An embedded low-cost solution for a fog computing device on the Internet of Things

An embedded low-cost solution for a fog computing device on the Internet of Things

Pietro d'Agostino
Department of Control and Computer Engineering (DAUIN)
Politecnico di Torino
Torino, Italy
pietro.dagostino@polito.it

Massimo Violante
Department of Control and Computer Engineering (DAUIN)
Politecnico di Torino
Torino, Italy
massimo.violante@polito.it

Gianpaolo Macario
AROL Closure Systems
Torino, Italy
gianpaolo.macario@arol.com

***Abstract*— The adaptability of devices can be significant for a customer that inserts them in an industrial production line. The ability to modify an object bought along with a machine that can be personalized with its features can change how they want to do measurements for different reasons, like predictive maintenance. Fog computing local centers already exist in the market, but they are usually on-the-shelf products with no margin of change for any user. However, with the usage of Docker and containers, this can change. This paper describes a fog computing local central called Concentrator, which can not only execute its essential functions built-in by the producer but also be customized by the user to add in the elaborations on other external sensors, expanding its capabilities and usage. We wanted to improve the device already tested on a Linux PC on a Raspberry Pi and try its performance and characteristics, seeing if it could be transformed into an embedded architecture and an industrial feature.**

Keywords*— *Arduino, container, docker, industrial automation, IoT, embedded systems

I. Introduction

Industry 4.0, known as the Fourth Industrial Revolution, reshapes the manufacturing landscape. It represents the convergence of digital technologies, automation, and data-driven processes to create smart, interconnected, and efficient manufacturing methods. This revolution is characterized by integrating physical production systems with digital technologies, enabling cyber-physical systems (CPS) creation. These use real-time data from sensors, actuators, and various connectivity for the different stages of the production process, increasing predictive maintenance and adaptive manufacturing [1].

At his core is the Internet of Things (IoT), in which machines, devices, and sensors are interconnected, creating a network in which data are shared across the production floor. This integration allows systems to communicate, collaborate, and make autonomous decisions [2]. New business models can be created with these properties, which develop into customer satisfaction, opening new revenue streams and market opportunities. However, there are some challenges that manufacturers must address: data security, privacy, and intellectual property. Also, adapting the workforce to digital transformation can be a significant hurdle. Standardization technologies, interoperability, and a profound renovation of establishments are crucial for adopting the Fourth Industrial Revolution [3].

In this view of interoperability, new systems have been created to make the different devices communicate between them, particularly the Cloud, Edge, and Fog computing. These are transformative technologies for different applications that revolutionize how businesses and individuals access computing resources.

Cloud computing enables organizations to reduce costs, improve flexibility, and enhance efficiency by providing scalable, on-demand, virtualized services over the Internet. Scalability is one of the key characteristics since resources can easily be scaled up or down depending on the usage and the organization's needs. This elasticity enables cost savings and ensures optimal resource optimization [4]. There are three main models that the industry can develop: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The first provides virtualized computing resources to build and manage their applications and environments, the second offers the development and deployment of a platform without worrying the user about infrastructure management, and the third provides complete software applications, eliminating the need for local installation and maintenance [5]. These models can be deployed in public clouds, private or hybrid, combining both to leverage both benefits, maintaining flexibility and security. This is an important topic for this kind of system since companies usually rely on third-party providers and the transfer of sensitive data on the Internet. Performance and latency issues may occur when accessing cloud services, requiring consideration of network connectivity and service level agreements [4].

Edge computing is another system that addresses computation, storage, and analytics requirements closer to the data source. It is a distributed paradigm that shifts the resources from the centralized cloud servers to the network edge, enabling real-time data processing, reduced latency, and improved application performance. This proximity to the sensors also minimizes network congestion, bandwidth usage, and dependence on cloud connectivity. The fundamental principle is to reduce time latency by avoiding the round-trip to cloud servers. It also enables offline operations and local processing and storage capabilities, enhancing privacy and reducing exposure to potential cyber threats [6].

The last paradigm is fog computing, which extends cloud computing capabilities to the network's edge and enables localized data processing, real-time analytics, and improved

efficiency. It takes both the benefits of cloud and edge computing and combines them. Fog computing shifts the cloud-based resources for data processing and storage toward the edge but not on the system's limit. It uses a series of local centers called fog nodes, routers, switches, or similar devices. This enables real-time operations, minimizes latency, and enhances overall system performance. Fog computing addresses the challenges posed by the influx of IoT devices by distributing resources along the system close to the edge. For this reason, it improves the efficiency and reliability of IoT applications, elaborating the data from the nodes and sending them to the cloud servers, reducing bandwidth usage, congestion, and potential failures. This localized processing also enhances privacy and security, as sensitive data can be processed and stored closer to its source, minimizing the risk of data breaches and ensuring compliance with data protection regulations [7].
However, fog computing has challenges. The distributed nature of fog deployments introduces complexities in managing and orchestrating resources, ensuring interoperability between heterogeneous devices, and handling intermittent connectivity. Additionally, fog nodes' dynamic and potentially resource-constrained nature requires efficient resource allocation, load balancing, and fault tolerance mechanisms [8]. The main actors of a fog computing system are the local centrals, which must gather and elaborate the data taken from the edge sensors.

This paper focuses on developing a fog central called Concentrator, created by the collaboration between Politecnico di Torino and AROL Closure Systems, a company producing capping machines. Apart from a default functioning, which consists of predictive maintenance of the different devices, the idea is to add a containerized sandbox in which a general customer can add methods or purposes by using third-party software or its own without affecting the basic functionalities. This is done by implementing one or more containers through a virtualization system like Docker, in which reports showing the data of interest are produced.

The contribution of this paper is to create this sandbox inside of an embedded on-the-shelf product, like a Raspberry Pi board, in which a customizable network can be deployed. The AROL assets are used to test and evaluate the proposed solution experimentally, obtaining data and simulating a first instance of the final implementation.

The presentation is organized as follows. Section II shows state-of-the-art and a series of papers we used to study and develop the idea in the fog computing context. Section III presents the architecture, detailing how the Concentrator has been developed and which node sensors have been chosen. Section IV offers the experiment conducted, and finally, section V draws some conclusions.

II. Related works

The usage of containers and virtualization in cloud computing systems has been discussed since the first decade of the 21st century [9][10]. Recent topics have shown a significant improvement in the matter, developing them in fog computing architectures.

In [11], the authors present a technological review of different utilization of virtualization depending on the operating system used and their performances. By explaining what Docker is and how it can be used for these kinds of applications, they explained the technological requirements and architectural principles for developing an edge cloud computing system. Even though three different methods have been studied (Windows, Linux, and Cloud Platform-as-a-Service(PaaS)), the central focus is on Linux systems, which help the implementation of embedded hardware, creating a PaaS and an orchestration method to control and manage the different containers. The techniques used are Docker Swarm and Kubernetes to have an overview of how the two services work and how to implement them efficiently.

A similar approach to our architecture has been presented in [12]. An intelligent fog computing home infrastructure has been created using low-cost embedded hardware (Arduino, Raspberry Pi board, and some sensors). The network has been divided between three houses with different sensors, and a Raggedpod, a micro data center, represents the central. The virtualization is applied through Docker Linux individual containers to cope with production-grade requirements and integration of various systems.

In [13], a different approach is explained: the authors show an on-demand fog system created based on the customer's need, depending on the nearest available devices that are part of the network, to have the least initialization cost possible. Using Docker, managed by Kubeadm, they deployed on-the-fly services using orchestrators that will enter the network whenever a request is issued, handling it, and sending the data to the cloud. The latencies are reduced not only due to the availability of different devices closest to the requester, updating fog's IP to localize it, but also by creating clusters in advance that can be used directly without waiting for them.

The paper [14] exemplifies how virtualization can be applied in a dew computing system. They created a network where a software component communicates with the cloud and stores data locally composed by a virtual resource. The service accomplishes the different user requests, saturating itself. When this happens, a replica is awakened to solve those attending, giving elasticity to the network on an edge level. Since they decided not to use central gateways, the device's risk of being corrupted is relatively high, and it is an issue that must be solved. For this reason, the system's security has also been developed, creating a blockchain protocol that achieves reliability and safety on an edge level.

Paper [15] shows the two main principal frameworks that an IoT service could have: *container-based pair-oriented IoT service provisioning*, where two devices cooperate and are responsible for the interactions, and *container-based edge-managed clustering mode*, where a manager supervises the operations between the cooperating devices forming a cluster. The difference between the two is how the creation of containerization is divided between the different instruments and how they work the IoT resources in terms of CPU and energy consumption.

In [16], the authors present a technological review of different utilization of virtualization depending on the operating system used and their performances. By explaining what Docker is and how it can be used for those kinds of applications, they created a test tool utilizing a program to evaluate the speed of arithmetic operations, the rate of working with RAM, and the speed of disk operations. They used C++ as the language and Clang++ as the compiler, performing 33 operations in total: 15 arithmetic functions with 15 assignments, 1 loop condition, 1 increment, and 1 loop variable assignment. The results showed that Linux systems perform slightly faster regarding arithmetic and memory operations than Windows.

The last two works we referred to are FogPi [17] and Con-Pi [18], describing fog computing local centrals exploiting the containerization principle through Docker for different purposes. They both chose and showed how the Raspberry Pi is a good choice for this kind of utilization, describing its suitability and performance.

FogPi is an embedded, portable, low-cost, and low-power consumption fog computing infrastructure running Docker, providing scalability and higher availability than traditional methods. The containers can be deployed using previously loaded and configured images, so it does not require an internet connection. They created this local central to control some nodes in real-time, and an evaluation of the latency given from Google Cloud concerning FogPi has been done, showing how the fog infrastructure can better manage the amount of data than the cloud method.

Con-Pi is a distributed framework to manage resources in a fog computing environment. It is based on different Small single-board computers (SBCs), but they have used some Raspberry Pis. It uses Docker to have containerization to run IoT and microservices, managing energy available in different use cases, for example, smart agriculture.
They used as hardware a Raspberry Pi, some sensors, and actuators to perceive the external world. Instead, the software is divided between a Controller, a service, and a MicroService: the first is used for resource management and control of the execution of microservices, the second understands the context information and configurations, while the third acts as an operator for the controller instructions. This configuration can lead to resource sharing: the execution of microservices and their respective containers are offloaded among RPis, while the sensor data and actuation interfaces can be accessed remotely. Also, how they managed the policy integration and their managing of different containers is very interesting. Information is taken that can be useful for future works.

These works utilize fog computing systems, Docker, and Raspberry Pi boards as local fog central, proving the idea's feasibility. However, none explained how a user could change the design to insert its paradigm or customize it depending on someone's need.

This paper presents the Concentrator, a hardware device that can be part of a fog computing system. It stores and elaborates data, sending them to the cloud. The innovation we propose regards the possibility of adding third-party applications to the system without interfering with the basic functionalities of the system, to be customizable for any user that would like to use it. We decided to use a virtualization system to create two domains: a native and a host. The native is dedicated to the producer and must contain the default application for which the Concentrator exists. The host, instead, can be modified by the customer to add services that can be useful for a specific application. The configuration is suggested by looking at the related work [15] since there are multiple nodes, and the IoT client must guarantee service continuity in case of failure. This approach can ensure a fast management procedure through direct interaction between the cooperating nodes.

Virtualization is used to create a sandbox where customers can modify a central device depending on their needs. Starting from the same operating system, the different applications will not interfere with each other, offering a secure implementation inside their production line.

III. Methodology and Architecture

This section presents the proposed system architecture, illustrated in Fig. 1. This figure has been retrieved by [19] to show a generic system in which the Concentrator is developed. It shows a fog computing system divided into three main parts: the Wide Area Network (WAN), the Local Area Network (LAN), and the Personal Area Network (PAN).

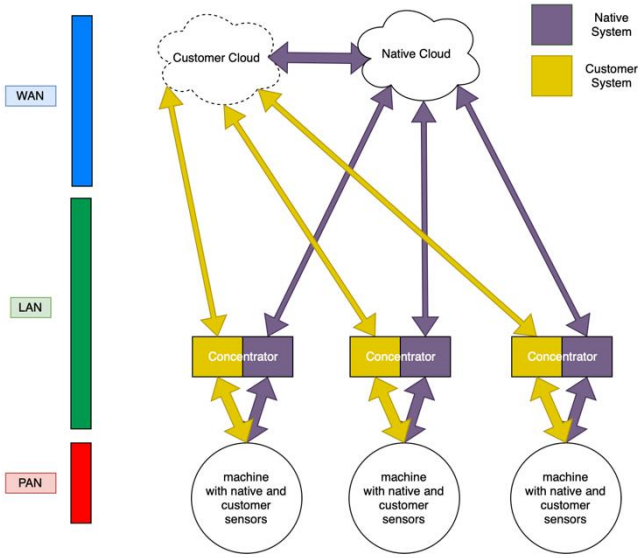

Fig. 1 - General architecture [16]

- The Wide Area Network

This layer of the network mainly focuses on the cloud architecture. It has the database role, storing all the data processing made by the central fog nodes and sending them the images requested for the native system virtualization to develop the basic functionalities they must perform. In this way, the history of every device is recorded, and these data can be used to do multiple analyses, understand whether damages can occur depending on the results taken, and

provide, if necessary, predictive maintenance services for the different machines. Since the customer can use its algorithms and sensors, the host devices' data is not stored in the leading cloud system. They will be uploaded to the customer cloud and can only be used by the producer if there is a legal agreement to share data between the two parties.

- The Local Area Network

The second layer regards the central fog nodes and the interconnections between the other layers. Those different Concentrators, one for every machine in the production line, gather the information from the peripheral fog nodes and process their data, sending them to the cloud. The producer and the customer can decide on the elaborations mentioned since they depend on what they need to perform predictive maintenance of the machines. They will also exchange information with local systems in case of error signals or urgent messages a user must notice. Its presence will allow to:

1. Reduce bandwidth usage and the number of data transmitted from the peripheral sensors to the cloud.
2. Delay reduction between when the data are gathered and when they are elaborated.
3. Better control implementation to solve warnings or other anomalies in the system.

This improves the system's usability and has better performance in terms of time since the delay is reduced, not only for the exchange of data but also for the anomalies and warnings, which can help make the architecture more secure.

The central fog devices can support different types of communication, principally ethernet, and wireless. These are shown in Fig. 2. The Concentrator, from [19], does not change the kind of information sent to the other components of the system, just how it is made and configured. The ethernet is used for registering the system's performance (number of pieces produced, machine efficiency, number of functioning hours, etc.) by other production line components. It is also linked to a local system (like a PLC on the machine itself) to set an event-based communication for anomalies, warnings, or other messages. Instead, wireless protocols, mainly Bluetooth, are used to exchange data with the peripheral fog nodes mounted on the machines to measure different aspects of their functioning. The third kind of connection is the internet for communication with the cloud. The virtualization of the two parts of the system, the native and the host one, will prevent the exchange of unnecessary information between them, increasing the privacy and security of the data taken. An example of a program that provides this security is Docker, which lets the two containers created not interfere with each other. The Concentrator will implement a system that will make a sandbox that the customer can modify depending on its necessities and perform different measurements using his sensors. Some data are available through a shared volume, but they are decided by the native system and protected by default since the program can keep the creation of containers secure and controlled.

- The Personal Area Network

The third layer regards the peripheral fog nodes and the data measured by them. They can be provided by the producer or the customer that buys the system, adding external functionalities other than the default ones. The aim of gathering these data is to perform analyses of the machines to understand if they are efficient and working correctly and to offer a service of predictive maintenance in case the system notices some anomalies. These sensors will be placed in a critical section of a machine to measure parameters like temperatures, vibrations, or any other valuable value for studying its working life. They will use wireless communication protocols to send their data to the Concentrator. They will be initialized to save battery power and register only valuable data when the machine is powered.

IV. Experiment and Discussion

This section is dedicated to describing the experiment and the implemented scenario to prove the efficiency and feasibility of the proposed system above. It will focus only on the PAN and LAN parts, leaving the communication with the cloud to future works. The general idea is to prove the feasibility of the Concentrator device, and so the fog computing system inside an industry production line. Without this device, there would not be a fog computing system but an edge computing system. For this reason, tests to prove the efficiency of the sensors without the local center have not been executed.

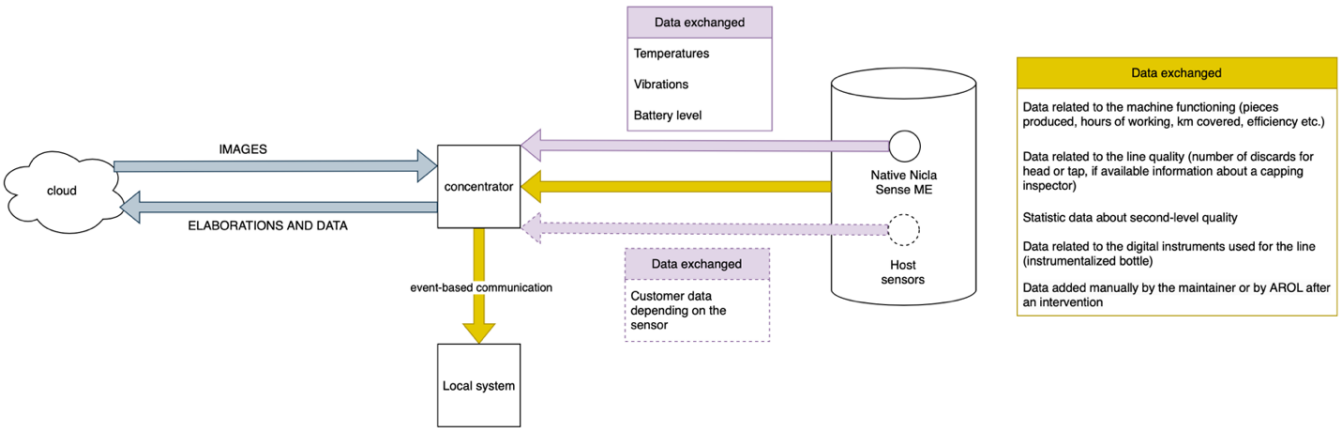

Fig. 2 - Concentrator data exchange [16]

The experiment aims to demonstrate the usability of a code that lets the creation of two containers, one called Native, which the producer of the Concentrator gives, and one called Host, provided by the customer using the device.
To do so, Docker has been chosen as an instrument for virtualization, particularly Docker Compose, which permits the creation of multiple containers and running them simultaneously. The two containers have a shared volume in which some data are published to make them communicate and exchange information. The peripheral fog node chosen for the experiments is the Nicla Sense ME from Arduino [20]. These sensors have been initialized to perform, one of them as host and three of them as native sensors, to try to simulate as best as possible a possible actual implementation.

The Nicla Sense ME is a small, low-cost, low-power device that combines four state-of-the-art sensors from Bosch Sensortec. It has the ME acronym, which means "Motion" and "Environment" since it can measure rotations, accelerations, temperatures, humidity, pressure, air quality, and CO2 levels with an industrial grade of precision. This board is their most minor form yet and can exchange data through Bluetooth Low Energy connectivity (version 4.2) through an ANNA-B112 module.

Regarding the Concentrator, we wanted to exploit the system on embedded hardware, so we chose a Raspberry Pi four model B. It runs Raspberry Pi OS 64-bit (based on Debian release 11), where the instances of Docker compose will be implemented using JavaScript and node-ble. The version of Docker is 24.0.0, Docker compose is 2.17.2, the BlueZ library is 5.55, and the Node.js library is 16.20.0.
The experiment is described in Fig. 3. It consists of creating the two images for the containers and building them.

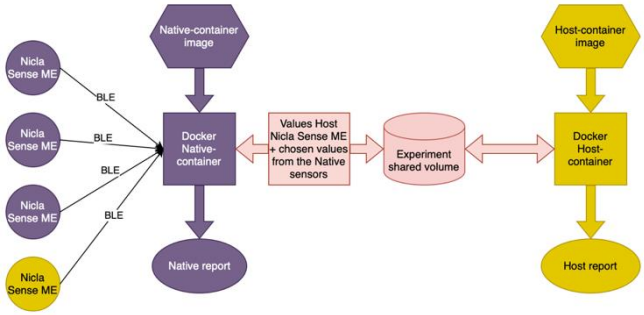

Fig. 3 - Experiment representation

The first container, called *Native*, represents the one provided by the producer, where the default functionalities are implemented. It must use Bluetooth to find the peripherals, gather their measurements, and process those data, producing a report that shows the results and the raw data collected. If there is an agreement between the manufacturer and the customer, some data can be shared, writing them inside the volume where all the data from the *host* peripheral will be published. The remaining information is secured inside the container and cannot be seen by others. The second container, *Host*, represents the sandbox a generic customer can use to add functionalities to the system. In this case, it will only publish a report with some elaborations given by all the data from the host peripherals and some from the native ones. The *native* and the *host* reports are printed on a local server and continuously updated.

The shared volume, called Experiment, is accessed through both read and write permission from the *Native* container, while the Host has only read permissions. This prevents data corruption and ensures that only the *Native* container can manage that information.

Four instances of the Nicla Sense ME have been used to realize the peripheral sensors: three are considered native, while only one is viewed as a host. In this way, the Native container will consider only the devices associated with it for the report and the data elaborations. At the same time, the Host will only see what the Native wants to share, and the single device will be signed as the host. This way, the two reports will show what the two containers want to measure without sharing too much information and augmenting the system's security.

To illustrate the advantage of our proposed architecture, we have considered the differences between running the solution with the host Linux system and running it with the Docker solution using the library node-trace-events [21] to see the timing of the different actions performed by the system.

The images must be built to implement the solution on Docker, unlike the native system. However, these times depend on the internet connection available and so will not be considered. Once the containers start, the following actions are examined: time taken to find a peripheral, connect to it, and retrieve its characteristics and data. A table is made for all four peripherals, showing the difference between the experiments done and the two cases studied: the host system and Docker. The state diagram of the code is presented in Fig. 4. Even though four nodes have been used for the experiments, only one table is shown for simplicity and space since the results are very similar from one peripheral to the other.

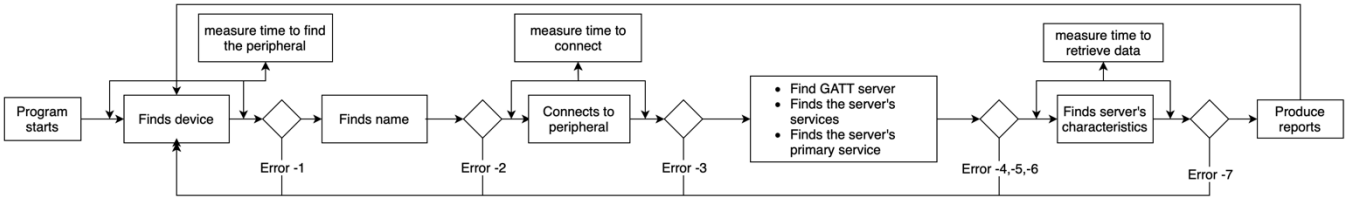

Fig. 4 - State diagram of the code

A. Time to find the device

The data are taken in the first step of the code, in which the code scans the different discoverable devices in the area until it finds the one specified in an array declared at the beginning. Once it has found the correct address, it tries to connect to it.

TABLE I. NICLA SENSE ME 09:E5:1E:9A:2A:1B

Experiment	Measurements		
	Date	*Average (s)*	*Variance (s^2)*
Docker	25-04-2023	1.009266	1.506005e-05
Docker	26-04-2023	1.009927	1.891779e-05
Docker	27-04-2023	1.009945	2.035555e-05
Linux system	21-04-2023	1.013111	3.258273e-05
Linux system	25-04-2023	1.019435	1.621683e-05
Linux system	26-04-2023	1.016153	2.751479e -05

It is possible to notice how the measurements are very similar for both the systems studied. No noticeable differences are present.

B. Time to connect to the device

Once the device is found, the code connects to it and proceeds once the connection is established. The second block represents this action in the state diagram.

TABLE II. NICLA SENSE ME 09:E5:1E:9A:2A:1B

Experiment	Measurements		
	Date	*Average (s)*	*Variance (s^2)*
Docker	25-04-2023	0.002580	2.177253e-06
Docker	26-04-2023	0.002701	7.662908e-07
Docker	27-04-2023	0.002428	6.531932e-07
Linux system	21-04-2023	0.003926	2.396214e-06
Linux system	25-04-2023	0.006169	1.196600e-06
Linux system	26-04-2023	0.004998	1.901245e-06

The connection values for some Nicla Sense ME are better for the Docker system than the host one. However, this can be explained by the fact that external noise can affect the time a sensor takes to connect to the central. Both systems use BlueZ and have been studied in the presence of numerous Bluetooth devices. The reason Docker is faster could be the driver activation, which is almost immediately after the creation of the container, instead of activating them manually in the Raspberry OS system.

C. Time to retrieve data

After the Concentrator is connected to the device, it founds the peripheral's GATT server, its services, and its characteristics. It connects once again and records the results given by the measurements done by the device. There is this measurement in the last block before the production of the reports in the state diagram.

TABLE III. NICLA SENSE ME 09:E5:1E:9A:2A:1B

Experiment	Measurements		
	Date	*Average (s)*	*Variance (s^2)*
Docker	25-04-2023	0.000449	2.455680e-08
Docker	26-04-2023	0.000362	5.440180e-08
Docker	27-04-2023	0.000479	1.155385e-07
Linux system	21-04-2023	0.000666	1.850928e-08
Linux system	25-04-2023	0.000621	9.439940e-09
Linux system	26-04-2023	0.000538	6.123639e-09

Regarding the characteristics retrieval, the results vary among the different Nicla. However, the time spent retrieving the data offered by the peripheral is acceptable since they have few differences on the scale of milliseconds. These results show better functioning in the Docker part, too, since even though the numbers are different, the timing is lower. This happens because the drivers for the communication are already immediately established due to the connection between the devices.

D. Resources

Regarding the resources used, the Docker system takes 102.6 Mb of space, while the native system is occupied by every package installation, with all their dependencies. The CPU usage, studied through the Raspberry CPU Usage Monitor, is the following:

TABLE IV. RESOURCES USED

Type of system	Measurements	
	Memory space	*CPU usage*
Application running on Raspberry OS host system	1.23Gb (16.12%) of 7.63Gb	20%
Application running on Docker – Native container	1.51Gb (19.79%) of 7.63Gb	30%

It is possible to notice that CPU usage increases when the main application (implemented in the *Native* container) is executed. Since it must only produce the report, the Host container is a little computationally heavy. The *Native,* instead, must use Bluetooth to retrieve the data and so takes more CPU to execute the commands. The difference between the host system and the Docker solution is that in the second case, the programs are executed simultaneously, with the presence of Docker itself creating a safe and secure environment for the containers. The memory space instead has been seen through the command "htop" on the terminal. It is possible to notice how the Docker solution occupies more memory than the Debian basic system. This is due to the following three main factors: the usage of containers, the reinstallation of dependencies and valuable programs, and the overhead introduced by Docker itself due to the copies of packages present in different images. We cannot determine how much memory space the Linux basic system takes since

the installed packages and all their dependencies are very sparse. To do so, a mint Linux system should be taken and studied.

E. Security

Security is another main topic of the comparison between the two systems. It is essential to ensure that the files produced and the data will not be accessed by those who should not read their content and that no corruption is done. Docker allows the creation of multiple environments that share the same operating system without interfering with each other. The containers limit access to the different files, avoiding unwanted actions (like deleting or modifying) and permitting control of how the data are shared and seen by the system's actors. They can access only the information inside the volume created. Depending on the selected permissions given in the configuration, the files produced can only be read or written by the chosen container.
On the other hand, the basic Linux system (Debian, in this case) can be faulty due to the basic privileges that the system itself provides. Files can be modified without administrator privileges, causing a program not to work or to have data modified without the user's consent. Also, the data between the two containers should not be shared because the producer may not want the host to see its report or vice versa, so the system must control them. Docker allows the information to be private and not accessed by everyone, augmenting the purpose of secrecy.

To summarize the behavior of the Docker system concerning the Raspberry OS, the experiments proved the creation of an instance for the division between a container using the default implementation (the *Native* one) and one that can be changed depending on the customer's needs (the *Host* one). In our case, data are principally centralized from the different sensors, cataloged, and shared depending on the needs of the *Host* container. In future work, it would be interesting to understand how to implement an actual real case of elaboration to see how much data it can handle and how the Docker system is affected depending on the resources used.

V. Conclusions

With the advancement of Industry 4.0, it has become essential to how data are treated and how to collect them. This paper presents different methods: cloud, edge, and fog computing. These permit reduced costs and time, improving efficiency in real-time measurements and storing helpful information for a generic industry. However, a device on the shelf usually cannot be modified, risking introducing in its default functions bug or unwanted behaviors in the software itself, so it is not a customizable asset. The new device presented called Concentrator solves this problem, letting a generic customer introduce third-party programs without interfering with its primary performance and adding peripherals or processing elements beneficial for its system. The solution can be created by using a Linux system. However, the experiments conducted showed not only that this architecture can be done in an embedded solution (like a Raspberry Pi) but also that it is possible to use a program, like Docker, to create an environment where everything is safe and secure since the containers permit the isolation of all the information, augmenting the privacy.

VI. Acknowledgments

The authors want to thank AROL Closure System, which shared its knowledge and assets, without which the project could not have been realized. Thanks also to our colleagues who helped us during the development of this paper.

VII. Bibliography

[1] W. W. a. H. J. Kagermann H., «Securing the Future of German Manufacturing Industry: Recommandations for Implementing the Strategic Initiative Industrie 4.0. Final Report of the Industrie 4.0 Working Group,» Acatech, 2013.

[2] R. R. a. T. S. Madakam S., «Internet of Things (IoT): A Literature Review,» *Journal of Computer and Communications,* vol. 3, pp. 164-173, 2015.

[3] E. a. S. A. a. H. S. a. J. U. a. G. M. Sisinni, «Industrial Internet of Things: Challenges, Opportunities,» *IEEE Transactions on Industrial Informatics,* vol. 14, n. 11, pp. 4724-4734, 2018.

[4] R. a. B. J. a. G. A. M. Buyya, Cloud computing: Principles and paradigms, John Wiley & Sons, 2010.

[5] P. M. a. T. Grance, «The NIST Definition of Cloud Computing,» 2011.

[6] W. a. C. J. a. Z. Q. a. L. Y. a. X. L. Shi, «Edge Computing: Vision and Challenges,» *IEEE Internet of Things Journal,* vol. 3, n. 5, pp. 637-646, 2016.

[7] F. a. M. R. a. Z. J. a. A. S. Bonomi, «Fog Computing and Its Role in the Internet,» in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012.

[8] C. &. L. Q. Yi Shanhe & Li, «A Survey of Fog Computing: Concepts, Applications, and Issues,» 2015.

[9] R. a. R. A. R. a. K. D. Dua, «Virtualization vs Containerization to Support PaaS,» in *IEEE International Conference on Cloud Engineering*, 2014.

[10] D. a. Z. L. Liu, «The research and implementation of cloud computing,» in *11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP)*, 2014.

[11] C. P. a. B. Lee, «Containers and Clusters for Edge Cloud Architectures -- A Technology Review,» in *3rd International Conference on Future Internet of Things and Cloud, 2015, pp. 379-386, doi: 10.1109/FiCloud.2015.35*, 2015.

[12] F. -G. O. a. T. C. L. Letondeur, «A demo of application lifecycle management for IoT collaborative neighborhood in the Fog: Practical experiments and lessons learned around docker,» in *2017 IEEE Fog World Congress (FWC)*, Santa Clara, CA, USA, 2017.

[13] H. S. a. A. Mourad, «Towards Dynamic On-Demand Fog Computing Formation Based On Containerization Technology,» in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2018.

[14] C. E. a. R. D. M. Samaniego, «Smart Virtualization for IoT,» in *IEEE International Conference on Smart Cloud (SmartCloud), pp. 125-128, doi: 10.1109/SmartCloud.2018.00028*, 2018.

[15] I. F. A. I. a. T. T. R. Morabito, «Evaluating performance of containerized IoT services for clustered devices at the network edge,» *IEEE Internet of Things Journal,* vol. 4, n. 4, p. 1019–1030, 2017.

[16] A. a. R. E. a. K. A. Sergeev, «Docker Container Performance Comparison on Windows and Linux Operating Systems,» in *2022 International Conference on Communications, Information, Electronic and Energy Systems (CIEES)*, 2022.

[17] C. a. T. D. R. a. D. M. a. R. B. Martín, «FogPi: A Portable Fog Infrastructure through Raspberry Pis,» in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 2020.

[18] R. a. T. A. N. Mahmud, «Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework,» *IEEE Internet of Things Journal,* vol. 9, n. 6, pp. 4125-4138, 2022.

[19] M. V. G. M. Pietro d'Agostino, «A user-extensible solution for deploying fog computing in industrial applications,» in *International Symposium on Industrial Electronics*, Helsinki, 2023.

[20] Arduino, «Arduino Nicla Sense ME,» Arduino, [Online]. Available: https://docs.arduino.cc/hardware/nicla-sense-me.

[21] B. B. Trent Mick, «node-trace-event,» [Online]. Available: https://github.com/TritonDataCenter/node-trace-event.