Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 235 (2024) 852–862

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Machine Learning and Data Engineering (ICMLDE 2023)

Deploying a web service application on the EdgeX open edge server: An evaluation of its viability for IoT services

Aalwahab Dhulfiqar[a], Mohammed A. Abdala[b], Norbert Pataki[a], Máté Tejfel[a,*]

[a]*Department of Programming Languages and Compilers, Faculty of Informatics, Eötvös Loránd University, 1/C Pázmány Péter st., Budapest, H-1117, Hungary*

[b]*AL-Hussain University College, Department of Medical Instrumentation Techniques Engineering, Karbala, Iraq*

Abstract

With the advent of 5G and the promise of 6G, edge computing is becoming increasingly important. Edge computing involves the proximity of computational resources to the devices and sensors responsible for data generation, reducing delay and improving real time data processing capabilities. EdgeX is an openly available framework designed for edge computing. that provides a platform for building IoT services and applications. In this paper, we evaluate the EdgeX open edge server for use in IoT services. We first deploy a web service application into the edge using a specific port, and then test the EdgeX server to see whether it can be used as a reliable and effective open-source edge server. Our results show that the EdgeX server is capable of supporting IoT services and can be used as an open-source edge server for a variety of applications. We conclude that EdgeX is a valuable resource for developing edge computing solutions, and that the open edge server can significantly contribute to the advancement of forthcoming IoT service innovations.

Keywords: Microservices; Edge-computing; EdgeX server; APIs; Edge devices; Sensor data

1. Introduction

Edge computing is now used as a remedy for addressing the constraints associated with cloud-computing. This approach involves situating computational resources, such as edge servers, in proximity to end devices to fulfill the demanding latency prerequisites of specific applications, as noted in the study by [15]. Through the implementation of edge-computing, the application services can be conveniently installed on edge-servers for prompt data processing and fast responses [2]. This results in diminished network bandwidth utilization and latency, ultimately augmenting the performance and utilization in IoT applications, as corroborated by previous research by [4, 13].

* A Alwahab Dhulfiqar
E-mail address: Dolfi@inf.elte.hu

10.1016/j.procs.2024.04.081

EdgeX is designed to enable communication across different protocols and promote compatibility between devices. Additionally, it provides a container-based application service, which allows for the convenient addition or removal of application services without disrupting other services already in place. EdgeX is a powerful and versatile open source Internet of Things (IoT) solution that is designed to facilitate the seamless ingestion of data from multiple sources and forward it to a centralized system. One of the key strengths of EdgeX is its ability to natively communicate using multiple protocols that are commonly used by IoT devices, including BACNET, OPC-UA, MQTT, and REST. This means that it can effortlessly receive data from a wide range of sources, no matter what protocols they use [6].

Moreover, EdgeX can be easily configured to accommodate individual data formats used by devices from different vendors. This is achieved through the use of device profiles, which are used to specify how data from each device should be handled. With this capability, EdgeX can efficiently and accurately parse data from a variety of devices, making it a truly versatile solution for IoT data integration.

In terms of its architecture, EdgeX is composed of a collection of microservices, each of which runs in a separate container. These microservices are designed to communicate with each other using REST API interfaces, allowing them to exchange data and collaborate on various tasks [12]. This architecture provides a high degree of flexibility and scalability, making it possible to tailor EdgeX to meet the needs of virtually any IoT application or use case [18].

In this paper, we are interested in the usability and performance of the EdgeX platform. Our approach involves an implementation of a sample scenario. We explore how the EdgeX can be utilized. For this, we take advantage of a Raspberry Pi device, InfluxDB, and Grafana software tools. The paper's contributions are multifaceted. Firstly, it identifies the EdgeX open edge server as a promising platform for deploying IoT services, backed by its demonstrated capability to efficiently manage web service applications (based on sensors use cases). Secondly, it underscores the increasing importance of edge computing in the evolving landscape of 5G and the forthcoming 6G networks, especially for real-time IoT services, where it significantly reduces latency and bolsters overall network efficiency. Additionally, the paper highlights the EdgeX server's value as an open-source and customizable solution, thanks to its modular architecture that facilitates the seamless integration of diverse components, making it adaptable to various use cases and ensuring compatibility with a range of IoT devices and systems. Furthermore, it recognizes the EdgeX server as a substantial advancement in the development of edge computing infrastructure, representing a significant step forward in the field.

The structure of this paper can be outlined as follows: We will delve into the relevant literature. in Section 2. The EdgeX platform and its architecture are introduced in Section 3. We show the opportunities and applied tools for the communication in Section 4. Section 5 presents how the microservice architecture is taken advantage of by the EdgeX. Later, in Section 6, we start to use EdgeX and we present how to create a device in EdgeX. We present the details of the a web service's deployment and utilization in Section 7. We implement a sample scenario and evaluate the performance of EdgeX in Section 8. Finally, Section 9 concludes the paper.

2. Related work

The rapid growth of industrial IoT (Internet of Things) applications has driven the development of edge computing, a paradigm where data processing occurs at the edge of the system, closer to devices and sensors, rather than routing it to centralized data centers [9]. In this context, the role of edge gateways has become increasingly crucial as intermediaries interfacing with multiple sensors and responsible for processing data and implementing data analytics.

Traditionally, software design and development for edge gateways have been undertaken by solution architects and developers. However, this process has posed challenges, including the potential for increased development effort and costs, as well as latency issues. One significant issue that has been noted is the absence of open-source frameworks tailored to the needs of edge computing [3].

In the absence of open-source frameworks, the development of scalable edge gateway software can be time-consuming and costly. Furthermore, addressing potential software bugs and ensuring robust testing becomes a complex task. This lack of an open-source solution can stifle innovation in the field of edge computing [16].

One notable solution to mitigate these challenges is the adoption of open-source edge computing frameworks. In this context, the paper introduces EdgeX Foundry, a prominent open-source framework designed to streamline the development process for edge gateways [11]. By leveraging EdgeX Foundry, the development cost can be significantly reduced, and latency issues can be minimized.

EdgeX Foundry offers a comprehensive platform that simplifies the creating and crafting of applications tailored for edge computing capabilities. By using this framework, developers can harness the power of container orchestration, such as Kubernetes, to enhance the scalability and efficiency of their edge gateway software [1].

However, It is essential to acknowledge e that despite the increasing adoption of edge computing and open-source frameworks like EdgeX Foundry, there remains a notable gap in the literature regarding the challenges and opportunities specific to this domain [10]. While previous research has explored the general benefits of edge computing, including reduced latency and improved network efficiency, few studies have delved into the practicalities of designing and implementing scalable edge gateway software using open-source solutions.

Several research articles have been published on the topic of EdgeX, covering various aspects such as security, interoperability, implementation, container orchestration, and comparative analysis with other open-source edge computing systems. In this section, we will briefly review some of the significant works on EdgeX.

John et al. propose the use of domain-specific languages (DSLs) for developing secure and interoperable automation systems using EdgeX [5]. The authors demonstrate the use of DSLs to model the system's security and interoperability requirements, providing a more efficient and effective way to develop EdgeX-based systems.

Kwon et al. describe the implementation of an IoT control system based on EdgeX [6]. The authors present a case study of an industrial IoT system and demonstrate how EdgeX can be used to enable interoperability and integration of devices from different manufacturers and technologies.

Lee et al. discuss the use of Kubernetes for container orchestration in EdgeX [7]. The authors demonstrate how Kubernetes can be used to manage EdgeX services and containers, enabling scalable and efficient deployment of edge computing solutions.

Liang et al. compare EdgeX with other open-source edge computing systems, including Apache IoTDB, OpenFog, and FogFlow [8]. The authors evaluate the systems based on various criteria such as architecture, data processing, and interoperability and provide a comparative analysis of the systems.

Han et al. propose an open framework of a gateway monitoring system for IoT in edge computing [4]. The authors describe the system's architecture and present a case study of its implementation, demonstrating its effectiveness in managing IoT devices and services at the edge.

Villali et al. provide an overview of open-source solutions for edge computing, including EdgeX, [18]. The authors discuss the challenges and opportunities of edge computing and review the various open-source solutions available for developing edge computing solutions.

Finally, Prabhu and Hanumanthaiah propose an EdgeX-based healthcare framework for providing telehealth services [13]. The authors describe the system's architecture and present a case study of its implementation, demonstrating its potential in providing remote healthcare services.

The above works demonstrate the growing interest and research in EdgeX and its potential for developing scalable, interoperable, and secure edge computing solutions.

3. EdgeX Server

EdgeX is an IoT edge computing platform that is built on open-source software [5]. Its purpose is to act as a mediator between the cloud and IoT devices, which are sometimes referred to as edge nodes, and to facilitate communication between them. EdgeX Foundry stands out due to its open-source, modular, and interoperable nature, as well as its cost-effectiveness and strong community support. When comparing it to existing edge computing systems, these advantages make EdgeX Foundry a compelling choice for many IoT and edge computing projects, especially when seeking flexibility, scalability, and reduced development costs. In practical terms, EdgeX can perform a variety of functions, such as collecting, storing, analyzing, Collecting data from IoT devices and transmitting it to a designated endpoint. It also allows users to monitor device data and control IoT devices directly from the platform. Tabel 1 shows the Environment Requirements.

EdgeX comprises four core layers: 1) Device Services, 2) Core Services, 3) Supporting Services, and 4) Export Services. The Device Services layer plays a pivotal role in interfacing with IoT devices, abstracting their connectivity protocols, and facilitating data interaction. Within the EdgeX ecosystem, microservices have the capability of requesting data from IoT devices or transmitting data to other microservices., including those situated in the Core Services layer. The second layer serves as a middleman bridge, connecting the first layer with the upper layers of the platform,

Environment Requirements
• Ubuntu 20.04 (preferably a VM or WSL)   – This work uses Ubuntu 20.04 : WSL.   • Internet access (for downloading container images and sending data via MQTT)   • Wifi Router.   • For IoT devices: – Raspberry Pi (a DHT sensor to send data to EdgeX)

Table 1. environment Requirements for EdgeX

thus ensuring seamless communication and functionality. The Core Services layer of EdgeX serves as a repository for initial information and sensor data pertaining to IoT devices linked to edgex nodes, storing this data in a local database until it is ready for transmission to higher-level layers or cloud systems. Additionally, this layer manages essential information about microservices, including details including attributes like hostname and port number, and facilitates connections between these microservices. When one of the microservices requires a connection to another, it connects with the Core Services layer to gather the necessary data for establishing the connection.

Moving on to the Supporting Services layer, it offers a range of standard software functionalities, encompassing tasks such as managing schedules, handling notifications, and managing alerts. This layer empowers individual microservices to execute designated tasks at specific times or in accordance with predefined rules. Moreover, it plays a crucial role in collecting and managing logs within the EdgeX environment, archiving them in either log files or databases, and disseminating notifications or alerts to external systems or users interconnected with EdgeX.

Lastly, the last layer within EdgeX houses an assemblage of functions that systematically handle messages, following a predefined function pipeline concept. These functions prepare data by applying transformations and filters, format it through reformatting, compression, and encryption procedures, and ultimately export the information to an external service specified as an endpoint.

4. Interacting with the Edge

EdgeX provides multiple options for interacting with its services, including the use of command-line tools like curl and graphical user interface (GUI) tools like Postman. Both of these tools allow users to make HTTP requests to EdgeX's REST APIs and receive responses in JSON format. The topology of the environment is presented in Fig. 1.

Curl is a command-line tool that is widely used for sending HTTP requests to servers. It is available on most operating systems and can be used to interact with EdgeX's REST APIs by sending GET, POST, PUT, and DELETE requests. The syntax for using curl is relatively simple, and it can be used to test EdgeX's APIs quickly.

For example, to retrieve a list of all the devices registered with EdgeX, a user can execute the following command using curl:

```
curl -X GET http://localhost:48081/api/v1/device
```

Table 2. Retrieve a list of all the devices

This command sends a GET request to EdgeX's device API, requesting a list of all registered devices. The response is sent back in JSON file format which can be easily parsed, using for example Postman, by the useres.

Postman, on the other hand, is a graphical user interface tool that allows users to send HTTP requests and view responses in a more user-friendly way. It can be used to send requests to EdgeX's REST APIs, and the response can be viewed in either JSON or XML format.

To use Postman with EdgeX, a user must first create a new request and specify the HTTP method (GET, POST, PUT, or DELETE) and the API endpoint. The payload can be expressed in JSON format by selecting the "raw" data type and specifying the payload in the request body.

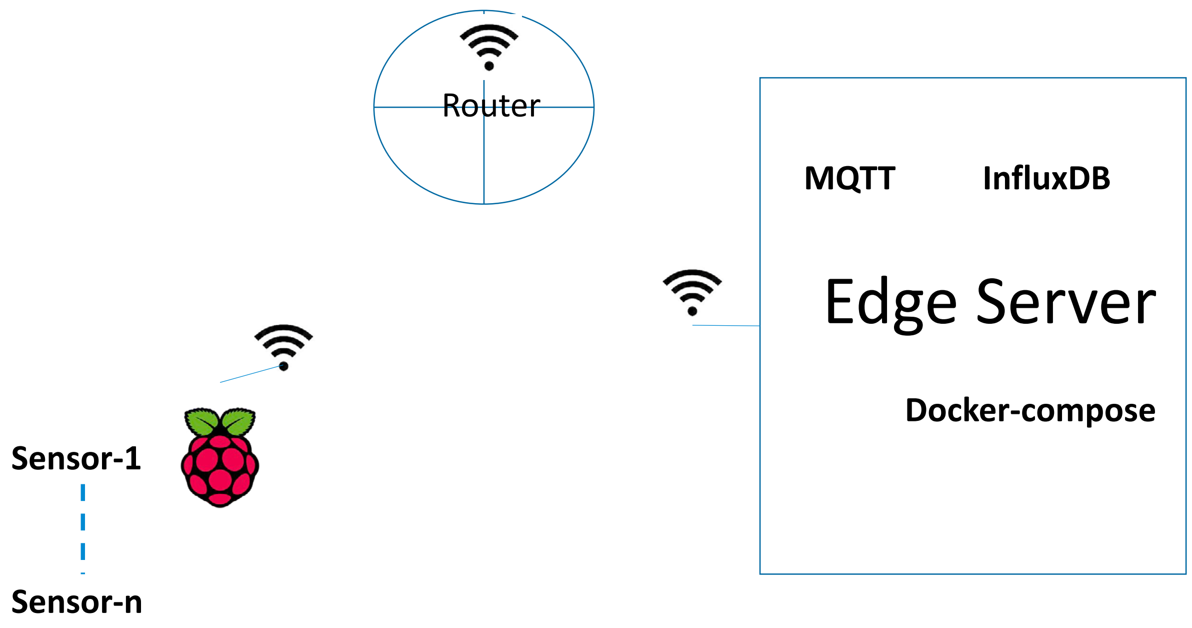


Fig. 1. Network Topology Architecture

For example, to add a new device to EdgeX, a user can create a new POST request in Postman and specify the following endpoint:

```
http :// localhost :48081/ api / v1 / device
```

`http://localhost:48081/api/v1/device`

Table 3. Add a new device to EdgeX

The user can then select the “raw” data type and specify the payload in JSON format, such as:

```
{
    ” name ”: ” New−Device ”,
    ”description”: ”New IoT device”,
    ”profileName”: ”New IoT Profile”,
    ”serviceName”: ”new−device−service”,
    ”labels”: [
        ”test −1”
    ]
}
```

`" name ": "New-Device", "description": "New IoT device", "profileName": "New IoT Profile", "serviceName": "new-device-service", "labels": [ "test" ]`

Table 4. Retrieve a list of all the devices

This payload specifies the name, description, profile name, service name, and labels for the new device.

In conclusion, both curl and Postman provide easy-to-use options for interacting with EdgeX's REST APIs. Users can express the payload in JSON format, making it easy to create and parse data. These tools make it possible for developers to quickly test and integrate EdgeX with other applications and systems.

5. Microservices concept in the Edge

EdgeX is a highly modular and scalable open-source IoT platform that is composed of multiple microservices. These microservices are designed to work together seamlessly and provide a range of functionalities, including data ingestion, storage, processing, and analytics. They are built on top of a containerization technology called Docker, which allows them to be easily deployed and scaled across different environments.

Docker Compose is a tool that is used to manage multiple containers running microservices [14]. It allows users to define the configuration and dependencies of the containers in a YAML file, making it easy to deploy and manage EdgeX on different machines.

Some of the key microservices that are included in EdgeX are:

1. Core Data: This microservice is responsible for ingesting data from various sources and storing it in a database. It supports multiple database systems, including MongoDB and Redis, and can be configured to store data in different formats.
   For example, to deploy the Core Data microservice using Docker Compose, a user can define the following configuration in a YAML file:

```
version: ”3”

 Services :
  core - data:
    image: edgex /docker - core - data-go
    ports:
      - 48080:48080
    Environment:
      SERVICE_HOST: core-data
      DATABASES_PRIMARY_HOST: mongo
      DATABASES_PRIMARY_PORT: 27017
```

   This configuration specifies the image to be used, the port mapping, and the environment variables required to run the Core Data microservice.
2. Rules Engine: This microservice is responsible for processing and analyzing data using predefined rules and algorithms. It allows users to define custom rules based on different criteria, such as time, data values, and device attributes.
   For example, to deploy the Rules Engine microservice using Docker Compose, a user can define the following configuration in a YAML file:

```
version: ”3”

services:
  rules-engine:
    image: edgex /docker-rulesengine-go
    ports:
      - 48099:48099
    environment:
      SERVICE_HOST: rules-engine
      COREDATA_HOST: core-data
      COREDATA_PORT: 48080
```

This configuration specifies the image to be used, the port mapping, and the environment variables required to run the Rules Engine microservice.

3. Export Services: These microservices are responsible for exporting data to external systems or applications. They support multiple protocols, including MQTT, REST, and CoAP, and can be easily configured to send data to different destinations.
   For example, to deploy the Export MQTT microservice using Docker Compose, a user can define the following configuration in a YAML file:

```
version: "3"

services:
  export-mqtt:
    image: edgex /docker-export-mqtt-go
    ports:
      - 49990:49990
    environment:
      SERVICE_HOST: export-mqtt
      MQTT_BROKER_ADDRESS: tcp://mqtt-broker:1883
      MQTT_CLIENT_ID: edgex-export-mqtt
```

   This configuration specifies the image to be used, the port mapping, and the environment variables required to run the Export MQTT microservice.

According to this section, one can say, EdgeX's microservices architecture, combined with Docker Compose, provides a highly modular and scalable IoT platform that can be easily customized and extended to meet different use cases and requirements. By deploying and managing the microservices using Docker Compose, users can easily set up and run EdgeX on different machines and environments.

6. Connecting to the Edge: Creating a Device in EdgeX

Edge computing has gained popularity due to its ability to manage processing and storing data nearer to the origin, reducing latency and improving overall performance [17]. EdgeX is a server with a feature of open-source that provides a framework for developing and deploying IoT solutions at the edge. Here in the following subsections, we explore the operation of creating an IoT device in EdgeX.

6.1. Device Profiles and Value Descriptors

To create a device in EdgeX, a device profile must first be created. The profile of a device is a description of the device and its abilities. It includes details like the device name, manufacturer, model number, and the types of data that the device can send and receive. In addition to this, value descriptors define the format and data type of each data point the device sends or receives.

Device profiles and value descriptors are crucial components of EdgeX, as they allow devices to be discovered and communicated with, regardless of the data they produce or the communication protocol they use. This interoperability is a fundamental aspect of EdgeX and enables seamless integration of devices from different manufacturers and technologies.

6.2. Creating a Device Profile

The EdgeX Device Profile Editor is a tool that allows for the creation of a device profile. The editor provides an intuitive interface that allows for the selection of the device type, defining the device characteristics, and specifying the data formats and protocols used by the device. It also enables the creation of value descriptors for each data point produced by the device.

Once the device profile is created, it is saved as a JSON file and can be used to register and connect the device to EdgeX.

6.3. *Creating the Device*

The next step in the process is to create the device itself. This is done by implementing the device driver, which is a software component that allows the device to communicate with EdgeX. The driver translates the device data into a format that can be understood by EdgeX.

To implement the device driver, the EdgeX Device SDK is used. This SDK provides a set of APIs that allow for interaction with EdgeX and communication with other devices and applications.

Once the device driver is implemented, the device can be tested using the EdgeX Device Service. This service provides a way to discover, register, and manage devices within EdgeX. The Device Service can be used to connect the device to EdgeX and test its functionality.

As a conclusion, creating a device in EdgeX requires a combination of hardware and software skills. However, the platform provides a framework for developing and deploying IoT solutions at the edge, allowing for the creation of interoperable devices from various manufacturers and technologies. By creating a device profile, defining value descriptors, and implementing the device driver, devices can be connected to EdgeX and contribute to the creation of a more connected, intelligent, and efficient edge ecosystem.

7. Deploying and Testing a Web Service

To further explore the capabilities of the EdgeX platform, we decided to deploy a web service that returns the IP address of the client when accessing it through a web page. We accomplished this by creating a simple web application and deploying it to EdgeX using the docker-compose file. The application was defined to run on port 8080, which allowed it to communicate with the edge devices.

After deploying the web service, we tested it by accessing the web page from a device connected to the EdgeX network. Specifically, we accessed the web page from a Raspberry Pi device connected to the EdgeX network. To our delight, the web application returned the IP address of the Raspberry Pi, proving that the web service was properly deployed and functioning as intended.

Overall, this successful deployment and testing of a web service on the EdgeX platform demonstrated the platform's ability to support and manage distributed applications, while also showcasing its flexibility and ease of use. Fig. 2 illustrates the success of the deployment and testing.

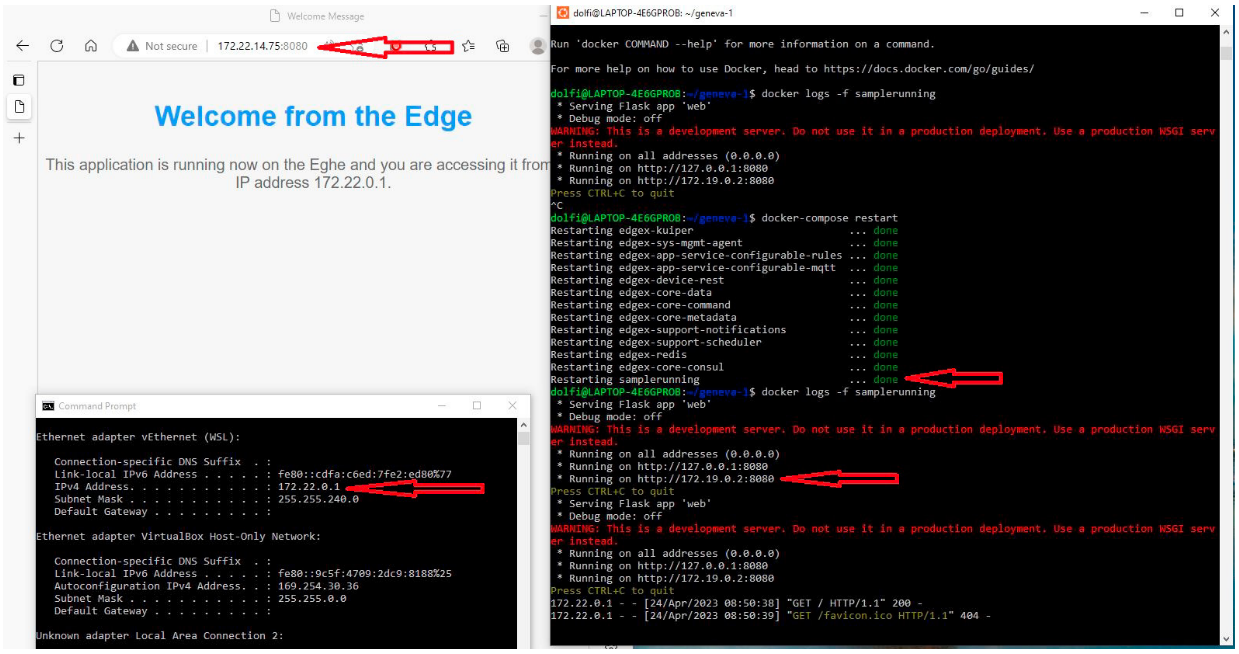
We conducted a deployment of the identical web service application on both the AWS server to meticulously assess latency within the context of EdgeX and AWS. As depicted in Fig 3, the results illustrate that under EdgeX, the latency is nearly negligible, in stark contrast to the AWS server, where a discernible delay is evident.

8. Reimplementation and Evaluation of the EdgeX Example Scenario

We implemented and evaluated the EdgeX example scenario that involves two sensors, one for temperature and one for humidity. This scenario reads and sends the data to the EdgeX platform, where the data is stored in InfluxDB by Mosquitto and displayed in real-time using Grafana. Our implementation followed the instructions provided in the EdgeX GitHub site[1], but we re-implemented the scenario to test the platform's capabilities as a potential environment for further work.

Fig. 4 presents the Linux terminal of the generated data, another terminal with the logs of one of the dockers, and the graphs from Grafana. The figure provides a clear visualization of the data and demonstrates the platform's ability to manage and display real-time sensor data.

[1] https://github.com/edgex



Welcome from the Edge

This application is running now on the Eghe and you are accessing it from IP address 172.22.0.1.

```
Command Prompt

Ethernet adapter vEthernet (WSL):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::cdfa:c6ed:7fe2:ed80%77
   IPv4 Address. . . . . . . . . . . : 172.22.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::9c5f:4709:2dc9:8188%25
   Autoconfiguration IPv4 Address. . : 169.254.30.36
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . :

Unknown adapter Local Area Connection 2:
```

```
dolfi@LAPTOP-4E6GPROB: ~/geneva-1

Run 'docker COMMAND --help' for more information on a command.

For more help on how to use Docker, head to https://docs.docker.com/go/guides/

dolfi@LAPTOP-4E6GPROB:~/geneva-1$ docker logs -f samplerunning
 * Serving Flask app 'web'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serv
er instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://172.19.0.2:8080
Press CTRL+C to quit
^C
dolfi@LAPTOP-4E6GPROB:~/geneva-1$ docker-compose restart
Restarting edgex-kuiper                          ... done
Restarting edgex-sys-mgmt-agent                  ... done
Restarting edgex-app-service-configurable-rules ... done
Restarting edgex-app-service-configurable-mqtt  ... done
Restarting edgex-device-rest                     ... done
Restarting edgex-core-data                       ... done
Restarting edgex-core-command                    ... done
Restarting edgex-core-metadata                   ... done
Restarting edgex-support-notifications           ... done
Restarting edgex-support-scheduler               ... done
Restarting edgex-redis                           ... done
Restarting edgex-core-consul                     ... done
Restarting samplerunning                         ... done
dolfi@LAPTOP-4E6GPROB:~/geneva-1$ docker logs -f samplerunning
 * Serving Flask app 'web'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serv
er instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://172.19.0.2:8080
Press CTRL+C to quit
 * Serving Flask app 'web'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serv
er instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://172.19.0.2:8080
Press CTRL+C to quit
172.22.0.1 - - [24/Apr/2023 08:50:38] "GET / HTTP/1.1" 200 -
172.22.0.1 - - [24/Apr/2023 08:50:39] "GET /favicon.ico HTTP/1.1" 404 -
```

Fig. 2. Web-Service application deployment using port 8080 to return the IP address of the connected client when requested

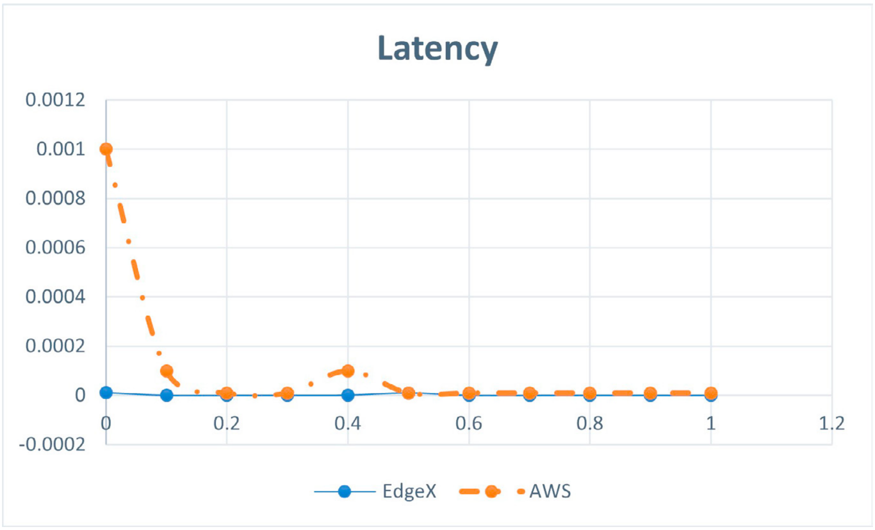
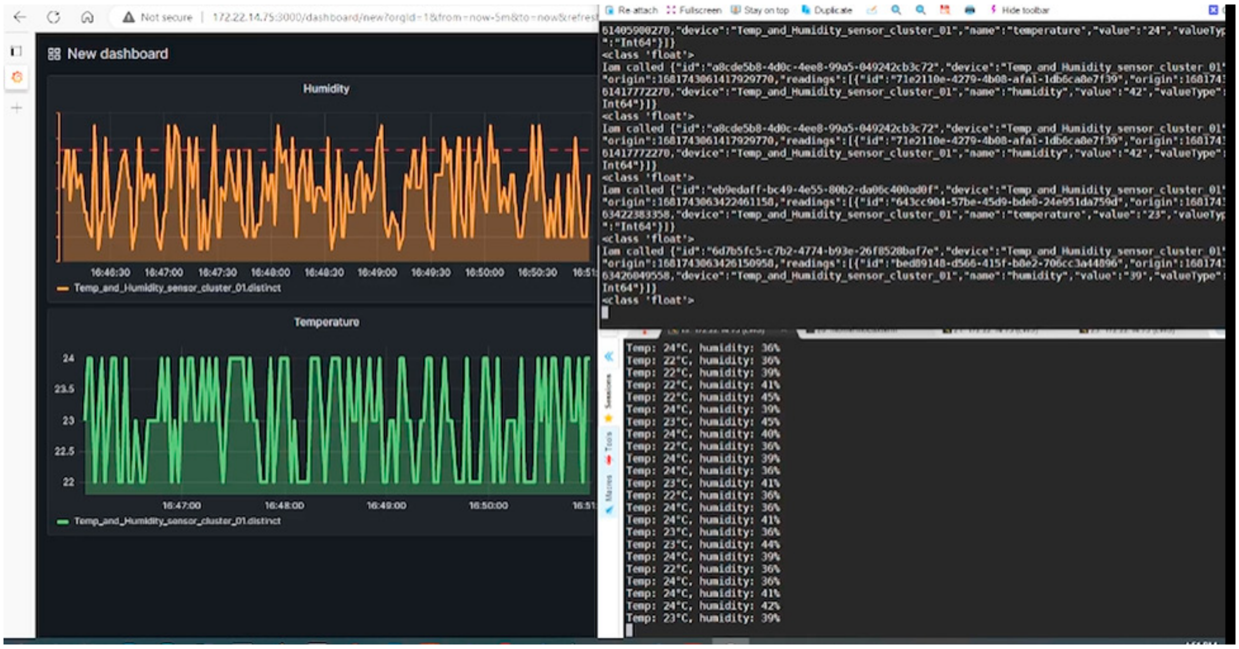


Fig. 3. Comparison of Application Deployment: EdgeX vs. AWS

Our implementation and evaluation of the EdgeX example scenario show the platform's potential to manage and display sensor data in real-time. This provides a foundation for further work in edge computing, where real-time data management and visualization are crucial for successful implementations.



```
61405900270,"device":"Temp_and_Humidity_sensor_cluster_01","name":"temperature","value":"24","valueTyp
":"Int64"}]}
<class 'float'>
Iam called {"id":"a8cde5b8-4d0c-4ee8-99a5-049242cb3c72","device":"Temp_and_Humidity_sensor_cluster_01"
"origin":1681743061417929770,"readings":[{"id":"71e2110e-4279-4b08-afa1-1db6ca8e7f39","origin":168174
61417772270,"device":"Temp_and_Humidity_sensor_cluster_01","name":"humidity","value":"42","valueType":
Int64"}]}
<class 'float'>
Iam called {"id":"a8cde5b8-4d0c-4ee8-99a5-049242cb3c72","device":"Temp_and_Humidity_sensor_cluster_01"
"origin":1681743061417929770,"readings":[{"id":"71e2110e-4279-4b08-afa1-1db6ca8e7f39","origin":168174
61417772270,"device":"Temp_and_Humidity_sensor_cluster_01","name":"humidity","value":"42","valueType":
Int64"}]}
<class 'float'>
Iam called {"id":"eb9edaff-bc49-4e55-80b2-da06c400ad0f","device":"Temp_and_Humidity_sensor_cluster_01"
"origin":1681743063422461150,"readings":[{"id":"643cc904-57be-45d9-bde0-24e951da759d","origin":168174
63422383358,"device":"Temp_and_Humidity_sensor_cluster_01","name":"temperature","value":"23","valueTyp
":"Int64"}]}
<class 'float'>
Iam called {"id":"6d7b5fc5-c7b2-4774-b93e-26f8528baf7e","device":"Temp_and_Humidity_sensor_cluster_01"
"origin":1681743063426150958,"readings":[{"id":"bed89148-d566-415f-b8e2-706cc3a44896","origin":168174
63426049558,"device":"Temp_and_Humidity_sensor_cluster_01","name":"humidity","value":"39","valueType":
Int64"}]}
<class 'float'>
```

```
Temp: 24°C, humidity: 36%
Temp: 22°C, humidity: 36%
Temp: 22°C, humidity: 39%
Temp: 22°C, humidity: 41%
Temp: 22°C, humidity: 45%
Temp: 24°C, humidity: 39%
Temp: 23°C, humidity: 45%
Temp: 24°C, humidity: 40%
Temp: 22°C, humidity: 36%
Temp: 24°C, humidity: 39%
Temp: 24°C, humidity: 36%
Temp: 23°C, humidity: 41%
Temp: 22°C, humidity: 36%
Temp: 24°C, humidity: 36%
Temp: 24°C, humidity: 41%
Temp: 23°C, humidity: 36%
Temp: 23°C, humidity: 44%
Temp: 24°C, humidity: 39%
Temp: 22°C, humidity: 36%
Temp: 24°C, humidity: 36%
Temp: 24°C, humidity: 41%
Temp: 24°C, humidity: 42%
Temp: 23°C, humidity: 39%
```

Fig. 4. Linux terminal for the generated data, Linux terminal for the Docker logs, and figures generated by Grafana container

9. Conclusion

In summary, the EdgeX open edge server emerges as a promising platform for deploying IoT services at the edge. We have substantiated its capabilities through the deployment of a web service application and the monitoring of connected IP addresses, which affirm its reliability and efficiency for IoT services.

As 5G and 6G networks advance, the significance of edge computing escalates. Its capacity to process data closer to the source becomes pivotal, reducing latency and enhancing network efficiency. This is particularly critical in the realm of IoT services, where real-time data processing is imperative.

The EdgeX open edge server stands out as an open-source, customizable solution for edge computing. Its modular architecture facilitates seamless integration of diverse components, rendering it adaptable to various use cases. Furthermore, its support for multiple protocols and standards ensures smooth interoperability across a diverse array of IoT devices and systems.

In essence, the EdgeX open edge server signifies a substantial leap forward in the evolution of edge computing infrastructure. As the demand for IoT services continues to surge, solutions like the EdgeX open edge server will play a pivotal role in enabling efficient and effective edge computing solutions.

In our forthcoming efforts, we intend to broaden our research by deploying the application with EdgeX into the Akranio solution and testing it in a blueprint solution. This will further validate the applicability and scalability of EdgeX in complex edge computing environments, providing valuable insights for the development of robust and comprehensive edge solutions in the future.

Acknowledgment

Supported by the NKP-22-4 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund. Also, Development and Innovation Fund. Also, the project was executed with the support of Ericsson.

References

[1] Chen, S., Zhou, M., 2021. Evolving container to unikernel for edge computing and applications in process industry. Processes 9, 351.
[2] Dhulfiqar, A., Pataki, N., 2023. Mec – applications deployment and tcp testing using simu5g, in: 2023 International Conference on Software and System Engineering (ICoSSE), pp. 38–43. doi:10.1109/ICoSSE58936.2023.00015.
[3] Du, J., Xu, M., Gill, S.S., Wu, H., 2023. Computation energy efficiency maximization for intelligent reflective surface-aided wireless powered mobile edge computing. IEEE Transactions on Sustainable Computing .
[4] Han, J.H., Kim, J.R., Kim, Y., Kim, H., 2020. Open framework of gateway monitoring system for internet of things in edge computing. IEEE Access 8, 122243–122253.
[5] John, J., Ghosal, A., Margaria, T., Pesch, D., 2021. DSLs for model driven development of secure interoperable automation systems with EdgeX Foundry, in: 2021 Forum on specification & Design Languages (FDL), pp. 1–8. doi:10.1109/FDL53530.2021.9568378.
[6] Kwon, M., Lee, S., 2018. Implementation of IoT control system based on EdgeX Foundry. IEEE Internet of Things Journal 5, 3595–3602.
[7] Lee, S., Phan, L.A., Park, D.H., Kim, S., Kim, T., 2022. EdgeX over Kubernetes: Enabling container orchestration in EdgeX. Applied Sciences 12. URL: https://www.mdpi.com/2076-3417/12/1/140, doi:10.3390/app12010140.
[8] Liang, X., Chen, Z., Zheng, Z., Hu, Y., 2019. A comparative research on open-source edge computing systems. Journal of Cloud Computing 8, 1–20.
[9] Mazzei, D., Baldi, G., Fantoni, G., Montelisciani, G., Pitasi, A., Ricci, L., Rizzello, L., 2020. A blockchain tokenizer for industrial iot trustless applications. Future Generation Computer Systems 105, 432–445.
[10] Mishra, S., Gupta, A., Jairam Naik, K., 2023. Efficient computing resource sharing for mobile edge-cloud computing networks, in: Frontiers of ICT in Healthcare: Proceedings of EAIT 2022. Springer, pp. 523–537.
[11] Murshed, M.S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F., 2021. Machine learning at the network edge: A survey. ACM Computing Surveys (CSUR) 54, 1–37.
[12] Newman, S., 2015. Building Microservices. O'Reilly.
[13] Prabhu, R., Hanumanthaiah, D., 2022. Edge computing-enabled healthcare framework to provide telehealth services. IEEE Transactions on Industrial Informatics .
[14] Révész, Á., Pataki, N., 2017. Containerized A/B testing, in: Budimac, Z. (Ed.), Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, CEUR-WS.org. pp. 14:1–14:8. URL: http://ceur-ws.org/Vol-1938/paper-rev.pdf.
[15] Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50, 30–39. doi:10.1109/MC.2017.9.
[16] Taherizadeh, S., Jones, A.C., Taylor, I., Zhao, Z., Stankovski, V., 2018. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. Journal of Systems and Software 136, 19–38.
[17] Zhang, Y., Lan, X., Ren, J., Cai, L., 2020. Efficient computing resource sharing for mobile edge-cloud computing networks. IEEE/ACM Transactions on Networking 28, 1227–1240. doi:10.1109/TNET.2020.2979807.
[18] Zhao, L., Zhou, G., Zheng, G., Chih-Lin, I., You, X., Hanzo, L., 2021. Open-source multi-access edge computing for 6g: Opportunities and challenges. IEEE Access 9, 158426–158439.