ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 2 (16)

UDC 681.3.06 DOI: https://doi.org/10.30837/ITSSI.2021.16.012

D. ZOLOTARIOV

THE PLATFORM FOR CREATION OF EVENT-DRIVEN APPLICATIONS BASED ON WOLFRAM MATHEMATICA AND APACHE KAFKA

The article is devoted to the study and development of the mechanism of interaction between Wolfram Mathematica programs and Apache Kafka queue to provide the ability to build event-driven applications based on it. The **subject** of the research is the practical principles of building a mechanism for interaction between Wolfram Mathematica and Apache Kafka through a proxy-server. The **purpose** of the article is to develop and substantiate practical recommendations regarding the formation of proxy-server and a mechanism for its work to publishing messages to the Apache Kafka queue and reading messages from it for programs of the mathematical processor Wolfram Mathematica, which will make it possible to build event-driven applications. The **tasks** are: to determine the mechanism of such interaction, prove the choice of tools for its implementation, create and test the obtained results. The research used the following **tools**: Apache Kafka, Kafkacat, servers Ubuntu 20 LTS, the method of developing the Wolfram Mathematica package. The **results** of the research: the mechanism of interaction between Wolfram Mathematica and Apache Kafka through a proxy-server was determined and the corresponding toolkit was created on its basis in the form of two Mathematica packages, which are built on using bash-scripts, Apache Kafka and third-party Kafkacat software. The first - for use on the end user's computer, the second – on a compute server with a remote Mathematica kernel. It is confirmed that the Mathematica processor is currently not suitable in its pure form for real-time data analysis. **Conclusions**. Practical recommendations have been developed and substantiated regarding the formation of the mechanism of interaction between the Wolfram Mathematica mathematical processor and the Apache Kafka queue manager through a proxy-server for the possibility of working in two directions with the queue: publishing messages and reading them. A toolkit for such interaction in the form of Mathematica packages has been created, their capabilities have been demonstrated. The economic benefit of using the described tools is shown. Future ways of its improvement are given.

Keywords: event-driven applications; queue manager; mathematical processor; saving resources and funds; cloud technologies; Kafka; Mathematica.

Introduction

The rapid development of distributed data processing and the transition in the construction of applications from monolithic architecture to service-oriented [1], and later micro service [2-4], which focuses on event processing (or messages), in recent years has led to improvement of real-time or near-real-time data transmission facilities, including such as queue managers or message brokers. One of the most common, refractory and powerful of them is Apache Kafka, which is demonstrated, for example, in [5-6]. It is able not only to dynamically adapt to the load from the queue customers, but also to scale vertically and horizontally.

Event-oriented software architecture is increasingly being implemented outside of commercial products. One such promising area is the processing of message flow by mathematical processors, which allows you to use their wide range of tools for data analysis: for example, developed and intensively developing interface for the interaction of MathWorks MATLAB with Kafka [7].

For the Wolfram Mathematica processor, which is also one of the world leaders in the field of intelligent data processing, this urgent task was solved in [8] - developed a mechanism for publishing messages and reading them from the Kafka queue. But the approaches outlined there are based on the local launch of applications that have the following shortcomings. Each of these approaches requires the installation and configuration of several complex components on the Kafka queue client computer. In addition, a direct connection to the Kafka cluster is associated with the following complications: each client needs to know the IP addresses of bootstrap servers, have authorization data on all servers in the cluster, and know all the necessary settings to work with them. This leads to such operational problems. First, when you change the IP addresses, ports, or other network settings of any of the Kafka cluster servers, and especially the bootstrap servers, you must make these changes on all clients. Second, when changing authentication settings, these changes must also be made on all clients. And, thirdly, when updating the version of Kafka, it may also be necessary to update it on all clients if it changes the data transfer protocol or other critical properties of the interface, because this may cause problems with interconnection, such as those described in [9 -10].

These shortcomings in the complex lead to the conclusion that such a system is unreliable and can be operated for a long time only when there is no need to make any changes to the Kafka cluster, and if necessary - can be used only with a small number of Kafka customers due to the large amount of work that needs to be performed synchronously in the event of changes in cluster settings.

This indicates that the task of building a reliable mechanism for the interaction of Wolfram Mathematica with Kafka remains unsolved. The most versatile way to increase this reliability is to use the approach of building a proxy server to centralize and hide all interaction settings and provide the client with a simple interface.

Therefore, the ***aim*** of this article is to develop and substantiate practical recommendations for the formation of the mechanism of bidirectional interaction of Wolfram Mathematica processor and queue manager Apache Kafka, which should be as simple as possible for queue clients, require a minimum of third-party applications, have a maximum deployment speed. cost of service. The task of the article is to identify the necessary elements of such a mechanism, to provide an analysis of the functional load for each of them, to justify the choice of tools for

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 2 (16) ISSN 2524-2296 (online)

their construction, to create and test the obtained result.

Building an intermediary server

All of the above issues and disadvantages of using Apache Kafka or Kafkacat locally as a queue client can be solved with a single approach - using a proxy server that hides all settings and gives clients a simple interface to interact with the Kafka queue.

The purpose of transferring the complexity of the client's interaction with the Kafka queue manager from the user's computer to the Linux-based proxy server is to solve the following tasks. First, centralizing the entire Kafka queue connection mechanism in one place and hiding all the details and settings. Second, guaranteeing the security of the network connection by using a network tunnel through the SSH channel from the end client to the proxy server. This minimizes the applications that need to be installed on the user's computer, reducing them to a single SSH client. The proxy server also solves another problem - incompatibility of the client and any of the Kafka cluster servers due to protocol versions or other differences - all settings are available at any time for change by the administrator, and the server can be updated and restarted at one time. thereby making changes for all customers to queue together.

Important non-technical advantages of this approach include cost savings - only one DevOps specialist is required to monitor such a server, whose function can also be performed by a Kafka cluster administrator.

The disadvantages of this approach include the lack of support for modern data compression algorithms (zStd, Snappy, and others) in the SSH tunnel. This can be partially corrected by the LZ77's built-in SSH compression algorithm. Tunnel security is achieved by using crypto-strong passwords (less recommended) or authorization keys. In contrast to the approach previously discussed in [8], such a server has access to all Kafka cluster servers, and therefore these servers can be easily protected from unauthorized network connections via a firewall.

In [8] it was proved that the use of third-party software Kafkacat [11] to interact with the queue gives at least an order of magnitude faster communication channel of the queue client with the cluster than the standard queue client Apache Kafka, and an order of magnitude less computer resources for work. Therefore, it is worth using it on the server, because each client launches a new copy of the application for publishing or consuming messages, and saving resources in this case is already very significant. But this application at the time of writing does not have the functionality to delete and create queues, so these rare operations use standard Apache Kafka scripts.

The diagram of the system built on this approach is shown below in fig. 1.

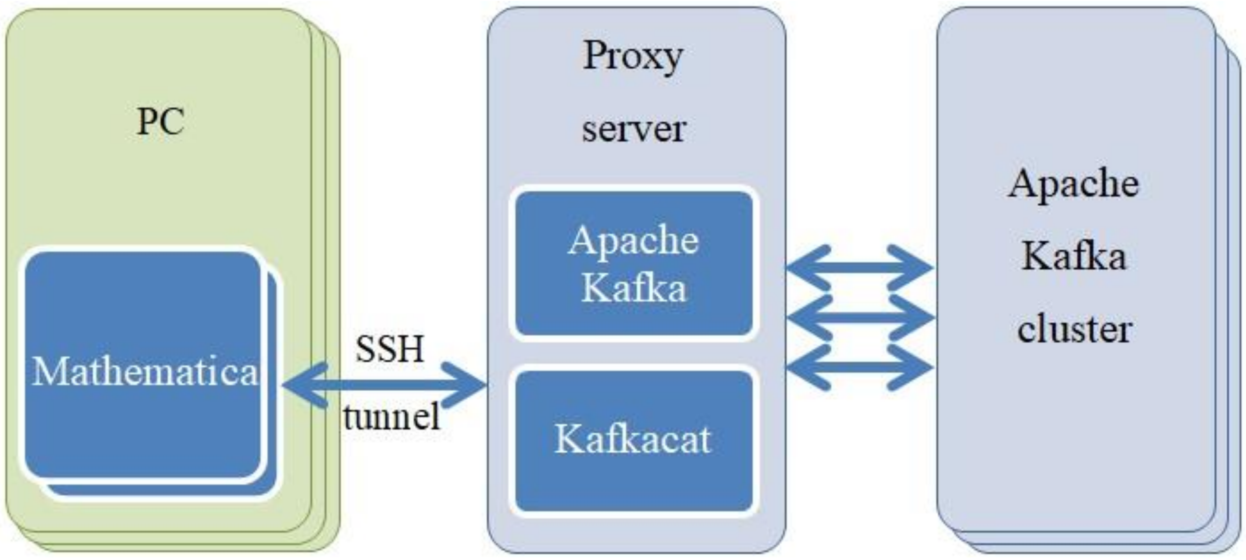


Fig. 1. Connecting Mathematica to the Kafka queue cluster through a proxy server

The well-known SSH package PuTTY is used to create an SSH tunnel [12]. Its significant features include cross-platform and portable distribution with a pre-configured connection profile, which hides from the end user the need to know the network settings of the SSH connection to the proxy server and greatly simplifies the use.

The basic idea of how Mathematica interacts with a proxy server is to open an SSH connection as a write channel and use it to execute remote commands on the proxy server. Such commands will be specially designed bash scripts that will be responsible for: reading and writing to the message queue - kafkaConsume.sh and kafkaPublish.sh, getting a list of existing queues - kafkaList.sh, deleting and creating queues - kafkaDelete.sh and kafkaCreate.sh.

Each script, except kafkaList.sh for obvious reasons, receives a queue name (variable "$ __ TOPIC" below the text), and kafkaPublish.sh also the key and text of the message published to the queue.

To simplify the configuration and ease of use of the developed bash-scripts on the server, environment variables $ KAFKA and $ BOOTSTRAP_SERVERS have been added, which describe the path to the "/bin/" subdirectory of the Kafka installation directory and the list of Kafka bootstrap servers, respectively.

The kafkaConsume.sh, kafkaPublish.sh, and kafkaList.sh scripts use the Kafkacat plugin. An important feature of this application is the default buffered output to the standard stream (stdout). This prevents reading from third-party applications like Mathematica. To disable this, use the "-u" argument.

The central design of the script for reading messages from the kafkaConsume.sh queue is:

```
kafkacat -C –J -u -q \
        -b "$BOOTSTRAP_SERVERS" \
        -t "${__TOPIC}"
```

The argument "-C" includes read mode, "-J" - receive a message in full JSON-format, "-u" - switches to unbuffered output mode, "-q" - disables the output of service information, the latter - understandable in content.

Posting in a queue differs only in the first line, which has the form:

```
echo "${__KEY}:${__VALUE}" | kafkacat -P -K: \
```

The argument "-P" includes the publishing mode, "-K:" - includes the division of the key and the value of the message by a colon.

The essence of the script kafkaList.sh, which is responsible for obtaining a list of queues, is reflected by the following construction:

```
kafkacat -L \
         -b $BOOTSTRAP_SERVERS \
          | egrep "^\s+topic" \
          | sed -r 's/^[^"]+"(.+)"[^"]+$/\1/'
```

This complexity is due to the fact that "kafkacat -L" displays detailed information in queues, to get only the names you need to separate them.

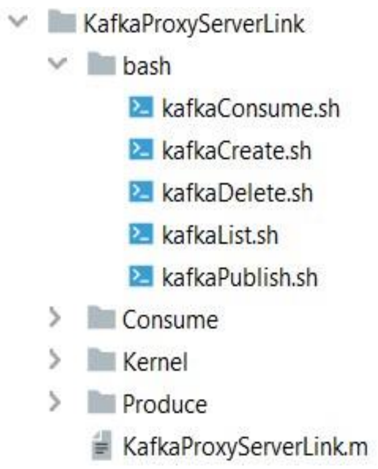
The kafkaDelete.sh and kafkaCreate.sh scripts use the same ideas and constructions as set out in [8] and are based on the standard Apache Kafka scripts due to the fact that the Kafkacat application does not have such functionality at the time of writing.

To increase the security of the described scripts, you need to create and configure a separate Linux user for Mathematica SSH clients on the proxy server.

Development of the Mathematica package

To build the described system, you also need to develop a Wolfram Mathematica package for the client side, which will use the developed bash-scripts on the side of the proxy server for operations with messages and queues.

By analogy with the Mathematica package [8], the KafkaProxyServerLink package was developed. Its structure of files and directories, shown in fig.2, is completely repeated except for the added directory "bash", which contains the above-developed bash-scripts that implement interaction with the queue manager Kafka.



```
KafkaProxyServerLink
    bash
        kafkaConsume.sh
        kafkaCreate.sh
        kafkaDelete.sh
        kafkaList.sh
        kafkaPublish.sh
    Consume
    Kernel
    Produce
    KafkaProxyServerLink.m
```

Fig. 2. Mathematica directory structure for SSH connection to proxy server

The "bash" subfolder must be uploaded to the proxy server and the files in it can be executed by a user connecting via SSH.

The interaction of the package with the queue differs from [8] only by using the Write function instead of Run to write to the stream (publish to the queue), which must be opened using the OpenWrite function as follows:

```
SSHStream = OpenWrite["!"<> SSHPuttyCommand, FormatType->OutputForm, PageWidth->Infinity];
```

Opening a write stream via OpenWrite is important because Mathematica can only have a thread for one operation: read or write - a combination of them is not possible. You should pay attention to the values of the For-matType and PageWidth options when opening the stream: the first sets the output of the string without shielding and other modifications, the second - disables automatic string transfers. The SSHPuttyCommand variable stores the command to run the plink component from the PuTTY program set with the parameters "-batch -v -ssh -t -C -x -l <username> -i <path / to / id_rsa.ppk> <ip: port>".

The open stream ID is stored in the global SSHStream variable for use, and is closed by the exit command from the SSH session:

```
Write[SSHStream, "exit"];
```

The function which has the following look is responsible for construction of correct term of a call of bash-scripts with parameters.:

```
KafkaGetBashScript[name_, args___:Null] := Module[{path},
        path = FileNameJoin[{SSHKafkaBashPath, name<>".sh"}, OperatingSystem->"Unix"];
        Return[ StringJoin[Riffle[{"bash", path, args}, " "]] ];
];
```

Which takes the name of the script and optionally its arguments through a comma, creates a full path to the bash script on the proxy server in Unix notation (the global variable SSHKaf-kaBashPath stores the path to the directory of bash scripts on the server) and combines it with arguments, separating them with spaces.

The use of the package in the Mathematica document is completely similar to the option with local start [8]. The JSON string returned by Kafkacat is processed in a similar way.

Using the remote Mathematica kernel

Mathematica processor is built on a client-server architecture with a platform-independent core [13]. Therefore, it is necessary to stop separately in case of use of a remote kernel [14] on the computing server built on the basis of Linux operating system which example can serve [15].

If you combine two servers into one: the Mathematica computing server and the proxy server for Kafka, then the connection to the queue core processor connection to the queue manager client will be local.

This allows you to combine the simplicity of the local connection package [8] to the Kafka queue manager client with the call from inside the bash scripts used in the package for the proxy server. This will provide the advantages of both packages without their disadvantages: no need to install additional software on the client other than PuTTY, the use of a fast application Kafkacat, direct execution of the core of bash-scripts instead of SSH. Also, obviously, you do not need a separate proxy server.

The system diagram for the computing server is shown in fig. 3 below.

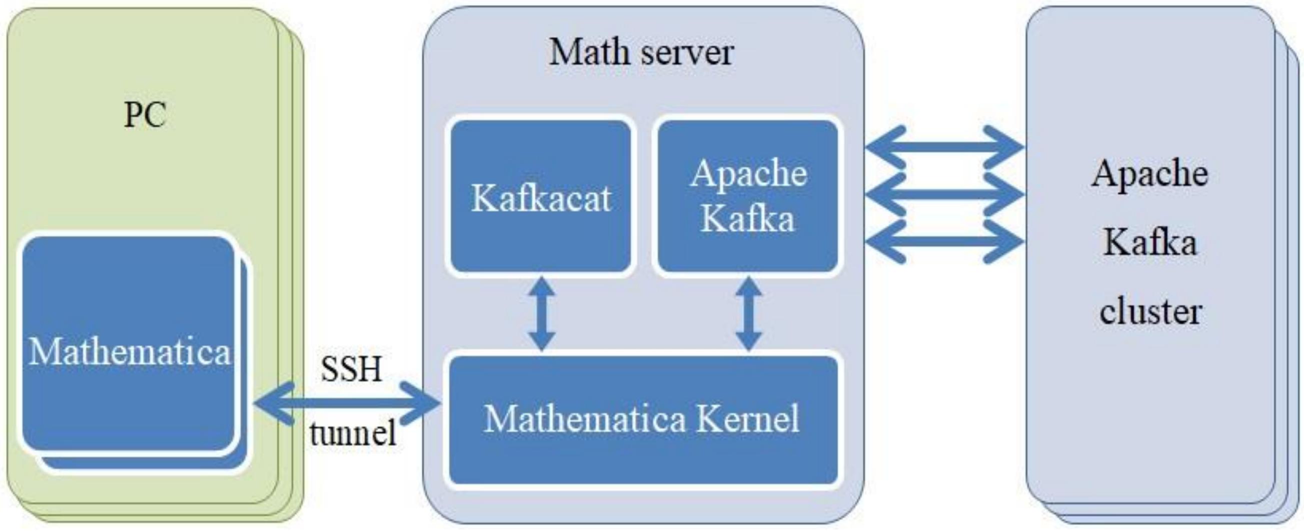
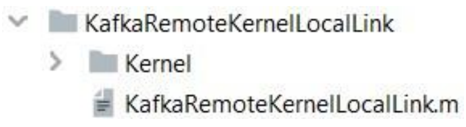


Fig. 3. Connecting the Mathematica client to the Kafka cluster via a computing server

The differences are, first, the description of all paths in the document using the FileNameJoin function, which uses a directory delimiter based on kernel settings, rather than Mathematica FrontEnd. Second, to use the Linux sh shell instead of calling PuTTY for local bash scripts to run on the server.

The package "KafkaRemoteKernelLocalLink" for Mathematica created for this case is a wrapper for the one developed above and has the following directory structure, which is shown in fig. 4.



KafkaRemoteKernelLocalLink
Kernel
KafkaRemoteKernelLocalLink.m

Fig. 4. The structure of the directories of the Mathematica package with the computing server as an intermediary server

It automatically downloads the above package as a dependency:

```
BeginPackage["KafkaRemoteKernelLocalLink`",
{"KafkaProxyServerLink`"}]
```

And replaces its function of opening a stream for reading or writing with one that uses the Linux application sh, as mentioned above.

Thus, it is achieved that from the Mathematica document both packages are identical in use.

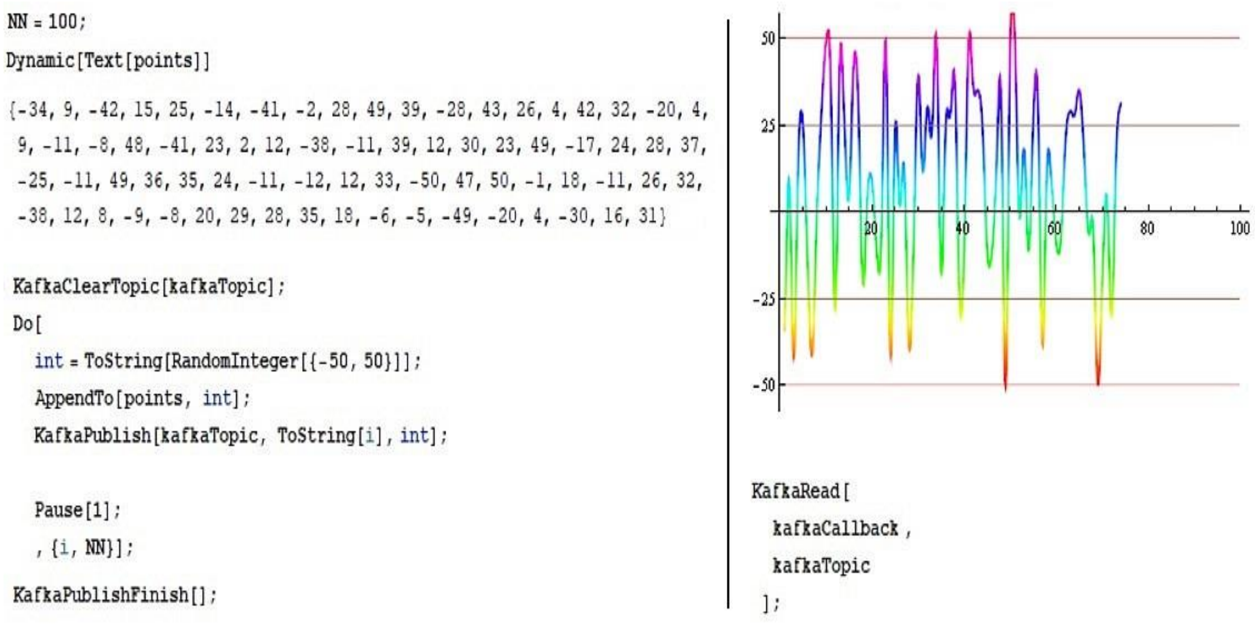
Demonstration of work

To demonstrate the work of the developed tools for building event-driven programs in Mathematica using an intermediary server, we will conduct an identical [8] experiment - generating an arbitrary number in the range [-50.50], repeated 100 times with a delay of one second.

To implement the experiment on a local computer, two Mathematica documents were created: a generator and a reader. The reader has a local core processor, which is allocated its own processor core for independent operation, and uses the developed package "KafkaProxyServerLink" to connect to the proxy server. The generator is connected to a computing server, which is built similarly to [15] and configured as an intermediary server in accordance with the requirements set out in the article, and uses the developed package "KafkaRemoteKernelLocalLink". DigitalOcean cloud technologies were chosen as the platform, where servers based on Ubuntu 20.04 LTS x64 OS were located. Choosing a cloud service as a basis allowed you to quickly deploy servers and change their configurations to meet the needs of the task.

The content of the experiment is as follows: an arbitrary number is generated on the publication side and added to the end of the points array, which has a dynamic update in the document, and is sent to the queue. At the same time, the client-reader receives a message with this number, adds it to its array, which automatically leads to the restructuring of the graph based on it, which also has a dynamic output. The graph is constructed by the ListPlot function for the whole definition area, with the option "InterpolationOrder -> 2", which makes the graph smoother, but at the boundary points leads to values out of the original range [-50.50], and serves as an additional load on message processing.

Since the client can only connect to a queue that already exists, the document with the publication of messages that creates the queue was launched first, followed by the reader's document. The result obtained at the same time for both documents is shown in fig. 5.



```
NN = 100;
Dynamic[Text[points]]

{-34, 9, -42, 15, 25, -14, -41, -2, 28, 49, 39, -28, 43, 26, 4, 42, 32, -20, 4,
 9, -11, -8, 48, -41, 23, 2, 12, -38, -11, 39, 12, 30, 23, 49, -17, 24, 28, 37,
 -25, -11, 49, 36, 35, 24, -11, -12, 12, 33, -50, 47, 50, -1, 18, -11, 26, 32,
 -38, 12, 8, -9, -8, 20, 29, 28, 35, 18, -6, -5, -49, -20, 4, -30, 16, 31}

KafkaClearTopic[kafkaTopic];
Do[
  int = ToString[RandomInteger[{-50, 50}]];
  AppendTo[points, int];
  KafkaPublish[kafkaTopic, ToString[i], int];

  Pause[1];
  , {i, NN}];
KafkaPublishFinish[];
```

```
KafkaRead[
  kafkaCallback,
  kafkaTopic
 ];
```

Fig. 5. Simultaneous publication (right) and reading (left) from the same queue

The figure above shows that the client-reader does not lag behind the client-publisher. The main delay, as in [8], is the updating of dynamic objects in the document.

The experiment was repeated 15 times to exclude measurement error, with one result – a delay on the side of Mathematica Frontend, i.e. compilation [16] is unlikely to solve this problem. This in turn confirms the conclusions made in [8] that the Mathematica matpack is not yet designed for real-time data analysis.

Prospects for further development of the system

Prospects for further development of the developed tools are the following improvements.

To completely eliminate the need for third-party client-side applications, it makes sense to consider using Mathematica's built-in features for SSH connections, starting with version 11.3: RemoteConnect and RemoteRunProcess [17], which could eventually replace the use of PuTTY software.

On the side of the intermediary server – it seems relevant to develop a Docker-container with built-in ready-made tools, which is described in the article, for the fastest deployment.

Conclusions

The paper develops and substantiates practical recommendations for the formation of the mechanism of interaction between the mathematical processor Wolfram Mathematica and the queue manager Apache Kafka using an intermediary server to work in two directions: publishing messages in the queue and consuming messages from it. Appropriate tools have been created.

The mechanism is developed in two variants: use of the intermediary server and combination of the last with the calculation server for a remote kernel. The second option allows you to have only one server for both tasks.

It is shown that the toolkit developed in the article allows you to easily build event-driven programs in Mathematica that require client-only PuTTY applications and data to connect to a single proxy server – that is, in the most simplified way, with maximum deployment speed on a new client and without maintenance needs by centralizing all interaction with the Kafka cluster on the proxy server. In addition, using a proxy server resolves the incompatibility issue between the client and any of the

Kafka cluster servers due to protocol versions or other differences due to its availability to change settings at any time.

The economic benefit of using the described tools is achieved by saving money on the maintenance of the described system: to monitor the proxy server requires only one DevOps specialist, whose function can be performed by the administrator of the Kafka cluster, on the client side such a specialist is not needed. Also, significant economic benefits are achieved by deploying an intermediary server in the cloud infrastructure, where the cost of ownership is much lower compared to real equipment and reduced to the time of its actual use, and the time and cost of its modification are the lowest.

References

1. Villamizar, M., Garcés, O., Ochoa, L. et al. (2017), "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures", *SOCA*, Vol. 11, P. 233–247. DOI: https://doi.org/10.1007/s11761-017-0208-y
2. Gutiérrez–Fernández, A. M., Resinas, M., Ruiz–Cortés, A. (2017), "Redefining a Process Engine as a Microservice Platform", *In: Dumas M., Fantinato M. (eds) Business Process Management Workshops. BPM 2016. Lecture Notes in Business Information Processing*, Vol. 281, Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-58457-7_19
3. Brogi, A., Canciani, A., Neri D., Rinaldi, L., Soldani, J. (2018), "Towards a Reference Dataset of Microservice-Based Applications", *In: Cerone A., Roveri M. (eds) Software Engineering and Formal Methods. SEFM 2017. Lecture Notes in Computer Science*, Vol. 10729, Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-74781-1_16
4. Monteiro, D., Gadelha, R., Maia, P. H. M., Rocha, L. S., Mendonça, N. C. (2018), "Beethoven: An Event-Driven Lightweight Platform for Microservice Orchestration", *In: Cuesta C., Garlan D., Pérez J. (eds) Software Architecture. ECSA 2018. Lecture Notes in Computer Science*, Vol. 11048, Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-00761-4_13
5. GitHub (2020), "Ultimate Message Broker Comparison", available at: https://ultimate-comparisons.github.io/ultimate-message-broker-comparison/ (last accessed 10 December 2020).
6. G2 (2020), "Best Message Queue (MQ) Software in 2021: Compare Reviews on 40+ MQs", available at: https://www.g2.com/categories/message-queue-mq (last accessed 10 December 2020).
7. GitHub (2020), "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 December 2020).
8. 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
9. Stack Overflow (2020), "Kafka bootstrap-servers vs zookeeper in kafka-console-consumer", available at: https://stackoverflow.com/questions/41774446/kafka-bootstrap-servers-vs-zookeeper-in-kafka-console-consumer (last accessed 10 December 2020).
10. Stack Overflow (2020), "Apache Kafka - bootstrap-server vs zookeeper params in consumer console", available at: https://stackoverflow.com/questions/53954877/bootstrap-server-vs-zookeeper-params-in-consumer-console (last accessed 10 December 2020).
11. GitHub (2020), "Edenhill/kafkacat: Generic command line non-JVM Apache Kafka producer and consumer", available at: https://github.com/edenhill/kafkacat (last accessed 10 December 2020).
12. PuTTY (2020), "Download PuTTY - a free SSH and telnet client for Windows", available at: https://www.putty.org/ (last accessed 10 December 2020).
13. Wolfram (2020), "Wolfram Mathematica: Modern technical calculations", available at: https://www.wolfram.com/mathematica/ (last accessed 12 November 2020).
14. Wolfram Library Archive (2020), "Remote Kernel Strategies", available at: https://library.wolfram.com/infocenter/Conferences/7250/ (last accessed 10 December 2020).
15. 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
16. Zolotariov, D. A. (2020), *Automation and optimization of scientific and engineering calculations in Wolfram Mathematica*, Kharkiv : FOP Panov A.M., ISBN: 978-617-7859-36-8 [In Ukrainian]
17. Wolfram Language Documentation (2020), "RemoteConnect", available at: https://reference.wolfram.com/language/ref/RemoteConnect.html (last accessed 10 December 2020).

Received 13.04.2021

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. 2 (16)*

ПЛАТФОРМА ДЛЯ ПОБУДОВИ КЕРОВАНИХ ПОДІЯМИ ДОДАТКІВ НА БАЗІ WOLFRAM MATHEMATICA ТА APACHE KAFKA

Стаття присвячена дослідженню та розробці механізму взаємодії програм Wolfram Mathematica із менеджером черги Apache Kafka для надання можливості побудови на його основі керованих подіями додатків. **Предметом** дослідження є практичні засади побудови механізму взаємодії Wolfram Mathematica із Apache Kafka через сервер-посередник. **Метою** статті є розробка та обґрунтування практичних рекомендацій щодо формування сервера-посередника та механізму його роботи для публікації повідомлень у чергу Apache Kafka та споживання повідомлень із неї програмами математичного процесору Wolfram Mathematica, що дасть можливість побудови керованих подіями додатків на його основі. **Завдання** роботи: визначити механізм такої взаємодії, обґрунтувати вибір інструментів для його реалізації, створити та протестувати отриманий результат. У ході дослідження використано **засоби**: інформаційні технології Apache Kafka, Kafkacat, сервера на базі Ubuntu 20 LTS, спосіб побудови пакету Wolfram Mathematica. **Результати** дослідження: визначений механізм взаємодії Wolfram Mathematica із Apache Kafka через сервер-посередник та створений відповідний інструментарій на його основі у вигляді двох пакетів Mathematica, що побудовані на використанні bash-скриптів, Apache Kafka та стороннього програмного забезпечення Kafkacat. Перший – для використання на комп’ютері кінцевого клієнта, другий – на сервері обчислень із віддаленим ядром Mathematica. Протестована їх робота. Підтверджено, що на даний момент математичний процесор Mathematica не підходить у чистому вигляді для аналізу даних у реальному часі. **Висновки.** Розроблені та обґрунтовані практичні рекомендації щодо формування механізму взаємодії математичного процесору Wolfram Mathematica та менеджеру черги Apache Kafka через сервер-посередник для можливості роботи у двох напрямках із чергою: публікації повідомлень та їх читання. Створений інструментарій для такої взаємодії у вигляді двох пакетів Mathematica, продемонстровані їх можливості. Показана економічна вигода від використання описаного інструментарію. Наведені майбутні шляхи його вдосконалення.

Ключові слова: керовані подіями додатки; менеджери черги; математичний процесор; хмарні технології; економія ресурсів та коштів; Kafka; Mathematica.

ПЛАТФОРМА ДЛЯ ПОСТРОЕНИЯ УПРАВЛЯЕМЫХ СОБЫТИЯМИ ПРИЛОЖЕНИЙ НА БАЗЕ WOLFRAM MATHEMATICA И APACHE KAFKA

Статья посвящена исследованию и разработке механизма взаимодействия программ Wolfram Mathematica с менеджером очереди Apache Kafka для предоставления возможности построения на его основе управляемых событиями приложений. **Предметом** исследования являются практические принципы построения механизма взаимодействия Wolfram Mathematica с Apache Kafka через сервер-посредник. **Целью** статьи является разработка и обоснования практических рекомендаций относительно формирования сервера-посредника и механизма его работы для публикации сообщений в очередь Apache Kafka и чтения сообщений из нее для программ математического процессора Wolfram Mathematica, что даст возможность построения управляемых событиями приложений на его основе. **Задача** работы: определить механизм такого взаимодействия, обосновать выбор инструментов для его реализации, создать и протестировать полученный результат. В ходе исследования использованы **средства**: информационные технологии Apache Kafka, Kafkacat, сервера на базе Ubuntu 20 LTS, способ разработки пакета Wolfram Mathematica. **Результат** исследования: определен механизм взаимодействия Wolfram Mathematica с Apache Kafka через сервер-посредник и создан соответствующий инструментарий на его основе в виде двух пакетов Mathematica, которые построены на использовании bash-скриптов, Apache Kafka и стороннего программного обеспечения Kafkacat. Первый – для использования на компьютере конечного пользователя, второй – на сервере вычислений с удаленным ядром Mathematica. Подтверждено, что на данный момент математический процессор Mathematica не подходит в чистом виде для анализа данных в реальном времени. **Выводы**. Разработаны и обоснованы практические рекомендации относительно формирования механизма взаимодействия математического процессора Wolfram Mathematica и менеджера очереди Apache Kafka через сервер-посредник для возможности работы в двух направлениях с очередью: публикации сообщений и их чтения. Создан инструментарий для такого взаимодействия в виде пакетов Mathematica, продемонстрированы их возможности. Показана экономическая выгода от использования описанного инструментария. Приведены будущие пути его усовершенствования.

Ключевые слова: управляемые событиями приложения; менеджеры очереди; математический процессор; облачные технологии; экономия ресурсов и средств; Kafka; Mathematica.

Бібліографічні описи / Bibliographic descriptions

Золотарьов Д. О. Платформа для побудови керованих подіями додатків на базі Wolfram Mathematica та Apache Kafka. *Сучасний стан наукових досліджень та технологій в промисловості*. 2021. № 2 (16). С. 12–18. DOI: https://doi.org/10.30837/ITSSI.2021.16.012

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