTOMATED COMPUTING SYSTEM IN A CLOUD INFRASTRUCTURE

The article is devoted to the research and development of a highly available distributed automated computing system by iterative algorithms based on the microservice architecture in a cloud infrastructure. The **subject** of the research is the practical foundations of building high-availability automated computing systems based on microservice architecture in a cloud-based distributed infrastructure. The **purpose** of the article is to develop and to substantiate practical recommendations for the formation of the infrastructure of a high-availability automated computing system based on the microservice architecture, the choice of its constituent elements and their components. The **task** of the work: to identify the necessary structural elements of a microservice automated computing system and to analyze the constituent components and functional load for each of them, set specific tasks for building each of them and justify the choice of tools for their creation. In the course of the research, **methods** of system analysis were used to decompose a complex system into elements and each element into functional components, and **tools**: information technologies Apache Kafka, Kafkacat, Wolfram Mathematica, nginx, Lumen, Telegram, Dropbox, and MySQL. As a **result** of the study, it was found that the system infrastructure should consist of: fault-tolerant interservice transport, a high-availability computing microservice, and communication microservices with end customers, which save or process the results. For each of them, recommendations are provided regarding the formation and selection of implementation tools. According to the recommendations, one variant of implementation of such system has been developed, the principles of its operation are shown and the results are presented. It has been proven that when using a Kafka queue it is efficient to publish batches of results rather than one at a time, which results to significant overhead on queue servers and data latency for its clients. Recommendations are given on the implementation of the CI/CD system to build a continuous cycle of adding and improving microservices. Conclusions. Practical foundations have been developed for the implementation of high availability distributed automated computing systems based on microservice architecture in a cloud infrastructure. The flexibility in processing the results of such a system is shown due to the possibility of adding microservices and using third-party analytical applications that support connection to the Kafka queue. The economic benefit of using the described system is shown. Future ways of its improvement are given.

Keywords: high availability; cloud technologies; distributed infrastructure; automated calculations; saving resources and funds; iterative algorithms; Mathematica; Kafka; Telegram.

Introduction

Recent years have been marked by the rapid development of software products built on microservice architecture: web services [1-2], Internet banking [3], data streaming [4-5], Internet of Things (IoT) [6-7] and others. Its advantages include: ease of construction, updating and modification of elements (microservices), ease of addition, removal and replacement due to the independence of operation – as a consequence, much greater flexibility in the development of the system as a whole. In addition, the distribution of infrastructure in such systems is an integral part: the elements are interconnected only through data delivery mechanisms – queue managers or other.

Therefore, it is not surprising that the advantages and prospects of such architecture are appreciated by more and more developers of complex services, and gradually bring its elements to non-profit areas. As shown in [8], the achievements of modern applied science are largely the result of interdisciplinary teams of scientists, researchers and engineers, resulting in the use of each member of the group familiar in its field tools for development, research and analysis that can be combined only through specialized interaction systems for the transfer of results between participants. Also, such systems that allow the exchange of research results in automatic mode in real or near real time, have become relevant in today's environment, when researchers may be distributed around the world or otherwise not be able to be with the team.

The introduction of calculation systems based on microservice architecture in scientific systems aims to give each participant the opportunity to process or generate data with the usual tools and transfer the result to other colleagues through a unified for all researchers' communication mechanism - queue manager. With the use of cloud IaaS-technologies, which allow to approach the construction of infrastructure by renting the technology park for the right time in the right amount and with the right resources, you can achieve maximum efficiency from the cost of deployment and maintenance of such a system.

This work is a logical continuation and extension of the ideas presented for the first time in [8]. This article focuses on the problem of building a system of automated calculations based on microservice architecture for the use of iterative algorithms based on cloud IaaS service, and is designed to use the widest range of both service microservices and end customers through the use of queue manager as a communication channel.

Iterative algorithms are chosen because their feature is the "natural" ability to save the state after each iteration – the results of all previously calculated iterations can be saved and loaded, thus completely restoring the saved state of the program for a certain step of the algorithm. "Window" algorithms, such as those used in [9-12], are best suited for these purposes. Non-iterative algorithms that are able to store their state and the results of intermediate calculations for processing can also be used.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2021. No. 3 (17)

The purpose of this article is to develop and substantiate practical recommendations for the formation of the infrastructure of such a system, the choice of its components and their components. The task is to identify the necessary structural elements and provide for each of them an analysis of the components and functional load, to justify the choice of tools for their construction.

General structure of the system

To build a system of automated calculations by iterative methods for microservice architecture, we identify the problems that need to be solved. These will be, first, the construction of inter-service transport for the transmission of near-real-time messages from one microservice to another. Secondly, launching the calculation program as the sole initiator of calculations and automatic supervision of its correct operation. Third, long-term storage of computational results in a random access system. And, fourth, maintaining backward compatibility with the customer system developed in [8].

To solve the problems we formulate problems in the form of characteristics of elements by directions.

Transport – is a channel of communication between microservices that generate or process data, transmits in real time the results of their work to other microservices, and is fault-tolerant in understanding the construction of a cluster and the failure of one of the servers.

Computing microservice – consists of a computing initiator and a computing server. The first – has secure network access to the second, launches the calculation program itself. The second – provides secure network access for the initiator of calculations, executes this program, instantly notifies of failure in the calculation process, transfers results of calculation of each iteration to interservice transport.

Long-term data storage microservice - accepts the results of calculations of each iteration from the transport, processes them for storage suitability in a relational DBMS and saves.

Microservice, which is responsible for maintaining backward compatibility with the system described in [8], - receives the results of calculations of each iteration from the transport, processes them for storage in separate files in the format of the specified system, saves and sends them to Dropbox.

Also, the system can have many other microservices, which is implied in the architectural approach. These can include both computational results processors and ancillary services: sending notifications to various communication channels, constructing graphical images of the calculation progress, and so on.

The block diagram of the basic implementation of such a system of calculations, which will be considered below, is shown in fig. 1. It should be noted that this is one of its possible implementations: each element of the system, including transport, can be replaced according to specific circumstances.

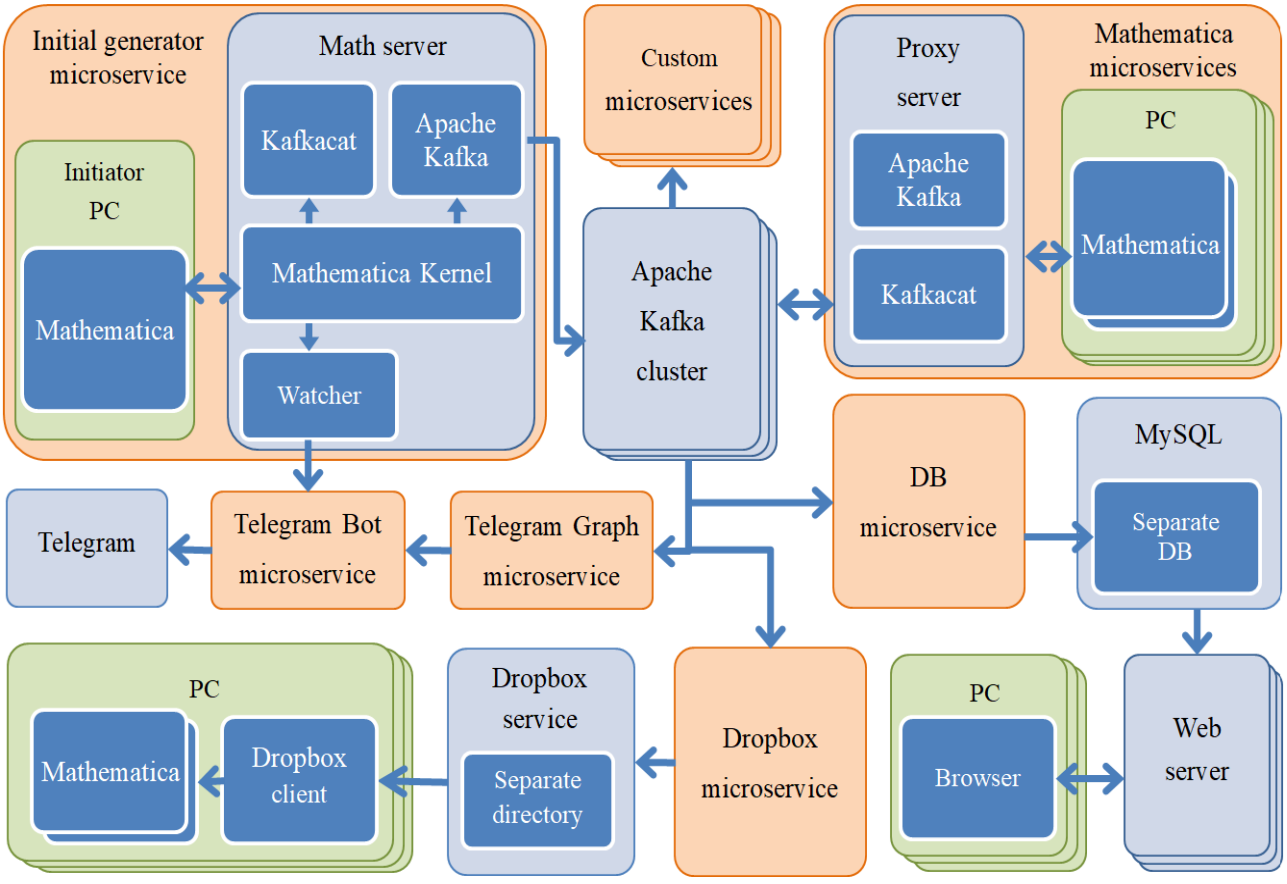

Fig. 1. Decomposition of the system of automated calculations into structural elements

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 3 (17) ISSN 2524-2296 (online)

The system consists structurally of Apache Kafka transport and subsequent microservices.

The main complex microservices "Initial generator microservice" is the center of computing, and "Mathematica microservices" are the main clients with mathematical processors.

Auxiliary microservices: "Telegram Bot microservice" is responsible for providing a mechanism for sending messages to the Telegram messenger, "Telegram Graph microservice" is responsible for displaying to Telegram in real time the progress of calculations and the form of text and graphs.

Designed for backward compatibility: "Dropbox microservice" is used to save the results of calculations on a third-party service, Dropbox and "DB microservice" is used to display progress in the web browser of the end customer.

The system also assumes in the future the possibility of connecting to the transport of other arbitrary microservices, which are indicated in fig. 1 as "Custom microservices", in unlimited quantities.

Let's consider each element of the system in more detail.

Transport

In the original work [8], a third-party Dropbox service was used as a transport, which has the main disadvantage of not being controlled by the administrator, and is a "bottleneck" of the whole system – in case of failure, the whole complex fails due to lack of communication. To avoid these shortcomings when using the queue manager, you need to achieve the following goals:

- use only own or leased servers,
- use a distributed server system that minimizes the possibility of their failure at the same time,
- as the software part to choose the most fault-tolerant products that work in real time,
- and those that are available for connection to customers from various fields of research and engineering.

The Apache Kafka real-time content delivery platform meets all these requirements. This is one of the most frequently used [14] and fault-tolerant queue managers, which allows you to build a distributed system of brokers, able to dynamically adapt to the load and, if necessary, easily scale both vertically and horizontally. It has many libraries already built to connect to the queue of different software packages and open mechanisms for developing their applications in different programming languages.

To be able to run smoothly on N servers from the queue cluster, the cluster must include at least 2N + 1 servers, which is also noted in the Kafka documentation.

It is also worth noting that the use of the queue manager as a delivery mechanism is not the only way to use it. With the correct queuing time, it can also be used for easy and fast operation of the calculation results storage mechanism, which guarantees their delivery to the client even when connected to the queue with a significant delay or even after the publication of data from the source. And gives the chance, having saved once results of calculation, to process them by various analytical and statistical software complexes, without disturbing the computing microservice or even having excluded it from system.

As a result, the transport configured in this way allows easy duplication of the microservice of calculations in case of failure of the working copy, and guarantees data preservation even in the event of a physical accident on any of the components of the microservice of calculation.

Microservice of calculations

The Wolfram Mathematica mathematical processor [15] was chosen as a tool for building a computational microservice through internal client-server architecture: Mathematica client and WolframKernel kernel - and server platform independence. It is one of the world leaders in the field of symbolic and numerical data processing and is used in almost every field of engineering and science, which is clearly seen, for example, in publications [16-18], where it is used to solve various application technologies.

The generator is a complex system, built by analogy with that in [8], consisting of a computer-initiator of calculations with a client of the mathematical processor Wolfram Mathematica and a calculation server based on the Mathematica kernel. It has network access between them organized through the ssh-channel using asymmetric encryption keys, which ensure the security of command transmission and reception of results from the computing server.

Monitoring the operation of the Mathematica core on the server with instant notification in case of failure is carried out in the same way as in the original work [8], with the addition of using as a notification channel Telegram messenger, which is connected using the microservice "Telegram Bot microservice ».

As shown in [19], the Wolfram Mathematica processor does not have built-in mechanisms to connect to the Kafka queue cluster. This opportunity to work in two directions: publishing messages in the queue and consuming messages from it - provides the use of a package with [19] with the extension [20] for use on a computing server. To run it on the computing server, Apache Kafka is additionally installed as a queue client and third-party Kafkacat software [21].

The following principles are used to construct the interaction of the Mathematica kernel with the Kafka queue manager. A separate queue is created for each calculation, which in its name has all the input parameters for unambiguous identification "calculation - queue". When you restart calculations, if such a queue already exists, it is cleared. This approach allows you to have only one source of truth for the results of calculations. All additional data to be stored with the iteration result is added to the headers fields of the message being written to the queue.

The system intentionally does not use the ability to store calculation results locally in dump files due to the use of the DumpSave function in Mathematica. This is

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 3 (17)

done to follow the paradigm of the only source of truth, which is the turn of Kafka. Of course, this approach leads to a longer recovery of the program while continuing the interrupted calculation.

The protocol of the calculation program is stored on a hybrid basis. On the computation initiator, it is output to the document and automatically saved with it using the Mathematica NotebookSave function or the NotebookAutoSave document option. On the computing server it is stored locally in separate files without access from external systems, quite similarly [8]. Despite the fact that the microservice does not have a built-in possibility of external use of the work protocol, it can be implemented according to [8] through a specially created application based on bash-scripts or other tools. This will allow it to be duplicated in Dropbox files, in a separate database table or even in a separate queue Kafka - in each case, the advantage will be the formation of a remote independent of the microservice computing copy, which is always stored, and access to it does not load the calculation server.

Microservices based on mathematical processors

For random processing of calculation results, the system may have microservices based on mathematical processors. Which allow flexible processing of results using all available arsenal of built-in analytical and numerous tools, various processing and analysis tools, as well as changing algorithms of such processing at any time. To implement this capability, it is necessary to have an appropriate mechanism for connecting the mapacket to the Kafka queue.

For example, MathWorks MATLAB already has such a module [22] with a rather rich functionality for publishing and consuming messages from the queue.

For the Wolfram Mathematica package, such a package was developed in [20], which allows a group of similar clients to connect to the queue through a single proxy server. For individual connection to the queue without additional implementation of the proxy server, you can use the method described in [19]. But it requires additional software to be installed locally on the client computer.

Such microservices can have a similar [8] functional load, and another depending on the challenges facing researchers at the moment. As noted above, these microservices can appear in the system and be excluded from it if necessary without affecting its other elements.

The disadvantage of such microservices is that access to the analysis results is available only on the computer where they were obtained. To provide access to them to other microservices or end customers, they can publish their results to Kafka queues specially designated for such tasks.

Backward Compatibility Microservices

The source system describes [8] the processing and visualization of the calculation process on client computers in close to real time is implemented in two ways. First, through a Mathematica-based client that processes files obtained through a third-party Dropbox file service into the computer's local directory. Second, a browser on the end-user device connecting to the web server connects to the database server where the calculation results are stored.

The following two microservices have been added to maintain backward compatibility with the previous system [8].

To display the calculation progress in the client browser, a microservice "DB microservice" has been added to the system, which in real time receives the results of calculating each iteration from the queue, parses them, prepares them for inclusion in the database and stores them in a dedicated database. Web servers are already connected to the latter, which are described in detail in [8], through which end users receive an integrated environment in a mobile or computer browser with display of graphic and text results.

The importance of maintaining this backward compatibility is due to the fact that through the multi-client architecture that is present in any web server, the number of clients connected through the browser is limited only by the power of the server and can be increased if necessary. In addition, the number of web servers connected to a single database is also limited only by the power of the latter. That is, despite the fact that the tool for analyzing results on the web server and in the browser is very limited, this is compensated by the speed of processing requests and the lack of load on the central microservices and the Kafka queue.

The second microservice is designed to preserve the ability of clients to work based on processing files containing the results of each iteration calculation. "Dropbox microservice" in real time receives data from the queue, formats it to write to a file, and saves these files on a third-party Dropbox file service according to the same principles as described in [8].

Maintaining this backward compatibility is important because Dropbox does not limit not only the number of service clients, but also the number of simultaneously running applications on one client, parallel processing the same data in different ways. That is, the same – this microservice and its customers load other microservices and the Kafka queue.

Auxiliary Microservices

The proposed system architecture has two auxiliary microservices.

The first microservice "Telegram Bot microservice" is responsible for providing a mechanism for sending messages to the Telegram messenger and processing feedback from the user. Messages with text or image can be sent through it, and already existing messages can be updated (replaced).

Microservice in the process of work refers to a specially registered bot in Telegram. It is important to note the security side: at the moment, Telegram does not provide for private bots, so privacy is achieved due to the

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 3 (17) ISSN 2524-2296 (online)

restriction of message processing only from certain Telegram users.

Unlike all others in the system, this microservice does not work through the Kafka queue, but through REST access using the HTTPS protocol. It is used, among other things, to work as a notification mechanism for starting the Mathematica kernel, closing it, and failing the Initial generator microservice calculation server.

To report real-time calculation progress in the form of graphs and text notifications, the Telegram Graph microservice is used, which is a constantly running set of systemd system services, with an automatic restart in case of failure.

Each service is responsible for monitoring a separate calculation (queue in Kafka): it receives messages from its Kafka queue, processes them, creating notifications about the calculation progress, and sends them to Telegram through the use of the capabilities of the previous microservice. At the first start - saves the identifiers of these Telegram messages, and in future work updates them. This achieves compactness and information for the end user.

The microservice has the following capabilities for operating services: Adds a service to a specific Kafka queue that creates a service description file and adds it to systemd but does not start. Start and stop running a previously added service for a specific Kafka queue. Deletes the service for a specific Kafka queue from the systemd and its description file. Also displays a description of all added services with queue names and the current status obtained by executing the systemctl status command.

Other Microservices

The set of microservices is not limited to those described above. Arbitrary consumers can be connected to the Kafka queue, having an integrated queue connection mechanism and the ability to process messages in JSON format.

It is also possible to develop individual-purpose microservices through the use of libraries created by Kafka developers and other users for almost every of the common programming languages [23]: C, PHP, Python, Go, Node.js and others.

Notes on iterative algorithms

Finally, we provide remarks on improving the efficiency of the considered system for iterative algorithms of computational programs.

The proposed system is based on a microservice architecture and allows simultaneous parallel processing in real time of the results of calculating each iteration by a set of distributed clients. With the ability to stop, restart and replace at any time of each client microservice system, including central microservice computing. The latter is possible due to the automatic full recovery of the iterative algorithm of the program after loading from the queue the results of all previously calculated iterations.

In the case of missing or damaged iteration results, the program must take only the results up to the first correct and full inclusive, and discard others. Its implementation will begin so that the allegedly rejected iterations are not yet calculated.

This way of working with iteration results is logging for high availability. This principle is well described by the authors of Apache Kafka: "the process that performs local calculations can be made fault-tolerant by leaving the changes that it makes to its local state so that another process can reload these changes and continue if it fails" [13]. This principle is used in the proposed system to significantly increase its reliability and fault tolerance of calculations.

It should also be borne in mind that in cases of short-term interruption of a network connection, data may be duplicated in the queue due to re-sending by the microservice. To solve this problem, you need to save the results of calculations in a way that relies not on order, but on the unambiguous difference between the results of two arbitrary iterations.

To create a more economical system, you can store the intermediate data of each step of the current iteration (and only its) in a separate queue, designed only to restart the calculations in case of a failure. This allows you to load the results of all the calculated iterations and the results of the intermediate steps of the last iteration from the queue, and fully restore the saved state of the program with accuracy to a certain step of the algorithm.

Demonstration of the work

To demonstrate the operation of the proposed system, cloud technologies were used as a basis for infrastructure construction. Choosing them has the following key benefits: Servers can be quickly deployed and quickly configured to meet the needs of the task. One of the world leaders in cloud technologies DigitalOcean, where servers based on Ubuntu 20.04 LTS x64 OS were located, was chosen as the capacity:

- Kafka queue cluster servers in the amount of 3 pieces based on Apache Kafka 2.6.0,
- computing server with Wolfram Mathematica 11.3.0 for Linux x86 (64-bit), Kafkacat 1.5.0 and Apache Kafka 2.6.0, built according to [8,20] for a computing microservice,
- intermediary server for queuing with Kafkacat 1.5.0 and Apache Kafka 2.6.0, built according to [20] for microservices "Mathematica microservices",
- server based on Lumen 8.x framework and MySQL 8.0.20 Community Server database for DB microservice,
- server based on nginx 1.18.0 web server and Lumen 8.x framework for REST-microservice "Telegram Bot microservice",
- server based on the Lumen 8.x framework for Dropbox microservice and Telegram Graph microservice, the latter using the high-performance Amenadiel \ JpGraph php library, which is responsible for plotting near real time.

The capabilities of the implemented microservice "Telegram Graph microservice" are presented in fig. 2 and

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 3 (17)

fully comply with the requirements set out above and include the following: "Add" - adding a service for a specific queue Kafka, returns an error if the queue does not exist. "Start" / "Stop" - starts / stops the previously added service for the specified Kafka queue. "Remove" - removes the service for the specified Kafka queue. "Status" - returns the status of all services.

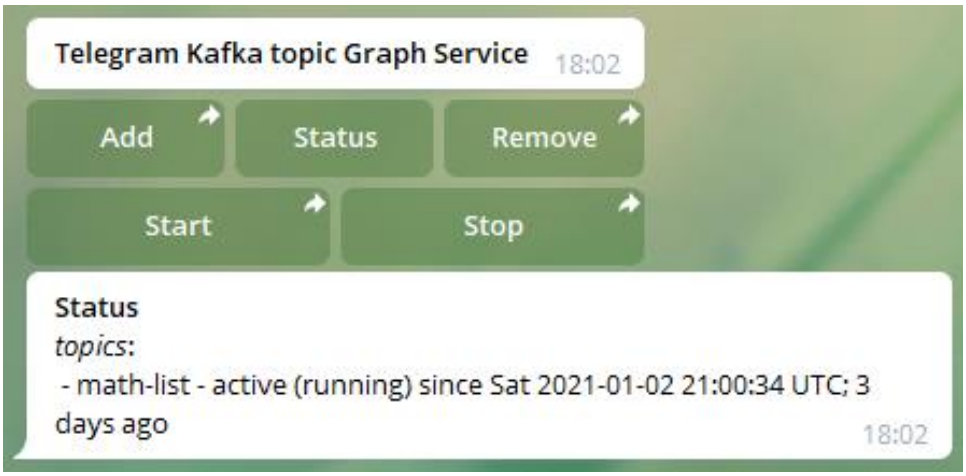

Fig. 2. Capabilities of microservice "Telegram Graph microservice"

The result of the Status command, which returned a description of the only added service instance to handle messages from the math-list queue that is currently running, is shown at the bottom of figure 2.

The calculation of the Airy momentum propagation [10,12] in a one-dimensional plane-layered medium by the method of approximating functions [9,11] is chosen as a problem. Simulation area: normalized time t = [0, 100] and normalized space x = [0, 1], simulation step h = 0.01. The result of solving the problem is the value of the desired function in the nodes of the simulation grid.

The intermediate progress of calculating the task built by the Telegram Graph microservice for the slice at x = 0.6 is shown in figure 3 below. The graphs are obtained at arbitrary times and are arranged chronologically as follows: from the top left, from the bottom left, from the right - and are displayed by replacing each other in a single message of the Telegram messenger, which is pointed to the right.

Graphs are built in two types at once and combined on one image for easy reading: red dots for each individual value of the function (grid node), and blue lines for clarity. Fig. 3 (right) shows a complete graphical Telegram message, which includes: a dynamically updated schedule, action buttons with the service (from left to right: start, get current status, stop). The buttons duplicate the same from fig. 2, but are designed for instant action, because they do not require entering the name of the service in contrast to the universal in fig. 2.

At the top of each graph is given the information about the name of the queue from which the data is taken, and the number of points - the values of the function present on the graph. Due to the fact that the microservice is designed to be universal, only the ordinal numbers of the graph points are given on the abscissa axis without using the real coordinates of the modeled area. This can be changed as needed, as the graphics library itself supports scaling of various kinds and relative markers on the coordinate axes.

After testing the system on a complete simulation and obtaining a complete solution for the problem, the experiment was continued in the direction of stress testing. To do this, the Kafka queue cluster was replaced by a single server with 1GB of RAM, a single processor core and an SSD - the minimum available server configuration on DigitalOcean. The Apache Kafka process was allocated RAM of at least 150MB. Using the solution of the initial problem obtained above allowed to send the finished result to the queue as quickly as possible without the cost of calculation. Thus, the testing of the system is artificially achieved at maximum loads on the queue manager and microservices-consumers.

The essence of the experiment is as follows. The initiator of the calculation selects a "window" of simulation, which contains 1000 point values of the desired function in the nodes of the grid, and publishes up to the queue 1000 messages with a single point value. Pauses for 1 minute, shifts the "window" to the next and repeats the operation. This algorithm emulates a real calculation on a high-performance workstation or server, where during one iteration lasting 1 minute, the solution of the problem for 1000 nodes of the simulation grid is obtained. The experiment was repeated 5 times to exclude error with the same result.

On the message generator side, 1000 messages are published almost instantly. On the side of the Telegram Graph microservice and Mathematica microservices, consumption is also almost real. But queuing messages takes a few minutes for every 1000 messages, and the Kafka process CPU load on the queue server is constantly around 17%. Confirmation that the system bottleneck is the queue server is provided by the fact that after full processing of all messages by the queue manager, the above microservices were restarted for the purity of the experiment and showed almost instantaneous retrieval of all values from the queue and their processing.

Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 3 (17)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

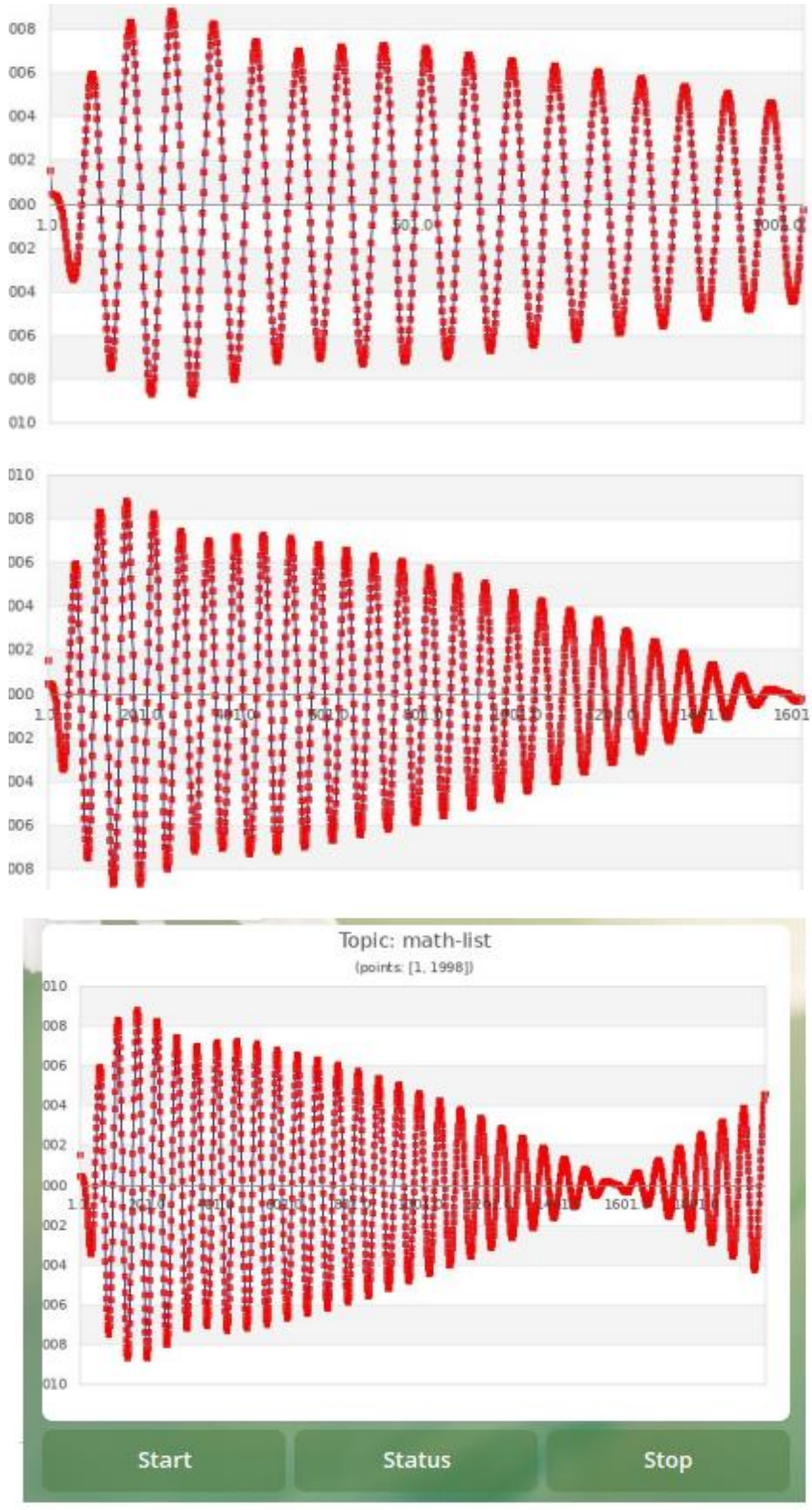

Fig. 3. Display of calculation progress in the form of graphs to Telegram through the microservice "Telegram Graph microservice"

The conclusions from the experiment are as follows. Improving the performance of the computing server will lead to the need to improve the performance of the Kafka cluster in the first place, and only in the second – the notification microservice or clients implemented on the basis of the mathematical package Mathematica. Publishing a large number of small messages is not effective for the Kafka queue manager, who works in compressed resource conditions.

The next step in the experiment was to change the publishing method to the opposite: the values of the function for one "window" of 200 points with the same pause were completely published once in a row. To create such a data package, you must first convert an array of

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 3 (17)

values to a JSON object by using the following construct in Mathematica.

```
str = ExportString[arr, "JSON", Compact -> True];
```

On the Mathematica client side of the queue, the inverse transformation takes the form:

```
json = ImportString[str, "JSON"];
payload = "payload" /. json;
points_array = ImportString[payload, "JSON"];
```

This approach put a negligible load on the Kafka queue manager on the CPU of the queue server with a peak value of 5-6% for 1-2 seconds. As a result, all customers received data in near real time. The speed of micro-client services and micro-service notifications remained at the same level. To confirm the result, the experiment was also repeated 5 times.

The conclusion from the experiment is as follows. Batch data acquisition and analysis by the above microservices does not lead to significant delays. Unlike a single publication of the results of calculations to the Kafka queue, batch publishing gives an insignificant and inconspicuous load on the servers of the queue and in no way slows down the work of its clients, so this method of publishing in the system can use much cheaper servers.

Prospects for further development of the system

Prospects for further development of the developed system are the following architectural improvements.

Transfer the proxy server from the mechanism of interaction with clients described in [20] to RESTfull interaction according to the HTTPS protocol. A similar mechanism has already been developed in [24-25], although at the moment none of them has built-in capabilities for queuing - only publishing or receiving messages.

Microservice architecture allows easy dynamic addition and change of its elements, often situational - for a specific task of analysis or data processing, followed by removal of the service from the system. Therefore, it is important to anticipate that the proposed system will dynamically add new elements and improve existing ones. To facilitate this process, it is promising to build a specially designed CI / CD system that will meet modern requirements for the creation, testing and implementation of software and infrastructure components.

Conclusions

The ideas developed in [8] on the formation of a system of flexibly interconnected elements were further developed in the work. Developed and substantiated practical bases for the implementation of a system of distributed automated calculations of high availability by iterative algorithms based on microservice architecture in the cloud infrastructure.

One of the possible options for such a system is implemented, consisting of: Apache Kafka distributed content delivery platform as high-reliability interservice transport, calculation microservice based on the core of the Mathematica math package, microservices for processing calculation results, including based on this matpack, microservice of visual notification of the calculation progress to the Telegram messenger, REST microservice of interaction with Telegram bot, and microservices of saving results to the database and Dropbox.

The backward compatibility with the elements of the original system [8] is preserved for easier replacement of the latter with the developed one.

It is explained that the proposed system has the greatest advantages when using iterative algorithms to build a calculation program that allows it to be interrupted at any step and continue from the last completed iteration, and with the described additional state savings - and the last interrupted iteration step.

The approaches described in the article to the formation of the system have increased the reliability of saving the results of calculation through the use of a failover queuing cluster and their duplication in Dropbox microservices and databases. Analyze and process data by various means and on different servers or computers with the ability to connect to the queue, provide user-friendly remote monitoring of computing progress and response to computing microservice failures through any smartphone or tablet.

It has been proven that publishing a large number of small messages is not effective for the Kafka queue manager operating under compressed resource conditions and leads to significant resource overruns of queue servers and data delays for queue clients. It is shown that batch publishing is much more profitable due to the fact that it gives an insignificant load on queue servers, does not slow down the work of its clients and provides the possibility of using less powerful, that is cheaper, queue servers.

It is shown that using the developed version of the proposed system, increasing the performance of the computing server will lead to the need to increase the performance of the Kafka cluster earlier than microservices clients.

The flexibility of the proposed system is due to the possibility of dynamically adding and eliminating situationally necessary microservices at an arbitrary moment of time without impeding the operation of the entire system and its individual elements. Prospects of development of this peculiarity are shown through implementation of CI/CD system for continuous improvement of microservices.

The economic benefit of using the described system is achieved through maximum automation, which leads to improved quality of work of researchers. Also due to the reduction of computer costs of employees - each of them has the opportunity to conduct research on the results of calculations on available and convenient for him equipment, if it has access to the queue. Due to the use of cloud technologies, the cost of ownership of the system infrastructure and the cost of its modification are the lowest.

Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 3 (17) *ISSN 2522-9818 (print)* *ISSN 2524-2296 (online)*

References

1. Lichtenthäler, R., Prechtl, M., Schwille, C. et al (2020), "Requirements for a model-driven cloud-native migration of monolithic web-based applications", *SICS Softw.-Inensiv. Cyber-Phys. Syst*., Vol. 35, P. 89–100. DOI: https://doi.org/10.1007/s00450-019-00414-9
2. Fernández-García, A. J., Iribarne, L., Corral, A. et al. (2019), "A microservice-based architecture for enhancing the user experience in cross-device distributed mashup UIs with multiple forms of interaction", *Univ Access Inf Soc*, Vol. 18, P. 747–770. DOI: https://doi.org/10.1007/s10209-017-0606-0
3. Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., Mazzara, M. (2018), "From Monolithic to Microservices: An Experience Report from the Banking Domain,", *IEEE Software*, Vol. 35, No. 3, P. 50–55, DOI: 10.1109/MS.2018.2141026.
4. Alaasam, A. B., Radchenko, G., Tchernykh, A. et al. (2020), "Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing", *Program Comput Soft*, Vol. 46, P. 511–525. DOI: https://doi.org/10.1134/S0361768820080083
5. Kim, Y. K., Kim, Y., Jeong, C. S. (2018), "RIDE: real-time massive image processing platform on distributed environment", *EURASIP Journal on Image and Video Processing*, Vol. 2018, No. 39. DOI: https://doi.org/10.1186/s13640-018-0279-5
6. Santana, C., Andrade, L., Delicato, F.C. et al. (2020), "Increasing the availability of IoT applications with reactive microservice", *SOCA*. DOI: https://doi.org/10.1007/s11761-020-00308-8
7. Razzaq, A. A (2020), "Systematic Review on Software Architectures for IoT Systems and Future Direction to the Adoption of Microservices Architecture", *SN COMPUT. SCI*, Vol. 1, Article 350. DOI: https://doi.org/10.1007/s42979-020-00359-w
8. Zolotariov, D. (2020), "The distributed system of automated computing based on cloud infrastructure", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (14), P. 47–55. DOI: https://doi.org/10.30837/ITSSI.2020.14.047
9. Zolotariov D., Nerukh A. (2011), "Extension of the approximation functions method for 2d nonlinear Volterra integral equations", *Applied Radio Electronics*, Vol. 10, No. 1, P. 39–44.
10. Nerukh, A. G., Zolotariov, D. A., Nerukh, D. A. (2012), "Properties of decelerating non-diffractive electromagnetic Airy pulses", *Applied Radio Electronics*, Vol. 11, No. 1, P. 77–81.
11. Nerukh, A., Zolotariov, D., Benson T. (2015), "The approximating functions method for nonlinear Volterra integral equations", *Optical and Quantum Electronics*, Vol. 47, P. 2565–2575. DOI: https://doi.org/10.1007/s11082-015-0141-2
12. Nerukh, A., Zolotariov, D., Kuryzheva, O., Benson T. (2016), "Dynamics of decelerating pulses at a dielectric layer", *Optical and Quantum Electronics*, Vol. 48, No. 89. DOI: https://doi.org/10.1007/s11082-016-0386-4
13. Apache Kafka (2021), "Apache Kafka", available at: https://kafka.apache.org/documentation/ (last accessed 10 January 2021).
14. GitHub (2021), "Ultimate Comparison", available at: https://ultimate-comparisons.github.io/ultimate-message-broker-comparison/ (last accessed 10 January 2021)
15. Wolfram (2021), "Wolfram Mathematica: Modern technical calculations", available at: https://www.wolfram.com/mathematica/ (last accessed 10 January 2021).
16. You, X., Chen, D. R. (2018), "A new sequence convergent to Euler–Mascheroni constant", *Journal of Inequalities and Applications*, Vol. 2018, Article 75. DOI: https://doi.org/10.1186/s13660-018-1670-6
17. Ghorbani, M. A., Singh, V. P., Sivakumar, B. et al. (2017), "Probability distribution functions for unit hydrographs with optimization using genetic algorithm", *Applied Water Science*, Vol. 7, P. 663–676. DOI: https://doi.org/10.1007/s13201-015-0278-y
18. Rehman, S., Idrees, M., Shah, R. A. et al. (2019), "Suction/injection effects on an unsteady MHD Casson thin film flow with slip and uniform thickness over a stretching sheet along variable flow properties", *Boundary Value Problems*, Vol. 2019, No. 26. DOI: https://doi.org/10.1186/s13661-019-1133-0
19. Zolotariov, D. (2021), "The mechanism for creation of event-driven applications based on Wolfram Mathematica and Apache Kafka", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (15), P. 53–58. DOI: https://doi.org/10.30837/ITSSI.2021.15.053
20. Zolotariov, D. (2021), "The platform for creation of event-driven applications based on Wolfram Mathematica and Apache Kafka", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (16), P. 12–18. DOI: https://doi.org/10.30837/ITSSI.2021.16.012
21. GitHub (2021), "edenhill/kafkacat: Generic command line non-JVM Apache Kafka producer and consumer", available at: https://github.com/edenhill/kafkacat (last accessed 10 January 2021).
22. GitHub (2021), "mathworks-ref-arch/matlab-apache-kafka: MATLAB Interface for Apache Kafka", available at: https://github.com/mathworks-ref-arch/matlab-apache-kafka (last accessed 10 January 2021).
23. Apache Software Foundation (2021), "Clients - Apache Kafka", available at: https://cwiki.apache.org/confluence/display/KAFKA/Clients (last accessed 5 January 2021).
24. GitHub (2021), "dspeterson/dory: Producer daemon for Apache Kafka", available at: https://github.com/dspeterson/dory (last accessed 10 January 2021).
25. Confluent Documentation (2021), "Confluent REST APIs", available at: https://docs.confluent.io/platform/current/kafka-rest/index.html (last accessed 10 January 2021).

Received 23.07.2021

Відомості про авторів / Сведения об авторах / About the Authors

Золотарьов Денис Олексійович – кандидат фізико-математичних наук, Харків, Україна; email: denis@zolotariov.org.ua, ORCID: https://orcid.org/0000-0003-4907-7810.

Золотарёв Денис Алексеевич – кандидат физико-математических наук, Харьков, Украина.

Zolotariov Denis – PhD (Physics and Mathematics Sciences), Kharkiv, Ukraine.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 3 (17)

МІКРОСЕРВІСНА АРХІТЕКТУРА ПОБУДОВИ РОЗПОДІЛЕНИХ АВТОМАТИЗОВАНИХ ОБЧИСЛЕНЬ ВИСОКОЇ ДОСТУПНОСТІ У ХМАРНІЙ ІНФРАСТРУКТУРІ

Стаття присвячена дослідженню та розробці розподіленої системи автоматизованих обчислень високої доступності ітераційними алгоритмами на базі мікросервісної архітектури у хмарній інфраструктурі. **Предметом** дослідження є практичні засади побудови систем автоматизованих розрахунків високої доступності, що засновані на мікросервісній архітектурі у розподіленій інфраструктурі на базі хмарних технологій. **Метою** статті є розробка та обґрунтування практичних рекомендацій щодо формування інфраструктури системи автоматизованих обчислень високої доступності на базі мікросервісної архітектури, вибору її складових елементів та їх компонентів. **Завдання** роботи: виявити необхідні структурні елементи мікросервісної системи автоматизованих обчислень та надати для кожного з них аналіз складових компонентів та функціонального навантаження, поставити конкретні задачі для побудови кожного із них та обґрунтувати вибір інструментів для їх вирішення. У ході дослідження використано **методи** системного аналізу для декомпозиції складної системи на елементи та кожного елемента на функціональні компоненти, та засоби: інформаційні технології Apache Kafka, Kafkacat, Wolfram Mathematica, nginx, Lumen, Telegram, Dropbox, MySQL. У **результаті** дослідження встановлено, що інфраструктура системи має складатися з: відмовостійкого міжсервісного транспорту, мікросервісу обчислень високої доступності, та мікросервісів зв'язку із кінцевими клієнтами, що зберігають або обробляють результати. Для кожного з них надані рекомендації щодо формування та вибору інструментарію для побудови. За отриманими рекомендаціями розроблений один із варіантів такої системи, показані принципи її роботи та наведені результати. Доведено, що при використанні черги Kafka ефективною є публікація пакетів результатів, а не по одному, що призводить до значних перевитрат ресурсів серверів черги та затримкам даних для її клієнтів. Дані рекомендації щодо впровадження системи CI/CD для побудови безперервного циклу додавання та вдосконалення мікросервісів. **Висновки.** Розроблені практичні основи для реалізації систем розподілених автоматизованих обчислень високої доступності на базі мікросервісної архітектури у хмарній інфраструктурі. Показана гнучкість у обробці результатів такої системи через можливість доповнення її мікросервісами та використання сторонніх аналітичних додатків, що підтримують завантаження даних із черги Kafka. Показана економічна вигода від використання описаної системи. Наведені майбутні шляхи її вдосконалення.

Ключові слова: висока доступність; хмарні технології; розподілена інфраструктура; автоматизовані обчислення; економія ресурсів та коштів; ітераційні алгоритми; Mathematica; Kafka; Telegram.

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА ПОСТРОЕНИЯ РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ ВЫЧИСЛЕНИЙ ВЫСОКОЙ ДОСТУПНОСТИ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

Статья посвящена исследованию и разработке распределенной системы автоматизированных вычислений высокой доступности итерационными алгоритмами на базе микросервисной архитектуры в облачной инфраструктуре. **Предметом** исследования являются практические основы построения систем автоматизированных вычислений высокой доступности, основанных на микросервисной архитектуре в распределенной инфраструктуре на базе облачных технологий. **Целью** статьи является разработка и обоснование практических рекомендаций к формированию инфраструктуры системы автоматизированных вычислений высокой доступности на базе микросервисной архитектуры, выбору ее составных элементов и их компонентов. **Задача** работы: выявить необходимые структурные элементы микросервисной системы автоматизированных вычислений и провести для каждого из них анализ составляющих компонентов и функциональной нагрузки, поставить конкретные задачи для построения каждого из них и обосновать выбор инструментов для их решения. В ходе исследования использованы **методы** системного анализа для декомпозиции сложной системы на элементы и каждого элемента на функциональные компоненты, и средства: информационные технологии Apache Kafka, Kafkacat, Wolfram Mathematica, nginx, Lumen, Telegram, Dropbox, MySQL. В **результате** исследования установлено, что инфраструктура системы должна состоять из: отказоустойчивого межсервисного транспорта, микросервиса вычислений высокой доступности, и микросервисов связи с конечными клиентами, которые сохраняют или обрабатывают результаты. Для каждого из них предоставлены рекомендации относительно формирования и выбора инструментов реализации. Согласно полученным рекомендациям разработан один из вариантов реализации такой системы, показаны принципы его работы и приведены результаты. Доказано, что при использовании очереди Kafka эффективной является публикация пакетов результатов, а не по одному, что приводит к значительному перерасходу ресурсов серверов очереди и задержкам данных для ее клиентов. Даны рекомендации относительно внедрения системы CI/CD для построения непрерывного цикла добавления и совершенствования микросервисов. **Выводы.** Разработаны практические основы для реализации систем распределенных автоматизированных вычислений высокой доступности на базе микросервисной архитектуры в облачной инфраструктуре. Показана гибкость в обработке результатов такой системой благодаря возможности дополнения ее микросервисами и использования сторонних аналитических приложений, которые поддерживают загрузку данных из очереди Kafka. Показана экономическая выгода от использования описанной системы. Приведены будущие пути ее усовершенствования.

Ключевые слова: высокая доступность; облачные технологии; распределенная инфраструктура; автоматизированные вычисления; экономия ресурсов и денег; итерационные алгоритмы; Mathematica; Kafka; Telegram.

Бібліографічні описи / Bibliographic descriptions

Золотарьов Д. О. Мікросервісна архітектура побудови розподілених автоматизованих обчислень високої доступності у хмарній інфраструктурі. *Сучасний стан наукових досліджень та технологій в промисловості*. 2021. № 3 (17). С. 13–22. DOI: https://doi.org/10.30837/ITSSI.2021.17.013

Zolotariov, D. (2021), "Microservice architecture for building high-availability distributed automated computing system in a cloud infrastructure", *Innovative Technologies and Scientific Solutions for Industries*, No. 3 (17), P. 13–22. DOI: https://doi.org/10.30837/ITSSI.2021.17.013