

Approach in the development of lightweight microservice architecture for small data center monitoring system

Milan Malić¹, Dalibor Dobrilović², Dušan Malić³, Željko Stojanov²

¹ Panonit, Novi Sad, Serbia

² University of Novi Sad / Technical Faculty "Mihajlo Pupin", Zrenjanin, Serbia

³ Technical College of Applied Sciences in Zrenjanin, Zrenjanin, Serbia

milanmalic@outlook.com, dalibor.dobrilovic@uns.ac.rs, dmalic@sbb.rs, zeljko.stojanov@uns.ac.

Abstract— In the past decade there is a significant trend of implementing IoT technologies and standards in different industries. This trend brings cost reductions to the companies and other benefits as well. One of the main benefits is real-time and uniform data collection. The data are transferred using diverse communication protocols, from the sensor nodes to the centralized application. So far, current approaches in developing applications are not proved itself to be efficient enough in scenarios when a significant amount of data needs to be stored and analyzed. The focus of this paper is on development of software architecture suitable for usage in Internet of Things (IoT) systems where the larger amount of data can be processed in real-time. The software architecture is developed in order to support the sensor network for monitoring the small data center and it is based on microservices. Besides the system and its architecture, this paper presents the method of analysis of system performances in real-time environment. The proposal for lightweight microservice architecture, presented in this paper, is developed with .NET Core and RabbitMQ, with the utilization of MongoDB and SQLite databases systems for storing data collected with IoT devices. In this paper, the system evaluation and research results in different stress scenarios are also presented. Because of its complexity, only the most significant segments of architecture will be presented in this paper. The proposed solution showed that proposed lightweight architecture based on microservices could deal with the larger amount of sensor data in the case of using MongoDB. On the other hand, the usage of SQLite database is not recommended due to the lower performances and test results.

Keywords- microservices; IoT; MongoDB; SQL; .NET Core; data center, RabbitMQ

I. INTRODUCTION

In the past few years, numerous studies have shown the growing importance of using IoT devices in industries, as well as the benefits and savings they bring. Direct and indirect benefits of such solutions have brought companies to significant reduction of financial expenses. This approach additionally brings a substantial inflow of new data collected with the help of IoT devices, which have disrupted the functionalities of the applications used so far. The existing and widely accepted architecture of monolithic applications is unable to respond to the rigorous requirements needed for applications that should function in real-time, and therefore it is necessary to respond to this problem in changing approach in application development.

Forbes, in cooperation with Hitachi, conducted a survey [1] related with usage of IoT in industry. The results of the survey indicate that over 64% of participants consider IoT today as a significant segment of their company and their operations,

This paper is a revised and expanded version of the paper presented at the XVIII International Symposium INFOTEH-JAHORINA 2019 [4]

while 36% remained neutral. None of the participants considered the IoT to be insignificant. When asked about the future, 91% of participants answered that IoT was considered as a very important segment of their business. Only 9% of participants remained neutral, while none of the participants considered it insignificant. Research [2], conducted by the Capgemini Digital Transformation Institute, points to the growth in the use of IoT devices in many industries around the world. According to the study of the Statistic [3] by 2025, over 75 billion of connected IoT devices are expected, three times more than today.

Previously mentioned statistics point to the importance of IoT devices in the operations of modern companies, both today and in the near future. In order to respond, in a more efficient way to the massive inflow of data that IoT devices generate, it is necessary to develop applications in order to enable easier data processing and pre-process data for future real-time analysis. In this paper is proposed a lightweight microservice architecture system developed in C# programming language, whose main functionality is small data center monitoring. In order to provide cross-platform support, the .NET Core 2.2 framework is used. For the messaging system, within the

microservice system, RabbitMQ is used. The data storage system consists of NoSQL MongoDB to store all incoming messages, while the SQLite relational database is used to store processed data.

The contribution of this paper is the proposal of the lightweight system architecture for monitoring small data centers as well as the methodology for its testing. The two stress tests are performed in order to evaluate architecture components, especially DBs used for data storage. This work is extension of the research presented in [4].

This paper is structured as follows. After the introduction, in Section II we presented the previous researches both in the field of microservice oriented architectures and data warehousing. In Section III, the example of microservice based application in IoT and WSN systems is given. In Section IV, the analyses of specific DBMS for potential usage in WSN and IoT environments are presented. In section V, the architecture of the data center, test and development platform as well as the architecture of the proposed lightweight system based on microservices is presented. The results of the system evaluation, as well as a comparative analysis of two different stress tests, are presented in Section VI. Finally, Section VII gives conclusion as well as future research directions.

II. RELATED RESEARCH

The savings made by usage of IoT devices in everyday life are numerous. The authors [5] present a platform for the monitoring and management of a wide variety of energy-related agents in buildings, emphasizing the importance of proper management and analysis of the collected data. On the other hand, research [6] presents potential opportunities available in the IoT embedded sustainable supply chain for Industry 4.0. As it can be concluded, IoT devices are becoming more and more present in various aspects of everyday life. In order to respond effectively to a large amount of generated data, transferred from the IoT devices, it is necessary to consider other aspects of the application architecture development. The authors in [7] present the challenges which modern applications are facing during their life cycles, and what new paradigms are adopted by the companies during this process. Also, they point out what rigorous standards need to be fulfilled by modern applications. As a potential solution, they propose application based on microservice architecture. Similar, the authors [8] present the advantages that are characteristic to microservice architecture such as agility, insulation, resistance, elasticity, robustness, tolerance for errors, scalability, etc.

The central segment of the micro service-based architecture is communication between microservices itself [9, 10, 11]. Message brokers are used to handle the transmission of messages between system components and they represent the backbone of the system. Today, there are many message brokers both in open-source and in closed-source versions, each of which has its advantages and disadvantages. For this research RabbitMQ developed by Pivotal [12] was used. Through research [13], RabbitMQ managed to receive and deliver over a million messages per second. These results place him to the level of high-performance brokers.

Today, more and more companies have demand, as one of the main requirements for IT departments, for services able to work between different and multiple platforms. In order to

achieve this in June 2016, Microsoft developed a new framework, .NET Core, which is open source and can be used on all platforms with identical source code. The advantages that this framework offers are numerous. In the research [14] it was indicated that the Bing Internet search engine with .NET Core version 2.1. framework achieved 31% better performances. According to IEEE research [15], C# is positioned in the five most popular programming languages. Based on this, it was decided that the development of the microservice system will be made with the usage of C# .NET Core with last stable version 2.2.

A large amount of data requires an efficient way of storing data. The research presented in [16] indicates a potential solution for using MongoDB for Big Data storage and real-time analytics. NoSQL MongoDB uses BSON (Binary JSON) [17, 18] to store zero or more key/value pairs as a single entity. This entity is also called a document. In this way, better performance is achieved because there is no conversion of data, from a human-readable such as JSON to a binary. This indicates that MongoDB is a good solution for microservice architecture. In [19], the analysis of the convenience between NoSQL and NewSQL for use in Big Data is performed.

III. MICROSERVICE BASED APPLICATIONS IN IOT AND WSN SYSTEMS

Since this paper proposes the lightweight microservice based architecture for data center monitoring system, in this section, various examples of similar applications being used in the IoT, WSN and Smart city systems will be described. These examples will help to understand different experiences and approaches in implementation of sensor based IoT monitoring systems as well as specific benefits of using microservices architectures in these environments.

In [20] the authors adopt a microservices based architecture in order to establish a real-time environmental sensors system that have highly scalable applications deployed in cloud environment. This architecture consists of a set of loosely coupled and independently deployable services. This system is designed to support various cooperating services such as real-time hazmat monitoring, risk-knowledge, hazmat routing, and transportation documents, in order to provide more effective and reliable transport problem solutions aimed at monitoring dangerous goods transportation in Mohammedia City.

According to authors [21] this paper is the first that deals with the integration of WSNs in the IoT using microservice approach through neural network, together with the cloud computing approach. The proposed model allows the design and the development of an extensible, distributed, and adaptable middleware system for lightweight WSN device integration. The focus on this work is on new middleware design based on microservice architecture. It is called MsM, and allows a complete WSNs integration with the internet and deals with them as services. To achieve this goal the microservice architecture is used.

The benefits of microservice architecture and Docker containers are explored in [22] with the optimization of application deployment in cloud data centers using microservice and Docker containers. Their goal is to minimize the application deployment costs as well as the operation costs while preserving service delay requirements for applications.

In the [23] authors present the implementation of a distributed middleware developed within the frame of MAYA European project. This middleware is tailored to enable scalable interoperability between enterprise applications and CPSs (Cyber-Physical-Systems). The proposed platform was made in effort to be the first solution based on both Microservices and Big Data paradigms to empower shop-floor CPSs and Digital Factory.

The definition of a simple model of the monitoring infrastructure that provides an interface between the user and the cloud management system is presented in [24]. The interface follows the guidelines of Open Cloud Computing Interface (OCCI), the cloud interface standard proposed by the Open Grid Forum. The monitoring functionalities that are defined through the interface are implemented as microservices embedded in containers. The internals of each microservice reflects the distinction between core functionalities which are bound to the standard, and custom plugin modules. In the paper, it is also described the engine that automatically deploys a system of microservices in the monitoring infrastructure.

The software architecture design that can be used as a template for the implementation of Smart City applications is presented in [25]. The design is based on the microservices architectural style, which provides properties that help in making Smart City applications scalable and flexible. The hybrid approach to securing sensitive data in the cloud is also described. The contribution of the paper is generic architecture design based on the microservice architectural style that can be used as a template for secure cloud-based Smart City applications. The hybrid model consists of a public and a private cloud environment.

IV. DBMS IN WSN AND IOT ENVIRONMENTS

One of the important components in proposed system architecture is DBMS (Database Management System) used for real-time data storage and data analyses. The issue of choosing suitable DBMS in IoT and WSN systems is very important.

In order to identify the best NoSQL DBMS to store tele-monitoring big data, in the paper [26] were consider four of the major current commercial solutions: MongoDB (adopting a document-based data model), Cassandra (adopting a column-based data model), Hbase (adopting a column-based data model), and Neo4J (adopting a graph-based data model). In the following, we provide a brief overview of such NoSQL DBMS solutions. The authors concluded that for processing batch tele-monitoring big data in background, apart MongoDB, also Hbase, Cassandra (both adopting a NoSQL column-based approach) represent good alternatives.

The paper [27] compares three databases for the purpose of sensor data storage. The compared databases are one open source SQL database (PostgreSQL) and two open source NoSQL databases (Cassandra and MongoDB). A comparison is also made between running these databases on a physical server and running them on a virtual machine.

In order to choose particular DBMS we used DB-Engines Ranking [28] scores. This source ranks database management systems according to their popularity. The ranking is updated monthly and the list of top 15 systems is presented in Table I.

TABLE I. DB-ENGINES RANKING SCORES [28]

Rank	DBMS	Type	Score
1.	Oracle	Relational, Multi-model	1336.07
2.	MySQL	Relational, Multi-model	1266.28
3.	Microsoft SQL Server	Relational, Multi-model	1081.91
4.	PostgreSQL	Relational, Multi-model	491.07
5.	MongoDB	Document, Multi-model	413.18
6.	IBM Db2	Relational, Multi-model	172.60
7.	Elasticsearch	Search engine, Multi-model	148.40
8.	Redis	Key-value, Multi-model	145.24
9.	Microsoft Access	Relational	130.07
10.	Cassandra	Wide column	123.23
11.	SQLite	Relational	121.03
12.	Splunk	Search engine	89.06
13.	MariaDB	Relational, Multi-model	85.57
14.	Hive	Relational	84.22
15.	Teradata	Relational, Multi-model	80.35

DB-Engines site represents the initiative that collects and presents information on DBMS. Both relational DBMS and growing NoSQL systems are covered with this ranking. The DB-Engines Ranking is a list of DBMSs ranked by their current popularity. Popularity of each database is reflected with its score.

Method of calculating the scores of the DB-Engines Ranking covers the measure of the system popularity calculated using the following parameters: (1) number of mentions of the system on websites (number of results in search engines queries such as Google, Bing and Yandex obtained with searching for <system name> together with the term database, e.g. "Oracle" and "database"); (2) general interest in the system based on data about frequency of searches in Google Trends; (3) frequency of technical discussions about the system based on the number of related questions and the number of interested users on the well-known IT-related Q&A sites such as Stack Overflow and DBA Stack Exchange; (4) number of job offers, in which the system is mentioned based on the number of offers on the leading job search engines such as Indeed and Simply Hired; (5) number of profiles in professional networks, in which the system is mentioned such as data from LinkedIn and Upwork; (6) relevance in social networks based on the count of the number of Twitter tweets, in which the system is mentioned.

The final score is calculated by standardizing and averaging of the individual parameters. The final score points to the following conclusion: when system A has twice as large a value in the DB-Engines Ranking as system B, then it is twice as popular.

The Fig. 1 shows the growth of popularity of MongoDB in last 6 years. MongoDB is the most popular non-relational DBMS, and ranked as 5th DBMS in total with total score 413.18. Fig. 2 shows the growth of popularity of SQLite DBMS. This database is ranked as 11th in total ranking and with significantly lower score (121.03). This DBMS is most popular database that requires no server and that is totally platform independent.

Because of its features, and high scores (high popularity) these two databases are chosen for implementation in proposed software architecture.

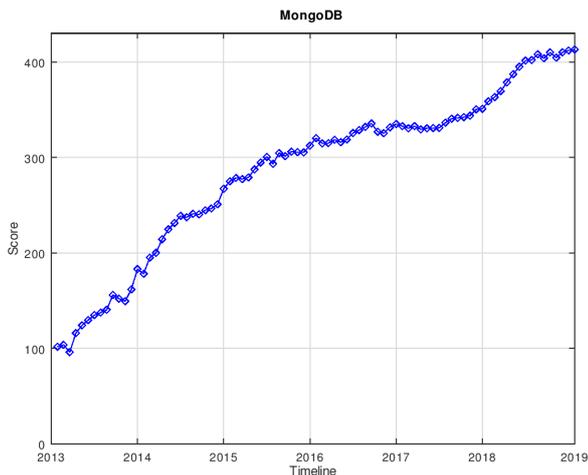


Figure 1. MongoDB score in six year period [28]

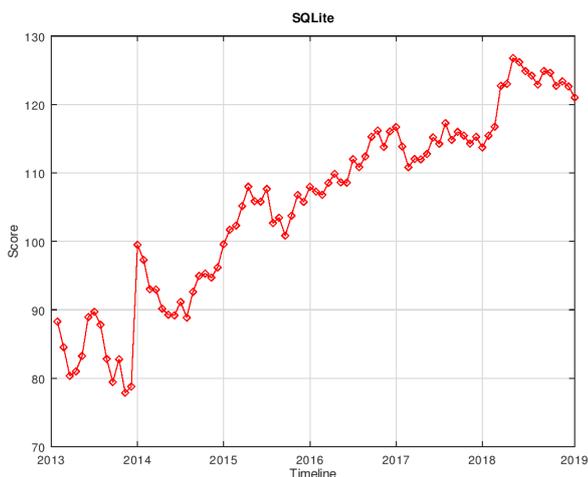


Figure 2. SQLite score in six year period [28]

V. SYSTEM ARCHITECTURE

The research covers three main elements: Data Center (A), Test Laboratory (B) and micro service-based architecture system (C). The Data Center was implemented as part of the MIS ETC 1379 project "Cross-border access infrastructure to high-level education through webcasts (EduWebCast)" within cross-border cooperation program Romania-Serbia and funded by the European Union under the instrument for pre-accession assistance (IPA) and co-financed by the participating countries.

The test environment is a development laboratory composed of a large number of computers for professional use. The primary purpose of such environment is the development of applications and their evaluation and testing.

Finally, the third element is system for monitoring and data processing and it is based on microservice architecture. The system consists of a number of components (microservices), message brokers, NoSQL, and SQL data storage systems.

A. Data Center Architecture

The function and detailed description of the system is given in [29, 30, 31]. The central parts of the system are two stacked

Brocade ICX 6610 switches, and they are shown in Fig. 3 as (1).

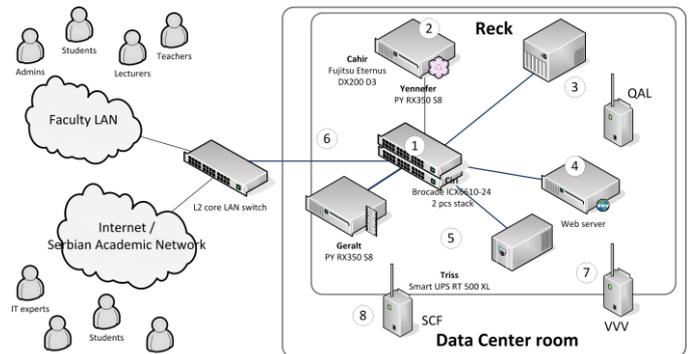


Figure 3. Data center architecture

Next element in the system is network storage device (Fujitsu Eternus DX200 D3) with 12 SAS disks with a capacity of 600GB, giving a total of 7.2 TB storage capacity (2). Ten disks are set in the working mode, and two are backup disks. Disk RAID is set to High Performance (RAID1 + 0). The total disk capacity of 2.67 TB is organized as follows: 900 GB uses server (3), and 1.79 TB uses a server (4).

Since the presented application is planned to be used for data center monitoring it is important to describe what environmental parameters can be acquired with sensors currently deployed in the system. These two devices give the following temperature sensor readings: cooler, MAC, CPU, sensor A, sensor B, sensor C, sensor D, and a stacked card. The MAC temperature is used as the reference temperature of the device. Two stacked switches are placed in the central part of the cabinet and the lower Brocade switch has a higher measured temperature, and this temperature was taken as the reference temperature of the entire cabinet and as one of the most important parameter for monitoring.

The primary server is the Fujitsu PY RX350 S8 with 64GB of RAM, two Xeon E5-2697 v2 12C/24T 2.7GHz CPUs, and two Tesla K20X GPUs (4). The secondary server is Fujitsu RX300 S8 has with two Xeon E5-2697 v2 with 10 cores (3). Temperature data of the variety of components of these two servers are collected in current system. In addition to temperature, there is the ability to monitor CPU and DRAM memory energy consumption, and the hard disk and memory utilization. The system is powered with a 5000VA UPS system (5). This unit has sensors for monitoring the temperature, output voltage, frequency, power, and current consumption.

The small data center is additionally monitored with sensor devices integrated in sensor stations built with Arduino/Genuino UNO development boards and open-source hardware. Three stations are used. SCF station uses the DHT11 and BH1750FVI sensors to monitor temperature, humidity, and light levels. The QAL station uses the Grove temperature sensor and the Bosch BMP85 sensor to monitor temperature and barometric pressure. VVV station uses BMP85 and DS18B20 sensors to monitor temperature and barometric pressure. All three sensor stations are able to acquire three sensor readings. Communication between the Arduino/Genuino devices is based on the wireless ZigBee technology.

Three sensor stations are placed at different locations in the data center in order to collect ambient temperature. The QAL

station (6) is located inside, the VVV station (7) is located at the top, and the third station SCF (8) is located at a distance of 2m from the cabinet.

Real data collected via SCF, QAL and VVV stations, during the system operation, will be used as dummy data in the development of the new lightweight system based on the microservice architecture for monitoring and analysis. This dummy data will be used for evaluation and testing purposes. Messages generated by stations and transferred with ZigBee technology. The data are in the following format: stationID, sensor1 readings, sensor2 readings, sensor2 readings. In front of these information, timestamp is inserted to help in analyzing of how much time passes from the moment of generating a message to its final storage in database.

B. Test & Development Hardware Architecture

The following platform is developed for testing purposes. Its primary function is to be used for analyses of time needed to process and to store data with proposed system architecture and usage of microservices. Also, the big advantage of this platform is in its mobility. All equipment is easily movable and transferable. With such devices in a short period many types of tests can be performed.

This platform contains several computers and network devices. The main computer that runs all microservices has following configuration:

- Intel-based i7-8700k 6 core / 12 threads CPU,
- Asus Z-370 motherboard
- 16 Gb DDR4,
- M.2 NVMe Samsung 960 256Gb,
- 1 Gbit/s network interface
- Nvidia Asus 1070 8 Gb
- Windows 10 x64 OS

Laptop Lenovo T440 is used for running RabbitMQ. The laptop has following configuration:

- Intel i5-4300U vPRO with 2 cores / 4 threads
- 8 Gb DDR3
- Intel 256 SSD
- 1 Gbit/s network interface
- Ubuntu Server 18.04 x64 LTE

As network device, MikroTik router RB2011UiAS-2HnD-IN is used. This device has five 1Gbit ports and five 100 Mbit ports. It also has a Wi-Fi connection. The latest version of RouterOS is installed on the router with a level 5 license. Gigabit network is used for communication between the central computer and Lenovo T440.

Simulation software is used to simulate QAL and VVV stations and to transmit data. This software is deployed on 2 laptops and connected via Wi-Fi 54Mbps network. In addition, the simulator of SCF station is deployed on the central computer. The configuration of these two laptops is irrelevant for this research.

C. Microservice Architecture

As mentioned in the introduction section, all microservices are written in the C# programming language supported by .NET Core 2.2. framework. The system consists of several components as it is shown in Fig. 4.

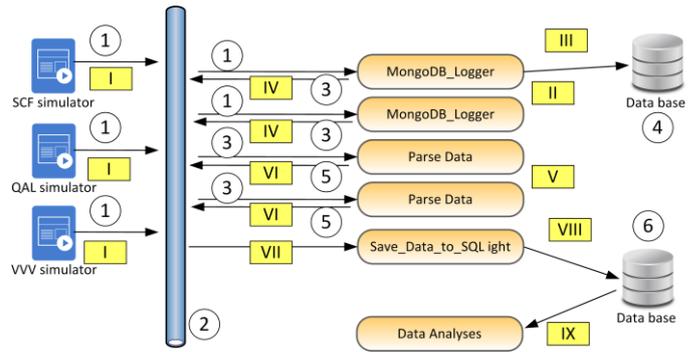


Figure 4. Schematics of data flow in microservice system with following elements: (1) task_queue, (2) RabbitMQ, (3) data_to_parse, (4) MongoDB database, (5) data_to_save, (6) SQLite database

As it can be seen from Fig. 4 station simulators are directly connected (I) to RabbitMQ (2) and send messages to it. In this model, all messages are sent to the same queue task_queue (1).

After publishing the message to the queue, an event is generated on the microservice side, and the message is processed further by one of the available MongoDB_Logger micro-services (II). For testing purposes in this research, the Round Robin system is implemented to download messages between two instances of micro-services. The downloaded message is forwarded to the MessagesDB collection located on MongoDB and placed in the appropriate collection (III). The timestamp is added when the message is taken and sent to the database. In addition to storing a message, the microservice sends a new message (IV) to a new queue data_to_parse (3).

Publishing message to the queue data_to_parse generates an event on the ParseData microservice (V). After that, a message is taken, and the same is processed in the appropriate data model. As in the first case, a timestamp processing is added. After the processing is done, a new message is published (VI) on the queue data_to_save (5). Round Robin system is used for taking messages from a queue by two instances of ParseData microservice.

Finally, after setting the message to the queue data_to_save, the event (VII) is generated on the microservice SaveDataToSQLite. The only task of this microservice is to forward the message with the processed data to store in the SQLite database. The Entity Framework Core was used to communicate with the database (VIII). Table in SQLite has trigger places on Insert, so timestamp is added after insert. The syntax for the trigger is:

```
CREATE TRIGGER log_insert_new_record_after_insert
AFTER INSERT ON Messages BEGIN update Messages SET
InsertDateTime = strftime("%Y-%m-%d %H:%M:%f", "now",
'localtime') WHERE ID = NEW.ID; END
```

After the data is stored in the SQLite database, the data application analytics can use the data for further processing. Communication with the SQLite database, as in the previous case, was achieved through Entity Frameworks Core (IX).

It is important to note that the system is not designed to allow simultaneous connections of several microservices to the database. In the initial phase of the system development, no competing queries and database modifications can be made at the same time. The reason for this approach is that only one record in the current system configuration can be downloaded

from the message broker and only one record can be inserted at a given moment. On the other hand, there may be more concurrent readings from the database and this does not affect the system performances.

VI. RESEARCH RESULTS

Two stress tests are performed in this research in order to evaluate proposed lightweight architecture. During each test, each of the station simulators sent 10,000 messages so that in the end, each database should have 30,000 records.

Queues

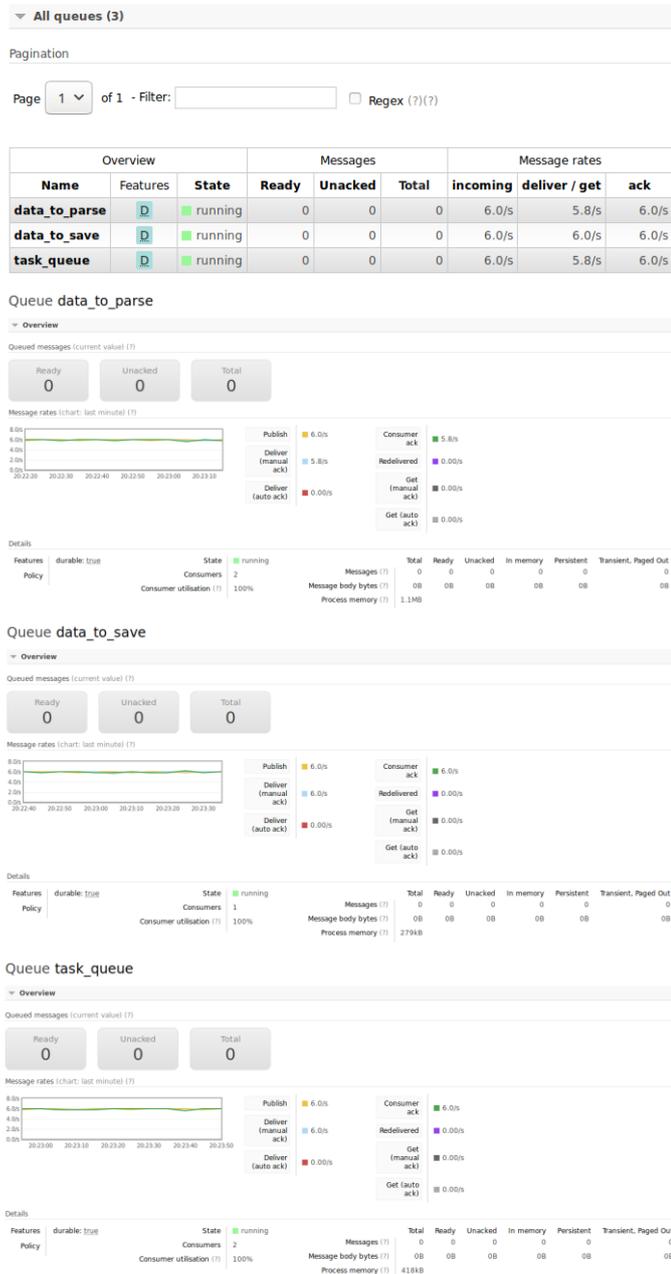


Figure 5. Statistical data of RabbitMQ performance during the first test

During the first test simulators generated a message every half of a second and during the second test every 0.1 seconds. In this way, simulation of the stressful environment will be achieved and the arrival of multiple messages on the system

will be simulated with the goal to collect performance results in these two modes of operation. At the same time, the identification of the system bottlenecks will be possible in the early stages of development in order to avoid them in the future.

During the execution of each test, RabbitMQ behavior was analyzed using web client monitoring tool that comes with its official installation. In this case, the number of messages on each queue as well as the amount of incoming and sent messages per second was monitored.

The first statistic data show the average time needed between sending the message by the simulator and forwarding this entry to MongoDB. Second statistics shows the average time between the sending data to the MongoDB and the processing of received message. The third statistical data presents the average time difference between processing and entering the data in the SQLite database.

During the first test, there were no major problems during processing and data storing. RabbitMQ received messages and forwarded them to its subscribers without a problem, as it can be seen in Fig. 5.

As shown in Fig. 5 statistical data recorded with RabbitMQ monitoring tool showed that in every queue average of 6 messages per second are processed. For every processed message microservices sent the acknowledgment back to RabbitMQ notifying it that message was successfully processed and that it can be deleted from the queue.

On the other hand, statistical data about computer resources utilization showed that RabbitMQ takes per queue from 279 KB to 1.1MB of memory. All retrieved statistical data indicates that there were no delays and that all messages were forwarded to the processing task in real-time. For each received message, its arrival to the destination is acknowledged.

The average time required to store the data to MongoDB from the moment of generation of the message is 331 milliseconds in the first test. It should be emphasized that the data transmission time is included in the message transfer time. The average data processing time is about 3.4 milliseconds, while in the first test the average time for entering data from processing up to the entry in the SQLite database is about 4.7 milliseconds. Also, it is necessary to emphasize that the maximum recorded time required from generating the data up to its registration in MongoDB is 5.337 milliseconds. Also, the maximum recorded processing time is 118 milliseconds and time needed for saving data to SQLite is 614 milliseconds.

During the second test, similar data were recorded, although it should be emphasized that on several occasions there were delays in process of storing data in the SQLite database. As in the first test, RabbitMQ correctly received messages and sent them to their subscribers. Figure 6. gives a statistical overview of RabbitMQ during the second test.

In Fig. 6 statistical data retrieved with RabbitMQ monitoring tool are presented. In the second test, average of 30 messages per second are processed in every queue. As in the previous test, every microservices sent an acknowledgment message back to RabbitMQ that the messages was successfully processed and that it can be deleted from the corresponding queue. Total of 30,000 messages are received and processed via microservices.

Queues

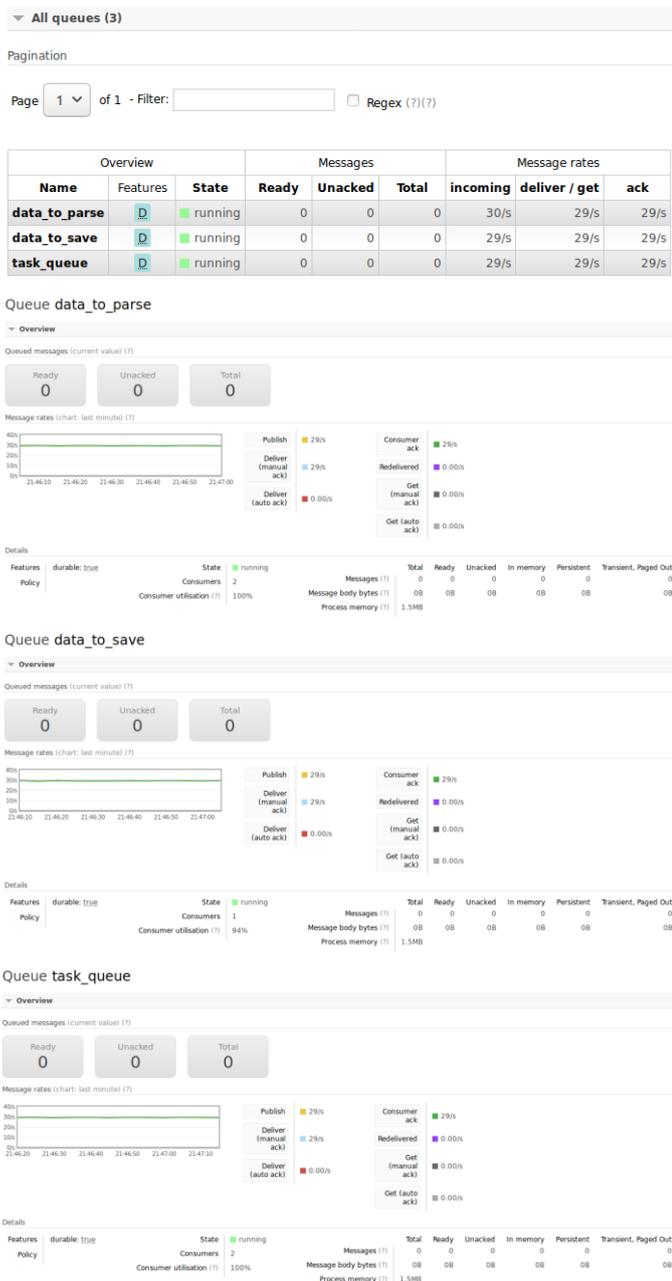


Figure 6. Statistical data of RabbitMQ performance during the second test

Looking at statistical data on computer resource utilization, it can be noticed that RabbitMQ uses per queue around 1.5MB. In this case, the majority of delays occurred during data storage process in SQLite database.

During the second test, average time required to store data in MongoDB from the moment of generating the message is about 440 milliseconds. About 3.2 milliseconds was needed for data processing, while the average time required to enter data in the SQLite database was 1,330 milliseconds. These results indicate the main problem with proposed architecture and usage of SQLite in it. Also, it is necessary to point out that the maximum recorded time required for the entry of one record in the SQLite database, from the moment of its processing is 29.646 seconds.

VII. CONCLUSION

Modern companies are increasingly taking advantage of IoT devices and use their ability to collect massive amounts of data. The usage of IoT devices and system could lead companies to significant cost reductions. Together with the advantages, a large amount of data generated by these devices has set rigorous standards for the development of new applications that need to be able to process and store data in real-time.

New architectures and approaches in the development of applications with the usage of microservices can respond to imposed challenges. As shown in this paper, processing of data and their storage is possible with the utilization of such architectures. In this paper we presented the proposal of lightweight system developed for small data center monitoring. The development of this system is in the initial phase, and the results indicate the possibility of its application in real environments. Further research will be performed on the data center system described in the paper.

The test platform, developed for testing and evaluation of the proposed system is also described in this paper. The focus in this testing is given to examination of two DBMS performances in IoT microservice based system. Two tested DBMS are MongoDB and SQLite. MongoDB is selected for the research because it represents most popular NoSQL database. SQLite is chosen because it represents most popular database that is completely platform independent and requires no server for its operation.

This platform showed that proposed lightweight architecture based on microservices could deal with the larger amount of sensor data in the case of using MongoDB. On the other hand, the usage of SQLite database is not recommended due to the lower performances and test results. The SQLite database shows a lower performance in the case of the higher data amount generated from the sensor stations. For future usage, the SQLite will be replaced with other DBMS. The selection of additional DBMS will be made on analyses of data from DB-Engines Ranking [28] scores.

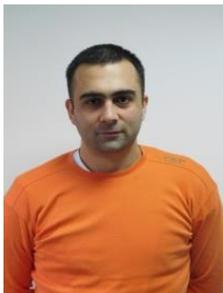
ACKNOWLEDGMENT

This research is supported by the Ministry of Education and Science of the Republic of Serbia under the project number TR32044 "The development of software tools for business process analysis and improvement," 2011-2019.

REFERENCES

- [1] Forbes Insights, "The Internet of Things: From theory to reality - How companies are leveraging the IoT to move their businesses forward", [Online]. Available: <http://info.forbes.com/rs/790-SNV-353/images/Hitachi%20IoT%20Report.pdf>, 2017. [Accessed: January 2019]
- [2] Capgemini Digital Transformation Institute, "Unlocking the business value of IoT in operations", [Online]. Available: https://www.capgemini.com/wp-content/uploads/2018/03/dti-research-report_iiot-in-operations_web.pdf, [Accessed: January 2019]
- [3] Statista, "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)", [Online]. Available: <https://www.statista.com/statistics/471264/iiot-number-of-connected-devices-worldwide/>, 2019. [Accessed: January 2019]
- [4] M. Malić, D. Dobrilović, D. Malić, "Lightweight microservice architecture for small data center monitoring supported with RabbitMQ", Proceedings of 18th International Symposium INFOTEH-

- JAHORINA, pp 436-441, 20-22 March, Jahorina, Republic of Srpska, Bosnia and Herzegovina, 2019.
- [5] F. Terroso-Saenz, A. González-Vidal, A. P. Ramallo-González, A. F. Skarmeta, "An open IoT platform for the management and analysis of energy data", *Future Generation Computer Systems*, Vol. 92, 2019, pp 1066-1079, <https://doi.org/10.1016/j.future.2017.08.046>.
- [6] E. Manavalan, K. Jayakrishna, "A review of Internet of Things (IoT) embedded sustainable supply chain for industry 4.0 requirements", *Computers & Industrial Engineering*, 2018, <https://doi.org/10.1016/j.cie.2018.11.030>.
- [7] B. Götz, D. Schel, D. Bauer, C. Henkel, P. Einberger, T. Bauernhansl, "Challenges of Production Microservices", *Procedia CIRP*, Vol. 67, 2018, pp 167-172, <https://doi.org/10.1016/j.procir.2017.12.194>.
- [8] M. Ciavotta, M. Alge, S. Menato, D. Rovere, P. Pedrazzoli, "A Microservice-based Middleware for the Digital Factory", *Procedia Manufacturing*, Vol. 11, 2017, pp 931-938, <https://doi.org/10.1016/j.promfg.2017.07.197>.
- [9] S. Newman, "Building Microservices - Designing Fine-Grained Systems", O'Reilly Media, 2015.
- [10] S. Fowler, "Production-Ready Microservices - Building Standardized Systems Across an Engineering Organization", O'Reilly Media, 2016
- [11] R. Rodger, "The Tao of Microservices", Manning Publications Co., helter Island, NY, USA, 2018. <https://www.rabbitmq.com/>
- [12] Pivotal Software, Inc., "Messaging that just works — RabbitMQ", [Online]. Available: <https://www.rabbitmq.com/>, Retrieved January 2019. [Accessed: January 2019]
- [13] J. Kuch, "RabbitMQ Hits One Million Messages Per Second on Google Compute Engine", [Online]. Available: <https://content.pivotal.io/blog/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine>, Posted March 25, 2014. [Accessed: January 2019]
- [14] D. Ramel, ".NET Core 2.1 Powers 34 Percent Bing Performance Boost", [Online]. Available: <https://visualstudiomagazine.com/articles/2018/08/22/bing-net-core.aspx>, Posted 08/22/2018. [Accessed: January 2019]
- [15] S. Cass. P. Bulusu, "Interactive: The Top Programming Languages 2018", [Online]. Available: <https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>, Posted 31 Jul 2018. [Accessed: January 2019]
- [16] A. Celesti, M. Fazio, "A framework for real time end to end monitoring and big data oriented management of smart environments", *Journal of Parallel and Distributed Computing*, 2018, <https://doi.org/10.1016/j.jpdc.2018.10.015>.
- [17] BSON Specification Version 1.1, [Online]. Available: <http://bsonspec.org/spec.html>, Retrieved January 2019. [Accessed: January 2019]
- [18] B. Dayley, "NoSQL with MongoDB in 24 Hours, Sams Teach Yourself", Pearson Education, USA, 2015.
- [19] P. Raj, "Chapter One - A Detailed Analysis of NoSQL and NewSQL Databases for Bigdata Analytics and Distributed Computing", Editor(s): Pethuru Raj, Ganesh Chandra Deka, *Advances in Computers*, Elsevier, Vol. 109, 2018, pp 1-48, <https://doi.org/10.1016/bs.adcom.2018.01.002>.
- [20] Cherradi, G., Bouziri, A. E., Boulmakoul, A., & Zeitouni, K. (2017). Real-Time Microservices Based Environmental Sensors System for Hazmat Transportation Networks Monitoring. *Transportation Research Procedia*, 27, 873–880. doi:10.1016/j.trpro.2017.12.087
- [21] Benayache, A., Bilami, A., Barkat, S., Lorenz, P., & Taleb, H. (2019). MsM: A microservice middleware for smart WSN-based IoT application. *Journal of Network and Computer Applications*. doi:10.1016/j.jnca.2019.06.015
- [22] Wan, X., Guan, X., Wang, T., Bai, G., & Choi, B.-Y. (2018). Application deployment using Microservice and Docker containers: Framework and optimization. *Journal of Network and Computer Applications*, 119, 97–109. doi:10.1016/j.jnca.2018.07.003
- [23] Ciavotta, M., Alge, M., Menato, S., Rovere, D., & Pedrazzoli, P. (2017). A Microservice-based Middleware for the Digital Factory. *Procedia Manufacturing*, 11, 931–938. doi:10.1016/j.promfg.2017.07.197
- [24] Ciuffoletti, A. (2015). Automated Deployment of a Microservice-based Monitoring Infrastructure. *Procedia Computer Science*, 68, 163–172. doi:10.1016/j.procs.2015.09.232
- [25] Krämer, M., Frese, S., & Kuijper, A. (2019). Implementing secure applications in smart city clouds using microservices. *Future Generation Computer Systems*, 99, 308–320. doi:10.1016/j.future.2019.04.042
- [26] Celesti, A., Lay-Ekuakille, A., Wan, J., Fazio, M., Celesti, F., Romano, A., Bramanti, P., Villari, M., "Information management in IoT cloud-based tele-rehabilitation as a service for smart cities: Comparison of NoSQL approaches", *Measurement*, Volume 151, 2020, 107218, <https://doi.org/10.1016/j.measurement.2019.107218>.
- [27] Van der Veen, J. S., van der Waaij, B., & Meijer, R. J., *Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual*. 2012 IEEE Fifth International Conference on Cloud Computing. doi:10.1109/cloud.2012.18
- [28] DB-Engines, <https://db-engines.com/en/ranking>
- [29] M. Marcu, S. Ficiu, D. Dobrilovic, M. Popa, B. Odadzic, „Cross-border infrastructure for educational webcasting“, *Proceedings of International Conference on Applied Internet and Information Technologies AIIT 2015*, pp 225-230, October 23, Zrenjanin, Serbia, 2015.
- [30] S. Fuicu, M. Popa, D. Dobrilovic, M. Marcu, R. Bogdan, "Developing Distance Learning Environments in the Context of Cross-Border Cooperation", *BRAIN Broad Research in Artificial Intelligence and Neuroscience*, pp 52-58, Vol. 8, Issue 1, April, 2017.
- [31] D. Dobrilović, Ž. Stojanov, Z. Čović, J. Simon and N. Petrov, "Model of data center temperature monitoring system with the use of open source hardware," 2016 IEEE 14th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, 2016, pp. 221-226. doi: 10.1109/SISY.2016.7601501



Milan Malić was born in 1984. Currently works as a Software Engineer at PanonIT in Novi Sad. M.Sc degree in Information Technology from University in Novi Sad, 2016. Student of PhD studies at University in Novi Sad, Technical faculty “Mihajlo Pupin” in Zrenjanin. Published more than 30 scientific and technical publications. His research interests include IoT, NoSQL DB, MapReduce, GIS and programming languages C#, Python, C/C++, PHP.



Dalibor Dobrilović received PhD degree in 2012 in Information Technology at University of Novi Sad. He works at Technical faculty “Mihajlo Pupin” in Zrenjanin (University of Novi Sad) as associated professor. Published more than 50 in refereed journals, books, and conferences' proceedings. His areas of interest include Wireless communications, IoT, Smart City technologies, Open-source Hardware, Computer networks, Computer network security etc. Member of IEEE and ACM.



Dušan Malić was born in 1975. Currently works as a Professor of Applied Sciences at the Technical College of Applied Sciences in Zrenjanin. He defended (2012) at the Faculty of Management in Sremski Karlovci, University Union - Nikola Tesla in Belgrade, PhD thesis entitled "The possibility of improving the efficiency of maintenance functions by applying lean concepts - the fourth generation system of maintenance" and earned scientific degrees doctor of science in the scientific field of Management. Published more than 50 scientific and technical publications. Interested in the research in the field of engineering management and information technology



Željko Stojanov received PhD degree in Computer science and Informatics at the University of Novi Sad in Serbia. He works as an associate professor at the University of Novi Sad, Technical Faculty “Mihajlo Pupin”, Serbia. His research interests are in the fields of software engineering, business informatics, learning and knowledge management, data analysis, qualitative research, and human aspects of engineering. He is coauthor of more than 50 articles published in refereed journals, books, and conferences' proceedings. He participated in several research and industrial projects. He is the member of IEEE and ACM.