University POLITEHNICA of Bucharest
Automatic Control and Computers Faculty
Automation and Industrial Informatics Department

PhD Thesis Summary

Smart Building Monitoring in Cloud for Detecting Hazardous Events

Presented by

Drd.Ing. Florin LĂCĂTUȘU

Supervised by

Prof.Dr.Ing. Anca Daniela IONIŢĂ

2023

Bucharest, Romania

Abstract

The transformation of people's lives through the use of smart buildings has become a trend for residential purposes, university and corporate campuses, and commercial complexes, where it is important to focus on both socioeconomic and environmental factors that can be facilitated by smart technologies, including the Internet of Things (IoT) and cloud computing. The interconnection between these two topics - smart buildings leveraged by IoT devices, and cloud computing for sustaining the management of software architecture components - has gained a lot of interest from the scientific community; it mostly benefits from the elastic computing and the ease of access for the building operators. One of the objectives of my thesis is to design smart building architecture by combining the advantages provided by an edge network located within a building and the cloud hosting the monitoring application. The result is the Edge Watcher System (EWS), which was conceived for monitoring a complex of smart buildings and detecting hazardous events via a cloud-native web application that runs on Kubernetes. Regarding the contributions to the aforementioned topics, I am proposing a new smart building monitoring system with hazard events detection capabilities including the design of distributed architecture solutions used to evaluate the proposed solution. Therefore, four main architecture design options were presented that cover the location of the monitoring software and the local building edge network configurations. Besides this, the thesis also shows the validation of the capabilities of the Edge Watcher System building monitoring system by integrating it into a real building where sixteen test cases were defined based on the design options presented. Also, another important topic that the thesis covers is represented by the evaluation of scaling capabilities using up to one thousand simulated edge nodes for different scenarios. Each scenario corresponds to a real-world building type such as a small apartment, a house, a small residential building, an office building, and a complex of buildings where a different number of edge nodes would be installed. Also, for each scenario, a comparison was made between Edge Watcher System monitoring service installed in a Kubernetes cluster located on the monitored building and Edge Watcher System monitoring application installed in the IBM Cloud Kubernetes service. Both local and cloud Kubernetes cluster configurations used for testing were able to perform well in most of the test cases. However, a more powerful cluster than the ones tested is recommended for a complex of buildings that contain more than 100 edge nodes. Based on the results provided by these evaluations both the real-world one and the scaling evaluation, the thesis also presents recommendations for the best architecture to be used for the Edge Watcher System.

Acknowledgements

First of all, I would like to thank Professor Anca Daniela Ioniță for her guidance, support, patience, and availability during my Ph.D. studies journey. She is the best mentor and advisor and I couldn't achieved the same results without her advices.

I want also to thank my Ph.D. committee, Professor Daniela Saru, Professor Florin Daniel Anton, and Professor Adriana Olteanu for their guidance and comments that helped me improve my work.

Furthermore, I want to thank my brother, Marian Lăcătușu, and my good friend Ioan Damian with whom I successfully collaborated on multiple articles during our Ph.D. journeys.

Last but not least, I want to thank my family for their support and encouragement during my Ph.D. work.

Contents

List of Tables

List of Figures

1. Introduction

One of the most discussed subjects in the scientific community and IT industry is the migration to cloud technologies of previous solutions. 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, Carstoiu et al. propose a cloud use case that can be applied in the medical field, to rehabilitate neurological patients [1]. 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 purpose of collecting various environmental parameters. Sensor networks are used in many domains, like in the medical field, to detect tumors [2]. More and more implementations will migrate to this new practice, because of the flexibility that is provided by it, and its underlying advantages. The concept of renting hardware is not new. It was used for many years in the IT industry in the form of Infrastructure as a Service (IaaS) solutions. The implementations that adhered to this method were migrated from local servers to cloud virtual machines. The advantages that came with this kind of deployment were strictly related to the hardware costs of the servers. Therefore, the acquisition of servers is replaced with cloud plans that offer the possibility of paying for the respective hardware only for the time that it is used. This possibility of using the hardware only for the amount of time that is needed always fascinated me and it represents one of my top subjects of interest. I begin the study of cloud solutions with my dissertation project, where I migrated an early warning system to the cloud. The latest trend in the world of cloud applications is the use of Platform as a Service (PaaS) solutions. The difference between these two types of cloud methodologies (PaaS and IaaS) is the fact that, in the case of PaaS, the user is concerned only about the application implementation. All the operating system administrative tasks are executed by the cloud provider. As a result, this type of development makes the cloud accessible to multiple developers who want to migrate their applications to the cloud without the operating system administration worries. In my doctorate thesis, I approached the best methodologies for implementation of a building monitoring system, in the cloud, using a platform as a service solution. For this reason, I studied the implementation of monitoring applications using Linux containers with the result of developing cloud-native solutions. Every important cloud provider implements in its offering catalogue a Linux containers service. One of the most popular container orchestration services is Kubernetes. It was developed by Google, and it is the most important container orchestration solution in the IT business. The greatest advantage that is offered by a container implementation is related to application isolation and it offers the baseline for implementing loosely coupled microservices. This separation of application components into multiple small parts is recommended because the dependence between them is reduced, and, if a component is offline, the others will continue to serve their purpose. I chose Kubernetes as a viable solution for a building monitoring application because it provides high availability for its pods. A pod is the smallest unit in Kubernetes, and it can contain one or

multiple containers. The separation of the building infrastructure from its monitoring software is an important step in ensuring that, in case of a disaster, this emergency system will continue to function and send alerts to the helping parties.

The objective of my research is to design smart building architecture by combining the advantages provided by an edge network located within a building and the cloud hosting the monitoring application. Similar to [3], the main functionalities are monitoring multiple environmental parameters using different sensors, detecting abnormal situations, and sending notifications in both centralized and decentralized ways. After evaluating multiple architectural options, the result is the Edge Watcher System (EWS), which was conceived for monitoring a complex of smart buildings and detecting emergency events via a cloud-native web application. The aim is to improve the workflow and access of management systems within a smart building, by providing a flexible architecture to monitor and configure the local building edge network. The system was designed as a cloud-native application, which can be accessed from any location using many device types, such as smartphones, tablets, laptops, etc, using container model deployment within a Kubernetes cluster. This brings the advantage of high availability and self-healing of containers, a necessary feature for a building monitoring system that can send alerts in case of emergency [4]. Regarding the configuration capabilities provided by our system, I propose a method of generating configuration files based on user preferences; this information will be further used within the setup of the edge network nodes, in order to automate their configuration process and the communication with the cloud system.

The novelty of my research is related to the proposal of a new system that leverages the advantages of cloud technologies and the latest development methods, to offer a platform that is used to both monitor and configure building edge nodes, with the aim of detecting different alerts. Also, my solution is developed to be integrated into a broader system composed of multiple smart buildings with the purpose of notifying each other of possible alerts. The research also included the performance evaluation for simulated settings correspondent to a small apartment, a house, a small residential building, an office building and a complex of buildings.

Regarding the thesis structure, it is composed of the following chapters. Chapter 2 presents the analysis of the state of the art, with investigation on two main topics – Building monitoring for emergency management and Cloud and edge computing. Chapter 3 refers to the research objectives and method. Chapter 4 presents the original contributions of my thesis regarding the building monitoring in cloud topic. In Chapter 5 the verification and validation of the Edge Watcher System are discussed. Chapter 6 represents the contributions regarding the scaling evaluation of EWS. The thesis concludes with the conclusions (Chapter 7), summary of the original contributions, future perspectives and an appendix with implementation details.

2. Analysis of the State of the Art

In the research field, a literature study provides the base ground for every inquiry by virtue of creating a personal solution that improves on the current ones. My research is based on the study of intelligent buildings and how monitoring systems can be implemented in this regard, with the primary scope of emergency signaling in case of disaster. Adding to the before-mentioned themes, I also studied the implementation of the previously specified monitoring system in a cloud environment intending to separate the monitoring infrastructure from the building. Thus, the study on this research initiative started with the investigation of different articles that provide insights into current developments in building monitoring for hazard

management topic, which includes: building monitoring, hazard management, and emergency situations, and continued with the cloud and edge computing topics, which include cloud, IoT, edge, and fog computing, and performance monitoring (see Fig. 2.1). These are essential for the thesis main objective, which refers to smart building monitoring using cloud computing. Therefore, the monitoring software will be hosted using a cloud solution that gathers data from different IoT devices that communicate within an edge network to gather data from the building.

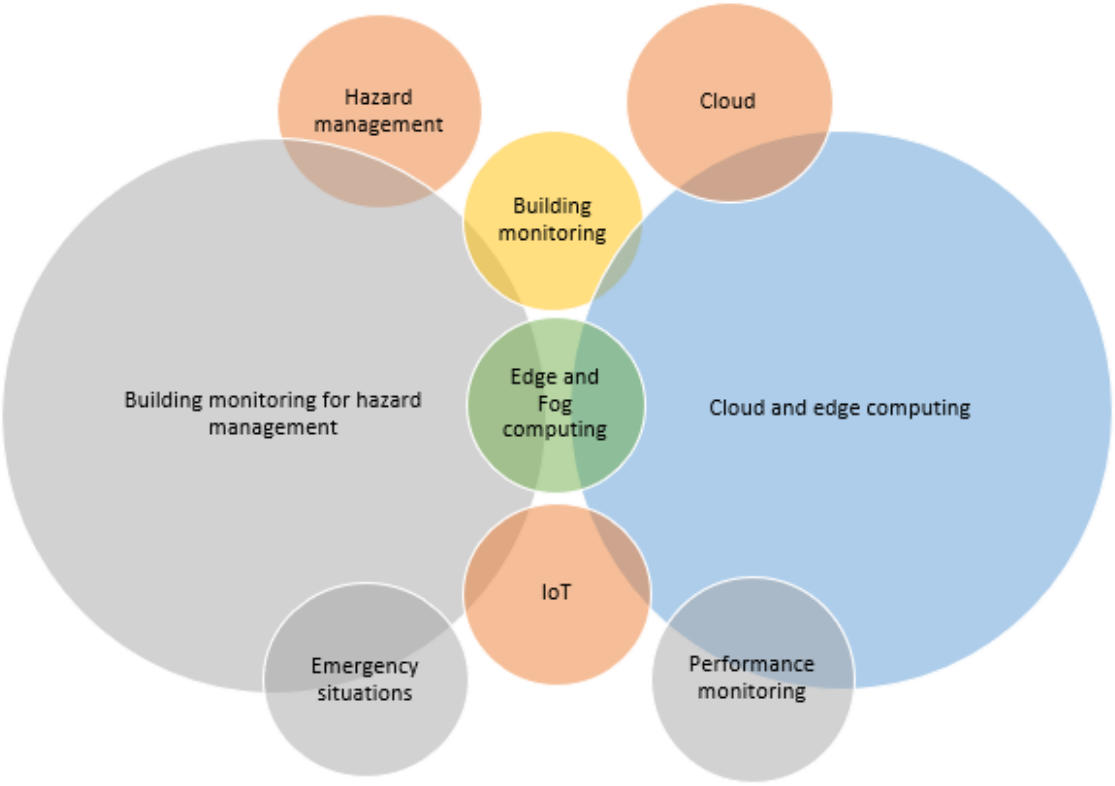

Fig. 2.1 Research topics

3. Monitoring Building Emergencies in Cloud

3.1 Research Objectives

My thesis objectives are related to the study of the topic of smart buildings integrated with cloud-native solutions. The goal is mainly related to the proposal of a new solution that is capable of integrating these two topics and can benefit from the cloud's elasticity and high availability. These advantages offered by cloud implementations, coupled with the flexibility of IoT devices installed at the building level, can provide a robust solution that can respond adequately in case of possible alerts detected at the building level. Besides these, one of the most important objectives would also be the evaluation of such a building monitoring system within different situations: from the real-world evaluation which implies the installation of the monitoring system in an actual building to the simulated scaling evaluation that employs hundreds and thousands of edge nodes. Therefore, below I have summarized the main thesis objectives:

1. New smart building monitoring system with hazard events detection capabilities

2. Design and evaluate distributed architecture solutions

3. Validate the building monitoring system by integrating it into a real building

4. Evaluate scaling capabilities using simulated edge nodes for different scenarios

3.2 Method

The method applied for conducting this research is presented in Fig. 3.1.

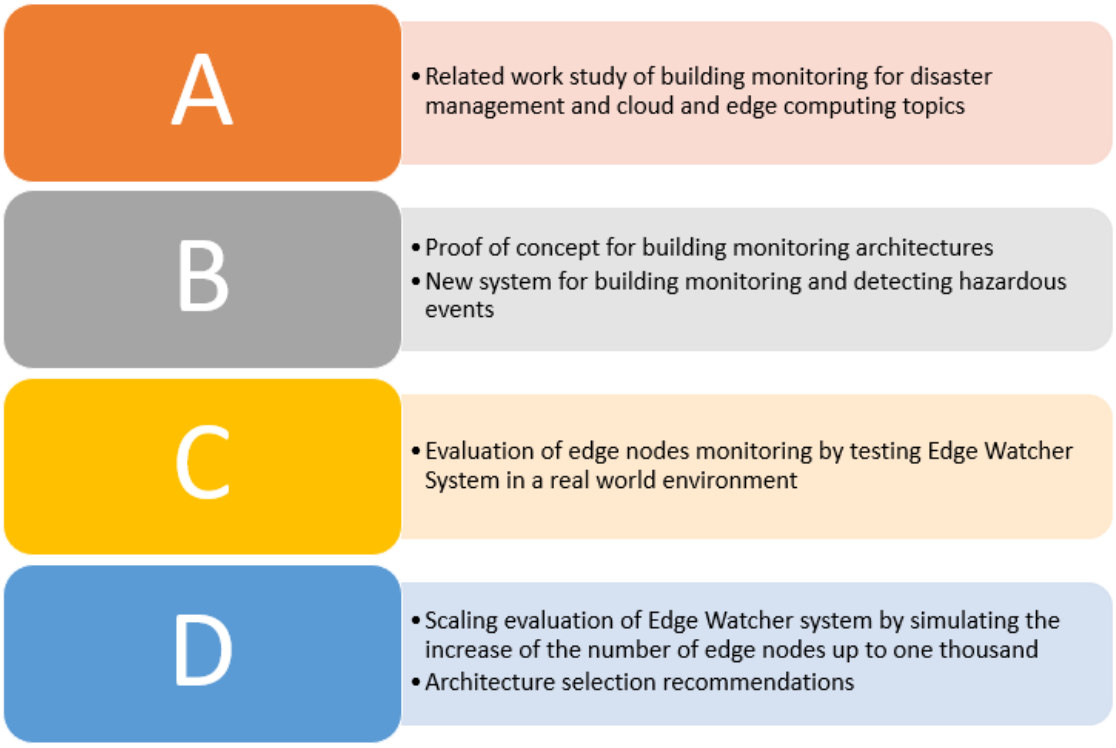

Fig. 3.1 Research method steps

4. Research Contributions for Building Monitoring in Cloud

4.1 Proof of Concept for a University Building

During my study, I first created a proof of concept for the system, and I conducted a comparison between multiple possibilities of implementations for EWS in a university environment, with the goal of detecting possible emergencies that may occur. Therefore, I made 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 system capable of monitoring the target building and sending notifications. The provided services are useful to individuals who can access this system and receive emergency notifications. Also, data gathered from sensors are also essential because they can represent events that can characterize hazardous situations. The next logical step for this kind of structure was the implementation of monitoring system that can send user reports. The goal of the system was to display the data from sensors and user reports, and display it in a dashbaord. The information received is shown in real-time. The systems consist of a sensor network that collects data, a monitoring and notification software, and an application that collects reports from its users.

The monitoring system provides information about the university building's environmental parameters; if one of these parameters comes out of a set threshold, it generates alerts. The

reporting application's goal 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.

4.2 Edge Watcher System (EWS)

Monitoring and automated hazard events detection are essential for every building with a high number of occupants, and it can be vital in detecting events that can affect the life of its inhabitants. Edge Watcher System is the solution that I propose for this building monitoring, with the capability to detect hazardous events and notify the responsible persons. There are many types of sensing devices that can be used in a building such as environment data collectors, video monitoring, equipment information gathering, and virtual sensors. Virtual sensors are represented by human reports that describe what happens when a particular event occurs. This type of input is essential because it comes from the building occupants, and it represents a very important input due to the fact that it shows the current state of the persons that are currently living or working in the monitored structure. Also, in the context of every monitoring system, the user input is the most important and it can have a critical impact on the possible intervention because it describes in detail the exact events that occur at a particular time in a building that can possibly affect the lives of its occupants. Besides the human and hardware input that serves as the collecting part of a monitoring system, an important part is the actual monitoring software that processes the data and based on the results sends alerts or displays the collected data in a structured form that can be read by the authorized personnel. In a software monitoring architecture, multiple services compose the solution such as services that run on nodes and gather data from the sensors along with the service that collects all the data from nodes. Also, along with the previously mentioned software component, another important part of the monitoring system is dedicated software that analyses the data and shows alerts, and a web dashboard that is accessed by the admins with the scope to display the processed data. The dashboard represents the interaction of the personnel with different data that come from the sensors. In the solution that I propose for this topic of software building monitoring, the dashboard is represented by a Web application and has two purposes: the main monitoring scope and the datastore configuration for the application. Therefore, the administrators of the system have the possibility of adding multiple buildings to monitor along with all the necessary components that will be queried for data such as nodes and sensors. Also, the application provides the configuration of structural characteristics regarding the location of each sensing device such as the floor where each node is located. The monitoring data has a dedicated page in the form of a dashboard where cards show statistics regarding the building such as the number of floors, nodes, sensors, and registered users.

4.2.1 EWS logical architecture

Edge Watcher System architecture is composed of multiple partitions. Therefore in Fig. 4.1, the EWS architecture is shown by underlining the three partitions that compose the solution: Hazardous Events Notifications, Cloud, Edge Network.

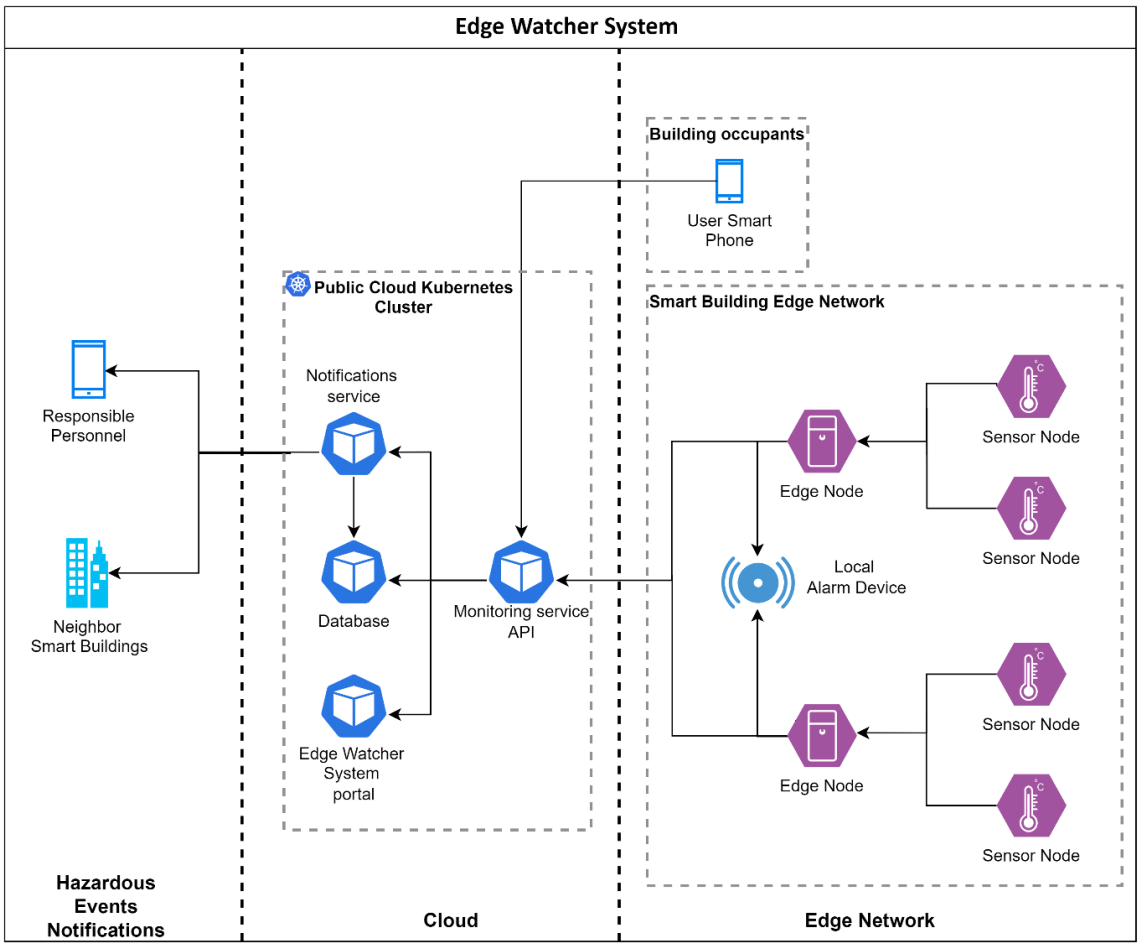

Fig. 4.1. EWS logical architecture

The cloud partition is represented by a microservices container implementation of the software monitoring system that processes the data that comes from the edge network. Multiple services compose this system such as the monitoring service that gathers the data from sensors along with the user reports. EWS centralizes the data from the other services, and it is very flexible and configurable to add multiple buildings, nodes, sensors, and users. It can be adaptable to add any building and integrate with different monitoring services and devices. The database represents the persistence part of the monitoring system where all configuration and data are added to be interrogated by the other services, especially by the EWS. Also, the cloud partition is represented by the Edge Watcher System software components deployed on a Kubernetes cluster located in a public cloud, such as Amazon Web Services, Microsoft Azure, or IBM Cloud. The software architecture is represented by loosely coupled services implemented on Kubernetes pods [5]. There is a pod for each of the services presented in Fig. 4.1. Regarding the data flow between the components, two scenarios can be described, which are representative of the two main functionalities that this application offers:

(i) **Scenario 1** - data gathering from the monitored building and alerting,

(ii) **Scenario 2** - configuration of the edge nodes within EWS.

The EWS software has four main components:

(1) a Web frontend written using the Angular framework
(2) a Node.js backend (Monitoring service API)

(3) a MySQL database to retain the user settings, configuration sensor, and human input
(4) the Notification service.

All of these are implemented on containers, and they run on Kubernetes. As a result, there is a dedicated pod for each of these components. This application is flexible and can run in every cloud with minimal modifications.

4.2.2 Hazardous events detection algorithm used in testing

The hazardous events detection algorithm was used to verify whether the collected sensor values exceeded a pre-defined (configurable) threshold. For the performance testing of EWS, the performance of the algorithm will be also taking into consideration using a Node.js dedicated library that measures the run time.

The purpose of the algorithm, located in the containerized environment, was to filter the data received from sensors in order to detect a possible emergency (Fig. 4.2) and then to send alerts to the responsible personnel. Hence, the environmental data were compared to a pre-defined threshold that was set up during the configuration of the EWS for each building/complex of buildings it was applied to. The critical values were stored in the database, and alerts were sent to the responsible personnel based on these values.

```
1   INPUT sensorId, sensorValue from edge node
2   CALL readDatabase with sensorId RETURNING criticalThreshold, location
3   IF sensorValue >= criticalThreshold THEN
4       CALL writeDatabase with sensorValue
5       CALL initiateAlert with location, sensorValue
6   END IF
```

Fig. 4.2 Algorithm for hazardous events detection.

For the algorithm, the metric employed was the average execution time, which was computed on the CSV file generated using the execution-time Node.js library. For each of the performance tests, there are two cases presented, corresponding to the two architectural options.

4.2.3 Hazard event detection and notification

Edge Watcher System provides the capability to send emergency notifications when a possible critical event is detected. The notification system is implemented within the Edge Watcher System's main detection method where data is received from the target monitored building. Therefore, when a critical event is detected – the received sensor value equals or exceeds the one that is set as a threshold, an emergency alert is sent to the authorized personnel (Fig.4.3). The system is implemented in such a manner that the sensor from where the emergency was detected is temporarily saved on the server, so if a new alert is detected by the sensor in the configured time (e.g., 5 minutes) a new alert will not be sent because it is considered related to the alert that was already sent. This notification system is essential for a building monitoring system since it detects possible critical data that comes from the sensors alerting the building's authorized personnel.

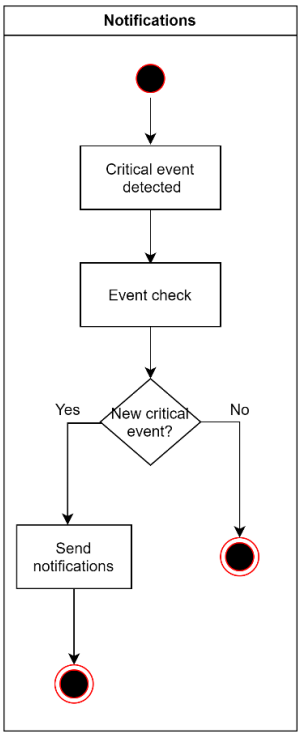

Fig. 4.3 Emergency notifications

4.3 Design Options Evaluated

4.3.1. Architectural options

The EWS services are based on containerization to automate application deployment and allow an easy configuration for various smart building models. Hence, regarding the location of the building manager services, there are two deployment possibilities: within a public cloud, or within a local building data center.

4.3.1.1. Architectural Option A — Public Cloud Kubernetes Cluster

The first analyzed solution locates the EWS service on a Kubernetes cluster deployed in a public cloud. The data are collected by the cloud monitoring system directly from the sensing devices distributed throughout the building with the aim of detecting emergency situations and notifying the responsible personnel (see Fig. 4.4). The advantage of deploying the Kubernetes cluster remotely is related to the principle of separating the monitoring system from the monitored target building. In this case, the building monitoring system would not be affected by the different outages that can appear when an emergency occurs. Nonetheless, there are other important aspects, such as maintenance costs and the initial hardware acquisition costs, which are zero with this public cloud option. There are, indeed, usage costs that, in the end, are lower compared to those required for the acquisition and maintenance of a small data center, including the personnel involved in these operations. However, a disadvantage that applies to a system using this approach is the dependence of the monitoring system on a reliable Internet connection for sending data to the cloud. This issue can be easily mitigated by providing backup connectivity in case the main Internet connectivity is not available by coupling the system with mobile Internet connectivity, which should be present on each edge node/smart sensor node, to independently send data to the monitoring system if other Internet options are not available.

In conclusion, the main advantage that supports this design option is that the monitor system does not depend on the building resources to function, and the costs to maintain a local data

center are removed. As a disadvantage, the cloud datacenter is located further from the building and the requests from the sensing devices take longer to be fulfilled

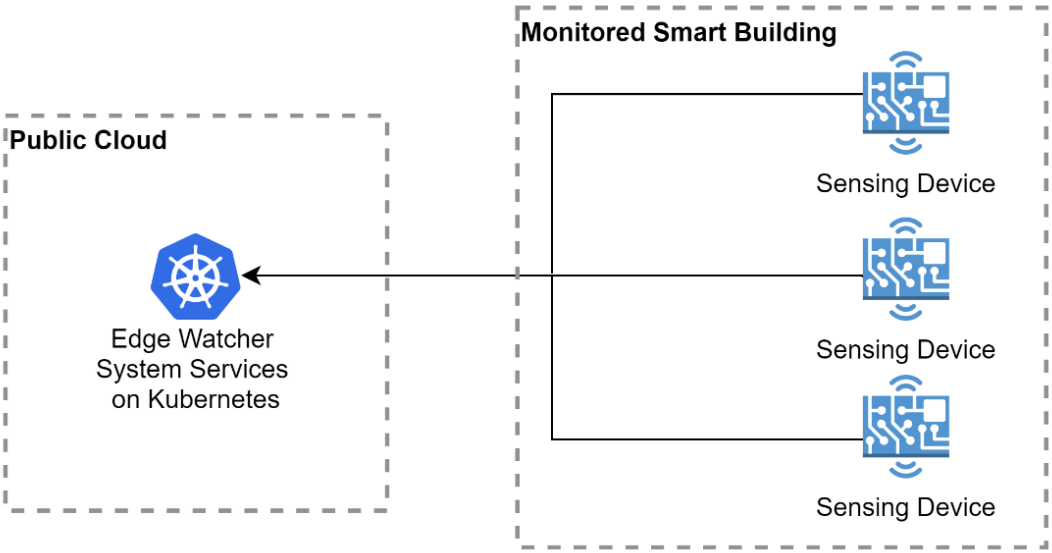

Fig.4.4. The EWS with a public cloud Kubernetes cluster

4.3.1.2. Architectural Option B — Local datacenter Kubernetes cluster

In the second solution, the local data center design implies that the monitoring system is installed on the hardware located inside the building (Fig. 4.5). The advantage of this approach is faster communication between the sensors and the monitoring system. Communication in the local network is faster than in the one that operates via the Internet. This is coupled with the fact that the system does not depend on having a reliable Internet connection to send environmental data to the monitoring system. The big drawback that comes with the implementation of this solution is the dependence of the monitoring system on the building's electrical grid. When there is a problem with the electrical system, the monitoring system cannot be kept online. This issue is only applicable to the data center's hardware. The sensing devices consisting of edge nodes and sensors are composed of low-power devices that can function on a battery for a very long time, providing the necessary data from the building environment.

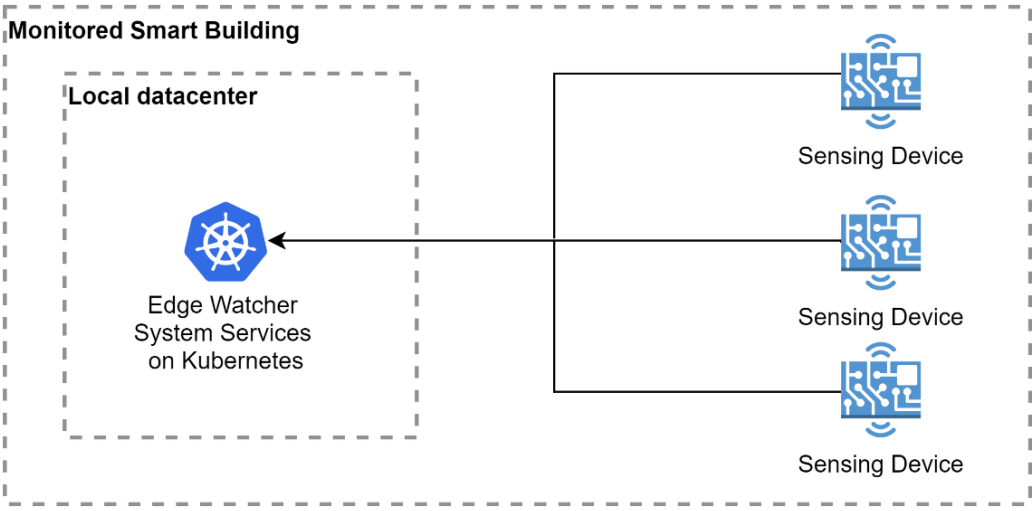

Fig. 4.5. EWS with a local datacenter Kubernetes cluster

Compared to the location design of the first method, with this approach, the main advantage is the lower latency period in the transmission of data between the local edge network and the local cluster. The main disadvantage is that if a critical event occurs, the monitoring system can be also affected since it is located in the same facility

4.3.2 Sensing devices options

The previous section analyzed design options based on the location of the monitoring system, i.e., public cloud and local data center deployments. Other important aspects regarding the edge topology, the options for the sensing devices, and how these devices are connected to the cloud and used to collect data for the monitoring system are discussed subsequently.

4.3.2.1. Edge Option A — Edge nodes.

The first design option considered for the sensing devices is based on an architecture composed of microprocessor-based edge nodes that gather data from wireless, low-power, microcontroller-based sensor nodes (Fig. 4.6). The purpose of the edge nodes is to gather data from multiple sensing devices and send them to the cloud monitoring system. This approach can be installed in any building to monitor environmental parameters. The edge nodes are connected to the Internet and can also function on a battery for shorter time frames compared with the sensor devices.

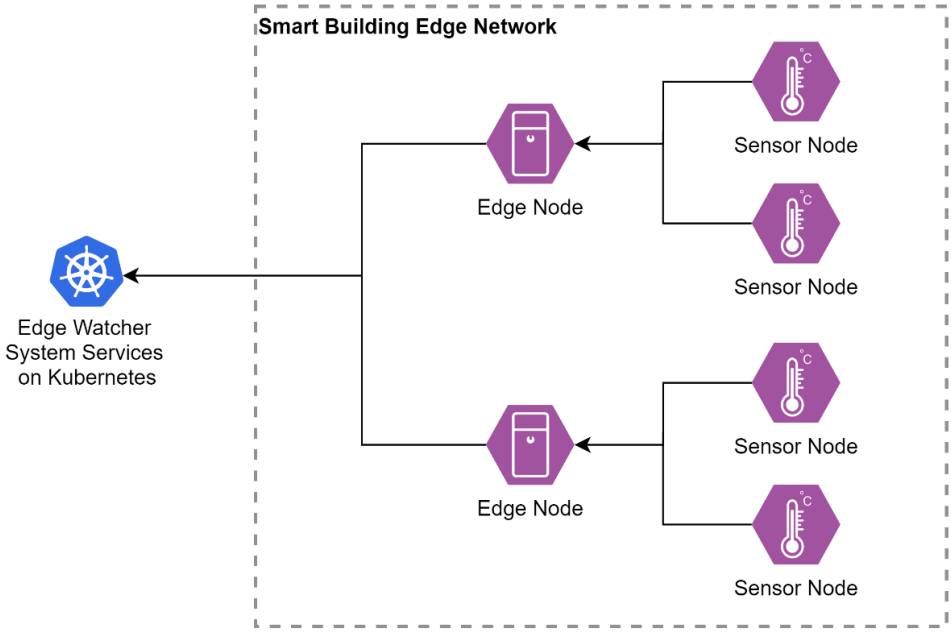

Fig. 4.6. EWS design with edge nodes.

4.3.2.2. Edge Option B — Sensing edge devices.

The second design option taken into consideration for the implementation of the sensing devices was to connect sensing edge devices directly to the cloud (Fig. 4.7). This system is similar to the edge node presented earlier, but the sensors are connected through a physical connection to the node. The sensing edge devices are based on microprocessors, and the number of sensors would be close to that of the microcontroller- based sensor nodes from the previous design choice. As microprocessors are more expensive than microcontrollers and consume more power, this approach would be a lot more expensive than the previous one while providing the same functionality in this use case.

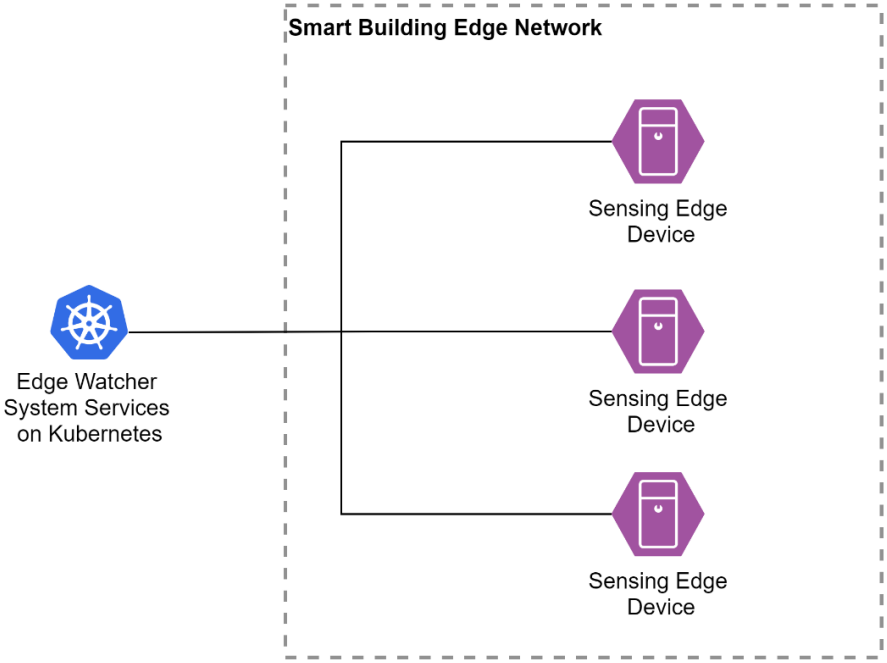

Fig. 4.7. EWS design with sensing edge devices.

5. Evaluation and Validation for a University Building

This chapter presents the validation experiments for the Edge Watcher System. The verification and validation are very important because the system is tested with different scenarios and offers the best view about how it works and behaves in the building environment. This chapter has two main parts: the experiment realized for a one-node EWS implementation and a real-world scenario experiment where EWS was tested within an University building.

5.1 Building Monitoring and Hazardous Event Detection Experiment

5.1.1 Experiment overview

Regarding the sensing devices options (), this chapter describes the experiments performed for the two alternatives presented in 4.3.1 : Edge Nodes and Sensing Edge devices. In order to do this, I created two different tests.

One node EWS is based on the implementation of a single edge node that will gather data from sensors or from a microcontroller-based device such as the Node MCU. Below there are two options presented.

5.1.2 Edge Option A. Edge nodes implementation

This architecture option was described in more detail in Chapter 4, and it involves the integration of two devices, an edge node and a microcontroller-based sensor node. Therefore, for this experiment I implemented the following system that was composed of:

- Raspberry Pi 3 as an edge node (MQTT Broker and Subscriber)
- NodeMCU sensor node and MQTT publisher
- BMP 180 sensor – used as an example to obtain data from the environment
- Local Kubernetes cluster deployed on the same virtual network

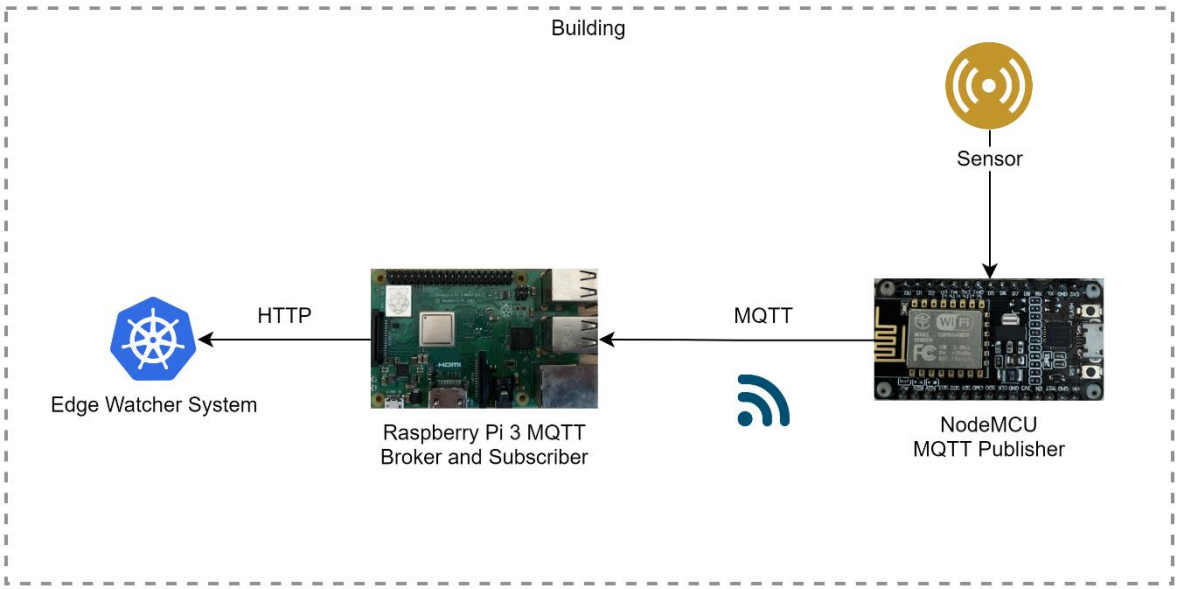

Fig. 5.1 Raspberry Pi Edge node and Node MCU Sensor Node

The scenario of this experiment is the following: The NodeMCU microcontroller-based node gathers data from the BMP 180 temperature sensor. After this, the data is published using MQTT to the Raspberry Pi 3 edge node. The role of this edge node is as MQTT broker and subscriber. Therefore, the data is sent to this node from one or multiple NodeMCU boards. This edge node also acts as an MQTT subscriber. Therefore, it also subscribes to the topics where the NodeMCU publisher is sending the sensor data as a JSON message. In the end, the data is analyzed on the Edge Node and is sent via an HTTP POST call to the Edge Watcher System. The message that comes from the sensor nodes contains both the detected value and the sensor id. This way the Edge Watcher System will know the identity and location of the sensor in order to display the values on the Dashboard or alert the responsible personnel if the detected value is critical.

5.1.2.3 EWS performance evaluation for Edge Option A

For the performance analysis test, I measured the response time and the hazard detection algorithm run time for 50 calls from the Edge Node to EWS. Each call was made every 5 seconds when the data was received from the NodeMCU sensor node. In the table below, there is the result of the tests. Therefore, for the 50 calls performed, the response time was between 27 and 54 ms with an average response of 32.25 ms. The decision algorithm run time is 11.27 ms.

Table 5.1 Option A performance analysis

Edge Nodes	**Calls**	**Interval between calls (s)**	**Average (ms)**	**Min**	**Max**	**Median**	**95th percentile**	**Run Time (ms)**
1	50	5	32.25404	27.24	53.801	30.924	37.52415	11.27152

5.1.3 Edge Option B. Sensing edge device implementation

This architectural option implies that a sensor is physically connected to the edge node Fig. 5.2. Therefore, for this experiment I implemented the following system that was composed of:

- Raspberry Pi 3 as a sensing edge device

- BMP 180 sensor – used as an example to obtain data from the environment
- Local Kubernetes cluster deployed on the same virtual network

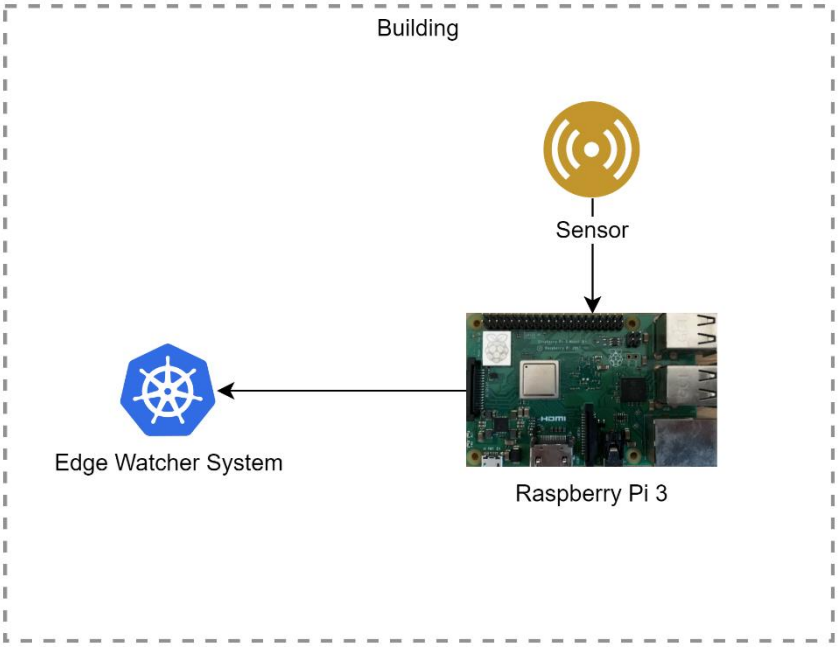

Fig. 5.2 Raspberry PI sensor node diagram

The scenario of this experiment is the following: The Raspberry Pi 3 sensing edge device will collect readings from the BMP 180 temperature sensor. At the level of the Edge Watcher System, the sensor edge device is configured in order to register its location, critical threshold. The Edge Watcher System will register only the values that are passing the configured threshold.

5.1.3.1 EWS performance evaluation for Edge Option B

For the testing of this scenario, the sensor is directly connected to the edge device. Therefore, 50 calls were executed from it to the EWS. Therefore, on the EWS, the recorded values are compared with the selected threshold from the database. If the value exceeds the threshold, the data is added to the database. The average response time is 37,93 ms, and the algorithm run time is approximately 12 ms. These recorded values show a good performance for the current load (Table 5.2).

Table 5.2 Option B performance analysis

Edge Nodes	**Calls**	**Interval between calls (s)**	**Average (ms)**	**Min**	**Max**	**Median**	**95th percentile**	**Run Time (ms)**
1	50	5	37.93	24.6	155.2	30.06	91.06	12.10

5.1.4 Discussion

For this one-node EWS, I demonstrate a base real-world experiment of the Edge Watcher System. The main theme of this chapter was the performance testing of the system with data that comes from a physical edge network. For the tests conducted in this paper, the local deployment of the Edge Watcher System was receiving calls from two configurations of the

edge network: Edge Option A and B. In the case of Edge Option A, the architecture is composed of an edge device and a sensor node. The data is gathered by the sensor node and published via MQTT to the broker (edge device). The edge device is both an MQTT broker and subscriber. When a new message is published to the topic, the subscriber sends the recorded value to the EWS via HTTP. The EWS compares the value with a pre-defined threshold and sends an alert if necessary. For Edge Option B, the architecture is composed of a sensing edge device that has a physical connection to a sensor. After the data is gathered, it is sent to EWS where it is processed in the same manner as it was described before.

As it was seen from the experiment results, the performance results in the case of a one-node implementation, was very similar for the two options, both recording an average response time of under 40 ms. The advantage of Edge Option B is represented only by the implementation simplicity, and it is recommended for a small apartment where a limited number of sensors is required. Edge Option A is more complex to implement but it is cheaper compared to Edge Option B, where multiple sensors will be needed.

5.2 Performance Testing for Multiple Design Options

Performance testing in a real-world environment is essential to validate that a system functions at the required parameters. Therefore, multiple scenarios must be implemented that should test the functioning of a system. For my thesis, I conducted multiple testing scenarios within University Politehnica of Bucharest “Precis” building.

5.2.1. Testing scenarios

The tests consisted mainly in implementing the physical edge network within this building and creating multiple configurations for testing. Therefore, the test includes the settings for the edge network and also for the location of the Edge Watcher System, which was installed on a machine at the building level and also in IBM Cloud. Another set of scenarios included the type of edge device that was used to form the edge network. The role of the edge network is to gather environmental data from the building and send it to the monitoring system. Within the Edge Watcher System, as soon as the data is received from the building edge network it is compared against a threshold value set prior by the administrator. For this case, there are two scenarios that were covered within the tests.

The first scenario is about the implementation of the hazardous event detection algorithm on the edge node. In this case, the microprocessor-powered Raspberry Pi, when it receives the data from the sensor it compares it to the threshold value that was prior set by an administrator.

The second scenario that was tested is regarding the number of edge nodes that were installed within the target area. As a consequence, I conducted tests with both one edge node and two edge node configurations. For the two-edge node test, two Raspberry Pi devices were used to send the sensor data to the EWS. Also, on the edge network topic, another type of scenario that was conducted was implementing two architecture choices: one in which the sensor was directly connected to the edge node through a physical connection and another one in which a new type of edge device was introduced: Node MCU. In the case of a large building such as University Politehnica of Bucharest “Precis” building the approach with microcontroller-based devices may be better suited since it costs less to implement.

5.2.2 EWS testing

The objective of my experiment is to test the Edge Watcher System building monitoring application against multiple edge device configurations. These devices would be installed then in a university building to provide the necessary data for detecting possible hazardous events.

For the planned test I identified multiple architectures configurations that can be integrated into a building environment and based on the Edge Option A and B and Architectural Option A and B.

5.2.2.1. Design options

The architectures that I propose for this performance test are classified into two categories that will be presented in detail below.

1. Design options for scenario 1

The most important characteristic of this architecture option is represented by a sensor that is physically connected to an edge node (Edge Option B). Another classification would be related to the location of the monitoring system which can be deployed locally (Design Option 1.1 - Fig.5.3) or in the cloud (Design Option 1.2 - Fig. 5.4) based on Architectural Option B and A.

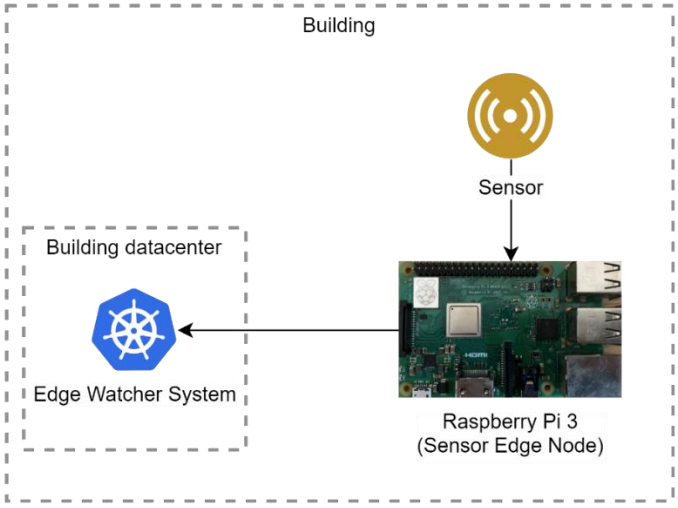

Fig. 5.3 Design 1.1 Sensor Edge Node, locally-deployed EWS

In the image below is represented the cloud deployed EWS which receives the building environment data from the Sensor Edge Node. The data is sent from the sensor edge node, which in our tests is represented by a Raspberry Pi 3 board, is sent via HTTP to the monitoring system.

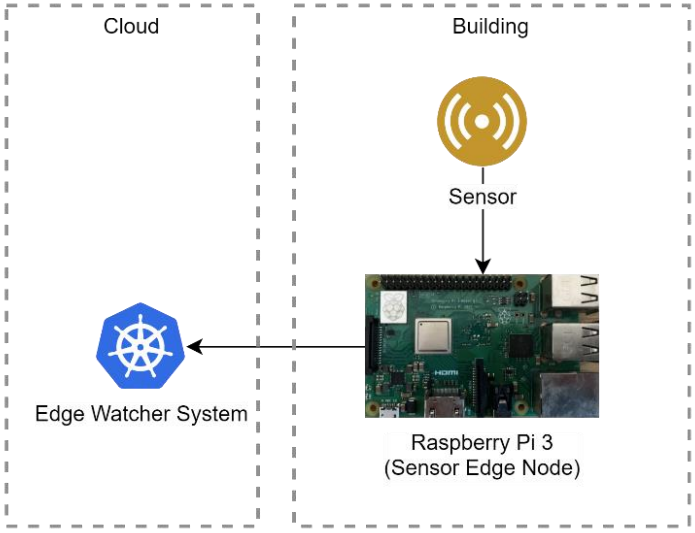

Fig. 5.4 Design Option 1.2 - Sensor Edge Node, cloud-deployed EWS

2. Design options for scenario 2

The main difference that comes up is the introduction of a new hardware device within the edge network – a dedicated sensor node (Edge Option A). This sensor node which is represented by a microcontroller-powered board - Node MCU collects the data from the sensors and publishes it via MQTT to the Raspberry Pi 3 edge node, which acts as both Broker and subscriber. After the data is received on the edge node, it is then sent to the monitoring system via HTTP.

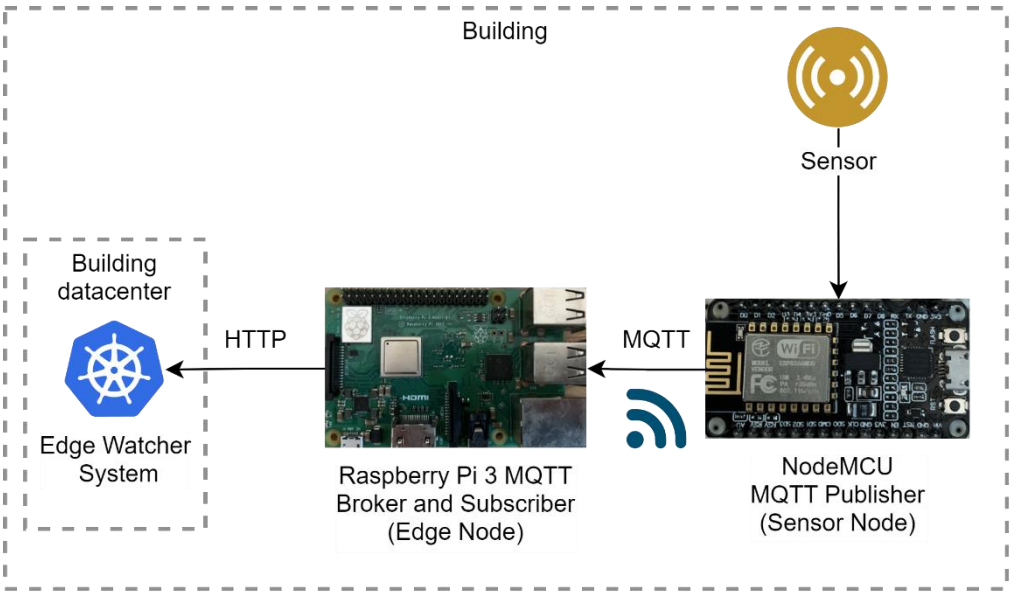

Fig. 5.5 Design Option 2.1 - Edge Node MQTT broker with Sensor Node, locally-deployed EWS

As with Design Option 1, there are two configurations available – locally deployed (Fig. 5.5) and cloud-deployed (Fig. 5.6) monitoring system.

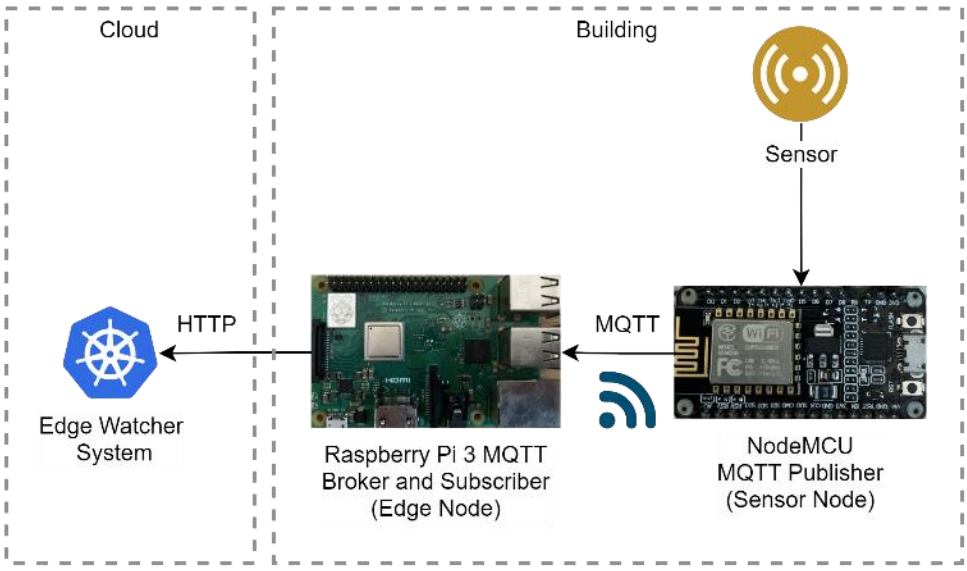

Fig. 5.6 Design Option 2.2 - Edge Node MQTT broker with Sensor Node, cloud-deployed EWS

3. List of tested design options

In total sixteen test cases were defined, starting from the design options presented above. The list is as follows:

Design Option 1

- Design Option 1.1
  1.1.1. One Sensor Edge Node, one sensor, hazardous event detection algorithm on Edge Node
  1.1.2. One Sensor Edge Node, one sensor, hazardous event detection algorithm on EWS
  1.1.3. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on EWS
  1.1.4. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on Edge Node
- Design Option 1.2
  1.2.1. One Sensor Edge Node, one sensor, hazardous event detection algorithm on Edge Node
  1.2.2. One Sensor Edge Node, one sensor, hazardous event detection algorithm on EWS
  1.2.3. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on EWS
  1.2.4. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on Edge Node

Design Option 2

- Design Option 2.1
  2.1.1. One Edge Node, one sensor node, one sensor, hazardous event detection algorithm on Edge Node
  2.1.2. One Edge Node, one sensor node, one sensor, hazardous event detection algorithm on EWS
  2.1.3. Two Edge Nodes, one sensor node, one sensor per node, hazardous event detection algorithm on EWS
  2.1.4. Two Edge Nodes, one sensor node, one sensor per node, hazardous event detection algorithm on Edge Node
- Design Option 2.2
  2.2.1. One Edge Node, one sensor node, one sensor per node, hazardous event detection algorithm on Edge Node
  2.2.2. One Edge Node, one sensor node, one sensor per node, hazardous event detection algorithm on EWS
  2.2.3. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on EWS
  2.2.4. Two Sensor Edge Node, one sensor per node, hazardous event detection algorithm on Edge Node

5.2.2.2. Implementation details

The tests that I employed were conducted in the University "Politehnica" of Bucharest "PRECIS" building where I implemented the four architectures presented in the previous chapters. Within the tests, multiple scenarios were defined that include also the variations of the number of edge nodes. Besides these the emergency detection algorithm, a simple algorithm used to detect if there is an emergency was either implemented on the node or in the cloud, based on the scenario.

Hardware used:

1. Design Option 1

- Sensor Edge Node: Raspberry Pi 3
- Sensor: BMP 180 temperature sensor
- Local cluster: Kubernetes Cluster placed in the building local network
- Cloud Cluster: IBM Cloud Kubernetes Cluster

2. Design Option 2:

- Edge Node: Raspberry Pi 3
- Sensor Node: Node MCU
- Sensor: BMP 180 temperature sensor
- Local cluster: Kubernetes Cluster placed in the building local network
- Cloud Cluster: IBM Cloud Kubernetes Cluster

5.2.2.3 Testing procedure

The purpose of the performance tests is to demonstrate the ability of the Edge Watcher System to function in a real-world scenario. Therefore, in order to test the performance of EWS, I evaluated the HTTP request calls that were made against it by different edge network configuration possibilities. Therefore, on the edge nodes, the response time from the HTTP call to insert data to EWS was measured. Usually, the HTTP response time is influenced by the location of the server, load, and processing that is done with each call. Regarding the processing, in the testing scenarios I included the possibility where the code that implements the small decision algorithm was run on the server and other scenarios where it ran on the edge node.

5.2.3 Results

The results were obtained for each of the design options presented. Therefore, in the table below I present the average response time for the calls executed from the edge nodes to the Edge Watcher System. For the scenarios presented in the previous chapter, there are ones where two edge nodes were utilized. Therefore, in the table below the average response time, E2 (Edge Node 2) will appear only for the scenarios where two edge nodes were deployed. For each scenario 50 calls were made from the edge node to EWS to send the environment data that was collected from the UPB "PRECIS" Building. The EWS is deployed in two locations, IBM Cloud (Milan location) and on the building network, therefore the results will differ also based on the location of the monitoring system. The last column shows where the decision algorithm runs – on the edge node or within EWS. The hazardous event detection algorithm compares the critical value that was set by the building administrator during the initial sensor configuration with the actual value that was gathered from the environment. If the value is critical, then the data is inserted in the EWS database, and it also sends a notification to the building's authorized personnel.

Table 5.3 Results

Design Options	**Average Response time E1 (ms)**	**Average Response time E2 (ms)**	**EWS Location**	**Hazardous event detection algorithm**
1.1.1	64.21662	-	Building	Edge Node
1.1.2	67.37904	-	Building	EWS

1.1.3	78.00704	92.02102	Building	EWS
1.1.4	76.4661	89.14806	Building	Edge Node
1.2.1	196.2808	-	IBM Cloud	Edge Node
1.2.2	207.4164	-	IBM Cloud	EWS
1.2.3	259.2715	222.0848	IBM Cloud	EWS
1.2.4	242.1902	222.6137	IBM Cloud	Edge Node
2.1.1	60.96322	-	Building	Edge Node
2.1.2	85.6524	-	Building	EWS
2.1.3	69.35771	92.97539	Building	EWS
2.1.4	65.2284	80.05818	Building	Edge Node
2.2.1	210.3509	-	IBM Cloud	Edge Node
2.2.2	213.7616	-	IBM Cloud	EWS
2.2.3	196.2808	222.6137	IBM Cloud	EWS
2.2.4	214.7974	220.3171	IBM Cloud	Edge Node

Edge Node 2 has fewer values since was tested only in the two edge nodes scenarios. What is common to both nodes is that they recorded values under 300 ms, which indicates adequate performance for a monitoring system.

6. Scaling Evaluation and Lessons Learned

6.1 Background

Evaluating performance is very important in edge and cloud computing and even more so for the use of these systems in hazard management. It requires multiple tests of "responsiveness, reliability, throughput, interoperability, and scalability" under a given workload [6]. Haseeb-Ur-Rehman et al. developed a sensor cloud taxonomy that covers network, communication, data management, architecture, heterogeneity, and security aspects [7].

6.2 Scaling Evaluation

Performance testing is an essential part in the development of an application because it provides the means to find if the tested system is running on the desired parameters with different inputs. The importance of this critical step relies mostly on the idea that every new development piece must be tested on different pre-defined scenarios where different metrics are monitored. In the case of my study, which has as objective the research on cloud building monitoring systems, there are multiple capabilities of this system that has to be tested and validated against a predefined test plan. Some of these capabilities that will be further detailed further are related to the performance monitoring of this solution. An objective that I propose for this part is to simulate various edge node configurations that are performing different HTTP requests to the cloud application. Some of these requests are represented by the data that is sent to EWS to be further analyzed. Another important aspect of this test is that with this setup, the actual building edge node configurations are simulated, larger buildings require multiple edge nodes in order to cover a larger area. The test will be performed with scenarios for small apartments, a house, small residential building, office buildings, and a complex of buildings along with a detailed comparison of the reports generated for all these cases. Another aspect that will be taken into consideration with the previously mentioned tests is the comparison between the cloud hosted

solution and a local cluster. Therefore, the same tests will be simulated for the application being locally deployed and the cloud deployment on a container-based service such as a Kubernetes cluster. This comparison will show in what measure the cloud implementation will affect the response times for the application compared to the same solution deployed on the same network with the edge nodes. The next test that I propose for this solution is the performance evaluation for the hazardous events detection algorithms that are used to detect if the collected data is critical or not. For this test, a very essential part that should be mentioned is the importance of the cloud deployment and how fast the algorithms would run on a cloud-based container solution compared to the same implementation deployed on a local container cluster. Another factor that is very important for this test is the performance of the decision algorithm against a high number of requests represented by a larger edge node network located in a building complex.

The results of the tests that were presented above represent also an important part of the validation process that will attest that the researched cloud solution achieves some performance metrics that are critical for the optimal functioning of such a system.

6.2.1 Performance evaluation of the containerized architecture options

Based on the design options presented in Chapter 4.3, I hereby present the method used to determine the performance of the system with multiple simulated sensing devices on the edge. The tests compared the performances of the two architectural design options: (A) the public cloud Kubernetes cluster; and (B) the local datacenter Kubernetes cluster. These were both connected with sensing devices through edge nodes based on the first option from Chapter 4.3. The tests conducted involved load and stress testing. The goal of this comparison was to gain a detailed view of the performance requirements needed for the cloud system when data are received from multiple nodes. Each test conducted corresponded to a real-world configuration for a type of building, with specific needs regarding the number of edge nodes and installed sensors required.

6.2.2 Testing scenarios

For both architectural options, my concern was to identify testing scenarios based on configurations of real-world building examples, from a small apartment to an entire complex of buildings, such as a university or a corporate campus. Therefore, the performance tests were executed for EWS services hosted on containerized environments following the scenarios described below. I selected examples inspired by the occupancy classification and definitions given in The International Building Code [8] and from the ten classes of buildings established in [9].

- **Small Apartment**. This scenario refers to an individual unit in a residential building. I considered the setup for a small apartment with two rooms and one edge node. In this scenario, the edge node needs to collect environmental data from sensors to detect motion, temperature, and contact when the door opens or closes. For testing purposes, an API call is simulated from the edge node to the EWS, which results in the addition of a new reading in the database. The hazardous events detection algorithm compares each value to a predefined threshold to verify if an alert should be considered. The scenario is equally applicable for a shop in a shopping center if the sensor monitoring is done separately by the shop tenant.
- **House**. The second scenario targeted a standalone residential building. I considered the example of a detached house with five rooms and two floors. In this scenario, an edge node is installed on each floor to collect environmental data and send it to the EWS. The

difference from the first scenario is in the use of two edge nodes; therefore, it may also be appropriate for a housing unit in a group of attached dwellings.

- **Small residential building**. The small building scenario encapsulates a total of five simulated edge nodes installed on each floor of the building. Each node receives environmental data sent by sensors installed in the public space and inside the individual apartments situated on the same floor. The monitoring and the hazardous event notifications are managed for the entire building by the responsible personnel.
- **Office building**. In this scenario, I simulated the case of a building with 10 floors and 20 edge nodes. In this scenario, two edge nodes are installed on each floor in order to receive environmental data from the sensors and send them to the EWS for further processing. I considered this example of a non-residential building used for professional or commercial purposes because this type of building is more often used as a smart building; therefore, it can take advantage of services for hazardous events detection and alerting, such as those considered in our study.
- **A complex of buildings**. I considered a scenario at a larger scale, corresponding to a group of related smart buildings with residential, business, or institutional usage. They may correspond to a shopping center, a university, a corporate campus, or a residential complex. They may occupy a smaller or larger area, with multiple sensors used to measure environmental data connected to EWS through multiple edge nodes. I first considered three cases for performing load and stress testing:
    - 50 edge nodes—scattered around multiple buildings that comprise the complex
    - 100 edge nodes—to test the capacity to work under a high load by registering a high number of environmental data points sent within a short period of time
    - 1000 edge nodes—to test the limits of the cluster configuration and the capacity to simulate 1000 requests sent to the system without a ramp-up period; the requests are sent immediately to the server, and it has to address each request as soon as the previous one has been fulfilled.

Then, I also executed tests for edge node numbers between 100 and 1000 with steps of 100.

6.2.3 Performance test settings

For the EWS, performance testing was conducted using Apache JMeter version 5.4.1, in order to simulate the requests from the edge nodes and verify whether they were addressed without error and if the response time was low enough. The machine used to run the JMeter tests was powered by a 4-core Intel i7 CPU, coupled with 8 GB of RAM.

Test cases were developed for different sizes, starting from a small apartment, and finishing with a complex of buildings. The tool setup consisted of creating threads to simulate groups of edge nodes and executing requests against the EWS reporting API. Regarding the JMeter configuration, there were three important parameters to set up:

- The number of threads: the number of edge nodes used to send environmental data to the application
- Ramp-up period: the time that it would take to get to the full number of threads
- Loop count: the number of tests to be executed.

Another objective for the performance testing, besides the JMeter HTTP calls is the actual testing of the decision algorithm that verifies if the collected sensor value exceeds the pre-defined threshold. For the performance testing of the decision algorithm, I employed a Node.js

dedicated library that measures the run time of different functions. The library name is "execution-time" and can be installed from the node package manager (npm).

6.2.4 Containerized environment setup

To test the containerized architecture options for the EWS, I implemented two configurations, corresponding to the analysis presented in Chapter 4.3:

(A) The IBM Cloud Kubernetes cluster was implemented for the public cloud Kubernetes cluster design option with the following technical details:

- Deployment in IBM Cloud data centers
- Free one node Kubernetes cluster with two cores and 4GB of RAM (default free tier)
- Kubernetes version: 1.21.7 (default).

(B) The Docker Desktop local cluster was implemented for the local datacenter Kubernetes cluster design option with the following technical details:

- Deployed on a Windows machine with 4-core Intel i7 CPU, coupled with 8 GB of RAM-Windows Subsystem for Linux (WSL) 2
- Docker Desktop WSL 2 backend version 4.1.1 with Kubernetes
- Memory and CPU are allocated dynamically to improve resource consumption
- Kubernetes version 1.21.5 (default).

6.2.5 Performance metrics

The Apache JMeter makes it possible to output an HTML result that offers multiple parameters that are related to the response time of the tested request. The attributes that are relevant to my study are the execution errors, the response time, and the throughput (to determine the system's performance). They are associated with the following metrics:

- Error % (for the execution)
- Average response time
- Minimum response time
- Maximum response time
- Median response time
- Percentiles
- Transactions/s (for the throughput)
- The ratio of the failed requests, the Error %, should be 0 for a successful test.

6.3 Scaling Evaluation Results

In this section, the results of the scaling evaluation are presented and discussed. For each of the two containerized environment setups (i.e., the IBM Cloud Kubernetes cluster and the Docker Desktop local cluster), multiple performance tests were executed to simulate the relevant scenarios identified in Chapter 6.2. The monitored results correspond to the metrics obtained for the JMeter HTTP calls, simulating the information gathered from a number of edge nodes plus the run time of the decision algorithm for hazardous event detection, which was measured with a Node.js library, as explained in Chapter 4.2. Thus, the performance testing was executed both for the local and the public cloud cluster implementations. JMeter can also be used to test the performance of web services applications such as the ones from [10], [11], [12].

Thus, I generated 30 JMeter reports, each containing a multitude of graphics and data. To analyse them comparatively, several important results are summarized in Table 6.1 for the IBM Cloud Kubernetes cluster and in Table 6.2 for the Docker Desktop local cluster.

Table 6.1. Summary of the test results for the IBM Cloud Kubernetes cluster.

Test	**Executions**		**Response Time (ms)**					**Throughput**	**Hazardous Event Detection Algorithm**
Scenario	Samples (Edge Nodes)	Error (%)	Average	Min	Max	Median	95th Percentile	Transactions /s	Run time (ms)
Small apartment	1	0	389	389	389	389	389	2.57	3.52
House	2	0	375.5	371	380	375.5	380	5.26	4.42
Small residential building	5	0	380	371	385	382	385	12.89	3.49
Office building	20	0	405.75	374	437	405.5	436.8	43.8	21.8
A complex of buildings	50	0	472.84	391	560	461	553.45	81.04	167.43
	100	0	574	376	748	574	729	115.74	144.42
	200	0	556.83	375	815	563.50	789.00	203.87	278.9
	300	0	906.49	446	1336	947.50	1309.95	196.34	564.4
	400	0	981.00	380	1461	936.00	1412.85	236.27	554.4
	500	0	1393.26	420	2090	1394.50	2014.85	212.04	560.3
	600	0	1614.98	391	2282	1602.00	2225.95	223.46	846.57
	700	0	1584.66	374	2373	1620.00	2288.90	246.05	448.3
	800	0	1832.57	398	2759	1855.0	2651.95	250.16	677
	900	0	2026.60	499	3056	1994.0	2954.85	230.00	1036.5
	1000	0	1992	384	3538	1927	3359	189.9	939.89

Table 6.2. Summary of test results for the Docker Desktop local cluster.

Test	**Executions**		**Response Time (ms)**					**Throughput**	**Hazardous Event Detection Algorithm**
Scenario	Samples (Edge Nodes)	Error (%)	Average	Min	Max	Median	95th Percentile	Transactions /s	Run time (ms)
Small apartment	1	0	9	9	9	9	9	111.11	3.22
House	2	0	11	9	13	11	13	153.85	3.59
Small residential building	5	0	24	16	32	23	32	151.52	19.62
Office building	20	0	68.2	31	109	69.5	107.7	176.99	57.3
A complex of buildings	50	0	228.44	97	288	239	285.8	130.89	155.5
	100	0	322.33	78	460	311	447.9	175.75	288.86
	200	0	543.18	138	978.6	553.7	930.7	176.3	447.3
	300	0	768.05	147	1390	769.3	1322.1	179.2	574.5
	400	0	995.9	164	1821.6	1002.5	1741.1	179.92	762.5
	500	0	1316	181	2378.3	1318.3	2293.3	177.93	1030.9
	600	0	1597.8	360	2855.6	1515.6	2737.8	175.79	1211.2
	700	0	2296.3	426	3262.4	2094.1	3151.5	164.04	1881.5
	800	0	2589.2	363	3698.7	2515.5	3560.75	173.05	2232.5
	900	0	2849.8	147	3994.4	2842.3	3757.9	162.14	2588.6
	1000	0	3098	243	4431	2914	4284	171.59	2975.23

I also present the results in comparative graphics to show the difference in the average response time for the two architectural design options: the IBM Cloud Kubernetes cluster and the Docker Desktop local cluster. Fig. 6.1 shows the response times versus the number of edge nodes corresponding to the first four scenarios: one edge node for the small apartment, two edge nodes for the house, five edge nodes for the small residential building, and 20 edge nodes for the office building. Fig. 6.2 represents the response times for a complex of buildings versus the number of edge nodes, ranging from 50 to 1000. One can, thus, observe the influence of the container orchestration decentralization. Fig. 6.3 and Fig. 6.4 illustrate similar comparisons for the run time of the decision algorithm for hazardous event detection.

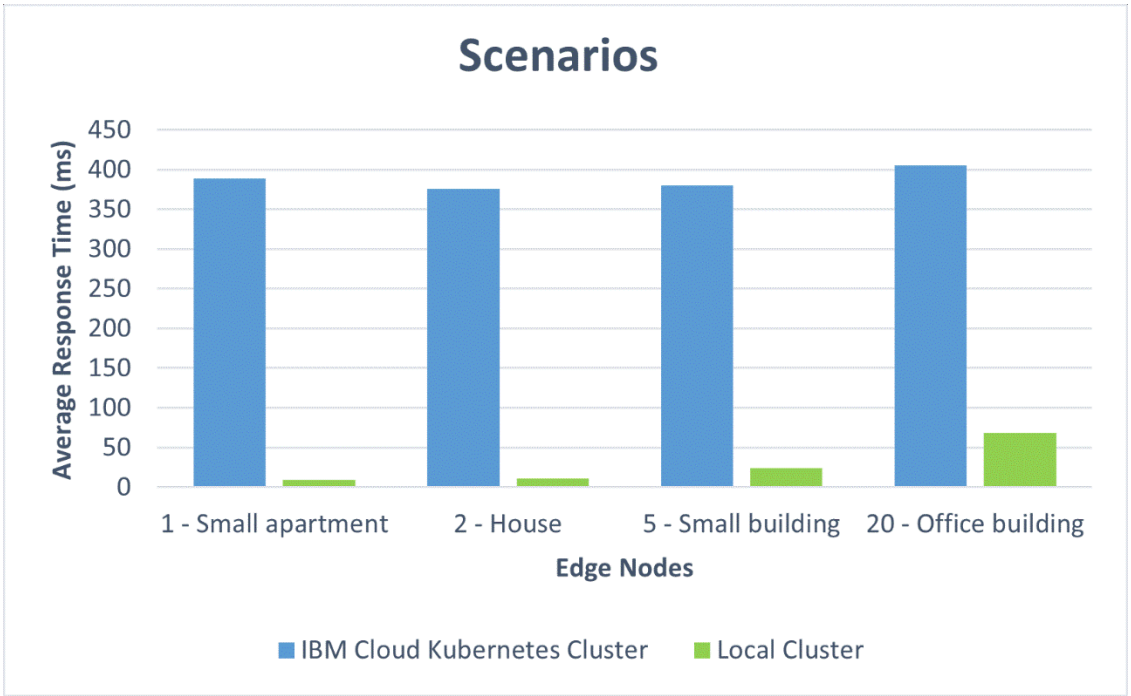

Fig.6.1 Comparative response times for different scenarios

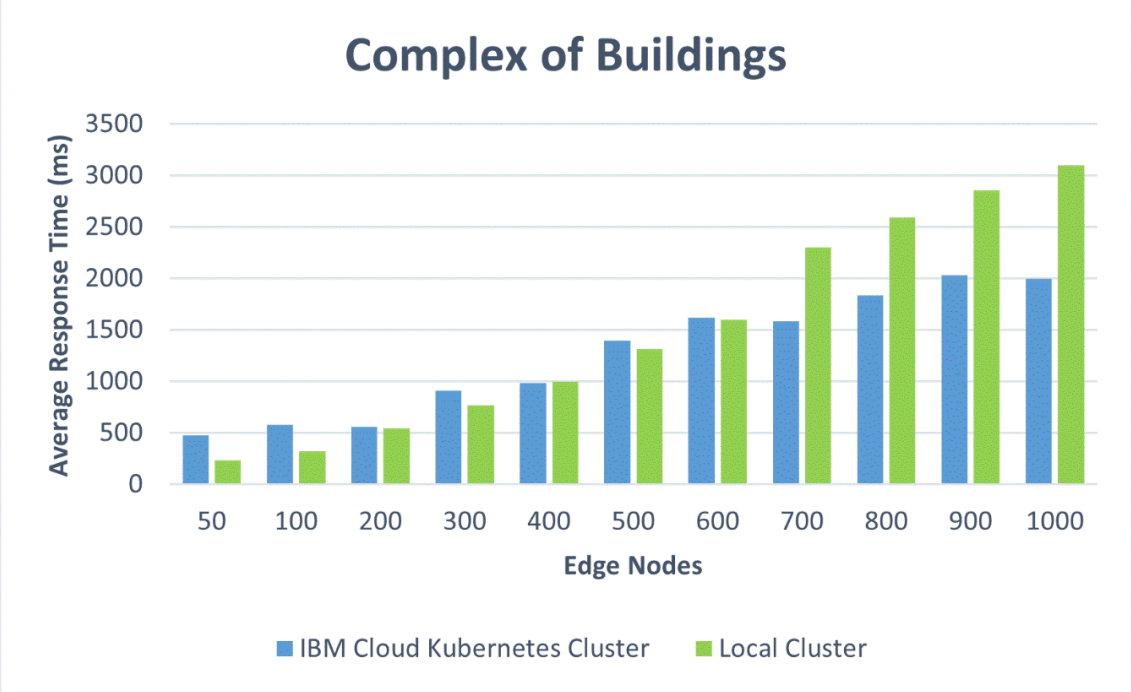

Fig. 6.2 Comparative response times for the complex of buildings.

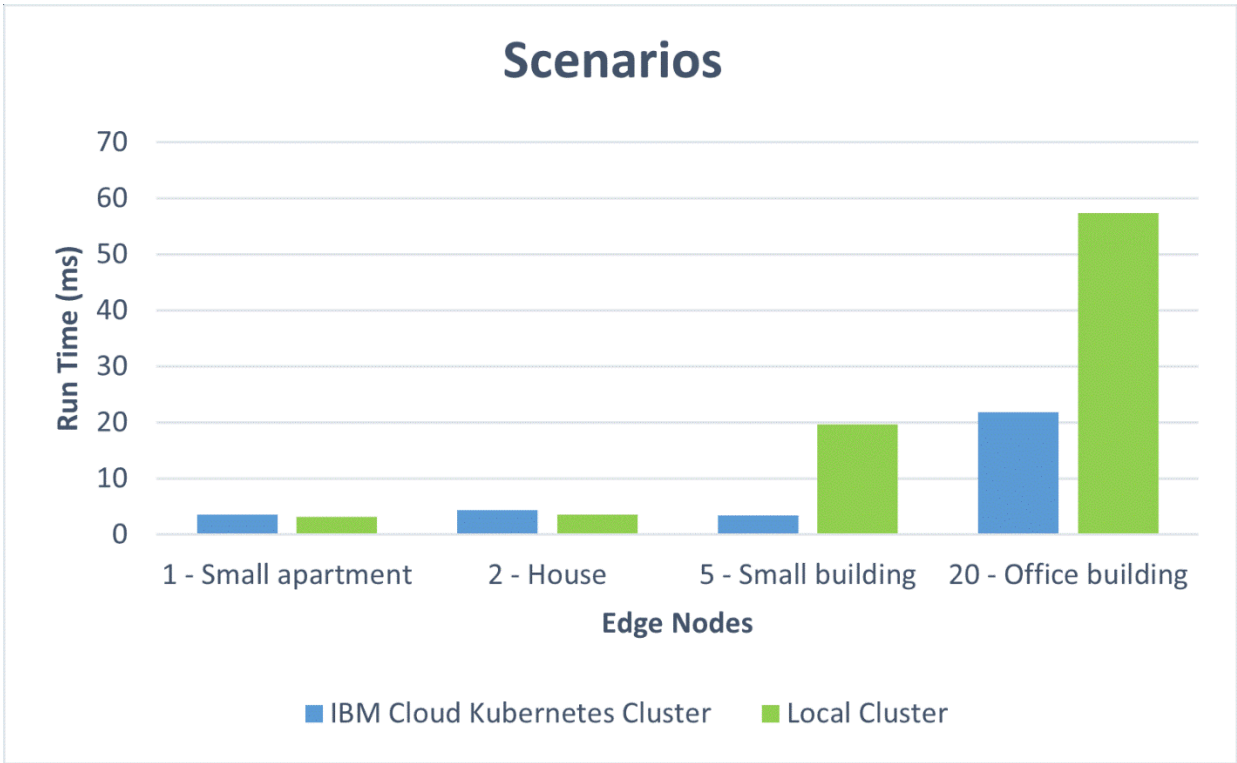

Fig. 6.3. Comparative run times for different scenarios.

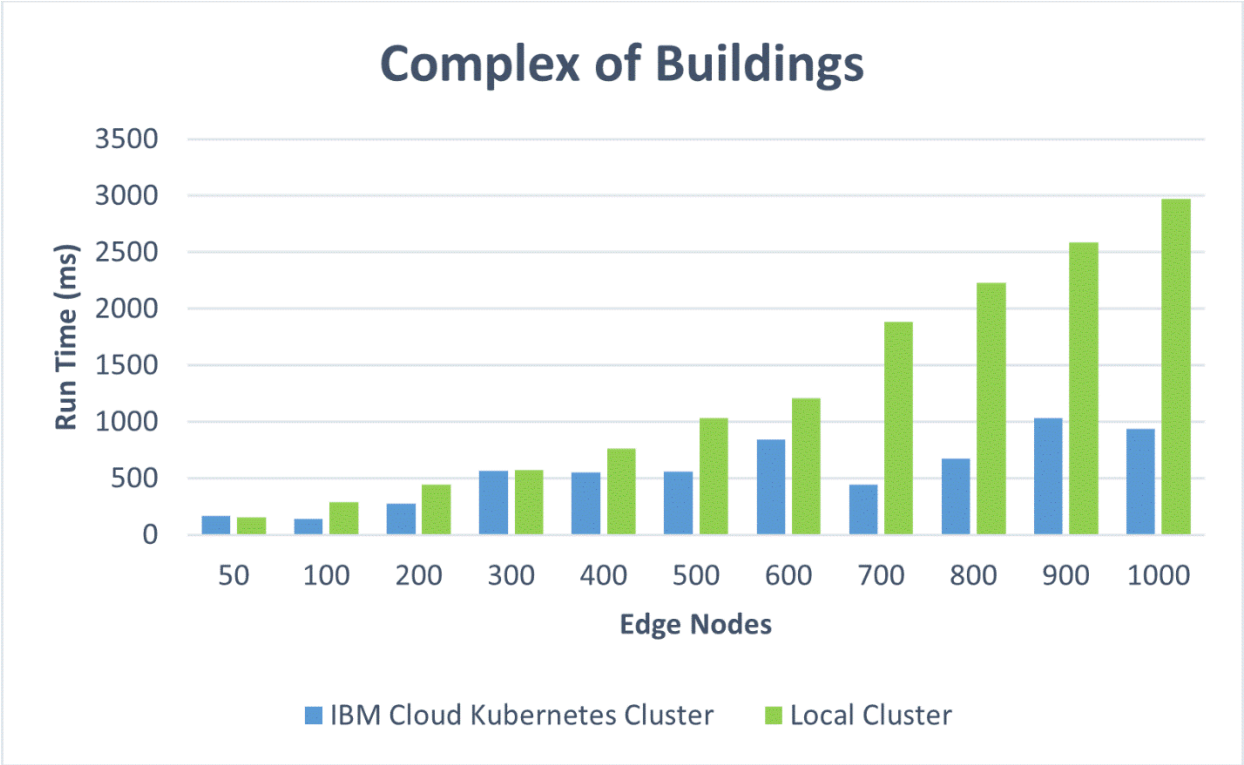

Fig. 6.4. Comparative run times for the complex of buildings.

I observed that the Docker Desktop local cluster worked faster for the first four scenarios with up to 20 edge nodes as a result of its location. However, the IBM Cloud Kubernetes cluster also

performed very well for the first four scenarios, keeping an average response time of under 500 ms [13].

The IBM Cloud Kubernetes cluster (corresponding to architectural option A) performed better than the Docker Desktop local cluster (architectural option B) on the most demanding test cases, from 700 to 1000 edge nodes (almost one second faster). Option A was represented by a cloud-dedicated cluster that was more powerful than Option B, which runs on a workstation. However, in cases with more than 100 edge nodes, both performed worse than the 500 ms limit, which is usually employed for web applications. As a conclusion to these tests, in a production environment, a more powerful cluster is recommended in order to provide satisfactory performance for cases with more than 100 edge nodes.

6.4 Discussion and Lessons Learned

6.4.1. Performance comparison based on scenarios

This section discusses the results obtained for the implementation of the two architectural options for each of the options presented in Chapter 4.3. I also present these results in [16].

Small apartment. For one edge node, the differences that can be seen in the response times (Fig. 6.1) are mainly due to the fact that the Docker Desktop local cluster is deployed in a local environment and the latency is lower compared with the IBM Cloud Kubernetes cluster setup. Even though there were large differences between the respective response times, the results for the public cloud implementation still fall under an acceptable response time, i.e., under 500 ms. The run times of the algorithm for emergency detection were similar.

House. For the two edge node configurations, the readings were not very different to those obtained with the 1 edge node approach (from the small apartment scenario). The results for both hosting options (Docker Desktop local cluster and IBM Cloud Kubernetes cluster) were very similar to those obtained for the previous scenario, offering a good performance.

Small residential building. With the increase in the number of edge nodes tested, most of the changes related to the response time were registered on the Docker Desktop local cluster (Fig. 6.3). The most notable difference and the cause for this slower response was related to the emergency detection algorithm run time, which increased significantly in the case of the Docker Desktop local cluster. For the IBM Cloud Kubernetes cluster, the values remained similar to those obtained with the two-edge-node approach.

Office building. After increasing the requester number to 20 edge nodes, the response times increased for both tested approaches: the Docker Desktop local cluster and the IBM Cloud Kubernetes cluster. Nonetheless, the results show that both local and cloud approaches are able to handle an office building where 20 edge nodes send environmental data at the same time.

A complex of buildings. For this scenario, the size of the complex and the number of edge nodes implanted can make an important difference from a performance point of view; the cases that were simulated are discussed separately. For both implementations of the containerized environment setup, the 50-edge-node experiments showed an increase in the average response time, with a more visible change for the Docker Desktop local cluster, where a developmental virtual machine was installed on a workstation. The smaller increase in response time for the IBM Cloud Kubernetes cluster was influenced by the use of a dedicated cluster to provide stability. Two-thirds of the requests had a response time below the recommended value of 500 ms (most between 400 and 500 ms), and the other third had response times of between 500 and 575 ms. The average response time was higher when the number of active threads was low,

because there were only a few users waiting for their calls to be executed; the others had already been served, i.e., the load was high. The run time of the algorithm for emergency detection increased for both approaches, providing a similar value. For the 100-edge-node stress test, both setups showed an increase in the average response times (Fig. 6.2). For the IBM Cloud Kubernetes cluster, the run time of the algorithm for emergency detection for the cloud system was very similar to the result obtained for the 50-edge-node experiment. For the Docker Desktop local cluster, the run time increased (Fig. 6.4), again illustrating the advantage provided by a dedicated cluster compared to the local workstation. Between 200 and 600 nodes, the response times for the two design options were quite similar; yet, starting from 700 edge nodes, a clear advantage towards the IBM Cloud Kubernetes cluster was shown, due to the fact that both the response times and the run times were better. The 1000-edge-node experiment verified the behavior of the system when 1000 calls were sent to it. The error rate for each of the systems was 0, which indicates that all environment data were successfully added to the EWS database. A big difference that only occurred for this experiment, but not the other scenarios, is that the Docker Desktop local cluster provided a higher average response time compared with the IBM Cloud Kubernetes cluster. For the other scenarios, the result was the opposite; the reason for this is that the request was sent locally, and the time difference was accounted for by the fact that, for the public cloud option, the request was sent via the Internet. Regarding the algorithm for emergency detection, the Docker Desktop local cluster implementation provided a run time that was almost three times higher than that obtained with the IBM Cloud Kubernetes cluster.

An important limitation when working with a cloud solution may be related to the location of the datacenter. This choice influences the network latency present in the call to the application. This remark also can be taken into consideration for other Cloud-related applications such as [14], [15], [17], [18].

This underlines the fact that the use of a dedicated cloud cluster can offer a more consistent performance under a high load. However, as a general remark for the results related to the public cloud option, an average response time of almost 2000 ms is too high for this type of system. Based on these results, for a complex of smart buildings, a more powerful cluster configuration than the one tested in this study and described in Chapter 6.2.4 is recommended. The 1000 edge nodes represent a load experiment that may occur in real-world situations for large building complexes, such as smart campuses or large shopping centers.

6.4.2 Recommendations

In Chapter 4, I analyzed several design options for the EWS, considering two criteria: the containerized architecture and the edge network topology. I also presented the EWS architecture in [19]. After the qualitative evaluation, the method applied for the performance testing of the two architectural design options, (A) the public cloud Kubernetes cluster and (B) the local datacenter Kubernetes cluster, was described in Chapter 6. Further, this section presents the design choices made for developing EWS.

Containerized Architectural Choice: Based on the test results from Chapter 5 and also in presented [20] and the performance comparison for the five scenarios, I conclude that architectural choice A (public cloud Kubernetes cluster) provides a stronger performance for a load corresponding to a complex of smart buildings, where the number of edge nodes used for gathering information is large. Another reason for this choice is the separation of the building monitoring system from the actual monitored building. The main advantage in case of a hazardous event that could produce an outage within the building is that the system would not

be affected, and it would retain the information in regard to the event. Another major reason for this choice is related to the initial and maintenance costs, which are lower with the public cloud approach. Cloud services also have a guaranteed Service Level Agreement that assures the system can run for more than 99.9 percent of the time. This aspect is crucial for a building monitoring system that includes the detection of different emergencies that can occur in the building in its scope.

Edge Choice: Based on the qualitative analysis presented in Chapter 4.3, I am in favor of the design choice that uses sensor nodes connected to microprocessor-powered edge nodes and not sensing devices based on microprocessors that are directly connected to the EWS services on Kubernetes. The edge node's role is to gather data, perform some basic decentralized processing, e.g., for detecting when to activate local alarm devices and sends data to the EWS to be processed in a centralized way using algorithms that require more resources. To estimate the costs corresponding to the two options for the edge network, I assumed that the same types and number of sensors were used, and we omitted other costs, such as those related to the containerized architecture. Let us consider a microprocessor-based node run with Raspberry Pi [21] - a very popular development board with the ability to act both as a sensing device (Edge Option B) and as a broker and data processor edge node (Edge Option A). For both options, Raspberry Pi provides enough computing power to run even more complex algorithms in the future [22]. The average cost for a Raspberry Pi 3 (4× ARM CPU, 1.2 GHz, 1 GB RAM, 10/100 Ethernet, 2.4 GHz 802.11n wireless) is $35. For Edge Option A, a microcontroller-based sensor node may be run with NodeMCU v3 (32-bit CPU, 80 MHz, 128 KB RAM), a microcontroller-based board with integrated Wi-Fi capability, to send data to an edge node. The average cost for a NodeMCU board is approximately $7. Thus, our estimations show that a solution containing only microprocessor-based sensing devices (Edge Option B) would cost five times more than one that also includes microcontroller-based sensor nodes (Edge Option A). This is especially important for the case of monitoring a complex of buildings with a large number of edge nodes. Therefore, Edge Option A, containing edge nodes, was selected because, overall, it is less expensive than the other option, and because it requires a smaller number of edge nodes, which is a premise for providing better performance.

7.Conclusions

7.1 Discussion

This thesis presented a solution to the building monitoring for hazardous events detection topic as well as proposing a new set of architectures for evaluating the performance of the system. Therefore, to achieve these results, articles regarding different topics such as building monitoring, emergencies, IoT, cloud, disaster management, fog, and edge computing were investigated. I focused on presenting the Edge Watcher System Architectures - software and edge network that will be used for the evaluation of the system. The research contribution part was used to present the Edge Watcher System, the application that I developed to solve the building monitoring problem. The integration of a dedicated configuration system provides the means necessary to set up the edge devices that are scattered throughout the monitored building to collect environmental data. The mix of these offerings developed as a responsive web and application provides the building personnel with the means necessary to always be connected to their building and manage the necessary parameters. Edge Watcher System is the application that I proposed and can satisfy the above-mentioned requirements in a smart environment. Besides this matter, the integration of a dedicated configuration system provides the means

necessary to set up the edge devices that are scattered throughout the monitored building to collect environmental data.

The verification and validation topic is represented by real-world testing which validates the implementation of EWS in real-world scenarios. As a consequence, firstly I tested the one-node configuration of the EWS installed on a local server. The two architectures involved in the test were showing similar results, both having an average response time of under 40 ms which is considered very fast for the local EWS installation. The advantage of the first architecture presented was the fact that it is cheaper when implemented with a very large number of sensors, compared to the other one (Edge Option B) whose main advantage is simplicity. The last step within the validation of the EWS was the implementation of the system within a real-world environment. Therefore, I chose to implement it within University "Politehnica" of Bucharest "Precis" building where I implemented multiple design options of the EWS used for performance testing. The results I obtained after running the performance tests show that all the performance tests achieved average response times under 300 ms for the data sent from Edge Nodes to EWS. As a general remark, both Options 1 and 2 registered similar results. From this perspective, either Design Option 1 or Design Option 2 will be able to perform similarly in a real-world environment. The difference between them will appear when the cost will be taken into discussion. Option 1 implements a physical connection from a microprocessor-based device (Raspberry Pi 3) to the sensor. A higher number of such devices needs to be acquired in order to implement Option 1 compared to Option 2. For Option 2, fewer microprocessor-based devices are needed since this edge node is used as an MQTT broker that receives data from multiple cheaper MQTT-based devices such as the Node MCU. If we go deeper into the scenarios presented, I observe that the scenarios where the emergency detection algorithm was deployed on the edge node receive the response from the EWS faster since it has less to process. The biggest difference is registered for the case where the EWS is deployed on the Cloud. In this case, the local installation is 3 times faster on average. This result was to be expected since the Cloud implementation of EWS was deployed in the Milan region which is very far, compared to the local network case. Another aspect is the number of edge nodes involved. When two edge nodes were involved the response times tend to be a little higher than the one node case. Nevertheless, the differences between these two options are not very high and both demonstrated to perform as intended in these situations.

Regarding the scaling evaluation topic, firstly I measured the performance of EWS with simulated edge nodes. This part evaluated several design options for a system that monitors one or multiple smart buildings with the purpose of gathering information from a large variety of sensors, detecting abnormal situations (such as flames, toxic gas, leaks, etc.), and notifying the responsible personnel when emergency events occur. More details about this topic of notifications can be found in [23]. The design options took into account the container-based software architecture and the edge sensing devices. In addition to a qualitative analysis, I presented work based on containerized environments to test the performance, which has an important weight in alerting systems. The provided response times must remain under a certain threshold in order for the solution to be approved and implemented in a production environment. In conclusion, performance evaluation is an important stage for a project as it provides the necessary results to determine an application capacity and based on the results the exact system configuration that is needed for the desired capacity. Therefore, scenario-based testing is replicating different real-world scenarios to test the limits of the system and how it is behaving against different inputs. Besides these aspects, another important fact is the error rate that is registered for each case. For an application to be deemed stable, this metric should be

near 0 for all the calls. These arguments are validated by different experiments that were conducted against various applications and presented in the research chapter. The results recorded provided a very strong case for each of the systems to define the maximum capacity and also means of improvements in order to increase this value. To accomplish this, multiple tools can be employed for performance testing, Apache JMeter is shown as being a very popular choice because of its free distribution model and also because of the utility it provides. Regarding my main objective – Edge Watcher System multiple scenarios were being created that measure the efficiency of this application deployed on two different environments: a local cluster and a free Cloud cluster. Therefore, I implemented two containerized environment setups (an IBM Cloud Kubernetes cluster and a Docker Desktop local cluster), and I simulated the behavior for multiple scenarios corresponding to real-world configurations with 1 to 1000 edge nodes. For settings corresponding to a small apartment, a house, a small residential building, and an office building, the average response time was 250 ms higher for the public cloud than for the local cluster. However, for a complex of buildings with more than 600 edge nodes, the response time was 700 ms lower for the cloud than for the local solution I used the performance evaluation findings and the edge options analysis to make design choices for the Edge Watcher System, a solution with microcontroller-based sensor nodes, microprocessor-based edge nodes, and monitoring, configuration, and notification services with an IBM Cloud Kubernetes cluster. The metrics that were targeted for these tests are: error ratio (to validate if all requests were successful), the response time (min, max, average and percentiles, these are the most important metrics that underline the actual experience to perform calls and plays an important role in the final decision regarding the system), throughput (shows the transaction capacity). Besides the JMeter testing, another measurement was performed for the decision algorithm used by Edge Watcher System to decide if the edge node request contains a possible alert or not. For this aspect, a Node.js library was used called "execution-time". The result shown by this library is very important because it shows the exact time spent by the algorithm and can be extracted from the total response time provided by JMeter in order to decide if a more powerful configuration is needed. The results that were obtained following these tests were satisfactory for both approaches. The basic difference between the results recorded for the two systems is mainly related to the fact that the local cluster provided a faster response time because the calls were done locally. For the IBM Cloud Kubernetes cluster, the requests took longer because the system is located in the Cloud. Another remark is that the Cloud system, being provided a dedicated worker node tend to provide more consistent results even for the 95th percentile, being near the average response time. The desired value for the response time should be in the vicinity of the 500 ms mark. This was easily achieved for the first five scenarios. If more than one hundred edge nodes are connected, then a more powerful system should be employed.

7.2 Summary of Original Contributions

1. Lessons learned from the literature study of cloud smart buildings, emergency situations, disaster management, and cloud computing.
2. Proof of concept of a system for monitoring university hazards
3. Comparison between the system deployed on a virtual machine and the same system deployed on cloud.
4. Monitoring system deployment using containers

5. Architectures for both local (using a virtual machine) and cloud systems (using a Kubernetes cluster)

Monitoring is essential for every building with a high number of occupants, and it can be vital in detecting hazardous events that can affect the life of its inhabitants. There are many types of sensing devices that can be used in a building, such as environment data collectors, video monitoring, equipment information gathering, and virtual sensors.

6. Proposing a building monitoring and hazardous detection system and architecture (Edge Watcher System)
7. Proposing a data model for the Edge Watcher System that underlines all the functionality of the application.
8. Presented a well-documented API for a smart building solution that can be utilized further to integrate different services.
9. Presented a responsive web frontend that is dedicated to the configuration and monitoring of a smart building facilitating its use
10. Proposing a cloud container approach for the deployment of this system
11. Proposed a set of technologies that integrate and serve as a robust solution for a building manager system.
12. Proposed a unified framework for building monitoring systems that can be used for multiple buildings.
13. Proposed an architecture for a cloud-based smart building manager that can monitor the local building and send notifications in case of hazardous events detection
14. Created a cloud-based configuration tool that offers the possibility of generating configuration for local building edge nodes.
15. Providing multiple scenarios that can be followed for the real-world usage of the smart building manager application.
16. Proposed a cloud-based Kubernetes deployment for the building monitoring and hazardous events detection system.
17. Presented a notification algorithm that takes into account if the detected hazardous event was sent from the same node

Real-life testing scenarios are essential in validating the functioning of a system. In the case of the Edge Watcher System, a real-life performance testing scenario is represented by the implementation of a physical edge network composed of IoT devices that will gather the environmental data from the target building. Within this scope, there are multiple possible architectures that can be deployed for this kind of system that imply the selection of device types that can fulfill this action. For my thesis, I compared two types of edge network architectures and measure the performance related to data collection.

18. I created the architectures for the two based edge network options (Edge Option A and Edge Option B)

19. I implemented both solutions (Edge Option A and Edge Option B) and connected them to the Edge Watcher System
20. I tested the performance of EWS in connection with the two implemented edge network options
21. Results comparison of the two proposed solutions

I implemented the EWS within a University building environment where I tested the performance of the EWS by employing a local and a Cloud version of the system along with two different edge network architectures.

22. Two design options were considered that contained two different versions (based on Edge Option A and B and Architectural Option A and B)
23. Measuring the average response time of the EWS by taking into consideration the location of the software, the location of the decision algorithm, and the edge network architecture involved.
24. Comparison of EWS architectures used for testing in the University building

Performance testing is an essential part in the development of an application because it provides the means to find if the tested system is running on the desired parameters with different inputs. The importance of this critical step relies mostly on the idea that every new development piece must be tested on different pre-defined scenarios where different metrics are monitored.

25. Performance testing for building management systems
26. System architecture for the testing environment.
27. Cloud and local deployments for performance testing.
28. Scenario-based testing that covers multiple types of buildings and environments such as: a small apartment, a house, a small residential building, an office building, and a complex of buildings
29. Simulation of building edge nodes API requests with Apache JMeter.
30. Measurement of the hazard events detection algorithm performance run time
31. Integration of a cloud-native monitoring solution.
32. Scaling evaluation of the Edge Watcher System up to one thousand edge nodes
33. Architectural discussion based on different design choices for edge nodes and cloud system hosting
34. Network latency discussion for the cloud system
35. Performance discussion for the cloud hosting hardware and edge nodes hardware
36. Cost discussion based on the edge nodes choices
37. In-depth discussion of the testing results
38. Architecture recommendations based on the scaling evaluation and real-world testing results

7.3 List of Publications

During my thesis writing, I used content from articles 1, 4, 6, and 9 which can be found below:

1. **Florin Lacatusu**, A. D. Ionita, Marian Lacatusu, I. Damian and D. Saru, "A Comparison of Cloud Edge Monitoring Solutions for a University Building," *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, 2022, pp. 253-257, doi: 10.1109/ICCP56966.2022.10053978.

2. Marian Lacatusu, A. D. Ionita, **Florin Lacatusu** and I. Damian, "Decision support for multicloud deployment of a modeling environment," *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, 2022, pp. 247-251, doi: 10.1109/ICCP56966.2022.10053951.

3. Ioan Damian, Marian Lacatusu, **Florin Lacatusu** and A. D. Ionita, "Web Services for Guiding Persons with Locomotor Impairments in Public Spaces," 2022 26th International Conference on System Theory, Control and Computing (ICSTCC), 2022, pp. 639-644, doi: 10.1109/ICSTCC55426.2022.9931861, **WOS:000889980600107**

4. **Lăcătușu, Florin**; Ionita, A.D.; Lăcătușu, Marian.; Olteanu, A. Performance Evaluation of Information Gathering from Edge Devices in a Complex of Smart Buildings. *Sensors* **2022**, *22*, 1002. https://doi.org/10.3390/s22031002 , *Impact factor:* 3.84 – Q2, **WOS:000760176500001**

5. Lăcătușu, Marian; Ionita, A.D.; Anton, F.D.; **Lăcătușu, Florin** Analysis of Complexity and Performance for Automated Deployment of a Software Environment into the Cloud. *Appl. Sci.* **2022**, *12*, 4183. https://doi.org/10.3390/app12094183, *Impact factor:* 2.84 - Q2, **WOS:000794735600001**

6. **Florin Lacatusu**, I. Damian, A. D. Ionita and Marian Lacatusu, "Smart Building Manager Software in Cloud", U.P.B. Sci. Bull., *Series C, vol. 83, no. 4*, 2021, *Impact factor:* 0.37, **WOS:000741473700003**

7. Marian Lacatusu, **Florin Lacatusu**, Ioan Damian, Anca Daniela Ionita, Multicloud Deployment to Support Remote Learning, *15th International Technology, Education and Development Conferenc*e (INTED 2021) Proceedings (2021), pp. 4601-4606, IATED Digital Library, doi: 10.21125/inted.2021.0936

8. Ioan Damian, Marian Lacatusu, Anca Daniela Ionita, **Florin Lacatusu**, Software Services to Support Faculty Management in Times of Pandemic, *15th International Technology, Education and Development Conferenc*e (INTED 2021) Proceedings (2021), pp. 4634-4639, IATED Digital Library, doi: 10.21125/inted.2021.0943

9. **Lăcătuşu, Florin**; Ionita, A.D. Architecture for Monitoring Risk Situations in a University Environment. Rev. Roum. Sci. Tech. Electrotech. **2020**, 65, 259–263, Impact factor: 0.44, **WOS:000608261900017**

10. Adriana Olteanu, **Florin Lacatusu**, Marian Lacatusu, Iulian Craciun, Anca Daniela Ionita, ”Mobile Application for Crisis Situations in a University Campus” - The International Scientific Conference eLearning and Software for Education; Bucharest Vol. 2, Bucharest: "Carol I" National Defence University. (2018): 280-287. **WOS:000467466800038**

11. Olteanu, Adriana**;** Lacatusu, Marian; Ionita, Anca Daniela; **Lacatusu, Florin**, ”Platform for Informal Education and Social Networking to Increase Awareness Regarding Nuclear Vulnerabilities” - The International Scientific Conference eLearning and Software for Education; Bucharest Vol. 2, Bucharest: "Carol I" National Defence University. (2018): 333-340. **WOS:000467466800045**

7.4 Future Perspectives

In terms of future work, the next step for the Edge Watcher System would be to be installed in a building as the main solution for monitoring and testing its performance when functioning for a long period of time. Another interesting fact would be testing the performance of the EWS when it’s integrated into a smart city infrastructure. Therefore, multiple instances of the EWS would be deployed for each of the buildings, and the functioning of the EWS can be tested in this scenario also.

Besides this, given the fact that Edge Watcher System is a cloud-native application, it will run similarly on any cloud provider with no or minimal changes. As a consequence, another future perspective would be to test the performance of the monitoring software that runs on Kubernetes services provided by a different cloud provider to see if the performance varies between them. Currently, during the tests conducted in my study, IBM Cloud was used as the cloud provider for the EWS Kubernetes service. The performance registered in this case was satisfactory for the hardware flavor that was chosen. Therefore, comparing multiple offerings from multiple providers would be interesting because the optimal solution can be achieved by studying the performance-to-price ratio.

The Kubernetes services deployed from multiple cloud providers should be compared at the same price point and the best performance-to-price option should be chosen. Afterward, a very interesting experiment would be, besides the price, the performance testing of EWS running on Kubernetes services provided by different public cloud providers. The main requirement for this type of comparison would be to deploy the Kubernetes cluster in a similar location for all the cloud providers. This way there would be little difference when we take into consideration the network latency that could be influenced by the Kubernetes cluster’s location. If we want to move further from the Kubernetes service, another possible implementation that can be considered for the EWS backend is represented by serverless functions. These types of resources are fully managed by the cloud provider and the developer only works on the application’s code. Therefore, a very interesting comparison would be between the container-based Kubernetes solution and serverless functions with PaaS database services. The comparison that would be made between these solutions should cover, first of all, the

performance topic and also the cost difference between them. In the end, of course, this comparison can be generalized also to experiment with the most important cloud provider's options regarding serverless functions and test their offering in order to find the most suitable option.

Selected 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] Kurniawan, F.; Meidia, H.; Salim, S. Building Monitoring System Based on Zigbee. JCSI 2013, 6, 65–69

[4] Kim, D.; Muhammad, H.; Kim, E.; Helal, S.; Lee, C. TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform. Appl. Sci. 2019, 9, 191.

[5] Kubernetes Concepts - Pods: https://kubernetes.io/docs/concepts/workloads/pods/, Accessed October 5 2020.

[6] Erinle, B. Performance Testing with JMeter 2.9; Packt Publishing: Birmingham, UK, 2013

[7] Haseeb-Ur-Rehman, R.M.A.; Liaqat, M.; Mohd Aman, A.H.; Ab Hamid, S.H.; Ali, R.L.; Shuja, J.; Khurram Khan, M. Sensor Cloud Frameworks: State-of-the-Art, Taxonomy, and Research Issues. *IEEE Sens. J.* **2021**, *21*, 22347–22370.

[8] International Building Code; International Code Council: Country Club Hills, IL, USA, 2018; ISBN 978-1-60983-735-8.

[9] National Construction Code 2019 Building Code of Australia, Volume One. Available online: https://ncc.abcb.gov.au/sites/default/files/ncc/NCC_2019_Volume_One_Amendment%201.pdf (accessed on 6 December 2021).

[10] I. Damian, M. Lacatusu, F. Lacatusu and A. D. Ionita, "Web Services for Guiding Persons with Locomotor Impairments in Public Spaces," 2022 26th International Conference on System Theory, Control and Computing (ICSTCC), 2022, pp. 639-644, doi: 10.1109/ICSTCC55426.2022.9931861.

[11] I. Damian, M. Lacatusu, A.D. Ionita, F. Lacatusu (2021) Software Services to Support Faculty Management in Times of Pandemic, *INTED2021 Proceedings*, pp. 4634-4639. INTED 2021

[12] Olteanu, Adriana**;** Lacatusu, Marian; Ionita, Anca Daniela; Lacatusu, Florin, "Platform for Informal Education and Social Networking to Increase Awareness Regarding Nuclear Vulnerabilities" - The International Scientific Conference eLearning and Software for Education; Bucharest Vol. 2, Bucharest: "Carol I" National Defence University. (2018): 333-340.

[13] Serrano, D.; Bouchenak, S.; Kouki, Y.; Alvares de Oliveira, F., Jr.; Ledoux, T.; Lejeune, J.; Sopena, J.; Arantes, L.; Sens, P. SLA guarantees for cloud services. Future Gener. Comput. Syst. 2016, 54, 233–246.

[14] Lăcătușu, F.; Ionita, A.D.; Lăcătușu, M.; Olteanu, A. Performance Evaluation of Information Gathering from Edge Devices in a Complex of Smart Buildings. *Sensors* **2022**, *22*, 1002.

[15] M. Lacatusu, A. D. Ionita, F. Lacatusu and I. Damian, "Decision support for multicloud deployment of a modeling environment," *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, 2022, pp. 247-251, doi: 10.1109/ICCP56966.2022.10053951.

[16] Lăcătușu, M.; Ionita, A.D.; Anton, F.D.; Lăcătușu, F. Analysis of Complexity and Performance for Automated Deployment of a Software Environment into the Cloud. *Appl. Sci.* **2022**, *12*, 4183.

[17] M. Lacatusu, F. Lacatusu, I. Damian, A.D. Ionita (2021) Multicloud Deployment to Support Remote Learning, *INTED2021 Proceedings*, pp. 4601-4606. INTED 2021

[18] Lăcătuşu, F.; Ionita, A.D. Architecture for Monitoring Risk Situations in a University Environment. Rev. Roum. Sci. Tech. Electrotech. **2020**, 65, 259–263

[19] F. Lacatusu, I. Damian, A. D. Ionita and M. Lacatusu, "Smart Building Manager Software in Cloud", U.P.B. Sci. Bull., *Series C, vol. 83, no. 4*, 2021

[20] F. Lacatusu, A. D. Ionita, M. Lacatusu, I. Damian and D. Saru, "A Comparison of Cloud Edge Monitoring Solutions for a University Building," *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, 2022, pp. 253-257, doi: 10.1109/ICCP56966.2022.10053978.

[21] Johnston, S.J.; Cox, S.J. The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams. *Electronics* **2017**, *6*, 51

[22] Cloutier, M.F.; Paradis, C.; Weaver, V.M. A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement. Electronics 2016, 5, 61

[23] Adriana Olteanu, Florin Lacatusu, Marian Lacatusu, Iulian Craciun, Anca Daniela Ionita, ”Mobile Application for Crisis Situations in a University Campus” - The International Scientific Conference eLearning and Software for Education; Bucharest Vol. 2, Bucharest: "Carol I" National Defence University. (2018): 280-287.