Athens Journal of Technology and Engineering - Volume 10, Issue 4,
December 2023 – Pages 215-226

A Data Streaming Architecture for Air Quality Monitoring in Smart Cities

By Aleksa Miletić[*], *Petar Lukovac*[±], *Tamara Naumović*[°], *Danijela Stojanović*[•] *& Aleksandra Labus*[♦]

This paper aims to present a modeling approach for the seamless data streaming process from smart IoT systems to Apache Kafka, leveraging the MQTT protocol. The paper begins by discussing the concept of real-time data streaming, emphasizing the need to transfer data from IoT/edge devices and sensors to Apache Kafka in a timely manner. The second part consists of a literature overview that shows the analysis and systematization of different types of architectures in the broad sense of crowdsensing, followed by specific architectures regarding edge and cloud computing. The methodology section will propose an infrastructure and data streaming architecture for smart environment services, such as air quality monitoring. Lastly, a discussion about results and future development will be shown in the last two sections. The proposed integration approach offers several advantages, including efficient and scalable data streaming, real-time analytics, and enhanced data processing capabilities.

Keywords: *real-time data streaming, smart healthcare, Apache Kafka, data integration, stream processing*

Introduction

Efficient message distribution systems have gained significant attention in recent years, as they contribute to optimizing architectural frameworks. These optimizations became a necessity having in mind that the Internet of Things (IoT) paradigm, a leader in Industry 4.0, increased the number of devices connected to the Internet (Akbar et al. 2017, Froiz-Míguez et al. 2018, Khriji et al. 2022), creating a climate for the successful development of numerous services in the domain of smart cities (Cheng et al. 2018, Khriji et al. 2022, Samizadeh Nikoui et al. 2021). Moreover, nearly all services of smart cities are in demand for ultralow latency and fast response time, further proving the need for efficient message distribution systems.

Different architectures require different approaches in the modelling and development of message distribution systems:

[*]Teaching Associate, University of Belgrade Faculty of Organizational Sciences, Serbia.
[±]Teaching Associate, University of Belgrade Faculty of Organizational Sciences, Serbia.
[°]Teaching Assistant, University of Belgrade Faculty of Organizational Sciences, Serbia.
[•]Research Associate, Institute of Economic Sciences, Serbia.
[♦]Full Professor, University of Belgrade Faculty of Organizational Sciences, Serbia.

https://doi.org/10.30958/ajte.10-4-2 doi=10.30958/ajte.10-4-2

- (Hugo et al. 2020) propose the use of Apache Kafka as a centralized message distribution system to streamline the architecture of monolithic applications.
- (Fridelin Panduman et al. 2019) also propose the use of Apache Kafka server, alongside MQTT protocol, for SEMAR (Smart Environment Monitoring and Analytics in Real-time), a cloud computing platform based on microservice architecture.
- (Khriji et al. 2022) proposed AWS (Amazon Web Services) cloud-based architecture with AWS message broker - REDA (Real-time Event-Driven Architecture).

Smart city services are relying on infrastructures based on IoT, edge, and cloud computing technologies, whose interoperability requires an efficient message distribution system. This paper proposes a methodology that enables continuous monitoring and collection of air quality data in real-time, from sensors deployed at different locations in smart cities. The methodology introduces a two-layer architecture: Edge – air quality data collection, and Cloud – receiving and processing data in a stream.

This paper is organized as follows. Next section presents a review of relevant literature related to smart city IoT architectures and data processing. Then comes the methodology and design of the proposed data streaming architecture. While insights about the implementation and discussion about results are provided afterwards and finally the paper is concluded with a discussion of the applicability of the proposed architecture and future development.

Literature Review

Based on the rapid growth and complex requests of modern systems, for cloud and edge computing it is important to study different methods and architectures which have proven to be successful. This analysis will allow us to better understand best practices and address the challenges that bring us this dynamic field of computing.

Edge computing is a decentralized computing infrastructure that allows remote devices to process data closer to the edge of the network, near the source (Mitrović et al. 2023). Several analyses have been conducted on an edge computing platform that proves edge computing is a good solution for cooperation with cloud, network communication, and edge equipment (Chen et al. 2018, Martin Fernandez et al. 2018, Raza et al. 2019). This approach offers several advantages, including reduced latency, bandwidth optimization, enhanced privacy and security, offline operation (Hassan et al. 2019, Shi et al. 2016, Varghese et al. 2016). Also, one of the most important things in edge is data privacy, reduced attack surface, local threats, communication security, trustworthiness of edge devices. Some of these problems are addressed in the following papers (Ali et al. 2021, Markham and Payne 2001, Xiao et al. 2019).

Cloud-based Data Stream Processing is a type of data processing system that executes a set of continuous queries over a potentially unbounded data stream (Heinze et al. 2014). It constantly outputs new results and is designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads. This streaming approach enables real-time or near-real-time data ingestion, allowing organizations to gain valuable insights, make data-driven decisions, and take timely actions based on the incoming data (De Souza et al. 2020, Sahni and Vidyarthi 2021). The main thing about cloud computing is that it allows users to quickly deploy their applications in an elastic setting through on-demand acquisition/release of resources at a low cost (Runsewe and Samaan 2021).

In recent years, there has been a growing interest in employing efficient message distribution systems to optimize architectural frameworks. The authors (Hugo et al. 2020) propose utilizing Apache Kafka as a centralized message distribution system to streamline the architecture for monolithic applications. The integration of MQTT and Apache Kafka is achieved through the development of a specialized Kafka Connect connector, enabling scalable and resilient data collection from MQTT sources and its transmission to Kafka server. This integration facilitates fast and reliable real-time data transfer between various applications, particularly for handling large data volumes. The platform called SEMAR (Smart Environment Monitoring and Analytical in Real-time) (Fridelin Panduman et al. 2019) presents a cloud computing platform based on microservice architecture. The system consists of a device that is placed in the car and sends data via MQTT protocol to the server on which Apache Kafka is installed. They found that the average delay in sending information is 0.09ms, which is enough for the system to be considered to work in real-time. REDA is a cloud-based event-driven architecture proposed for real-time data processing in wireless sensor networks and IoT devices (Khriji et al. 2022). This architecture offers flexibility, high availability, and distributed computing while achieving high throughput and minimizing latency. By utilizing open-source frameworks and components such as a sensing unit, gateway unit, lightweight messaging protocol, event-stream processing unit, and distributed document database, REDA presents a cost-effective solution for real-time data processing. Authors (Mitrović et al. 2023) presented an IoT based framework for air quality measurements, that is suitable for implementation into different smart environments.

The article of Javed et al. (2018) presents a fault-tolerant architecture for IoT applications, which combines cloud and edge computing. This architecture, named CEFIoT, allows flexible compute placement on edge or cloud without any changes in source code. Also, the article shows a case study of a security camera system to show fault tolerance possibility in architecture. Cao et al. (n.d.) explain edge computing as a computing paradigm that does a computation on the edge of a network. Proximity to users and the provision of better intelligent services are highlighted by better performance of data transfer and decreasing data latency. Edge computing aims to provide services at the edge of the network and meet the needs of the IT industry in high-speed connectivity, real-time processing, and data security. Also, one interesting topic is EMMA which serves as a middleware

solution that enhances the efficiency, reliability, and performance of MQTT-based communication in edge computing environments, enabling more effective and responsive IoT applications. The authors (Rausch et al. 2018) consider how edge computing can improve the performances of just cloud-based architectures. The opinion of Koziolek et al. (n.d.) is that MQTT Broker is a good choice for distributed IoT Edge Computing for a few reasons. First, MQTT brokers such as Eclipse Mosquitto, EMQX, HiveMQ i VerneMQ support the MQTT protocol which is lightweight, effective, and good for IoT communication. Therefore, these brokers provide support for clustering, enabling scalability and high system availability. As open-source code, they are customizable and available in different variants, providing opportunities to adapt to specific project requirements. These features make MQTT Brokers a popular choice for distributed IoT Edge Computing scenarios.

Sotskov et al. (2023) presented IRONEDGE architectural framework that can be used in different edge Stream Processing solutions. The first layer of this architecture is the Data Source layer which is used for processing and sending data. Data that is collected and processed in the Data Source layer is ready for publishing and called Data Reports. The second layer consists of a few services. The first one being Data Collection service which receives data from corresponding Data Sources sending it to the Stream processing service which is used to derive knowledge from the accumulated Data Reports, which creates a base for event creation. Data transfer is enabled by AMM (Asynchronous Messaging Middleware). Local data, events, and reports are stored in Local Storage for further analysis. Data stream processing can be divided into more phases: receiving data, mapping in a particular object, decomposition of data according to certain criteria, analysis of grouped data, and transferring to the cloud. As AMM (Sotskov et al. 2023) chose Apache Kafka because of easy connectivity with external systems. The authors have observed the performances of the system in two cases: with Kafka Connect (K0-WC) and without Kafka Connect (K1-NC). They concluded that for loss rates and latency, K1-NC offered lower event loss and lower processing latency compared to K0-WC. For actual throughput and log time, K1-NC showed an increase in throughput but never matched the input rate for the Data Reports, while K0-WC clearly showed low throughput resulting from high losses.

As it is shown in previous examples of architectures, to send data in constantly changing environments and at the same time in real-time, MQTT is particularly useful, while Apache Kafka is used to store large amounts of data or to integrate different applications or data centers. Because of their similarities, they can be considered a great couple. Both have topics and publish/subscribe patterns. There are several ways to connect MQTT and Kafka (*MQTT and Kafka. How to Combine Two Complementary… | by Techletters | Python Point | Medium* n.d.):

1. Sending messages from edge devices, using MQTT and Kafka broker. The problem arises in the fact that the device has to send data using two protocols, and it has to be sure that the data either arrive on both or does not arrive on either of them.

2. Creating an application that will represent a bridge between MQTT and Kafka.
3. Connecting to Kafka via MQTT proxy. It is used if there is a need to send messages that will for sure arrive on the other side.
4. Connection of MQTT broker and Kafka cluster via Kafka Connect – a good solution that uses connectors as an extension to connect.
5. Connection between MQTT broker and Kafka via MQTT broker extension - this extension enables the injection of messages from IoT devices into the Kafka cluster.

Most authors include mobile phones in their systems to gather data from mobile device sensors. In the research of Kraft et al. (2020), a mobile collective sensing system is proposed that enables the implementation of a noise level map for tinnitus patients. After the requirements analysis and design phase, the system is decomposed into bounded contexts to achieve a clear and shared definition of consistency between team members. Five bounded contexts were identified, including user identity, social aspects, measurement, incentives, and communication. This architecture uses a cloud-native approach, which enables efficient and scalable processing of concurrent noise measurement requests, using microservices and containerization technologies such as Docker and Kubernetes. In addition, the system uses in-stream processing and the Apache Kafka platform for real-time data processing and enabling decoupled processing of incoming geospatial data. To display polluted areas on the map, the authors decided to use geospatial data partitioning techniques. Hierarchical partitioning of geospatial data, such as the implementation of the Discrete Global Grid System (DGGS), allows data to be divided into different levels of detail. DGGS enables the representation of data in different partition sizes, which enables aggregation and visualization at different scales. The map has Hexagonal Hierarchical Spatial Index (H3) system. H3 systems allow precise positioning of geospatial data in appropriate partitions based on their coordinates. This technique facilitates analyzing and visualizing polluted parts on the map.

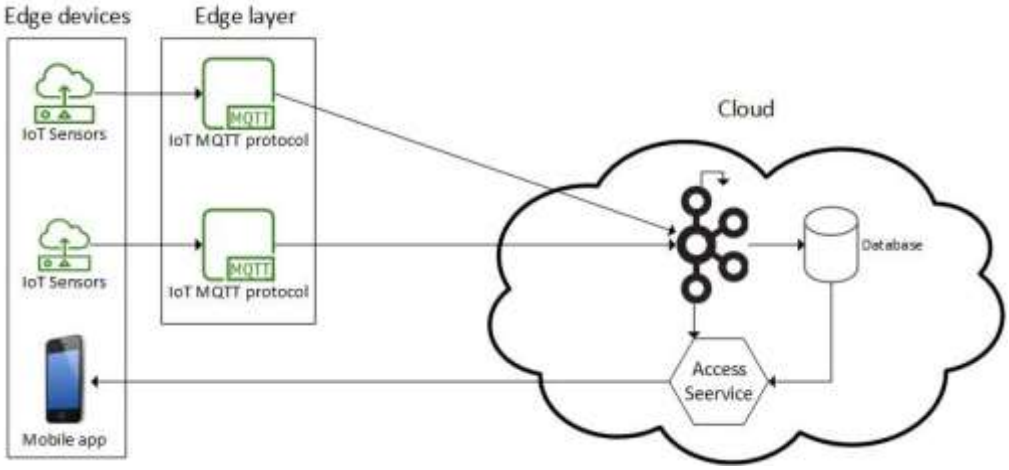
The development of the mobile crowdsensing system for monitoring noise pollution for decision-making purposes in smart cities by collecting, storing, and visualizing data on noise pollution in real time has been described in the article (Jezdović et al. 2021). These authors also point out the experiment results conducted in Belgrade, Serbia, and recommendations on how this system can be applied in other cities. The application presented in this paper is a mobile crowdsensing system for detecting noise in smart cities. The system contains a crowdsensing mobile application, cloud, and big data infrastructure. The mobile application enables noise recording using a microphone on mobile devices, recording the location of detected noise using a GPS device, performing spectral analysis on audio data, and storing transformed data and location data in a cloud database. The web application allows the view of polluted data on Google Maps. The type of the map is a heatmap on which is presented red, orange, and green areas for the high, medium, and low levels of noise respectively. In the last two

papers we can see that the authors have used two different approaches to represent their data on the map.

Methodology

This paper presents a methodology that enables continuous monitoring and collection of air quality data in real time, from sensors deployed at different locations in smart cities. With this architecture, data is transferred from Edge devices to Cloud servers, where it can be further analysed and used for decision-making and air quality management in smart cities. The methodology of this research includes the use of a two-layer architecture: Edge and Cloud. In the Edge layer, air quality data is collected from sensors connected to a Raspberry Pi (RPi) device. According to the Environmental Protection Agency, the data that needs to be measured in Belgrade are SO_2, PM_{10}, NO_2, CO, NO, $PM_{2.5}$ (*Агенција За Заштиту Животне Средине - Министарство Заштите Животне Средине* n.d.). Such data is sent via the MQTT protocol to the cloud (see Figure 1).

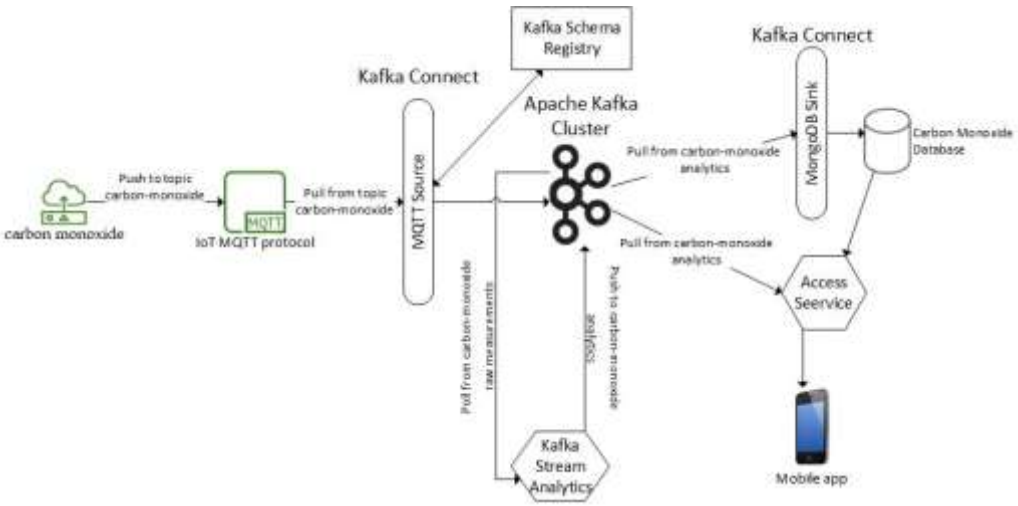
Figure 1. *Data Streaming Architecture for Air Quality Monitoring in Smart Cities*



To enable data transfer, the MQTT protocol is used and the Mosquitto MQTT broker, which is installed on the RPi device. Mosquito was chosen because it is the MQTT protocol which provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for IoT messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers (*What Is Mosquitto MQTT?* n.d.). These tools allow sending data to Apache Kafka, which resides in the Cloud layer of the architecture. Apache Kafka is used as a central mechanism for receiving and processing data in a stream. Also, it is chosen as the tool because it is highly scalable for processing and streaming data in real time. Kafka enables data replication and data availability even in case of network failures or interruptions, which is important for continuous monitoring of air quality (Korab n.d.). Also, Kafka supports simultaneous sending and receiving of data, which enables real-time data analysis and processing. This is

a very important feature in smart cities because it provides quick detection and reaction to changes in air quality. The data coming to Kafka would be processed and saved in the database and sent in real time via the access series to end user applications.

Figure 1. *Data Streaming Architecture for Air Quality Monitoring in Smart Cities with Data Flow*



This paper proposes a detail architecture which is presented on Figure 2. The system is designed to connect to devices across the city. A Raspberry Pi device is set up to which a group of sensors for measurement is attached SO_2, PM_{10}, NO_2, CO, NO, CO_2, NH_3, $PM_{2.5}$. The image shows the flow of data from the carbon monoxide sensor. Data is prepared for sending by creating an object with all the necessary information and metadata (see Figure 3). From this point data will be sent to corresponding MQTT topic and from there streamed to the Apache Kafka. Specifically, carbon monoxide data will be sent to the topic named carbon-monoxide. Connection between MQTT and Apache Kafka using Kafka Connect was chosen. In order to connect MQTT with Kafka, there is a special Kafka connect that collects data and writes it to the Kafka topic (carbon-monoxide) – MQTT Source Connector. Before the data reaches the Kafka Cluster, it is necessary to check if data is valid. Kafka Schema Registry serves to validate the object that arrives to Kafka. If the structure of the objects does not match, that data will not be processed. When the data reaches the Kafka Cluster, i.e. on the carbon-monoxide topic, it is forwarded to Kafka Stream Analytics, where data analysis is performed, such as geographic analysis based on location, time analysis for tracking data history etc. The processed data is forwarded to the carbon-monoxide analytics topic. From this topic, data is collected by the Access Service and forwarded to all users for monitoring pollution in real time. The processed data is also stored in the database to monitor the history of air quality. The data is first pulled from the topic via a Kafka Connect called MongoDB Sink, which is used to write data to MongoDB.

Figure 2. *Example of Object Sent to Apache Kafka*


```
{
    "measurement": {
        "type": "carbon_monoxide",
        "value": 2.54
    },
    "device_info": {
        "manufacturer": "Raspberry Pi",
        "model": "RPi-4",
        "device_id": "uuid",
        "height": 10
    },
    "timestamp": "2023-06-09T10:15:00Z",
    "units": "ppm",
    "location": {
        "coordinates": [9.967101535538086, 48.384883089298114],
        "municipality": "Vožđovac"
    }
}
```


Results and Discussion

A prototype data streaming system was developed to display measurements of air quality, focusing on monitoring Sulfur Dioxide (SO_2), Particulate Matter (PM10, PM2.5), Nitrogen Dioxide (NO_2), Carbon Monoxide (CO) and Nitric Oxide (NO), Ammonia (NH_3), carbon dioxide (CO_2) levels. The system was based on a Raspberry Pi microcomputer with various sensors connected, including the MQ-7, MQ-137, MQ-135, MQ-136, MQ-165, MG-811, WSP-1110, and SDS011. Moreover, instances where a sensor malfunctioned or produced default values that significantly deviated from previous measurements resulted in the exclusion of data from being sent to the cloud for processing.

To evaluate the quality of data, it was necessary to take into consideration that sensors might exhibit a certain degree of imprecision. To ensure reliable values, averaging calculations were performed using data from multiple sensors. This approach allowed for a more accurate assessment of data quality and minimized the impact of potential inaccuracies from individual sensors. The collected raw data was streamed via Apache Kafka. Every type of sensor has its topic in the MQTT broker which sends data to the Kafka topic for that specific measured value. Apache Kafka performed real-time analyses of collected data, stored them to the MongoDB and streamed live. NoSQL database has been chosen because large amounts of data are expected from IoT systems with numerous writing operations (Tudorica and Bucur 2011). By leveraging the power of Apache Kafka, the system facilitated efficient data transmission, allowing for real-time monitoring and analysis of air quality metrics. To derive meaningful insights, calculations were performed to determine the average pollution levels for each specific hour (see Figure 4) and specific periods of the day (see Figure 5). All measurements are shown on different

diagrams due to the sensors different output value ranges. These metrics provided valuable information for assessing air quality trends and identifying potential pollution patterns during different periods of the day in Belgrade.

Figure 3. *Line Chart Based on Average Values for Every Hour in the Day*

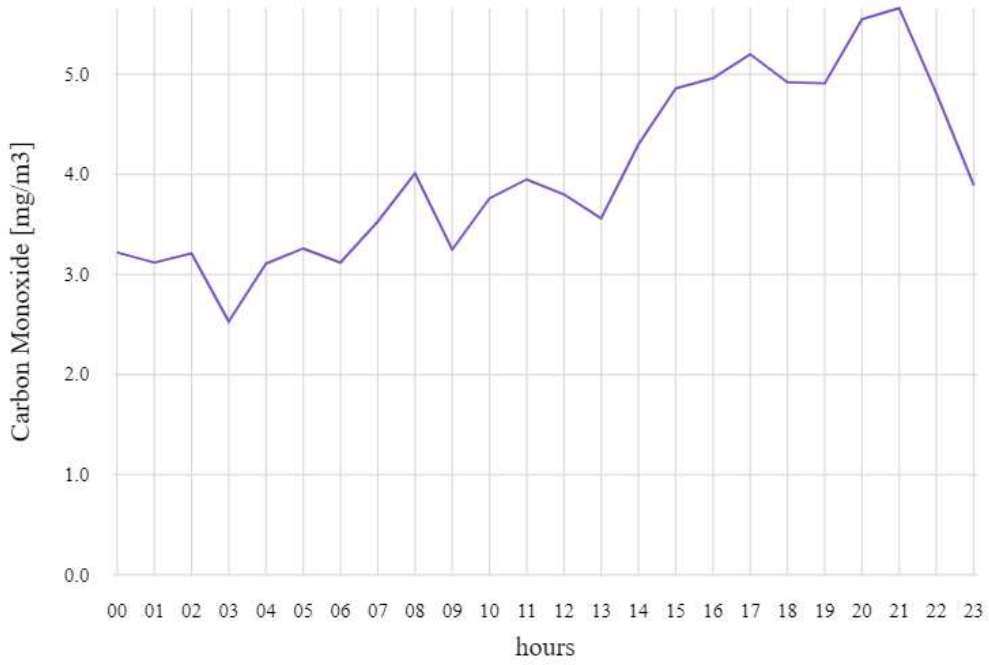


Figure 4. *Bar Chart Based on Average Values for Specific Times of the Day*

Conclusions

The main goal of this paper is to present the design and development of the robust architecture for crowdsensing systems in smart cities. This architecture provides the base for effective collection and analysis of data in real-time, opening new possibilities for measuring and monitoring level of pollution and other parameters in local environment. Even though the architecture shows easy processing and analyses of data, some shortcomings have also been identified. Setting optimal number of topics and replication factor of topics, and how many brokers is it needed. For now the replication number is set to three, because it is the golden rule (Ibryam n.d.), but with the further analysis and testing that can be changed because of a number of sensors that will be sending data to the Apache Kafka.

Implementation of the developed architecture has applicability in practice for creating crowdsensing systems in smart cities, especially for measuring air quality levels. This enables fast and efficient data collection from various sensors and devices, providing valuable information for management and monitoring of environmental quality. Future steps may include implementing the developed architecture on the Docker platform, scaling the system via Kubernetes, and displaying measurements via a map. This would enable better resource management and scaling, as well as greater flexibility and fault tolerance in the system environment. Through this work, it was observed that creating an efficient architecture for data streaming in crowdsensing systems is essential for successful real-time data collection and analysis. These results can serve as guidelines for other researchers dealing with similar problems in the field of smart cities and air quality meters. Future research will be focused on exploring the possibility of applying this architecture in wider contexts of smart cities, as well as optimizing the number of topics, replications, and brokers to achieve maximum efficiency and scalability. Also, research into the integration of this architecture with other relevant technologies and platforms opens the door for further improvement of crowdsensing systems in smart cities.

References

Akbar A, Khan A, Carrez F, Moessner K (2017) Predictive analytics for complex IoT data streams. *IEEE Internet of Things Journal* 4(5): 1571–1582.

Ali B, Gregory MA, Li S (2021) Multi-access edge computing architecture, data security and privacy: a review. *IEEE Access* 9: 18706–18721.

Cao K, Liu Y, Meng G, Sun Q (n.d.) *An overview on edge computing research*. Available at: https://doi.org/10.1109/ACCESS.2020.2991734.

Chen B, Wan J, Celesti A, Li D, Abbas H, Zhang Q (2018) Edge computing in IoT-based manufacturing. *IEEE Communications Magazine* 56(9): 103–109.

Cheng B, Solmaz G, Cirillo F, Kovacs E, Terasawa K, Kitazawa A (2018) FogFlow: easy programming of IoT services over cloud and edges for smart cities. *IEEE Internet of Things Journal* 5(2): 696–707.

De Souza PRR, Matteussi KJ, Veith ADS, Zanchetta BF, Leithardt VRQ, Murciego AL,

et al. (2020) Boosting big data streaming applications in clouds with burstflow. *IEEE Access* 8: 219124–219136.

Fridelin Panduman YY, Ulil Albaab MR, Anom Besari AR, Sukaridhoto S, Tjahjono A, Nourma Budiarti RP (2019) Implementation of data abstraction layer using kafka on SEMAR platform for air quality monitoring. *International Journal on Advanced Science, Engineering and Information Technology* 9(5): 1520–1527.

Froiz-Míguez I, Fernández-Caramés TM, Fraga-Lamas P, Castedo L (2018) Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on MQTT and ZigBee-WiFi Sensor Nodes. *Sensors* 18(8): 2660.

Hassan N, Yau KLA, Wu C (2019) Edge computing in 5G: A review. *IEEE Access* 7: 127276–127289.

Heinze T, Aniello L, Querzoni L, Jerzak Z (2014) Cloud-based data stream processing. In *DEBS 2014 - Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, 238–245.

Hugo A, Morin B, Svantorp K (2020) Bridging MQTT and Kafka to support C-ITS: a feasibility study. In *Proceedings - IEEE International Conference on Mobile Data Management*, *2020-January*, 371–376.

Ibryam B (n.d.) *Fine-tune Kafka performance with the Kafka optimization theorem.* Red Hat Developer. Available at: https://developers.redhat.com/ articles/2022/05/03/fine-tune-kafka-performance-kafka-optimization-theorem#the_kafka_optimization_theorem.

Javed A, Heljanko K, Buda A, Framling K (2018) CEFIoT: a fault-tolerant IoT architecture for edge and cloud. In *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings*, *2018-January*, 813–818.

Jezdović I, Popović S, Radenković M, Labus A, Bogdanović Z (2021) A crowdsensing platform for real-time monitoring and analysis of noise pollution in smart cities. In *Sustainable Computing: Informatics and Systems*, 31.

Khriji S, Benbelgacem Y, Chéour R, Houssaini DE, Kanoun O (2022) Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *Journal of Supercomputing* 78(3): 3374–3401.

Korab J (n.d.) *How to survive a Kafka outage*. Available at: https://www.confluent.io/ blog/how-to-survive-a-kafka-outage/.

Koziolek H, Grüner S, Rückert J (n.d.) *A comparison of MQTT brokers for distributed IoT edge computing*.

Kraft R, Birk F, Reichert M, Deshpande A, Schlee W, Langguth B, et al. (2020) Efficient processing of geospatial mhealth data using a scalable crowdsensing platform. *Sensors (Switzerland)* 20(12): 1–21.

Markham T, Payne C (2001) Security at the network edge: a distributed firewall architecture. In *Proceedings - DARPA Information Survivability Conference and Exposition II, DISCEX 2001*, *1*, 279–286.

Martin Fernandez C, Diaz Rodriguez M, Rubio Munoz B (2018) An edge computing architecture in the internet of things. In *Proceedings - 2018 IEEE 21st International Symposium on Real-Time Computing, ISORC 2018*, 99–102.

Mitrović N, Đorđević M, Veljković S, Danković D (2023) *View of IoT enabled software platform for air quality measurements*. Available at: https://www.ebt.rs/journals/index.php/conf-proc/article/view/188/135.

MQTT and Kafka. How to combine two complementary… | by Techletters | Python Point | Medium (n.d.) Available at: https://medium.com/python-point/ mqtt-and-kafka-8e470eff606b.

Rausch T, Nastic S, Dustdar S (2018) EMMA: Distributed QoS-aware MQTT middleware for edge computing applications. In *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, 191–197.

Raza S, Wang S, Ahmed M, Anwar MR (2019) A survey on vehicular edge computing: architecture, applications, technical issues, and future directions. *Wireless Communications and Mobile Computing* 2019: 3159762.

Runsewe O, Samaan N (2021) Cloud resource scaling for time-bounded and unbounded big data streaming applications. *IEEE Transactions on Cloud Computing* 9(2): 504–517.

Sahni J, Vidyarthi DP (2021) Heterogeneity-aware elastic scaling of streaming applications on cloud platforms. *Journal of Supercomputing* 77(9): 10512–10539.

Samizadeh Nikoui T, Rahmani AM, Balador A, Haj Seyyed Javadi H (2021) Internet of Things architecture challenges: a systematic review. *International Journal of Communication Systems* 34(4): e4678.

Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. *IEEE Internet of Things Journal* 3(5): 637–646.

Sotskov YN, Tchernykh A, Werner F, Vitorino JP, Simão J, Datia N, et al. (2023) IRONEDGE: stream processing architecture for edge applications. *Algorithms 2023* 16(2): 123.

Tudorica BG, Bucur C (2011) A comparison between several NoSQL databases with comments and notes. In *Proceedings - RoEduNet IEEE International Conference*.

Varghese B, Wang N, Barbhuiya S, Kilpatrick P, Nikolopoulos DS (2016) Challenges and opportunities in edge computing. In *Proceedings - 2016 IEEE International Conference on Smart Cloud, SmartCloud 2016*, 20–26.

What is Mosquitto MQTT? (n.d.) Available at: https://www.eginnovations.com/documentation/Mosquitto-MQTT/What-is-Mosquitto-MQTT.htm.

Xiao Y, Jia Y, Liu C, Cheng X, Yu J, Lv W (2019) Edge computing security: state of the art and challenges. In *Proceedings of the IEEE*.

Агенција за заштиту животне средине - Министарство заштите животне средине (n.d.) Available at: rom http://www.sepa.gov.rs/.