- ORIGINAL ARTICLE -

Analysis, Deployment and Integration of Platforms for Fog Computing

Análisis, Despliegue e Integración de Plataformas para Fog Computing

Joaquín de Antueno, Santiago Medina[1], Laura De Giusti[1 2] and Armando De Giusti[1 3]

[1] *Instituto de Investigación en Informática III LIDI, Facultad de Informática, Universidad Nacional de La Plata - Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, La Plata, Argentina*
[2] *CICPBA - Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, La Plata, Argentina*
[3] *Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), CABA, Argentina*
{jdeantueno, smedina, ldegiusti, degiusti}@lidi.info.unlp.edu.ar

Abstract

In IoT applications, data capture in a sensor network can generate a large flow of information between the nodes and the cloud, affecting response times and device complexity but, above all, increasing costs. Fog computing refers to the use of pre-processing tools to improve local data management and communication with the cloud. This work presents an analysis of the features that platforms implementing fog computing solutions should have. Additionally, an experimental work integrating two specific platforms used for controlling devices in a sensor network, processing the generated data, and communicating with the cloud is presented.

Keywords: Cloud Computing, Fog Computing, Internet of Things, IoT Platforms.

Resumen

En las aplicaciones de IoT la captación de datos en una red de sensores puede generar un gran flujo de información entre los nodos y el cloud, condicionando los tiempos de respuesta, la complejidad en los dispositivos y por sobre todo incrementando los costos. Fog Computing se refiere a la utilización de herramientas de pre-procesamiento para mejorar el manejo de datos locales y la comunicación con la nube. En este trabajo se presenta el análisis de las características que deben tener las plataformas que implementan soluciones de Fog Computing. Adicionalmente se presenta un trabajo experimental que integra dos plataformas específicas, aplicadas al control de dispositivos en una red de sensores, el procesamiento de los datos generados y la comunicación con la nube.

Palabras claves: Cloud Computing, Fog Computing, Internet de las Cosas, Plataformas de IoT.

1. Introduction

The cloud computing model is designed to provide remote, on-demand access to a shared set of resources, such as servers, networks, storage, or other services. These resources can be acquired and released with minimal effort on the part of the user, regardless of the infrastructure required to use them, and with reduced interaction with the service provider [1].

As a result of the proliferation of Internet-of-Things (IoT) applications in recent years, cloud service providers are turning more and more attention to developing platforms for the deployment of such applications. The challenge lies in the new needs that some IoT systems pose, which the cloud has difficulties covering, including communication with a large number of nodes maintaining a low response time and receiving large volumes of information recorded by sensors to be analyzed, stored and ultimately displayed to the user. This last aspect is of particular interest, given that the traffic that goes to the cloud in this type of applications is one of the main costs generated.
In response to this problem, a new model, called fog computing, was proposed, which consists of an intermediate processing, storage and connectivity layer for IoT nodes, located close to them, on the edge of the network [2]. In this model, communication load with all nodes is distributed among several geographically distributed servers, with the goal of providing shorter response times

and managing the volume of data generated locally. This intermediate layer, in turn, interacts with the cloud, and delegates to it only the tasks for which a greater computational capacity, storage or a global vision of the system is required. This model also reduces the flow of data to the cloud, acting as a filter or by preprocessing the information, thus reducing communication costs.

The main goals are:

- Integration of a platform (Node Red) that is a flow based programming tool designed to manage and handle MQTT topics which allows communication with cloud platforms and an IoT platform (Thinger.io) that provides communication with system nodes, storage, data visualization and other features as efficient data transmission for real time communication.
- Experimentation with one possible implementation of the integration, in the context of a real life Fog Computing system.

This work is formed by 5 sections, the first of which is this introduction. Section 2 provides a definition for the Fog Computing model as well as for the main attributes of applications that follow the model. In section 3 the different functionalities to be provided by an IoT platform to be used in Fog Computing are described, and it is analyzed which of them and how they are implemented by Thinger.io and Node-RED. In section 4 an integration of these two platforms is proposed and analyzed, and an example implementation of the integration in the context of a real Fog Computing application is described. Finally, section 5 examines how the necessary functionalities for the development of Fog Computing systems are provided by the proposed integration and describes related future work.

2. Fog Computing

The fog computing model comes as a response to an emerging wave of developments related to the Internet of Things, originally conceived by Bonomi [2]. These developments have characteristics that make their development more complex using cloud computing technologies exclusively, but still benefit from their use. Fog computing then consists of an intermediate platform that provides processing, storage and network communication services between edge devices that collect information and data centers in the cloud. The defining characteristics of this model are:

- Real-time applications.
- Wide geographic distribution.
- Large number of nodes.
- Node heterogeneity.
- Mobility.
- Predominantly wireless communication network.
- Low latency and location awareness.

At the same time, it should be noted that this architecture does not work as a replacement for the cloud, but interaction with it is crucial, with limited resources available at the edge of the network.

3. IoT Platforms Applied to Fog Computing

In [3], Assemani et al. identify the main functionality that should be offered by a fog computing-oriented IoT platform that is implemented as the middle layer of the model. Their goal was to define a model architecture for these platforms and have an evaluation framework for the different existing platforms.

In particular, the following components or functionality to be offered by the fog layer in the platform are specified:

- Device management: It provides real-time communication, identification and response to devices at the edge of the network.
- Data processing: It is responsible for sending data to the cloud, as well as local processing, which includes storage, filtering and analysis. It can also respond to events generated by devices in real time.
- Security: This includes both communications security and edge devices authentication, as well as controlling access to these devices and their data and establishing secure communication with the cloud.
- Gateway features: Protocol translation and managing devices with various characteristics.
- Application life cycle management: Application development, distribution, deployment and updates.

This set of components, derived from 10 platforms in use in the market, allows defining the needs that a fog computing-oriented system should cover, as well as having a clear criterion to assess results.

Additionally, the implementation priority for each of the components is defined, based on the order in which they were incorporated into the main market platforms. Device management, data processing, and gateway features are the highest priorities for these platforms, followed by security measures and, finally, application life cycle management, which is offered only by a handful of the platforms in the market.

Related works evaluating and comparing different IoT platform's features exist, both in the context of Fog Computing and outside of it [3, 4]. To the author's knowledge no previous works exist regarding integrations between IoT platforms similar to the one described in this paper.

Our work began in 2018 using Node-RED extended functionality to communicate with cloud platforms. In this paper we present the required integration with a Thinger.io to provide efficient communication with system nodes. The result is a Fog Computing platform that allows new cloud services for IoT applications.

Different IoT platforms were considered, including Thinger.io, Thingsboard [5, 6], Mainflux [7] and SiteWhere [8]. Our choice of integrating Node-RED with Thinger.io was based on the features this platform provides for communication with external services, and the third party Node-RED libraries specifically designed to work with Thinger.io.

The features of Thinger.io and Node-RED, the platforms to be used for this integration, are described below. Also, some missing features that are required to adapt to the model described are pointed out.

3.1. Thinger.io

Thinger.io [9] is an open source IoT platform that follows the cloud computing model, providing communication with system nodes, storage and data visualization, among other features. It places special emphasis on efficient data transmission, real-time two-way communication, interoperability, and deployment simplicity.

Specifically, Thinger.io provides the following features:

- Device models: Each node is modeled by a device on Thinger.io, and is identified by a unique credential that allows connecting to the platform securely.
- Resources: These are a complement to device model, representing the information that flows from or to the nodes. They are defined in a simple way in the node code, and are encoded using Protoson, a proprietary library, to reduce packet size.
- Data Buckets: These are a chronologically ordered form of storage in which devices can publish information, to which a timestamp is added when it gets to the platform.
- Endpoints: These allow devices to communicate with services external to Thinger, either through HTTP requests or by sending e-mails. These communications may include resources defined by the devices in JSON format, or they may use a different format that incorporates this information as necessary.
- Dashboards: These consist of a configurable user interface that allows graphically representing data in real time from the devices or extracting historical information from the buckets.
- REST API: It allows interaction with Thinger from external services, either to manage buckets, endpoints, devices or dashboards, or to communicate directly with the nodes.

This platform is used in various IoT applications; [10, 11, 12] are some examples of systems implemented using Thinger.io.

3.2. Node-RED

Node-Red is a flow-based open source programming tool, originally designed to manage and handle MQTT topics. It is ultimately a general purpose tool, although nowadays it is strongly oriented towards the development of IoT applications [13, 14]. The different features of the platform are modularized into "nodes" that interconnect to create the application, communicating with each other through JSON packages. The main nodes used in this work are:

- HTTP In: It allows creating an HTTP access point through which other platforms can communicate. As output, it produces a packet with the body of the request and other data used to send a response.
- HTTP Request: It allows sending an HTTP request to an external service, producing as output a raw buffer or a JSON object.
- Function Nodes: These are general-purpose nodes that run JavaScript code every time they receive a package from another node, generating a JSON object as output or stopping the flow if necessary.
- Bucket Write: It allows writing the contents of the object received as input to a Thinger.io data bucket.
- TCP In: It allows establishing a TCP connection to a remote server, or accepting incoming connections, returning as output the packets received together with connection characteristics.

- TCP Out: It allows replying to messages received through the TCP In node.

Node-RED functionality can also be extended through numerous libraries. For the purposes of our work, the most relevant ones are those that allow communication with other cloud platforms, including Thinger.io, AWS IoT and IBM Watson, among others.

3.3. Feature Analysis

After considering the use of Thinger.io as a fog computing platform, based on the criteria mentioned above, it is found that it only partially fulfills the required functionality:

- When it comes to device management, Thinger.io provides efficient, real-time communication through its device models and resources. We noted the need for automatic device registration, crucial for scalability and ease of deployment on systems with a large number of nodes.

- Data processing is the biggest flaw we found on the platform, since it only allows storage through data buckets, which can be displayed on dashboards. That is, data cannot be locally analyzed to obtain the information that is more useful for the user, nor to respond to node events.

- Additionally, the only mechanism to connect with external services, the endpoint, only facilitates the direct transfer from a device to a service via HTTP, which excludes the application of any type of platform filters or pre-processing if the data are sent to the cloud.

- Regarding security, Thinger.io provides encrypted communications using TLS for connections to nodes. As already mentioned, communication with the cloud would occur through endpoints, which do not provide any type of security feature.

- As regards gateway features, protocol compatibility is solved by using Thinger.io's proprietary one, built directly on TCP. In turn, libraries are provided that allow connecting to a variety of hardware devices, regardless of their characteristics, through the defined resources.

- The platform does not currently provide the necessary functionality for application life cycle management.

Node-RED complements the features offered by Thinger.io, with the possibility of assembling processing flows that work with the data buckets that store the data. In addition, it allows secure interaction with cloud services through dedicated nodes.

The complete analysis of Node-RED features is beyond the scope of this work, since it is a general-purpose tool. For this reason, we will only discuss the features that are useful for our development.

4. Model Architecture and Integration

Having described the features of each platform and their deficiencies in the context of a Fog Computing application, we proceed to describe the integration between them, conceived with the objective of solving these deficiencies.

The model architecture would then include all the edge devices, each equipped with their respective sensors and actuators. These devices are distributed over a wide geographic area and they may be provided with mobility. The intermediary servers are deployed near to them in order to fulfill the low latency requirements and analyze the data the edge devices generate. Each of the servers runs an instance of both Node-RED and Thinger.io, which complement their capabilities to accomplish all required tasks. These servers are only responsible for tasks with limited resource requirements. The rest of the tasks, which demand more processing or storage capacity than the intermediary servers can offer and are not limited by the same time constraints, are performed by the cloud. This reduces its usage and the volume of data it receives to the minimum levels. Figure 1 demonstrates the interaction between the three layers of the model. It is important to notice how each intermediary server only manages the (relatively reduced) group of edge devices deployed nearest to it.

4.1. Integration

In this section we will describe how the integration between Thinger.io and Node-RED works, in a way that allows them to complement their functionalities and provides us with a platform for developing Fog Computing applications.

The first point to solve is edge device registration on Thinger.io, which is required to communicate with the platform. By default, this must be done manually, which is very limiting in terms of deployment and scalability in a system with many

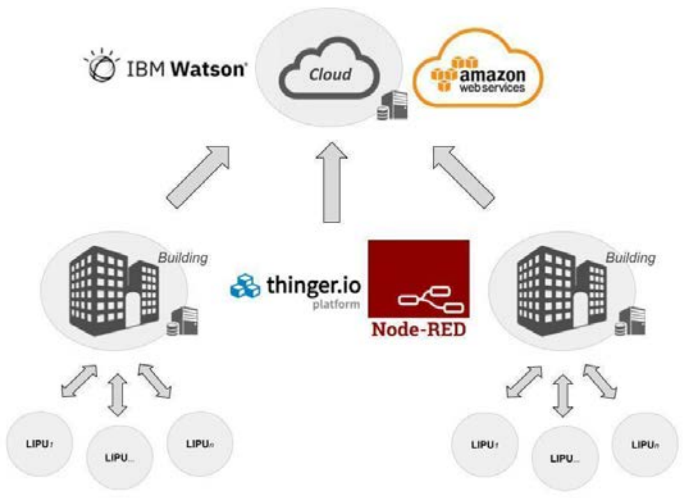

Fig. 1 - Architecture with platform integration.

devices. The proposed solution consists of using the Thinger.io API in Node-RED to create both the device model and the data buckets that are used to store data, dashboards to display data or tokens to access the data remotely, depending on the requirements of the specific system. The devices are then responsible for connecting with Node-RED through TCP only once, receiving in response the credentials they will have to use for all communications with Thinger.io from then onwards.

After this first connection, all communications with the devices will be established through Thinger.io, taking advantage of its transport efficiency and data encoding features.

Thus, each device will define its interface using Thinger.io resources, periodically transmitting information that can be stored in data buckets.

The second aspect to be considered is the analysis of these data by the intermediary servers. To communicate with Node-RED, Thinger.io HTTP endpoints are used, creating one for each type of package that needs to be sent, depending on application requirements. An API is built in Node-RED to receive the packets using HTTP-In nodes. This provides a communication mechanism between the devices and Node-RED. The latter is in charge of processing all information, storing it temporarily, grouping it and filtering it before sending it to the cloud, responding to events with a reduced response time compared to that of the cloud and generating statistics and useful information from the system to local level, which will be sent to Thinger.io through the dedicated nodes to be written in data buckets and shown to the user in dashboards.

At this point, the flexibility of Node-RED when acting on this information should be highlighted. The most basic processing, and the one implemented as part of this work, is carried out by function nodes, which run JavaScript.

In turn, the possibility of using the Thinger.io API from Node-RED in a dynamic way should be considered, allowing the creation and modification of data buckets and dashboards to improve how information is presented to the end user automatically from the information collected.

Beyond local collaboration with Thinger.io, Node-RED meets the other important requirement for this type of systems – the interaction with the cloud. This can be carried out using several dedicated node packages, which include using services from Amazon Web Services, Google Cloud, IBM Watson and Microsoft Azure, among others. Using a Thinger.io cloud instance integrated with Node-RED is also a possibility, although this implementation will differ from that used on intermediary servers.

Finally, cloud services are responsible for tasks that do not have low latency requirements; for example, analyzing information, generating statistics at a global level, and allowing the implementation of system-wide operational policies.

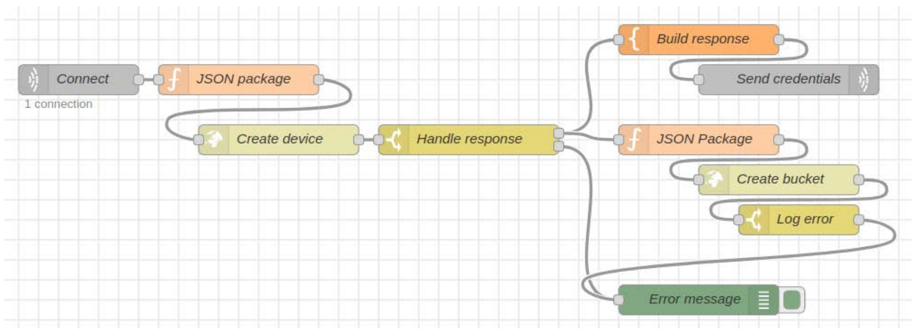

Fig. 2 - Node registration implementation in Node-RED.

4.2. Experimentation

In this section, a possible implementation of the integration suggested above is described, aimed at showing its application in a real system.
An energy control system with the typical requirements of the fog computing model is used [15, 16].
To integrate both platforms, a Raspberry PI 3 B+ was used as a server, with the Ubuntu operating system, on which the Thinger.io and Node-RED instances were deployed. A server developed as part of the "Intelligent Distributed System for Energy Control" project was also installed.
It should be noted that, despite the fact that there is currently a plug-in that allows running Node-RED within the Thinger.io platform, it is not used in this development (and this does not change the nature of the integration).
The nodes used are Espressif ESP8266 micro-controllers, each connected to a DHT22 temperature and humidity sensor, an HC-SR501 motion sensor, a real-time clock (RTC), two relays, and an infrared emitting LED to simulate shutdown by a remote control. The Thinger.io library is used for communication with the platform.
The system consists of a large number of nodes, distributed throughout the various environments of the buildings to be controlled. These nodes generate regular readings on the use of electronic devices, temperature, humidity and presence of people in each of the environments. In addition to this, they can be used to control these devices, either with programmed responses based on the readings mentioned above, or in real time, by local or remote system users. To facilitate their control, devices are organized into configurable groups (CG) in each environment, the most common being lights and ACUs.
Specifically, with the use of the integration proposed here, we hope to obtain real-time information such as number of groups in use by type of device, number of individual devices in use by how they are controlled, number of CGs turned on despite no people being detected in the room, total operation hours per CG for all environments, average temperature and humidity in the building, and operation hours per environment.
The evolution of this information throughout the day is also useful to optimize environment use.
The node registration mechanism described above was implemented, which is responsible for creating the device model and data buckets to store information. For this purpose, a Node-RED flow is built, which starts by receiving a TCP message from the new node to be registered. After this, a random identifier for the node and a credential are generated, which are used to build a JSON package and send the request to the Thinger.io API to register the device, using the HTTP Request node. If this request is successful, a second call is made, which creates a data bucket corresponding to the node. In parallel, the new credentials are sent to the node. Figure 2 shows the implementation in Node-RED.

Once the node knows its credentials, it connects to Thinger.io and defines its interface using the following resources:

- group_status: This contains information about the status of each group of devices in the room, which can be on, off, or in standby (i.e., temporarily turned off until someone is detected in the room). In addition, the package contains information about the number of devices turned on, grouped by those controlled by relays and those controlled by infrared emitters.
- ambient_status: This contains room temperature and humidity readings, or a notification in case of sensor failure.
- motion_report: It indicates if movement was detected in the room since the last report sent; it also includes the configured time range for the reports.

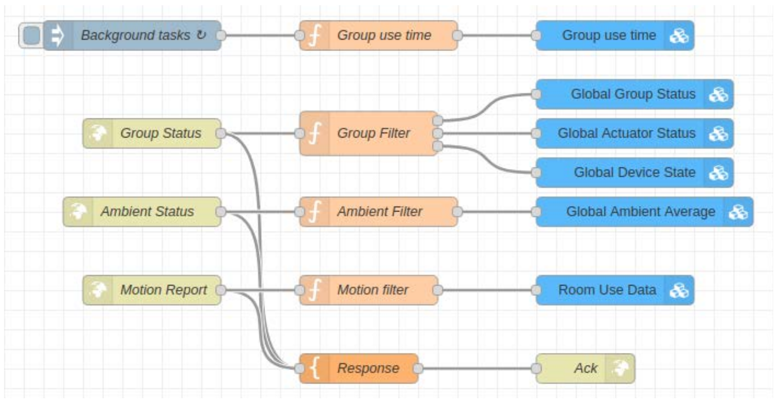

Fig. 3 - Data processing implementation in Node-RED.

All these resources also attach the corresponding device IDs, which is useful for processing in Node-RED. Each of these will be periodically transmitted, stored in the data bucket corresponding to the node, and sent to Node-RED for processing. Specifically, the information corresponding to groups and devices is temporarily stored in order to group all the rooms and calculate the number of groups in use of each type, their usage time, the groups that are turned on in empty rooms, and the number of devices currently in use.
Temperature and humidity readings receive a similar treatment, calculating an average or generating a notification if any undesirable value is reported, for example, to optimize the use of ACUs.
The information generated by the motion detector is used in combination with the groups to calculate device usage time in empty rooms.
At the same time, Node-RED allows scheduling tasks to be carried out periodically for example, to continually update group usage hours.

Figure 3 demonstrates the implementation for data reception, processing and storage in Node-RED.

After the information is processed, it is sent to Thinger.io for storage and display. Write nodes are used in data buckets to keep a historical record of all data, and there is some flexibility for displaying it.
Finally, a dashboard was built on Thinger.io for monitoring the entire building. This dashboard uses the information stored in the data buckets and displays it in real time and shows its evolution throughout the day. Device use, group use and room temperature were displayed as daily evolution, while the rest of the information was displayed in real time. Figure 4 shows the finished dashboard.

5. Conclusions and Future Work

The features to be considered for fog computing platforms were presented and the integration of two separate platforms was carried out aimed at providing a basis for future developments related to

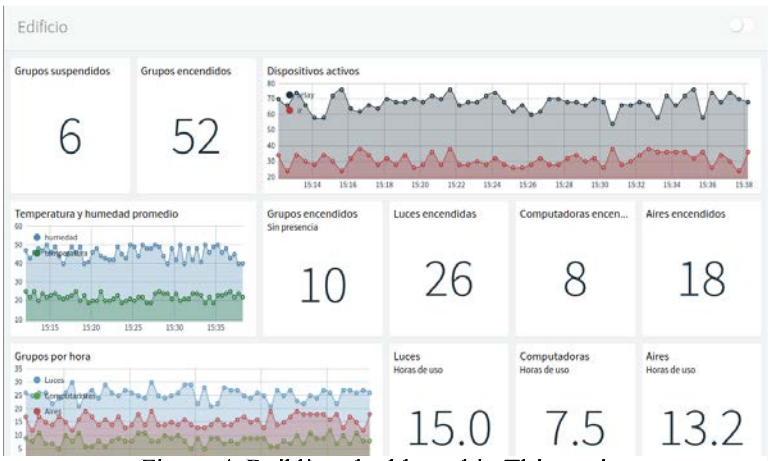

Figure 4. Building dashboard in Thinger.io.

fog computing. This integration is used for meeting the requirements of this type of applications:

- Device management: Thinger.io administration is complemented with an automatic registration mechanism in Node-RED, crucial for the deployment of systems with a large number of nodes.
- Data processing: Data processing is implemented using JavaScript, generating information and response to events in real time, which allows monitoring the system on different levels.
- Security: The secure communication features provided by Thinger.io are used to communicate with devices, processing is done locally, and dedicated Node-RED nodes are used for communication with the cloud, protecting connection security.
- Gateway: Thinger.io's proprietary communication protocol is used, as well as its packet encryption, which provides efficiency and flexibility. A variety of devices can be connected, including Espressif ESP8266, ESP32, Arduino and Raspberry Pi, among others.

In the future, the deployment of several integration instances communicated using some specific cloud service will be studied, and tests with higher volumes of data and architecture stress tests will also be performed. On the other hand, an increasingly common feature in this type of systems, and offered by most IoT platforms in the cloud, is using machine learning predictions on the data generated by the nodes. Node-RED provides this option through different dedicated node libraries, capable of making predictions based on a model trained in the cloud, taking advantage of its greater processing capacity. This option is one of the features planned for addition to this integration in the future.

Competing interests

The authors have declared that no competing interests exist.

Authors' contribution

JA and SM conceived and developed the integration. All authors analyzed the results, wrote, revised and approved the manuscript.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing". Special Publication 800-145, National Institute of Standards and Technology, U.S. Department of Commerce, 2010.

[2] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", in *MCC '12: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. Association for Computing Machinery, New York, NY, United States*, 2012.

[3] M. Asemani, F. Jabbari, F. Abdollahei and P. Bellavista, "A Comprehensive Fog-enabled Architecture for IoT Platforms", *High-Performance Computing and Big Data Analysis. TopHPC 2019. Communications in Computer and Information Science, vol 891. Springer, Cham,* 2019.

[4] M. Cruz, J. Rodriguez, A. K. Sangaiah, J. Al-Muhtadi and V. Korotaev, "Performance evaluation of IoT middleware", *Journal of Network and Computer Applications, Volume 109,* 2018.

[5] "Thingboard Open Source IoT Platform". Available at: https://thingsboard.io/. Accessed on 2020-10-10.

[6] T. L. Scott and A. Eleyan, "CoAP based IoT data transfer from a Raspberry Pi to Cloud". *International Symposium on Networks, Computers and Communication (ISNCC2019), Istanbul, Turkey, pp. 1-6,* 2019.

[7] "Mainflux Open Source IoT Platform" . Available at: https://www.mainflux.com/. Last accessed: 2020-10-10.

[8] "SiteWhere Open Source Internet of Things Platform". Available at: https://sitewhere.io/. Last accessed: 2020-10-10.

[9] A. L. Bustamante, M. A. Patricio and J. M. Molina, "Thinger.io: An Open Source Platform for Deploying Data Fusion Applications in IoT Environments", *Sensors 2019,19 (5), 1044, MDPI,* 2019.

[10] A. A. Rodriguez Aya, J. A. Figueredo Luna and J. A. Chica García, "Sistema de control y telemetría de datos mediante una aplicación móvil en Android basado en IoT para el monitoreo de datos". *Espacios 39(22):30, Espacios Inc.,* 2018.

[11] L. Aghenta and T. Iqbal, "Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing", *Electronics 2019, 8(8), 822, MDPI*, 2019.

[12] W. S. Aung and S. Aung Nyein Oo, "Monitoring and Controlling Device for Smart Greenhouse by using Thinger.io IoT Server". *International Journal of Trend in Scientific Research and Development*, 2019.

[13] Dr. S. K. Selvaperumal, W. Al-Gumaei, R. Abdulla and V. Thiruchelvam, "Integrated Wireless Monitoring System Using LoRa and Node-Red for University Building". *Journal of Computational and Theoretical Nanoscience, Volume 16, Number 8, American Scientific Publishers*, 2019.

[14] S. Sicari, A. Rizzardi and A. Coen-Porisini, "Smart Transport and Logistics: a Node-RED implementation", *Internet Technology Letters, Volume 2, Issue 2, John Wiley & Sons*, 2019.

[15] Proyecto "Unidad Inteligente para Control de Consumo Eléctrico (UICCE)", *Convocatoria "Vinculación Tecnológica. Agregando Valor 2017", Secretaría de Políticas Universitarias (SPU). Approved and financed by SPU and UNLP,* Supervision: Dra. Laura De Giusti, 2017.

[16] M. Pi Puig, J. M. Paniego, S. Medina, S. Rodriguez Eguren, L. Libutti, J. Lanciotti, J. de Antueno, C. Estrebou, F. Chichizola and L. De Giusti, "Intelligent Distributed System for Energy Efficient Control", *Cloud Computing and Big Data. JCC&BD 2019. Communications in Computer and Information Science, vol 1050. Springer, Cham*, 2019.

Citation: J. de Antueno, S. Medina, L. De Giusti and A. De Giusti. *Analysis, Deployment and Integration of Platforms for Fog Computing*. Journal of Computer Science & Technology, vol. 20, no. 2, pp. 108-116, 2020.
DOI: 10.24215/16666038.20.e12.
Received: June 1, 2020 **Accepted:** October 10, 2020.