applied sciences

Article

A Smart Parking Solution Architecture Based on LoRaWAN and Kubernetes

Jhonattan J. Barriga [1,2], Juan Sulca [1,2], José León [1,2], Alejandro Ulloa [1,2], Diego Portero [1,2], José García [1,2] and Sang Guun Yoo [1,2,*]

1 Facultad de Ingeniería de Sistemas, Escuela Politécnica Nacional, Quito 170525, Ecuador; jhonattan.barriga@epn.edu.ec (J.J.B.); juan.sulca@epn.edu.ec (J.S.); jose.leon01@epn.edu.ec (J.L.); victor.ulloa02@epn.edu.ec (A.U.); diego.portero@epn.edu.ec (D.P.); jose.garcia01@epn.edu.ec (J.G.)
2 Smart Lab, Escuela Politécnica Nacional, Quito 170525, Ecuador
* Correspondence: sang.yoo@epn.edu.ec

Received: 13 April 2020; Accepted: 19 June 2020; Published: 7 July 2020

Abstract: Finding a parking space in a city is one of the most common activities of a driver. This becomes more difficult when the city is unknown or has huge vehicular congestion. A solution to address this issue is called smart parking. Smart parking solutions rely on Internet of Things (IoT) and several technologies to achieve their purpose. This paper proposes an architecture for deploying a smart parking solution based on Long-Range Wide Area Network (LoRaWAN) sensors, LoRaWAN and a cluster of Kubernetes. This approach provides an open architecture able to share information with other parties through a REST API interface. Likewise, it contains a mobile and a web application for user interaction. This solution provides an administration interface for managing parking lots. The user interface lets a user to find, view information, display available spaces and rate a parking lot in real time. This solution could be used as an application as service parking system. The proposed architecture is fully portable and scalable due to the use of Kubernetes.

Keywords: IoT; smart parking; architecture; LoRaWAN; Kubernetes; MQTT; MongoDB

1. Introduction

One of the most common activities of a driver in a city, is to find a parking space. Finding a parking space turns difficult when a driver is not familiar with the city or when the city has vehicular congestion. The activity of finding space is manual and demands time. An average of 30% of the vehicles on the street are looking for a parking spot and the time it takes is around 8 min in average [1]. In addition, a vehicle spends 10% of its time moving whilst the rest of the time it is stopped as mentioned in [2]. Smart parking solutions are useful because they help drivers to reduce fuel consumption, save time and money, reduce stress and increase safety as do not get distracted while finding a parking place [3].

There are several novel smart parking solutions with different architectures and ways of implementation as reviewed in [1]. However, these solutions have several limitations from the technological perspective that do not allow to have a comprehensive smart parking system. First, in terms of sensors, they are not specialized ones with specific-purpose (i.e., general sensors for Arduino, Raspberry or Espressif (ESP) boards). Additionally, communication protocols to collect data from sensors are not appropriate for handling small payloads within long distances. Some existing solutions do not emphasize on power-consumption which is required for most of Internet of Things (IoT) scenarios. Furthermore, previous solutions' backend infrastructures are based on a virtual machine approaches or traditional dedicated server deployments. In addition, these solutions do not consider mechanisms for holding data for extended periods of time in case of failure. Moreover, even though high-availability is crucial in smart parking since it provide information in real-time, most of systems do not consider this requirement. Finally, integrating previous solutions to other systems is complex and demands more effort as they do not have standardized interfaces such as APIs.

One of the solutions that helps to solve the aforementioned problem is smart parking. Smart parking is one of the most remarkable use cases within smart cities context [4]. Smart parking solutions can reduce congestion, optimize parking time and reinforce traffic laws. Smart parking solutions leverage on the use of Internet of Things (IoT) technologies to expand and fulfill their purpose. IoT relies on technologies that support low energy consumption, profitability, low data rate and long-reach among others to deliver solutions that benefit humankind [5]. In this scenario, Low-Power Wide-Area Network (LPWAN) transmission technologies provide many benefits for smart parking solutions, such as wide coverage in rural and urban areas at low energy consumption. Among the LPWAN protocols, the most popular are: Narrowband IoT (NB-IoT), SigFox and LoRaWAN as stated in [5]. These protocols have similarities in terms of architecture but differ in other parameters such as frequency of operation, security and connection fees. Within LPWAN protocols, the most flexible and used protocol is LoRaWAN [6]. LoRaWAN is an open standard protocol which is the most attractive reason for being widely used. It is based on LoRa which works over Industrial Scientific and Medical (ISM) bands and does not require a connection fee like NB-IoT. Likewise, it does not require a third-party infrastructure (backend servers) for its deployment as occurs in SigFox [7].

In terms of architecture, there are several ways to deploy a smart parking solution. For instance, authors in [8] used Arduino boards as sensing infrastructure components. There are other solutions as [9] that focus on building their own sensors rather than deploying a specific purpose sensor as in [10]. This proposed solution suggests the use of specific type sensors to collect information at the sensing infrastructure layer. In terms of communications several works uses Wi-Fi for connecting sensors with backend infrastructure [11]. However, Wi-Fi connectivity consumes more power and does not cover long-range areas. Our solution uses LoRaWAN which is a low power consumption and long-range protocol. In terms of backend infrastructure, some smart parking solutions use open-source software elements such as Message Queuing Telemetry Transport (MQTT) for exchanging information [12] between sensors and the backend. Others like [13] uses a whole backend platform and only take care about sensors and communication protocols. In the proposed scenario we promote the use of open-source software by designing a cluster architecture to increase availability and scalability.

This work reviewed several deployments, proposed and built an architecture based on Libelium smart parking sensors, LoRaWAN and open-source technologies for the backend like a cluster of Kubernetes with MQTT and MongoDB. The proposed architecture has aimed to combine several technologies to develop a robust smart parking system from the perspectives of stability and availability of the service. This combination intends to provide high availability, scalability and portability to a smart parking system. Likewise, this architecture aims to provide a base guide for building this type of solutions by considering the use of specific purpose sensors, adequate low-power and long-range communication protocols and a clustered backend infrastructure. From the availability and stability perspective, other solutions do not provide redundancy over a clustered-environment. They do not use a messaging server (i.e., Rabbit) which can be a bottleneck. In addition, if the main server handling and processing the information fails, data from sensors would be lost. In comparison to other works, in the proposed architecture if the main server in charge of handling the information collected fails, all messages are accumulated in the messaging server for further processing. There is better performance based on the modularity of the system. The main contribution of this work is to provide a base architecture for building smart parking systems by considering all elements that could be easily connected by using several technology components.

The paper is structured as follows. Section 2 describes other work reviewed in this work. Section 3, describes a brief analysis of the previous works. Then, the Section 4 explains the proposed smart parking solution. Later, Section 5 presents the results gotten after the implementation of the proposed solution. After, Section 6 discusses strengths of the proposed solution, and finally, the present work is concluded in Section 7.

2. Related Works

To perform the following review, a methodology for selecting and discarding papers was applied as described in our previous research work [1]. The methodology used is based on a semi cyclic process and combined with a systematic review. It is comprised of three phases: plan phase, perform review phase and report results phase. First of all, during the plan phase, search strings and digital research repositories were defined. In the perform review phase, search strings were adapted for every repository, preliminary results were collected, relevant information was extracted and candidate papers were selected. Finally, during the last phase, selected documents were fully reviewed. The methodology used is shown in Figure 1.

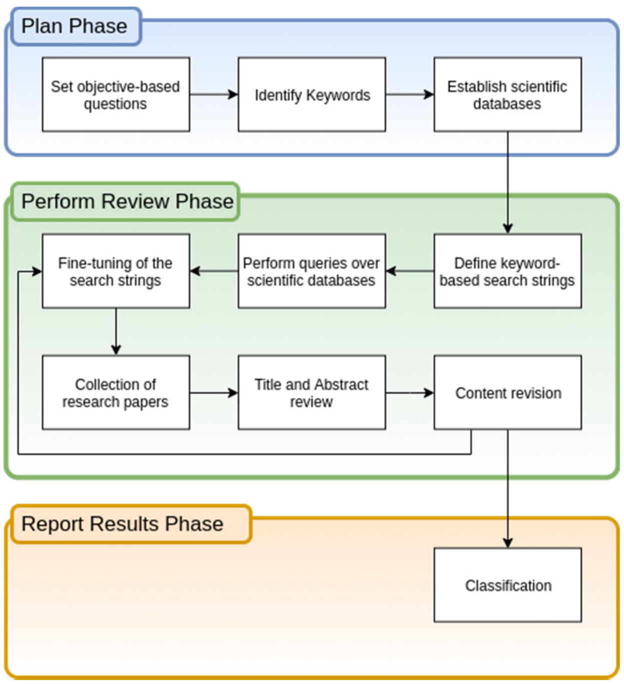

Plan Phase
Set objective-based questions
Identify Keywords
Establish scientific databases
Perform Review Phase
Fine-tuning of the search strings
Perform queries over scientific databases
Define keyword-based search strings
Collection of research papers
Title and Abstract review
Content revision
Report Results Phase
Classification

Figure 1. Detailed research method.

This work reviewed the technologies used by smart parking solutions, such as networking, backend technologies and sensor technologies. Most of the analyzed research papers had similar features, but different components in terms of sensors, network technologies and backend infrastructures. For example, [14] uses FIWARE and oneM2M in addition to the LoRaWAN infrastructure to solve the interoperability of multiple IoT protocols. On the other hand, [11] only supports Wi-Fi; to go around the compatibility problem but have a distributed architecture to capture all the information. While on one side the communication protocol makes easier the overall implementation of the system, it brings the interoperability to the table. Meanwhile, supporting only one protocol like Wi-Fi do not have such a problem, but due to the nature of the protocol it is necessary to create a distributed architecture to deal with information gathering needs.

The work called an IoT-based E-Parking System for Smart Cities [11] describes a system composed of sensors that are connected via Wi-Fi (WLAN) to a local server for detecting the availability of parking spots and getting information about the car. In addition, an ultrasonic sensor, this solution implements a camera for license plate recognition; so, it can detect whether the vehicle is properly parked. If a vehicle is improperly parked it alerts the user to encourage them to park the car properly. This solution uses SMS for communication with users. Another feature of this solution is parking slot reservations. This implementation can be costly because of the use of SMS to communicate with the end-user. This solution uses multiple tiers; one for detection, one for local management, a cloud server to host the backend and the user interface.

Another work [14] is focused on providing system interoperability using middleware based on one M2M standard and FIWARE. The solution was built with different IoT infrastructures (one of which was LoRaWAN) and two different interoperability layers. The solution includes the delivery of parking spot availability information via mobile apps. It also provides free parking slot search, route provisioning, walking routes back to the car and parking expiration ticket reminder functionalities. In terms of system management, a web interface is provided for parking lot managers to simplify parking sensor configuration and provide information including statistics on parking slot usage.

On the other hand, IoT based smart parking system [15] implements a smart parking solution which provides parking lot availability information, parking slot reservation and payments using a mobile app. A key differentiator of this application is the use of timer. This timer is used to trigger an alert and let the user know when the time of permanence has expired so it could be extended. The app also has a parking occupancy confirmation, that expects a user to reserve a spot and then when the sensor detects that a car is in place, it sends a notification through the app. In terms of communications, the sensor uses Wi-Fi and MQTT as the backend to support the application. The infrastructure comprises a raspberry pi that functions as a local server and communicates with the cloud to process information collected.

In [16], a sensor collects the status of a parking lot. The data collected are passed to a microcontroller. The microcontroller forwards the data to a network server through a gateway. At last the network server forwards the data to the application server. The protocol used for communication between the microcontroller and the network server is User Datagram Protocol (UDP), while the communication made on the network server is through the MQTT protocol. In the application server the state of the battery and the parking lots data are shown in real time through a dashboard.

The authors in [17] use sensors for detecting the occupancy of a parking lot. A microcontroller recollects all the data from the sensors and forward to the MQTT Server over WI-FI. The MQTT server forwards the recollected data to a platform that will process and store the received data. In addition, the platform provides a user interface for letting the users reserve a parking lot. It also bills the users for the reservation time of a parking lot. The users are given an RFID card along with a unique login credentials for accessing to parking lot and being identified, respectively.

Solution presented in [18] describes a system that uses sensors that send the occupancy information to an information center via Zigbee. This solution also handles a Bluetooth station used to locate users inside the parking lot. It helps users to find the shortest path to an empty slot using a proposed "Shortest path search algorithm". All the gathered information is sent to over Wi-Fi to a parking management menu. The number of information centers will grow proportionally to the size of each parking lot where the solution is implemented. Likewise, the solution proposed in in [19] uses Zigbee and Wi-Fi. This last solution in addition, uses an IoT Middleware Layer using RESTful communication so it enables data retrieval from external entities.

Real-Time Smart Parking Systems Integration in Distributed ITS for Smart Cities [20] describes a solution for a Distributed Intelligent Transportation System (ITS). The architecture considers the data flow from the edges of the system towards distributed gateway architecture, where the intelligence and storage are distributed across limited geographical areas to avoid some limitations. This architecture is highly scalable since the addition of new components of the system is automatically accommodated by a proportional increase in the number of gateways. For the parking lot system component, the solution works with any kind of detection system since the architecture was built thinking on a highly scalable factor. Collected information is sent to a determined gateway so it can be further processed and sent or presented directly to end-users.

Zigbee is another technology widely used for smart parking, as presented in [21]. A Zigbee router covers each parking slot. The information is sent through a multi-hop network to the Coordinator. It transmits the information to the Smart Gateway which analyzes and joins it with the parking slot position, sending it to the Central Server. Finally, the information received feeds two mobile applications: Parking app (users) and Policeman app (traffic cops). The solutions provide a payment feature using NFC in the smart gateway and smartphone.

Node Based Architecture for IoT is shown in [8]. In this solution, the end-node is an ultrasonic sensor that collects the information. An Arduino Uno is the processing node that connects and sends the information to the cloud server through an ESP8266-01 Wi-Fi Module. The Server feeds the desktop and mobile applications. It brings the possibility to search for parking slots, booking a place and guide the user to it.

In terms of sensor and smart parking system communication, several solutions are proposed. Most of them are focused on wireless communications because they are easy to implement and cheap. Solutions like [8,11,15,22] make use of Wi-Fi technologies for sensor communications. These solutions present multiple limitations like signal range and energy consumption in contrast with other technologies like IEEE 802.15.4 (i.e., LoRaWAN) that are wide-range coverage and low power consumption. Even though Wi-Fi offers greater data rates, this is not considered a strength since sensors should not send a lot of data in a short time. On the other side, technologies based on IEEE 802.15.4 are lighter, so they require little processing power.

As for technologies based on IEEE 802.15.4, there are several IoT-oriented protocols adopted. Solutions such as [18,19,21] use ZigBee for sensor communication. It only presents a drawback; the range of coverage is shorter compared to other IEEE 802.15.4 technologies; despite its ability to send long-distance messages using a mesh topology. Some solutions like [14,16,20] make use of LoRaWAN because of the low power consumption; a feature that enhance battery duration, and a range comparable to cellular communication technologies. The only disadvantage of technologies based on IEEE 802.15.4 is low data rate, but IoT devices does not require huge data rates.

Infrastructure showed in [11] uses Wi-Fi Access points (AP) to gather information from each sensor of for each determined section. This information is sent to every Local parking management system and uploaded to the Central Parking Management System (CPMS). This infrastructure uses more communication resources and needs more hardware devices per parking lot. Hence, it can become too expensive depending on the size of the parking lot facilities.

In [20], authors explain the benefits of local data processing before arriving at the control center. This architecture provides greater reliability, confining faults within each gateway while the rest of the system continues to operate. In [23] local data processing is performed to optimize the monitoring of parking spaces. In our implementation, the data is processed before being stored in the NoSQL repository. This is mandatory since the data collected by the sensors have unnecessary additional information and processing it reduces its size and increases their informational value. In this implementation the data are stored in a NoSQL repository for the versatility it offers, the system generates a large amount of data that must be recorded. The horizontal growth that characterizes this kind of repository is adapted to the cluster of Kubernetes implemented in the system with new nodes that could be implemented to balance the load. In addition, it does not require a large amount of resources to operate.

The most well-known approaches for sending sensor's collected data are through MQTT protocol, HTTP protocol and directly to a database. MQTT stands for Message Queuing Telemetry Transport and it is one of the best ways to send data from a WSN network, since it is designed for networks with high latency and low bandwidth. Another advantage of MQTT is the existence of levels of quality of services which ensures high delivery guarantees. In [8,15,23] they make use of an MQTT broker which implies that the size of the packets and power consumption are less than implementation. In [5], the solution proposed uses the HTTP protocol for sending data from the gateway to the server. However, the use of this protocol generates additional metadata that is not useful for IOT implementations. In [24], the implementation sends the data directly to the database. This data are not processed and the database is vulnerable to injection attacks. Implementations with this approach are less likely to escalate over time because making repeatedly database connections involves high latency in the system. Therefore, adding new parking lots to the system could result in low quality service.

Similarly, in [15], sensors are connected wirelessly via Wi-Fi to a raspberry pi that acts like an intermediate between this and the cloud. This infrastructure uses an IBM MQTT server that allows the mobile application to connect through a secure channel and a 2-factor authentication. In order to ensure proper communication both the raspberry pi and mobile application must be subscribed to a particular channel on IBM MQTT server. The chip used to connect parking sensors on each parking lot has a maximum of 26 different connection, even though this number can increase using a multiplexer, it will also increase cost on a large scale since it would have to be done on each parking lot depending on the number of slots that these would handle.

In [16,19], the described infrastructures are similar since both sense the information from each parking lot slot. Then, send it to a gateway that will not necessarily be for a single parking lot. Later, information is sent to a network server. Finally, the networks server will be in charge of delivering the information to end users through an application or dashboard.

In terms of sensors, the most widely used option is ultrasonic sensors as described in [8,11,15,18,24]; however, these sensors present some problems: noise can interfere with the reading of the parking slot status, and for the general purpose of the sensor, it can detect large objects, even if it is not a car.

Induction Proximity Sensor [24] is used to detect when a vehicle enters or leaves from a parking lot. This sensor is usually buried under the ground. It detects vehicles by measuring the change in the earth's magnetic field as the vehicle passes over. If the magnetic field varies by a certain threshold value, the sensor determines that a vehicle has passed over. This sensor is very good at detecting metal objects over the sensor (such as vehicles) and will not send a false positive if someone walks over the sensor. The disadvantage is that it needs to be buried under the parking spot.

Radio Frequency Identification sensors (RFID) [24], can be used to detect if a vehicle is occupying a parking spot. An advantage of these sensors is that they do not detect other objects, such as people or animals. A disadvantage of these sensors is that it is required that the RFID must be present in the vehicle. The authors also propose LIDAR sensors which are very similar to ultrasonic sensors but use pulses of light instead of RF signals. These sensors are more precise, and their range is greater, but they are more expensive than an ultrasonic sensor.

The solution in [22] implements RFID as sensors. Readers are at the entrance and exit of the parking lot, opening the barriers for the allowed cars with RFID tags. These tags are for registered and temporal users. Each tag belongs to a parking slot, also it helps to collect information about occupation, entrance and exit time. The information collected is sent through Wi-Fi to a cloud server that saves it in a database for future study.

Finally, cameras in [25] are also used to detect parking slots, but they have to be strategically positioned to cover a wide range of spaces, these require more processing and the designed algorithms are not a 100% reliable. Although this solution aims to detect vehicle presence, it lacks of further components to share data with other elements like application servers or user devices.

The following table (see Table 1) provides a summary of the technologies, smart parking layers, features and issues identified among the related works reviewed. The table is structured in the following is composed of five columns. First column is Reference which is used to identify the reviewed work. The Technology column contains the described technology used in the proposed work of the authors pointed out in first column. This technology applies to the components described for deploying a smart parking system. Then, Smart Parking Layer column points out the layer (Sensing, Network Infrastructure and Application) where the technology is used. Finally, the last column is used to identify if the technology used represents a positive feature or represents an issue for that particular implementation by using a ∅ to mark the feature as an issue, whilst a $\surd$ to identify a positive feature that is remarkable from the reviewed work.

Table 1. Summary of technologies, layers, features and issues identified.

Reference	Technology	Smart Parking Layer	Features Identified
[5]	HTTP	Application	∅ Unnecessary metadata
[14]	LoRaWAN	Network Infrastructure	✓ Long-range ✓ Low-power consumption
[11]	Wi-Fi (WLAN)	Network Infrastructure	∅ Short-Range ∅ Power consumption
	Ultrasonic sensors	Sensing	∅ Noise interference
[15]	Ultrasonic sensors	Sensing	∅ Noise interference
	Wi-Fi (WLAN)	Network Infrastructure	∅ Short-Range ∅ Power consumption
	Raspberry	Network Infrastructure	∅ Limited resources ∅ Not a gateway
	MQTT	Application	✓ Message Queuing
[16]	UDP	Network Infrastructure	Prone to packet loss
[18] [17]	LoRaWAN	Network Infrastructure	✓ Long-range ∅ Low-power consumption
	MQTT	Application	∅ Message Queuing
	RFID	Sensing	∅ Invasive ∅ Very short-range coverage
[21]	Wi-Fi (WLAN)	Network Infrastructure	∅ Short-Range ∅ Power consumption
	MQTT	Application	∅ Message Queuing
[8]	Arduino Uno	Sensing	∅ No specific purpose
	Ultrasonic sensors	Sensing	∅ Noise interference
	Wi-Fi (WLAN)	Network Infrastructure	∅ Short-Range ∅ Power consumption
[21]	RFID	Sensing	∅ Invasive ∅ Very short-range coverage
	Wi-Fi (WLAN)	Network Infrastructure	∅ Short-Range ∅ Power consumption
[22]	MQTT	Network Infrastructure	✓ Message Queuing
[24]	Ultrasonic sensors	Sensing	∅ Noise interference
	Direct connection	Application	∅ Scalability ∅ Multiple connections
[25]	Cameras	Sensing	∅ Require more processing ∅ Lack of reliable algorithms

3. Analysis of Previous Works

3.1. Analysis of Works Based on Sensors

The reviewed propose several ways to deploy a smart parking solution. Some of them emphasize on the types of sensors that might be used for deploying such solution as in [24]. Meanwhile, there are others that do not use specific purpose sensors and they attach sensing modules to Arduino or similar devices. These microcontrollers do not have the same energy autonomy properties as specific purpose sensors. Moreover, these microcontrollers are not IP68 compliant which make them no suitable for adverse weather conditions. Sensors must be able to sense the presence of a vehicle no matter the weather. Therefore, sensors in a smart parking solution have to be able to work under severe weather conditions (i.e., rains, dust). Likewise, sensors should be able to transmit information over long-range perimeters with low energy consumption. These sensors should not be invasive as RFID, but effective when determining vehicle presence and avoid false-positive detections. Based on that, magnetic sensors are the ones that best fit the detection requirement and will be part of the proposed solution.

3.2. Analysis of Works That Emphasize Communication Protocols

While Wi-Fi, Bluetooth Low Energy (BLE) and Zigbee are appropriate for short distances (less than 500 m). Network connectivity must reach long distances. In real conditions, parking slots are bigger and cover hundreds of square meters particularly in places like hospitals, universities, schools, cities. LPWAN protocols are suitable for covering long-range distances at low power consumption. Protocols like SigFox, LoRaWAN or NB-IoT could be considered for implementing a smart parking solution. Due to its versatility and openness, LoRaWAN is suitable protocol for a smart parking solution as discussed in [14,16,20].

3.3. Analysis of Works That Focus on Backend Solutions

The reviewed papers propose several ways to deploy a smart parking solution. Some of them emphasize on the types of sensors that might be used for deploying such solution as in [24]. Meanwhile, there are others that do not use specific purpose sensors and they attach sensing modules to Arduino or similar devices. These microcontrollers do not have the same energy autonomy properties as specific purpose sensors. Moreover, these microcontrollers are not IP68 compliant which make them no suitable for adverse weather conditions. Sensors must be able to sense the presence of a vehicle no matter the weather. Therefore, sensors in a smart parking solution have to be able to work under severe weather conditions (i.e., rains, dust). Likewise, sensors should be able to transmit information over long-range perimeters with low energy consumption. These sensors should not be invasive as RFID, but effective when determining vehicle presence and avoid false-positive detections. Based on that, magnetic sensors are the ones that best fit the detection requirement and will be part of the proposed solution.

Wi-Fi, BLE and Zigbee are appropriate for short distances (less than 500 m). Network connectivity must reach long distances. In real conditions, parking slots are bigger and cover hundreds of square meters particularly in places like hospitals, universities, schools, cities. LPWAN protocols are suitable for covering long-range distances at low power consumption. Protocols like SigFox, LoRaWAN or NB-IoT could be considered for implementing a smart parking solution. Due to its versatility and openness, LoRaWAN is suitable protocol for a smart parking solution as discussed in [14,16,20].

In terms of messaging, some solutions agree that MQTT is a suitable protocol for moving information collected from sensors to a backend infrastructure as in [17]. However, such architecture proposes the use of a unique server with no redundancy for collecting information and then storing information in a database. In that scenario, data would be lost as sensors do not have a mechanism for queuing packets. Therefore, not considering a redundancy mechanism would compromise the availability of information for further processing to obtain a particular result.

Smart parking solutions are intended to serve thousands of users (citizens). The backend infrastructure is in charge of processing and sharing data with their different actors. These actors could be other systems, services or citizens. The number of actors is constantly growing and tend to be unlimited. An appropriate way to keep up with this growth is by using components that could grow at the same rate and that provide appropriate levels of availability. Although using one physical server per service involved is a best practice, it does not allow a high level of scalability and availability. This a technical and economic limitation that would considerably increase the cost of the final solution. An approach to address these issues is by clustering nodes involved and avoid using physical servers. Kubernetes are a feasible alternative to configure a cluster of every service involved (i.e., data storage, messaging, publishing) as every node has its own computational resources [23].

4. Proposed Solution

The proposed solution shown in Figure 2 aims to cover some of the limitations discussed in the previous section. This solution is comprised of three major components: sensing infrastructure, backend infrastructure and user interface. First, sensing infrastructure oversees collecting information in real-time of the state of a parking slot whether if it is busy or available using a long-range communication. Those information payloads are sent to the backend-infrastructure with the help of a MQTT server. Then, in the backend infrastructure, a software component (Raw Data Bridge) moves those payloads to a Message Queue (MQ) Server. In that instance, there are two other software components: one for parsing data and another for processing payloads before its storage. The parsing component will take raw payloads from a "in" queue of the MQ Server and then post it over a new "out" queue of that MQ Server. Later, the logic component takes the payloads posted over the out queue and store them into a NoSQL data repository for publishing over a REST API. Finally, the stored data is consumed by a set of microservices which are used by mobile and web applications.

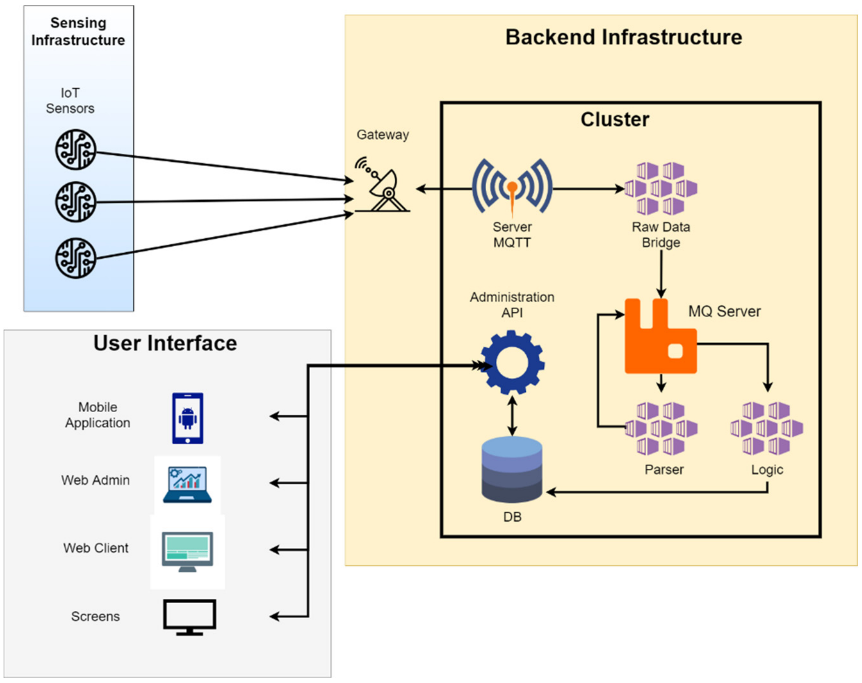


Figure 2. Proposed solution architecture.

4.1. Environment Setup

The previous architecture was tested over a small parking lot of a university campus. The environment consisted of 10 sensors connected to a single MultiTech Conduit gateway through LoRaWAN protocol. The gateway was connected through MQTT protocol to a MQTT Server node, an MQ Server node and a DB Instance. All the previous devices were deployed and connected in an isolated Local Area Network within the university premises. The REST API was deployed for public consumption over a free domain.

Every node of the cluster has the following hardware capabilities as shown in the following table (see Table 2).

Table 2. Hardware resources per node.

Feature Per Node	Amount
vCPUs	1
RAM	1.75 GB
Hard Drive	100 GB

To measure the performance of the system, response time tests were carried out when carrying out a series of transactions taking into account the proposed scenario (based on Kubernetes) and the one used by some implementations (based on virtual machines). The results obtained were then were plotted in charts to understand the behavior of the proposed solution and identify if there was potential impact on system performance. The results obtained during the tests are shown in the next section (see Section 5).

4.2. Sensing Infrastructure

As discussed before, sensing infrastructure should be able to communicate over kilometers as smart parking solutions are intended to cover wide areas. In this sense, the usage of LPWAN protocols/technologies such as LoRaWAN which allows to communicate over long-range distances with low power consumption could be suitable. The proposed solution has used LoRaWAN because of its openness which allows to the developer to implement all the components of its architecture which is composed of end-devices, gateway, network server and application server (see Figure 3). As an additional information, it is important to indicate that the proposed architecture is based on the specification 1.0.2 of LoRaWAN. The components within the architecture perform specific tasks. For example, smart parking sensors are in charge of sensing vehicle presence and passing such data to the gateway over LoRaWAN network. The gateway transforms LoRa payloads into TCP/IP packets which are delivered to the network and application servers.

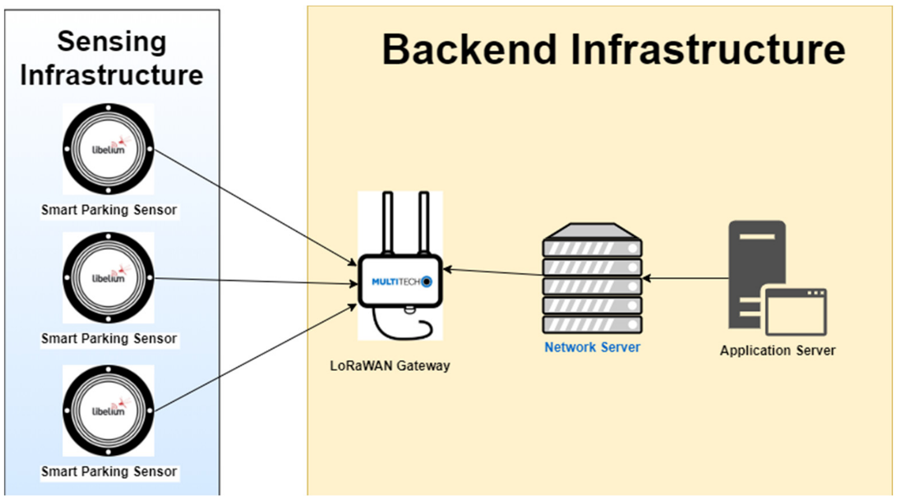


Figure 3. Smart parking sensing infrastructure based on LoRaWAN.

For the implementation of the proposed solution, we have used sensors manufactured by Libelium which support SigFox and LoRaWAN protocols. These devices use magnetic sensors for detecting the presence of a car and they are IP68 compliant, which means that are dust-resistant and water-resistant (up to 1.5 m deep). Their operation frequency is rated between 863 MHz to 923 MHz which is an unlicensed ISM specter. For this implementation, the US902–918 MHz frequency was selected as recommended by the manufacturer according to the region of implementation. It is also important to indicate that these sensors hold a power autonomy of 4–6 years according to the manufacturer's specifications.

On the other hand, the proposed solution used the MultiTech Conduit IP67 Base Station for implementing the LoRaWAN Gateway. It supports public and private LoRa network deployments working at 868 MHz and 915 MHz ISM bands, and it is designed for outdoor deployments. The main task performed by this device is to convert Radio Frequency (LoRa) packets into IP packets and vice versa; in other word, this device routes the uplink and downlink packets. An uplink packet is a packet sent from to the sensor to an application whilst a downlink packet is packet sent from an application to a sensor. The Gateway used in the proposed solution has a network server embedded. This server is in charge of routing uplink packets to the respective application server or route downlink packets to a particular sensor through the gateway. In addition, the Conduit Gateway is able to provide an application server to forward packets via MQTT protocol to a third-party application server.

Security within the Sensing Infrastructure

In terms of security, LoRaWAN supports two joining methods. Activation by Personalization (ABP) and Over The Air Activation (OTAA). For the present solution, the OTAA method was configured in the end-nodes, network and application server. For OTAA mode, the following parameters and keys have to be pre-configured and shared: **DevEUI (Device Unique Identifier)**, **AppEUI (Application Unique Identifier)** and **AppKey (Application Root Key)**. The LoRaWAN specification 1.0.2 specifies that the **AppKey** is an AES-128 bits key. This key is used to derive two other keys used to cipher the payloads. Those keys are AES-128 bits keys and are **Network Session Key (NwkSKey)** and **Application Session Key (AppSKey)**. Each key is used by each server so that the **Network Session Key (NwkSKey)** is used between the end-device and the network server to validate the integrity of the messages (MIC fields). The **AppSKey** is used between the end-device and the application server to encrypt the payloads providing confidentiality [26].

4.3. Backend Infrastructure

This part of the solution is in charge of processing the information collected by the sensors. The backend was mainly built over a cluster of Kubernetes. Kubernetes is an open-source system that allows to automate the deployment of applications based on containers. In this solution, Kubernetes have been used to deploy database server, message queue server, MQTT server and the REST API. The proposed architecture is shown in Figure 4.

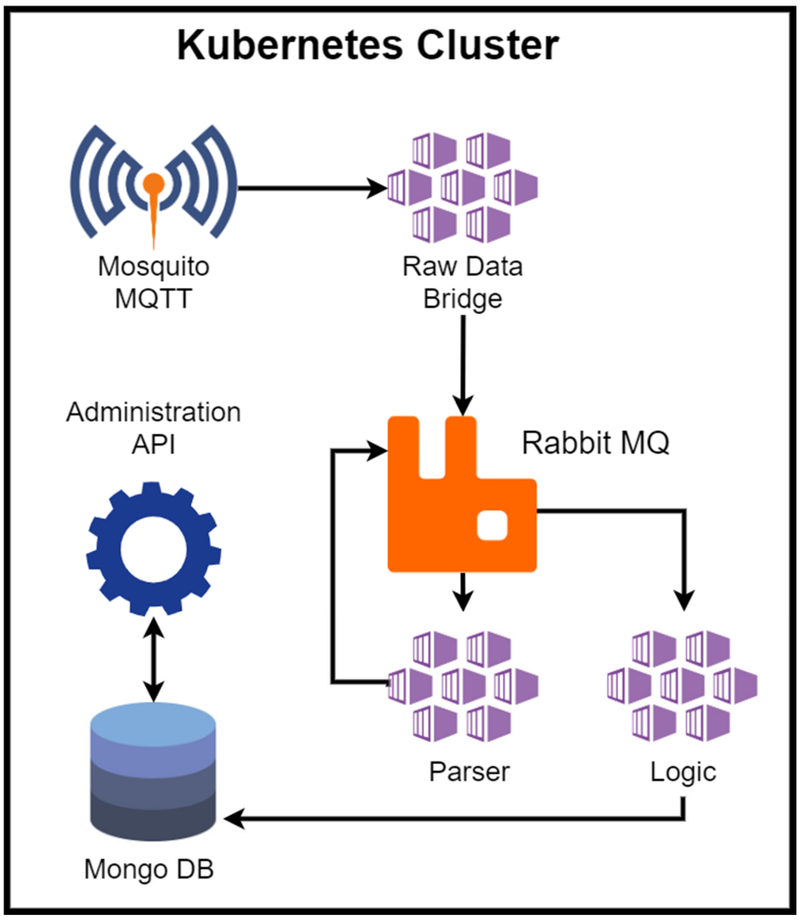
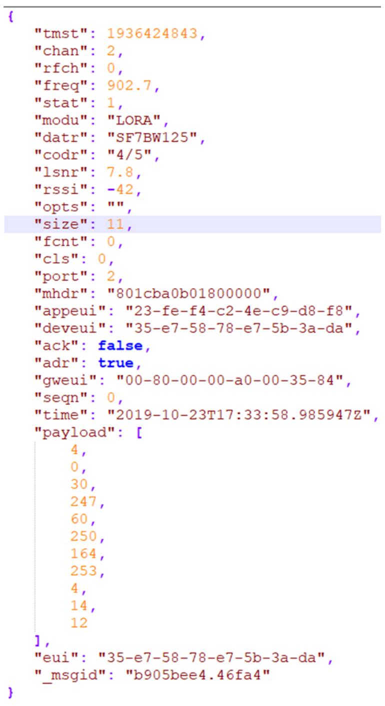


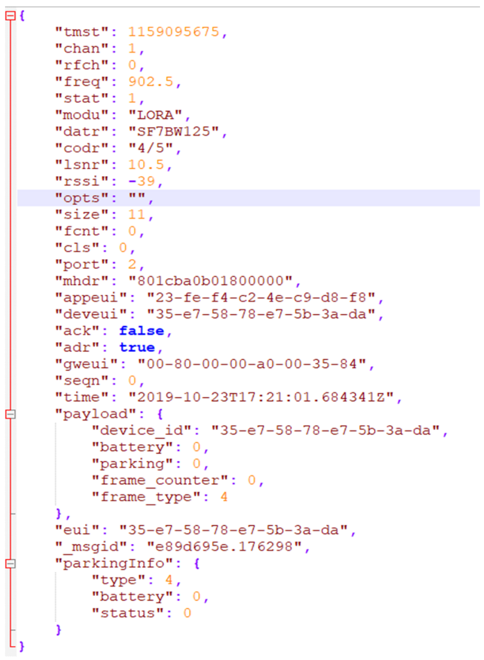
Figure 4. Smart parking solution backend-architecture.

The main point of entry is the MQTT Server which is based on Mosquito (an open-source MQTT broker). The information forwarded by the Network Server is collected through this server via MQTT protocol. MQTT protocol delivers a lightweight method to handle messaging over a publisher/subscriber model [27]. The information received by Mosquito comes in Javascript Object Notation (JSON) format as shown in Figures 5 and 6. Mosquito server forwards unencrypted payloads to a software component called "Raw Data Bridge".



```
{
   "tmst": 1936424843,
   "chan": 2,
   "rfch": 0,
   "freq": 902.7,
   "stat": 1,
   "modu": "LORA",
   "datr": "SF7BW125",
   "codr": "4/5",
   "lsnr": 7.8,
   "rssi": -42,
   "opts": "",
   "size": 11,
   "fcnt": 0,
   "cls": 0,
   "port": 2,
   "mhdr": "801cba0b01800000",
   "appeui": "23-fe-f4-c2-4e-c9-d8-f8",
   "deveui": "35-e7-58-78-e7-5b-3a-da",
   "ack": false,
   "adr": true,
   "gweui": "00-80-00-00-a0-00-35-84",
   "seqn": 0,
   "time": "2019-10-23T17:33:58.985947Z",
   "payload": [
       4,
       0,
       30,
       247,
       60,
       250,
       164,
       253,
       4,
       14,
       12
   ],
   "eui": "35-e7-58-78-e7-5b-3a-da",
   "_msgid": "b905bee4.46fa4"
}
```

Figure 5. JSON Payload sent by sensor node and forwarded by Mosquito.



```
{
    "tmst": 1159095675,
    "chan": 1,
    "rfch": 0,
    "freq": 902.5,
    "stat": 1,
    "modu": "LORA",
    "datr": "SF7BW125",
    "codr": "4/5",
    "lsnr": 10.5,
    "rssi": -39,
    "opts": "",
    "size": 11,
    "fcnt": 0,
    "cls": 0,
    "port": 2,
    "mhdr": "801cba0b01800000",
    "appeui": "23-fe-f4-c2-4e-c9-d8-f8",
    "deveui": "35-e7-58-78-e7-5b-3a-da",
    "ack": false,
    "adr": true,
    "gweui": "00-80-00-00-a0-00-35-84",
    "seqn": 0,
    "time": "2019-10-23T17:21:01.684341Z",
    "payload": {
        "device_id": "35-e7-58-78-e7-5b-3a-da",
        "battery": 0,
        "parking": 0,
        "frame_counter": 0,
        "frame_type": 4
    },
    "eui": "35-e7-58-78-e7-5b-3a-da",
    "_msgid": "e89d695e.176298",
    "parkingInfo": {
        "type": 4,
        "battery": 0,
        "status": 0
    }
}
```

Figure 6. JSON Payload generated by Mosquito MQTT.

The Raw Data Bridge takes the payload received from Mosquito and delivers it to an inbound message queue with a specific topic. The Raw Data Bridge acts as a producer that publishes messages to an Exchange. This exchange delivers messages to a queue based on the type of exchange (direct, topic and fanout). Based on that behavior, RabbitMQ Server (an open source message broker that natively supports AMQP protocol and others like MQTT, WebSocket, etc.) delivers messages to other exchanges that will be later consumed by the Parser and Logic components respectively. These components were built in C#. The Parser component, is a software program that transforms payloads received from the sensor to a lighter JSON format that will be later stored in a database repository. On the other hand, payloads processed by the Parser are then published to a new Exchange in the RabbitMQ Server, which will be later consumed in the Logic component. The Logic component is in charge of storing parsed information into a NoSQL repository (MongoDB was used in this implementation). MongoDB stores enriched information generated by the sensors. Finally, all the information stored in the database is exposed through an API REST running over IIS with ASP.NET Core.

The following versions were used in the proposed architecture RabbitMQ (Version 3.7.7) developed and maintained by VMWare located in Palo Alto, CA, United States, MongoDB (Version 4) developed and maintained by MongoDB Inc, located in New York City, NY, United States and Mosquitto (Versión 1.4.12) supported and maintained by Eclipse Foundation, located in Ottawa, Canada, for every particular component of the backend infrastructure detailed in Figure 4.

REST API

This system is intended to be integrated with any type of applications e.g., mobile or web; therefore, an API REST interface to expose several services that would be consumed by any other actor if required was implemented. This interface possesses a security feature to prevent unauthorized access implemented using Auth0 which is an authentication token platform. Authentication tokens are used to validate requests to the services. If an actor fails to provide such token, the service will return an HTTP 403 error (Forbidden). The API has been well documented with the aid of swagger UI which is an open source project to render API documentation in HTML [28]. Swagger (see Figure 7) helps developers to clearly understand the way to consume a particular service. It allows to specify HTTP verbs and parameters with particular format that need to be sent to consume a service.

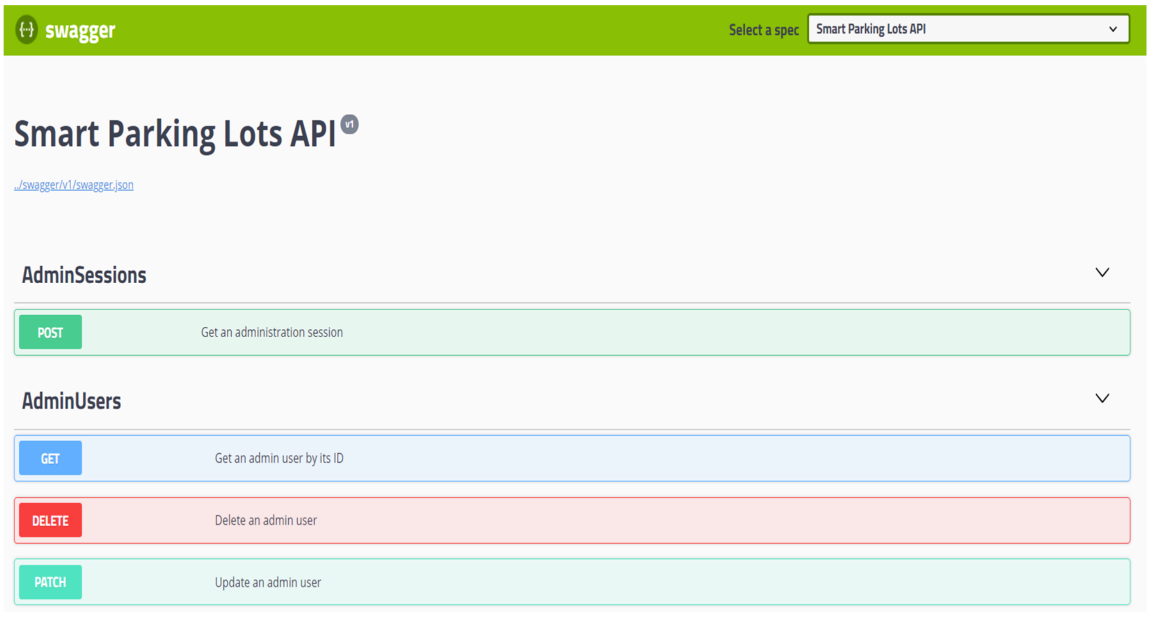
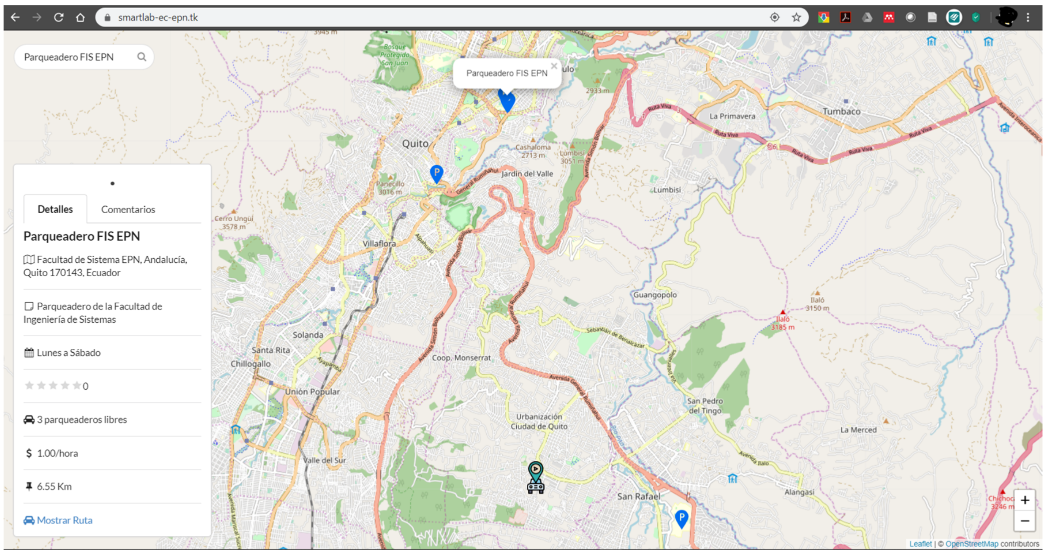


Figure 7. REST API documented with swagger.

4.4. User Interface

Two types of applications were developed: one for end users and another for system administrators. The end user applications are composed of a mobile application for Android and a web application.

The main features created for the end users included in both mobile and web applications are viewing parking lots near the location of the user, displaying information about a parking lot (cost, opening hours, location) and rating a parking lot (see Figure 8).



Detalles | Comentarios

Parqueadero FIS EPN

Facultad de Sistema EPN, Andalucía, Quito 170143, Ecuador

Parqueadero de la Facultad de Ingeniería de Sistemas

Lunes a Sábado

☆☆☆☆☆ 0

3 parqueaderos libres

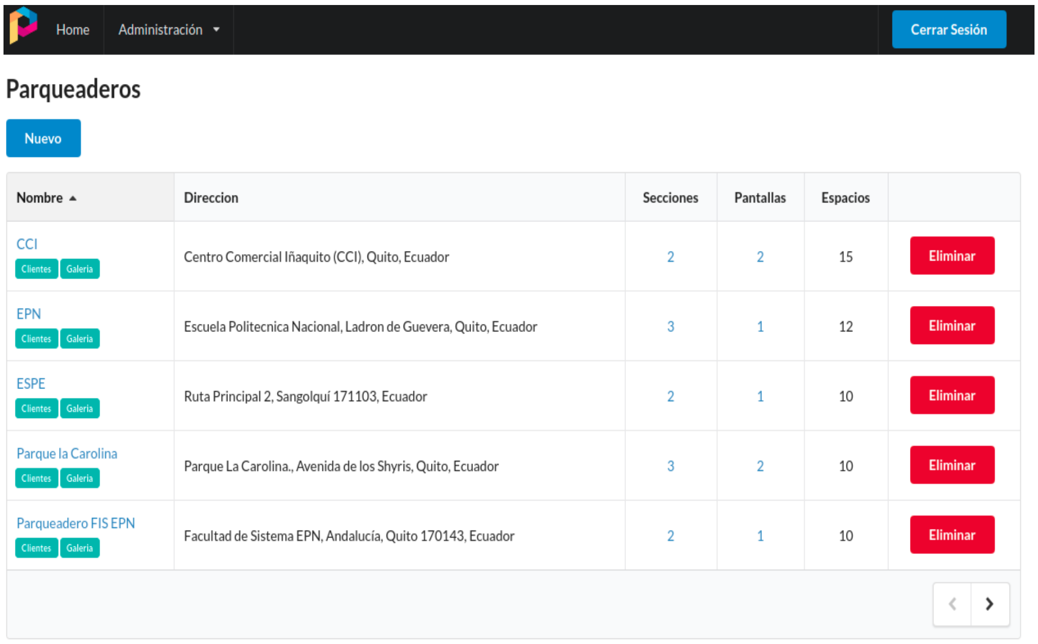
$ 1.00/hora

6.55 Km

Mostrar Ruta

Figure 8. Smart parking web user interface.

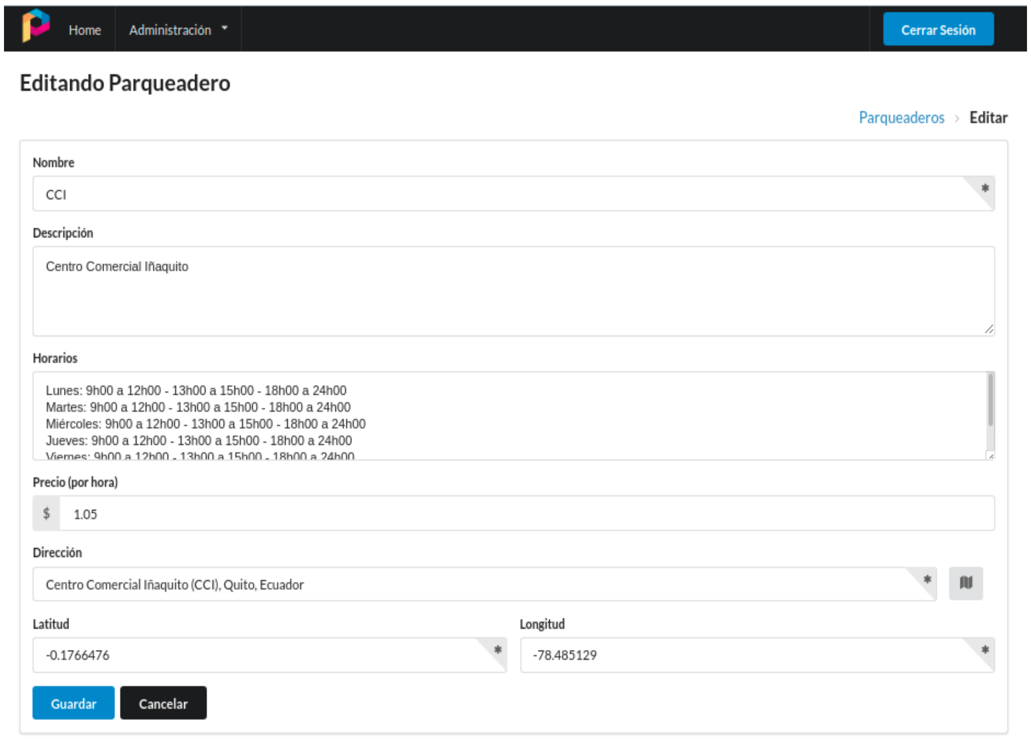
On the other hand, the main features created for system administrators includes: user management (add, update, delete), parking lot management (add, update, delete) including features related with parking sections, screens, RFID cards, parking levels, etc. (see Figures 9 and 10).



Parqueaderos

Nombre ▲	Direccion	Secciones	Pantallas	Espacios	
CCI Clientes Galeria	Centro Comercial Iñaquito (CCI), Quito, Ecuador	2	2	15	Eliminar
EPN Clientes Galeria	Escuela Politecnica Nacional, Ladron de Guevera, Quito, Ecuador	3	1	12	Eliminar
ESPE Clientes Galeria	Ruta Principal 2, Sangolquí 171103, Ecuador	2	1	10	Eliminar
Parque la Carolina Clientes Galeria	Parque La Carolina., Avenida de los Shyris, Quito, Ecuador	3	2	10	Eliminar
Parqueadero FIS EPN Clientes Galeria	Facultad de Sistema EPN, Andalucía, Quito 170143, Ecuador	2	1	10	Eliminar

Figure 9. Web administration interface list parking lots.



Editando Parqueadero

Nombre
CCI

Descripción
Centro Comercial Iñaquito

Horarios
Lunes: 9h00 a 12h00 - 13h00 a 15h00 - 18h00 a 24h00
Martes: 9h00 a 12h00 - 13h00 a 15h00 - 18h00 a 24h00
Miércoles: 9h00 a 12h00 - 13h00 a 15h00 - 18h00 a 24h00
Jueves: 9h00 a 12h00 - 13h00 a 15h00 - 18h00 a 24h00
Viernes: 9h00 a 12h00 - 13h00 a 15h00 - 18h00 a 24h00

Precio (por hora)
$ 1.05

Dirección
Centro Comercial Iñaquito (CCI), Quito, Ecuador

Latitud
-0.1766476

Longitud
-78.485129

Figure 10. Web administration interface editing parking lots.

5. Results

In the following, some important achievements of the proposed solution are presented:

The inclusion of Kubernetes within this architecture helped to have several instances of services running at the same time with limited OS capabilities. Kubernetes allowed to focus on the implementation of the service itself rather than configuring the whole server and the operating system for developing high available infrastructure. This technology runs over Linux servers and its management is easy to handle. Using containers in this implementation provided flexibility and scalability without much technical effort and hardware resources. In addition, capabilities of Kubernetes allowed the proposed solution to become a "software as a service" solution since it delivers all the layers of the smart parking solution.

The following Figure 11 shows a comparative chart between the use of containers and virtual machines when deploying and RabbitMQ server. The chart shows one hundred tests and the time taken in collecting data sent from a sensor by a RabbitMQ consumer. The tests were performed by using a single sensor and one consumer, respectively. The orange line corresponds to the behavior of the RabbitMQ consumer over a container whilst the yellow line represents a consumer deployed over a virtual machine (VM). The average time taken to consume the information over a RabbitMQ container deployment borders 282 milliseconds whilst the average time of the same consumer over a VM borders 348 milliseconds which represents an increase of approximately 22%. This shows that the proposed approach in this paper is significantly better in terms of performance compared to a VM deployment. The architecture proposed helps to increase the performance of the solution.

RabbitMQ fulfilled the purpose of having a centralized element to process payloads and take different actions depending on structures or content. The fact of publishing and subscribing into topics allowed to enrich raw data for further processing and storage into MongoDB.

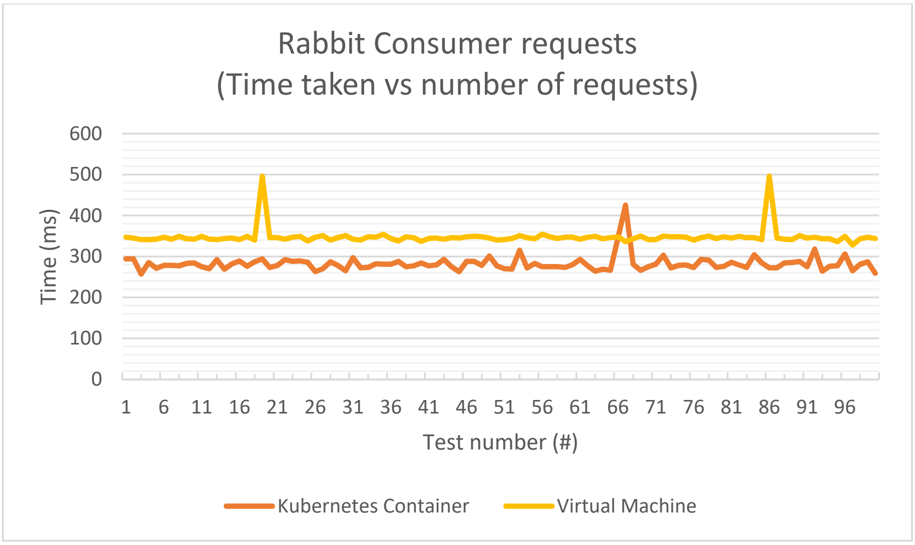


Figure 11. Time taken to process RabbitMQ Consumer requests.

Mosquitto acted as a pass-through bridge which was able to deliver all generated payloads from different sensors through the same gateway. At this point, there was no need to perform additional development to support such feature. The following chart (see Figure 12) denotes the time taken in milliseconds by the Mosquitto component to process a set of requests. The Kubernetes deployment took an average of 156 ms to process a hundred request whilst the VM version took an average of 155 ms to process the same number of requests. In both scenarios, the maximum amount of time taken was 172 ms. However, the minimum amount of time taken by the Kubernetes deployment was 149 ms whilst the VM scenario tool 146 ms, this scenario took 3 ms less. In terms of general performance, Mosquitto behaved very similar in both scenarios showing that performance was not degraded because of the Kubernetes deployment making it a feasible component that could be deployed over the proposed architecture.

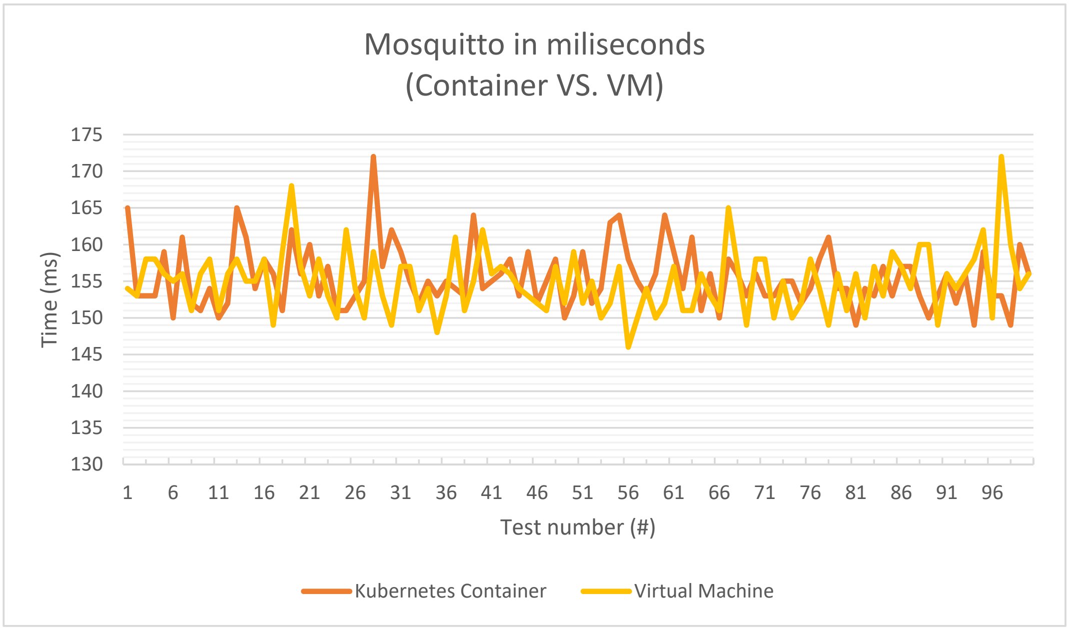


Figure 12. Time taken to process Mosquitto requests.

MongoDB, a NoSQL database, allowed the generation of complex information structures. The selected structures for the proposed solution were JSON based documents which contains all the required information. These documents can handle several fields of information without performing complex queries between different tables (as in standard SQL databases). NoSQL repository did not degrade the performance of the solution.

Several tests were performed to measure the performance impact over MongoDB when performing several requests against the database. The chart in Figure 13 shows the performance of MongoDB in both deployment scenarios. First of all, the orange line shows the behavior in a Kubernetes scenario where the average time to process a request was 0.51 ms, the maximum time was 0.78 ms and the minimum was 0.43 ms. On the other hand, regarding the VM deployment the average time showed 0.57 ms, the maximum was 0.93 and the minimum was 0.48 ms. With these results, it could be said that the Kubernetes deployment is 10.5% faster than the VM deployment. Thereby, this difference helps to ratify that the inclusion of this component in the architecture with the Kubernetes deployment scenario, improves the performance of the system.

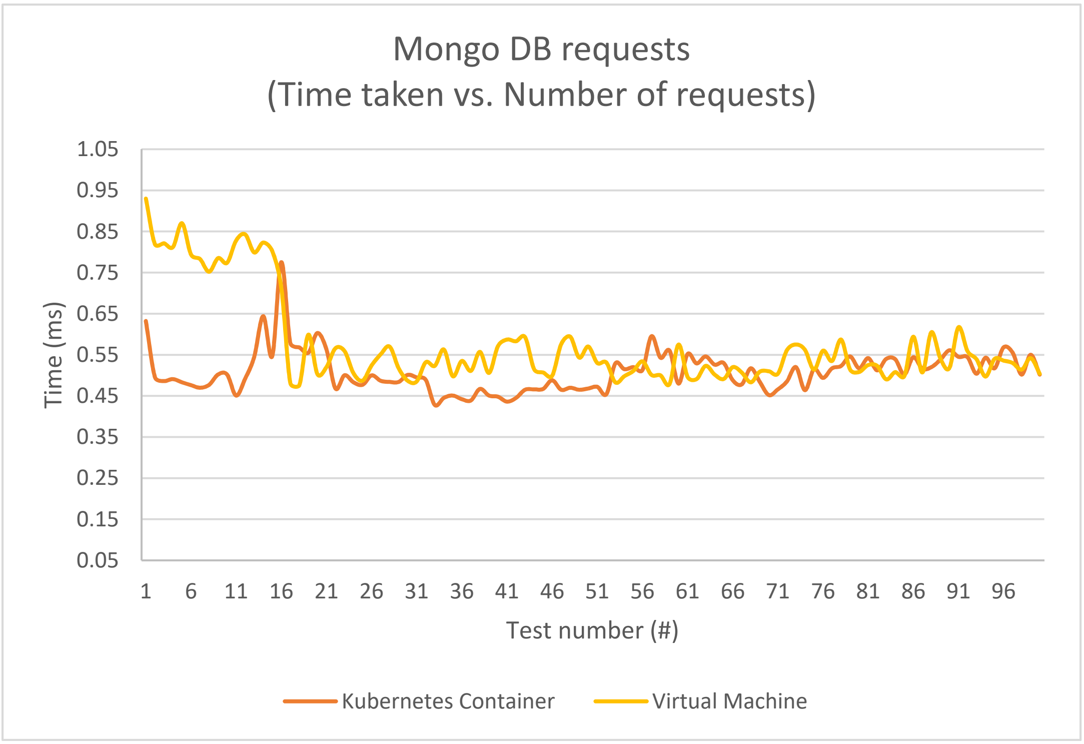


Figure 13. Time taken to process MongoDB requests.

The following chart (see Figure 14) shows the time taken to process information generated by the sensor and written it into MongoDB repository. As shown, the inclusion of an MQTT server and a RabbitMQ server did not degrade the performance of the solution. According to the chart, the average time borders 5 milliseconds which is an appropriate time for real-time solutions. This chart is not considering the reporting time pre-configured in the sensor which has been established in 1 min by default; anyhow, every minute information would be processed and ready to be consumed in less than one second. These tests were performed with one sensor. The whole solution initially considers 10 sensors sensing and sending information every minute, if all sensors transmit at the same time, the maximum delay that the system would experiment would be about 1 s which is still a proper time for a real-time solution.

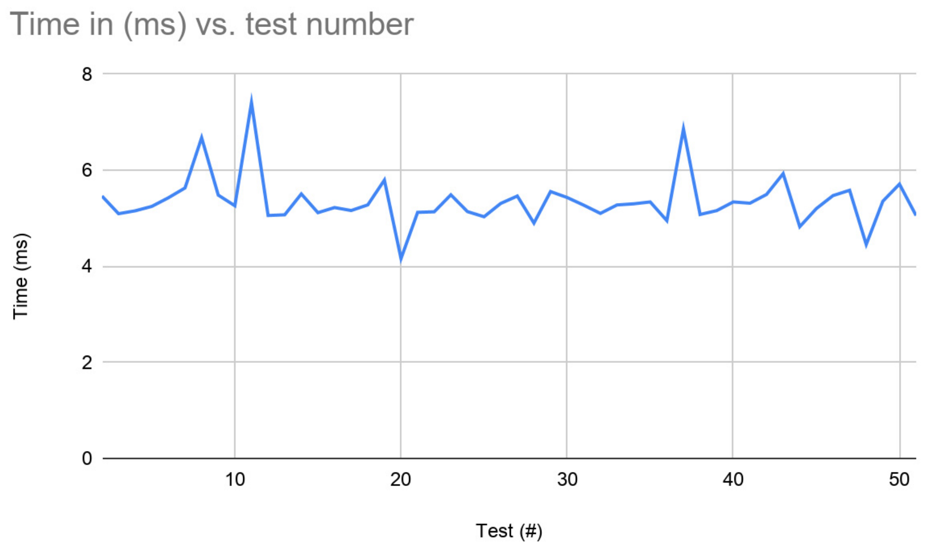


Figure 14. Time taken to write data in a NoSQL repository.

The proposed solution provides several useful features compared to previous solutions which can be divided from two perspectives. First, from the functional perspective, the proposed solution provides useful features such as wireless vehicle detection which uses specific purpose sensors to sense a vehicle. In addition, including an IoT protocol (LoRaWAN) for transmitting information, the proposed solution allows to reduce power consumption and cover larger areas, in contrast to other works that have used short-range protocols for transmitting small payloads of data. The proposed solution does not require installing tags in vehicles to detect slot occupancy. Our solution uses RFID as an additional security mechanism for sites such as housing complexes or buildings where users have to verify their identity before leaving the parking site. In addition, the proposed solution can easily escalate since it has implemented a messaging server to keep generated data by the sensors; if there is an increase of messages, these servers can increase its capacity either to keep or process more data at the same time without affecting system's performance. Likewise, the inclusion of Kubernetes allows the proposed solution to increase its capacity without affecting the overall performance. Kubernetes are containers with reduced OS functionalities used only to deploy a particular service or set of services. Using Kubernetes in the proposed architecture, the proposed system allowed to build and deploy a cluster of services with high availability making the proposed solution fault tolerant. Compared to traditional virtualization, containers have the flexibility to build an image once and deploy it as many times as required. From the reviewed works, other solutions suggested the use of traditional virtualization which demands more resources and they are tied to an installed OS to deploy the service. In terms of security when deploying Kubernetes container, it is easy to detect vulnerabilities and patch them during the building process rather than waiting for the deployment process. The proposed infrastructure is deployable to any cloud service provider (i.e., Google Cloud, AWS, IBM Cloud, etc.), and it does not depend on a particular OS or hardware infrastructure. Its level of flexibility, in terms of deployment is bigger than the reviewed works. The proposed architecture is modular compared to reviewed works and it does not depend on a sensor, software or hardware component. The proposed architecture intends to provide an easy deployable smart parking as service system.

Second, from the software components perspective, the proposed solution has a more global conception compared to reviewed works, since it includes several components required for managing a parking place. The components of the system address different needs such as management, usability and integration. First, the proposed approach provides the ability to manage several parking places at once by using the administration software. The person in charge of management can use a web-based management interface to configure several spots, levels, parking monitors or parking places. There is a software module in place to manage and configure parking screens. In addition, there is a software module for handling and registering users and parking managers. This solution also provides a module so that end-users can locate, get information and rate parking places through a mobile or a web application. The most important feature of the proposed solution in contrast to reviewed works is that it provides a REST API which is a best practice for integrating our whole backend infrastructure to any mobile or web application. It is also important to indicate that we documented all services with Swagger which helped to build a comprehensive documentation, so that any developer can integrate or build customized client interfaces.

6. Discussion

People looking for a parking spot is a source of traffic and air pollution, because of the additional time and fuel people must spend until they find an available spot. A solution for this problem is a smart parking system that comprises deploying sensors on parking lots, processing the retrieved data and making the information available for people looking for a parking spot. Several smart parking solutions are proposed with completely different architectures and technologies. With the aim of improving the existing solutions, a new architecture is proposed using state of the art technologies and solve the parking search problem.

The solution proposed in this paper manages information through a message broker, this is a crucial part of the infrastructure since it reduces the processing load to the rest of the system. Another advantage of using a message broker is sending only the necessary information to the rest of the system. This prevents the inclusion of unnecessary data generated by most communication protocols. In this solution it is only required to know whether a slot is empty or occupied.

From the results obtained in the previous section, graphs have been generated from the previous results that show the behavior of our components based on Kubernetes vs. deployments in VMs. These tables show the maximum, average and minimum times obtained during the test rounds.

Frist of all, the entry point of our proposed architecture is the Mosquitto Server. This component is in charge of collecting payloads received from the gateway. From the test performed, we figured out that this particular component could worked very similar on both deployment scenarios. Mosquitto is 1 ms in average slower in the Kubernetes deployment; however, this is not as significant as expected since it represents 1% approximately. Anyhow, this component could be tuned up to improve the response times and probably that time could be reduced. In the proposed scenario this component showed a good performance for being the entry point of the data. The results summarized for this component are shown in Figure 15.

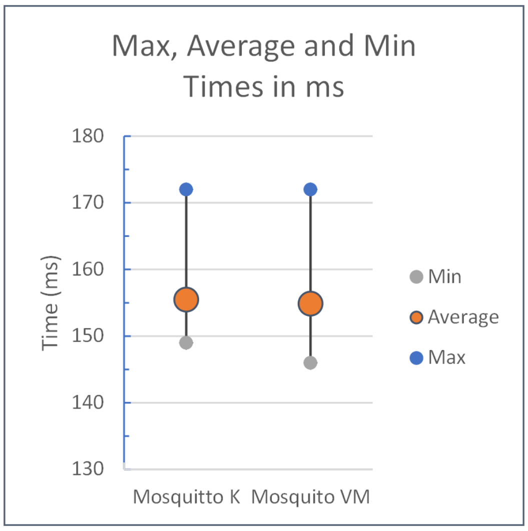


Figure 15. Mosquitto Kubernetes vs. virtual machine (VM) summarized times.

Another important component of the backend infrastructure is the Rabbit MQ server. Their main tasks are to collect raw payloads, enrich them and let them ready to be stored in the NoSQL repository. From the results obtained (see Figure 16), it could be clearly seen that the component in our approach has a considerable better performance than the approach based on VM deployment. The inclusion of this component in Kubernetes environment has shown that our approach is in average approximately 22% faster than the VM approach. Considering that this component will have to process data several times, it is important to have good response times to prevent delays in the delivery of information. These results show that Rabbit MQ performs much better in this environment in spite of having less hardware resources than a virtual machine. Therefore, the proposed approach improves the performance of the system.

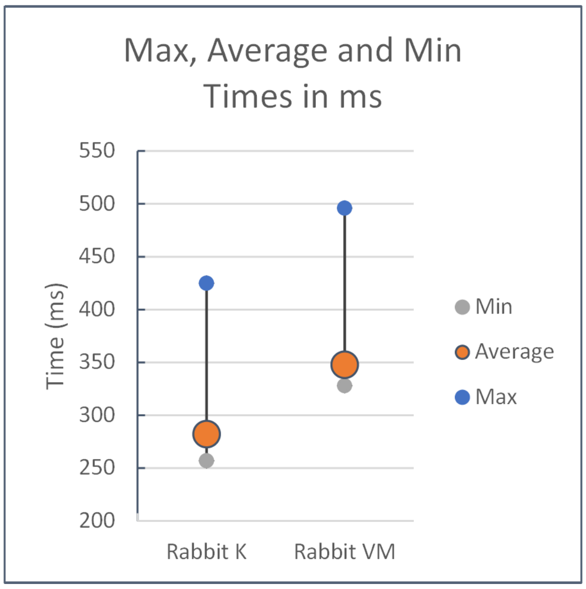


Figure 16. Rabbit MQ Kubernetes vs. VM summarized times.

Finally, the last component of our backend infrastructure is the NoSQL repository which will store all the enriched payloads delivered by Rabbit MQ. From the results shown in Figure 17, it could be seen that the deployment based on Kubernetes if faster in terms of response time, it is in average 10.5% better than the VM approach. Considering that the repository will have to provide and store information at the same time, the obtained response time is very adequate and would help to have a better performance in general. Anyhow, the times obtained within the VM approach are not bad at all and could work for a basic implementation. However, the approach proposed in this architecture could work for medium or complex systems.

The use of Kubernetes, has helped to resource the amount of resources required to deploy an instance of a particular component. In addition, it has provided flexibility at the moment of deploying additional images of the same component as every new node is based on a previous tested and working image. From the experience of the team, we spent less time generating docker images than VMs for every particular component, it could be said that it reduced our work in at least 60%. Finally, the use of Kubernetes to deploy nodes with the "required and necessary" elements to work rather than installing a whole VM with its OS from scratch. The node deployment schema offered by the Kubernetes approach is very useful when escalating solutions as it is easier and faster.

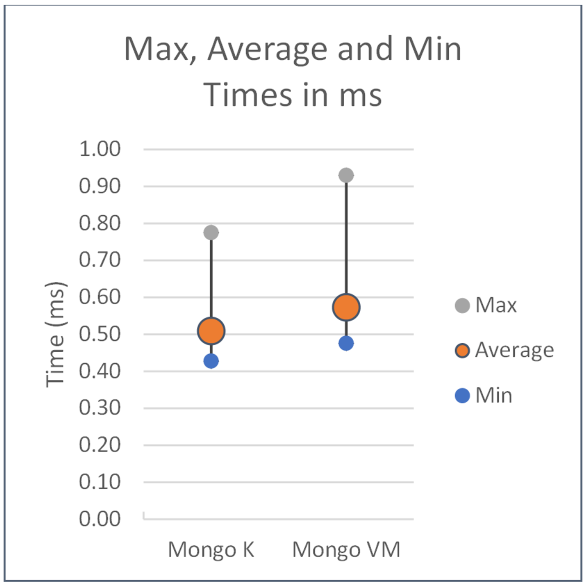


Figure 17. MongoDB Kubernetes vs. VM summarized times.

7. Conclusions

This work presents the implementation of a smart parking architecture that solves some of the problems of today's big city. A series of applications is provided to the end user to accelerate the parking process of their vehicle in a convenient location. Additionally, it is important to indicate this work has shown how LoRaWAN is very useful for implementing smart parking solutions since allows sensing and communicating the state of a parking slots within long-range distances using low amount of energy. Moreover, this work also has tested and verified that Kubernetes and open-source platforms are suitable for building smart parking solution as software as a service since they deliver scalability and availability features without performing complicated configuration tasks.

Finally, it is important to indicate that some of the benefits of implementing smart parking solutions are reduction of the search time for available parking spaces, reduction of fuel consumption by the vehicles, reduction of carbon emissions of the vehicles and reduction of vehicular traffic in urban areas.

Author Contributions: Conceptualization, S.G.Y. and J.J.B.; data curation, J.S. and J.L.; funding acquisition, S.G.Y.; investigation, J.S., J.G., J.L., A.U., D.P., J.J.B., S.G.Y.; methodology, J.J.B. and S.G.Y.; project administration, S.G.Y.; resources, S.G.Y. and J.J.B.; supervision, J.J.B. and S.G.Y.; visualization, J.S., J.L., A.U., D.P.; writing—original draft preparation, J.S., J.L., A.U., D.P., J.G., J.J.B.; writing—review and editing, J.J.B., S.G.Y., J.S.; All authors have read and agreed to the published version of the manuscript.

Funding: The authors gratefully acknowledge the financial support provided by the Escuela Politécnica Nacional, for the development of the project PIJ-17-08—"Diseño e implementación de un Sistema de parqueadero inteligente".

Acknowledgments: The authors gratefully acknowledge the financial support provided by the Escuela Politécnica Nacional, for the development of the project PIJ-17-08—"Diseño e implementación de un Sistema de parqueadero inteligente".

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barriga, J.J.; Sulca, J.; Luis, J.L.; Ulloa, A.; Portero, D.; Andrade, R.; Guun, S.Y. Smart parking: A literature review from the technological perspective. *Appl. Sci.* **2019**, *9*, 4569. [CrossRef]
2. Thinking Highways. Smart parking in the thinking city. *Think. Cities* **2016**, *3*, 1–15.
3. Lin, T.; Rivano, H.; Le Mouel, F. A survey of smart parking solutions. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 3229–3253. [CrossRef]

4. Emc, D. *Smart Cities and Communities GDT Smart City Solutions on Intel®-based Dell EMC infrastructure*; Dell EMC Infrastructure: Hopkinton, MA, USA, 2017.
5. Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express* **2019**, *5*, 1–7. [CrossRef]
6. Sandoval, R.M.; Garcia-Sanchez, A.J.; Garcia-Haro, J. Performance optimization of LoRa nodes for the future smart city/industry. *Eurasip J. Wirel. Commun. Netw.* **2019**, *2019*. [CrossRef]
7. Internet, F.; Access, W.; View, C.; Payments, S.M.; View, N.F.C.; Ert, M.A. A survey on LoRaWAN architecture, protocol and technologies. *Future Internet* **2019**, 216. [CrossRef]
8. Gupta, A.; Kulkarni, S.; Jathar, V.; Sharma, V.; Jain, N. Smart car parking management system using IoT. *Am. J. Sci. Eng. Technol.* **2017**, *2*, 112–119.
9. Gopal, D.G.; Jerlin, M.A.; Abirami, M. A smart parking system using IoT. *World Rev. Entrep. Manag. Sustain. Dev.* **2019**, *15*, 335–345. [CrossRef]
10. Vieira, A.; Rosa, I.; Santos, I.; Paulo, T.; Costa, N.; Maximiano, M.; Reis, C.I. Smart campus parking–parking made easy. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11540, pp. 70–83.
11. Sadhukhan, P. An IoT-based E-parking system for smart cities. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017, Udupi, India, 13–16 September 2017; pp. 1062–1066.
12. Alqazzaz, A.; Aloufi, E.; Alharthi, R.; Zohdy, M.A.; Alrashdi, I.; Ming, H. A practical evaluation of a secure and energy-efficient smart parking system using the MQTT protocol. *ICISDM* **2019**, 165–170. [CrossRef]
13. Rocha, A.; Souza, A.; Johnathan, D.; Aquino, G.; Queiroz, R.; Melo, M. Evaluating thingspeak as an IoT event platform on buildinga smart parking application. In Proceedings of the XI Simpósio Brasileiro de Computação Ubíqua e Pervasiva, Porto Alegre, Brasil, 16–18 July 2019.
14. Sotres, P.; la Torre, C.L.D.; Sanchez, L.; Jeong, S.; Kim, J. Smart city services over a global interoperable internet-of-things system: The smart parking case. In Proceedings of the 2018 Global Internet of Things Summit (GIoTS), Bilbao, Spain, 4–7 June 2018.
15. Khanna, A.; Anand, R. IoT based smart parking system. In Proceedings of the 2016 International Conference on Internet of Things and Applications, IOTA, Pune, India, 22–24 January 2016; pp. 266–270.
16. A'Ssri, S.A.; Zaman, F.H.K.; Mubdi, S. The efficient parking bay allocation and management system using LoRaWAN. In Proceedings of the 2017 IEEE 8th Control and System Graduate Research Colloquium, ICSGRC, Shah Alam, Malaysia, 4–5 August 2017; pp. 127–131.
17. Swamy, N.; Satyanarayana, S.; Babu S, L.; Sharma, T. Design and implementation of a smart parking system using IoT technology. *Int. J. Eng. Res. Comput. Sci. Eng.* **2018**, *6*.
18. Cai, W.; Zhang, D.; Pan, Y. Implementation of smart parking guidance system based on parking lots sensors networks. In Proceedings of the 2015 IEEE 16th International Conference on Communication Technology (ICCT), Hangzhou, China, 18–20 October 2015; pp. 419–424.
19. Suryady, Z.; Sinniah, G.R.; Haseeb, S.; Siddique, M.T.; Ezani, M.F.M. Rapid development of smart parking system with cloud-based platforms. In Proceedings of the 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M), Kuching, Malaysia, 17–18 November 2014.
20. Alam, M.; Moroni, D.; Pieri, G.; Tampucci, M.; Gomes, M.; Fonseca, J.; Ferreira, J.C. Real-time smart parking systems integration in distributed ITS for smart cities. *J. Adv. Transp.* **2018**, *2018*, 1–13. [CrossRef]
21. Mainetti, L.; Palano, L.; Patrono, L.; Stefanizzi, M.L.; Vergallo, R. Integration of RFID and WSN technologies in a Smart Parking System. In Proceedings of the 2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 17–19 September 2014; pp. 104–110.
22. Tsiropoulou, E.E.; Baras, J.S.; Papavassiliou, S.; Sinha, S. RFID-based smart parking management system. *Cyber Phys. Syst.* **2017**, *3*, 22–41. [CrossRef]
23. Hamed, M.M.; Latiff, L.A.; Kamil, I.A.; Dziyauddin, R.A.; Kaidi, H.M. Design and implementation of sensing infrastructure for an indoor smart parking system. In Proceedings of the 2018 2nd International Conference on Telematics and Future Generation Networks (TAFGEN), Kuching, Malaysia, 24–26 July 2018; pp. 31–36.
24. Grodi, R.; Rawat, D.B.; Rios-Gutierrez, F. Smart parking: Parking occupancy monitoring and visualization system for smart cities. In Proceedings of the SoutheastCon 2016, Norfolk, VA, USA, 30 March–3 April 2016.

25. Baroffio, L.; Bondi, L.; Cesana, M.; Redondi, A.E.; Tagliasacchi, M. A visual sensor network for parking lot occupancy detection in Smart Cities. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 745–750.
26. LoRa Alliance. *LoRaWAN^{TM} 1.0.2 Regional Parameters*; LoRa Alliance: Fremont, CA, USA, 2017.
27. Eclipse Mosquitto. Available online: https://mosquitto.org/ (accessed on 31 December 2019).
28. Download Swagger UI|Swagger. Available online: https://swagger.io/tools/swagger-ui/download/ (accessed on 31 December 2019).