Rev. Roum. Sci. Techn.– Électrotechn. et Énerg.
Vol. **65**, *3–4*, pp. 259–263, Bucarest, 2020

ARCHITECTURE FOR MONITORING RISK SITUATIONS IN A UNIVERSITY ENVIRONMENT

FLORIN LĂCĂTUȘU[1], ANCA-DANIELA IONIȚĂ[1]

Key words: Cloud Computing, Risk situations, Building monitoring systems.

The use of Cloud technologies has become frequent in our daily lives. Our study was focused on the comparison of two solutions for the implementation of a monitoring system for a university building. Both of them collect data from sensors and a mobile app, but one is based on a virtual machine deployed on a local server and the other is hosted in a public Cloud. The latter is developed as a Cloud-native application implemented with Linux containers. The two solutions were analyzed for scenarios of building monitoring, suited for identifying risk situations.

1. INTRODUCTION

The implementations using Cloud technologies have become more frequent because of their underlying advantages and prevalence. This growing trend is determined by the multitude of available offerings from different Cloud providers. Another factor of influence is represented by the flexibility offered by Cloud Computing. Therefore, the concerns of hardware choices, costs, installation, and maintenance do not exist in this kind of environment. The advantages of Cloud were exploited in multiple industries, from personal use cases to complex processing, solving elaborate problems that require a significant amount of computing resources. For example, [1] proposes a Cloud use case that can be applied in the medical field, to rehabilitate neurological patients.

The biggest problem that had to be solved in the past was the cost of acquiring such hardware for the respective need. In the Cloud, every resource is taxed for the time it is used. Thus, the use of high-performance equipment is accessible to a larger group of people.

Another important task in the context of building monitoring systems is the implementation of a sensor network, with the scope of collecting various environmental parameters. Sensor networks are used in many domains. A use case where the sensor networks are used in the medical field, to detect tumors is presented in [2].

In our study, we conducted a comparison of two implementations of a university building monitoring system, using a local server approach and a Cloud-native implementation respectively. For both approaches, the result was a building monitoring system with the facility to provide alerts based on data received from sensors and user reports. The provided services can be used by individuals who have access to this architecture, to monitor the building, and to be alerted in case of emergency. The data gathered from sensors are also essential because they can resemble events that can characterize emergencies. The next logical step for this kind of structure was the implementation of a monitoring software application and a user reporting system. The goal of the monitoring application was to display the information received from sensors and the user reports, in the form of a dashboard. The data received is displayed in real-time.

The next section of the paper presents some related work. Then, Section 3 presents two implementations of the system for monitoring risk situations in a university environment. In Section 4 there is a comparison between the use case of building monitoring using a local server and the one based on the deployment in IBM Cloud, using containers. The purpose is to investigate what kind of solution is best suited to implement such a monitoring and alerting system.

2. RELATED WORK

The environment analysis using sensors is often present in the scientific literature. Our work is related to the approach presented in [3], related to the monitoring of a laboratory environment using a publish-subscribe deployment model; based on different parameters, alarms are triggered when a certain threshold is achieved. The implementation of a building monitoring system was also presented in [7], where the scope was to create an optimal environment for its occupants and to reduce its energy footprint.

Concerning our architecture based on Cloud Computing, a relevant study was conducted in [4], regarding the performance of containers vs. virtual machines. It gives a comparison between the boot times of containers and those of virtual machines that are virtualized using the KVM (Kernel-based Virtual Machine) hypervisor. The results confirmed the theory of containers, i.e. the fact that the containers boot faster than the virtual machines. They also realized a comparison of the calculation speed of both technologies, and the containers proved to be well faster for the computations tested using the Python language.

The underlying advantages of using Cloud Computing were also presented by [5], for various deployment options, i.e., public, private, and hybrid, as well as for the most used models, such as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service).

Kubernetes characteristics of high availability were discussed in detail in [6]. The study was conducted using multiple applications, to measure the healing capacity of Kubernetes in different situations.

3. SYSTEM FOR MONITORING UNIVERSITY RISKS

The system presented in this article consists of: a sensor network that collects data, monitoring, and notification software, an application that collects reports from its users.
The main component of the sensor network is the node. It hosts both the data acquisition and the messaging software, as well as the hardware connection to the sensors. On one node it is possible to connect several types of sensors that capture different environmental data. Also, the software

processes raw data from sensors and sends them forward. The system users can also connect to a node independently and thus, get the data captured by it. The monitoring system provides information about the university building environmental parameters; if one of these parameters comes out of a set threshold, it generates alerts. The reporting application, called UniCris, was previously described in [8]. Its scope is to collect user reports in case of an emergency. The reports are sent over the Internet or as SMS. The monitoring system collects reports from users and compares them with the data that originated from sensors.

Local virtual machine implementation

The monitoring and reporting application is composed of multiple parts (Fig.1). First, some sensors collect the environmental data from each university room where they are installed.

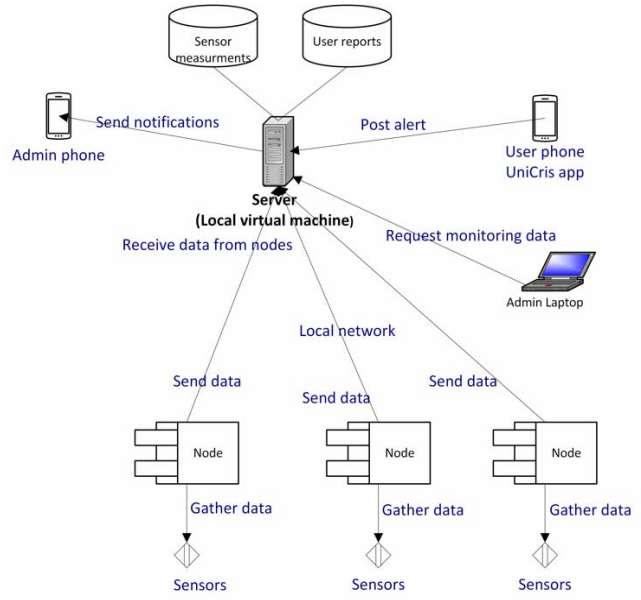

Fig.1 – Local system architecture.

The data gathered from sensors are sent to a server and stored in a database. Another part is represented by the monitoring application that is hosted on the same local server. This application gathers data from the database and uses an algorithm to decide if there is an alert or not. The alerting system is composed of an Android application that is used by students and teachers from the building to report events. These reports are sent to the backend application on the server, and an algorithm checks if the data from users correspond with the data gathered from sensors.

If the algorithm decides that there is an alert, a notification is sent to the administrator's mobile phone. As in the case of data that come from sensors, the user reports are saved on a database. This feature is related to another component of the system – the monitoring application, which is practically a web dashboard, written in Angular; it offers statistics and analyses of reports from the database.

For the local implementation, we chose a Linux virtual machine; we used CentOS as the version of Linux for this machine. For testing purposes, we allocated 4 GB of RAM and 2 cores for this server.

Raspberry Pi was chosen as a sensor node because it has a Linux OS installed on it and it supports a further extension if multiple nodes must be attached to this network. Because it is dependent on the wireless network to send data, it needs a constant power source. This type of node can be easily swapped with an ESP8266 that drives the NodeMCU development board, an alternative recommended if the power source is important because this microcontroller could run on batteries for months, compared with the Raspberry, which consumes more power.

On the data acquisition side, we used BMP 180 as pressure, temperature, barometer, and altitude sensor, as well as an MQ02 gas sensor that senses different types of gases, such as Methane, Butane, LPG, and Smoke. If the detected value is greater than 300 ppm, smoke is detected.

On the Raspberry node, a Python script was used to collect data from sensors. The data was written on the database hosted on the local server. Only reports were the temperature is greater than 50 degrees Celsius, or smoke is greater than 300 ppm are recorded. These values were simulated for testing purposes. The date and time of the respective events were saved in the database. The data are then compared with the information sent by users for the same time intervals, to decide whether a notification is sent or not.

The image from Fig. 2 presents a sequence diagram for this system, represented in UML (Unified Modeling Language). The user makes a report; the report is sent to the database; the Raspberry node realizes a continuous data acquisition from the sensor. The data are sent to the server. If the temperature is bigger than 50 degrees Celsius, or smoke is greater than 300 ppm, the data is saved into the database. The backend makes requests to the database every minute. It uses an algorithm to compare the reported data with the data acquired by the Raspberry. If the time and the location match, it sends a notification to a responsible person that a possible risk event is in progress.

4. CLOUD-BASED IMPLEMENTATION

This section presents the implementation of the alerting system in a public cloud environment. This may have several advantages, like the ability to make it accessible to multiple users, who are connected to different networks. IBM Cloud offers multiple services to implement different types of applications. In the case of this monitoring and alerting application, the service used is Kubernetes – a container orchestration system [9]. IBM Cloud was selected because it offered a free trial for the Kubernetes Cluster. Other Cloud providers were also investigated, such as AWS and Microsoft Azure, but IBM Cloud was the most financially convenient for our work.

Thus, the goal for this research was to migrate the system that was presented in the previous section to a Cloud environment. Earlier, we implemented the system on a Linux virtual machine. In that case, the necessary tools were installed in a local environment. In this second implementation, the free Kubernetes service offered a worker node to host the containers. The components of the application practically remain the same as in the previous version; the only difference is that the applications run in separate containers.

This approach simplifies the whole process because the application is not hosted on a local virtual machine and all the components are isolated. Also, because it is hosted in the Cloud, it can be accessed at any time. Another advantage is offered by the property of Kubernetes container orchestrator, to keep the application online, even

if something happens and a crash occurs; Kubernetes will keep the number of replicas online. The pod is the smallest unit in Kubernetes. One pod may contain one or more containers.

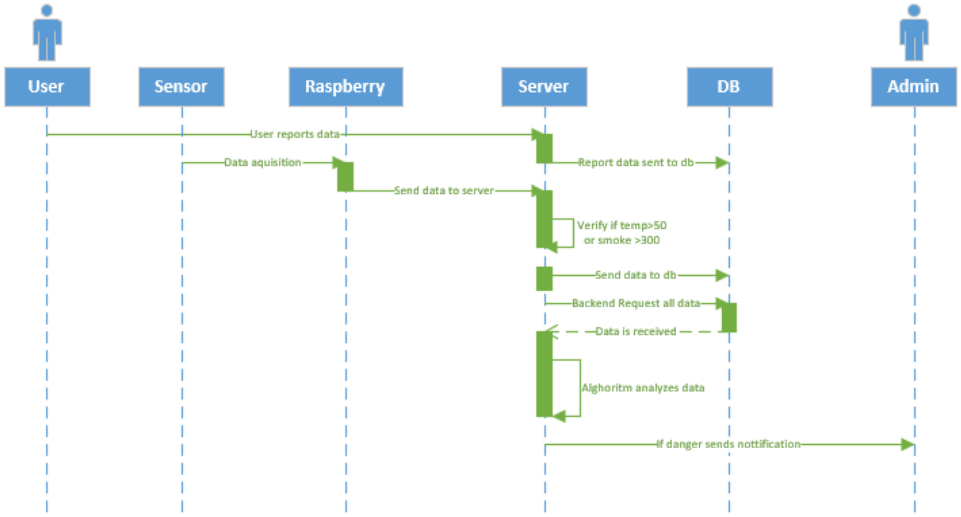

Fig. 2 – System flow (Sequence diagram).

Containers that reside in the same pod are usually those that depend on each other to function. The Kubernetes service offered by IBM Cloud in the free version allows the creation of a cluster with a single wor kerWhenn a service is created, a cluster of one or more nodes is deployed. Thus, the configuration has a master node that deals with the administration of the pods on the nodes. The nodes are practically virtual machines where pods are found. Like any Cloud provider, IBM offers a service that guarantees its availability over 99% of time. There before, the service will be available as needed. Besides, the ability of Kubernetes to manage nodes and maintain the available pods is a plus for the high availability of the system [10].

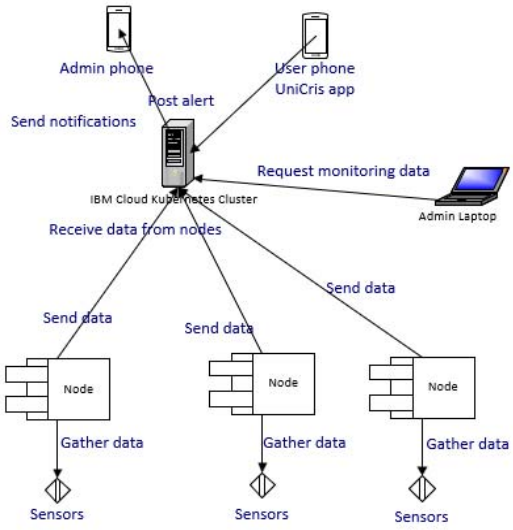

Fig.3 - Cloud system architecture.

Figure 3 presents our application architecture, using the IBM Cloud Kubernetes service. Thus, the user posts an alert that is added directly to the database found on a pod, in the cluster. The development boards collect data from the sensors in every room where they are installed. Based on the collected data, alerts are sent directly from them to the database located on a pod. The data is sent when a parameter received from the sensor exceeds a certain threshold. To write directly to the database, we use a helper pod that contains PHP and Python scripts that insert data into the database.

If one compares this architecture with the local one, presented previously, the main difference between them is the implementation of the system using a Kubernetes cluster in IBM Cloud, as opposed to the implementation of the system on a local server. This approach takes the data processing away from the university building where the event happens.

Fig. 4 presents the pods running on the Kubernetes cluster worker node, i.e.:

• DB – the measurements database for sensors and user reports;

• DB Admin - the web administrator for the database;

• Utility Scripts - for scripts that send data to de database; the scripts from this pod are called by the application that runs on the development board and the Android smartphone;

• NodeJS Backend – a pod containing a NodeJS container; it runs the comparison algorithm and sends notifications to the Administrator; it also saves the notifications in the database and offers an API for the frontend, to see reported data;

• Angular Front-end - running on a pod containing a web server (Nginx).

This solution is more effective than installing the same tools on a virtual machine, as presented in the previous section. There is isolation between different applications running in the container and one does not need to install these tools anymore, because one uses images containing the desired software. Another important aspect is virtual machine management, which in the case of containers is resolved by the IBM Cloud support team. The purchase and the costs of the hardware running the machine are inexistent because everything is stored in the Cloud.

5. COMPARISON AND DISCUSSION

The virtual machine approach has the advantage of being locally deployed, as well as the sensor nodes (Fig. 5). Thus, data gathering can be done over a local network. This aspect has advantages such as the fact that it does not depend on a

public Internet connection, and the data are sent on a local network. As a disadvantage, a disaster that occurs on a building that also stores the report can raise problems if the electrical grid is affected and the system is automatically shut down. In this case, the reporting application (UniCris) can be used, because it sends the reports via SMS and Internet (Fig. 6).

Thus, this kind of report also depends on the operation of the local network where the disaster occurs. From the performance point of view, the application is directly influenced by the hardware that runs the backend system. So, a performant system would require the acquisition of expensive hardware that would add unnecessary up-front costs, in a world that offers the possibility to "rent" the hardware for its use period. The situation where this approach would have an upper edge compared with the Cloud deployment is when the Internet connection is down and the data from sensors cannot be sent to the server. In this case, half of the system functionality would be down.

The biggest advantage of the Cloud system is the separation of the processing infrastructure from the actual building (Fig. 7). Hence, in the case the electrical grid of the building is down, the user reports will continue to work as usual, because the system is not dependent on the local infrastructure. The only problem that arises when this kind of issue happens is that the reports from sensors are not available anymore, since they use the local network connection of the building. Moreover, the system in the Cloud has an availability rate higher than 99 %, so one is assured by the Cloud provider that the system is up.

From a performance point of view, in Cloud, as well as for the local deployment, a more performant system would be pricier. The difference, when one works in Cloud, is that there are no up-front costs; one only pays for as long as the system is used. In IBM Cloud, the Kubernetes service has an hourly payment system. Another advantage of the Cloud system is that the administration of the virtual machines that sustain the cluster is done by the Cloud provider.

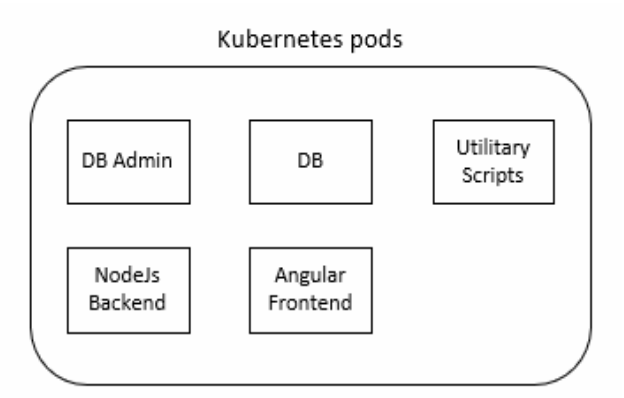

Fig. 4 – Kubernetes cluster pods.

The Cloud system is suited to handle the notifications and building monitoring in case of an emergency because it is not dependent on the local infrastructure. In this case, the users can still make reports that are sent to the administrator. If the Internet connection is down, the only system that is affected is represented by the sensor network, which will not be able to send alerts. Therefore, in case of an emergency where the electrical system of the building or Internet connection are affected, the user reports will not be influenced by the downtime of the system (Fig. 8).

The Cloud system is suited to handle the notifications and building monitoring in case of an emergency because it is not dependent on the local infrastructure. In this case, the users can still make reports that are sent to the administrator.

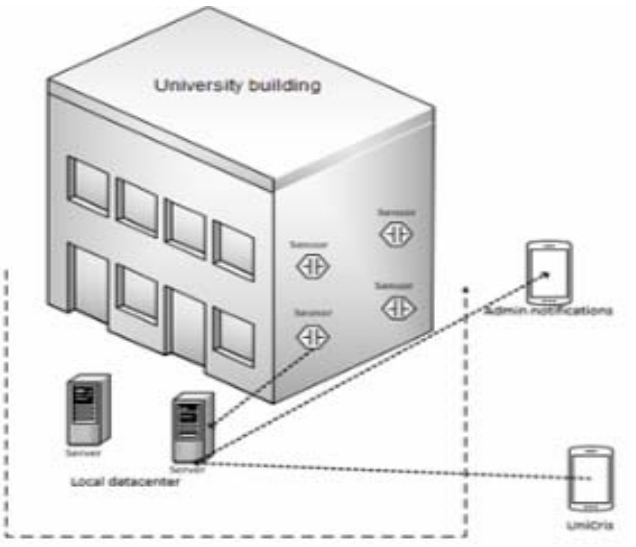

Fig. 5 – Local system.

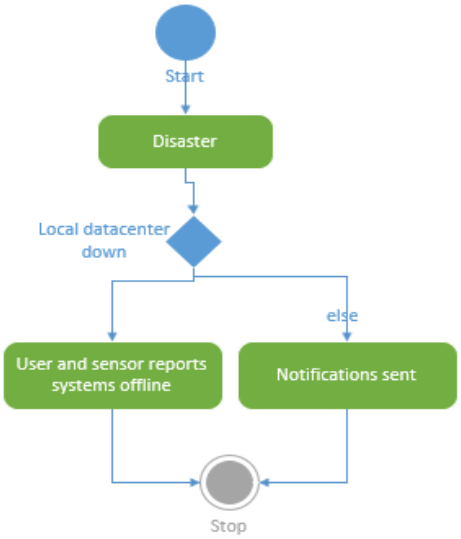

Fig. 6 – Local system scenario.

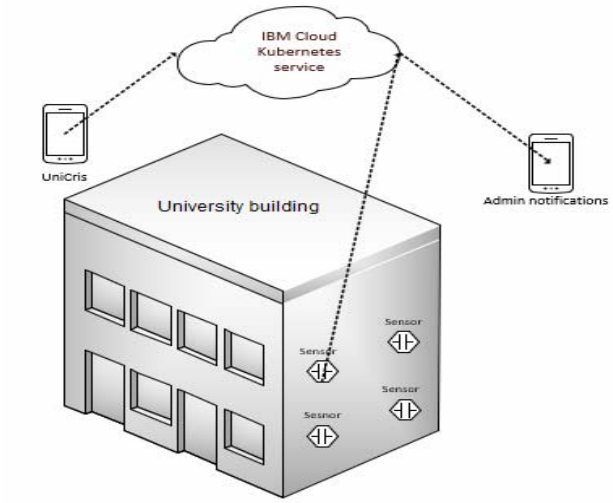

Fig. 7 – Cloud system.

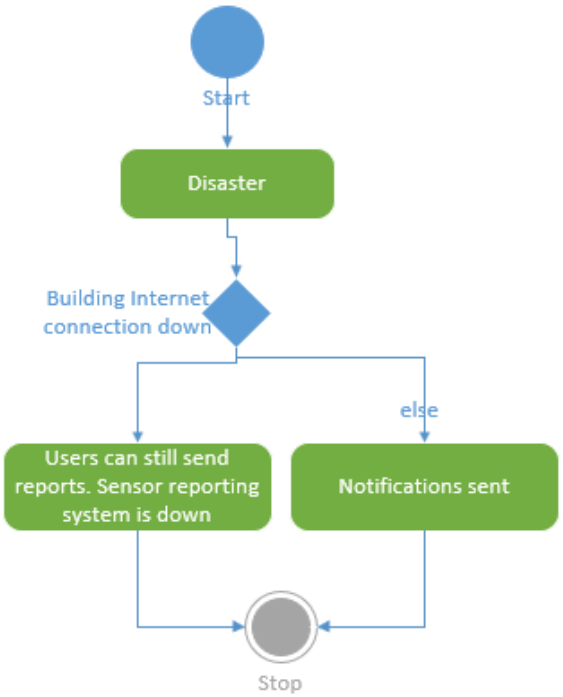

Fig. 8 – Cloud system scenario.

If the Internet connection is down, the only system that is affected is represented by the sensor network, which will not be able to send alerts. Therefore, in case of an emergency where the electrical system of the building or Internet connection are affected, the user reports will not be influenced by the downtime of the system (Fig. 8).

6. CONCLUSIONS

The article presented and analyzed two implementations of a university building monitoring system. The underlying functionalities are the same: data acquisition from the environment, and user reports. The difference is represented by the location where data are processed, and the technologies used to implement the system. Using a public Cloud and a Platform as a service (PaaS) implementation is much simpler when compared with a local server approach because the user does not have the OS administration problems that usually appear when using virtual machines, such as security patches, OS updates, etc. Moreover, for testing purposes, it is easier to have all the data stored in Cloud than resident on a virtual machine. The installation of the Kubernetes Cluster and all its administrative tasks is done automatically.

Another advantage is the high availability of 99% + application readiness that is provided by the Cloud itself. If the application sits in a private environment, this thing must be assured by the user. In the case of a university building monitoring application, the use of these Cloud technologies offers the possibility of high availability of the system. The only thing that must be assured to keep this property is a strong connection from the reporting tools to the Cloud.

The disadvantage of the Cloud approach is the dependence on the Internet connection. Thus, all the system functions are available only when the connection between the university building and public Cloud is established. As a solution for this inconvenience, one can use Direct Link and create a private connection between the public Cloud datacenters and the university building, using the Internet provider infrastructure. Furthermore, for redundancy purposes, multiple Internet providers can be used.

In conclusion, the use of Cloud technologies to monitor a university building is beneficial for providing the necessary high availability of the system and is cheaper than implementing a system that requires the acquisition of computing hardware, provided that one should assure a strong connection between the Cloud and the local system.

Received on March 23, 2020

REFERENCES

1. D. Cârstoiu, V.E. Oltean, S.M. Nica, G. Spiridon, *A cloud-based architecture proposal for rehabilitation of aphasia patients,* Rev. Roum. Sci. Techn. – Électrotechn. Et Énerg., **62**, 3, pp. 332–337 (2017).
2. G.M. Vasilescu, I. Bârsan, G. Kacso, M.E. Marin, M. Maricaru, L.N. Demeter, *Two devices equipped with temperature sensors used to detect and locate incipient breast tumors*, Rev. Room. Sci. Technol. – Électrotechn. Et Énerg., **63**, 4, pp. 441–445 (2018).
3. R.N. Pietraru, L.G. Zegrea, A.D. Ioniță, *Publish-subscribe deployment alternatives for scenarios related to university laboratory safety*, The XIth International Symposium on Advanced Topics in Electrical Engineering, March 28-30, Bucharest, Romania, 2019.
4. B.B. Rad, H.J. Bhatti, M. Ahmadi, *An Introduction to Docker and analysis of its performance*. IJCSNS International Journal of Computer Science and Network Security. **173**, 8 (2017).
5. O.A. Isaac, M. Ananya, FAgonyno, R. Goddy-World, *Cloud computing architecture: a critical analysis.,* IEEE Proceedings of the 2018 18th International Conference on Computational Science and Its Applications, Melbourne, Australia, 2 – 5 July 2018, p 1-7.
6. L.A. Vaughan, M.A. Saied, M. Toeroe, F. Khendek, *Kubernetes as an availability manager for microservice applications,*arXiv.org, Cornell University (2019).
7. T. Stavropoulos, A. Tsioliaridou, G. Koutitas, D. Vrakas, I. Vlahavas. *System architecture for a smart university building.* ICANN '10 Intelligent Environmental Monitoring, Modelling and Management Systems for better QoL Workshop, Thessaloniki, Greece, 2010, p. 477-482.
8. A. Olteanu, F. Lăcătușu, M. Lăcătușu, I. Crăciun, A.D. Ioniță. *Mobile application for crisis situations in a university campus*, The International Scientific Conference eLearning and Software for Education, Buchares, Romania, 2018, p. 280-287.
9. ∴ *Kubernetes definition and features* - https://kubernetes.io/, accessed 2019.
10. ∴ *Kubernetes Networking* - https://kubernetes.io/docs/concepts/services-networking/, accessed 2019