



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

– **TELECOM** ESCUELA
TÉCNICA **VLC** SUPERIOR
DE INGENIERÍA DE
TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Telecommunications Engineering

Design, Development and Testing of a Virtualized
Database for Monitoring 5G-connected AGVs in an Industry
4.0 environment

Master's Thesis

Master's Degree in Telecommunication Engineering

AUTHOR: Fernández Sierra, Andrea

Tutor: Gómez Barquero, David

External cotutor: LORENZO HERNANDEZ, MANUEL

ACADEMIC YEAR: 2021/2022

Abstract

The main goal of this thesis is to design, develop and test a virtualized database in a 5G-enabled Industry 4.0 environment. We will develop a InfluxDB database with Docker in which the information of 5G-connected AGVs and modems of the deployment will be stored on a Docker container. It will allow monitoring the generated data and the Key Performance Indicators (KPIs) from the Industry 4.0 real-life use cases with Grafana, including core 5G KPIs such as latency, and user data rates for both downlink and uplink communications involved. The outcome of this thesis will include an evaluation of the obtained results to improve and optimize the 5G network deployment of the 5G-INDUCE project in the Ford factory of Valencia.

Resumen

El objetivo principal de esta tesis es diseñar, desarrollar y testear una base de datos virtualizada en un entorno de Industria 4.0 con 5G. Se desarrollará una base de datos InfluxDB con Docker en la que se almacenará la información de los AGVs conectados a 5G y módems del despliegue en un contenedor Docker. Esto permitirá monitorizar los datos generados y los indicadores claves de rendimiento (KPIs) de casos de uso reales de Industria 4.0 con Grafana, incluidos los KPIs del núcleo 5G, como la latencia y las tasas de datos de usuario para las comunicaciones ascendentes y descendentes involucradas. El resultado de esta tesis incluirá una evaluación de los resultados obtenidos para mejorar y optimizar el despliegue de la red 5G del proyecto 5G-INDUCE en la fábrica Ford de Valencia.

Resum

L'objectiu principal d'aquesta tesi és dissenyar, desenvolupar i testar una base de dades virtualitzada en un entorn d'Indústria 4.0 amb 5G. Es desenvoluparà una base de dades InfluxDB amb Docker en la qual s'emmagatzemarà la informació dels AGVs connectats a 5G i mòdems del desplegament en un contenidor Docker. Això permetrà monitorar les dades generades i els indicadors claus de rendiment (KPIs) de casos d'ús reals d'Indústria 4.0 amb Grafana, inclosos els KPIs del nucli 5G, com la latència i les taxes de dades d'usuari per a les comunicacions ascendents i descendents involucrades. El resultat d'aquesta tesi inclourà una avaluació dels resultats obtinguts per a millorar i optimitzar el desplegament de la xarxa 5G del projecte 5G-INDUEIX a la fàbrica Ford de València.

Subjects: 5G, Industry 4.0, database, MQTT, AGV, InfluxDB, Grafana, Docker, KPIs.

Acknowledgment

It has been nine months since I started this project, and today I would like to say thank you to everyone who has been supporting me during this period.

It has been an important period of personal and academic learning. I had never been involved before in a project of this magnitude and its development has helped me to better understand the topic. At personal level, I have been resolute with new situations and I have easily adapted to new challenges.

I would like to thank my tutor David Gómez Barquero for offering me the opportunity to develop this project and support me during the process.

I thank my Ericsson coordinator, Manuel Lorenzo Hernández(Ericsson) for their support from the beginning offering help me and facilitating everything always guiding me in the best way.

Finally, I thank my parents, my sister and my friends for their patience and understanding. For holding me day and night even from the distance. Because you always have words of encouragement that is what I feel more strength and desire to continue.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 7 |
| 1.1 | Motivation | 7 |
| 1.2 | Objectives | 7 |
| 1.3 | Methodology | 8 |
| 2 | State of Art | 9 |
| 2.1 | Docker Compose and Containerization | 9 |
| 2.2 | Time-Series Oriented Database (TSBD) | 9 |
| 2.2.1 | InfluxDB | 10 |
| 2.2.2 | Grafana | 10 |
| 2.3 | Message Queuing Telemetry Transport (MQTT) | 12 |
| 2.3.1 | Performance | 12 |
| 2.3.2 | Message Format | 14 |
| 2.3.3 | Quality of service | 14 |
| 2.3.4 | Security | 14 |
| 2.4 | RabbitMQ | 15 |
| 2.4.1 | When and why to use RabbitMQ | 16 |
| 3 | 5G-INDUCE | 17 |
| 3.1 | Private 5G Network Deployment | 17 |
| 4 | Development and Results | 20 |
| 4.1 | VPN Connection Setup | 20 |
| 4.2 | Connectivity tests | 24 |
| 4.2.1 | PowerPing | 24 |
| 4.3 | Container Deployment with Docker Compose | 28 |
| 4.4 | InfluxDB Database | 30 |
| 4.4.1 | InfluxDB Data Stored Manually | 31 |
| 4.4.2 | InfluxDB Data Stored From File | 32 |
| 4.5 | MQTT Integration | 36 |
| 4.5.1 | Python Programming Code | 39 |
| 4.5.2 | MQTT Packet Exchange | 46 |
| 4.5.3 | RabbitMQ Managing Interface | 48 |
| 4.6 | Grafana | 56 |
| 4.7 | 5G Coverage Measurements | 60 |
| 4.7.1 | Used Equipment | 62 |
| 4.7.2 | How are measurements made? | 65 |
| 4.7.3 | Measured parameters | 66 |
| 4.7.4 | Results | 67 |
| 5 | Conclusions and Lessons Learned | 72 |
| 6 | Proposals for Future work | 73 |

7 Publications

76

List of Figures

| | | |
|----|---|----|
| 1 | Available charts in Grafana. | 11 |
| 2 | MQTT protocol schematic. | 12 |
| 3 | MQTT connection establishment. | 13 |
| 4 | Subscription to a MQTT topic. | 13 |
| 5 | Publication on a MQTT topic. | 13 |
| 6 | MQTT message format. | 14 |
| 7 | Logo RabbitMQ. | 15 |
| 8 | Application architecture with RabbitMQ. | 16 |
| 9 | Spanish consortium. | 17 |
| 10 | AGV itinerary in Ford Factory. | 18 |
| 11 | Current infrastructure diagram. | 19 |
| 12 | Location of Ericsson antennas. | 19 |
| 13 | Connection OpenVPN tab. | 20 |
| 14 | Successful connection pop-up window. | 21 |
| 15 | Local IP address at UPV. | 21 |
| 16 | Local IP address at 5TONIC. | 21 |
| 17 | VPN connection scheme. | 22 |
| 18 | MobaXterm homepage. | 22 |
| 19 | SSH configuration tab. | 23 |
| 20 | SSH connection established. | 23 |
| 21 | <i>"database_induce"</i> directory. | 23 |
| 22 | Sample 1 of M1. | 25 |
| 23 | Default ping to 10.3.3.30. | 25 |
| 24 | Sample 1 of M2. | 26 |
| 25 | Ping to 10.3.3.30 - 1020 bytes. | 26 |
| 26 | Sample 1 of M3. | 27 |
| 27 | 500 Pings to 10.3.3.30. | 27 |
| 28 | Sample 1 of M4. | 28 |
| 29 | 500 Pings to 10.3.3.30 - 1020 bytes | 28 |
| 30 | <i>docker-compose.yml</i> | 29 |
| 31 | Default ports availability. | 29 |
| 32 | Containers up to date. | 30 |
| 33 | Up containers ready to be used. | 30 |
| 34 | Connected to InfluxDB. | 30 |
| 35 | Statement to create a <i>"induce"</i> database. | 31 |
| 36 | Statement to show the existing databases. | 31 |
| 37 | Statement to setting the use of a specific database. | 31 |
| 38 | Statement to insert a single data point. | 31 |
| 39 | Data inserted in <i>"modem"</i> measurement in <i>"induce"</i> database | 32 |
| 40 | Statement to show all measurements. | 32 |
| 41 | Statements for querying databases. | 32 |
| 42 | <i>"modem_data"</i> file structure. | 33 |

| | | |
|----|--|----|
| 43 | <i>“agv_data”</i> file structure. | 33 |
| 44 | Load <i>“.csv”</i> programming code. | 34 |
| 45 | Libraries imported. | 35 |
| 46 | Making a InfluxDB connection. | 35 |
| 47 | Data parsing. | 36 |
| 48 | Write data into InfluxDB. | 36 |
| 49 | Measurements of <i>“induce”</i> database. | 36 |
| 50 | <i>“AgvInformation”</i> content. | 37 |
| 51 | <i>“ModemInformation”</i> content. | 37 |
| 52 | <i>“RabbitMQ”</i> container. | 38 |
| 53 | Dockerfile of RabbitMQ container. | 38 |
| 54 | MQTT integration operation scheme. | 38 |
| 55 | Publisher programming code. | 39 |
| 56 | Libraries imported. | 40 |
| 57 | MQTT variables declaration. | 40 |
| 58 | Main function. | 40 |
| 59 | Reading data from file. | 41 |
| 60 | Sending data to RabbitMQ broker. | 41 |
| 61 | Subscriber programming code. | 42 |
| 62 | Libraries imported. | 43 |
| 63 | Influx variables declaration. | 43 |
| 64 | Main function. | 43 |
| 65 | Existing databases. | 44 |
| 66 | RabbitMQ subscription. | 44 |
| 67 | Login in RabbitMQ. | 44 |
| 68 | <i>“on_connect”</i> function. | 45 |
| 69 | <i>“on_message”</i> function. | 45 |
| 70 | MQTT live subscriber connection. | 46 |
| 71 | Linux remote directory status. | 46 |
| 72 | MobaXterm session. | 47 |
| 73 | Subscriber connected. | 47 |
| 74 | Data already sent by the producer. | 47 |
| 75 | Blue alarm signal by the producer. | 48 |
| 76 | Data sent and printed. | 48 |
| 77 | <i>“induce”</i> database has been created. | 48 |
| 78 | Stored data from <i>“ModemInformation”</i> | 49 |
| 79 | RabbitMQ interface. | 50 |
| 80 | RabbitMQ views. | 50 |
| 81 | RabbitMQ state. | 50 |
| 82 | Subscriber connected. | 51 |
| 83 | Subscriber connected. | 51 |
| 84 | Generated connection with one subscriber. | 51 |
| 85 | Generated queue with one subscriber. | 51 |
| 86 | RabbitMQ state with one subscriber. | 51 |

| | | |
|-----|---|----|
| 87 | Message data rate with one message incoming. | 52 |
| 88 | In-use binding in the 5G-INDUCE project. | 52 |
| 89 | Incoming messages to MQTT broker. | 53 |
| 90 | Zoom into incoming messages to MQTT broker. | 53 |
| 91 | Data rate of incoming messages to MQTT broker. | 54 |
| 92 | Connected subscriber. | 54 |
| 93 | RabbitMQ state with two subscribers. | 54 |
| 94 | Generated connections with two subscribers. | 55 |
| 95 | Generated channels with two subscribers. | 55 |
| 96 | Generated queues with two subscribers. | 55 |
| 97 | Blue alarm signal in both subscribers | 56 |
| 98 | Messages received for both subscribers. | 56 |
| 99 | Incoming messages to MQTT broker. | 56 |
| 100 | Grafana access interface. | 57 |
| 101 | Homepage of Grafana. | 57 |
| 102 | Configuration menu. | 58 |
| 103 | Data sources tab. | 58 |
| 104 | InfluxDB data source configuration. | 58 |
| 105 | Database access configuration. | 59 |
| 106 | New dashboard. | 59 |
| 107 | Measurement. | 60 |
| 108 | Field(value). | 60 |
| 109 | 5G-INDUCE Dashboard. | 61 |
| 110 | 5G Rack inside a protection cupboard at Ford factory. | 61 |
| 111 | 5G DOTS installed at Ford factory. | 62 |
| 112 | 5G Rohde & Schwarz scanner. | 62 |
| 113 | Coverage measurement equipment. | 63 |
| 114 | Computer for making measurements at Ford Factory. | 63 |
| 115 | SmartOne license and RJ45 adapter. | 64 |
| 116 | Configuration tab. | 64 |
| 117 | <i>Trama Jumbo</i> with value 9014 bytes. | 64 |
| 118 | SmartOne Homepage. | 65 |
| 119 | <i>“Way Points”</i> | 66 |
| 120 | 5G NR parameters. | 67 |
| 121 | Connectivity level according to signal strength. | 67 |
| 122 | RSRP level measured. | 68 |
| 123 | RSRQ level measured. | 69 |
| 124 | SINR level measured. | 70 |
| 125 | PCI numbers received. | 70 |
| 126 | SmartOne 5G NR scanner interface. | 71 |

List of Tables

| | | |
|---|---|----|
| 1 | Average latency of each sample of M1. | 24 |
| 2 | Average latency of each sample of M2. | 26 |
| 3 | Average latency of each sample of M3. | 27 |
| 4 | Average latency of each sample of M4. | 27 |
| 5 | Average of performed tests. | 28 |

1 Introduction

This section explains the motivation behind this project, the primary objectives aimed to be achieved and the methodology followed for its development.

1.1 Motivation

The main motivation of this project is to improve the existing solutions under the 5G-INDUCE project through the automation as much as possible of the storage process of the data generated.

The transition to Industry 4.0 is driven by the successful adoption of multiple new technologies. To accelerate smart manufacturing, digital twins of machines and operations will be a necessity, as will factory automation and real-time control and equipment and tasks monitoring. Industry 4.0 will increase the intelligence of machines. It will lead to more efficient factories with less wasteful processes, more flexible production lines and higher productivity [1]

Built on the foundation of smart, secure, wireless connectivity, there are opportunities to extend machine life through predictive maintenance, support rapid material handling, monitor every detail of the shop floor, and leverage collaborative robots simultaneously with mobile communication. This will help factories realize their goal of becoming a fully automated factory.

Industry 4.0 needs the reliability and responsiveness of wired connections and the flexibility and cost-effectiveness of wireless connections. Because of 5G, the best of both worlds can potentially be had. Published in July 2020, 3GPP Release 16 introduced new capabilities to 5G, such as enhanced ultra-reliable low latency communication (eURLLC) with millisecond latencies, 99.9999% reliability, and TSN support [2].

5G commercial network rollout is already happening in Europe and vertical industry companies from different sectors have a need to test their 5G-based applications in a 5G-enabled infrastructure flexible enough to reproduce different live network operation conditions, in order to ensure successful commercialization.

1.2 Objectives

The objectives of this project are:

- To contribute the development of the 5G-INDUCE project.
- To demonstrate how 5G (5th generation) private networks could transform manufacturing.
- To acquire knowledge about InfluxDB, Docker containers and the Python programming language which the database is intended to be developed.
- To design, develop and test a virtualized database in a 5G-enabled Industry 4.0 environment for the three proposed use cases (UCs) of the project: autonomous

indoor fleet management, AGV (Autonomous guided vehicle) operation based on human gesture recognition and VR (Virtual Reality) immersion for AGV control.

- Implementation of the MQTT (Message Queuing Telemetry Transport) communication protocol for the exchange of data files generated by the aforementioned use cases, as well as the deployment of virtualized RabbitMQ (MQTT broker).
- The creation of a Grafana dashboard which monitors in real time the data of the project use cases stored in the database: data from AGVs, modems, etc.
- To carry out coverage measurements on the 5G network deployment (suggested in [3]) in Ford factory.

1.3 Methodology

The development of this project started in January 2022 until September. To accomplish this, the starting point is the development of a database to store the generated data in the use cases of the project. From this idea, a set of phases are established that enhance the main idea and even additional phases with the purpose of solving possible open issues of the 5G-INDUCE project in which it is developed.

For its development, the 5G-INDUCE project documentation has been consulted, such as deliverables, proposal, etc. In addition, academic scientific articles by authors with recognized prestige and company reports have been consulted on the topics to be discussed. Afterwards, this information has been interpreted and reported in this document. Throughout the development of this project fortnightly meetings have been held with the European consortium and weekly meetings with the Spanish consortium. Besides, It should be emphasized that this project is developed as a part of a collaboration grant between the UPV and Ericsson, with which I hold daily scrum meetings.

2 State of Art

This section will detail the technical aspects that will be implemented in this project, explaining its particularities from a theoretical point of view.

It should be noted that this project is preceded by another project previously carried out by me, Andrea Fernández Sierra [3]. In [3] the potential 5G architectures in a real-life Industry 4.0 case in the Ford factory in Valencia was analysed.

As a result, after evaluating the obtained results and understanding the main differences by each mentioned architecture a 5G architecture was proposed for this Industry 4.0 environment. It was also developed under the European 5G-INDUCE project.

This project will focus on the development of a virtualized database in a 5G-enabled Industry 4.0 environment. We will develop a InfluxDB database with Docker in which the information of 5G-connected AGVs or modems of the deployment will be stored on a Docker container. The outcome of this thesis will include an evaluation of the obtained results to improve and optimize the 5G network deployment of the 5G-INDUCE project in the Ford factory of Valencia. It is carried out under 5G-INDUCE project and could be considered a continuation of [3].

This requires understanding and studying the following technical aspects:

2.1 Docker Compose and Containerization

Docker-compose is a tool for running multi-container applications on Docker defined using the compose file format. A compose file (YAML format) will be used to define how one or more container that make up the application already mentioned is configured. In the YAML file it could be define, the image it wants to be used, the port exposure, the volumes. Once you have the compose file, you can create and start the application. The advantages of using containers are significant. Many companies have embraced the use of containers, due to their scalability, their flexibility, and their adaptability to any environment. Containers are considered as the standard software unit, which packages code and all its dependencies, so that the application can run reliably in different environments[4]

In addition, with containers, the different services within the same application are decoupled. In the event of a service failure, the whole system does not have to go down. The damage to the system will be minimal and, in order to solve the failure, it is perfectly localized.

The integration of containers and container orchestrators, as well as system monitoring, allows the deployment of services in a manufacturing environment to be done reliably and robustly.

2.2 Time-Series Oriented Database (TSBD)

The particularity of the data to be stored in this project is that it has a time component which makes it necessary to have a time series database that is optimized for time series data.

Although other alternatives exist, the reason for choosing InfluxDB is because this database has already been used in previous Ericssons projects and the results have been satisfactory. The same approach is used by Grafana for the visualization of this data.

2.2.1 InfluxDB

InfluxDB is a time series database management system developed by the company Influx-Data, Inc. InfluxDB is an open source software, and it is free. This product launched a new cloud version available totally customizable with a web-based user interface to collect and visualize the data[5].

Some of the advantages of TSBD are the following:

- **Free access database:** No proprietary server infrastructure is required; the free version is designed to introduce you to InfluxDB.
- **High performance for time series data with high consumption and real time queries:** Key factor due to the fact that later the amount of data in the project will be increased and it is necessary to optimize all the factors to the maximum.
- **InfluxQL for interacting with data using SQL as a query language:** It allows TSBD use with a certain level of simplicity thanks to the fact that data can be processed using SQL commands.
- **Retention policies for obsolete data:** This allows to keep the database clean. A data to be stored have a different timestamp that the existing one in the database, otherwise, the database just keeps one.
- **Data exportation:** InfluxDB allows you to export data at any time with no need to insert the data again.
- **Information abundance:** There is a lot of information about this system to be able to perform all possible actions and find solutions to possible problems.

InfluxDB offers the possibility of integration with other systems such as Grafana to be able to represent the data in a simple way without having to create intermediate infrastructures. Having the administrator credentials would be enough to be able to use Grafana, more details will be explained later[6].

2.2.2 Grafana

Grafana is a tool for visualizing time series data. From the collected data a graphical overview (dashboards) of the generated data and the Key Performance Indicators (KPIs) from the Industry 4.0 real-life use cases will be obtained. It would include core 5G KPIS such as latency, and user data rates for both downlink and uplink communications involved, etc [7].

Some of the advantages after the selection of this software are:

- It has flexible quick graphics **with numerous options**, as well as dynamic and reusable dashboards.
- **Highly extensible:** It is possible to use many dashboards and plug-ins available in the official library.
- **Enabled collaboration:** It allows data and dashboards to be exchanged between teams.

As a result, the user is provided a very useful application in order to efficiently manage his data and services, allowing him to have a global knowledge of his situation.

In addition, Grafana allows the representation of different dashboards, Figure 1, among them:

- **Graph:** It shows data over time so that it can be viewed as it evolves.
- **Table:** It displays the data table, but without any visual chart.
- **Bar Gauge:** It display a bar chart over time.



Figure 1: Available charts in Grafana.

2.3 Message Queuing Telemetry Transport (MQTT)

MQTT is one of the oldest machine to machine (M2M) communication protocols, developed by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Control Systems Ltd in 1999 [8].

MQTT works on top of a reliable data transfer protocol, typically TCP/IP. This protocol has become one of the most widely used in the field of the Internet of Things (IoT), this is due to the many advantages it offers, including the small amount of bandwidth it consumes and the easy implementation of this, or what is the same the reduced amount of resources required for its correct operation, which makes it compatible with a large number of devices on the market.

MQTT implements a scheme based on the PUB/SUB model, as shown in Figure 2, in which three components or participants are involved, namely the Publisher, the Subscriber and the Broker, each of them in charge of carrying out a specific function.

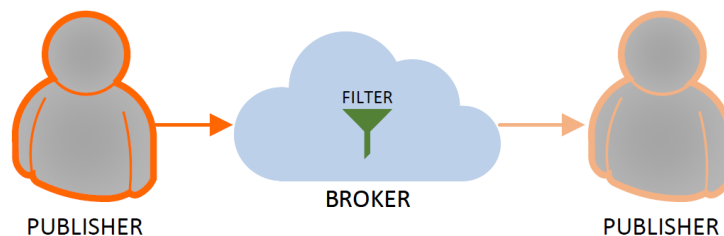


Figure 2: MQTT protocol schematic.

The functions of each element are the following:

- **Publisher:** It is the responsible for sending messages to a topic located in the broker.
- **Subscriber:** It receives all the messages sent to the topic to which he/she has previously subscribed.
- **Broker:** It is in charge of managing the messages received by the publisher and forwarding them to the specified subscribers. In order to do so, it sorts the messages received into topics and forwards them to the subscribers who have subscribed to the topics mentioned above.

2.3.1 Performance

First of all, a TCP/IP connection must be established between the client and the broker, this connection will remain active until the client terminates it. The connection will be made when the client sends a `CONNECT` message, which will contain the necessary information to make the connection. Once the client has received a `CONNACK` message

from the broker confirming the connection, the connection is ready to work, Figure 3. The ports used in the connection are port 1883 for a normal connection and port 8883 if the connection is made over TLS [10].

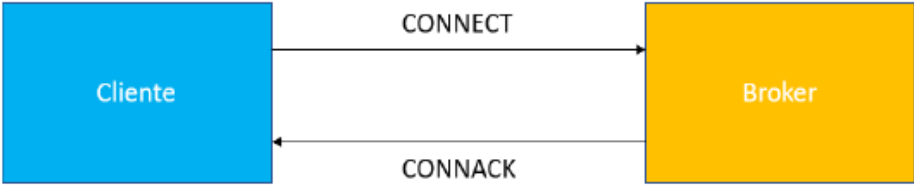


Figure 3: MQTT connection establishment.

The client can acquire the role of publisher, subscriber, or both at the same time, depending on the client’s needs. On the other hand, the customer can subscribe or unsubscribe from a topic, using the SUBSCRIBE/UNSUBSCRIBE messages respectively, which will be sent by the customer, Figure 4. Both the SUBSCRIBE message and the UNSUBSCRIBE message will contain the name of the topic on which the action is to be performed. Once the confirmation message SUBACK/UNSUBACK has been received, the action will have been carried out successfully.

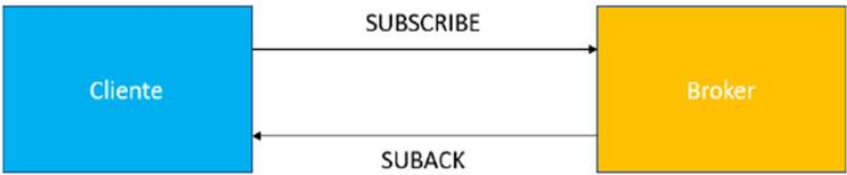


Figure 4: Subscription to a MQTT topic.

Finally, the publication of messages by the client in a topic of the broker, will be done through a PUBLISH message, which will contain the name of the topic, in which the client wishes to publish, and the message to be published, Figure 5.



Figure 5: Publication on a MQTT topic.

2.3.2 Message Format

Messages sent in the MQTT protocol are composed of three parts, two of which are optional, Figure 6[10]:

- **Header:** its size is between 2 and 5 bytes and is mandatory for all messages. The first byte contains the control information, the 8 bits of which are divided into two groups of four bits, the first group contains QoS, duplicity and retention information, while the second group indicates the type of message. The remaining bytes indicate the size of the message.
- **Optional header:** this is not mandatory, and it will depend on the type of message sent. Its function is to hold additional information about the message.
- **Payload:** It contains the actual message of the protocol, for example, if it were a SUBSCRIBE message, the data would be the topic and the message to be published.

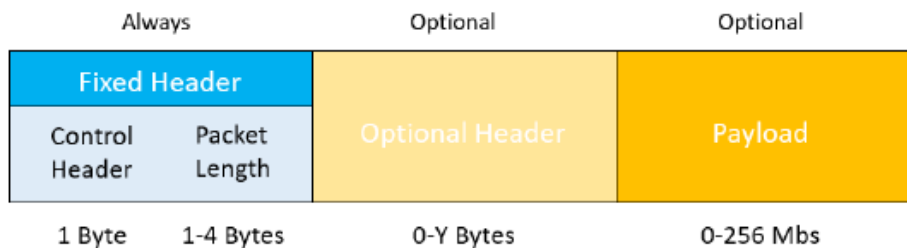


Figure 6: MQTT message format.

2.3.3 Quality of service

MQTT incorporates mechanisms to guarantee QoS quality of service, for which it has three levels[9]:

- **QoS 0:** the message is sent without guarantee of reception.
- **QoS 1:** the message is sent until a message confirming its reception is received. Because of the time that may elapse between sending the message and receiving confirmation, the receiver may receive duplicate messages.
- **QoS 2:** the message is delivered only once to the receiver, and it is verified that it has been received.

2.3.4 Security

Regarding security, it is worth noting that MQTT supports multiple security mechanisms such as user and password authentication, as well as encrypted communication via SSL/TLS. The use of the latter requires the use of port 8883, by means of which the client validates the certificate that belongs to the server. the client-server communication is established.

2.4 RabbitMQ

The RabbitMQ messaging server is an open source implementation of the AMQP 0-9-1 protocol (Advanced Message Queuing Protocol) that defines how messages should be queued, routed, and delivered in a reliable and secure manner [11], Figure 7.



Figure 7: Logo RabbitMQ.

RabbitMQ is also known as a message broker or queue manager. It is software used to define queues, connect applications, and accept messages. Message queues enable asynchronous communication, allowing other applications (endpoints) that are producing and consuming messages to interact with the queue instead of communicating directly with each other.

A message can include any type of information. For example, a message could contain information about a process or job that should start on another application, possibly even on another server, or it might be a simple text message.

The message broker stores the messages until a receiving application connects and consumes a message from the queue. The receiving application then processes the message appropriately. A message producer adds messages to a queue without having to wait for them to be processed.

It will introduce some terminologies from the RabbitMQ world that we will use frequently[11]:

- **Exchanges:** RabbitMQ server endpoints (each one identified by a unique key), that way the client will be connected and sending messages.
- **Queues:** RabbitMQ server components, buffer messages coming from one or more exchanges and send them to the corresponding message receivers. Messages in a queue can also be downloaded into persistent storage (durable queues) which provides a higher degree of reliability in case of a messaging server failure; once the server is back up and running, messages in persistent storage are placed back into the appropriate queues (identified with a unique key) for transfer to recipients. Each queue is identified with a unique key.
- **Bindings:** Link between exchanges and queues. Each binding is a rule that specifies how the exchanges should route messages to queues. A binding may have a routing key that can be used by clients in order to specify the routing semantics of a message.
- **Virtual hosts:** Logical units that divide RabbitMQ server components (such as exchanges, queues, and users) into separate groups for better administration and access control. Each AMQP client connection is bound to a concrete virtual host.

2.4.1 When and why to use RabbitMQ

Message queues allow web servers to respond to requests in their own time rather than being forced to perform resource-intensive procedures immediately. Message queues is also useful for distributing a message to multiple recipients for consumption or balancing the load between workers.

The consumer application removes a message from the queue and processes the PDF while the producer pushes new messages to the queue. The consumer can be on the same or an entirely different server than the publisher, it makes no difference. Requests can be created in one programming language and handled in another programming language, as the two applications only communicate through the messages, they are sending to each other. The two services have what is known as low coupling between the sender and the receiver, Figure 8 [12].

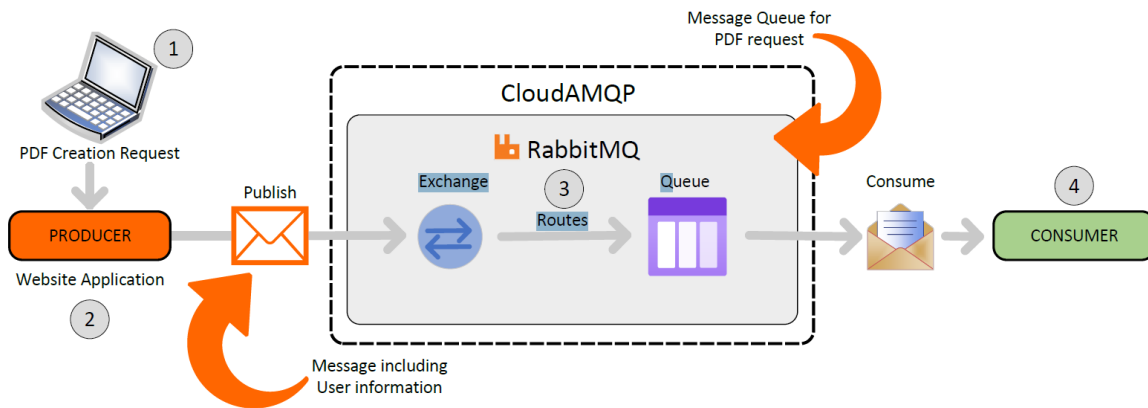


Figure 8: Application architecture with RabbitMQ.

3 5G-INDUCE

The 5G-INDUCE project develops and open and cooperative 5G network platforms that will allow the showcasing and evaluation of advanced network applications, supporting emerging and innovative services related to the industry 4.0 [TFM anterior].

The goal is to provide an end-to-end orchestration platform over enabling experimentation infrastructures for advanced 5G network applications that can be applied for the realisation of extensive 5G UCs in the broader Industry 4.0 sector, leading to technological and business validation of 5G technologies by multiple collaborating tenants (e.g. manufacturing, logistics, maintenance power management, security/surveillance and more).

Focus is given on validation of the 5G-readiness of both telecom operators and applications providers. The 5G-INDUCE experimentation facilities (ExFas) are deployed with the goal to validate and showcase over a real industrial 5G environment the developed NetApps.

Three ExFas are envisioned Spain, Italy, and Greece, all linked with large industrial facilities (Ford, Whirlpool and PPC respectively) while being supported by advance 5G infrastructures, this document will focus on just in the Spain experimentation facility in Ford. The overall purpose of the adopted ExFa sites is to address actual Industry 4.0 needs in a diverse set of industrial environments, showcasing the beneficial use of 5G technology in terms of latency, optimized interoperability and management, security, and safety.

Following the evolution of AGVs use in Ford three UCs will be tested to achieve a higher level of automation and increase human-machine iteration:

- **UC1:** Autonomous AGV fleet management.
- **UC2:** Smart operation of AGVs based on human gesture recognition.
- **UC3:** VR immersion of AGVs control.

Figure 9 shows the Spanish consortium of the project.



Figure 9: Spanish consortium.

3.1 Private 5G Network Deployment

The Spanish 5G Experimentation Facility infrastructure is deployed with the goal to validate and showcase the developed network applications over a real industrial 5G environment. The main experimentation area is a large industrial facility, where Ford operates,

located in Almussafes (Valencia, Spain). The experimentation facility includes three use cases mentioned above. Moreover, the experimentation facility defines two areas, one for indoor transportation in Ford warehouses, coordinated with another for outdoor transportation in which the indoor AGV is replaced by the outdoor AGV. Figure 10 shows the AGV path in Ford premises.



Figure 10: AGV itinerary in Ford Factory.

The deployed infrastructure must guarantee 5G coverage on this area, as well as the computing resources required to run the use cases applications. Regarding the current implementation of such infrastructure, the solution includes a distributed 5G network between 5TONIC lab in Madrid (Spain) and the already mentioned Ford factory in Almussafes, and also is connected with the European part for the applications orchestration of the project. Figure 11 shows the current infrastructure distributed along Ford and 5TONIC.

The 5G-INDUCE NetApps is deployed on dedicated compute server located at Ford facilities. The orchestration of the NetApps is managed from the central 5G-INDUCE Orchestrator at CNIT premises (Italy), connected to 5TONIC lab, which is in turn interconnected to Ford premises.

The selected architecture option is 5G SA, which is deployed in a distributed way. The RAN and the user plane function (UPF) are installed at Ford premises, while the rest of the 5G SA Core (control plane) is installed at 5TONIC. In this way, there is 5G coverage at Ford premises providing connectivity to the user end devices. The 5G RAN equipment splits the user plane traffic and the control traffic. The use cases traffic is directed towards the vertical applications executing also locally at Ford premises, so that it remains geographically close to the end user devices, keeping the latency very low. Complementarily, the control and management planes of the 5G equipment are done from a remote location (5TONIC lab in Madrid, 350 km away from Ford factory), as it does not require as low latency and high-speed levels as those of the local user plane. Two areas of coverage are provided, one provided by all the 5G radio DOTs (indoor coverage)

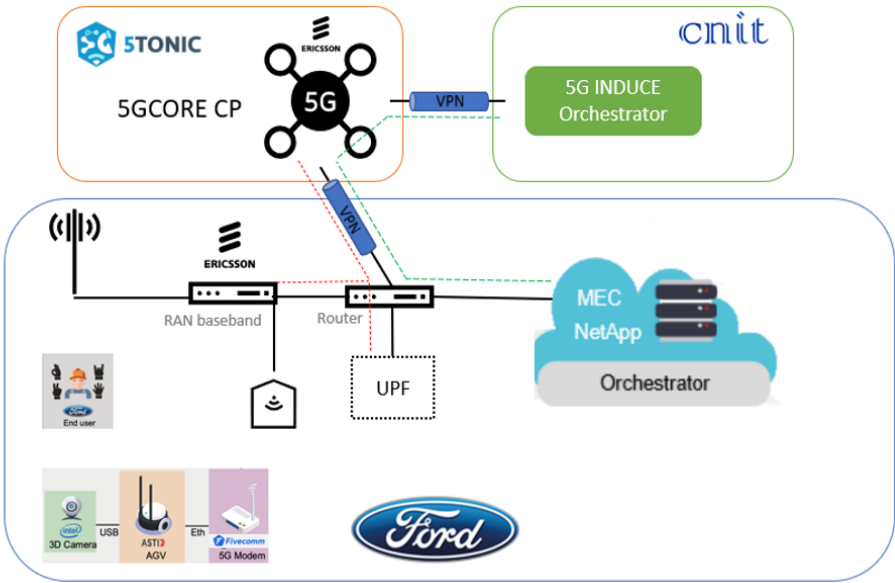


Figure 11: Current infrastructure diagram.

inside the factory, and one provided by the outdoor antenna, both operating on 3.5 GHz (50 MHz bandwidth), Figure 12.

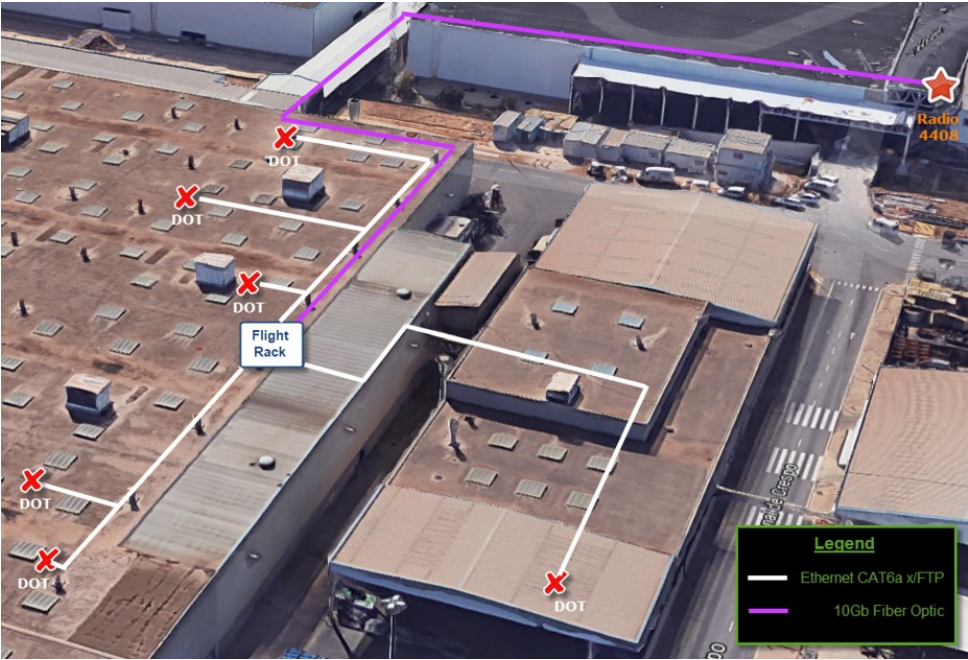


Figure 12: Location of Ericsson antennas.

4 Development and Results

This section shows the results obtained as a part of this project. First, it is necessary to create the connection between both sites Valencia and Madrid. For this purpose, a VPN will be created between the 5TONIC laboratory in Madrid and the UPV in Valencia. Moreover, connectivity tests will be carried out between both sites. Through this VPN the InfluxDB will be deployed in 5TONIC. To facilitate the database integration with the European part of the project, an MQTT RabbitMQ Broker will be developed, implemented and monitoring with Grafana. In parallel, the network deployment at Ford factory will be completed and coverage measurements of the area in which the use cases are implemented will be carried out.

4.1 VPN Connection Setup

This section explain in detail how has been setup the VPN connection between UPV and 5TONIC. The connectivity tool used has been OpenVPN. It offers point to point connectivity between users and host connected remotely. In addition, it should be known that OpenVPN is an open source connection protocol used to facilitate a secure tunnel between two points in a network [13].

OpenVPN requires configuration files that define how a connection is carried out. You should be able to find all the configuration files you need on internet but in this case, Ericsson as a partner of the project provides configuration files. The file extension is *.ovpn*

Then, OpenVPN GUI application is run to import the configuration file. To establish the connection, click on the OVPN files in OpenVPN application. It should enter the login credentials. If everything goes well, you should see a log screen with some status commands, which will disappear when the connection is established, Figure 13.

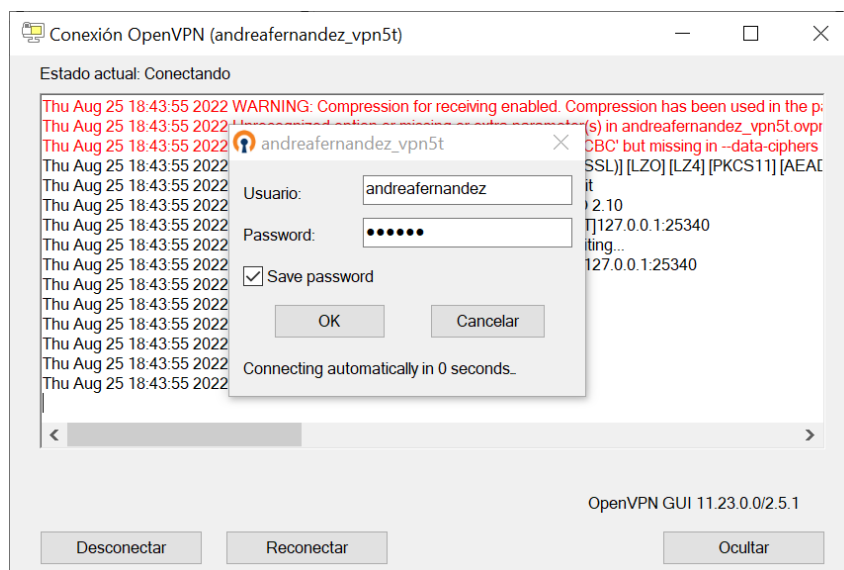


Figure 13: Connection OpenVPN tab.

You should receive a notification on your desktop informing you that the connection was successful, Figure 14.

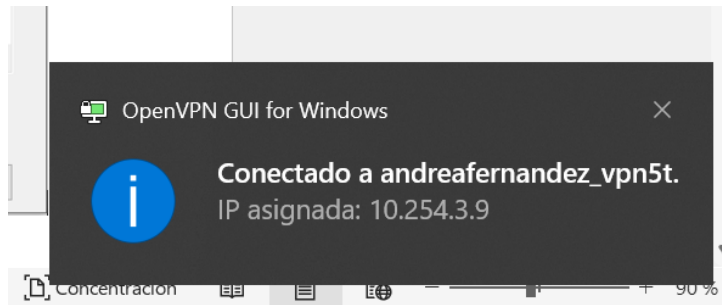


Figure 14: Successful connection pop-up window.

The assigned IP address is the IP address of the UPV inside the 5TONIC network. In fact, if the `ipconfig` command is used in Windows it is obtained that the local IP address is **158.42.160.30** (server from UPV), Figure 15, and the remote IP address of this one is **10.254.3.9** (server from UPV in 5TONIC Madrid), Figure 16. Besides, Figure 17 shows a scheme of the VPN connection created.

```
Adaptador de Ethernet Ethernet 2:

Sufijo DNS específico para la conexión. . . : upv.es
Dirección IPv6 . . . . . : 2001:720:101c:560:1c2d:b471:e6e2:6b39
Dirección IPv6 temporal. . . . . : 2001:720:101c:560:a002:5218:7f1d:5617
Vínculo: dirección IPv6 local. . . : fe80::1c2d:b471:e6e2:6b39%5
Dirección IPv4. . . . . : 158.42.160.30
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : fe80::9624:e1ff:fe5a:feb1%5
                                           158.42.160.1
```

Figure 15: Local IP address at UPV.

```
Adaptador desconocido OpenVPN TAP-Windows6:

Sufijo DNS específico para la conexión. . . :
Vínculo: dirección IPv6 local. . . : fe80::9c43:302f:d324:e251%26
Dirección IPv4. . . . . : 10.254.3.9
Máscara de subred . . . . . : 255.255.255.252
Puerta de enlace predeterminada . . . . . :
```

Figure 16: Local IP address at 5TONIC.

Once connected to access to the server it is necessary to stablish an SSH connection.



Figure 17: VPN connection scheme.

For this purpose, there are many options, in this case it is going to be use MobaXterm. It is an SSH connection manager.

Figure 18 shows the MobaXterm homepage. Click on session and on SSH. Then, the connection should be configure as it is shown in Figure 19. The SSH connection is carried out with the remote server **10.3.3.30, port 22**.

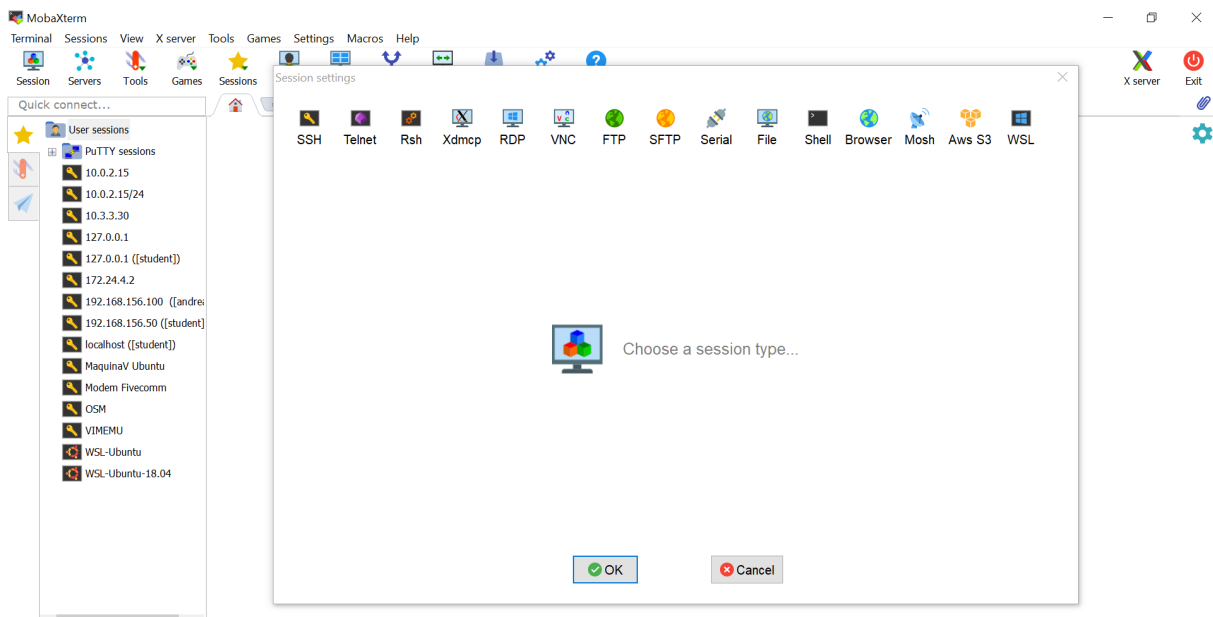


Figure 18: MobaXterm homepage.

If the connection has been successfully established, the log in information (*user:vecp*) will be requested on the screen, 20. In addition, in the left part of Figure 21 it can be seen the directories of the server to which we are connected. As the server is used for other tasks, with the *mkdir* command a directory has been created in which to concentrate all the project files.

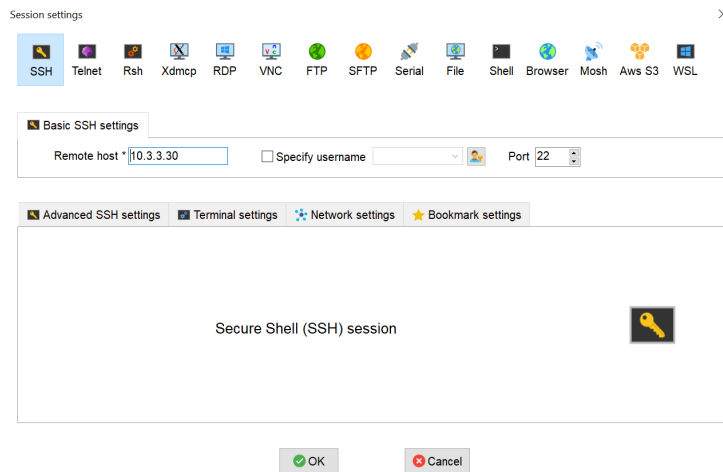


Figure 19: SSH configuration tab.

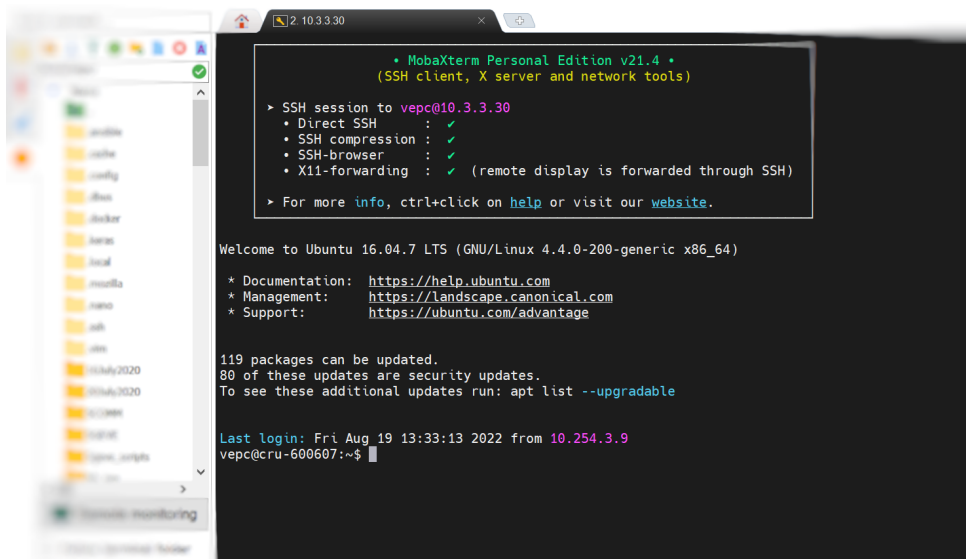


Figure 20: SSH connection established.

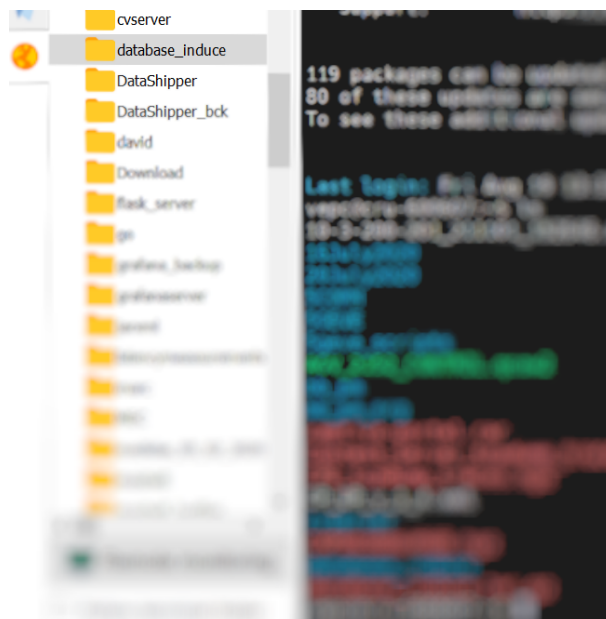


Figure 21: "database_induce" directory.

4.2 Connectivity tests

In this section the connectivity tests will be explained in detail. The aim of carrying out it is to check the performance and quality of services provided by the VPN created. This VPN by the end of this year will be used to send use cases generated data from the UPV to 5TONIC. It is necessary that the uplink bandwidth be as high as possible to send the video from the 360 camera in the UC3.

The quality of a connection can be tested under different metrics i.e. Latency (Time to Respond - RTT) using Ping command, Jitter (Latency variation) can be measured with Iperf running a User Datagram Protocol (UDP) test. Datagram loss can be measured with Iperf running a UDP test too and the bandwidth with Iperf but running a Transmission Control Protocol (TCP) test.

In this project PowerPing is going to be used. One way to stress a VPN connection will be increasing the number of packets, modifying the packet size, modifying the bandwidth and the communication direction (downlink and uplink communications). In this project, we will modify the first two.

4.2.1 PowerPing

PowerPing is an advanced command-line ping protocol. It will be use extending the normal functionality of ping in Windows with additional ICMP features such as graphing, listening, flooding, etc. The ping is a command or diagnostic tool used to know the time it takes for your local connection to communicate with a remote computer on the IP network[14]. This latency is measured in milliseconds [15].

10 samples are carried out for each measurement to obtain average network latency. In all measurements just the first sample is shown, to see all the values obtained see Annex A.

- **Measurement 1 (M1):** default ping to **10.3.3.30**. The definition of default ping in PowerPing is packet size of 35 bytes plus 28 bytes of the header of a ICMP packet, 63 bytes and 5 pings sent. The next Figure 22 shows the first sample of the M1. Figure 23 and Table 1 show the average latency obtained in the ten samples and the average latency of the measurement (**28,18 ms**).

| | S.1 | S.2 | S.3 | S.4 | S.5 | S.6 | S.7 | S.8 | S.9 | S.10 |
|------------------|------|------|------|------|------|------|-----|------|------|------|
| Av. Latency (ms) | 20.3 | 18.5 | 18.3 | 18.9 | 20.3 | 20.1 | 19 | 19.1 | 30.3 | 17 |

Table 1: Average latency of each sample of M1.

- **Measurement 2 (M2):** ping to **10.3.3.30** modifying the packet size to 1020 bytes plus 28 bytes, 1048 bytes (maximum size). Once again, 5 pings sent. To do this, the PowerPing format command is:

PowerPing 10.3.3.30 -c count of ping number s size of the packet in bytes

```

C:\Users\anfer\Documents>PowerPing 10.3.3.30

Pinging 10.3.3.30 with 35 bytes of data [Type=8 Code=0] [TTL=255]:
Reply from: 10.3.3.30:0 seq=1 bytes=63 type=ECHO REPLY time=24,1ms
Reply from: 10.3.3.30:0 seq=2 bytes=63 type=ECHO REPLY time=21,1ms
Reply from: 10.3.3.30:0 seq=3 bytes=63 type=ECHO REPLY time=18,6ms
Reply from: 10.3.3.30:0 seq=4 bytes=63 type=ECHO REPLY time=15,1ms
Reply from: 10.3.3.30:0 seq=5 bytes=63 type=ECHO REPLY time=22,6ms

--- Stats for 10.3.3.30 ---
General: Sent [ 5 ], Recieved [ 5 ], Lost [ 0 ] (0% loss)
Times: Min [ 15,1ms ], Max [ 24,1ms ], Avg [ 20,3ms ]
Types: Good [ 5 ], Errors [ 0 ], Unknown [ 0 ]
Started at: 04/08/2022 15:32:27 (local time)
Runtime: 00:00:04.1

```

Figure 22: Sample 1 of M1.

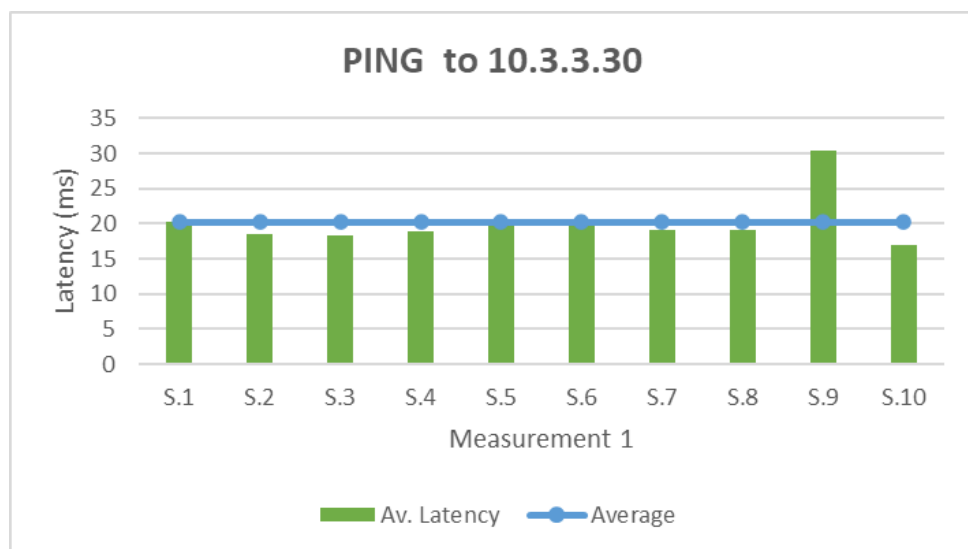


Figure 23: Default ping to 10.3.3.30.

The command to use is:

PowerPing 10.3.3.30 -s 1020

The next Figure 24 shows the first sample of the M2. Figure 25 and Table 2 show the average latency obtained in the ten samples and the average latency of the measurement (**19.16 ms**).

- **Measurement 3 (M3):** ping to **10.3.3.30** modifying the packet size to 1020 bytes plus 28 bytes, 1048 bytes (maximum size). Once again, 5 pings sent. To do this, the PowerPing format command is:

PowerPing 10.3.3.30 -c 500

```

C:\Users\anfer\Documents>PowerPing 10.3.3.30 -s 1020

Pinging 10.3.3.30 with 1020 bytes of data [Type=8 Code=0] [TTL=255]:
Reply from: 10.3.3.30:0 seq=1 bytes=1048 type=ECHO REPLY time=19,2ms
Reply from: 10.3.3.30:0 seq=2 bytes=1048 type=ECHO REPLY time=14,8ms
Reply from: 10.3.3.30:0 seq=3 bytes=1048 type=ECHO REPLY time=21,2ms
Reply from: 10.3.3.30:0 seq=4 bytes=1048 type=ECHO REPLY time=19,8ms
Reply from: 10.3.3.30:0 seq=5 bytes=1048 type=ECHO REPLY time=15,4ms

--- Stats for 10.3.3.30 ---
General: Sent [ 5 ], Recieved [ 5 ], Lost [ 0 ] (0% loss)
Times: Min [ 14,8ms ], Max [ 21,2ms ], Avg [ 18,1ms ]
Types: Good [ 5 ], Errors [ 0 ], Unknown [ 0 ]
Started at: 04/08/2022 16:38:08 (local time)
Runtime: 00:00:04.1

```

Figure 24: Sample 1 of M2.

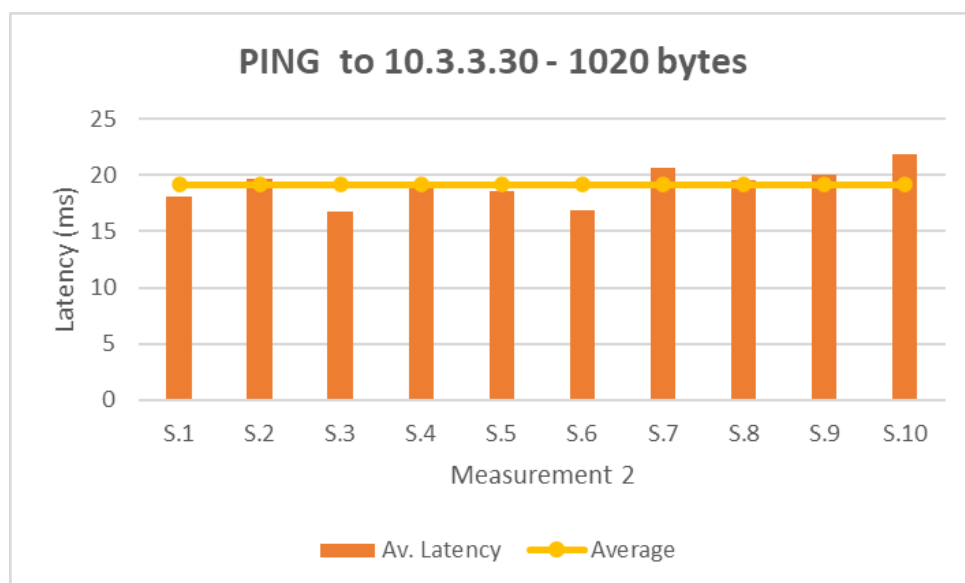


Figure 25: Ping to 10.3.3.30 - 1020 bytes.

| | S.1 | S.2 | S.3 | S.4 | S.5 | S.6 | S.7 | S.8 | S.9 | S.10 |
|------------------|------|------|------|------|------|------|------|------|------|------|
| Av. Latency (ms) | 18.1 | 19.7 | 16.7 | 19.3 | 18.6 | 16.9 | 20.7 | 19.6 | 20.1 | 21.9 |

Table 2: Average latency of each sample of M2.

The next Figure 26 shows the first sample of the M3. Figure 27 and Table 3 show the average latency obtained in the ten samples and the average latency of the measurement (**22.15 ms**).

- **Measurement 4 (M4):** ping to **10.3.3.30** modifying the number of pings to 500 and the packet size to 1020 bytes (maximum size) plus 28 bytes of the header of a ICMP packet, 1048 bytes. To do this, the PowerPing command to use is:

```

Reply from: 10.3.3.30:0 seq=498 bytes=63 type=ECHO REPLY time=25,5ms
Reply from: 10.3.3.30:0 seq=499 bytes=63 type=ECHO REPLY time=21,9ms
Reply from: 10.3.3.30:0 seq=500 bytes=63 type=ECHO REPLY time=24,2ms

--- Stats for 10.3.3.30 ---
General: Sent [ 500 ], Recieved [ 500 ], Lost [ 0 ] (0% loss)
Times: Min [ 14,5ms ], Max [ 130,8ms ], Avg [ 22,5ms ]
Types: Good [ 500 ], Errors [ 0 ], Unknown [ 0 ]
Started at: 06/08/2022 18:15:24 (local time)
Runtime: 00:08:35.5

```

Figure 26: Sample 1 of M3.

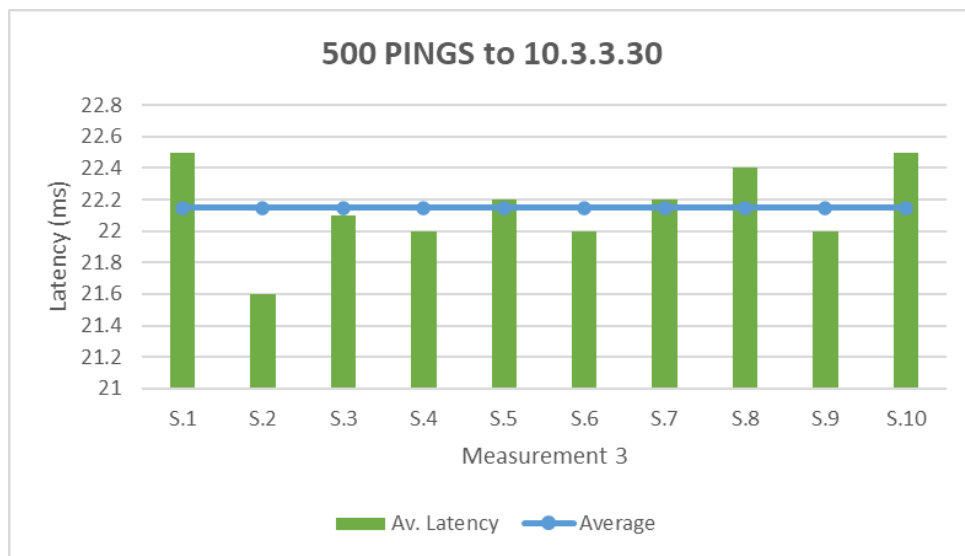


Figure 27: 500 Pings to 10.3.3.30.

| | S.1 | S.2 | S.3 | S.4 | S.5 | S.6 | S.7 | S.8 | S.9 | S.10 |
|------------------|------|------|------|-----|------|-----|------|------|-----|------|
| Av. Latency (ms) | 22.5 | 21.6 | 22.1 | 22 | 22.2 | 22 | 22.2 | 22.4 | 22 | 22.5 |

Table 3: Average latency of each sample of M3.

PowerPing 10.3.3.30 -c 500 -s 1020

The next Figure 28 shows the first sample of the M4. Figure 29 and Table 4 show the average latency obtained in the ten samples and the average latency of the measurement (**22.6 ms**).

| | S.1 | S.2 | S.3 | S.4 | S.5 | S.6 | S.7 | S.8 | S.9 | S.10 |
|------------------|------|------|------|------|-----|------|------|------|------|------|
| Av. Latency (ms) | 23.8 | 22.7 | 22.7 | 22.3 | 22 | 22.6 | 21.9 | 22.9 | 22.6 | 23.1 |

Table 4: Average latency of each sample of M4.

```

Reply from: 10.3.3.30:0 seq=496 bytes=1048 type=ECHO REPLY time=19,0ms
Reply from: 10.3.3.30:0 seq=497 bytes=1048 type=ECHO REPLY time=23,4ms
Reply from: 10.3.3.30:0 seq=498 bytes=1048 type=ECHO REPLY time=25,3ms
Reply from: 10.3.3.30:0 seq=499 bytes=1048 type=ECHO REPLY time=24,5ms
Reply from: 10.3.3.30:0 seq=500 bytes=1048 type=ECHO REPLY time=26,8ms

--- Stats for 10.3.3.30 ---
General: Sent [ 500 ], Recieved [ 500 ], Lost [ 0 ] (0% loss)
Times: Min [ 15,3ms ], Max [ 105,5ms ], Avg [ 23,8ms ]
Types: Good [ 500 ], Errors [ 0 ], Unknown [ 0 ]
Started at: 08/08/2022 11:40:30 (local time)
Runtime: 00:08:36.8

```

Figure 28: Sample 1 of M4.

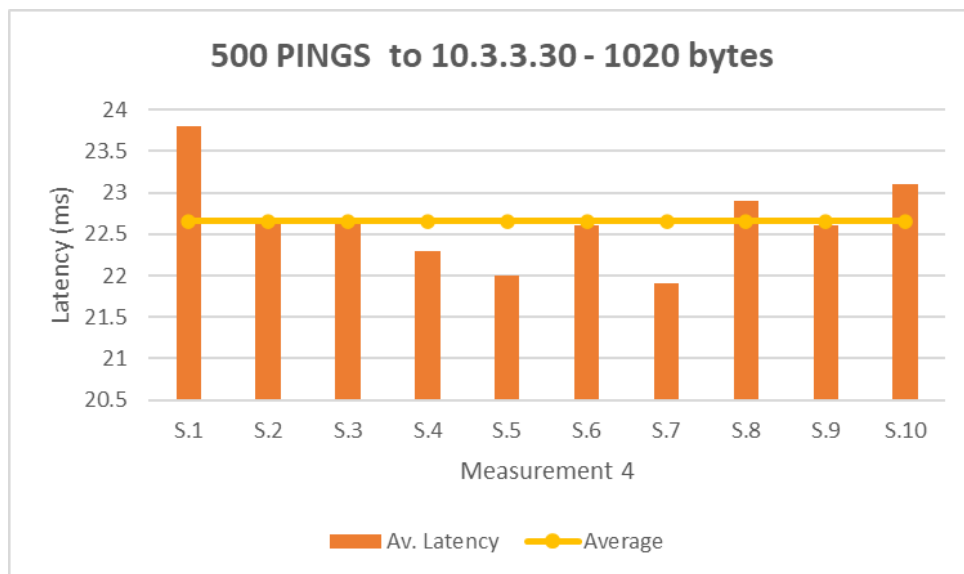


Figure 29: 500 Pings to 10.3.3.30 - 1020 bytes

The average latency values obtained as a conclusion to the tests performed are as follows, Table 5:

| | 35 bytes (Packet size) | 1020 bytes (Packet size) |
|--------------|------------------------|--------------------------|
| Default ping | 28.18 ms | 19.16 ms |
| 500 pings | 22.15 ms | 22.66 ms |

Table 5: Average of performed tests.

4.3 Container Deployment with Docker Compose

In this section, docker-compose tool is used to run applications in different Docker containers. Making use of docker containers makes easy a remotely roll out, in this case in 5TONIC server. 5TONICS server works with Linux. The two applications virtualized

are InfluxDB and Grafana. For this purpose, it is necessary to have installed Docker and Docker-compose [16]

Figure 30 shows the programming code of both applications: InfluxDB on version 1.6.4 and mapping on port 8088 because the default port (8086) is on use (Figure X). Grafana is mapped on port 3001 because port 3000 is in use (Figure 31). The file name must be “*docker-compose.yml*” before executing it.

```
version: '3'
services:
  influxdb:
    image: influxdb:1.6.4
    user: "1000"
    ports:
      - "8088:8086"
    volumes:
      - $HOME/database_induce/var/lib/influxdb:/var/lib/influxdb
  grafana:
    image: grafana/grafana
    user: "1000"
    ports:
      - "3001:3000"
    volumes:
      - $HOME/database_induce/var/lib/grafana:/var/lib/grafana
    environment:
      - GF_INSTALL_PLUGINS=ryantxu-ajax-panel
}
```

Figure 30: *docker-compose.yml*.

```
vepc@cru-600607:~/database_induce$ netstat -putona | grep 8086
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp6      0      0 :::8086          :::*              LISTEN
```

```
vepc@cru-600607:~/database_induce$ netstat -putona | grep 3000
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp6      0      0 :::3000          :::*              LISTEN
```

Figure 31: Default ports availability.

With the following command, Docker is given instructions to create the containers and to run them, Figure 32.

docker-compose up -d

When it is executed, as a result it should see on the screen something similar to Figure X. If it is the first time file execution, you obtain “*done*”.

After that, next command is executed to see all created containers in 5TONIC server. Two created are highlighted in Figure 33. The containers are ready to be used.


```
database_induce_grafana_1 is up-to-date
database_induce_influxdb_1 is up-to-date
```

Figure 32: Containers up to date.

docker ps -a

```
lamboyant_hawking
37401396b02f c77361e47b62 "/bin/sh -c 'go buil..." 3 weeks ago Exited (2) 3 weeks ago v
igorous_newton
3b1b59f52ba3 database_induce_rabbitmq "docker-entrypoint.s..." 8 weeks ago Up 2 weeks 4369/tcp, d
5671/tcp, 0.0.0.0:1883->1883/tcp, 15671/tcp, 15691-15692/tcp, 0.0.0.0:15672->15672/tcp, 25672/tcp, 0.0.0.0:5673->5672/tcp
atabase_induce_rabbitmq_1
c725562aee17 gesture_recognition "bash" 8 weeks ago Exited (0) 8 weeks ago n
ostalgic_nash
3a7860c1b9a7 gesture_recognition "bash" 8 weeks ago Exited (0) 8 weeks ago t
hirsty_rosalind
e46a877986ba gestoos_tesdk_1_7_6 "bash" 8 weeks ago Exited (0) 8 weeks ago s
harp_northcutt
0c579bf418d3 grafana/grafana "/run.sh" 3 months ago Up 2 weeks 0.0.0.0:3 g
000->3000/tcp
rafanaserver_grafana_1
73ff5aabfe70 influxdb "/entrypoint.sh infl..." 3 months ago Up 2 weeks 0.0.0.0:8 g
086->8086/tcp
rafanaserver_influxdb_1
7b167de35279 influxdb:1.6.4 "/entrypoint.sh infl..." 3 months ago Up 2 weeks 0.0.0.0:8 d
088->8086/tcp
atabase_induce_influxdb_1
324ba6649743 grafana/grafana "/run.sh" 3 months ago Up 2 weeks 0.0.0.0:3 d
001->3000/tcp
```

Figure 33: Up containers ready to be used.

4.4 InfluxDB Database

In this section the technology used in relation to data storage will be explained in detail. The use cases generated data needs to be stored because of this an InfluxDB database have been created.

In the server there are more than one InfluxDB, but they are on a different version. It wants to access to the InfluxDB in port 8088 so inside the “*database.induce*” folder the command to execute is:

influx --port 8088

A good way to check that it is the correct influx is check the version shown matches with the version put in the docker-compose file (Figure 34).

```
vepc@cru-600607:~/database_induce$ influx --port 8088
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server management, and monitoring.
Connected to http://localhost:8088 version 1.6.4
InfluxDB shell 0.10.0
>
```

Figure 34: Connected to InfluxDB.

To learn how the database works, at first data will be inserted into the database manually.

4.4.1 InfluxDB Data Stored Manually

The InfluxQL statement or command for creating databases is shown in Figure 35 , it will use the database name “*induce*”[5]:

```
> create database induce
```

Figure 35: Statement to create a “*induce*” database.

Now the database “*induce*” is created, it will use the following command to show the complete set of existing databases, Figure 36:

```
> show databases
name: databases
-----
name
_ internal
 induce
```

Figure 36: Statement to show the existing databases.

It can verify that the database has been created, the “*_internal*” one is created by influxDB to collect the internal metrics at runtime. Most InfluxQL statements only work with a specific database. One option could be mention in each statement the database name, but the following statement automatically set the database for all future queries, Figure 37.

```
> use induce
Using database induce
```

Figure 37: Statement to setting the use of a specific database.

Future commands will only be executed relative to “*modem*” database. InfluxDB is ready to accept queries and writes. As mentioned above Data in InfluxDB is structured by time series which contain a measured value (measurement) as in this case “*modem*”.

Following Figure 38 is an example of data that can be written to InfluxDB manually. Modem is the measurement; “*location*” is a tag-key with “*Ford_VLC*” as a tag-value; and finally, RSSI and BER are both fields with their field-value.

```
> insert modem,location="Ford_VLC" RSSI=-95,BER=0.00000000013
> insert modem,location="Ford_VLC" RSSI=-77,BER=0.00000000011
> insert modem,location="Ford_VLC" RSSI=-98,BER=0.00000000001
```

Figure 38: Statement to insert a single data point.

It will query for the data that we have just wrote, to return all fields and tags with a query it uses the * operator, Figure 39.

```

> select * from modem
name: modem
-----
time                BER      RSSI      location
1660225347775323912 1.3e-10 -95      "Ford_VLC"
1660225358005459278 1.1e-10 -77      "Ford_VLC"
1660225370123818324 1e-11   -98      "Ford_VLC"

```

Figure 39: Data inserted in “*modem*” measurement in “*induce*” database

```

> show measurements
name: measurements
-----
name
modem

```

Figure 40: Statement to show all measurements.

To see all measurements in using database, Figure 40:

It is also possible to display the series from database, the field-keys, the tag, etc. Next figure shows some query examples of the stored data, Figure 41.

```

> show series
key
modem,location="Ford_VLC"

```

```

> show field keys
name: modem
-----
fieldKey      fieldType
BER           float
RSSI          float

```

```

> show tag keys
name: modem
-----
tagKey
location

```

Figure 41: Statements for querying databases.

Before going to the next section, it will be necessary to delete the manually entered information and create the database again. The InfluxQL statement or command to be executed to remove to delete the database created is:

Drop database induce

4.4.2 InfluxDB Data Stored From File

In order to automate the data writing process in InfluxDB, it will in explain in detail the python programming code to load data from a “.*csv*” file, when this script is executed the data is automatically wroten in the database created “*induce*”.

The data generated in the use cases demo currently are not available due to timing considerations. The AGV data for example works with IDs an the translations of the frames is confidential, so “*agv_data*” has been randomly filled just to represent more than one measurement in the database. On the other hand, the modem data have beenn extracted with the 5G commercial network of Orange using the 5G modem of Fivecomm (5G-INDUCE modem). Figure 42 and Figure 43 shows the structure of collected data.

| | A | B | C | D | E | F |
|---|---------------------|------|-----|----------|----------|----------|
| 1 | TIMESTAMP | RSRP | SNR | PING_MIN | PING_AVG | PING_MAX |
| 2 | 2022/07/27-13:10:23 | -92 | 7.5 | 17.492 | 33.843 | 50.369 |
| 3 | 2022/07/27-13:10:31 | -92 | 6.5 | 18.293 | 25.424 | 31.475 |
| 4 | 2022/07/27-13:10:39 | -92 | 5 | 15.959 | 35.819 | 50.895 |
| 5 | 2022/07/27-13:10:48 | -92 | 7.5 | 16.063 | 18 | 18.805 |

Figure 42: “modem_data” file structure.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | |
|---|----------|---|----|-------|---------|------------|-----------|----------|----------------|-----------|----------|-------|-------|---------|---------|---------|-------|---------|----------|---------|---------|--------|---|
| 1 | agv_name | x | y | angle | quality | angleerror | errordist | penddist | directionangle | velocityC | velocity | route | point | segment | lasttag | boolean | laser | display | powerbat | battery | outputs | inputs | |
| 2 | agv_1 | | 10 | 35 | 85 | 9.7 | 0.03 | 0.01 | 13.7 | 12.3 | 5.2 | 2.7 | 5 | 3 | 5 | 4 | 1 | 2 | 3 | 1.2 | 80 | 2 | 3 |
| 3 | agv_2 | | 10 | 35 | 85 | 9.7 | 0.03 | 0.01 | 13.7 | 11.3 | 5.2 | 2.7 | 5 | 3 | 5 | 4 | 1 | 2 | 3 | 1.2 | 80 | 2 | 3 |
| 4 | agv_3 | | 10 | 35 | 85 | 9.7 | 0.03 | 0.01 | 11.7 | 12.3 | 5.2 | 2.7 | 5 | 3 | 5 | 4 | 1 | 2 | 3 | 1.2 | 80 | 2 | 3 |
| 5 | agv_4 | | 10 | 35 | 85 | 9.7 | 0.03 | 0.01 | 13.7 | 11.7 | 5.2 | 1.8 | 5 | 3 | 5 | 4 | 1 | 2 | 3 | 1.2 | 80 | 2 | 3 |

Figure 43: “agv_data” file structure.

The code used to automatically load the “.csv” file to the InfluxDB database is explained below. It is a python script that will load both files to the database as different Measurements, “AgvInformation” and “ModemInformation”.

Figure 44 shows the entire **client** code in Python programming language for reading “modem_data” and “agv_data”.

```
#!/usr/bin/env python3.8
from influxdb import InfluxDBClient
from datetime import datetime

#setup data
import pandas as pd
import sys

def main():
    client = InfluxDBClient(host='10.3.3.30',port=8086)
    client.switch_database('induce') # es el use del influxdb

    data=pd.read_csv('agv_data.csv', sep=';')
    data2=pd.read_csv('modem_data.csv', sep=';')
    json_body= []
    json_body2= []

    for rows_index,row in data.iterrows():#iteracion de cada linea del archivo data
        fields= {}
        for key in row.keys(): #iteracion de cada nombre de columna
            if key != "agv_name": # key es el nombre de columna
                fields[key] = row[key]

        json_body.append({
            "measurement": "AgvInformation",
            "tags": {
                "agv_name": row["agv_name"]
            },
            "fields": fields
        })

    #el for acaba aqui y deja json_body con toda la información

    for rows_index,row in data2.iterrows():#iteracion de cada linea del archivo data
        fields= {}
        for key in row.keys(): #iteracion de cada nombre de columna
            fields[key] = row[key]

        json_body2.append({
            "measurement": "ModemInformation",
            "fields": fields
        })

    print(json_body)
    print(json_body2)

    #send the data
    for i in json_body+json_body2:
        client.write_points([i],protocol='json')

if __name__== '__main__':
    main()
```

Figure 44: Load “.csv” programming code.

First all libraries that are going to be needed are imported. “*InfluxDBClient*” is used to write data to InfluxDB. “*datetime*” just to manipulate dates and times. “*pandas*” is a popular python-based data parsing toolkit that can be imported using `import pandas as pd`. In this case it has been used to read a comma-separated values (csv) file into DataFrame. “*sys*” provides functions and variables used to manipulate different parts of

the Python runtime environment, Figure 45.

```
#!/usr/bin/env python3.8
from influxdb import InfluxDBClient
from datetime import datetime

#setup data
import pandas as pd
import sys
```

Figure 45: Libraries imported.

A new instance of the InfluxDBClient (API) has to be created, the information about the server that it wants to access is **10.3.3.30** (5TONIC remote server). “*switch_database*” is equivalent to the “*use*” command when we inserted data manually. The lines of “*read_csv*” of the code read a “.csv” file separated by “;” . Moreover, a “*json_body*” and “*json_body2*” list is declared in which the data read are collected, Figure 46.

```
def main():
    client = InfluxDBClient(host='10.3.3.30',port=8086)
    client.switch_database('induce') # es el use del influxdb

    data=pd.read_csv('agv_data.csv', sep=';')
    data2=pd.read_csv('modem_data.csv', sep=';')
    json_body= []
    json_body2= []
```

Figure 46: Making a InfluxDB connection.

Next, there are two nested loops. The first one goes through the lines of the file, and the second one moves between the different columns of the line. The “*append()*” function collects the data of each line and it adds the data to the end of the given list. In the last part of Figure 47, the information is parsed to have it prepared before writing it in the database. “*json_body*” and “*json_body2*” list will look like this [17]:

```
json_body=[ {line1} , {line2} , {line3} , {line4} , ... ]
json_body2=[ {line1} , {line2} , {line3} , {line4} , ... ]
```

Finally, the data has to be write in InfluxDB (Figure 48), for this purpose a loop is run through the addition of “*json_body*” and “*json_body2*”. In each iteration the read data is wroten in InfluxDB, “*write_points*” writes to multiple time series names.

It must be verified that the information has been written in “*induce*” InfluxDB. It has to be executed the “*cliente.py*” programming code with Python3. Figure 49 shows that “*induce*” is already created with two measurements and both with the information collected before. So the information has been correctly parsed.

It is verified that in addition to being created, both measurements have been filled in, Figure 50, Figure 51.

```

for rows_index,row in data.iterrows():#iteracion de cada linea del archivo data
fields= {}
for key in row.keys(): #iteracion de cada nombre de columna
    if key != "agv_name": # key es el nombre de columna
        fields[key] = row[key]

json_body.append({
    "measurement": "AgvInformation",
    "tags": {
        "agv_name": row["agv_name"]
    },
    "fields": fields
})

#el for acaba aqui y deja json_body con toda la información

for rows_index,row in data2.iterrows():#iteracion de cada linea del archivo data
fields= {}
for key in row.keys(): #iteracion de cada nombre de columna
    fields[key] = row[key]

json_body2.append({
    "measurement": "ModemInformation",
    "fields": fields
})

print(json_body)
print(json_body2)

```

Figure 47: Data parsing.

```

#send the data
for i in json_body+json_body2:
    client.write_points([i],protocol='json')

if __name__ == '__main__':
    main()

```

Figure 48: Write data into InfluxDB.

```

> show measurements
name: measurements
-----
name
AgvInformation
ModemInformation

```

Figure 49: Measurements of “*induce*” database.

4.5 MQTT Integration

This section describes the RabbitMQ integration in this project. Although the project has always been discussed at a national level, at European level, it is planned to be integrated with the developed platform. In this way, use cases (containers) get implemented on the platform deployed along the project, another reason for virtualizing the database has been necessary.

It is suggested that an easy way to exchange the data generated in the use cases is through an MQTT broker, in this case RabbitMQ. As already mentioned in section 2.4,

```
> select * from ModemInformation
name: ModemInformation
-----
```

| time | PING_AVG | PING_MAX | PING_MIN | RSRP | SNR | TIMESTAMP |
|------------------------------|--------------------|--------------------|--------------------|------|-----|----------------|
| 1661870713846776906 10:23 | 33.843 | 50.369 | 17.492 | -92 | 7.5 | 2022/07/27-13: |
| 1661870713850542586 10:31 | 25.424 | 31.475 | 18.293 | -92 | 6.5 | 2022/07/27-13: |
| 1661870713853265679 10:39 | 35.819 | 50.895 | 15.959000000000001 | -92 | 5 | 2022/07/27-13: |
| 1661870713855771768 10:48 | 18 | 18.805 | 16.063 | -92 | 7.5 | 2022/07/27-13: |
| 1661870713858205323 10:56 | 16.357 | 17.566 | 15.677 | -92 | 6 | 2022/07/27-13: |
| 1661870713860663906 11:04 | 20.953000000000003 | 38.233000000000004 | 14.937000000000001 | -92 | 6.5 | 2022/07/27-13: |
| 1661870713863081013 11:12 | 27.546999999999997 | 38.094 | 15.095 | -92 | 6 | 2022/07/27-13: |
| 1661870713865498189 11:21 | 29.454 | 36.773 | 19.014 | -92 | 6 | 2022/07/27-13: |
| 1661870713867998754 11:29 | 27.638 | 42.818000000000005 | 15.03 | -92 | 6 | 2022/07/27-13: |
| 1661870713870428648 11:37 | 27.345 | 52.9 | 17.148 | -92 | 4 | 2022/07/27-13: |
| 1661870713872809380 11:45 | 24.368000000000002 | 32.272 | 17.726 | -92 | 6 | 2022/07/27-13: |
| 1661870713875300537 11:53 | 30.092 | 51.785 | 19.149 | -92 | 6 | 2022/07/27-13: |
| 1661870713877666291 12:02 | 29.785 | 43.413999999999994 | 18.846 | -92 | 7 | 2022/07/27-13: |
| 1661870713880178558 12:10 | 33.11 | 49.141000000000005 | 18.674 | -92 | 7 | 2022/07/27-13: |

Figure 50: “*AgvInformation*” content.

```
> select * from AgvInformation
name: AgvInformation
-----
```

| time | laser | lasttag | outputs | agv_name | penddist | angle point | angleerror powerbat | battery quality | boolean route | directionangle segment | display velocity | errordist velocityC | inputs x y |
|---------------------|-------|---------|---------|----------|----------|----------------|------------------------|--------------------|------------------|---------------------------|---------------------|------------------------|---------------|
| 1661870713821761367 | | | | agv_1 | | 85 | 0,03 | 80 | 1 | 12,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 13,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 2,7 | 5,2 | 10 | 35 |
| 1661870713830030043 | | | | agv_2 | | 85 | 0,03 | 80 | 1 | 11,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 13,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 2,7 | 5,2 | 10 | 35 |
| 1661870713833283674 | | | | agv_3 | | 85 | 0,03 | 80 | 1 | 12,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 11,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 2,7 | 5,2 | 10 | 35 |
| 1661870713835936271 | | | | agv_4 | | 85 | 0,03 | 80 | 1 | 11,7 | 3 | 0,01 | 3 2 |
| 4 | 2 | 13,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 1,8 | 5,2 | 10 | 35 |
| 1661870713838705012 | | | | agv_5 | | 85 | 0,03 | 80 | 1 | 12,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 10,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 0,7 | 5,2 | 10 | 35 |
| 1661870713841390110 | | | | agv_6 | | 85 | 0,03 | 80 | 1 | 12,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 13,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 2,7 | 5,2 | 10 | 35 |
| 1661870713844162772 | | | | agv_7 | | 85 | 0,03 | 80 | 1 | 12,3 | 3 | 0,01 | 3 2 |
| 4 | 2 | 11,7 | | | 3 | 1,2 | 9,7 | 5 | 5 | 1,8 | 5,2 | 10 | 35 |

Figure 51: “*ModemInformation*” content.

it works as an intermediary between producers and consumers. The RabbitMQ container must be inserted into the generated docker-compose generated above. The following lines will be inserted into the “*services*” section of the code, Figure 52.

The container uses configuration files as “*definitions.json*” and “*rabbitmq.conf*” that have been configured by default. Moreover, this container is also configured with the following Dockerfile. Ports are mapped as 5672 for AMQP, 15672 for HTTP and 1883 for MQTT, Figure 53.

For a good understanding of the whole process, we will split this section into three parts: an explanation of python programming code, publish and receive packets and explanation of RabbitMQ interface functionality. Moreover, Figure 54 shows each of the steps in this process.


```

grafana:
  image: grafana/grafana
  user: "1000"
  ports:
    - "3001:3000"
  volumes:
    - $HOME/database_induce/var/lib/grafana:/var/lib/grafana
  environment:
    - GF_INSTALL_PLUGINS=ryantxu-ajax-panel
rabbitmq:
  build:
    context: .
    dockerfile: Dockerfile
  hostname: my-rabbit
  volumes:
    - ./mqtt-python-examples/rabbitmq/etc/definitions.json:/etc/rabbitmq/definitions.json
    - ./mqtt-python-examples/rabbitmq/etc/rabbitmq.conf:/etc/rabbitmq/rabbitmq.conf
    - ./mqtt-python-examples/rabbitmq/data:/var/lib/rabbitmq/mnesia/rabbit@my-rabbit
    - ./mqtt-python-examples/rabbitmq/logs:/var/log/rabbitmq/log
  ports:
    - 5673:5672
    - 15672:15672
    - 1883:1883
  networks:
    app-network:
      driver: bridge

```

Figure 52: “RabbitMQ” container.

```

FROM rabbitmq:3-management
RUN rabbitmq-plugins enable --offline rabbitmq_mqtt rabbitmq_federation_management rabbitmq_stomp
    rabbitmq_web_mqtt
#RUN echo 'NODENAME=rabbit@localhost' > /etc/rabbitmq/rabbitmq-env.conf

```

Figure 53: Dockerfile of RabbitMQ container.

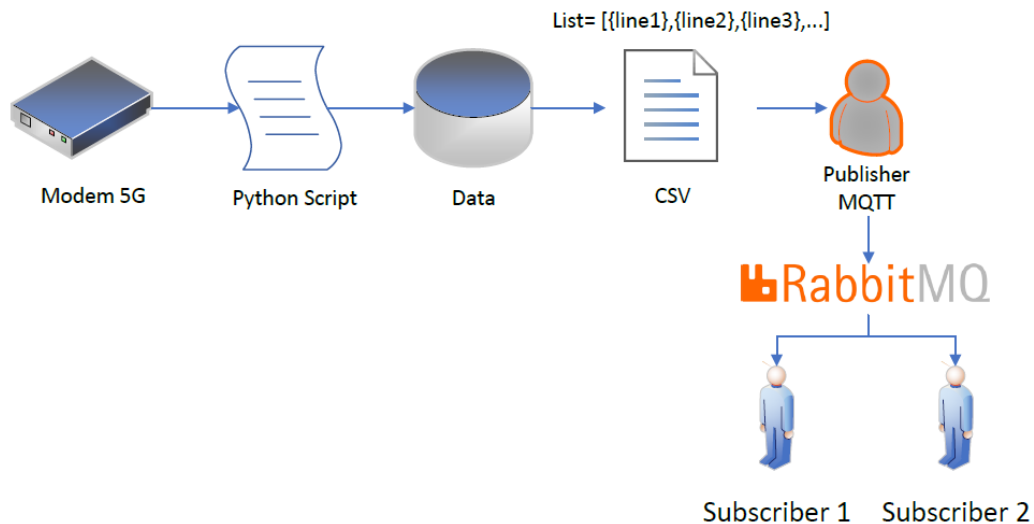


Figure 54: MQTT integration operation scheme.

4.5.1 Python Programming Code

Two programming codes are required, on one hand the producer code who publishing message on RabbitMQ Broker and the subscriber code who is subscribe to the RabbitMQ Broker as a “*listener*” to receive any message that is published. For this purpose, it is necessary to have installed paho-mqtt [18]

Producer code is adapted to read the given “.csv” files, “*modem_data*” and “*agv_data*”. It would be necessary to make small modifications to know how to read the data if other files are going to be use.

Figure 55 shows the entire **producer** code in Python programming language for reading “*modem_data*”.

```
#!/usr/bin/env python3.5
import pandas as pd
import paho.mqtt.client as mqtt
import time, datetime

MQTT_HOST="10.3.3.30"
MQTT_PORT=1883
MQTT_KEEP_ALIVE=60
MQTT_TOPIC="prueba"
MQTT_CLIENT_NAME="PRODUCER"

def main():
    MQTT = mqtt.Client(MQTT_CLIENT_NAME)
    x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE) #broker,port, everytime
    print(x) #si es 0 successful connection

    data2=pd.read_csv('modem_data.csv', sep=';')
    json_body2= []

    for rows_index,row in data2.iterrows():#iteracion de cada linea del archivo data
        fields= {}
        for key in row.keys(): #iteracion de cada nombre de columna
            fields[key] = row[key]

        json_body2.append({ #array de json
            "measurement": "ModemInformation",
            "fields": fields
        })

    #send the data
    for i in json_body2:
        temp_data= str(i).replace("'", '"')
        print(temp_data)
        MQTT.publish(MQTT_TOPIC,temp_data,qos=1)
        MQTT.loop(2)

if __name__== '__main__':
    main()
```

Figure 55: Publisher programming code.

First all libraries that are going to be needed are imported. “*pandas*” already explained. “*paho.mqtt.client*” is a python application programming interface (API) that supports different versions of MQTT. “*time*” and “*datetime*” just to manipulate dates and times, Figure 56.

The variables that are declared before starting the code are (Figure 57):

- **MQTT_HOST=10.3.3.30** is the IP direction of the remotely server.
- **MQTT_PORT=1883** is the traffic port for MQTT traffic on RabbitMQ.

```
#!/usr/bin/env python3.5
import pandas as pd
import paho.mqtt.client as mqtt
import time, datetime
```

Figure 56: Libraries imported.

- **MQTT_KEEP_ALIVE** is just to keep a network connection alive, the default keep alive period for the python MQTT client is 60 seconds.
- **MQTT_TOPIC=prueba** is the name of the topic in which the publication is carried out. Clients publish messages indicating a single topic, but you can be subscriber in more than one topic. However, the RabbitMQ Broker accepts all the topics.
- **MQTT_CLIENT_NAME** It is the client name.

```
MQTT_HOST="10.3.3.30"
MQTT_PORT=1883
MQTT_KEEP_ALIVE=60
MQTT_TOPIC="prueba"
MQTT_CLIENT_NAME="PRODUCER"
```

Figure 57: MQTT variables declaration.

The definition of the main function is the most significant part of the programming code, Figure 58.

```
def main():
    MQTT = mqtt.Client(MQTT_CLIENT_NAME)
    x = MQTT.connect(MQTT_HOST, MQTT_PORT, MQTT_KEEP_ALIVE) #broker, port, everytime
    print(x) #si es 0 successful connection

    data2=pd.read_csv('modem_data.csv', sep=';')
    json_body2= []

    for rows_index,row in data2.iterrows():#iteracion de cada linea del archivo data
        fields= {}
        for key in row.keys(): #iteracion de cada nombre de columna
            fields[key] = row[key]

        json_body2.append({ #array de json
            "measurement": "ModemInformation",
            "fields": fields
        })
```

Figure 58: Main function.

The paho mqtt client class has several methods for connecting, disconnecting, subscribing, unsubscribing, publishing, etc. Each one is associated with a callback.

First of all, again a client instance is created with “*mqtt.client*”, it acts as the client identifier in the connection. Before publishing a message or subscribing to a topic establishing a connection to broker is needed, “*MQTT.connect*” with the variables defined at first. The “*print(x)*” is to know the connection return code. If the connection it is successful the value of x is 0, if the value of x is from 1 to 5 the connection has been refused and if it is higher than 6 is currently unused [20].

Figure 59 shows a reused code from Section 4.4.2.

```
def main():
    MQTT = mqtt.Client(MQTT_CLIENT_NAME)
    x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE) #broker,port,everytime
    print(x) #si es 0 successful connection

    data2=pd.read_csv('modem_data.csv', sep=';')
    json_body2= []

    for rows_index,row in data2.iterrows():#iteracion de cada linea del archivo data
        fields= {}
        for key in row.keys(): #iteracion de cada nombre de columna
            fields[key] = row[key]

        json_body2.append({ #array de json
            "measurement": "ModemInformation",
            "fields": fields
        })
```

Figure 59: Reading data from file.

Finally, the data has to be sent to RabbitMQ broker (Figure 60), for this purpose “*json_body2*” list is run through to convert each line to a string replacing by . To publish messages, it uses the publish method of Paho MQTT “*MQTT.publish*”. The publish one present 4 parameters: topic, payload, quality of service (QoS) and retain. QoS at 1 guarantees that the message will be delivered at least one. “*MQTT.loop*” is set at 2 seconds, that way, program waits two seconds to seconds before closing.

```
#send the data
for i in json_body2:
    temp_data= str(i).replace('"', '')
    print(temp_data)
    MQTT.publish(MQTT_TOPIC,temp_data,qos=1)
    MQTT.loop(2)

if __name__== '__main__':
    main()
```

Figure 60: Sending data to RabbitMQ broker.

Similarly, the programming code of the subscriber is carried out. Figure 61 shows the entire **subscriber** code in Python programming language.

```

#!/usr/bin/env python3.5
#De MQTT a InfluxDB

import paho.mqtt.client as mqtt
import time
import pandas as pd
import json

from influxdb import InfluxDBClient
from datetime import datetime

MQTT_HOST="10.3.3.30"
MQTT_PORT=1883
MQTT_KEEP_ALIVE=60
MQTT_TOPIC="prueba"
MQTT_CLIENT_NAME="CONSUMER"

INFLUX_HOST="10.3.3.30"
INFLUX_PORT=8088
INFLUX_DATABASE="induce"

def main():

    def on_connect(client, userdata, flags, rc):
        print("Connected with result code "+str(rc))
        client.subscribe(MQTT_TOPIC,1)#qoS a 1 para que los mensajes permanezcan

    def on_message(client, userdata, msg):
        payload=msg.payload.decode()#string
        print("Message Received: ",payload)
        print("Message Topic= ",msg.topic)
        print("Message QoS= ",msg.qos)

        #pass str data to json object
        try:
            data=json.loads(payload)
            print(data)
            Iclient.write_points([data], protocol='json')
        except Exception as err:
            print("An error has occurred", str(err))

    #INICIO CÓDIGO

    Iclient = InfluxDBClient(INFLUX_HOST,INFLUX_PORT)

    databases = Iclient.get_list_database()

    if databases != INFLUX_DATABASE:
        Iclient.create_database(INFLUX_DATABASE)
    Iclient.switch_database(INFLUX_DATABASE)

    MQTT = mqtt.Client(MQTT_CLIENT_NAME) #aquí se crea la cola
    MQTT.username_pw_set("guest", "guest")
    MQTT.max_inflight_messages_set(200)
    MQTT.on_connect = on_connect #se conecta
    MQTT.on_message = on_message #se recibe un mensaje

    x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE)
    #brokers.py
    MQTT.loop_forever()#para que se conecte todo el tiempo

if __name__ == '__main__':
    main()

```

Figure 61: Subscriber programming code.

This code uses the libraries already mentioned. Moreover, it needs to import the JSON package and the InfluxDB client. To connect to a InfluxDB, it must create a InfluxDBClient object, Figure 62.

Besides the variables previously declared, it must be declares the Influx variables

```
import paho.mqtt.client as mqtt
import time
import pandas as pd
import json

from influxdb import InfluxDBClient
from datetime import datetime
```

Figure 62: Libraries imported.

,Figure 63.

- **INFLUX_HOST=10.3.3.30** is the IP direction of the remotely server.
- **INFLUX_PORT=8088** is the parsed port for InfluxDB.
- **INFLUX_DATABASE** is the name of the created database or the name of the database to be created.

```
MQTT_HOST="10.3.3.30"
MQTT_PORT=1883
MQTT_KEEP_ALIVE=60
MQTT_TOPIC="prueba"
MQTT_CLIENT_NAME="CONSUMER"

INFLUX_HOST="10.3.3.30"
INFLUX_PORT=8088
INFLUX_DATABASE="induce"
```

Figure 63: Influx variables declaration.

Again, the definition of the main function is the most significant part of the programming code, Figure 64 [21] [22].

```
#INICIO CÓDIGO

Iclient = InfluxDBClient(INFLUX_HOST,INFLUX_PORT)

databases = Iclient.get_list_database()

if databases != INFLUX_DATABASE:
    Iclient.create_database(INFLUX_DATABASE)
Iclient.switch_database(INFLUX_DATABASE)

MQTT = mqtt.Client(MQTT_CLIENT_NAME) #aquí se crea la cola
MQTT.username_pw_set("guest", "guest")
MQTT.max_inflight_messages_set(200)
MQTT.on_connect = on_connect #se conecta
MQTT.on_message = on_message #se recibe un mensaje

x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE)
MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE)
MQTT.loop_forever()#para que se conecte todo el tiempo
```

Figure 64: Main function.

The name of the InfluxDBClient object is *"Iclient"*, It is connected to InfluxDB host in a defined port 8088. The *"databases"* variable is created in which the name of the existing databases are collected. If *"INFLUX_DATABASE"* database is not found in the database list, it is created, if it exists, it is chosen as the database to use, Figure 65.

```
#INICIO CÓDIGO

Iclient = InfluxDBClient(INFLUX_HOST,INFLUX_PORT)

databases = Iclient.get_list_database()

if databases != INFLUX_DATABASE:
    Iclient.create_database(INFLUX_DATABASE)
Iclient.switch_database(INFLUX_DATABASE)
```

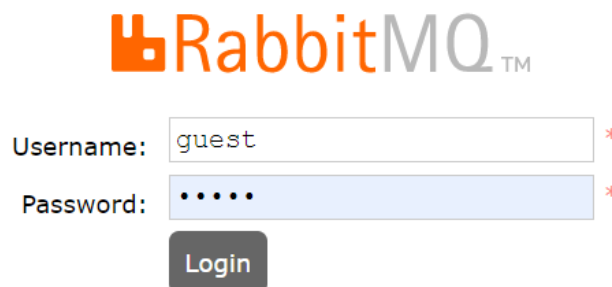
Figure 65: Existing databases.

Again, a client instance is created with *"mqtt.client"*, it acts as the client identifier in the connection. With *"username_pw_set"* set the username and password to login to the RabbitMQ interface. In this case, it is configured with *guest/guest*, Figure 66 and Figure 67.

```
MQTT = mqtt.Client(MQTT_CLIENT_NAME) #aqui se crea la cola
MQTT.username_pw_set("guest", "guest")
MQTT.max_inflight_messages_set(200)
MQTT.on_connect = on_connect #se conecta
MQTT.on_message = on_message #se recibe un mensaje

x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE)
MQTT.on_connect, everytime
MQTT.loop_forever()#para que se conecte todo el tiempo
```

Figure 66: RabbitMQ subscription.



The image shows the RabbitMQ logo at the top. Below it is a login form with two input fields: 'Username:' containing the text 'guest' and 'Password:' containing five dots. Both fields have a red asterisk to their right. Below the password field is a dark grey 'Login' button.

Figure 67: Login in RabbitMQ.

Regarding the size message queue, two parameters must be considered. “*Max_inflight_messages_set*” set the maximum number of messages that can be part of network flow. By default, is limited to 20 but it could be adapted when the final data files are known. Currently, it is set on 200. Increasing this value consume more memory but increase throughput. “*Max_queued_messages_set*” set the maximum number of messages that can be pending in the outgoing message queue. When the queue is full, any further outgoing messages would be dropped. By default, it is unlimited. Currently, It is configured by default.

An aspect to take into account is the size of messages to be sent. Although the theoretical message size limit in RabbitMQ is 2 GB until version 3.7.0, it is not recommend sending messages larger than 128 MB, which is also the new maximum size limit in version 3.8.0 and later. Large messages are especially problematic as they can cause memory and performance problems [23].

Two functions are defined “*on_connect*”, this function is executed when the subscriber is connected to MQTT, Figure 68. If the connection is successfull it gets the message that you are connected with result code 0. In this function, the client also subscribes to the “*MQTT_TOPIC*” in which the data of this topic will be received. The function “*on_message*” is executed each time a message is received, when it happens the following messages are printed: this message has been received, in this topic and with this QoS, Figure 69.

```
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe(MQTT_TOPIC,1)#qoS a 1 para que los mensajes permanezcan
```

Figure 68: “*on_connect*” function.

It has obtain an error when the data is stored into InfluxDB, Figure X. This is because if you have a JSON string data in a program, you have to turn in into JSON in python using “*json.loads*”. It converts it to a python dictionary. Again, it will be use the method “*write_points*”.

```
def on_message(client, userdata, msg):
    payload=msg.payload.decode()#string
    print("Message Received: ",payload)
    print("Message Topic= ",msg.topic)
    print("Message QoS= ",msg.qos)

    #pass str data to json object
    try:
        data=json.loads(payload)
        print(data)
        Iclient.write_points([data], protocol='json')
    except Exception as err:
        print("An error has ocurred", str(err))
```

Figure 69: “*on_message*” function.

It has obtain an error when the data is stored into InfluxDB. This is because if you have a JSON string data in a program, you have to turn in into JSON in python using “*json.loads*”. It converts it to a python dictionary. Again, it will be use the method “*write_points*”.

To improve the code, a try and except was added. When the try part fails, the except is executed. It should mention the type of error generated, in this case Exception, i.e., if a gap of data read is empty, this is an exception [19].

Lastly, the connect line already mention above and the “*loop_forever*”. This loop is a continuous repetitive conditional loop that gets executed until an external factor interferes in the execution flow. The subscriber is always listening for any incoming message, Figure 70.

```
x = MQTT.connect(MQTT_HOST,MQTT_PORT,MQTT_KEEP_ALIVE)
while True:
    MQTT.loop_forever()#para que se conecte todo el tiempo
```

Figure 70: MQTT live subscriber connection.

4.5.2 MQTT Packet Exchange

In this section the programming code of the producer and the subscriber are going to be executed to show how they works. Currently, the Linux directory status in the remote server, 5TONIC is shown in Figure 71. The “*pub2_mqtt.py*” is the final programming code of the producer and the “*sub3_mqtt.py*” is the final code of the subscriber.

```
vepc@cru-600607:~/database_induce$ ls
1 cliente.py Dockerfile mqtt-python-examples sub2-mqtt.py sub-mqtt.py
agv_data.csv docker-compose_influx_grafana.yml modem_data.csv MQTT_Sub.py sub3-mqtt.py var
agv.py docker-compose.yml MQTT_Pub.py pub2-mqtt.py sub4-mqtt.py
```

Figure 71: Linux remote directory status.

Two windows must be open in MobaXterm to **10.3.3.30**, Figure 72. The left window represents the producer, and the right one represents the subscriber.

To execute both files, it is going to be use “*python3*” and the file name, Figure 73. The subscriber is already connected to MQTT successfully. It is waiting any incoming messages.

A message will be sent “*pub2_mqtt.py*” to the same topic in which the subscriber is subscribed “*MQTT_TOPIC*”, Figure 74. Data sent from “*modem_data*” is displayed on the screen. At the same time, an alarm signal (it turns blue) is detected in the window of the subscriber, Figure 75.

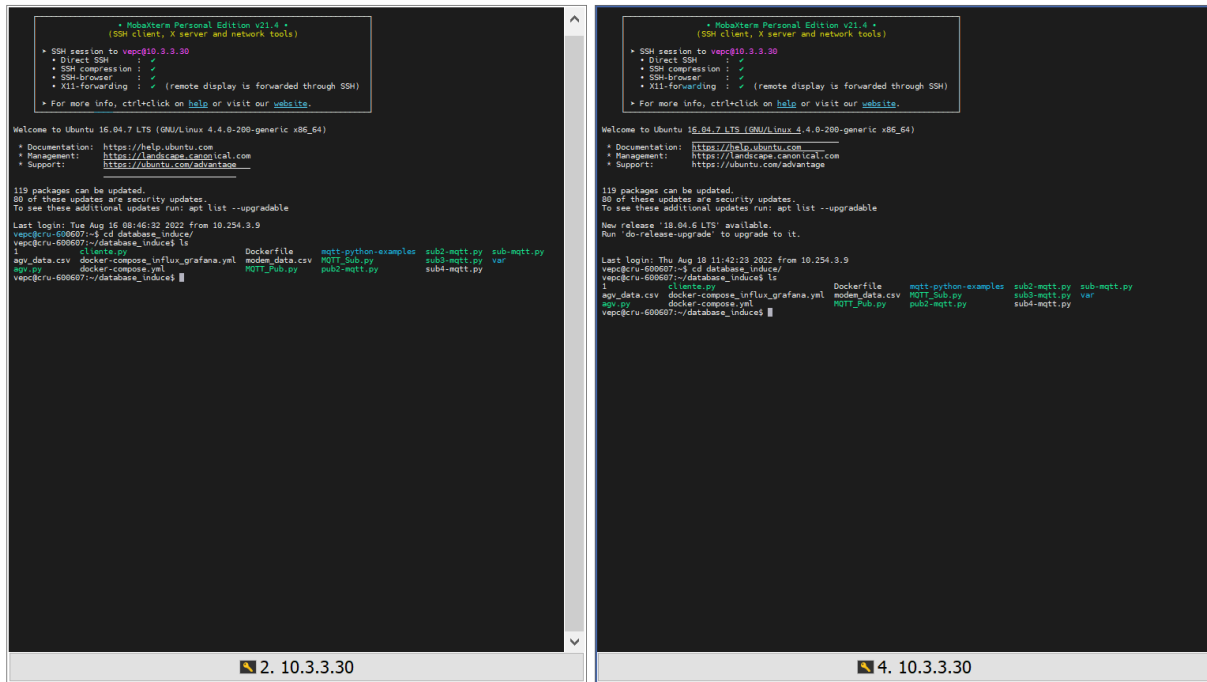


Figure 72: MobaXterm session.

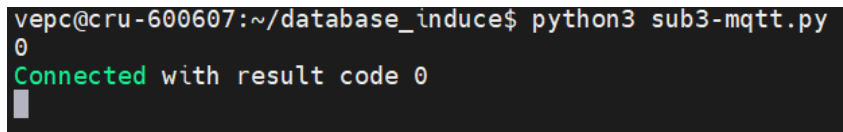


Figure 73: Subscriber connected.

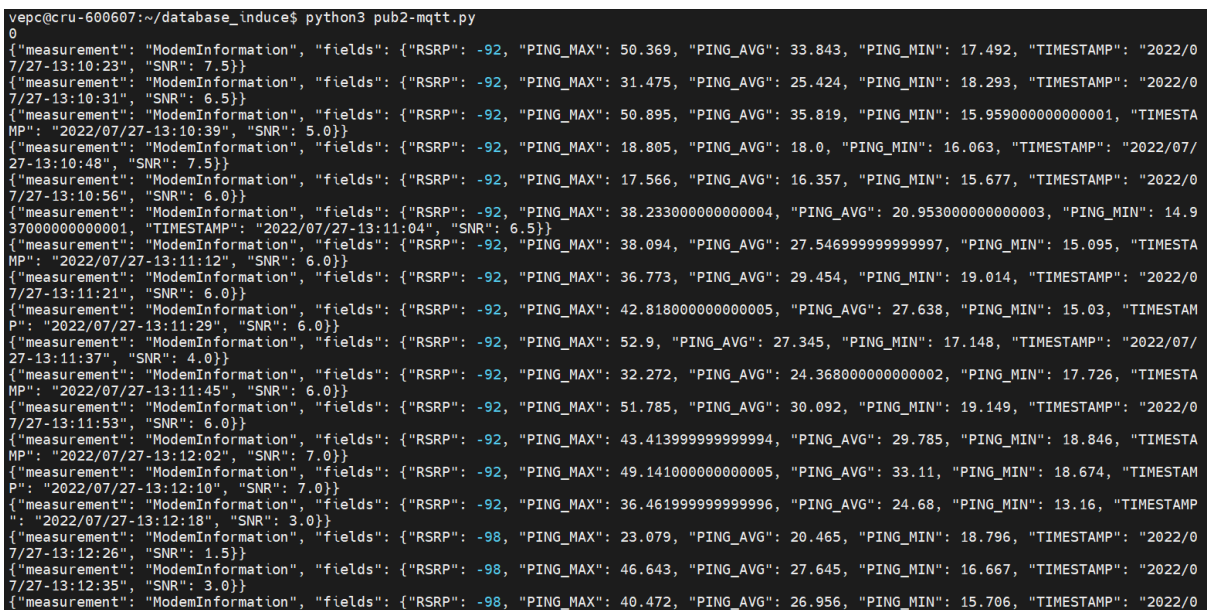


Figure 74: Data already sent by the producer.

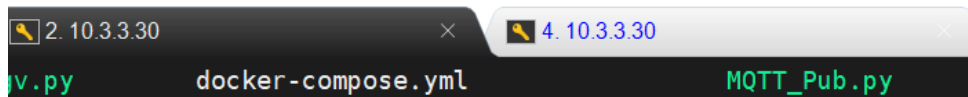


Figure 75: Blue alarm signal by the producer.

In the subscriber, it can be checked the payload, the topic and the QoS of each line sent and printed, Figure 76.

```
vepc@cru-600607:~/database_induce$ python3 sub3-mqtt.py
0
Connected with result code 0
Message Received: {"measurement": "ModemInformation", "fields": {"RSRP": -92, "PING_MAX": 50.369, "PING_AVG": 33.843, "PING_MIN": 17.492, "TIMESTAMP": "2022/07/27-13:10:23", "SNR": 7.5}}
Message Topic= prueba
Message QoS= 1
{"measurement": "ModemInformation", "fields": {"TIMESTAMP": "2022/07/27-13:10:23", "RSRP": -92, "PING_MAX": 50.369, "PING_MIN": 17.492, "PING_AVG": 33.843, "SNR": 7.5}}
Message Received: {"measurement": "ModemInformation", "fields": {"RSRP": -92, "PING_MAX": 31.475, "PING_AVG": 25.424, "PING_MIN": 18.293, "TIMESTAMP": "2022/07/27-13:10:31", "SNR": 6.5}}
Message Topic= prueba
Message QoS= 1
{"measurement": "ModemInformation", "fields": {"TIMESTAMP": "2022/07/27-13:10:31", "RSRP": -92, "PING_MAX": 31.475, "PING_MIN": 18.293, "PING_AVG": 25.424, "SNR": 6.5}}
Message Received: {"measurement": "ModemInformation", "fields": {"RSRP": -92, "PING_MAX": 50.895, "PING_AVG": 35.819, "PING_MIN": 15.959000000000001, "TIMESTAMP": "2022/07/27-13:10:39", "SNR": 5.0}}
Message Topic= prueba
Message QoS= 1
{"measurement": "ModemInformation", "fields": {"TIMESTAMP": "2022/07/27-13:10:39", "RSRP": -92, "PING_MAX": 50.895, "PING_MIN": 15.959000000000001, "PING_AVG": 35.819, "SNR": 5.0}}
Message Received: {"measurement": "ModemInformation", "fields": {"RSRP": -92, "PING_MAX": 18.805, "PING_AVG": 18.0, "PING_MIN": 16.063, "TIMESTAMP": "2022/07/27-13:10:48", "SNR": 7.5}}
Message Topic= prueba
Message QoS= 1
{"measurement": "ModemInformation", "fields": {"TIMESTAMP": "2022/07/27-13:10:48", "RSRP": -92, "PING_MAX": 18.805, "PING_MIN": 16.063, "PING_AVG": 18.0, "SNR": 7.5}}
Message Received: {"measurement": "ModemInformation", "fields": {"RSRP": -92, "PING_MAX": 17.566, "PING_AVG": 16.357, "PING_MIN": 15.677, "TIMESTAMP": "2022/07/27-13:10:56", "SNR": 6.0}}
Message Topic= prueba
Message QoS= 1
```

Figure 76: Data sent and printed.

It must be verified that the information has been written in “*induce*” InfluxDB. Figure 77 shows that “*induce*” is already created. Besides, Figure 78 shows the stored data in InfluxDB from “*ModemInformation*”.

```
vepc@cru-600607:~$ cd database_induce/
vepc@cru-600607:~/database_induce$ influx --port 8088
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server management, and monitoring.
Connected to http://localhost:8088 version 1.6.4
InfluxDB shell 0.10.0
> show databases
name: databases
-----
name
 _internal
 induce
```

Figure 77: “*induce*” database has been created.

4.5.3 RabbitMQ Managing Interface

RabbitMQ Management is an easy to use interface that allows to monitor and manage your RabbitMQ server from a web browser. Queues, connexions, channels, exchanges and etc can be managed in the browser. Moreover, message rates can be monitored,

```
> select * from ModemInformation
name: ModemInformation
-----
```

| time | PING_AVG | PING_MAX | PING_MIN | RSRP | SNR | TIMESTAMP |
|---------------------|--------------------|--------------------|--------------------|------|-----|--------------------|
| 1660820804173608860 | 33.843 | 50.369 | 17.492 | -92 | 7.5 | 2022/07/27-13:10:2 |
| 3 | | | | | | |
| 1660820804181759299 | 25.424 | 31.475 | 18.293 | -92 | 6.5 | 2022/07/27-13:10:3 |
| 1 | | | | | | |
| 1660820804184432245 | 35.819 | 50.895 | 15.959000000000001 | -92 | 5 | 2022/07/27-13:10:3 |
| 9 | | | | | | |
| 1660820804187167726 | 18 | 18.805 | 16.063 | -92 | 7.5 | 2022/07/27-13:10:4 |
| 8 | | | | | | |
| 1660820804190035003 | 16.357 | 17.566 | 15.677 | -92 | 6 | 2022/07/27-13:10:5 |
| 6 | | | | | | |
| 1660820804192775604 | 20.953000000000003 | 38.233000000000004 | 14.937000000000001 | -92 | 6.5 | 2022/07/27-13:11:0 |
| 4 | | | | | | |
| 1660820804195484298 | 27.546999999999997 | 38.094 | 15.095 | -92 | 6 | 2022/07/27-13:11:1 |
| 2 | | | | | | |
| 1660820804198240827 | 29.454 | 36.773 | 19.014 | -92 | 6 | 2022/07/27-13:11:2 |
| 1 | | | | | | |
| 1660820804201003114 | 27.638 | 42.818000000000005 | 15.03 | -92 | 6 | 2022/07/27-13:11:2 |
| 9 | | | | | | |
| 1660820804203790461 | 27.345 | 52.9 | 17.148 | -92 | 4 | 2022/07/27-13:11:3 |
| 7 | | | | | | |
| 1660820804206994306 | 24.368000000000002 | 32.272 | 17.726 | -92 | 6 | 2022/07/27-13:11:4 |
| 5 | | | | | | |
| 1660820804209769985 | 30.092 | 51.785 | 19.149 | -92 | 6 | 2022/07/27-13:11:5 |
| 3 | | | | | | |
| 1660820804212483931 | 29.785 | 43.413999999999994 | 18.846 | -92 | 7 | 2022/07/27-13:12:0 |
| 2 | | | | | | |
| 1660820804215077558 | 33.11 | 49.141000000000005 | 18.674 | -92 | 7 | 2022/07/27-13:12:1 |
| 0 | | | | | | |
| 1660820804217729818 | 24.68 | 36.461999999999996 | 13.16 | -92 | 3 | 2022/07/27-13:12:1 |
| 8 | | | | | | |
| 1660820804220285720 | 20.465 | 23.079 | 18.796 | -98 | 1.5 | 2022/07/27-13:12:2 |
| 6 | | | | | | |
| 1660820804222888529 | 27.645 | 46.643 | 16.667 | -98 | 3 | 2022/07/27-13:12:3 |

Figure 78: Stored data from “*ModemInformation*”

and messages can be sent/received manually. This section gives information about the different views that can be found in RabbitMQ Management interface.

RabbitMQ Management provides a single static HTML page that performs background queries to the RabbitMQ HTTP API. Information from the management interface can be useful when you are debugging your application. If RabbitMQ is install, in this case in the remote server 5TONIC, to access it go to [RabbitMQ Management](#), Figure 79 [24].

RabbitMQ has different views in which the time interval shown can be changed, Figure 80[25]:

- **Overview:** he overview window includes several charts and tables. For instance, queue messages , messages rate, global account, listening ports, and etc.
- **Connections:** It shows the connections established to the RabbitMQ server.
- **Channels:** This tab shows information about all current channels.
- **Exchanges:** It receives messages from producers and pushes them to queues. They should know exactly what to do when a message is received.
- **Queues:** The queue tab shows the queues for all or one selected vhost.
- **Admin:** From the Admin view, it is possible to add users and change user permissions.

In order to visualize in the RabbitMQ interface the process of the producer and the subscriber. It is going to be executed the created programming codes “*pub2_mqtt.py*” and

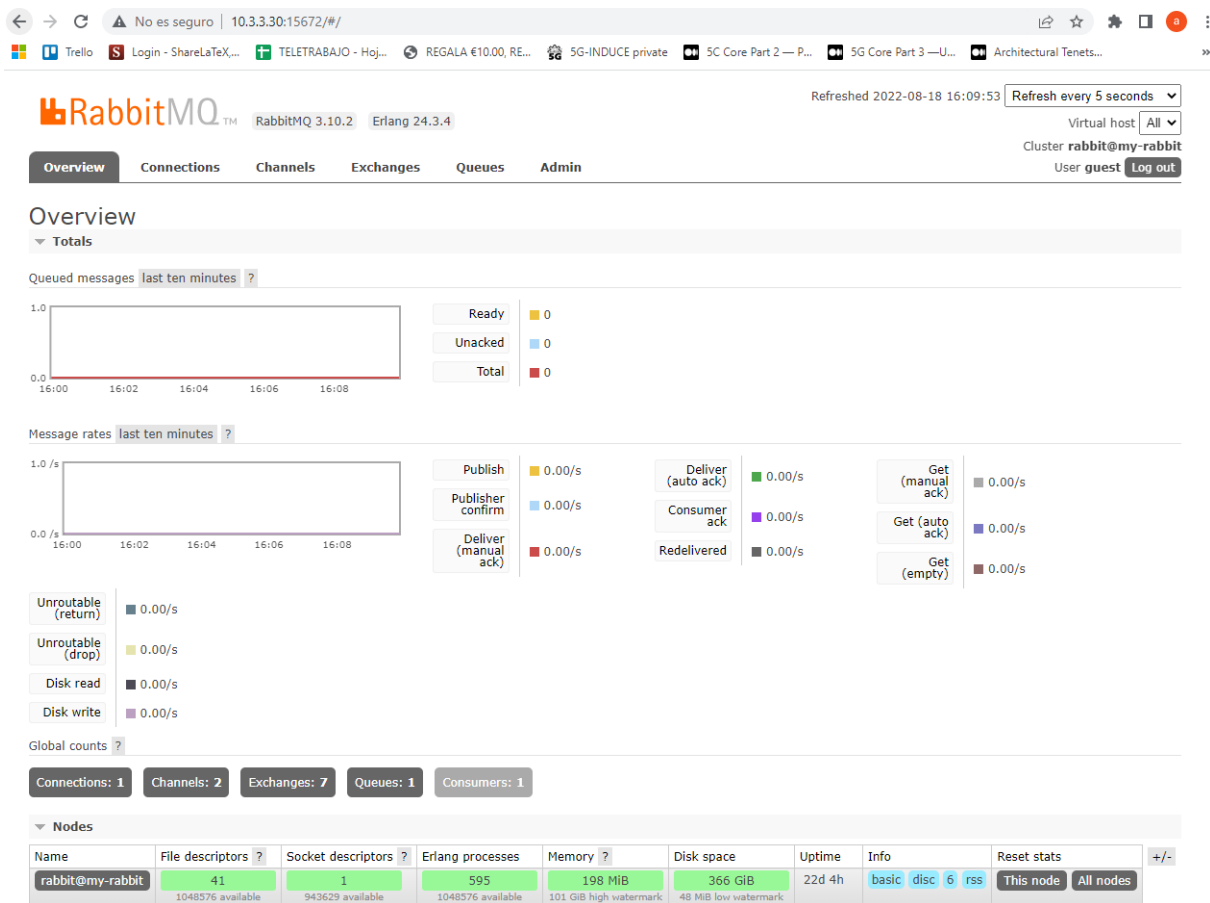


Figure 79: RabbitMQ interface.



Figure 80: RabbitMQ views.

“sub3_mqtt.py” Depending on the number of times we run the programs we will send one or more messages and it will have one or more subscribers.

- Publishing messages with one subscriber.

The currently state of RabbitMQ interface is shown in Figure 81.

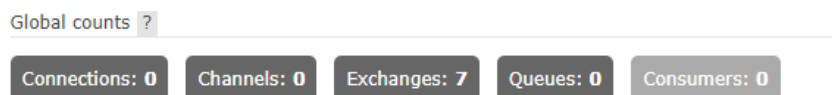


Figure 81: RabbitMQ state.

When a subscriber is connected, Figure 82, it generates one connection, Figure 83, two channels Figure 84, one queue, Figure 85, and one consumer, Figure 86.

```
vepc@cru-600607:~/database_induce$ python3 sub3-mqtt.py
0
Connected with result code 0
```

Figure 82: Subscriber connected.

Connections

▼ All connections (1)

Page 1 of 1 - Filter: Regex ?

Displaying 1 item , page size up to: 100

| Overview | | | Details | | | Network | | +/- | |
|-------------------|-----------|---------|-----------|------------|----------|-------------|-----------|-----|--|
| Name | User name | State | SSL / TLS | Protocol | Channels | From client | To client | | |
| 10.3.3.30:51591 ? | guest | running | o | MQTT 3.1.1 | 1 | 0 B/s | 0 B/s | | |

Figure 83: Subscriber connected.

Channels

▼ All channels (2)

Page 1 of 1 - Filter: Regex ?

Displaying 2 items , page size up to: 100

| Overview | | | | Details | | | Message rates | | | | | +/- | |
|---------------------|-----------|--------|-------|-------------|------------|---------|---------------|---------|-------------------|---------------|-----|-----|--|
| Channel | User name | Mode ? | State | Unconfirmed | Prefetch ? | Unacked | publish | confirm | unroutable (drop) | deliver / get | ack | | |
| 10.3.3.30:51591 (1) | guest | | idle | 0 | 10 | 0 | | | | | | | |
| 10.3.3.30:51591 (2) | guest | | idle | 0 | | 0 | | | | | | | |

Figure 84: Generated connection with one subscriber.

Queues

▼ All queues (1)

Page 1 of 1 - Filter: Regex ?

Displaying 1 item , page size up to: 100

| Overview | | | | | Messages | | | | Message rates | | | +/- | |
|--------------------------------|---------|----------|--------|-------|----------|---------|------------|-------|---------------|---------------|-----|-----|--|
| Name | Type | Features | Policy | State | Ready | Unacked | Persistent | Total | incoming | deliver / get | ack | | |
| mqtt-subscription-CONSUMERqos1 | classic | D AD | ? | idle | 0 | 0 | 0 | 0 | | | | | |

Figure 85: Generated queue with one subscriber.

Global counts ?

Connections: 1 Channels: 2 Exchanges: 7 Queues: 1 Consumers: 1

Figure 86: RabbitMQ state with one subscriber.

The connection established is between **10.3.3.30** and **172.20.0.3** in port 1883, IP address of the container in remotely 5TONIC server. Figure X shown all the information of the created connection: user (in this case guest), the client ID, configured

in the above programming code, the product MQTT client, even the message rates when get an incoming message, Figure 87.

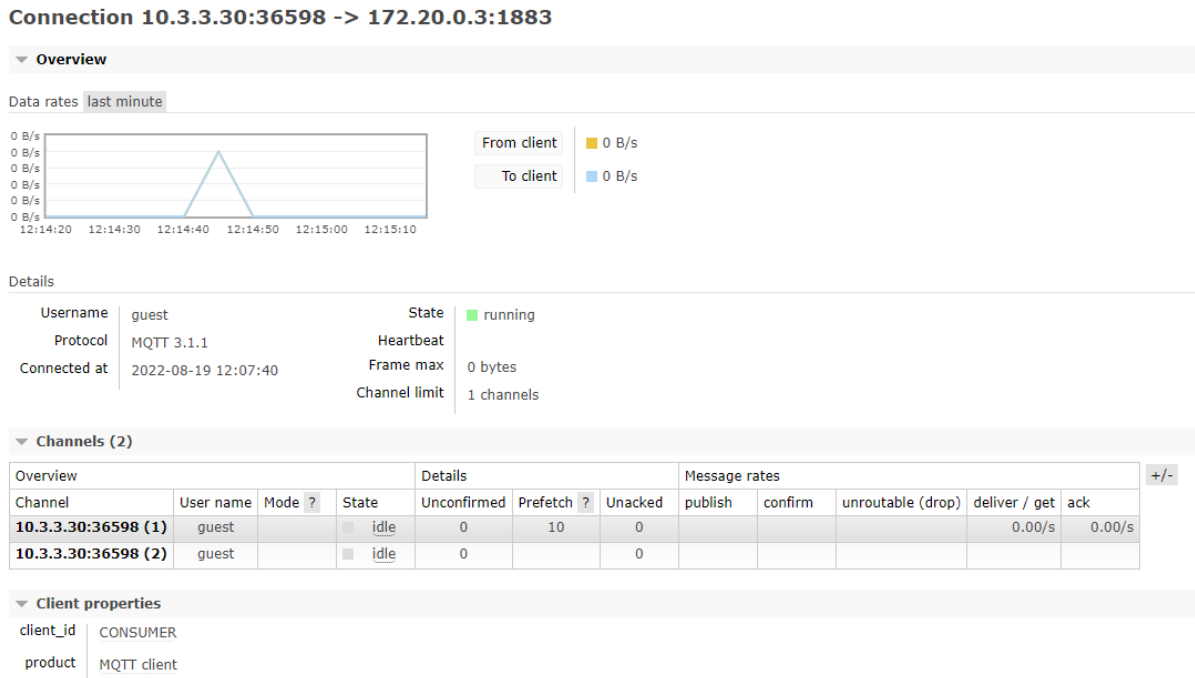


Figure 87: Message data rate with one message incoming.

A binding can be created between an exchange and a queue. All active bindings to the queue are shown under bindings. You can also create a new binding to a queue from here or unbind a queue from an exchange. In this case it uses the default exchange binding “amq.topic” and with the routing key “prueba”, Figure 88.

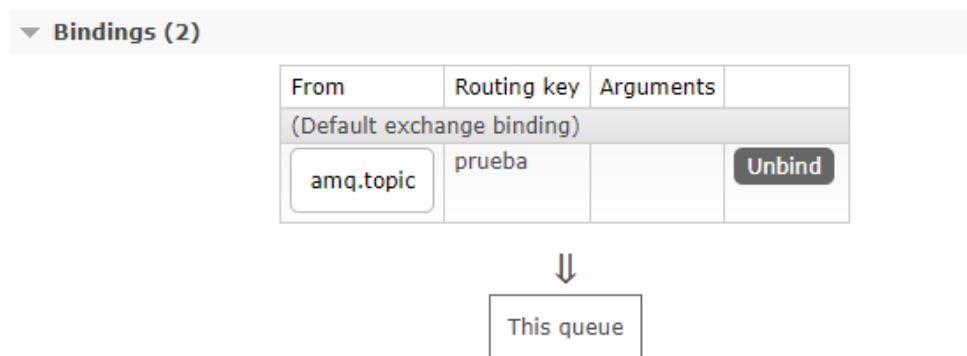


Figure 88: In-use binding in the 5G-INDUCE project.

Then, the number of published messages is increased and instead of publishing just one message it publishes messages regularly. This is displayed in RabbitMQ interface, Figure 89. Zooming in the incoming messages, Figure 90, you can see

the different lines and rates per second depending on the time of sending, publisher confirm, consumer ack, get manual ack, etc.

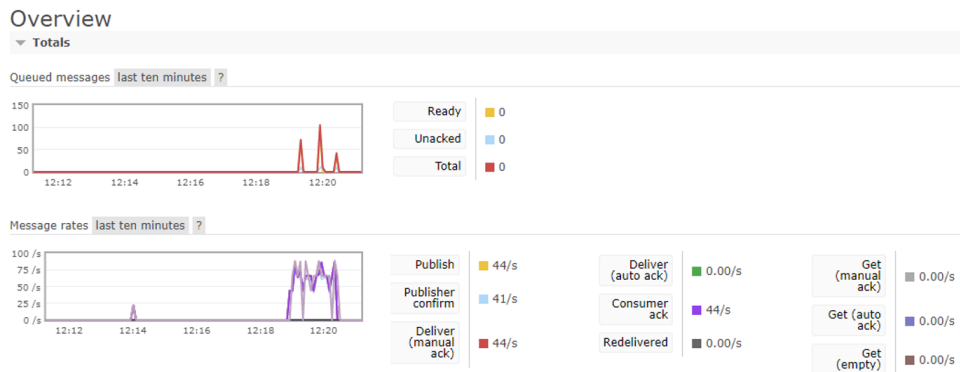


Figure 89: Incoming messages to MQTT broker.

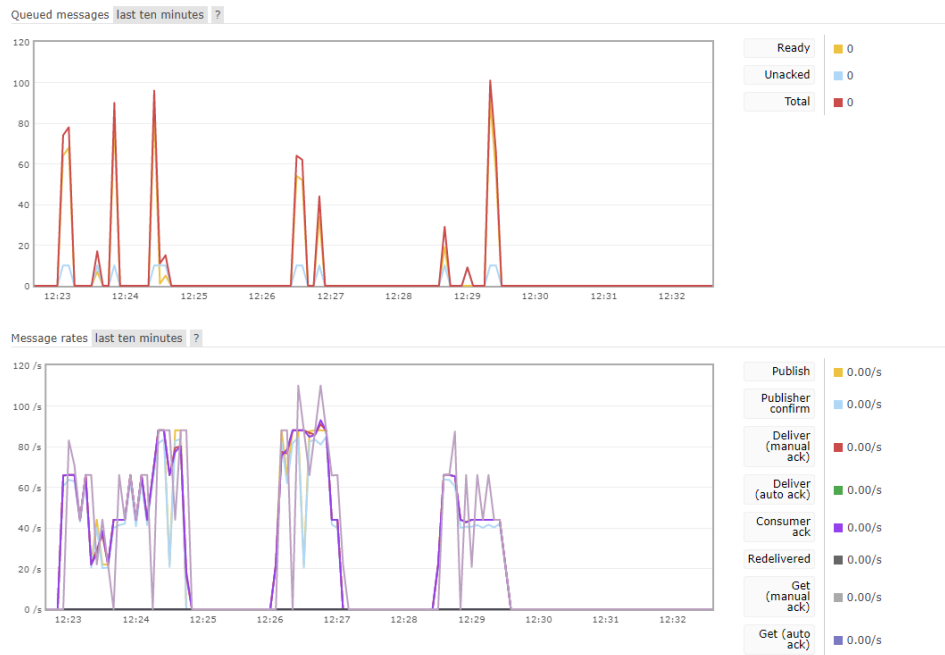


Figure 90: Zoom into incoming messages to MQTT broker.

If you click on the generated connection, you can see the graph of the connection speed during transmission (data rate), Figure 91.

To see a video of the RabbitMQ interface status during regular sending of messages, click on: [OneDrive-RabbitMQ video](#)

- **Publishing messages with two subscribers.**

It is going to be added another subscriber to the same topic, Figure 92.

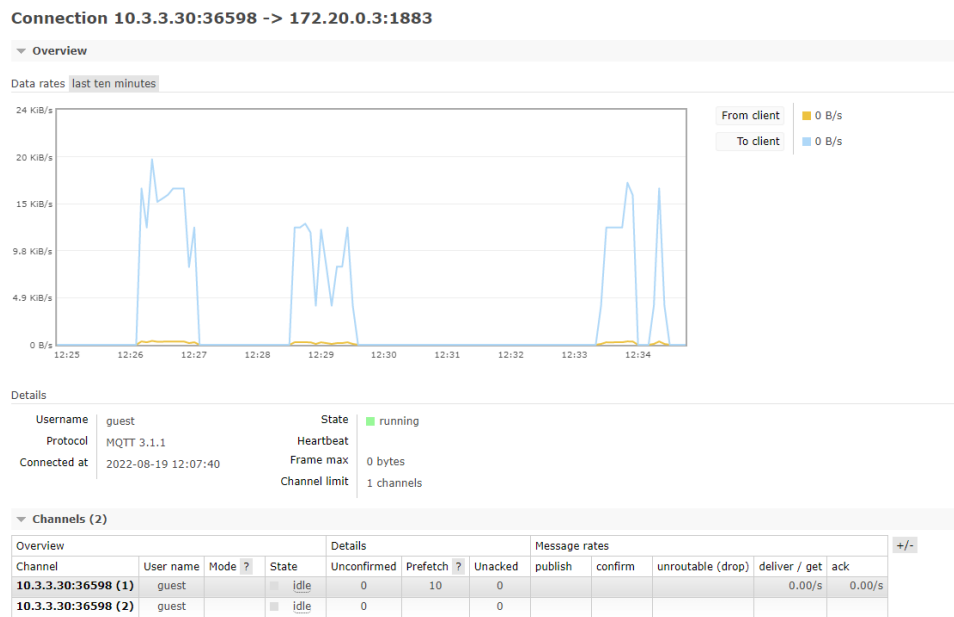


Figure 91: Data rate of incoming messages to MQTT broker.

```
vepc@cru-600607:~/database_induce$ python3 sub3-mqtt.py
0
Connected with result code 0
```

Figure 92: Connected subscriber.

So, right now the currently state of RabbitMQ interface is shown in Figure 93. When two subscribers are connected, Figure 93, they generate two connections, Figure 94, four channels Figure 95, two queues, Figure 96. It is important to be carefully with the “*MQTT_CLIENT_NAME*” variable because reusing the same code and connecting two subscribers with the same one, both subscribers would be subscribed with the same name. This way the connection loops and does not receive any messages. If this variable is empty a random name is generated for each subscription.

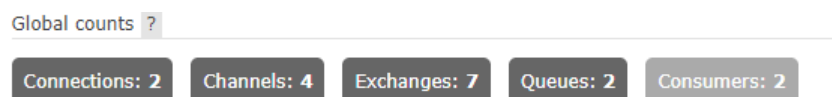


Figure 93: RabbitMQ state with two subscribers.

Connections

▼ All connections (2)

Pagination

Page of 1 - Filter: Regex ?

| Overview | | | Details | | | Network | | +/- |
|----------------------|-----------|--|-----------|------------|----------|-------------|-----------|-----|
| Name | User name | State | SSL / TLS | Protocol | Channels | From client | To client | |
| 10.3.3.30:40561 ? | guest | ■ running | ○ | MQTT 3.1.1 | 1 | 0 B/s | 0 B/s | |
| 10.3.3.30:58891 ? | guest | ■ running | ○ | MQTT 3.1.1 | 1 | 0 B/s | 0 B/s | |

Figure 94: Generated connections with two subscribers.

Channels

▼ All channels (4)

Pagination

Page of 1 - Filter: Regex ?

Displaying 4 items , page size up to:

| Overview | | | | Details | | | Message rates | | | | | | +/- |
|---------------------|-----------|--------|--|-------------|------------|---------|---------------|---------|-------------------|---------------|-----|--|-----|
| Channel | User name | Mode ? | State | Unconfirmed | Prefetch ? | Unacked | publish | confirm | unroutable (drop) | deliver / get | ack | | |
| 10.3.3.30:40561 (1) | guest | | ■ idle | 0 | 10 | 0 | | | | | | | |
| 10.3.3.30:40561 (2) | guest | | ■ idle | 0 | | 0 | | | | | | | |
| 10.3.3.30:58891 (1) | guest | | ■ idle | 0 | 10 | 0 | | | | | | | |
| 10.3.3.30:58891 (2) | guest | | ■ idle | 0 | | 0 | | | | | | | |

Figure 95: Generated channels with two subscribers.

Queues

▼ All queues (2)

Pagination

Page of 1 - Filter: Regex ?

Displaying 2 items , page size up to:

| Overview | | | | | | Messages | | | | Message rates | | | +/- |
|--|---------|--|--------|--|-------|----------|------------|-------|----------|---------------|-----|--|-----|
| Name | Type | Features | Policy | State | Ready | Unacked | Persistent | Total | incoming | deliver / get | ack | | |
| mqtt-subscription-EaKFc0A3KICezLyqWmI-Ywqos1 | classic | D AD | ? | ■ idle | 0 | 0 | 0 | 0 | | | | | |
| mqtt-subscription-HUY1JfeF4hH1eG-SYlWwSwqos1 | classic | D AD | ? | ■ idle | 0 | 0 | 0 | 0 | | | | | |

Figure 96: Generated queues with two subscribers.

At the same time in MobaXterm, an alarm signal (it turns blue) is detected in the window of both subscriber, Figure 97, Figure 98.

Then, it publishes messages regularly. Again, this is displayed in RabbitMQ interface, Figure 99.



Figure 97: Blue alarm signal in both subscribers

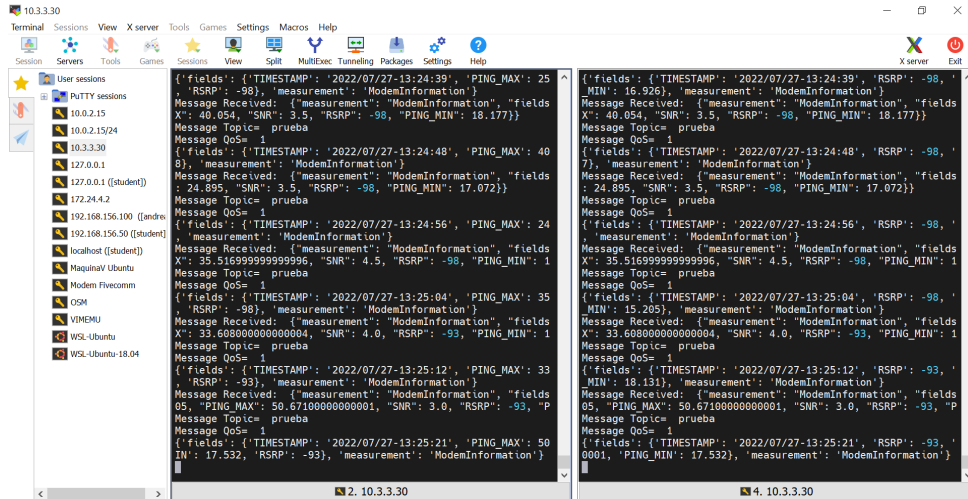


Figure 98: Messages received for both subscribers.



Figure 99: Incoming messages to MQTT broker.

4.6 Grafana

The data stored in InfluxDB want to be monitoring with Grafana. To access to the Grafana interface the browser has to be opened with the IP address of the server, 10.3.3.30 and the port in which it is mapped 3001, to access it go to [Grafana Interface](#), Figure 100 and Figure 101. The default username and password are admin/admin. Once it has been entered, it gets the homepage of Grafana.

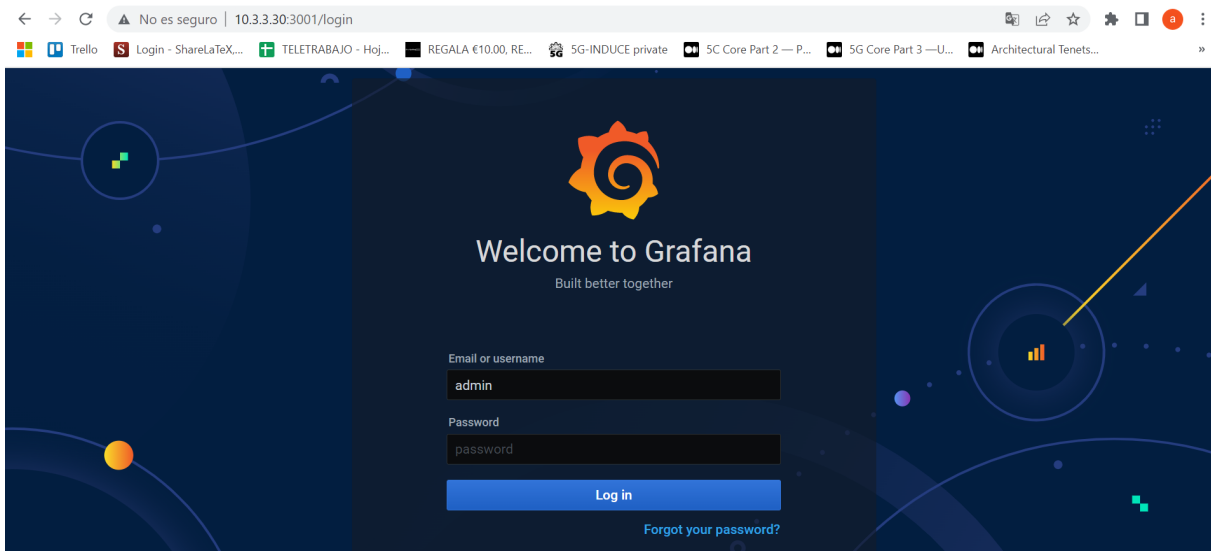


Figure 100: Grafana access interface.

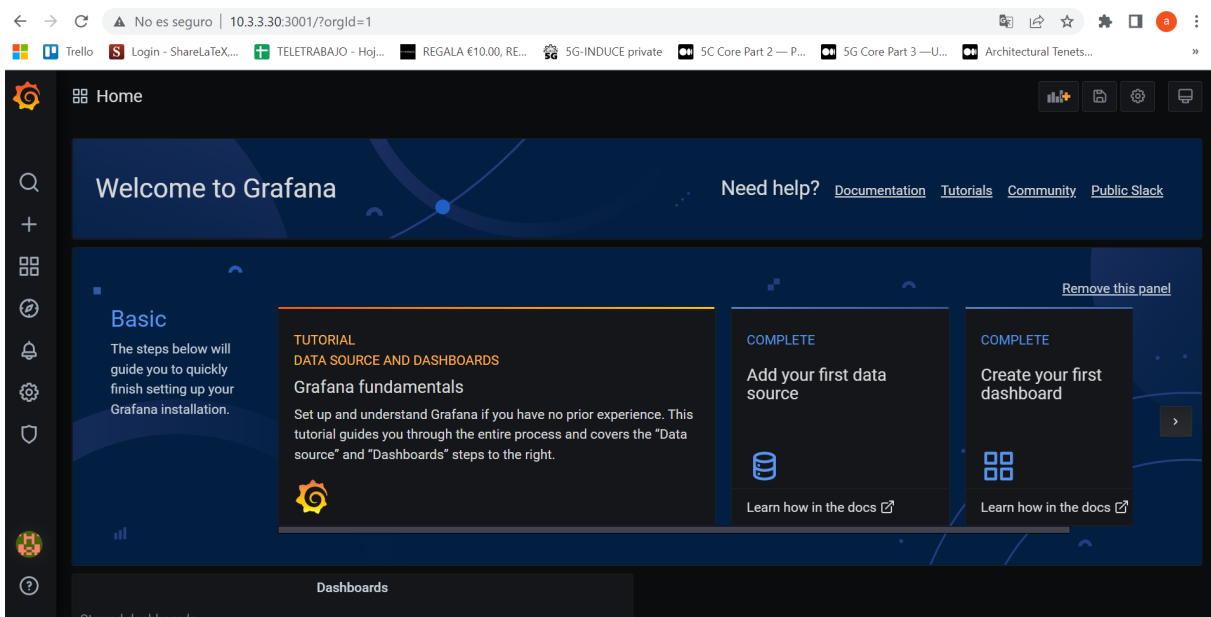


Figure 101: Homepage of Grafana.

Before creating the first dashboard, it is needed to add a data source. To add a data source, it should move to the configuration icon in the left menu. It shows the configuration options, Figure 102.

The data sources page displays a list of pre-configured data sources. Clicking on add data source and introducing a specific data source, in this case *"InfluxDB"* in the search dialog, Figure 103, it is carried out the configuration of a InfluxDB query. In this example, InfluxDB it is just configured for the project.

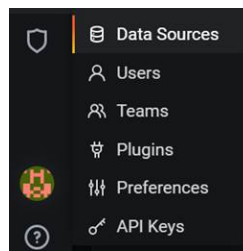


Figure 102: Configuration menu.

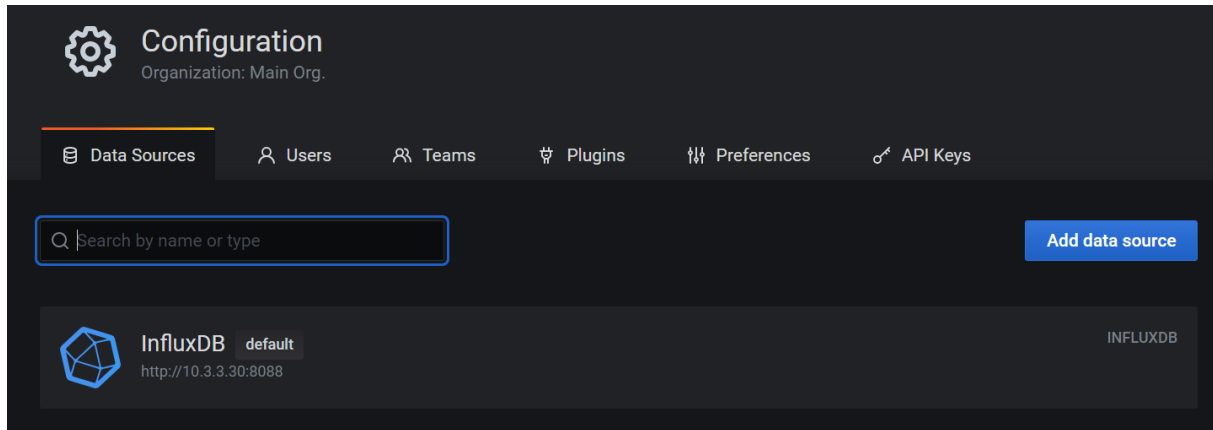


Figure 103: Data sources tab.

Some parameters of the default settings should be maintained. The name is changed into “*InfluxDB*”. The query language is InfluxQL due to the InfluxDB version in use 1.6. Through an HTTP request the URL `http://10.3.3.30:8088` is accessed (browser), Figure 104.

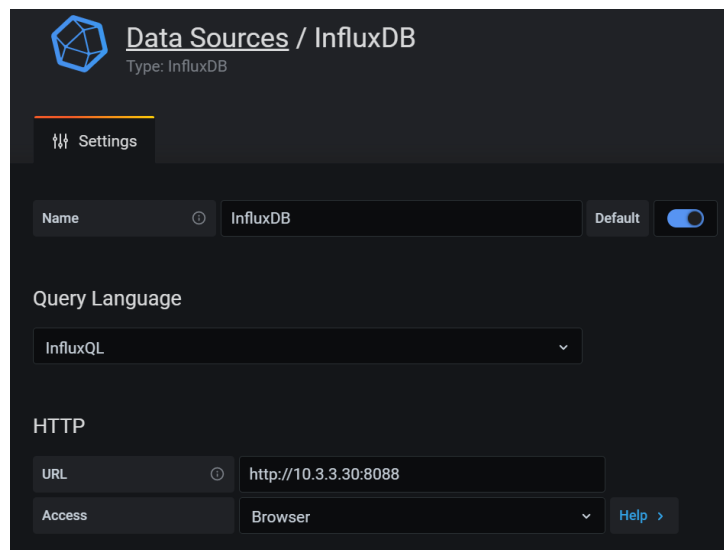


Figure 104: InfluxDB data source configuration.

The database has to be define, it should put the same name that the database created in the python programming code, *“induce”*. Moreover, the rest of parameters are configured by default, Figure 105.

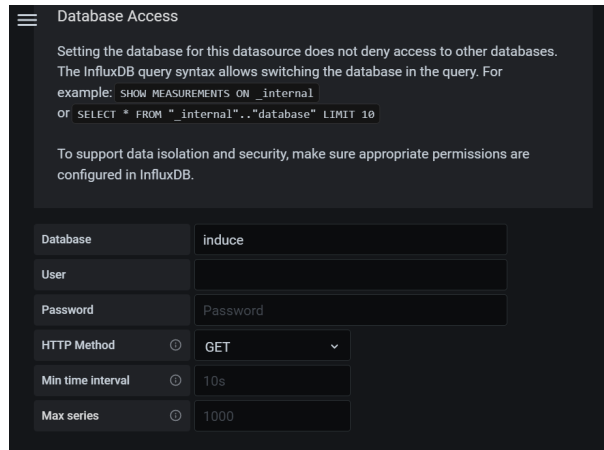


Figure 105: Database access configuration.

A dashboard has been created to see all data stored. For this, click on New Dashboard and choose a name, in this case 5G_INDUCE, Figure 106. Once inside, it is time to add panels to visualize the information stored in the database.

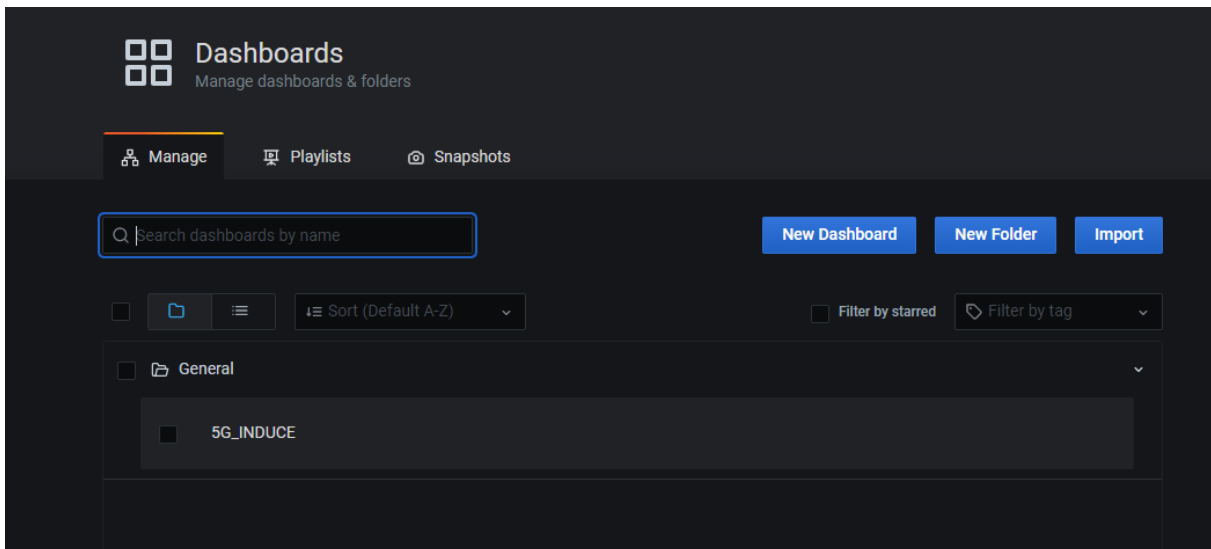


Figure 106: New dashboard.

As you can see in Figure 107 and Figure 108, each of the parts of a query to a database appear in the interface, if we click on measurement, we can see the two stored measurements. The same happens in field(value) where you can see each of the modem parameters.

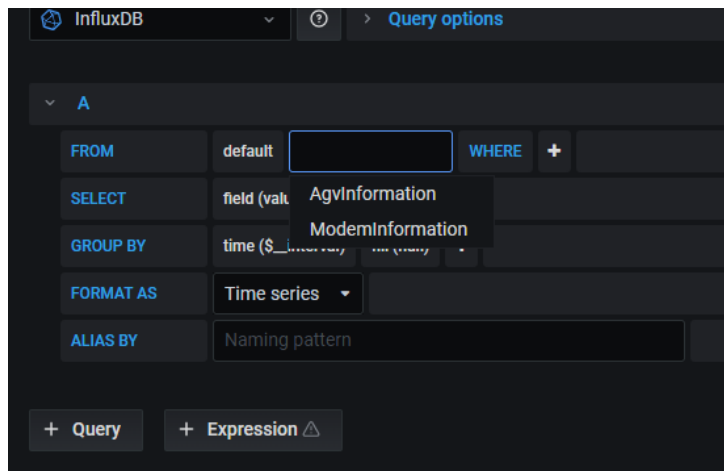


Figure 107: Measurement.

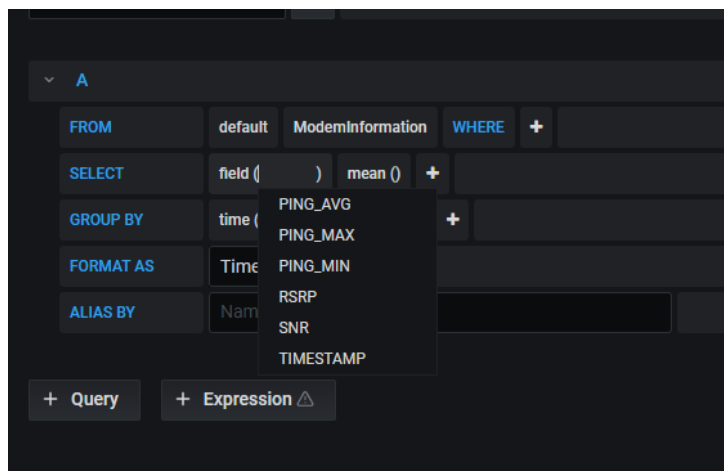


Figure 108: Field(value).

Finally, the dashboard is configured with all the modem parameters stored, as you can see in Figure 109 it is only in an interval, since the data in the files are limited, if it were a real situation, the visualization would be carried out in real time.

4.7 5G Coverage Measurements

This section carries out the following coverage measurements at Ford factory. In [3], previous thesis developed based on 5G-INDUCE project, explained a possible rethinking of the network infrastructure. Currently, the infrastructure has already been implemented, as mentioned above, pending the installation of the outdoor antenna which has not yet arrived at Ford due to time constraints.

It should be emphasized that coverage is required over the entire deployment area (approximately 23,000 m². Annex B shows the deployment and location of each the antennas already installed but not the entire deployment (outdoor antenna pending).

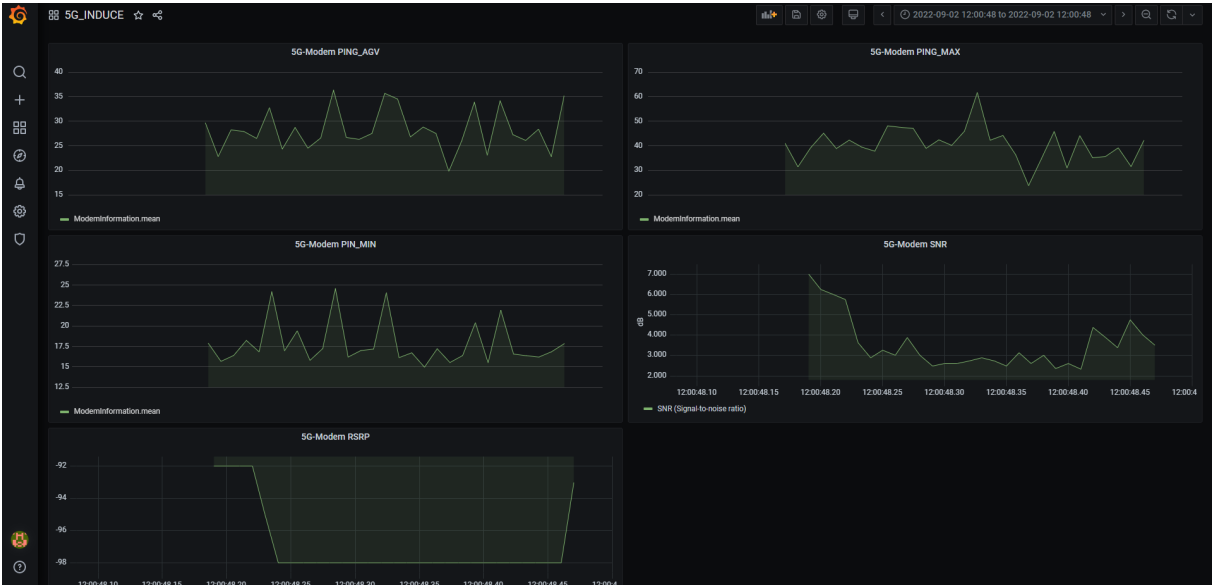


Figure 109: 5G-INDUCE Dashboard.

Coverage measurements are going to be carried out of these antennas, in particular, five 5G DOTS. The red point shows the start of the route, load point. The blue line corresponds with part of the AGV itinerary inside the motor factory. Highlighted with an orange cross is the location of the DOTS and with purple box the position of the Flight Rack.

The Flight Rack, Figure 110, it is already installed inside the motor factory. Its content is explained in detail in [3]. Due to the existing dust in the factory, the Flight Rack has been located in a cupboard to avoid malfunctions, Figure Y. In addition, the temperature inside the cupboard was very high so the solid cupboards doors have been replace with doors with holes for a better ventilation.



Figure 110: 5G Rack inside a protection cupboard at Ford factory.

The DOTs are placed on the upper part of the columns of the Ford factory, as shown in Figure 111.

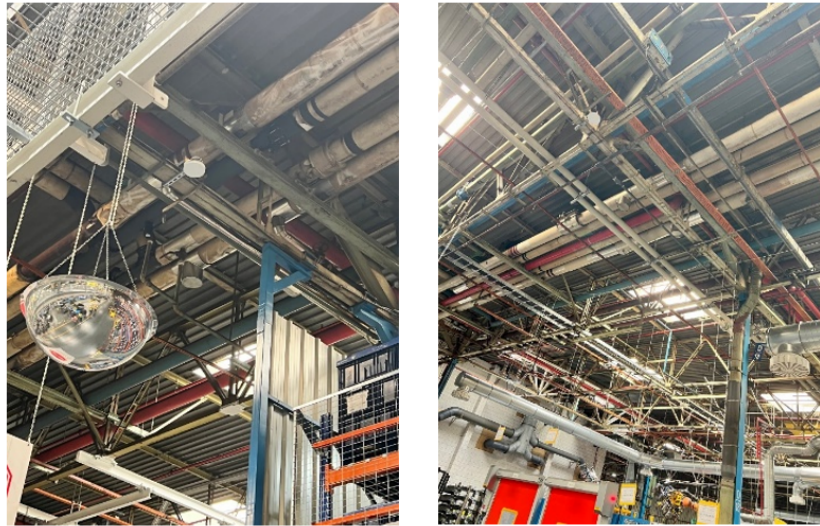


Figure 111: 5G DOTs installed at Ford factory.

4.7.1 Used Equipment

To carry out the measurements, the equipment used is a Rohde & Schwarz TSME6 5G scan which measure the Sub6-GHz bands. Connections can be checked in Figure 112, from left to right there are the GPS connection, an auxiliary one to connect a second scanner, the LAN port to connect the scanner to the computer, the top port for the receiving antenna and finally, the power supply port.



Figure 112: 5G Rohde & Schwarz scanner.

The set up used at Ford factory is shown in Figure 113 and Figure 114. An external battery has been used because measurements are made walking around the factory (AGV path). Figure X shows the receiving antenna and the LAN cable (green cable) that is connected to the computer. The computer is a Intel Core i7 and 16 GB of RAM (necessary computer requirements).



Figure 113: Coverage measurement equipment.



Figure 114: Computer for making measurements at Ford Factory.

The software to be used is SmartOne, which is Rohde & Schwarz's proprietary system, and it is used with all Rohde & Schwarz devices. For this purpose, two elements are needed. On the one hand the program license, without the license inserted in the computer you cannot measure, and you can only reproduce previously made measurements. On the other hand, if the computer does not have a physical Ethernet port, RJ45, an adapter, Figure 115, would be needed. The adapter must support the frames used by SmartOne, Jumbo frames, and therefore it must be configured.

Click on *Ajustes* $\dot{\bar{e}}$ *Red e Internet* $\dot{\bar{e}}$ *Cambiar Opciones de adaptador*. Right clicking on *Ethernet* $\dot{\bar{e}}$ *propiedades*. Click on *Configurar*, Figure 116. Select *Trama Jumbo 9014 bytes*, Figure 117.



Figure 115: SmartOne license and RJ45 adapter.

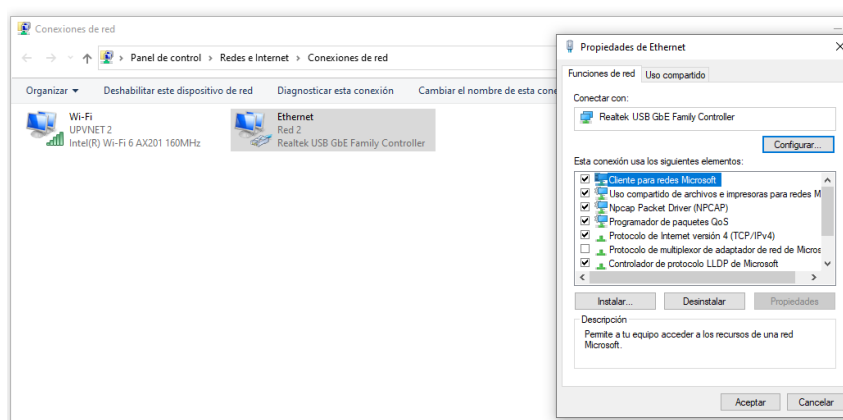


Figure 116: Configuration tab.

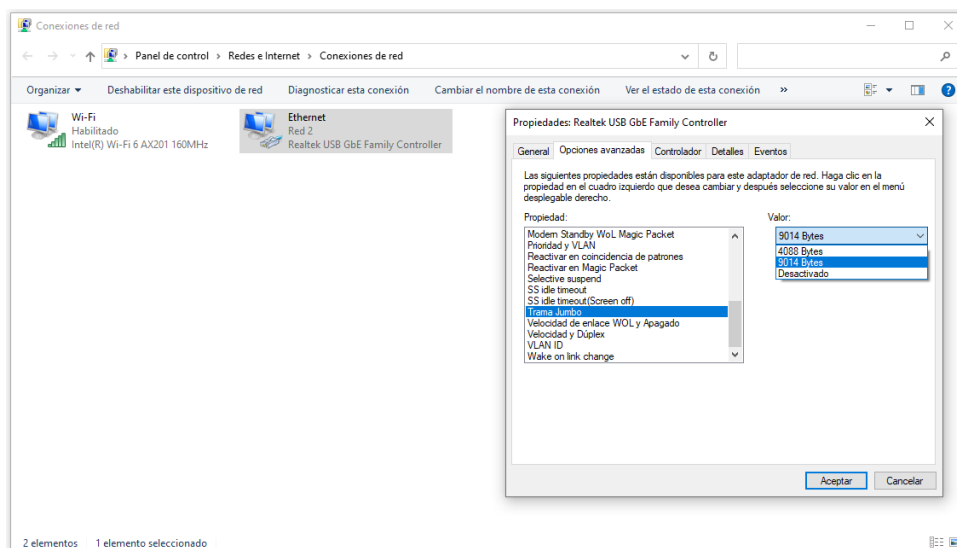


Figure 117: *Trama Jumbo* with value 9014 bytes.

4.7.2 How are measurements made?

Once the license has been obtained and it is verified that the ethernet port is receiving the information correctly, SmartOne can be opened, as it is shown in Figure 118. In this case, the GPS is not necessary because it is an indoor measurement. On the SmartOne homepage you will see the 5G and LTE scanner, this page will vary depending on the licenses available, in this case we have both.

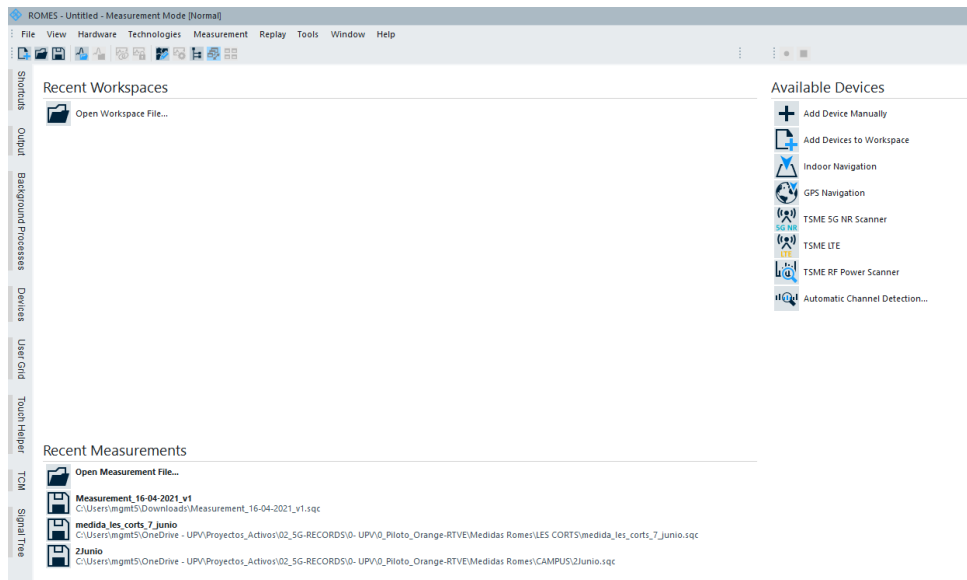


Figure 118: SmartOne Homepage.

To start the measurement click on “*Automatic channel detection*” where it is selected what is going to be measured, in this case 5G NR. Also, manually it is selected the n78 band in which the network deployment works. In addition, as the GPS is not being used it is going to be necessary to insert a plane of the site. In [REF] there is available a plane of the facilities, so it is going to be re-used. In case of not having a plane available with Paint or any design tool it could be drawn.

Once already the plane is inserted, Annex B, the next step is to establish “*Way Points*”, points are added when you click on the plane. The points of the map through which we are going to pass are established, this helps the program to interpolate the measurements between the moment that you indicate that you leave from a point and the moment that you indicate that you arrive to another one. They are distributed in time. It is recommended to make the measurements walking in a regular way, at a constant speed. Figure 119 shows the “*Way Points*”.

We are now ready to start the measurements.

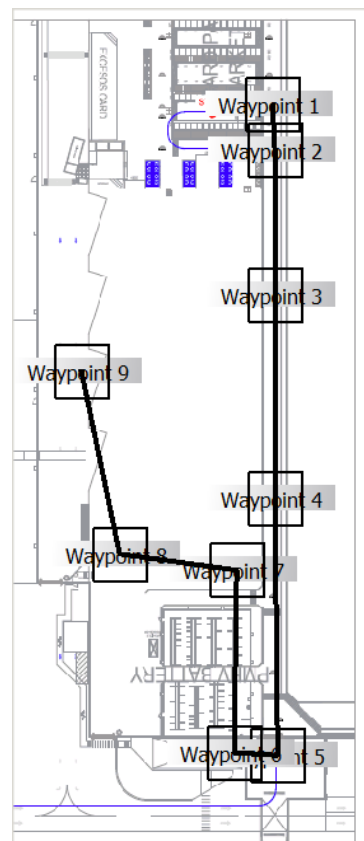


Figure 119: “Way Points”.

4.7.3 Measured parameters

Although part of the network is missing, it wants to check that so far there is coverage along the entire AGV path covered by the antennas. The parameters measured by ROMES are not configured anywhere, i.e., for each technology it has a set of parameters and all of them are always measured. Below the parameters of 5G NR are shown, Figure 120. The meaning of Top N is like a signal ranking, if we configure top 1, it gives you the signal with the best value that it considers to be the best.

The basic parameters that have been analyzed are:

- SS-RSRP (Reference Signal Received Power).
- SS-RSRQ (Reference Signal Received Quality).
- SS-SINR (Signal-to-Noise Ratio).
- PCIs (Physical Cell IDs).

According to [REF], Figure 121, shows quality intervals of three of the aforementioned parameters.

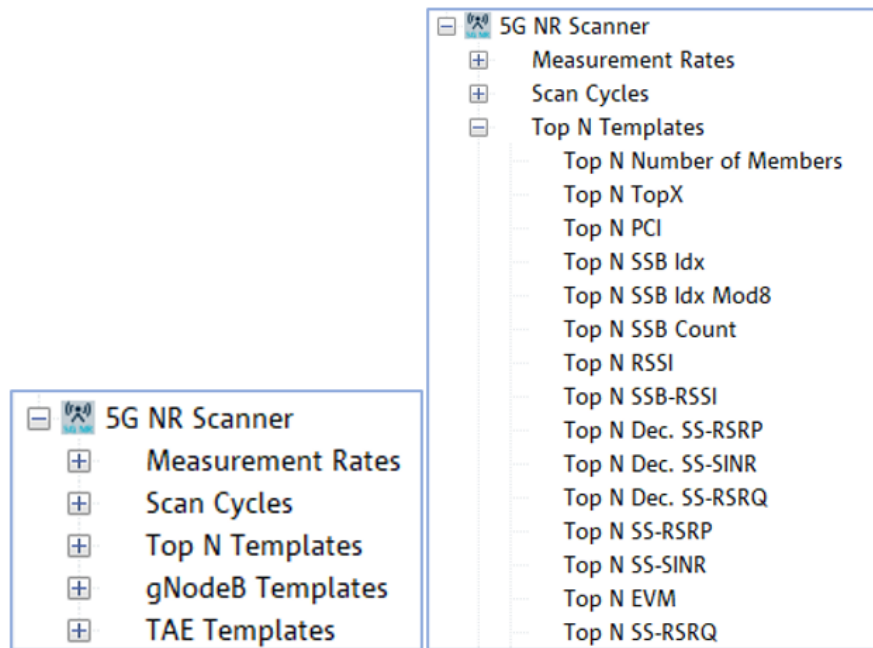


Figure 120: 5G NR parameters.

| RF Connectivity | RSRP (dBm) | RSRQ (dB) | SINR (dB) |
|-----------------|-------------|------------|-----------|
| Excellent | >=-80 | >=-10 | >=20 |
| Good | -80 to -90 | -10 to -15 | 13 to 20 |
| Medium | -90 to -100 | -15 to -20 | 0 to 13 |
| Weak | <=-100 | <-20 | <=0 |

Figure 121: Connectivity level according to signal strength.

4.7.4 Results

As a result, graphs are obtained for each of the parameters mentioned above.

SS-RSRP: There are sections where the scanner does not report an RSRP value. At first, between *Waypoint 1 and Waypoint 2*, no RSRP value is received. This is due to the fact that from the moment that you start the measurement, the scanner takes some time

to start measuring (start time). In the other sections where no RSRP value is received, it is believed that this is because the measurements were not taken at a constant speed. For safety reasons in the factory, at many times it was necessary to increase the speed of the pace. Even so, the received signal level is between -75 and -95 dBm along the entire path. Figure 122 shows the color-coded signal level received by the scanner. According to [REF] it is obtained an excellent to good quality. At the end of the trip there is a point where the RSRP value is between -95 and 110 dBm, this is probably because the equipment has been disconnected and the program takes time to stop the measurement.

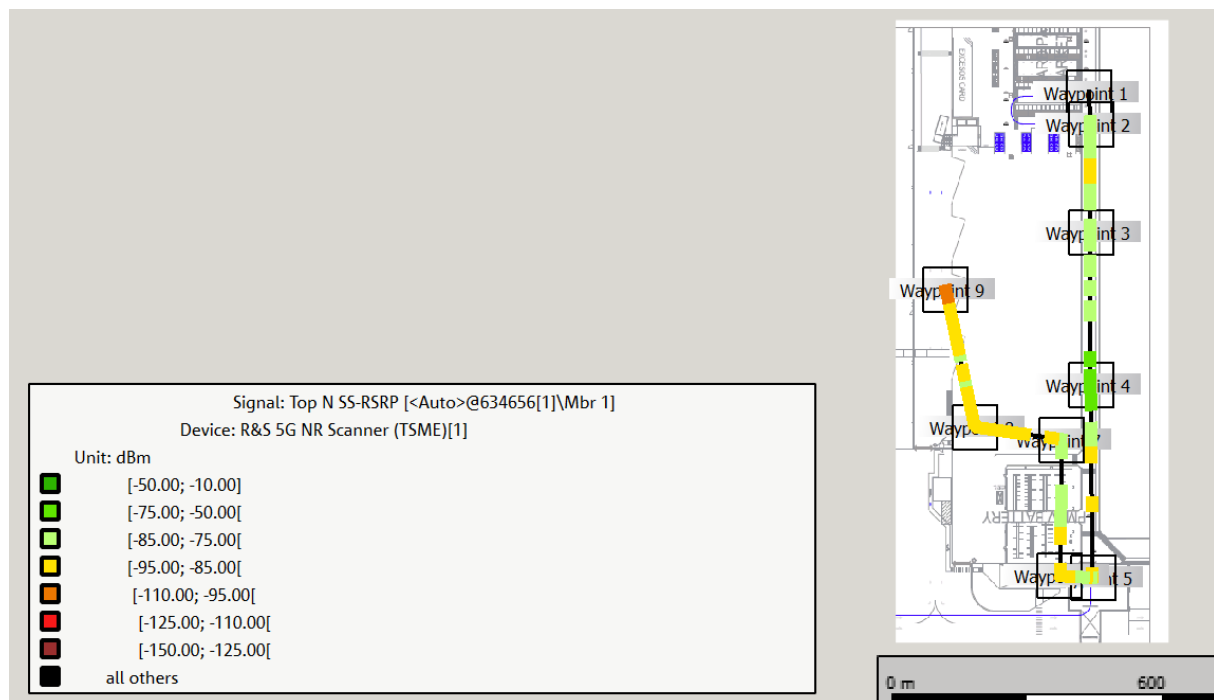


Figure 122: RSRP level measured.

SS-RSRQ: Again, In this case at the beginning of the measurement there is a section with no RSRQ level. However, signal was received on the rest of the path. The received signal level is between -15 and -10 dBm. Figure 123 shows the color-coded signal level received by the scanner. According to [REF] it is obtained a good quality along the entire path. There is a point where the RSRQ value is between -35 and -25 dBm, according to [REF] is a weak level, perhaps it is an area with more interference.

SS-SINR: In this case at the beginning of the measurement there is a section with no RSRQ level. However, signal was received on the rest of the path. SINR as an indicator is commonly used for network quality. The received signal level is between 20 and 30 dB. Figure 124 shows the color-coded signal level received by the scanner. According to [REF] it is obtained an excellent quality along the entire path. It can be checked that in the same section of the path where the RSRQ decreased in value, SINR also decreases around 10-20 dB.

PCIs: At the beginning of the measurement there is a section with no PCI. However,

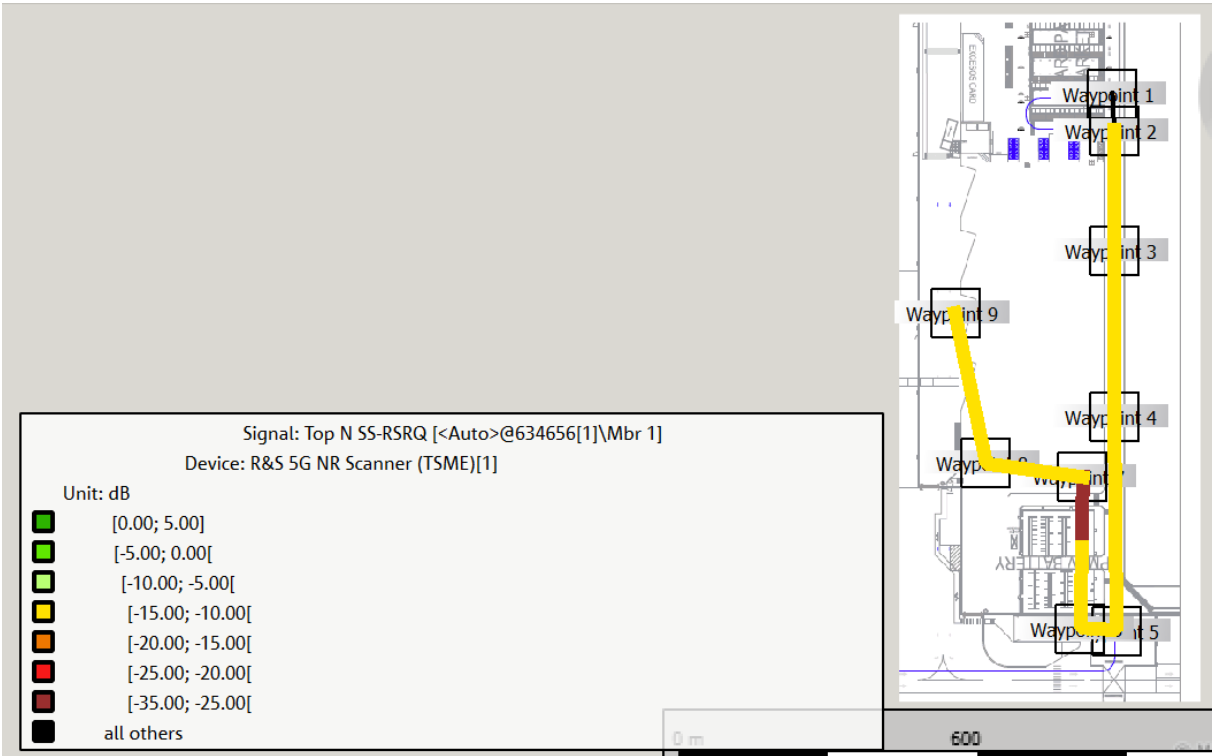


Figure 123: RSRQ level measured.

the PCI was received on the rest of the path. 317 is the identifier of the base station from which the signal is received. 5G DOTs work as if they were a single cell, for that reason they all have the same identifier, Figure 125.

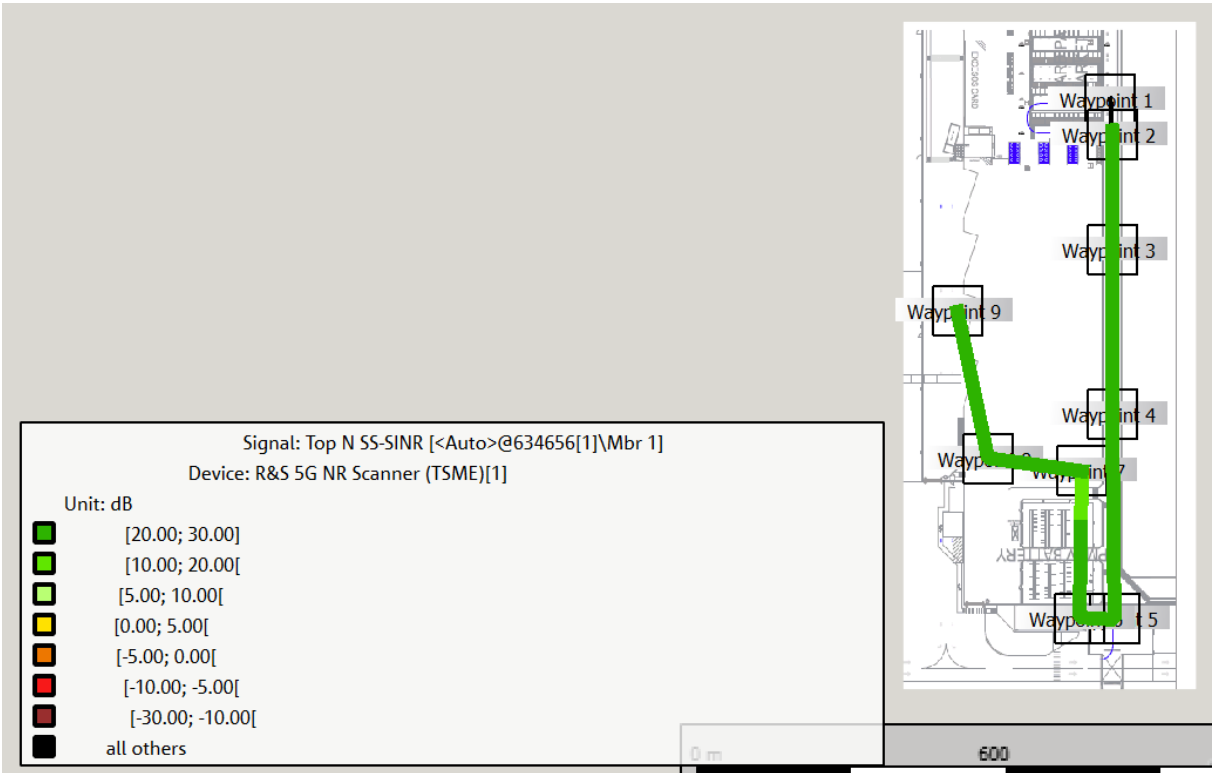


Figure 124: SINR level measured.

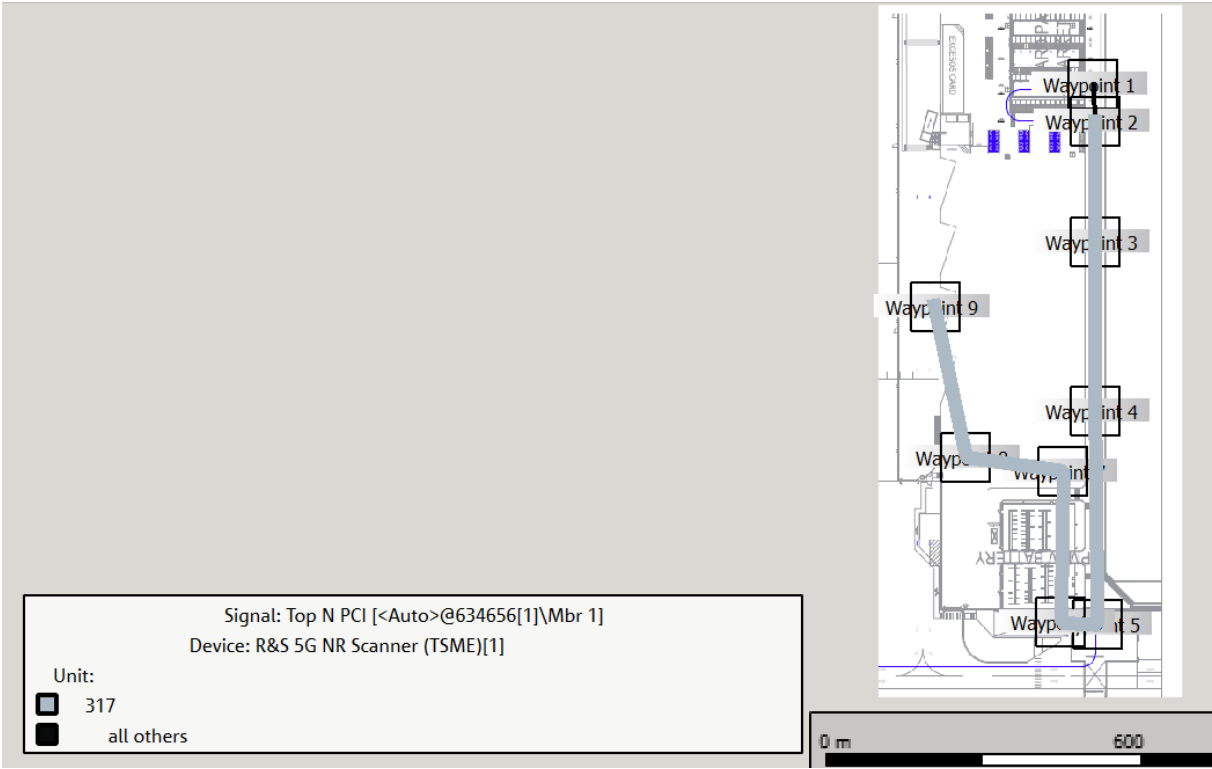


Figure 125: PCI numbers received.

SmartOne offers the possibility to visualize the parameters being measured in real time. Figure 126 shows the 5G NR scanner interface. In addition, on the top right we can see that it is indeed measuring at 3.519 GHz (5G NR).

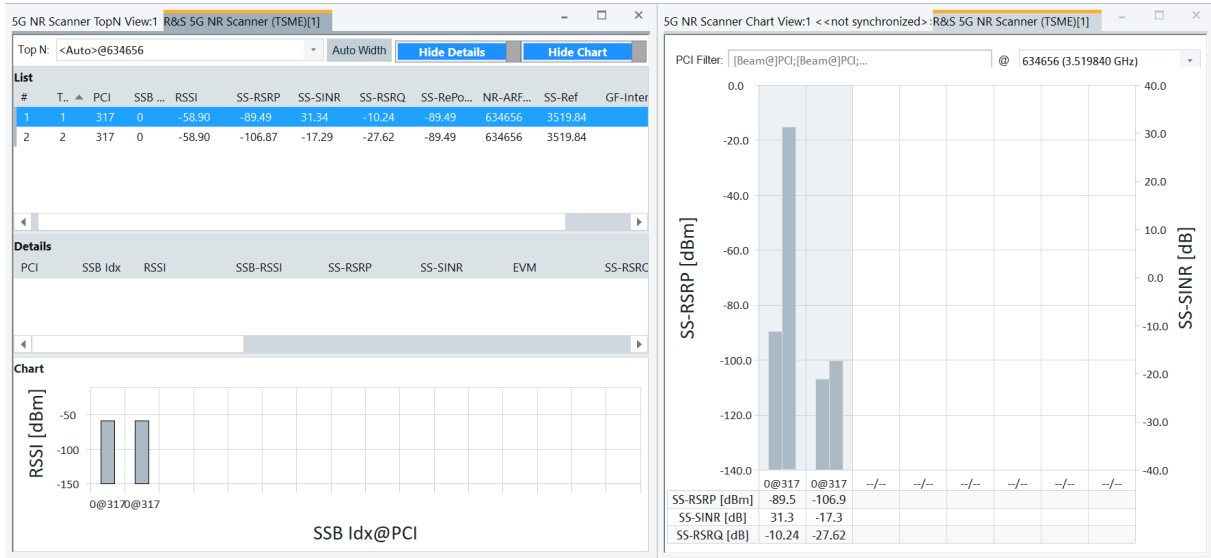


Figure 126: SmartOne 5G NR scanner interface.

5 Conclusions and Lessons Learned

As a result of the development of this work, a series of objectives defined at the beginning of this work have been achieved. Among them, the planning and design of the different phases of a project, as well as the design, development, and testing of a virtualized database. Additionally, the information used in the project has been collected and exposed in this document with the intention that the reader can draw their own conclusions and evaluate the importance of moving toward Industry 4.0.

The conclusions obtained as part of this project are presented below:

- 5G has driven the industry's global economic growth. This mobile technology increases connection speed, minimizes latency, and multiplies exponentially multiplies the number of connected devices. Combining 5G with other technologies such as digital twins, machine learning, etc.
- The VPN has been tested and has not presented any problem in the transmission even when the data size was increased the latency obtained was similar.
- It has been verified that the virtualization is a key factor in 5G deployments. Virtualization allows to reduce costs in practically all fields from the installation and configuration of equipment to the processes of backup, monitoring, management, and administration of the infrastructure. It decreases the number of physical servers needed and the percentage of disuse of the resources they have, increasing their energy efficiency. In addition, the integration of the project at European level with the database has been made possible.
- The importance of choosing a project that makes use of software tools and new technologies that you have not used before, increases the level of learning during the process: Docker, Python, MQTT, Grafana, etc.
- In terms of results obtained, the database has been created, it has been virtualized, the data management has been carried out in different phases, first the data was inserted manually, then it was automated and finally it was integrated with MQTT for the exchange of files. Additionally, a project dashboard has been created to represent the stored information.
- Through coverage measurements, it has been verified that the deployment performed so far is correct and the entire implementation area has 5G coverage.
- Finally, it should be noted that the project has been developed under the European 5G-induce project and the content of this project was one of the phases of the project to be carried out.

6 Proposals for Future work

Throughout the development of this project, ideas of interest have been arisen to progress it. The first idea is to make use of the database created by collecting data in real time and identifying the points that can be optimized in the network and in the use cases. To carry out connectivity tests but between other sites, such as Ford-5TONIC or UPV-FORD or even with different distributed architectures. The tests could also be improved by making more samples but in automated way or with other coverages tools such an iperf. Moreover, another project could be integrate the Spanish use cases and the database with the Italian orchestrator. Finally, when the network deployment at Ford is totally finished, coverage measurements of the entire deployment can be performed.

References

- [1] *Andrew Stevens, Simon Jacobson, The Pivotal Role of 5G in Manufacturing Operations, 29 July 2021* <https://www.gartner.com/doc/reprints?id=1-2AREP13N&ct=220805&st=sb>
- [2] *Qualcomm Technologies, Bosch, Bosch Rexroth, Robert Bosch Elektronik: How Ultra Reliable 5G Could Transform Manufacturing, 29 July 2021* <https://www.gsma.com/5GHub/5gmanufacturing>
- [3] *Andrea Fernandez Sierra, Distributed 5G system architecture for applications virtualization in Industry 4.0 and Edge Computing, 28 September 2021* <https://m.riunet.upv.es/handle/10251/174407>
- [4] *Cloud Native wiki, to Adopt a Containerized Architecture, 2022* <https://www.aquasec.com/cloud-native-academy/docker-container/container-advantages/>
- [5] *Influx Data Documentation* <https://docs.influxdata.com/influxdb/v1.6/tools/shell/>
- [6] *Rubén Sainz Gilarranz, Automatización de infraestructura para orquestación, gestión y monitorización de aplicaciones basadas en contenedores en el dominio IoT Edge y despliegue continuo de software, 2020* <http://msde.etsisi.upm.es/wp-content/uploads/2020/05/TFM-Automatizaci%C3%B3n-de-Infraestructura-para-Orquestaci%C3%B3n-Ruben-Sainz.pdf>
- [7] *Grafana Documentation* <https://grafana.com/docs/>
- [8] *MQTT: The Standard for IoT Messaging* <https://mqtt.org/>
- [9] *Calidades de servicio que proporciona un cliente MQTT* <https://www.ibm.com/docs/es/ibm-mq/7.5?topic=ssfksj-7-5-0-com-ibm-mm-tc-doc-tc60340--htm>
- [10] *Matthew ORiordan ,Everything You Need To Know About Publish/Subscribe, July of 2020* <https://ably.com/topic/pub-sub>
- [11] *Toshev, M. Learning RabbitMQ, 2015, 9781783984572* <https://books.google.es/books?id=ASH1CwAAQBAJ>
- [12] *Vulkan, The optimal RabbitMQ guide: from beginner to advanced, 2018, 9789163994326* <https://mqtt.org/>
- [13] *What is OpenVPN?, 2022* [https://www.allthingssecured.com/vpn/faq/what-is-openvpn/#:~:text=OpenVPN%20is%20an%20open%20source,internet%20is%20encrypted%20and%](https://www.allthingssecured.com/vpn/faq/what-is-openvpn/#:~:text=OpenVPN%20is%20an%20open%20source,internet%20is%20encrypted%20and%20)
- [14] *Cmo utilizar el comando Ping en Windows , 2019* <https://www.ionos.es/digitalguide/servidores/herramientas/comando-ping>

- [15] *PowerPing* <https://sourceforge.net/projects/powerping/>
- [16] <https://docs.docker.com/compose/install/>
- [17] <https://docs.python.org/3/tutorial/datastructures.html>
- [18] <https://pypi.org/project/paho-mqtt/>
- [19] *Json Decode error Python* <https://researchdatapod.com/how-to-solve-python-jsondecodeerror-expecting-value-line-1-column-1-char-0/>
- [20] *Store IoT Metrics with InfluxDB, 2022* <https://www.influxdata.com/blog/python-mqtt-tutorial-store-iot-metrics-influxdb/>
- [21] *into-mqtt-python-client* <http://www.steves-internet-guide.com/into-mqtt-python-client/>
- [22] *into-mqtt-python-client* <https://www.prometec.net/mqtt-influx-con-python/>
- [23] *What is the message size limit in RabbitMQ?, 2019* <https://www.cloudamqp.com/blog/what-is-the-message-size-limit-in-rabbitmq.html#:~:text=While%20the%20theoretical%20message%20size,cause%20memory%20and%20performance%20issues>
- [24] *RabbitMQ Oficial Page* <https://www.rabbitmq.com/publishers.html#message-properties>
- [25] *Python Paho Client How To Consume From Rabbitmq Existing Queue* <https://www.adoclib.com/blog/python-paho-client-how-to-consume-from-rabbitmq-existing-queue.html>

7 Publications

- 5G-Enabled AGVS for industrial and logistics environments, Andrea Fernández Sierra, Ral Lozano, Ivn Ibez, David Gomez-Barquero, Manuel Lorenzo, & Manuel Fuentes, *Revista Waves*, 2021. <https://doi.org/10.5281/zenodo.6966240>, ISSN: 1889-8297