Instituto Universitário de Lisboa

Department of Information Science and Technology

Universal Internet of Things System Powered by FIWARE

Diogo Alexandre Rodrigues Lopes

Dissertation presented in partial fulfillment of the requirements for the degree of

Master in Telecommunications and Computer Engineering

Supervisor:

Pedro Joaquim Amaro Sebastião, Assistant Professor

ISCTE-IUL

December 2018

To those who continue to live forever within our memories

Resumo

A Internet das Coisas tem crescido exponencialmente nos últimos anos e continuará a crescer por algum tempo, com cada vez mais dispositivos IoT disponíveis no mercado de consumo e específicos, havendo também cada vez mais sistemas e plataformas que utilizam e suportam estes dispositivos, fornecendo assim a possibilidade de visualizar informação por estes recolhida ou controlar os mesmos através de uma interface gráfica, que pode ser um website ou uma aplicação.

Devido á expansão do mercado da Internet das Coisas resultante de haver uma grande variedade de dispositivos e sistemas de diferentes fabricantes é difícil encontrar sistemas que sejam compatíveis com todos ou vários dispositivos de diferentes fabricantes, pois muitos utilizam protocolos de comunicação proprietários. Esta dissertação tem como objectivo o desenvolvimento de um sistema IoT universal, utilizando-se para tal a plataforma FIWARE, que foi impulsionada pela Comissão Europeia, e que permite utilizando os componentes modulares que compõem esta plataforma, desenvolver o sistema universal pretendido.

Para testar o sistema e comprovar o bom funcionamento do mesmo e de cada componente FIWARE utilizado, serão utilizados um conjunto de microcontroladores acoplados a diversos sensores e actuadores, que comunicarão com o sistema transmitindo os dados recolhidos ou recebendo comandos no caso dos actuadores.

Estas “coisas” foram utilizadas no âmbito de um caso de estudo fictício simulando uma implementação real do sistema, tendo-se conseguido com que este funcionasse correctamente, capaz de receber dados dos sensores, apresentar os mesmos quando necessário, e de controlar os actuadores.

Palavras-chave: Internet of Things; FIWARE; Microcontrolador; Sensor; Actuador

Abstract

Internet of Things has grown exponentially in recent years and will continue to grow for some time, with more and more IoT devices available in the consumer market and specific, there are also increasingly systems and platforms that use and support these devices, thus providing the possibility to view information by these collected or control them through a graphical interface, which can be a website or an application.

Due to the expansion of the Internet market of Things resulting from a wide variety of devices and systems from different manufacturers it is difficult to find systems that are compatible with all or several devices from different manufacturers, since many use proprietary communication protocols. This dissertation aims at the development of an universal IoT system using the FIWARE Platform, promoted by the European Commission, which allows the use of the modular components that make up this platform to develop the intended universal system.

A set of microcontrollers coupled to various sensors and actuators will be used to test the system and to verify the proper functioning of the same and each FIWARE component used, which will communicate with the system transmitting the collected data or receiving commands in the case of the actuators.

These "things" were used in the context of a fictional use case simulating a real implementation of the system, having been able to function properly, able to receive data from the sensors, present data when necessary, and control the actuators.

Keywords: Internet of Things; FIWARE; Microcontroller; Sensor; Actuator

Contents

List of Figures

List of Tables

Abbreviations

AC – Alternating Current

ADC – Analog-to-Digital Converter

ALG – Application-Layer Gateway

API – Application Programming Interfaces

AWS – Amazon Web Services

CKAN – Comprehensive Knowledge Archive Network

CLI – Command Line Interface

CoAP – Constrained Application Protocol

CPU – Central Processing Unit

CRM – Customer Relationship Management

CS – Chip Select

cURL – Client URL

DB – Database

DC – Direct Current

DoS – Denial-of-Service

DTLS – Datagram Transport Layer Security

EEPROM – Electrically Erasable Programmable Read-Only Memory

EN – Enable

ETSI – European Telecommunications Standards Institute

FastRTPS – Fast Real Time Publish Subscribe

FI-PPP – Future Internet Public Private Partnership

GE – Generic Enablers

GHz – Gigahertz

GND – Ground

GPIO – General Purpose Input/Output

GUI – Graphical User Interface

H2020 – Horizon 2020

HCS – Hardware Chip Select

HMISO – Hardware Master In / Slave Out

HMOSI – Hardware Master Out / Slave In

HSCLK – Hardware Serial Clock

HSPI – Hardware Serial Peripheral Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hyper Text Transfer Protocol Secure

I/O – Input / Output

I2C – Inter-Integrated Circuit

IAB – Internet Architecture Board

ICT – Information and Communications Technology

IDE – Integrated Development Environment

IETF – Internet Engineering Task Force

IoT – Internet of Things

IP – Internet Protocol

JSON – JavaScript Object Notation

LDR – Light-Dependent Resistor

LED – Light Emitting Diode

LoRaWAN – Long Range Wide Area Network

LWM2M – Lightweight Machine-to-Machine

M2M	Machine-to-Machine
MISO	Master In / Slave Out
MOSI	Master Out / Slave In
MQTT	Message Queuing Telemetry Transport
NGSI	Next Generation Services Interface
NGSI-LD	Next Generation Services Interface - Linked Data
OAuth2	Open Authorization 2.0
OneM2M	One Machine-to-Machine
OPC-UA	Open Platform Communications-Unified Architecture
OpenMTC	Open Machine Type Communication
OS	Operative System
PAP	Policy Authorization Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PLL	Phase Locked Loop
PMU	Power Management Unit
PWM	Pulse-Width Modulation
QoS	Quality of Service
RAM	Random-Access Memory
REST	Representational State Transfer
RF	Radio frequency
RFC	Request for Comments
ROM	Read-Only Memory
RS232	Recommended Standard 232

RST – Reset	
RX – Receiver	

RST – Reset

RX – Receiver

SCLK – Serial Clock

SDCLK – Secure Digital Clock

SDCMD – Secure Digital Command Line

SDD – Secure Digital Data

SDIO – Secure Digital Input Output

SMEs – Small-to-Medium Enterprises

SPI – Serial Peripheral Interface

SQL – Structured Query Language

SRAM – Static Random-Access Memory

SSH – Secure Shell

SSO – Single Sign-On

STH-Comet – Short-Term History - Comet

TCP – Transmission Control Protocol

TLS – Transport Layer Security

TOUT – Timer Output

TV – Television

TX – Transmitter

UART – Universal Asynchronous Receiver-Transmitter

UDP – User Datagram Protocol

UL2.0 – Ultralight 2.0 Protocol

URL – Uniform Resource Locator

URN – Uniform Resource Name

USB – Universal Serial Bus

VCC – Voltage Common Collector

VCO – Voltage-Controlled Oscillator

VM – Virtual Machine

WPA – Wi-Fi Protected Access

XACML – eXtensible Access Control Markup Language

YAML – YAML Ain't Markup Language

Chapter 1 – Introduction

The Internet of Things is becoming more and more popular, transition from only being know and used in the industry to the general people, which are becoming ever more dependent on the new IoT devices and services that come out almost every day. For the general people these devices can transform their way of live, making tasks easier or even provide constant health monitoring, and all appears to just work like a miracle, it is possible to connect these IoT devices to a computer or smartphone and control them from there and visualize the data collected by them. However, in reality, things are more complex than that, it is necessary to have a whole system behind these devices and, above all, they all have to be able to communicate with each other, which can be done directly or through the Internet.

To capitalize on the growth of the Internet of Things, the European Commission promoted and created FIWARE, an opensource smart solution platform, with the aim of bringing the benefits of the Internet of Things and the Internet to everyone. This is done by allowing everyone to use the FIWARE technologies to develop new smart solutions, easing the creation of new and innovative services and products before inexistent. Nowadays FIWARE is an independent foundation with a community, made of general people and enterprises, which is growing year after year making FIWARE increasingly known and adopted by new people and business.

1.1 Objectives

The main objective of this project is to develop an Universal Internet of Things System Powered by FIWARE, which as the name indicates it implies the implementation of the available FIWARE technologies and some other complementary technologies as necessary, to create an IoT System which can be used with an array of different devices, sensors and actuators.

To prove that the system is working correctly it is necessary to connect devices to it, a practical use case will be used, making testing more interesting and serving as an example of a type of application of the system.

1.2 Contributions

The main contribution of this dissertation is paving the way for whoever wants to use the FIWARE Platform in future IoT projects.

A scientific paper based on the work done was also submitted to the IEEE 5th World Forum on Internet of Things. The paper is available in Annex AA.

1.3 Document Structure

This document has the following structure:

- Chapter 1 – Introduction;
- Chapter 2 – Literature Review, where all technical-scientific knowledge which serves as the bases for this project and necessary to understand it is given;
- Chapter 3 – Universal IoT System Powered by FIWARE, where all the pieces that make the System are explained in detail;
- Chapter 4 – IoT System Tests and Results, where all the results of the experiments done to the System are presented;
- Chapter 5 – Conclusion, where the main conclusions about the project and future work that can be done are presented;
- Annexes, where extra and complementary information is presented.

Chapter 2 – Literature Review

This chapter contains all the technical-scientific knowledge collected from different sources during the investigation phase, and then edited into an easy to read and understand format with the purpose of allowing the reader to better understand the work done. All credit for the collected knowledge goes to the respective authors.

2.1 Internet of Things

Over the last years, the Internet of Things (IoT) has become an increasingly growing topic in technology, political and social spheres, becoming more and more popular year after year as a wide variety of new products based on IoT that target the public and not only the industry become available [1].

This technology is, in a simply way, the interconnection of an extensive of networked products, systems and sensors, that take advantage of the newest advancements in computing power, electronics miniaturization and network technologies, to offer new and revolutionary capabilities that were once considered impossible [1].

2.1.1 Origins

The term "Internet of Things" was first coined by Kevin Ashton, a British technology inventor, in 1999 to describe a system in which objects in the physical world could be connected to the "Internet of Sensors", currently, this term is used to designate situations in which a series of objects, devices, sensors, and ordinary items are connected to the Internet and have some computer capabilities [1].

Even though the term IoT was created in 1999, and has only recently became "famous", the truth is that the concept of interconnecting computers and networks to monitor and control devices has been around for a long time [1]. By the end of 1970, systems that relied on telephone lines to remotely monitor the power grid were already a reality [1]. In the 90s, industrial solutions for equipment monitoring and operation become widespread due to advancements in wireless technologies that allowed Machine-to-Machine (M2M) scenarios [1]. However, many of these solutions were built on dedicated and closed networks, and proprietary or industry specific standards, rather than on open Internet standards and Internet Protocol (IP) based networks [1], which increased the complexity and cost of implementation of such solutions.

In 1990, at an Internet conference, an IP-enabled toaster that could be turned on and off over the Internet was introduced [2], paving the way for other “things” being connected over the next years and originating a robust field of research and development into “smart object networking”, creating the foundations of today’s Internet of Things [1].

2.1.2 Popularity

The “Internet of Things” popularity derives from a combination of factors, resulting from the evolution and advancements made by the industry and technology [1]:

1) Low-cost, high-speed and widespread network connectivity;
2) Widespread IP-based networking;
3) Superior computing power at lower prices and better power efficiency;
4) Technology miniaturization;
5) Improvement of data analytics;
6) Growth of cloud computing.

2.1.3 The Growth of Devices and Traffic

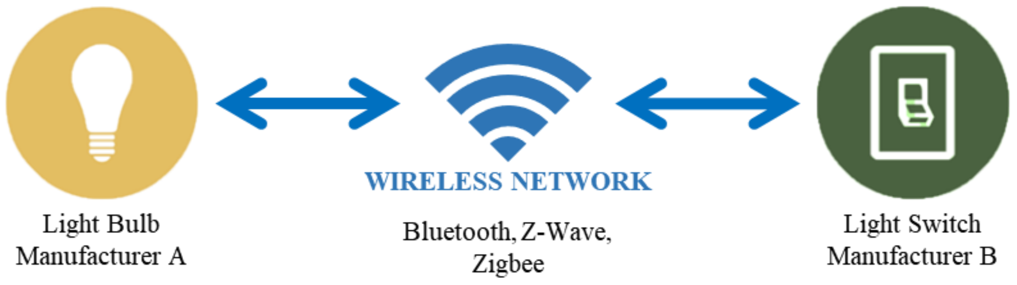
As the number of “things” connected to the Internet rises, the amount of traffic generated also rises significantly. Cisco estimates that Internet traffic generated by these devices will rise to just about 70% in 2019, also forecasting that the number of M2M connections will also rise to 43% [1]. These numbers will continue to grow as the number of smart, connected devices continues to increase, being expected to exist 500 billion of “things” connected to the Internet by 2030 [3] generating data that IoT applications use to aggregate, analyze and deliver insight, helping drive more informed decisions and actions.

2.1.4 Communication Models

In March 2015, a guiding architectural document for networking of smart objects (RFC 7452) [4] was released by the Internet Architecture Board (IAB) with the purpose of outlining the four most common communication models used by IoT devices [1] [4]:

1) **Device-to-Device:**

The device-to-device communication model represents two or more devices that directly communicate with each other, rather than through an intermediary application server. The devices can communicate over many type of networks, including IP networks, however the communication is more often established using Bluetooth, Z-Wave or ZigBee, as shown in Figure 2.1.



WIRELESS NETWORK
Light Bulb Manufacturer A
Bluetooth, Z-Wave, Zigbee
Light Switch Manufacturer B

Figure 2.1 Exemple of Device-to-Device Communication Pattern (Based on Sources: [1] [4])

This model is commonly used in home automation systems or similar, which use small data packets to communicate between devices with low data requirements, e.g. light bulbs, light switches, thermostats, door locks and some appliances.

This approach illustrates many of the interoperability challenges to be presented in Sub-Section 2.1.7. An Internet Engineering Task Force (IETF) Journal article describes, “these devices often have a direct relationship, they usually have built-in security and trust [mechanisms], but they also use device-specific data models that require redundant development efforts [by device manufacturers]” [5], meaning that manufacturers need to invest time and money to implement device-specific data formats rather than use open approaches that empower standard data formats.

On the other hand, this situation limits the user’s choice since most of the time devices of different manufacturers use different protocols that are not compatible, forcing the user to select a family of devices that employ a common protocol and/or are all of the same manufacturer. Although, the user can also benefit from knowing that products within a family tend to communicate well.

2) **Device-to-Cloud:**

In this communication model, the devices connect directly the Internet, more precisely to a cloud service like an application provider to exchange data and control traffic. This model takes advantage of already existing Ethernet or Wi-Fi networks to establish a connection between the device and the cloud, as shown in Figure 2.2.

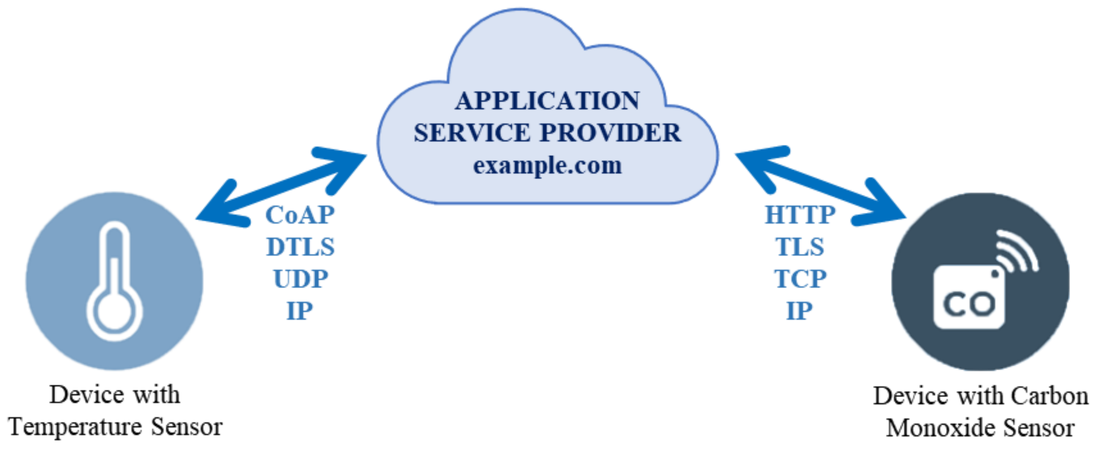

APPLICATION SERVICE PROVIDER example.com
CoAP DTLS UDP IP
HTTP TLS TCP IP
CO
Device with Temperature Sensor
Device with Carbon Monoxide Sensor

Figure 2.2 Example of Device-to-Cloud Communication Pattern (Based on Sources: [1] [4])

This model allows the device to send relevant data to a cloud database where the data can be analyzed and provide relevant information to the user. It also enables the user to obtain remote access to their device via a smartphone app or Web interface, allowing also the manufacturer to update the software/firmware of device. These or similar cases, add value to the user by extending the functionalities of the device(s) beyond its native features.

Once again, like the model before, interoperability challenges can arise when attempting to integrate devices of different vendors. Usually the devices and the cloud service are of the same manufacturers, limiting the user choice, even more if the proprietary data protocols are used to communicate to/from the cloud service. At the same time, users can generally have assurance that devices designed for the specific platform can be integrated seamlessly.

However, there are also opensource solutions like FIWARE that allow the integration of devices that use different communication protocols, by translating these protocols to one used internally by FIWARE.

3) Device-to-Gateway:

In this communication model, the device connects to an Application-Layer Gateway (ALG) as a channel to reach a cloud service. This gateway acts as an intermediary between the device and the cloud server, providing enhanced security and other functionalities such as data or protocol translation. The model is shown in Figure 2.3.

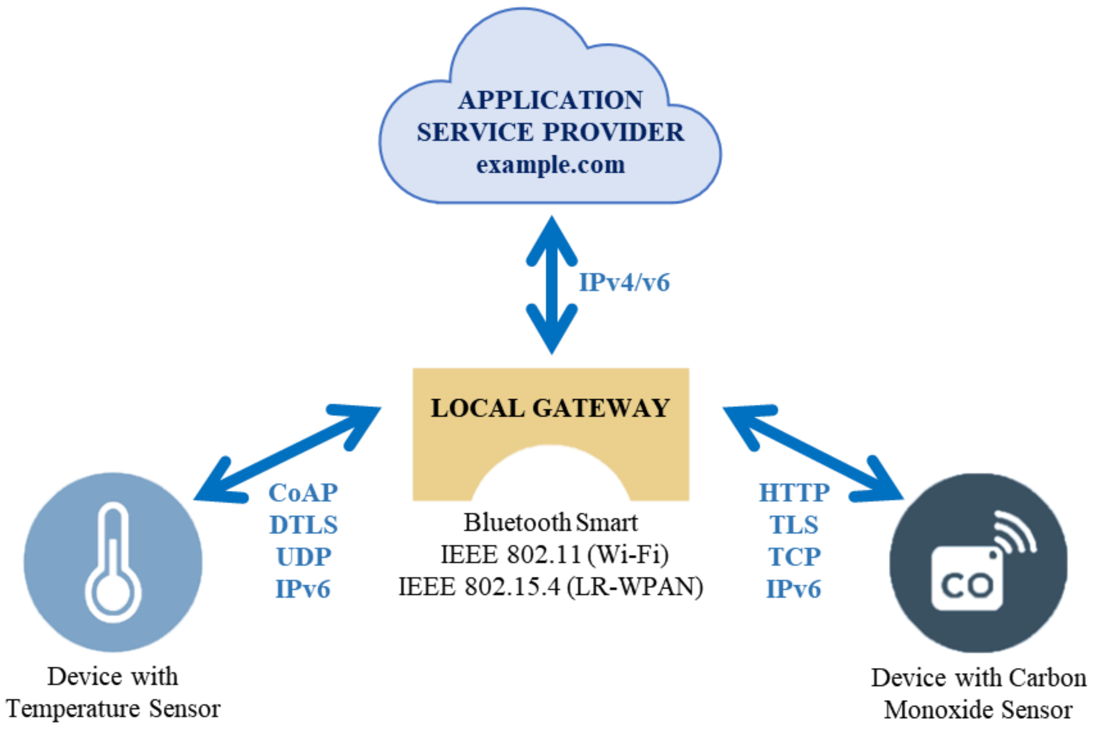


Figure 2.3 Example of Device-to-Gateway Communication Pattern (Based on Sources: [1] [4])

This model is often found in consumer devices, being the most common case, a smartphone acting as a gateway to a device, e.g., fitness band communicates with the smartphone running an app that relays the information to the cloud service.

This communication model is also usually used as a bridge when integrating new devices into a legacy system not natively interoperable with.

The downside of this approach is that the development of the gateway increases the complexity and cost of the system.

4) **Back-End Sharing:**

This communication model, denotes a communication architecture that enables users to export and analyze devices data from a cloud service in combination with data from other sources, also allowing sharing uploaded data with third parties.

This architecture allows the data collected from IoT devices to be aggregated and analyzes in the cloud, also allowing the users to move their data when switching between IoT services, breaking down the traditional data silos. This model also tries to achieve interoperability between back-end systems. A representation of this architecture is shown in Figure 2.4.

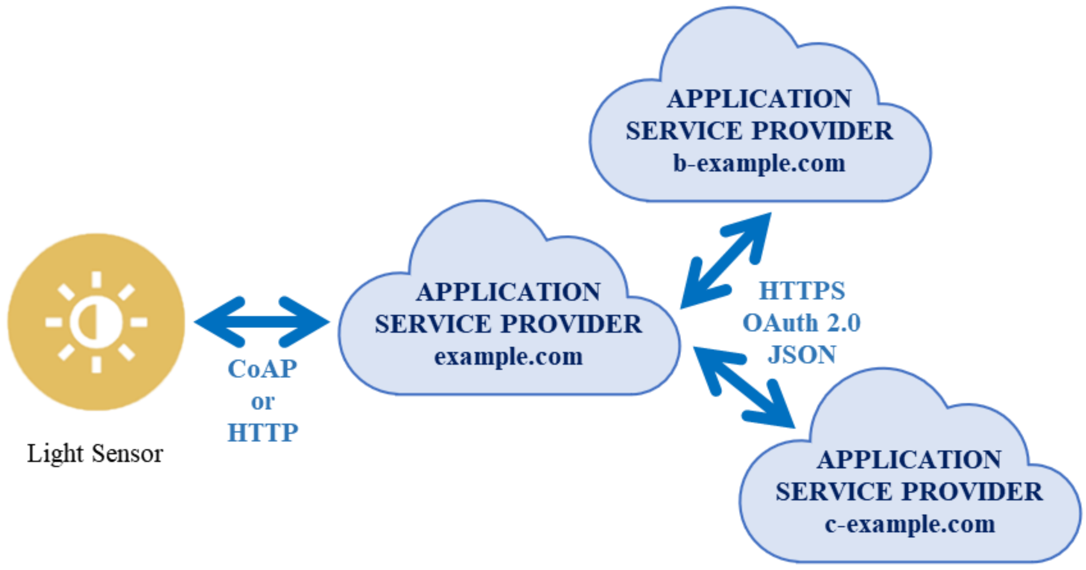

APPLICATION SERVICE PROVIDER
b-example.com
Light Sensor
CoAP
or
HTTP
APPLICATION SERVICE PROVIDER
example.com
HTTPS
OAuth 2.0
JSON
APPLICATION SERVICE PROVIDER
c-example.com

Figure 2.4 Example of Back-End Data Sharing Pattern (Based on Sources: [1] [4])

2.1.5 Security

Guaranteeing the security, reliability, resilience and stability of Internet applications and services is critical to promoting trust and use of the Internet [1].

Internet users need to have the guarantee that the Internet, its applications and devices linked to it are secure enough to use it, and the Internet of Things is no different in this aspect, as security in IoT is deeply linked to the user's capability to trust this environment [1]. If people don't believe in their connected devices and information are secure from misuse or harm, the trust and reluctance in using the Internet and its services starts to spread [1], e.g., Facebook and Cambridge Analytic scandal in 2018 [6]. Ensuring security in IoT products and services should be a top priority [1].

As the number of connected devices to Internet increases, new opportunities to exploit potential security vulnerabilities arise. Badly designed devices can expose data to theft by leaving data streams inadequately protected; failing or malfunctioning devices can also create security holes [1]. These problems are just as crucial for the devices in the Internet of Things as they are for the computers that have been the endpoints of the Internet and should be taken seriously [1]. Although due to the need of creating IoT devices with competitive cost and the technical constraints that come with it, manufacturers often don't adequately design security features into them, originating security and long-term maintainability vulnerabilities greater that their computer counterparts [1].

When combining the security design deficiencies in IoT devices with the sheer number of these devices that continues to grow from day to day, coupling that to the highly interconnected nature of such devices, every poorly secured device that is connected online affects the security and resilience of the Internet globally [1], e.g., DoS attacks that used millions of IoT devices [7].

Therefore, securing IoT devices should be considered a critical issue, has the number of essential services that depend on these devices increase [1].

2.1.6 Privacy

The Internet of Things is frequently referred to as a large network of sensor-enabled devices that collect data about the physical world, which often includes data related to people [1]. This data often provides a benefit to the device's owner, but most of the times also to the manufacturer [1]. IoT data collection and use becomes a privacy consideration when the observed individuals have different privacy outlooks about the use and scope of that data than those of the data collector [1].

Benign collection of data and combination of IoT data streams can also jeopardize people's privacy [1]. The combination and correlation of several data streams is more invasive as a detailed digital profile of an individual can be easily created, in contrast to a single IoT data stream. It becomes particularly critical when IoT devices produce additional metadata like time stamps and geolocation information, which add even more detail to the user profile [1].

There may also be situations in which users are not aware that an IoT device is collecting data about the individual and potentially sharing it with third parties [1]. An individual may be in the presence of such devices without knowing that their conversations or activities are being monitored [1], e.g., Samsung smart TVs recorded audio without the users knowing [8]. Although these features can be of benefit to an informed user, they can also pose a privacy problem for those unaware of the device's presence or have no influence on how the collected data is used [1].

The privacy concerns that come with the widespread of IoT must be addressed as they have implications on people's basic rights and the trust put onto the Internet [1].

2.1.7 Interoperability and Standards

Interoperability is the core value of the Internet, as the Internet can only work if connected systems are able to understand each other, meaning using the same protocols and encodings [1]. It is so important that the early Internet workshops for equipment vendors were called "Interops", also being the goal of the entire Internet Standards created and published by the IETF [1].

Interoperability is also the basis of the open Internet, as barriers purposely erected to impede the exchange of data can deny users the ability to connect, speak, share and innovate [1]. Environments, in which users are only allowed to use a select subset of sites and services, can considerably lessen the social, political and economic benefits of the access to the whole Internet [1].

In theory, in a fully interoperable environment, any IoT device would be able to connect and exchange data with any device or system, however, realistically interoperability is complex, as it happens in varying degrees at different layers within the protocol stack used [1]. Additionally, complete interoperability transversely is not always possible, required or wanted, and if forcefully imposed, could provide deterrents for investment and innovation [1].

Ignoring the technical aspects, interoperability has an enormous effect on the economic impact of IoT [1]. Device interoperability, if well-defined and well-functioning, encourages innovation and provides efficiencies for device manufacturers, increasing the value of the IoT market [1]. Additionally, the implementation of current standards and development of new open ones, help lessen entry barriers to the IoT world [1].

There are also some companies that see competitive and strategic advantages and, incentives in building proprietary systems and having a curated environment, however, economic opportunities may be hampered in a marketplace of silos [1].

Also, from the point of view of users of IoT devices, interoperability should be a fundamental value, as it facilitates the choice of devices with the best features at the best value and integrate them to work together [1]. Customers may hesitate to buy IoT devices and services if there is inflexibility in integrating devices, high ownership complexity, vendor lock-in or fear of deprecation due to changing standards [1].

The Figures 2.5, 2.6 and 2.7 show the vast Internet of Things world, as of 2018, divided by three main groups: applications, platforms and building blocks. These figures demonstrate why interoperability and standards are imperative.

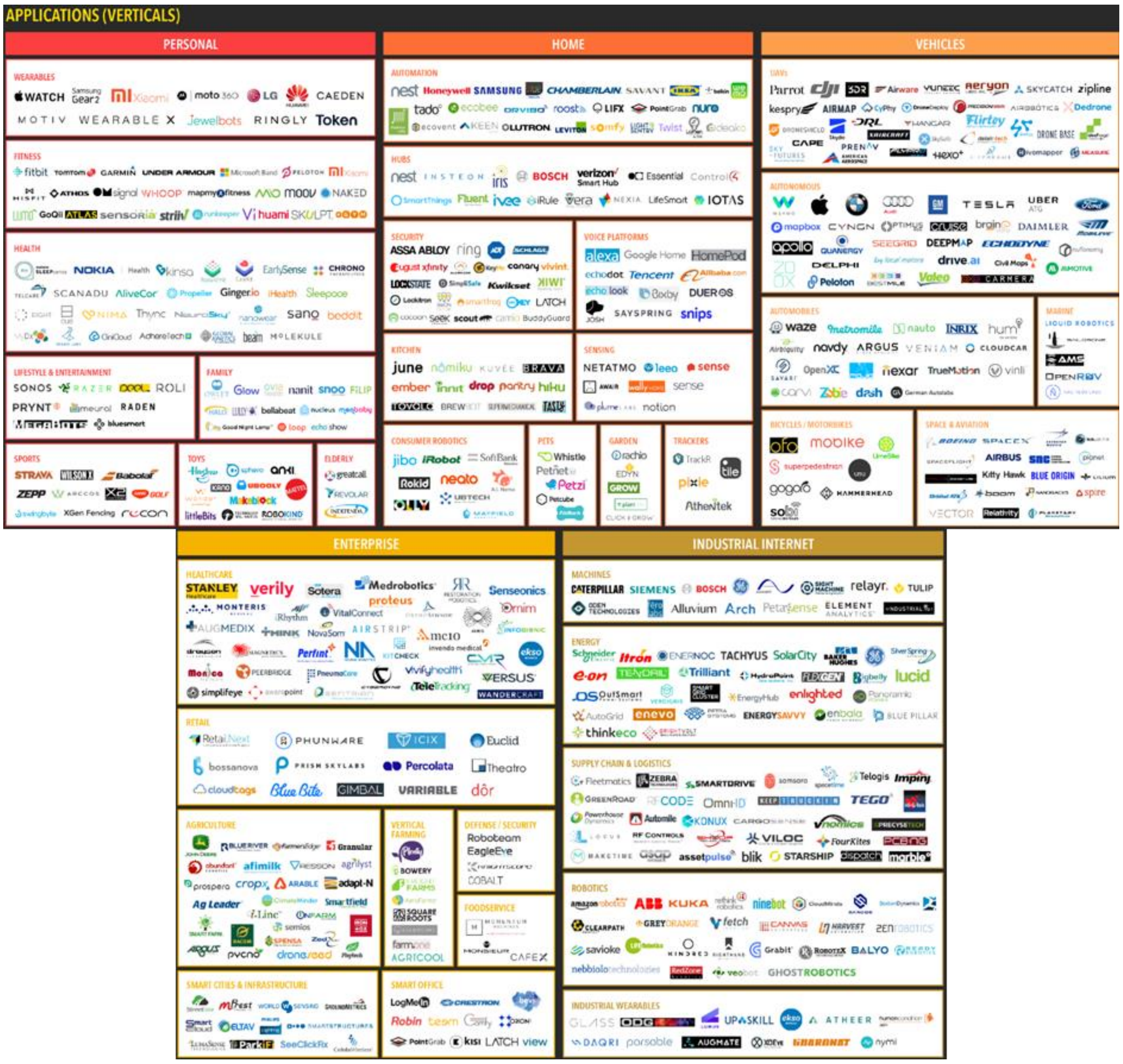


Figure 2.5 IoT 2018 Landscape - Applications (Based on Source: [9])

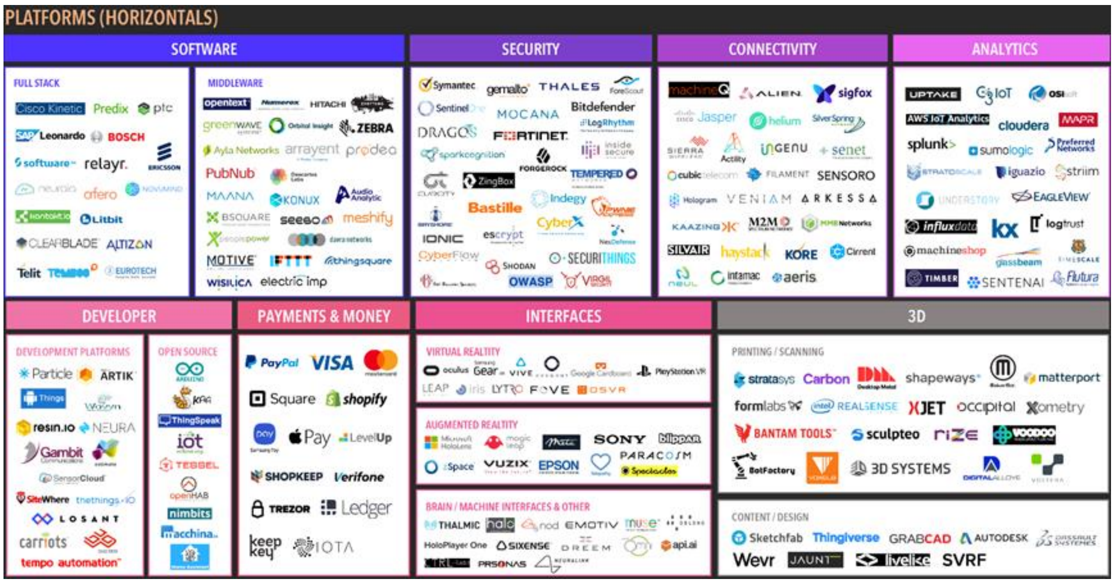


Figure 2.6 IoT 2018 Landscape - Platforms (Based on Source: [9])

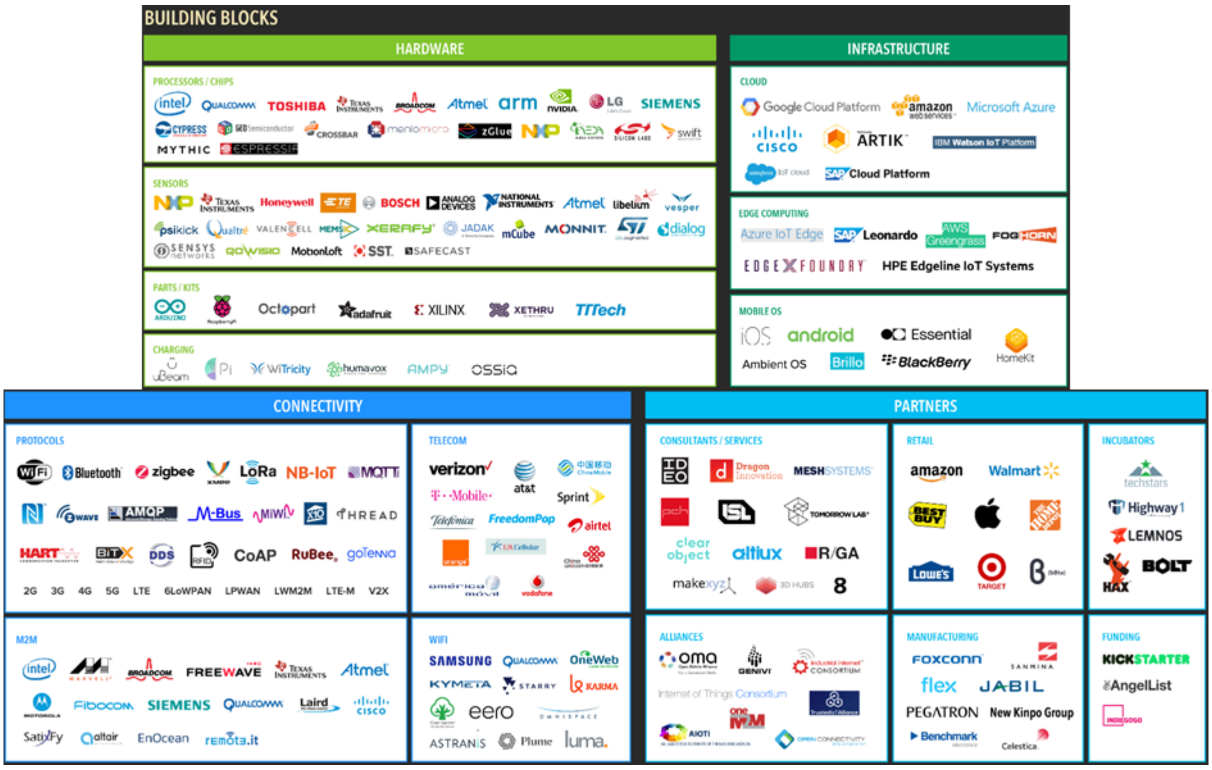


Figure 2.7 IoT 2018 Landscape - Building Blocks (Based on Source: [9])

2.2 FIWARE

In 2011, the Internet had almost two billion users, The European Commission launched a €300 million Future Internet Public Private Partnership (FI-PPP) with the objective of increasing and sharing the social and economic benefits of the future Internet with consumers, citizens, private and public sectors [10]. The FI-PPP developed FIWARE, which combined the best existing technologies to create an opensource platform of components that could be used to develop smart applications [10]. The FI-PPP also assisted entrepreneurs, startups, companies, researchers, engineers and academics using the FIWARE components from the investigation and innovation stage up to the market ready stage [10].

In 2016, five years later, the Internet had more than 3.75 billion users and hundreds of startups and dozens of municipalities in Europe were already using FIWARE to provide advanced digital services and smart apps, develop faster and at lower cost, since FIWARE avoids vendor lock-in, removes commercial and technical barriers, and is based in standard open service platform components [10].

In autumn 2016, four big companies, Atos, Engineering, Orange and Telefonica launched the FIWARE Foundation, an open body within the FIWARE Community, with the intent of promoting, augmenting, protecting and validating the FIWARE brand and its technologies (FIWARE Platform) [10] [11]. The founding members were soon joined by others, (e.g., companies, cities, institutions and individual contributors who wanted to support FIWARE Community [10] [11]). The Foundation is financed by its members and funds received for participation in several H2020 projects [10].

Currently the FIWARE Foundation has become the main interlocutor between the opensource developer community, the industry and the end users in different vertical business [10].

2.2.1 What is FIWARE

FIWARE can be easily defined as the opensource smart solution platform of choice [12].

The FIWARE Community is an independent open community devoted to the FIWARE mission: “to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors” [12]. The Community has as founding principles: “independence in decision making, openness, transparency and meritocracy” [12].

The Community is formed by contributors to the FIWARE Platform and by those who contribute in building and making the FIWARE ecosystem sustainable, committing resources in FIWARE Lab activities or activities of the FIWARE Accelerator, FIWARE Mundus or FIWARE iHubs programs [12].

The FIWARE Platform provides a set of public and royalty-free Application Programming Interfaces (APIs) that facilitate the development of smart applications in vertical sectors. In addition, an opensource reference implementation of every FIWARE component is also freely available [12].

The FIWARE Community is structured in such a way that encourages all forms of contributions and provides safeguards in case the balance between the members of the community is lost [12]. The Community is organized in three teams: FIWARE Chapters, Technical Committees, responsible for activities of technical nature, and Ecosystem

Support Committees, responsible for non-technical activities related to the FIWARE Accelerator, FIWARE Mundus and FIWARE iHubs programs [12].

2.2.1.1 FIWARE Lab

FIWARE Lab is a non-commercial sandbox environment where members of the FIWARE Community can research, experiment and test the FIWARE technologies as well as their applications, making use of Open Data published by cities and other organizations [13]. The Lab is set up over a geographically distributed network of federated nodes leveraging on an ample variety of experimental infrastructures [13]. It is important to note that resources are limited for trial members and unlimited for members that have an approved project by FIWARE.

2.2.1.2 FIWARE Accelerate

The FIWARE Accelerator Program has the objective of incentivizing the use of FIWARE technologies amongst solution integrators and application developers, with special emphasis on Small-to-Medium Enterprises (SMEs) and start-ups [13].

2.2.1.3 FIWARE Mundus

Even thought FIWARE was created in Europe, it was designed from the start with the objective of going global. The FIWARE Mundus program exists in order win over local Information and Communications Technology (ICT) players and domain stakeholders into using FIWARE, ultimately cooperating with local governments in diverse parts of the world [13].

2.2.1.4 FIWARE iHubs

The FIWARE iHubs Program has the objective of supporting the formation and the operations of iHubs nodes worldwide, eventually creating a network of iHubs that will play an important role in building the community of developers adopting and contributing to FIWARE [13].

2.2.2 Powered by FIWARE

The FIWARE Platform is a curated framework of opensource components, which can be combined with other third-party platform components to hasten the development of smart solutions [14].

In every smart solution it is essential to gather and manage context information, process it and inform external actors, allowing them to actuate and so change or enrich the current context. The FIWARE Context Broker component is the core constituent of any "Powered by FIWARE" solution, as it enables the system to update and access the current state of context, as depicted in Figure 2.8 [14].

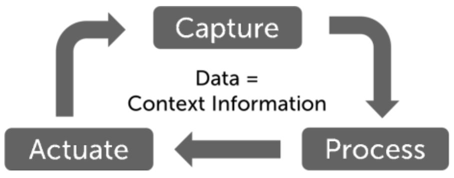


Figure 2.8 Context Broker Processes (Source: [14])

As the core, the Context Broker is in turn surrounded by additional components, which can supply context data from various sources (e.g., a Customer Relationship Management (CRM) system, social networks, mobile apps, IoT sensors), support to data processing, analysis and visualization, or adding support to data access control, publication or monetization [14].

The Figure 2.9 shows an example of a FIWARE reference architecture where it is possible to visualize the Context Broker surrounded by other components that together form a system.

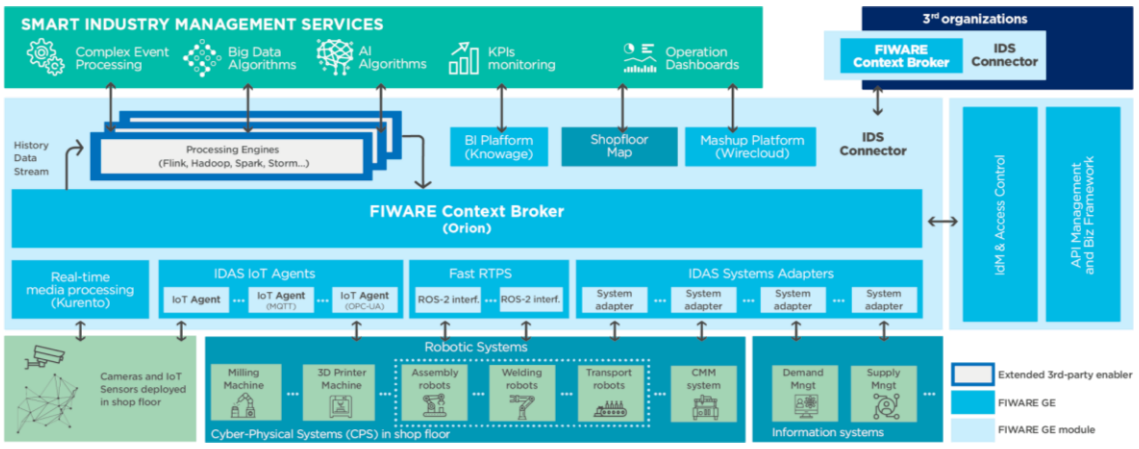


Figure 2.9 Example of a FIWARE Reference Architecture for Smart Industry (Source: [15])

All communications between applications (frontend) or platform components and the Context Broker (together form the backend) are done with the use of the FIWARE NGSIv2 RESTful API [16], a simple and powerful open standard that in the future will align with the ETSI NGSI-LD [17] specifications that are based and an evolution of the former, and are currently available for public review [14].

The open standard characteristic of the FIWARE NGSI API allows developers to port their applications across different "Powered by FIWARE" platforms and a guarantee of a stable framework for future development [14]. Also, additional functionalities can easily be added to a solution by using FIWARE or third-party components that comply to the FIWARE NGSI API. Since all components comply to the same API, integration is simplified as all components use the same standard interface, eliminating vendor lock-in [14]. The use of FIWARE also allows for rearchitecting solutions according to the user or business needs, as all FIWARE architectures are modular due to being made up of independent components [14].

2.2.3 FIWARE Platform Components

As said before, the FIWARE Platform is a curated framework of opensource components, these components are named as Generic Enablers (GEs), and can be assembled together and with other third-party components to build smart solutions [18].

Building around the FIWARE Orion Context Broker Generic Enabler, the core and only mandatory GE of any "Powered by FIWARE" solution, a rich collection of complementary FIWARE GEs are available, dealing with [18]:

- Interfacing with the IoT, Robots and third-party systems;
- Context Data/API management, publication and monetization;
- Processing, analysis and visualization of context information.

All of the available FIWARE GEs can be seen in Figure 2.10, although some are still in incubation.

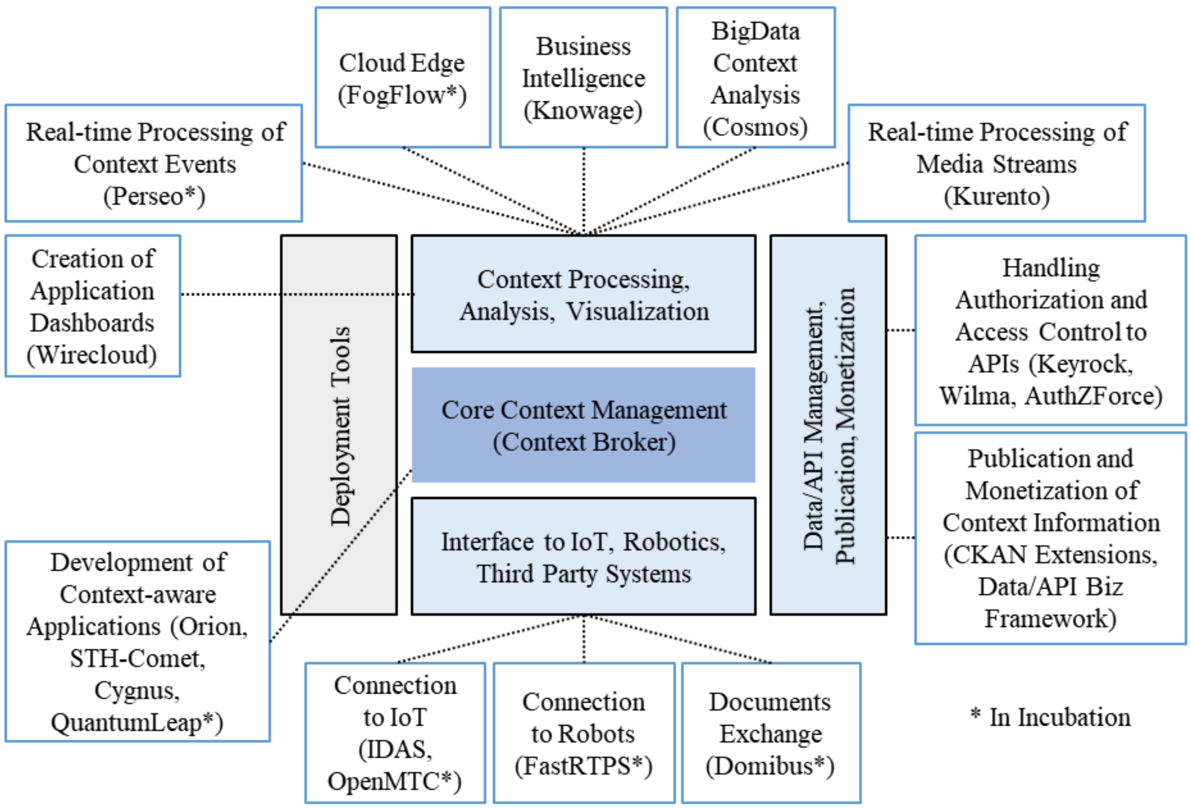

Cloud Edge (FogFlow*)
Business Intelligence (Knowage)
BigData Context Analysis (Cosmos)
Real-time Processing of Context Events (Perseo*)
Real-time Processing of Media Streams (Kurento)
Creation of Application Dashboards (Wirecloud)
Deployment Tools
Context Processing, Analysis, Visualization
Core Context Management (Context Broker)
Interface to IoT, Robotics, Third Party Systems
Data/API Management, Publication, Monetization
Handling Authorization and Access Control to APIs (Keyrock, Wilma, AuthZForce)
Publication and Monetization of Context Information (CKAN Extensions, Data/API Biz Framework)
Development of Context-aware Applications (Orion, STH-Comet, Cygnus, QuantumLeap*)
Connection to IoT (IDAS, OpenMTC*)
Connection to Robots (FastRTPS*)
Documents Exchange (Domibus*)
* In Incubation

Figure 2.10 FIWARE Generic Enablers (Based on Sources: [18] [19])

Contrary to the FIWARE Orion Context Broker, the use of complementary FIWARE GEs is not obligatory, as it is possible to develop a “Powered by FIWARE” solution with the Orion GE and third-party components [18].

2.2.3.1 Core Context Management

The Orion Context Broker, as the core component, allows the management of context information in a highly decentralized and large-scale manner. It also, provides the FIWARE NGSIv2 RESTful API, enabling updates, queries or subscriptions to changes on context information [18]. This GE only holds the latest information about the current context, however, as context information changes over time it is important to save this context history [18]. For that the following GEs, as part of the Core Context Management Chapter, complement the Orion Context Broker [18]:

- The STH Comet Generic Enabler enables storing a Short-Term History of context data (typically months) on MongoDB;
- The Cygnus Generic Enabler enables managing the history of context, created as a stream of data which can be injected into several data sinks, including some of

the most popular databases like PostgreSQL, MySQL, MongoDB or AWS DynamoDB as well as BigData platforms like Hadoop, Storm, Spark or Flink.

2.2.3.2 Interface to IoT, Robots and Third-Party Systems

Several GEs are available to facilitate the connection with the Internet of Things, Robots and third-party systems for the purpose of collecting context information or trigger actuations in response to context updates [18]:

- The Backend Device Management - IDAS Generic Enabler offers a wide range of IoT Agents which make it easier to interface with devices using the most widely used IoT protocols:
    - Lightweight Machine-to-Machine (LWM2M) over Constrained Application Protocol (CoAP);
    - JavaScript Object Notation (JSON) over Hypertext Transfer Protocol/ Message Queuing Telemetry Transport (HTTP/MQTT);
    - Ultralight 2.0 over HTTP/MQTT;
    - Open Platform Communications-Unified Architecture (OPC-UA);
    - Long Range Wide Area Network (LoRaWAN).

  The tools to develop custom IoT Agents for specific protocols are also available for developers.

The following Generic Enablers are, at the date of this document, under incubation within this chapter [18]:

- The Fast Real Time Publish Subscribe (FastRTPS) Incubated Generic Enabler helps to interface with robotics systems, having been adopted as the default middleware in ROS2 (Robot Operating System 2.0);
- The Open Machine Type Communication (OpenMTC) Incubated Generic Enabler is an open source implementation of the One Machine-to-Machine (OneM2M) standard. A northbound interface with the Orion Context Broker is already implemented in this GE.

2.2.3.3 Processing, Analysis and Visualization of Context Information

Various GEs are available, to ease the processing, analyzing or visualizing of context information [18]:

- The Application Mashup - Wirecloud Generic Enabler, a powerful web mashup platform which makes it easier to develop operational dashboards highly customizable by end users;
- The Data Visualization - Knowage Generic Enabler, the implementation of Knowage, a powerful Business Intelligence platform empowering business analytics and analytics on context data [20];
- The Stream-oriented - Kurento Generic Enabler enables real-time processing of media streams supporting the use of video cameras as sensors, and the integration of advanced application functions, e.g., integrated audiovisual communications, augmented reality, flexible media playing, recording;
- The BigData Analysis - Cosmos Generic Enabler enables an easy Bigdata analysis over context information.

The following Generic Enablers are, at the date of this document, under incubation [18]:

- The FogFlow Incubated Generic Enabler is a distributed execution framework to orchestrate dynamic processing flows over cloud and edges;
- The Cloud Messaging - AEON Incubated Generic Enabler provides a communication channel middleware for the fast distribution of messages among entities;
- The Electronic Data Exchange - Domibus Incubated Generic Enabler enables the exchange of electronic data and documents in a reliable and trusted way.

2.2.3.4 Context Data/API Management, Publication and Monetization

The implementation of secure access to the components of a solution architecture is done using the following Generic Enablers [18]:

- The Identity Management - Keyrock Generic Enabler provides secure and private OAuth2 authentication of users and devices, management of user profiles, safekeeping of personal data, Single Sign-On (SSO) and Identity Federation over several administration domains;

- The PEP-Proxy - Wilma Generic Enabler enables proxy functions within OAuth2 authentication schemas and applies Policy Enforcement Point (PEP) functions within an eXtensible Access Control Markup Language (XACML) schema;
- The Authorization Policy Decision Point (PDP) – AuthZForce Generic Enabler enforces Policy Decision Point/Policy Authorization Point (PDP/PAP) functions within an access XACML schema.

Generic Enablers for the publication and monetization of context data resources [18]:

- The Comprehensive Knowledge Archive Network (CKAN) extensions Generic Enabler provides several add-ons permitting to extend current capabilities of the CKAN Open Data publication platform, allowing the publication of datasets matching right-time context data, the assignment of access terms and policies to those datasets and the assignment of pricing and pay-per-use schemas to datasets;
- The Business API Ecosystem - Biz Framework Generic Enabler provides backend support for Context API/Data monetization built on TM-Forum Business APIs.

2.3 Used IoT Protocols

As mentioned before the Backend Device Management - IDAS Generic Enabler offers a wide range of IoT Agents that are responsible for translating the different IoT protocols used by connected devices to the NGSIv2 API, the only language known by the other FIWARE GEs. Of all the protocols IoT Agents available (each responsible for a different protocol), only two were considered for this project: IoT Agent for Ultralight 2.0 over HTTP/MQTT and IoT Agent for LoRa.

2.3.1 Ultralight 2.0 Protocol

Ultralight 2.0 is a lightweight text-based protocol designed for constrained devices and communications, whose bandwidth and memory may be limited [21].

2.3.1.1 Measure Payload Syntax

The payload send from the devices is composed of a list of key-values pairs separated by the "|" character, as shown in the example below [21].

```
t|30|h|60
```

In this example, two attributes are present, one named “t” with the value of “30” and another named “h” with the value of “60”. It is also possible to send characters instead of numbers [21].

It is also possible for the device to send a payload with a timestamp, but it is normally not done as the IoT Agent can add a timestamp to the received messages, reducing the size of the messages [21] [22].

```
2018-08-10T00:35:30Z|t|30|h|60
```

The attributes in the messages received are then mapped by the system to the correct entities attributes, later explained and exemplified.

2.3.1.2 Commands Syntax

Commands are messages sent from the IoT Agent to devices, following the format below [21].

```
<device name>@<command name>|<command value>
```

Example, in which a Robot is commanded to turn right:

```
Robot1@turn|right
```

In case a command requires parameters, the “command value” can be used as exemplified [21]:

```
Robot1@turn|param1:1|param2:2
```

Since the character “=” is a forbidden character, “:” is used instead, otherwise the command will fail and an error message will be returned [21].

After receiving commands, the devices shall respond following the format underneath [22].

```
<device name>@<command name>|<result>
```

Example, of a reply:

```
Robot1@turn|Right Ok
```

The Ultralight 2.0 protocol defines the measurements and commands syntax; however, it does not specify the transport protocol [21]. The IoT Agent currently supports two transport protocol bindings: HTTP and MQTT [21]; of the two the MQTT binding was chosen due to the existence of a tutorial describing how to configure the IoT Agent to use MQTT, and due to HTPP not being the ideal protocol for IoT [22].

2.3.2 MQTT

Message Queuing Telemetry Transport (MQTT) is a publish-subscribe messaging protocol used in the Internet of Things, where a “small code footprint” is required and the network bandwidth is limited [22]. This protocol has the key characteristic of being bandwidth and power efficient [22].

The Ultralight 2.0 protocol can be carried over HTTP or MQTT [21]. HTTP uses a request/response model where the devices connect directly to the IoT Agent and therefore simplifies the system architecture when compared to MQTT which uses a publish subscribe model that is event driven, publishes messages to clients and requires a central communication point, known as the MQTT Broker, which is responsible for dispatching all messages between the senders and the correct receivers, effectively working as a router [22].

Table 2.1 HTTP vs MQTT [22]

HTTP vs MQTT	
IoT Agent communicates directly with the devices (things)	IoT Agent communicates indirectly with the devices, via an MQTT Broker
Request-Response model	Publish-Subscribe model
Devices must be always ready to receive messages	Devices choose when to receive messages
High power requirement	Low power requirement

Every message published must include a topic, that is essentially the routing information for the broker [22]. To receive messages, a client must inform the broker that it wants to subscribe to a Topic, informing the broker to deliver all messages with the subscribed topic to the client [22].

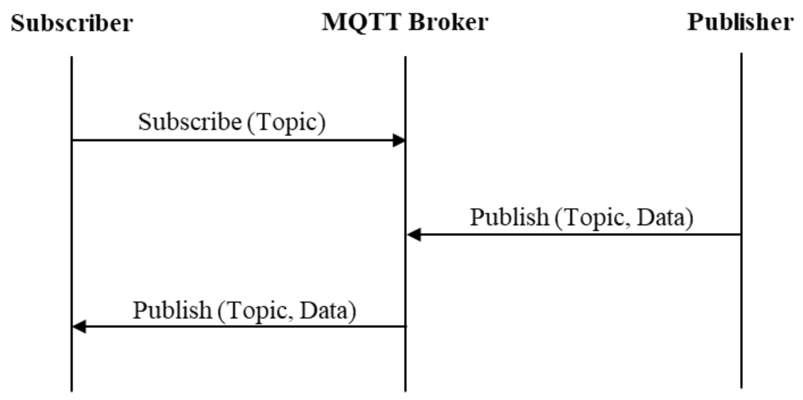


Figure 2.11 MQTT Publish-Subscribe Model

As the clients only communicate over the Topic and don't have to know each other, it allows for highly scalable solutions without dependencies between subscribers and publishers, as the only shared point of communication is the MQTT Broker [22].

MQTT also the publisher to select a Quality of Service (QoS) level, thus increasing the reliability of the communications. It is possible to select between tree QoS levels [23]:

- QoS 0 – at most once, which guarantees a best effort delivery;
- QoS 1 – at least once, the message is delivered at least one time;
- QoS 2 – exactly once, the message is delivered exactly one time.

The MQTT Broker used in this project is the Mosquitto MQTT Broker, a readily available and opensource broker.

In Figure 2.12 and 2.13 is possible to compare how the IoT Devices are connected when using HTTP and MQTT.

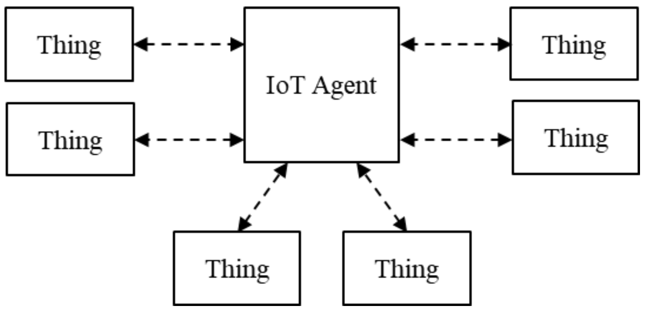


Figure 2.12 Ultralight over HTTP (Based on Source: [22])

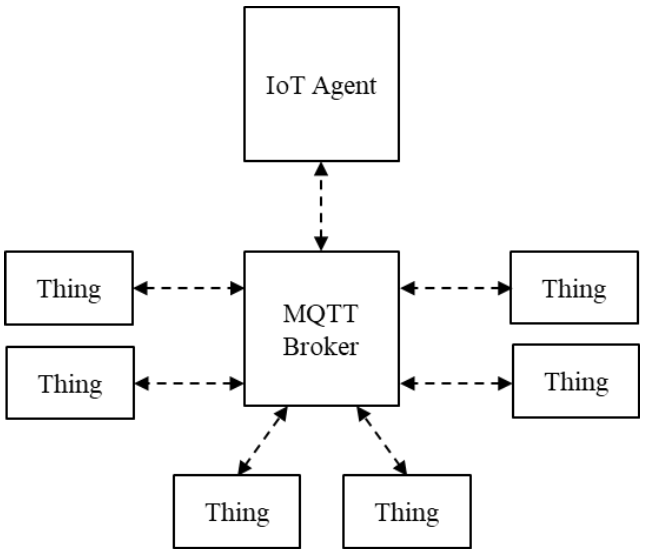


Figure 2.13 Ultralight over MQTT (Based on Source: [22])

2.4 Docker

Docker is an open platform for developing, shipping, and running applications through the use of environments called containers [24]. The isolation and security offered by containers allow to simultaneously run several containers on a given host [24]. Since containers are lightweight they don’t require the use of a hypervisor, running directly in the host kernel, meaning it is possible to run more containers on a given hardware combination than on virtual machines (VM) [24]. Although it is also possible to run containers on hosts that are virtual machines [24]. Figure 2.14 compares a container with a virtual machine, illustrating the differences between them.

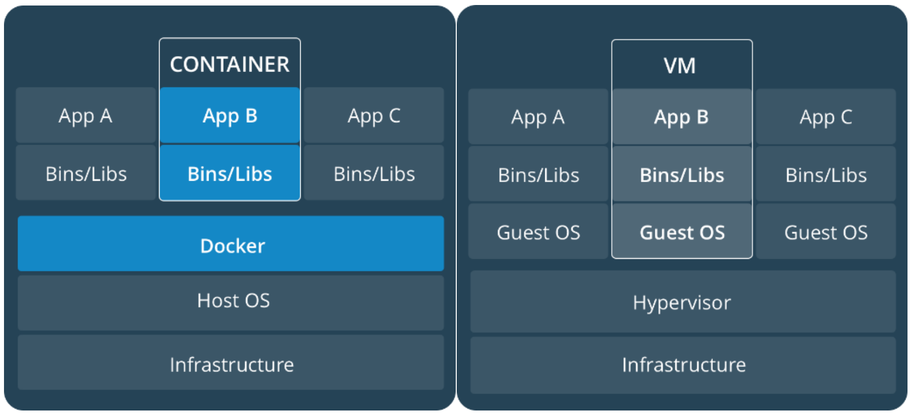


Figure 2.14 Containers vs Virtual Machines (Source: [25])

2.4.1 Docker Engine

The Docker Engine is a client-server application constituted by three key components, as shown in Figure 2.15 [24]:

- A Server called a Daemon process, responsible for creating and managing Docker objects: images, containers, networks and volumes;
- A REST API, specifies the interfaces that programs can use to communicate with de daemon and direct it;
- A Command Line Interface (CLI) Client, that uses the REST API in to control and interact with the Daemon.

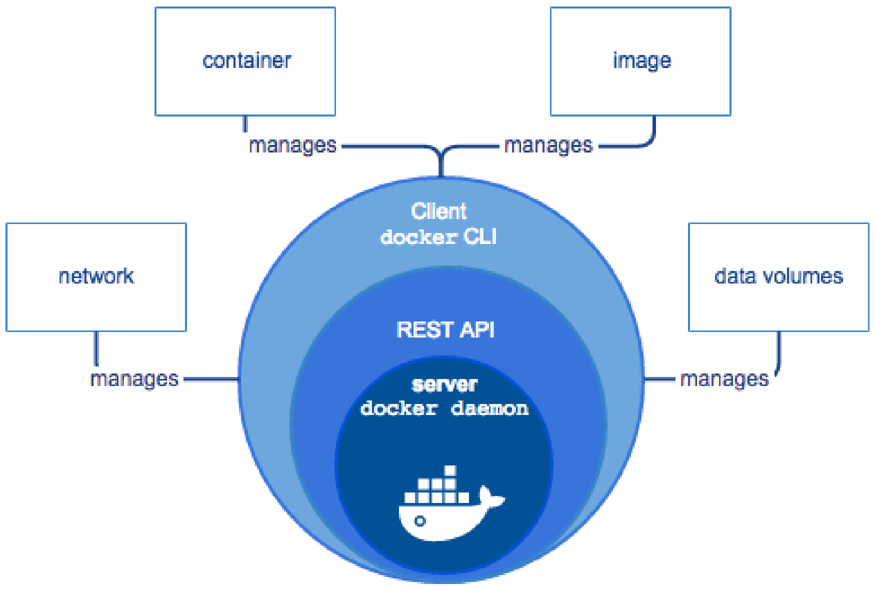


Figure 2.15 Representation of the Docker Engine (Source: [24])

2.4.2 Docker Architecture

Docker operates on a client-server architecture as can be seen in Figure 2.16 [24].

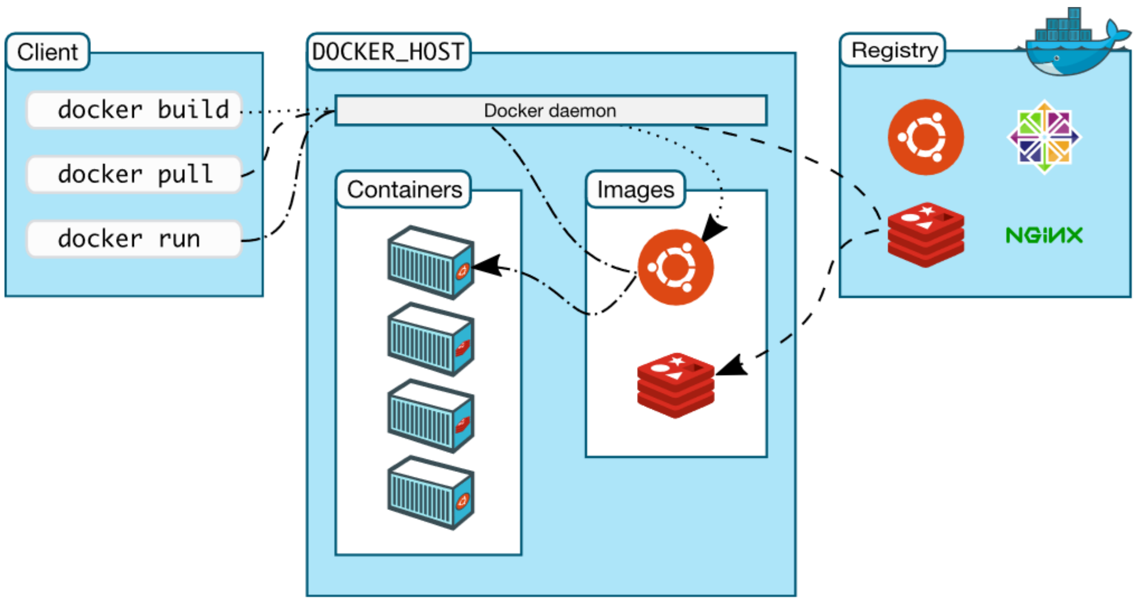


Figure 2.16 Representation of an Model Docker Architecture (Source: [24])

The Client (e.g., Docker CLI) communicates with the Docker Daemon, that does all the work of building, running and distributing the Docker containers [24]. Although the Client and the Daemon can run on the same host, it is possible to connect a Docker Client remotely to a Docker Daemon, as the communication between both is done using a REST API over UNIX sockets or network interfaces [24].

2.4.3 Docker Images and Containers

Docker Image

A Docker Image is an executable read-only template that has everything necessary for creating a Docker Container: the code, a runtime, libraries, environment variables and configuration files [24] [25].

An Image is created using a Dockerfile, that has all the steps defined in it to create an Image and run it [24]. Each instruction in a Docker files creates a new layer in an Image [24]. When a Dockerfile is modified and the Image rebuild, only the altered layers are rebuilt, making the Images lightweight, small and fast [24]. It is also possible to build an Image based on other Images [24].

Docker Container

A Docker Container is a runtime instance of a Docker Image [25]. By using the Docker API or CLI a Container can be created, started, stopped, moved and deleted [24]. It is also possible to attach storage to it, connect to networks, and create a new Image based on the curren9t state of the Container [24].

Although, by default, a Container is well isolated from other Containers and its host, it is possible to control how isolated a Container's network, storage and other subsystems are from other Containers and the host [24].

It is very important that all vital data used by (or created) a Container is kept in persistent storage (e.g., storage attached), as any changes to the state of a Container when it is deleted, disappear [24].

2.4.4 Docker Compose

Docker Compose is a tool for defining and deploying multi-container Docker applications through the use of a YAML Ain't Markup Language (YAML) file, named "docker-compose.yml", which contains all the configurations of the needed services [26]. This allows for the creation and starting of all services with a single command: "docker-compose up" [26].

```
version: "3"
services:

  mosquitto: ← Container 1
    image: eclipse-mosquitto
    hostname: mosquitto
    container_name: mosquitto
    expose:
      - "1883"
      - "9001"
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
    networks:
      - default

  Container 2:
      …
```

2.5 MongoDB

MongoDB is a cross-platform, document-oriented database which offers high performance, high availability and scalability [27]. Data is stored in JavaScript Object Notation (JSON) like documents that provide a flexible way of storing data, as fields can vary from document to document and the data structure can be changed if needed [28].

A Graphical User Interface (GUI) for MongoDB, named MongoDB Compass is also available, allowing the visualization and management of databases [29].

2.6 Hardware

This section contains some information about microcontrollers, sensors and actuators, with the purpose of helping the reader understand the hardware that make up the "Things" later used in this work.

2.6.1 Microcontrollers

Microcontrollers were created with the purpose of serving as the base of embedded systems, that is, systems that work without interruption and human intervention, and therefore are a very useful tool to control something with low resources [6-1]. Microcontrollers have a great flexibility in the creation of software and in the development of the hardware that surrounds it, taking advantage of the communication between both [30].

2.6.1.1 Microcontroller Basic Architecture

Just like a computer, a microcontroller is made up of several components (modules), that while in a computer are separated and visible, and can be easily replaced, in a microcontroller these modules are all concentrated in a small encapsulation (a chip), thus ensuring the basic operation of the microcontroller.

A microcontroller typical consists of several modules: memory unit, Central Processing Unit (CPU), BUS, Input/Output (I/O) ports, serial communication ports, timers, Analog-to-Digital Converter (ADC) [30]; as can be seen in Figure 2.17, which represents the general block diagram of a microcontroller.

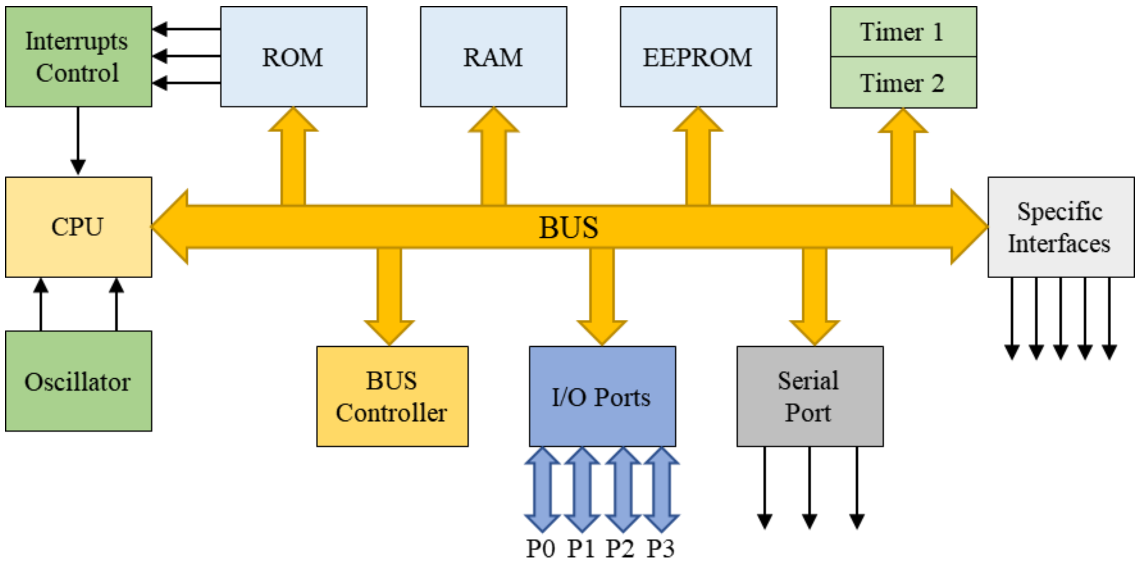


Figure 2.17 Basic Architecture of a Microcontroller (Based on Source: [30])

The memory module consists of two types of memories: the Read-Only Memory/Electrically Erasable Programmable Read-Only Memory (ROM/EEPROM) memory, in which is saved all the critical and essential data when the power is turned off, and the Random-Access Memory (RAM) which contains all the data used by a program (kept in the ROM/EEPROM) during its execution [30]. This data is temporary and not crucial for the operation of the microcontroller and therefore no damage is done when the power is turned off and the RAM erased [30].

The CPU is the brain of the microcontroller, being capable of multiplying, dividing, subtracting summing, and managing the contents of the memories. The CPU is interconnected with the memory and all other modules via the BUS [30].

The BUS is a group of 8, 16 or more transmission lines, that interconnects all modules inside the microcontroller [30].

The Serial Communication module has the function of allowing communication with the outside [30]. This communication is usually done via a Universal Serial Bus (USB), a Recommended Standard 232 (RS232) port, an Ethernet port or Wi-Fi [30].

The I/O ports are used to connect external components to the microcontrollers, therefore extending the capabilities of the microcontroller [30].

Timers are configurable counters whose register value increases a unit in a fixed time interval, saving its value during the time instants (T1 and T2) then calculating their difference, thus obtaining the amount of elapsed time [30]. Timers can provide information about time slots, protocols used and generate signals, namely Pulse-Width Modulation (PWM) signals, widely used in motor speed control [30].

The ADC has the function of converting analog input signals into digital output signals.

2.6.2 Sensors

Sensors are devices whose behavior changes under the influence of a physical property, originating directly or indirectly a signal that indicates this greatness [31]. When they operate directly, they convert a form of neutral energy and are therefore called transducers; those that operate indirectly alter their physical properties, such as resistance, capacitance or inductance, under the action of a magnitude of more or less proportional [31].

Sensors are fundamental for the Internet of Things, as most of the “Things” are sensors, placed in a medium where measurements are to be carried out, converting the measured quantity into an electrical signal, which is then processed through conditioning circuits. After the treatment, the signal is read by a microcontroller programed for the effect and sent through the Internet to the system, thus allowing the monitoring and automatic control of the quantities in question.

2.6.2.1 Sensors Classification

Sensors can be divided into three distinct classes: passive and active sensors, both analog, and digital sensors [31].

Passive sensors are characterized by the occurrence of impedance variations when a variation of the measured quantity occurs [31]. These sensors can be resistive, capacitive, inductive and differential [31].

Active sensors are characterized by directly harnessing the energy of the process to be measured [31]. These sensors can be thermoelectric, pyroelectric, photovoltaic and electromagnetic [31].

Digital sensors allow the measurement of discrete quantities such as counters and devices with frequency output [31].

2.6.2.2 Passive Sensors

Resistive sensors are characterized by having a resistive output, which may have a linear variation, e.g., potentiometers and Light-Dependent Resistors (LDRs), or a non-linear variation, e.g., resistive temperature sensors and force sensors [31]. Figure 2.18 displays a LDR.

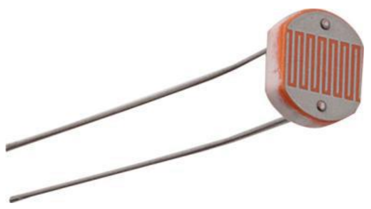


Figure 2.18 LDR (Source: [32])

In the case of capacitive sensors, its output variation is capacitive similar to a variable capacitor, and its therefore an alternative to resistive sensors due to its high resolution, stability and immunity to temperature [31]. These sensors are used to measure linear or angular displacements, distances, liquid level and moisture, being usually used in the detection of failures in industrial manufacturing process [31]. Figure 2.19 shows two capacitive sensors.

Figure 2.19 Capacitive Sensors (Source: [31])

Inductive sensors are characterized by having an inductive output, similar to a variable coil, as they internally consist of a conductive coil, which may have a core, where the passage of variable electric current in time produces a magnetic field, also variable in time [31]. These sensors are used to measure displacements, as they have high sensitivity, resolution and repeatability [31]. Figure 2.20 displays several inductive sensors with different shapes.

Figure 2.20 Inductive Sensors (Source: [31])

Pressure sensors are intended to measure low pressure, presenting the result in the form of voltage [31]. This sensor varies its resistance because of a force being applied on it, thus having a linear variation [31]. The sensor works by having as reference the atmospheric pressure outside the outside the environment of the system to be measured, and the other pin inside the system environment [31]. When a positive pressure is applied to the inner pin, the differential voltage increases linearly, however if the pin is in a vacuum system the differential voltage decreases linearly [31]. Figure 2.21 displays a pressure sensor.

Figure 2.21 Pressure Sensor (Source: [31])

2.6.2.3 Active Sensors

Active sensors behave like generators, producing an electric signal through when a physical phenomenon is detected [31]. Below are some of the most common active sensors.

The main characteristic of electromagnetic sensors is the variation of the magnetic field, which reflects the variability of the measured physical property, without influencing the sensor inductance [31]. These sensors are based on Faraday's law, which states that when there is a relative movement between the conductor and a magnetic field, an electromotive force appears in the conductor [31]. Figure 2.22 shows an electromagnetic sensor.

Figure 2.22 Electromagnetic Sensor (Source: [31])

Thermoelectric sensors, provide the required temperature control in industrial and commercial processes [31]. To carry out the control, this type of sensor equips a device called Thermocouple, whose operation is based on three effects: the Seebeck Effect, which states that different temperatures cause an electric current, the Peltier Effect, which states the heating or cooling of a junction when traversed by a current, and the Thompson Effect, which states the absorption or release of a homogeneous conductor with an inhomogeneous temperature when run by a current [31]. The operation of a Thermocouple consists of the use of a circuit with two distinct metals joined by two junctions, so that if one of the junctions is maintained at a reference temperature, the other junction will serve as a measurement junction, thus converting thermal energy in electricity [31]. The Thermocouple, as shown in Figure 2.23, has the advantages of a large measuring range, a rapid response to temperature variation and good reliability [31]. However, the maximum supported temperature must be lower than the semiconductor melting temperature [31]. Figure 2.23 shows a thermocouple.

Figure 2.23 Thermocouple (Source: [31])

Piezoelectric sensors operation id based on the piezoelectric effect, present in some metals, which consists in the appearance of a potential difference between opposite faces of a metal when submitted to mechanical tension [31]. However, the piezoelectric effect is reversible when a tension is applied between the opposing faces of the material, causing a deformation thereof [31]. This effect can be applied to both actuators and sensors, being applied force, pressure, acceleration, humidity and ultrasonic sensors [31]. Figure 2.24 shows a pair of piezoelectric sensors.

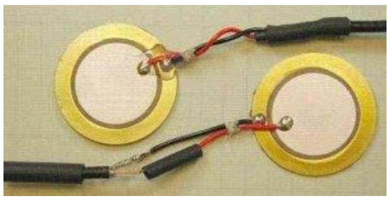


Figure 2.24 Piezoelectric Sensors (Source: [31])

Pyroelectric sensors operation is based on the pyroelectric effect that occurs in crystalline materials when subjected to a temperature variation, which originates surface electrical charges [31]. This type of sensors works similarly to piezoelectric sensors, but instead of being made of a metal, these have a polarized pyroelectric crystal with two metal electrodes on opposite faces [31]. The sensors generate a charge due to changes in its temperature because of incident infrared radiation [31]. These sensors can be used in the detection of thermal radiation at room temperature, non-contact temperature measurement (pyrometers) and temperature-triggered alarm systems [31]. Figure 2.25 displays a pyroelectric sensor.

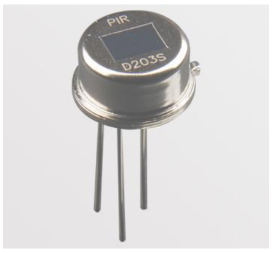


Figure 2.25 Pyroelectric Sensor (Source: [31])

2.6.2.4 Digital Sensors

Digital sensors are the easiest to use as they only have two logic states, 0 or 1, e.g., switches, microswitches, buttons and position switches [31]. Figure 2.26 shows a microswitch.

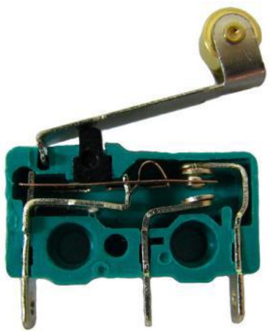


Figure 2.26 Microswitch (Source: [33])

2.6.3 Actuators

Actuators are devices capable of converting electrical, hydraulic or pneumatic energy into mechanical energy [34]. Through transmission systems normally composed of shafts, chains or gears the mechanical energy generated by the actuators is sent to the device that needs to be moved [34]. The actuators can be divided into four groups: hydraulic, pneumatic, electromechanical and signaletic [34].

Hydraulic actuators are components driven by moving fluids compressed at high pressures, usually pressurized oil, whereas pneumatic actuators use compressed air which, when it is at high pressures, assumes the characteristics of a fluid [34]. Hydraulic actuators have the form of linear cylinders to generate linear movements whereas the pneumatic actuators can take the form of linear cylinders, which internally have a piston, or rotating cylinders that have a fin, to provide angular displacements, as seen in Figure 2.27 [34].

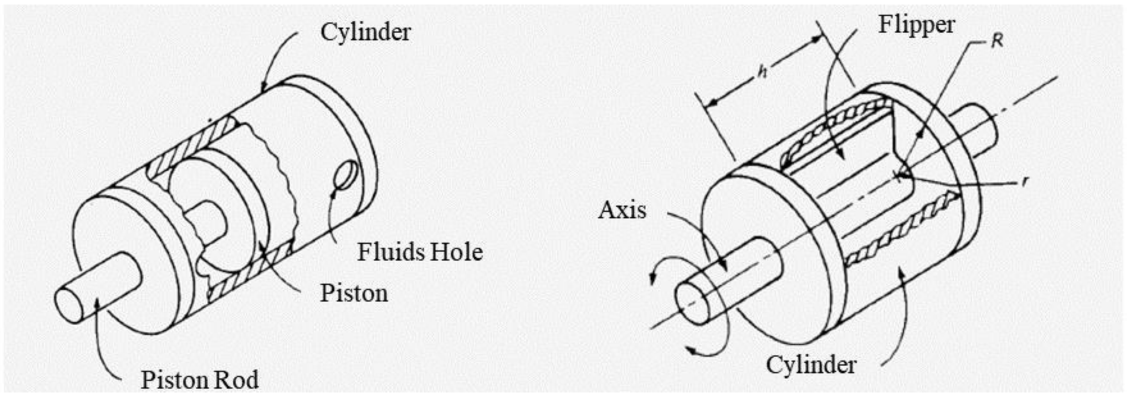


Figure 2.27 Internal Schema of Actuators (Based on Source: [34])

Hydraulic actuators have the advantage of allowing continuous and precise control of movement and speed due to the incompressibility of the fluid used, but they have the disadvantage of being very difficult the exerted force, whereas the pneumatic actuators have the advantage of allowing smooth movements, are simple to control and inexpensive, having the disadvantage of having little stiffness due to the compressibility of the air and being imprecise [34].

Electromagnetic actuators, have a wide variety of models and types, such as Alternating Current (AC) motors, Direct Current (DC) motors, servomotors and stepper motors [34]. Electric motors in general have the advantages of having a great diversity of manufacturers and models, when associated with sensors, they can be used both to control something (e.g., open a valve) and are easy to control, using signals such as PWM signals and H bridges.

DC Electromagnetic Motors, are relatively compact and have a torque that is kept constant with the speed variation of the motor, however these reach a greater mechanical efficiency if used at high speeds, as such, usually gears are used to reduce the generated output speed without changing the working speed of the engine, which also has the effect of increasing the motor force, as can be seen in Figure 2.28 [34].

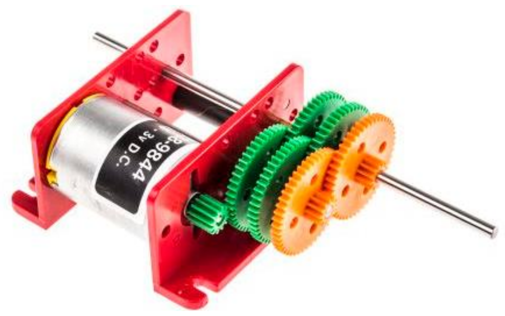


Figure 2.28 DC Motor with Gears (Source: [35])

AC Motors are highly used in industrial applications, especially in linear motors, which are motors that generate linear motion without the use of gears or motion control mechanisms [34]. In Figure 2.29 it is possible to observe AC linear actuators of various sizes.

Figure 2.29 AC Motors (Source: [34])

Luminous Signals, encompass all components whose purpose is to inform or illuminate, e.g., Light Emitting Diodes (LEDs), displays, lamps.

LED is a semiconductor of p-n junction that when subjected to an electric current emits visible light when connected correctly, to emit light a led must be polarized directly, such as shown in Figure 2.30, otherwise the LED can burn.

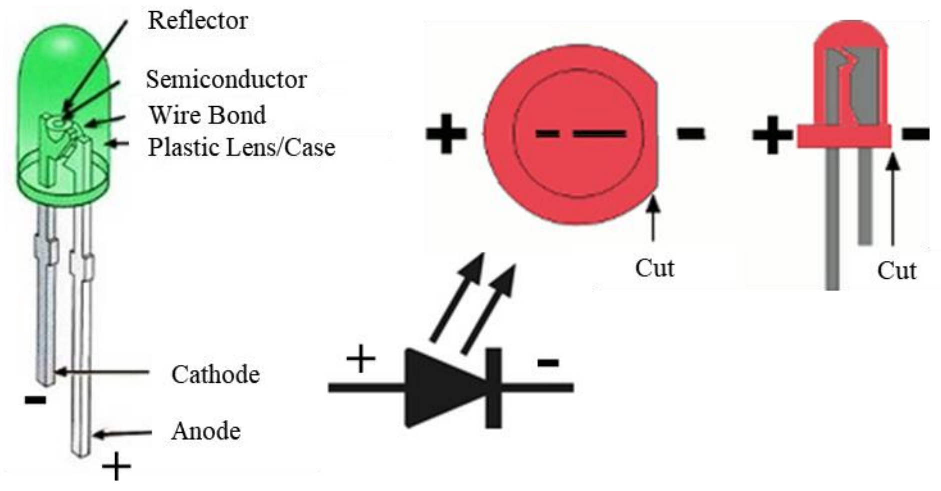


Figure 2.30 LEDs (Based on Source: [36])

Chapter 3 – Universal IoT System Powered by FIWARE

The development of this project happened in two phases, from the end of 2017 to May of 2018 and then to the end of July.

In the first phase of development, enormous difficulties were encountered due to poor documentation about FIWARE and nonexistence of tutorials to show how to combine the FIWARE Generic Enablers. At the time, the existing some of the documentation was outdated, contained dead links, and didn't had examples, being the only decent documentation about the Orion Context Broker Generic Enabler that had some examples explaining how to work with it. The existence of documentation in different places, FIWARE webpage, FIWARE and Telefonica GitHub, and Generic Enablers individual ReadTheDocs webpages, and with different versions between them also didn't help. However, it was clear that the use of Docker was necessary and the simplest way of using the FIWARE technologies, being also the needed to meet the requirement of delivering the work done.

In the second and last phase of development, which started with the attendance of the FIWARE Global Summit 2018 in Porto (8th and 9th of May) and sessions aimed at developers which made it possible to better understand FIWARE and related technologies. It was also announced at this Summit that a collection of introductory tutorials to FIWARE were being created. These tutorials [37] [38] proved to be fundamental to the development of this project, having finally allowed to understand the FIWARE technologies and how to connect and interact with the FIWARE GEs, due to the explanations being done step-by-step.

In the following months after the Summit, a new FIWARE webpage was released with better information, the documentation about the FIWARE technologies was also improved, although it continues to be available in different places, it is now the same in all. New tutorials have been added, and existing ones are continuously updated.

This chapter contains the system architecture diagram, a description of each of the FIWARE Generic Enablers used, and the tests done to the system.

3.1 System Architecture

Based on the available tutorials, mainly on the ones about Orion Context Broker, IoT Agents and Historic Management, it was put together the system architecture seen in Figure 3.1.

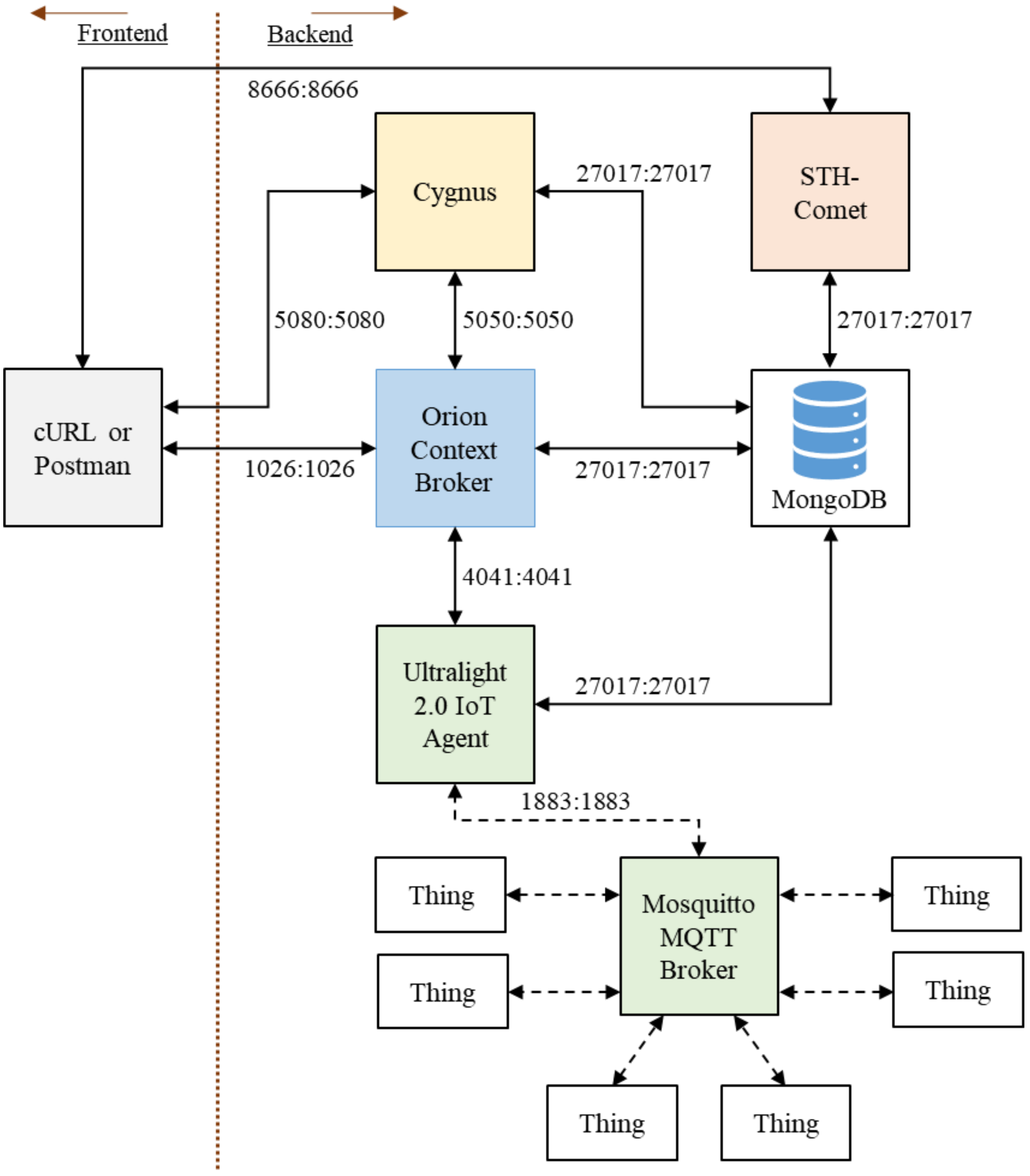


Figure 3.1 System Architecture Block Diagram

The backend part of the system is made up of four FIWARE Generic Enablers: Orion Context Broker, Cygnus, STH-Comet and the Ultralight 2.0 IoT Agent; a MongoDB and a Mosquitto MQTT Broker and several IoT devices (Things).

As there was not enough time to develop a frontend webpage or application, it is instead used cURL commands or the Postman [39] program (used to test and develop APIs), that allows to send commands just like cURL and visualize the responses in a pretty way, instead of raw. This allows to simulate the interactions that the frontend would have with the backend.

3.1.1 Orion Context Broker

The FIWARE Orion Context Broker Generic Enabler is a C++ server implementation of the FIWARE NGSIv2 REST API binding that allows the management of context information (updates, queries, registrations and subscriptions) and is availability [40].

Orion relies on MongoDB to keep persistence of the context data such as data entities, subscriptions and registrations [41].

3.1.1.1 Data Model Guidelines

Although the structure of each data entity within a context can vary according to the use case, the common structure of each entity type should be standardized to promote reuse [41]. The full FIWARE data model guidelines [42] are extensive therefore only the following recommendations are presented here [41]:

- The value fields of context data may be in any language, but all attributes and types must be in American English;
- Entity types names must start with a Capital letter;
- Entity IDs must follow the NGSI-LD guidelines: `urn:ngsi-ld:<entity-type>:<entity-id>`. This guarantees that every ID is unique;
- Data type names must reuse schema.org [43] data types when possible;
- Attribute names must use camel case syntax, e.g., streetAddress;
- Location information must be defined using address and geographical coordinates;
- Geospatial properties must be GeoJSON [44].

3.1.1.2 Service Health

To check if the Orion Context Broker is running, the following HTTP request is used [41]:

```
curl -X GET 'http://localhost:1026/version'
```

Expected response:

```
{
    "orion": {
        "version": "1.12.0-next",
        "uptime": "0 d, 0 h, 3 m, 21 s",
        "git_hash": "e2ff1a8d9515ade24cf8d4b90d27af7a616c7725",
        "compile_time": "Wed Apr 4 19:08:02 UTC 2018",
        "compiled_by": "root",
        "compiled_in": "2f4a69bdc191",
        "release_date": "Wed Apr 4 19:08:02 UTC 2018",
        "doc": "https://fiware-orion.readthedocs.org/en/master/"
    }
}
```

3.1.1.3 Context Data Creation, Update, Delete

HTTP request to create context data, in this case an example entity of the Store type [41]:

```
curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -d '
{
    "id": "urn:ngsi-ld:Store:001",
    "type": "Store",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Bornholmer Straße 65",
            "addressRegion": "Berlin",
            "addressLocality": "Prenzlauer Berg",
            "postalCode": "10439"
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [13.3986, 52.5547]
        }
    },
    "name": {
        "type": "Text",
        "value": "Bösebrücke Einkauf"
    }
}'
```

It is possible to create several entities at once using the same request [45]:

```
curl -iX POST \
  'http://localhost:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -d '{
  "actionType":"APPEND",
  "entities":[
    {
      "id":"urn:ngsi-ld:Shelf:unit001", "type":"Shelf",
      "location":{
        "type":"geo:json", "value":{
"type":"Point","coordinates":[13.3986112, 52.554699]}
      },
      "name":{
        "type":"Text", "value":"Corner Unit"
      },
      "maxCapacity":{
        "type":"Integer", "value":50
      }
    },
    {
      "id":"urn:ngsi-ld:Shelf:unit002", "type":"Shelf",
      "location":{

"type":"geo:json","value":{"type":"Point","coordinates":[13.3987221,
52.5546640]}
      },
      "name":{
        "type":"Text", "value":"Wall Unit 1"
      },
      "maxCapacity":{
        "type":"Integer", "value":100
      }
    }
  ]
}'
```

Update Context Data

This request updates the price attribute of the Product 001 entity [46].

```
curl -iX PATCH \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Product:001/attrs' \
  --header 'Content-Type: application/json' \
  --data ' {
      "price":{"type":"Integer", "value": 89}
}'
```

Delete Context Data

This request deletes the Product 010 entity [46].

```
curl -iX DELETE \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:Product:010'
```

Although is section contains some of the most basic operations that can be done to manage context information, more complex operations are also available [46], like batch operations.

3.1.1.4 Context Data Relationships

Although MongoDB doesn't support relationships like SQL databases, the Orion Context Broker can emulate SQL like relationships through the use of references [45], however data integrity is not guaranteed (e.g., it is possible to make a reference to an inexistent entity).

One-to-Many Relationship

The following example associates shelfs to stores [45]:

```
curl -iX POST \
  'http://localhost:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -d '{
  "actionType":"APPEND",
  "entities":[
    {
      "id":"urn:ngsi-ld:Shelf:unit001", "type":"Shelf",
      "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Shelf:unit002", "type":"Shelf",
      "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Shelf:unit003", "type":"Shelf",
      "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Shelf:unit004", "type":"Shelf",
      "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:002"
      }
    },
    {
      "id":"urn:ngsi-ld:Shelf:unit005", "type":"Shelf",
      "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:002"
      }
    }
  ]
}'
```

Many-to-Many Relationships

The following example associates an item to several entities [45]:

```
curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": "urn:ngsi-ld:InventoryItem:001", "type": "InventoryItem",
    "refStore": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Store:001"
    },
    "refShelf": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Shelf:unit001"
    },
    "refProduct": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Product:001"
    },
    "stockCount":{
        "type":"Integer", "value": 10000
    },
    "shelfCount":{
        "type":"Integer", "value": 50
    }
}'
```

3.1.1.5 Context Data Querying

By ID

This query returns all the entity fields [45].

```
curl -G -X GET \
   'http://localhost:1026/v2/entities/urn:ngsi-ld:Store:001' \
   -d 'options=keyValues'
```

By Type

This query returns all entities whose type is Store [45].

```
curl -G -X GET \
    'http://localhost:1026/v2/entities' \
    -d 'type=Store' \
    -d 'options=keyValues'
```

By Comparing the Values of an Attribute

This query returns all entities of the Store type located in the Kreuzberg district [45].

```
curl -G -X GET \
    'http://localhost:1026/v2/entities' \
    -d 'type=Store' \
    -d 'q=address.addressLocality==Kreuzberg' \
    -d 'options=keyValues’
```

By Comparing the Values of a Geo:Json Attribute

This query returns all entities of the Store type within 1.5km of the given coordinates [45].

```
curl -G -X GET \
  'http://localhost:1026/v2/entities' \
  -d 'type=Store' \
  -d 'georel=near;maxDistance:1500' \
  -d 'geometry=point' \
  -d 'coords=52.5162,13.3777'
```

Reading from Child Entity to Parent Entity

This query returns the Store entity to which the Shelf 001 entity is related [45].

```
curl -G -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Shelf:unit001' \
  -d 'type=Shelf' \
  -d 'options=values' \
  -d 'attrs=refStore'
```

Reading from Parent Entity to Child Entity

This query returns all Shelfs associated with Store 001 [45].

```
curl -G -X GET \
  'http://localhost:1026/v2/entities' \
  -d 'q=refStore==urn:ngsi-ld:Store:001' \
  -d 'options=count' \
  -d 'attrs=type' \
  -d 'type=Shelf'
```

Reading from a Bridge Table

This query returns all Stores selling the Product 001 [45].

```
curl -G -X GET \
  'http://localhost:1026/v2/entities' \
  -d 'q=refProduct==urn:ngsi-ld:Product:001' \
  -d 'options=values' \
  -d 'attrs=refStore'\
  -d 'type=InventoryItem'
```

Visualize all Relationships of an Entity

This query returns all entities that have associated to the Store 001 [45]. This request is helpful to maintain data integrity.

```
curl -G -X GET \
  'http://localhost:1026/v2/entities' \
  -d 'q=refStore==urn:ngsi-ld:Store:001' \
  -d 'options=count' \
  -d 'attrs=type'
```

3.1.1.6 Subscriptions

The Orion Context Broker allows the creation of subscriptions to context information, so that when a change to an attribute of said context information a notification is sent to the frontend [41] [47].

A subscription is set up using a POST request, just like in the following example [47]:

```
curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Notify me of all product price changes",
  "subject": {
    "entities": [{"idPattern": ".*", "type": "Product"}],
    "condition": {
      "attrs": [ "price" ]
    }
  },
  "notification": {
    "http": {
      "url":  "http://<frontendIP>:<frontendPort>/subscription/price-
change"
    }
  }
}'
```

This subscription sends a notification to the frontend every time a product price changes.

Delete a Subscription

This request deletes a subscription based on its ID [47].

```
curl -iX DELETE \
  --url 'http://localhost:1026/v2/subscriptions/<subscription-id>'
```

Update a Subscription

This request updates a subscription notification URL [47].

```
curl -iX PATCH \
  --url
'http://localhost:1026/v2/subscriptions/5ae07c7e6e4f353c5163c93e' \
  --header 'content-type: application/json' \
  --data '{
    "status": "active",
    "notification": {
        "http": {
            "url":  "http://<frontendIP>:<frontendPort>/notify/price-
change"
        }
    }
}'
```

List Subscriptions

This request lists all subscription [47].

```
curl -X GET --url 'http://localhost:1026/v2/subscriptions'
```

View a Subscription Details

This request allows the visualization of a subscription details [47].

```
curl -X GET \
  --url 'http://localhost:1026/v2/subscriptions/<subscription-id>'
```

3.1.2 Ultralight 2.0 IoT Agent and Mosquitto MQTT Broker

An IoT Agent is a FIWARE component that lets Internet of Things devices send their data to and be managed from a Context Broker, in this case the Orion Context Broker, using their native protocols [48]. IoT Agents should also deal with security aspects (authentication and authorization of the channel) [48].

The Orion Context Broker only uses the NGSI API for all interactions, therefore, every IoT Agent provides a North Port NGSI interface used to interact with the Orion, and a South Port to interact with the native protocol of the attached devices [48].

This means, that every IoT device can use their proprietary protocols and transport mechanism, whilst the IoT Agent converts them to NGSI request that the Orion Context Broker can understand [48].

The IoT Agent for the Ultralight 2.0 protocol, used in this project, provides a bridge between HTTP/MQTT messaging with a UL2.0 payload and the Orion Context Broker (NGSI) [48].

The IoT Agent saves all information such as devices URLs and Keys in a MongoDB database [48].

3.1.2.1 Interaction Between the IoT Agent and Mosquitto MQTT Broker

The FIWARE IoT Agent for Ultralight 2.0 protocol will receive southbound request using NGSI and convert them to Ultralight 2.0 MQTT topics for the Mosquitto [49] MQTT Broker [50]. It also listens to the Mosquitto MQTT Broker on registered topics to send measurement northbound the Orion Context Broker [50].

The Mosquitto MQTT Broker acts the central communication point between the IoT Agent and the IoT devices, passing topics between them as necessary [50].

3.1.2.2 IoT Agent Service Health

To check if the IoT Agent is running, the following HTTP request is used [50]:

```
curl -X GET 'http://localhost:4041/iot/about'
```

Expected response:

```
{
    "libVersion": "2.6.0-next",
    "port": "4041",
    "baseRoot": "/",
    "version": "1.6.0-next"
}
```

3.1.2.3 Mosquitto MQTT Broker Service Health

To check if the Mosquitto MQTT Broker is working properly, a MQTT subscriber and a publisher are used, and if the messages sent by the publisher are received by the subscriber, then the Broker is working [50].

Starting a MQTT Subscriber

An MQTT subscriber is started by running a mqtt-subscriber Docker container in a new terminal [48]:

```
docker run -it --rm --name mqtt-subscriber \
  --network fiware_default efrecon/mqtt-client sub -h mosquitto -t
"/#"
```

In which the "-h" flag indicates the Mosquitto hostname, and the "-t" flag the subscribed topic. In this case the used topic allows the subscriber to receive all messages, independent of the topic used by the publisher.

Starting a MQTT Publisher

An MQTT publisher is started by running a mqtt-publisher Docker container in a new terminal [48]:

```
docker run -it --rm --name mqtt-publisher \
  --network fiware_default efrecon/mqtt-client pub -h mosquitto -m
"HELLO WORLD" -t "/test"
```

In which the "-h" flag indicates the Mosquitto hostname, the "-m" flag the message to be sent, and the "-t" flag the published topic. In this case it is sent a "HELLO WORLD" message to the "/test" topic, that if everything is working well, should be received by the subscriber and shown in the subscriber terminal.

3.1.2.4 Connection of IoT Devices

To connect IoT Devices to the system (IoT Agent) it is necessary first to provision a service group and then provision the devices [50]. Provisioning a service group allows to set up a authentication key for a group of devices, and provisioning the devices serves to map the connected devices to the correct entities, otherwise if the devices are not provisioned the IoT Agent will automatically create new entities for the connected devices, however, these entities will have a random id and its attributes will be incomplete [50].

As there is no guarantee that the IoT devise will always have a unique ID, when provisioning a device or a service group the following headers are required [50]:

- fiware-service, used to define the MongoDB database where entities for a given service are held;
- fiware-servicepath, used to distinguish between arrays of devices.

For example, in a smart city system “fiware-service” headers can be used to differentiate between departments (e.g., parks, transport, buildings, etc) and “fiware-servicepath” headers would be used to refer to a specific park (e.g., downtown park, uptown park, etc) [50].

The use of these headers means that data and devices for each service can be identified and separated as needed without being siloed, as data from different devices and other entities in other services and paths can be combined and used as necessary [50].

The use these headers also ensures that there is no overlap of used device IDs [50].

Provisioning a Service Group for MQTT

Provisioning a service group is always the first step done when connecting IoT devices, and it is done as in the example below [50]:

```
curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'
```

In this example, the service is “openiot” and there is no separation between devices as the servicepath is “/” (root) [50]. The authentication key (apikey) is unique to a service group and is used by when devices when communicating as the topic must contain the key (e.g., `/4jggokgpepnvsb2uv4s40d59ov`) [50].

The “resource” attribute is empty since it is only used when the HTTP transport protocol is used [50].

The “cbroker” attribute is optional, since if not provided the IoT Agent will use the default context broker URL defined in its configuration file [50].

Provisioning a Sensor

Provisioning a sensor is mapping the connected IoT device to a corresponding entity, like in the following example [50]:

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
   {
     "device_id":   "motion001",
     "entity_name": "urn:ngsd-ld:Motion:001",
     "entity_type": "Motion",
     "protocol":    "PDI-IoTA-UltraLight",
     "transport":   "MQTT",
     "timezone":    "Europe/Berlin",
     "attributes": [
       {"object_id": "c", "name": "count", "type": "Integer"}
     ],
     "static_attributes": [
       {"name":"refStore",    "type":    "Relationship",    "value":
"urn:ngsi-ld:Store:001"}
     ]
   }
 ]
}'
```

In this example a motion sensor with the “motion001” ID is associated to an entity with the “`urn:ngsd-ld:Motion:001`” ID that is also created at the same time. This association allows the mapping of the sensor reading to a context attribute, in this case the reading “c” is mapped to the attribute “count” which is defined as an Integer, and allows to place the sensor in a store through a reference.

By defining the “transport” attribute as MQTT the IoT Agent knows that it should subscribe to the “`/<api-key>/<device-id>`” topic to receive measurements from this sensor [50].

The provisioning of devices is important because since devices are managed by the IoT Agent the Orion Context Broker would not be able to retrieve data from the device, and by provisioning the device a corresponding entity is created in the data base managed by Orion allowing it to assess the device (entity) data, as shown in the example below [50].

```
curl -G -X GET \
  'http://localhost:1026/v2/entities/urn:ngsd-ld:Motion:001'
  -d 'type=Motion' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

Response [50].

```
{
    "id": "urn:ngsd-ld:Motion:001", "type": "Motion",
    "TimeInstant": {
        "type": "ISO8601","value": "2018-05-25T10:51:32.00Z",
        "metadata": {}
    },
    "count": {
        "type": "Integer","value": "1",
        "metadata": {
            "TimeInstant":   {"type":   "ISO8601","value":   "2018-05-
25T10:51:32.646Z"}
        }
    }
}
```

The timestamp is created automatically by the IoT Agent when it receives messages.

Provisioning an Actuator

Provisioning an actuator is just like provisioning a sensor, except for the fact that commands can defined to control the actuator [50].

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
```

```
  "devices": [
    {
      "device_id": "door001",
      "entity_name": "urn:ngsi-ld:Door:001",
      "entity_type": "Door",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "MQTT",
      "commands": [
        {"name": "unlock","type": "command"},
        {"name": "open","type": "command"},
        {"name": "close","type": "command"},
        {"name": "lock","type": "command"}
       ],
       "attributes": [
        {"object_id": "s", "name": "state", "type":"Text"}
       ],
       "static_attributes": [
         {"name":"refStore",      "type":      "Relationship","value":
"urn:ngsi-ld:Store:001"}
       ]
    }
  ]
}'
```

3.1.2.5 Enable Context Broker Commands

After provisioning the actuators, it is necessary to inform the Orion Context Broker that commands are available to control the actuators [50]. Example of commands registration [50]:

```
curl -iX POST \
  'http://localhost:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "description": "Door Commands",
  "dataProvided": {
    "entities": [
      {
        "id": "urn:ngsi-ld:Door:001", "type": "Door"
      }
    ],
    "attrs": [ "lock", "unlock", "open", "close"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'
```

In order to send commands to the IoT Agent that then forwards them to the actuator, it is necessary to use the NGSIv1 API endpoint and legacy forwarding due to the NGSIv2 API not yet supporting commands.

Example of a command (open) invocation [50]:

```
curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Lamp:001/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "open": {
      "type" : "command",
      "value" : ""
  }
}'
```

3.1.3 Cygnus

Since the Orion Context Broker only holds the most recent context information, and rather than overload Orion with the task of keeping the context history, this task was delegated to other components FIWARE Cygnus Generic Enabler and FIWARE STH-Comet Generic Enabler (next sub-section) [51].

The FIWARE Cygnus Generic Enabler can persist context data into one or several databases, creating a historical view of the context data to which is subscribed to [51].

3.1.3.1 Cygnus Service Health

To check if Cygnus is running, the following HTTP request is used [51]:

```
curl -X GET 'http://localhost:5080/v1/version'
```

Expected response:

```
{
    "success": "true",
    "version":
"1.8.0_SNAPSHOT.ed50706880829e97fd4cf926df434f1ef4fac147"
}
```

3.1.3.2 Subscribing to Context Changes

For Cygnus keep a history of context information, it must be aware of context changes, being informed by the Orion Context Broker through one or more subscriptions, as shown below [51].

```
curl -iX POST \
  'http://localhost:1026/v2/subscriptions' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "description": "Notify Cygnus of all context changes",
  "subject": {
    "entities": [
      {
        "idPattern": ".*"
      }
    ]
  },
  "notification": {
    "http": {
      "url": "http://cygnus:5050/notify"
    },
    "attrsFormat": "legacy"
  },
  "throttling": 5
}'
```

The “fiware-service” and “fiware-servicepath” headers are used to filter the connected IoT Sensors [51].

The “idPattern” defines which type of sensors are to be listened (e.g., motion, temp, hum, etc), in this case Cygnus is informed of all context data changes of every sensor [51].

The “notification url” must match the Cygnus API port [51].

Once again legacy forwarding is used since Cygnus only accepts notification in the older NGSIv1 API format [51].

The “throttling” value defines the changes sample rate [51].

The database used by Cygnus to persist context data has no influence on the subscription, as the database or databases used are defined in the initial configuration of Cygnus [51].

3.1.4 STH-Comet

Both Cygnus and STH-Comet Generic Enablers can be used to keep a record of context information changes, however Cygnus is only capable of saving such changes into several types of databases, while the STH-Comet can only save changes to a MongoDB database it can also retrieve time-based data aggregations [52].

STH-Comet can be configured to work in the following operation modes [52]:

- Minimal mode, STH-Comet is responsible for data collection and interpretation;

- Formal mode, the collection of data is done by Cygnus and the STH-Comet only reads data from an existing database.

The differences between the more flexible and future proof “formal mode” and the simpler and easier to set-up “minimal mode” are summarized in the Table 3.1 [52].

Table 3.1 STH-Comet Minimal Mode vs Formal Mode (Source: [52])

	Minimal Mode	**Formal Mode**
System set-up	Only one configuration supported – easy to set-up	Highly configurable – complex to set-up
Component responsible for data persistence	STH-Comet	Cygnus
Role of STH-Comet	Read and write data	Data read
Role of Cygnus	Not used	Data write
Data aggregation local	MongoDB database connected to STH-Comet	MongoDB database connected to Cygnus and STH-Comet
Multiple databases	No	Yes – MongoDB, PostgreSQL, MySQL
Solution scalability	Does not scale easily – for simple systems	Scales easily – for complex systems (future proof)
Throughput rate	Use where throughput is low	Use where throughput is high

In this work the “formal mode” is used, therefore it is the only mode that will be described, with focus on the STH-Comet as Cygnus was already described in the previous section.

3.1.4.1 STH-Comet Service Health

To check if STH-Comet is running, the following HTTP request is used [52]:

```
curl -X GET 'http://localhost:8666/version'
```

Expected response:

```
{
    "version": "2.3.0-next"
}
```

3.1.4.2 Formal Mode Data Aggregation

As the "formal mode" uses Cygnus to aggregate data, the subscription to context changes is done in the same way as described in the Cygnus related section.

3.1.4.3 Time Series Data Queries

For STH-Comet to retrieve time series data if an adequate amount of data has already been aggregated and some time has passed [52]. Below are presented some examples of possible data queries.

To get the history of a context entity attribute is necessary to send a GET request to the URL: "`.../STH/v1/contextEntities/type/<Entity>/id/<entity-id>/attributes/<attribute>`" [52] (URL used in all examples).

List the first N sampled values

This example request retrieves the first 3 sampled "luminosity" values from "Lamp:001":

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Lamp/id/Lamp:001/
attributes/luminosity' \
  -d 'hLimit=3' \
  -d 'hOffset=0' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The "hLimit" parameter restricts the result to "N" values, and the "hOffset=0" parameter restricts the query start to the first value.

List N sampled values at an Offset

This example request retrieves the fourth, fifth and sixth sampled "count" values from "Motion:001":

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Motion/id/Motion:
001/attributes/count' \
  -d 'hLimit=3' \
  -d 'hOffset=3' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The "hLimit" parameter restricts the result to "N" values, and the "hOffset!=0" parameter makes the query start from the Nth measurement.

List the latest N sampled values

This example request retrieves the latest three sampled “count” values from “Motion:001”:

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Motion/id/Motion:
001/attributes/count' \
  -d 'lastN=3' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The “lastN” parameter restricts the result to the last “N” values.

List the sum of values over a period

This example request shows the total “count” values from “Motion:001” over each minute:

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Motion/id/Motion:
001/attributes/count' \
  -d 'aggrMethod=sum' \
  -d 'aggrPeriod=minute' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The “aggrMethod” parameter defines the type of aggregation to perform over the time series, and the “aggrPeriod” parameter determines the data aggregation period that can be: second, minute, hour or day.

List the minimum of a value over a period

This example request shows the minimum “luminosity” values from “Lamp:001” over each minute:

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Lamp/id/Lamp:001/
attributes/luminosity' \
  -d 'aggrMethod=min' \
  -d 'aggrPeriod=minute' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The “aggrMethod” parameter defines the type of aggregation to perform over the time series, and the “aggrPeriod” parameter determines the data aggregation period that can be: second, minute, hour or day.

List the maximum of a value over a period

This example request shows the maximum “luminosity” values from “Lamp:001” over each minute:

```
curl -G -X GET \
'http://localhost:8666/STH/v1/contextEntities/type/Lamp/id/Lamp:001/
attributes/luminosity' \
  -d 'aggrMethod=max' \
  -d 'aggrPeriod=minute' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

The “aggrMethod” parameter defines the type of aggregation to perform over the time series, and the “aggrPeriod” parameter determines the data aggregation period that can be: second, minute, hour or day.

Querying the mean value of an attribute within a period is not supported, however it can be calculated by combining the sum of the attribute values with the number of samples.

3.1.5 System Configuration Using Docker Compose

As mentioned before, every FIWARE component used, the MongoDB database and the Mosquitto MQTT Broker are implemented using Docker. To rapidly to assemble the system architecture, the Docker Compose tool is used, as it allows to configure and run the components (as containers) by using and running a simple YAML file. When executing the YAML file, it automatically pulls the necessary Docker images from the FIWARE and other Docker Hubs, and creates the corresponding containers already configured.

3.1.5.1 “docker-compose.yml” File

Below is the “docker-compose.yml” file used in this project.

```
version: "3"
services:

  mosquitto:
    image: eclipse-mosquitto
    hostname: mosquitto
    container_name: mosquitto
    expose:
      - "1883"
      - "9001"
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
    networks:
      - default


  mongo-db:
    image: mongo:3.6
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "27017"
    ports:
      - "27017:27017"
    networks:
      - default
    command: --bind_ip_all --smallfiles
    volumes:
      - mongo-db:/data

  orion:
    image: fiware/orion:1.14.0
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "1026"
    ports:
      - "1026:1026"
    command: -dbhost mongo-db -logLevel DEBUG

  cygnus:
    image: fiware/cygnus-ngsi:latest
    hostname: cygnus
    container_name: fiware-cygnus
    depends_on:
        - mongo-db
    networks:
        - default
    expose:
        - "5050"
        - "5080"
    ports:
        - "5050:5050"
        - "5080:5080"
    environment:
        - "CYGNUS_MONGO_HOSTS=mongo-db:27017" #Comma separated list
of Mongo-DB servers which Cygnus will contact to persist historical
context data
        - "CYGNUS_LOG_LEVEL=DEBUG" #The logging level for Cygnus
        - "CYGNUS_SERVICE_PORT=5050" #Notification Port that Cygnus
listens when subcribing to context data changes
        - "CYGNUS_API_PORT=5080" #Port that Cygnus listens on for
operational reasons

  sth-comet:
    image: fiware/sth-comet
    hostname: sth-comet
    container_name: fiware-sth-comet
    depends_on:
        - cygnus
        - mongo-db
    networks:
        - default
    ports:
        - "8666:8666"
    environment:
        - STH_HOST=0.0.0.0
        - STH_PORT=8666
        - DB_PREFIX=sth_
        - DB_URI=mongo-db:27017
      - LOGOPS_LEVEL=DEBUG

    iot-agent:
    image: fiware/iotagent-ul:1.6.0
    hostname: iot-agent
    container_name: fiware-iot-agent
    depends_on:
      - mongo-db
      - mosquitto
    networks:
      - default
    expose:
      - "4041"
      - "7896"
    ports:
      - "4041:4041"
      - "7896:7896"
    environment:
      - "IOTA_CB_HOST=orion" #name of the context broker to update
context
      - "IOTA_CB_PORT=1026" #port the context broker listens on to
update context
      - "IOTA_NORTH_PORT=4041"
      - "IOTA_REGISTRY_TYPE=mongodb" #Whether to hold IoT device
info in memory or in a database
      - "IOTA_LOG_LEVEL=DEBUG" #The log level of the IoT Agent
      - "IOTA_TIMESTAMP=true"
      - "IOTA_MONGO_HOST=mongo-db" #The host name of ongoDB
      - "IOTA_MONGO_PORT=27017" # The port mongoDB is listening on
      - "IOTA_MONGO_DB=iotagentul" #The name of the database used in
mongoDB
      - "IOTA_MQTT_HOST=mosquitto" #The host name of the MQTT Broker
      - "IOTA_MQTT_PORT=1883" #The port the MQTT Broker is listening
on to receive topics
      - "IOTA_MQTT_QOS=2" #MQTT QoS
      - "IOTA_PROVIDER_URL=http://iot-agent:4041"

networks:
  default:

volumes:
  mongo-db:
```

The MongoDB database must be initiated before every other FIWARE Generic Enabler because some of the GEs depend on the existence of the database to initiate.

The configuration of most of the FIWARE GEs are done with the environment variables as it will be detailed in the following sections.

3.1.5.2 MongoDB Configuration

Configuration extracted from the “docker-compose.yml” file:

```
mongo-db:
    image: mongo:3.6
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "27017"
    ports:
      - "27017:27017"
    networks:
      - default
    command: --bind_ip_all –smallfiles #binds to all ip addresses
and uses a smaller default file size
    volumes:
      - mongo-db:/data
```

The most important thing about the MongoDB configuration is to create a volume in the local file system, so that when the container is stopped or deleted the databases are not lost with it. The network ports are the default ones.

3.1.5.3 Orion Context Broker Configuration

Configuration extracted from the “docker-compose.yml” file:

```
orion:
    image: fiware/orion:1.14.0
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "1026"
    ports:
      - "1026:1026"
    command: -dbhost mongo-db -logLevel DEBUG
```

While the other FIWARE GEs used in this project can be configured using environment variables, it was not encountered information that proved that the Orion Context Broker could also be similarly configured, and after some trial and error experimentation it was verified that it is not possible to use environment variables. Instead it is necessary to use commands (a long list of commands is available in [53]) to configure the Orion Context

Broker, these commands are then mapped to Orion's configuration [53] [54] [55]. The configuration file is available, as an example, in Annex A, and it was used as it is with everything by default. Other FIWARE components also have similar configuration files.

3.1.5.4 Mosquitto MQTT Broker Configuration

Configuration extracted from the "docker-compose.yml" file:

```
mosquitto:
    image: eclipse-mosquitto
    hostname: mosquitto
    container_name: mosquitto
    expose:
      - "1883"
      - "9001"
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
    networks:
      - default
```

The Mosquitto uses the port 1883 for MQTT topics and the port 9001 for HTTP/WebSocket communications [50]. The attached volume is the Mosquitto configuration file [50], available in Annex B.

3.1.5.5 IoT Agent Configuration

Configuration extracted from the "docker-compose.yml" file:

```
iot-agent:
    image: fiware/iotagent-ul:1.6.0
    hostname: iot-agent
    container_name: fiware-iot-agent
    depends_on:
      - mongo-db
      - mosquitto
    networks:
      - default
    expose:
      - "4041"
      - "7896"
    ports:
      - "4041:4041"
      - "7896:7896"
environment:
      - "IOTA_CB_HOST=orion" # name of the context broker to update
context
      - "IOTA_CB_PORT=1026" # port the context broker listens on to
update context
```

```
      - "IOTA_NORTH_PORT=4041"
      - "IOTA_REGISTRY_TYPE=mongodb" #Whether to hold IoT device
info in memory or in a database
      - "IOTA_LOG_LEVEL=DEBUG" #The log level of the IoT Agent
      - "IOTA_TIMESTAMP=true"
      - "IOTA_MONGO_HOST=mongo-db" # The host name of ongoDB
      - "IOTA_MONGO_PORT=27017" # The port mongoDB is listening on
      - "IOTA_MONGO_DB=iotagentul" # The name of the database used
in mongoDB
      - "IOTA_MQTT_HOST=mosquitto" # The host name of the MQTT
Broker
      - "IOTA_MQTT_PORT=1883" # The port the MQTT Broker is
listening on to receive topics
      - "IOTA_MQTT_QOS=2" # MQTT QoS
      - "IOTA_PROVIDER_URL=http://iot-agent:4041"
```

The 4041 port is used northbound traffic and the 7896 for southbound traffic. The IoT Agent can be configured by using the environment variables in Table 3.2, although not all are used or needed [50] [56].

Table 3.2 IoT Agent Environment Variables – Part 1 (Source: [50] [56])

Variable	Value	Description
IOTA_CB_HOST	orion	Hostname of the context broker to update context
IOTA_CB_PORT	1026	Port the context broker listens on to update context
IOTA_NORTH_PORT	4041	Port used for configuring the IoT Agent and receiving context updates from the context broker
IOTA_REGISTRY_TYPE	mongodb	Whether to hold IoT device info in memory or in a database
IOTA_LOG_LEVEL	DEBUG	The log level of the IoT Agent
IOTA_TIMESTAMP	true	Whether to supply timestamp information with each measurement received from attached devices
IOTA_MONGO_HOST	mongo-db	The host name of mongoDB - used for holding device information
IOTA_MONGO_PORT	27017	The port mongoDB is listening on
IOTA_MONGO_DB	iotagentul	The name of the database used in mongoDB

Table 3.2 IoT Agent Environment Variables – Part 2 (Source: [50] [56])

IOTA_MONGO_USER	-	Username for the MongoDB database user
IOTA_MONGO_PASS	-	Password for the MongoDB database user
IOTA_PROVIDER_URL	http://iot-agent:4041	URL passed to the Context Broker when commands are registered, used as a forwarding URL location when the Context Broker issues a command to a device
IOTA_MQTT_HOST	mosquitto	The host name of the MQTT Broker
IOTA_MQTT_PORT	1883	The port the MQTT Broker is listening on to receive topics
IOTA_MQTT_USERNAME	-	Client username for authentication on the MQTT Broker
IOTA_MQTT_PASSWORD	-	Client password for authentication on the MQTT Broker
IOTA_MQTT_QOS	2	Quality of Service level
IOTA_HTTP_PORT	7896	The port where the IoT Agent listens for IoT device traffic over HTTP

All values shown above are the default ones, that can be used in an initial phase but must be changed later (usernames and passwords).

Although MQTT supports authentication of clients by using usernames and passwords, and encryption of the communication channel using Transport Layer Security/Secure Sockets Layer (TSL/SSL) certificates, these security features that should be implemented in every solution are not being used in this project stage. However only IoT devices that know the API Key of a service group can communicate with the system.

3.1.5.6 Cygnus Configuration

Configuration extracted from the "docker-compose.yml" file:

```
cygnus:
    image: fiware/cygnus-ngsi:latest
    hostname: cygnus
    container_name: fiware-cygnus
    depends_on:
        - mongo-db
    networks:
        - default
    expose:
        - "5050"
        - "5080"
    ports:
        - "5050:5050"
        - "5080:5080"
    environment:
        - "CYGNUS_MONGO_HOSTS=mongo-db:27017" # Comma separated list
of Mongo-DB servers which Cygnus will contact to persist historical
context data
        - "CYGNUS_LOG_LEVEL=DEBUG" # The logging level for Cygnus
        - "CYGNUS_SERVICE_PORT=5050" # Notification Port that Cygnus
listens when subcribing to context data changes
        - "CYGNUS_API_PORT=5080" # Port that Cygnus listens on for
operational reasons
```

The 5050 port is used to listen for notifications from the Orion Context Broker, and the 5080 port is used for administration purposes [51]. Cygnus can be configured by using the environment variables in Table 3.3[51] [57].

Table 3.3 Cygnus Environment Variables – Part 1 (Source: [51] [57])

Variable	Value	Description
CYGNUS_MONGO_HOSTS	mongo-db:27017	Comma separated list of Mongo-DB servers which Cygnus will contact to persist historical context data
CYGNUS_MONGO_USER	-	Username for the MongoDB database user
CYGNUS_MONGO_PASS	-	Password for the MongoDB database user
CYGNUS_LOG_LEVEL	DEBUG	The logging level for Cygnus
CYGNUS_SERVICE_PORT	5050	Notification Port that Cygnus listens when subscribing to context data changes
CYGNUS_API_PORT	5080	Port that Cygnus listens on for operational reasons
CYGNUS_POSTGRESQL_HOST	postgres-db	Hostname of the PostgreSQL server used to persist historical context data

Table 3.3 Cygnus Environment Variables – Part 2 (Source: [51] [57])

CYGNUS_POSTGRESQL_PORT	5432	Port that the PostgreSQL server uses to listen to commands
CYGNUS_POSTGRESQL_USER	postgres	Username for the PostgreSQL database user
CYGNUS_POSTGRESQL_PASS	password	Password for the PostgreSQL database user
CYGNUS_POSTGRESQL_ENABLE_CACHE	true	Switch to enable caching within the PostgreSQL configuration
CYGNUS_MYSQL_HOST	mysql-db	Hostname of the MySQL server used to persist historical context data
CYGNUS_MYSQL_PORT	3306	Port that the MySQL server uses to listen to commands
CYGNUS_MYSQL_USER	root	Username for the MySQL database user
CYGNUS_MYSQL_PASS	123	Password for the MySQL database user
CYGNUS_MULTIAGENT	true	Whether to persist data into multiple databases.

Not all the environment variables are used, and all sensitive information such as passwords and usernames should be passed Docker Secrets instead of environment variables [57]. All values shown above are the default ones, that can be used in an initial phase but must be changed later (usernames and passwords).

3.1.5.7 STH-Comet Configuration

Configuration extracted from the "docker-compose.yml" file:

```
sth-comet:
    image: fiware/sth-comet
    hostname: sth-comet
    container_name: fiware-sth-comet
    depends_on:
        - cygnus
        - mongo-db
    networks:
        - default
    ports:
        - "8666:8666"
    environment:
        - STH_HOST=0.0.0.0
        - STH_PORT=8666
        - DB_PREFIX=sth_
        - DB_URI=mongo-db:27017
      - LOGOPS_LEVEL=DEBUG
```

The 8666 port is used to listen for notifications from the Orion Context Broker, and time-based queries. The STH-Comet can be configured by using the environment variables in Table 3.4 [52] [58].

Table 3.4 STH-Comet Environment Variables (Source: [52] [58])

Variable	Value	Description
STH_HOST	0.0.0.0	The address where STH-Comet is hosted - within this container it means all IPv4 addresses on the local machine
STH_PORT	8666	Operations Port that STH-Comet listens on, it is also used when subscribing to context data changes
DB_PREFIX	sth_	The prefix added to each database entity if none is provided
DB_URI	mongo-db:27017	The Mongo-DB server which STH-Comet will contact to persist historical context data
DB_USERNAME	-	Username for the MongoDB database user
DB_PASSWORD	-	Password for the MongoDB database user
LOGOPS_LEVEL	DEBUG	The logging level for STH-Comet

3.2 IoT Device and Sensors Used

This section contains a description of the microcontroller used in this project, to which sensors and actuators are connected. The sensors used are some of the most common types of sensors that can be found in almost all IoT applications. Actuators, such as motors, switches, valves and others similar are not used, instead as most of control commands of actuators are ON and OFF commands, LEDs are used to visualize the result of said commands.

3.2.1 Microcontroller: NodeMcu Devkit v1.0 ESP8266 Wi-Fi Module ESP-12E

NodeMcu is an opensource firmware e development kit that eases the development of IoT products [59]. The NodeMcu development kit version 1.0 is a board (Figures 3.2 and 3.3), similar to Arduino, that integrates the ESP8266-12E Wi-Fi microcontroller [59].

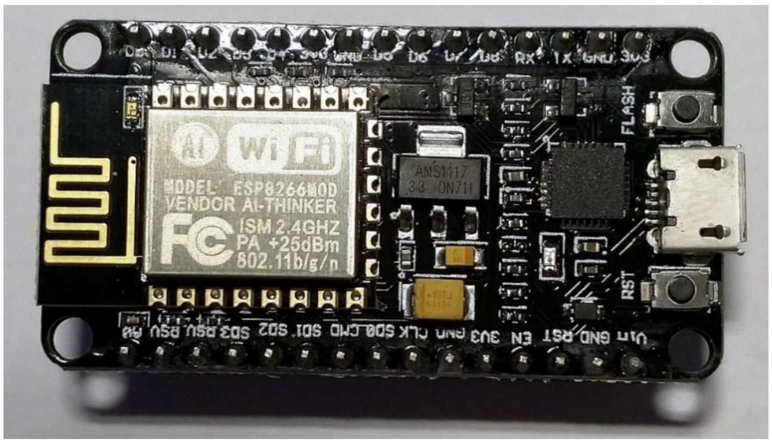


Figure 3.2 NodeMcu Devkit v1.0 (front)

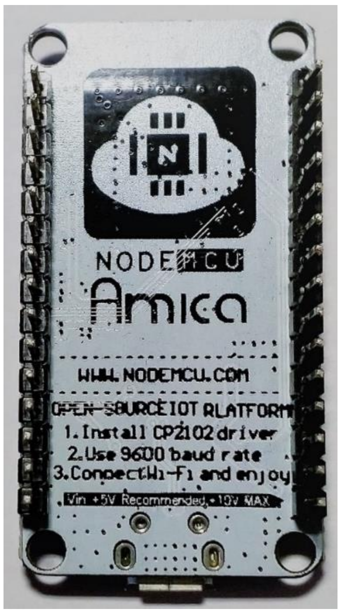


Figure 3.3 NodeMcu Devkit v1.0 (back)

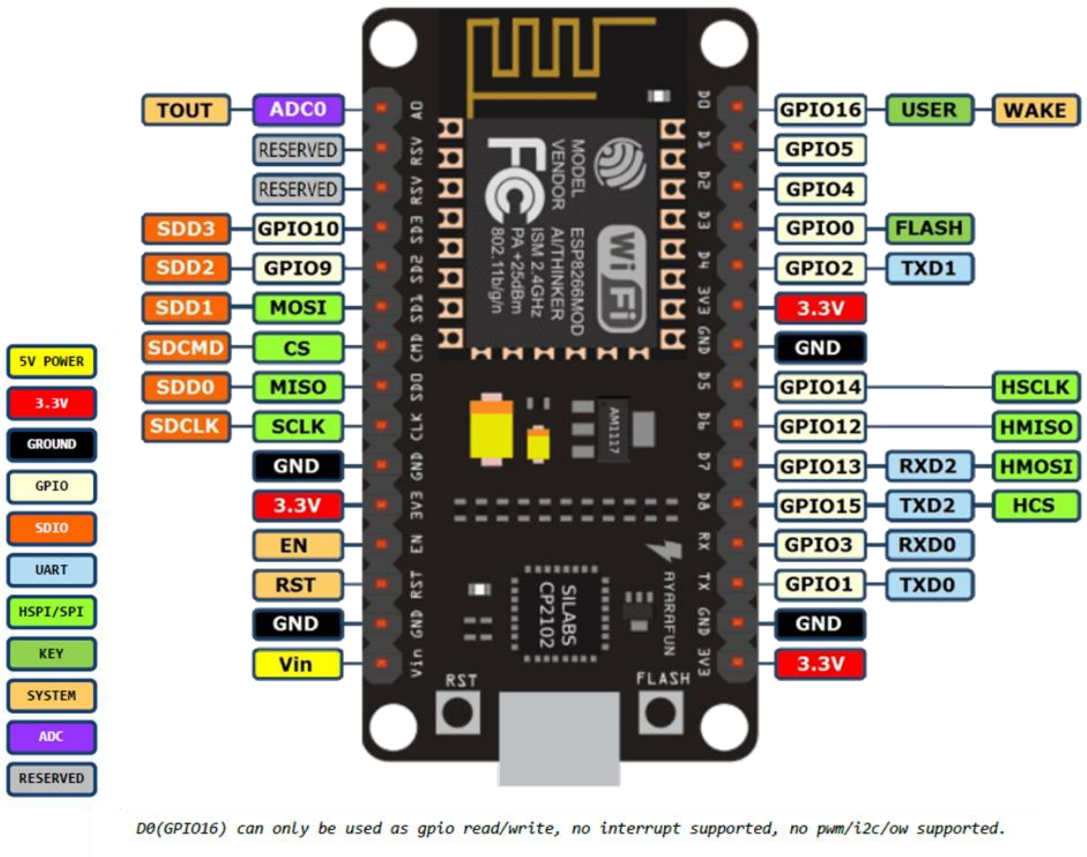


Figure 3.4 NodeMcu Devkit v1.0 Pinout (Based on Sources: [60] [61])

To operate correctly, the board must be supplied with 5V via the micro-USB port or by using the Vin pin, however the ESP8266-12E microcontroller operates between 3V and 3.6V [62]. The board has a total of 30 pins, of those:

- 15 pins are General Purpose Input/Output (GPIO), of which 4 pins can also be used for Hardware Serial Peripheral Interface (HSPI) communication, 5 pins for Universal Asynchronous Receiver-Transmitter (UART) communication, 2 pins for Secure Digital Input Output (SDIO), 1 pin for Flash (for flashing firmware), and 1 pin (Wake) that can be used to wake the microcontroller from sleep mode;
- 4 pins that can be either used for Serial Peripheral Interface (SPI) communication or for SDIO;
- 1 ADC pin;
- 1 reset (RST) pin;
- 1 chip enable (EN) pin;
- 2 reserved pins;
- 1 pin for Vin;

- 4 ground pins;
- 3 pins that supply 3.3V.

Only the GPIO pins indicated in the Figure 3.5 can be used for PWM signals through software programming [63].

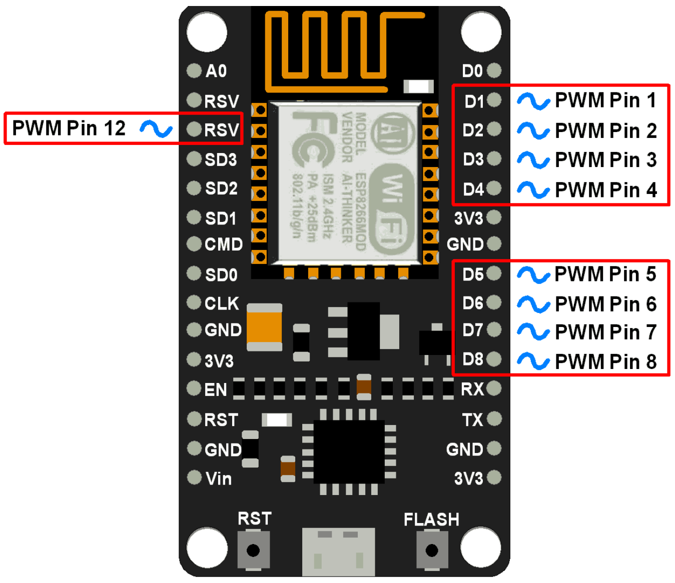


Figure 3.5 NodeMcu Devkit v1.0 PWM Pins (Source: [63])

The ESP8266-12E microcontroller has the architecture displayed in Figure 3.6, which has some of the blocks previous mentioned [62].

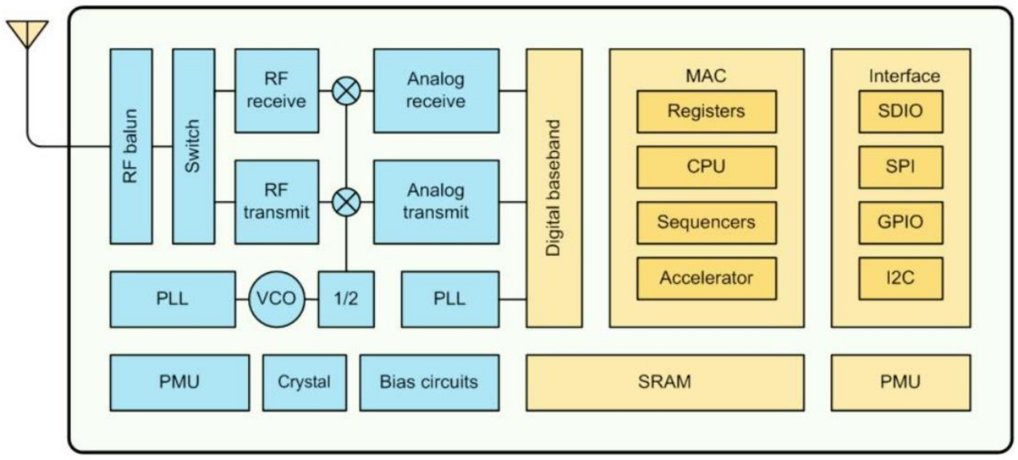


Figure 3.6 ESP8266 Block Diagram (Source: [62])

The microcontroller has as main characteristics [62]:

- Support for 802.11 b/g/n Wi-Fi protocols;
- Wi-Fi 2.4 GHz, with WPA/WPA2;

- Support for antenna diversity;
- Integrated TCP/IP protocol stack;
- Station/Access Point/Station + Access Point (STA/AP/STA+AP) operation modes;
- +20dBm output power in 802.11 b mode;
- Operating voltage between 3V and 3.6V;
- Operating current of 80mA;
- Deep sleep current <10uA;
- Power down leakage current < 5uA;
- Standby power consumption of < 1.0mW;
- Operating temperature range between 40ºC and 125ºC.

More details can be found on the ESP8266-12E datasheet [62] available in Annex C.

This board and the ESP8266 microcontroller are fully compatible with the Arduino IDE, meaning it is possible to use the millions of libraries available for Arduino. In Annex D, a user manual is available for the board that also has step-by-step instructions on how to configure the Arduino IDE to be used with the board [64].

3.2.2 Sensors

As mentioned before the sensors used are some of the most common types of sensors that can be found in almost all IoT applications, such as air temperature sensors, air humidity sensors, ultrasonic sensors, and a more specific, an earth humidity sensor.

3.2.2.1 DHT22 Sensor (Air Temperature and Humidity Sensor)

The DHT22 or AM2303 sensor, Figure 3.7, is a high precision and stable capacitive sensor that measures the air temperature between -40ºC and 80ºC, and air humidity between 0% to 100% [65].

Figure 3.7 DHT22 Sensor (Source: [66])

The sensor must be powered by 3.3V to 6V, and typically takes about 2 seconds to collect data [65]. The sensor datasheet [65] is available in Annex E.

In this project it was used a solution by DFROBOT [67] which offers the sensor in a ready to use module, as seen in Figure 3.8 [67].

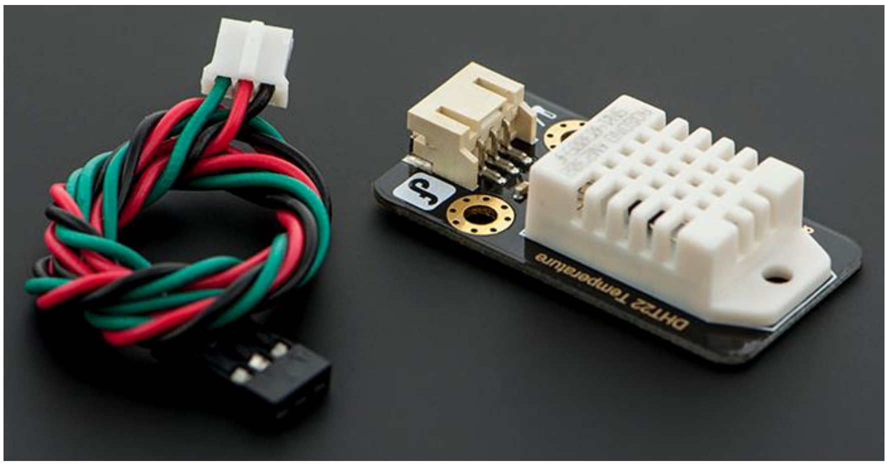


Figure 3.8 DFROBOT DHT22 Module (Source: [67])

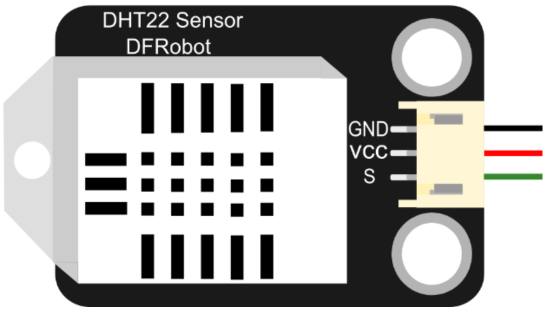


Figure 3.9 DFROBOT DHT22 Module Pinout (Source: [67])

3.2.2.2 XL-MaxSonar-EZ MB1260 Sensor (Ultrasonic Sensor)

Ultrasonic sensors use high frequency sound to detect and localize objects, by measuring the time of flight for sound which has been transmitted to and reflected it is possible to calculate the distance at which the object is.

The sensor used, Figure 3.10, has a minimum range of 22 cm and a maximum range of 7.50 m, a resolution of 1 cm, and requires a power supply of 5V [68]. The sensor datasheet [68] is available in Annex F.

Figure 3.10 XL-MaxSonar-EZ MB1260 Ultrasonic Sensor [Source: [68])

3.2.2.3 Earth-Humidity Sensor

The Earth-Humidity sensor, Figure 3.11, just like the name suggests, it is a sensor used the measure the humidity value of the soil. The sensor used is fabricated by ITEAD and is offered in a ready to use module that only needs to be connected to the microcontroller.

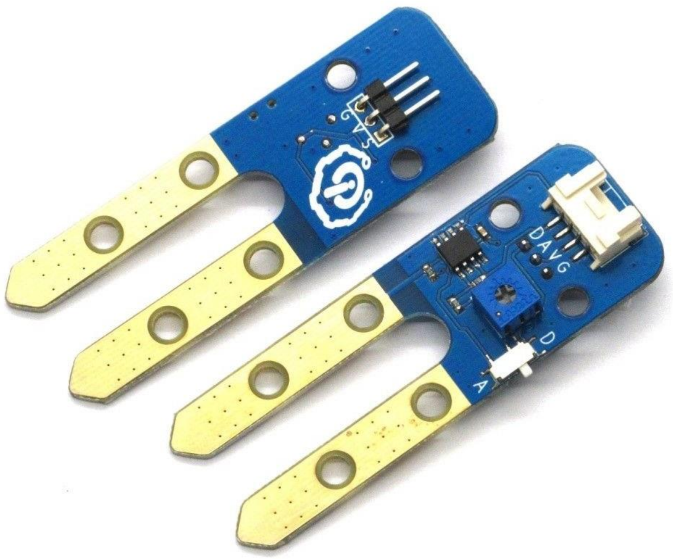


Figure 3.11 Earth-Humidity Sensor Module (Source: [4-33])

The sensor can either be powered by 3.3V or 5V, and has two interfaces with three pins and four pins, the former can only the use of the digital or analog output defined according to the switch and the latter allows the use of both the digital and analog outputs. The sensor datasheet [69] is available in Annex G.

3.2.3 Power Supply

As most of the IoT devices are used in remote environments or places without electrical power, it is necessary to use batteries to power the devices. However as large capacity batteries are expensive, and they must last for years before being changed, techniques such as limiting the number of transmissions, put the device in sleep mode when not used or between transmissions, or even use energy harvesting methods, are employed.

Energy harvesting methods include the use of solar panels, generate energy from vibration and heat, use energy from electromagnetic waves, or even the use of mini wind-powered generators.

Chapter 4 – IoT System Tests and Results

This chapter contains all tests done to the IoT System and the results obtained. However, since to test the system it was always necessary to create context entities and use IoT devices with attached sensors and actuators, a simple fictional use case was devised to make testing more interesting and to also serve as an example of a practical application of the Universal IoT System Powered by FIWARE developed.

4.1 Use Case: Control of Water (Irrigation and Supply)

As water is an increasingly scarce natural resource due to global warming and other environmental factors, it is increasingly important to find innovative and effective ways to manage this vital resource for life on Earth. It is therefore proposed to create a water management system, namely on water supply and irrigation, applied to the agricultural sector, since it is highly dependent on water.

Farms are typical composed of multiple plantation fields and can also have several water sources, such as wells, reservoirs, boreholes, dams, canals and tanks (springs). The irrigation system used also depends on the type of plantation, if cereals or similar plants are being grown then a wide area irrigation system (e.g., sprinklers) is used as the field is full of millions of individual plants, however, if fruit trees or similar are being grown then localized irrigation systems (e.g., grip irrigation) are used. For both irrigation systems it is always essential to monitor the earth humidity to know when the pants need to be watered, to turn on the irrigation system. If a localized irrigation system is used it is possible to monitor the needs of each plant individually and only turn water on for plants in need.

It is also important to monitor the water sources for water level and water temperature as water to hot or cold can damage the plants. Electro valves, which work like taps, are used to open or close the irrigation hose for the irrigation system (wide area) and for each individual plant (localized system, several hoses). This sensors and actuators allow the System to understand and control the world.

In Figure 4.1 a general scheme for this use case is represented.

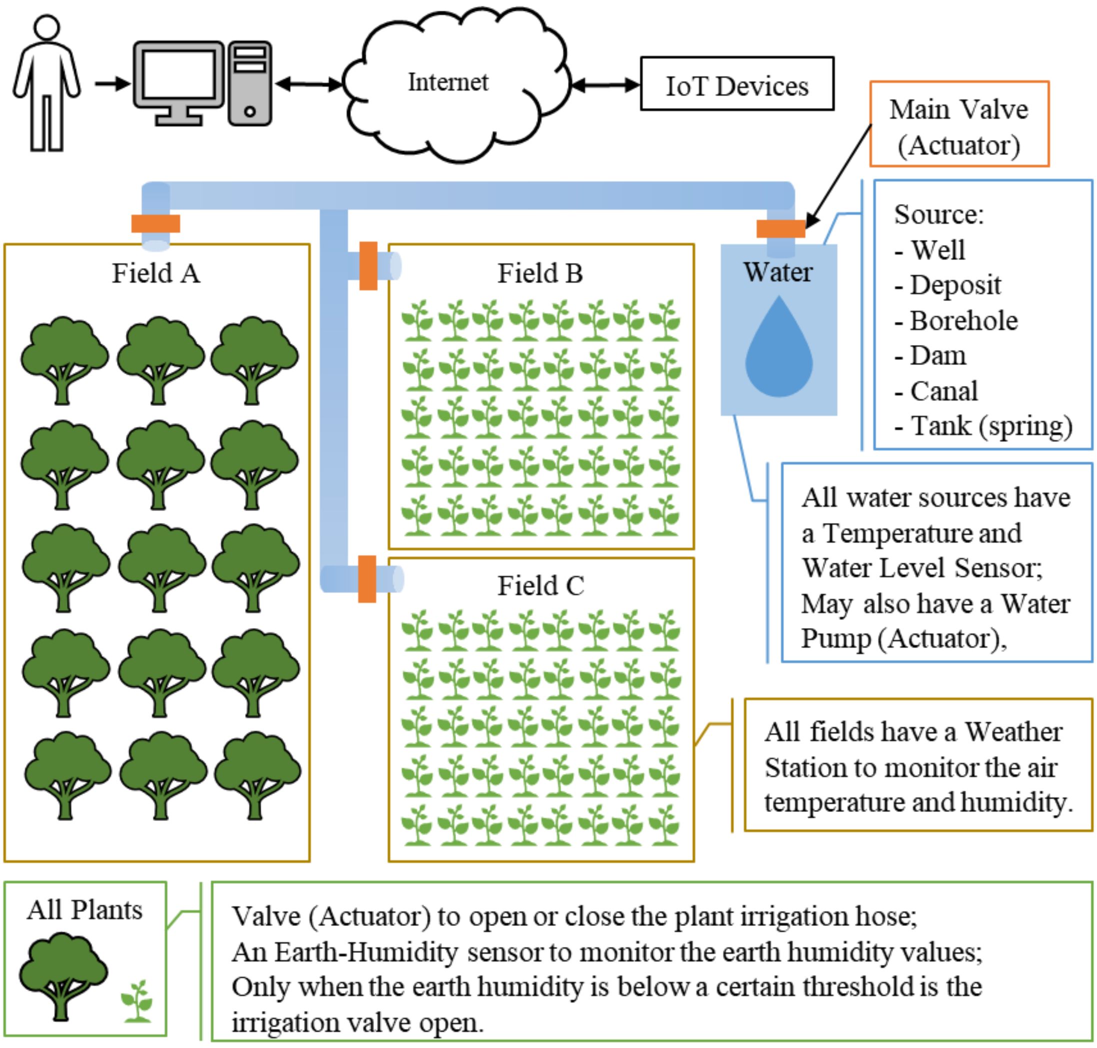

Internet
IoT Devices
Main Valve (Actuator)
Field A
Field B
Water
Source:
- Well
- Deposit
- Borehole
- Dam
- Canal
- Tank (spring)
All water sources have a Temperature and Water Level Sensor; May also have a Water Pump (Actuator),
Field C
All fields have a Weather Station to monitor the air temperature and humidity.
All Plants
Valve (Actuator) to open or close the plant irrigation hose;
An Earth-Humidity sensor to monitor the earth humidity values;
Only when the earth humidity is below a certain threshold is the irrigation valve open.

Figure 4.1 Fictional Use Case Scheme

4.2 System Tests and Results

The Universal IoT System here morphed to a IoT System to Control Water (Irrigation and Supply) as per the defined use case for demonstration and testing purposes, encompasses a farm with three fields (A, B and C), and a well and a tank as water sources. However, due to hardware limitations only the tank is monitored with an ultrasonic sensor to observe the water level and simulated valve. For the fields, only field A will have implemented a weather station to monitor the air temperature and humidity, and an earth-humidity sensor and valve as part of a localized irrigation system.

The following tests were performed:

- System set-up;
- FIWARE components health check;

- Entities creation;
- Entities association;
- Entities modification;
- Entities removal;
- Mosquitto MQTT Broker health check;
- Service group provisioning;
- Sensors provisioning;
- Actuators provisioning;
- Enabling of Context Broker commands;
- Obtain measurements from the DHT22 sensor;
- Obtain measurements from the ultrasonic sensor;
- Obtain measurements from the earth-humidity sensor;
- Send measurements from the DHT22 sensor to the System;
- Send measurements from the ultrasonic sensor to the System;
- Send measurements from the earth-humidity sensor to the System;
- Verify if commands are received by IoT Devices with actuators (by turning ON and OFF LEDs);
- Subscriptions;
- Data persistence;
- Time-series data queries.

4.2.1 System Set-Up

To simplify the IoT System set-up, a simple script named “setup.sh” was created, whose instructions on how to use are available in Annex H.

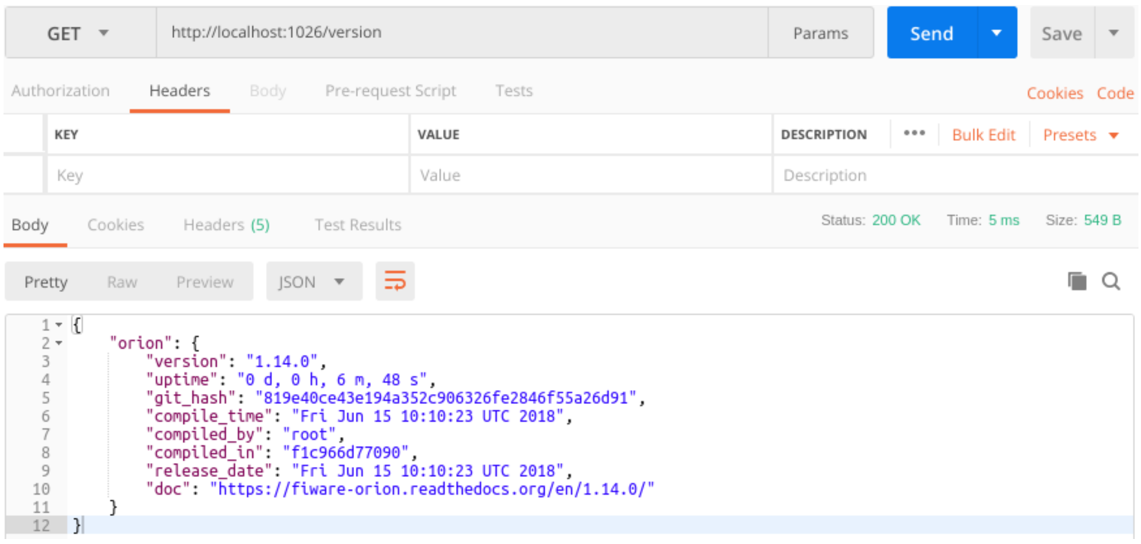
4.2.2 FIWARE Components Health Check

After running the set-up script, “create” command, containers for all the architecture components should have been successfully created and initialized. However, it is important to check the health of the components, if they are properly working, which is done by asking every component for its version and by verifying if the databases were created as they should.

For this test and others following it, the Postman program will be used for simple interactions with the System and cURL commands for complex interactions, such as creating entities. It will be also used the MongoDB GUI, named Compass, to visualize the databases.

4.2.2.1 Orion Context Broker Health Check

Figure 4.2 shows the request for verifying if Orion is running and the obtained response.



http://localhost:1026/version

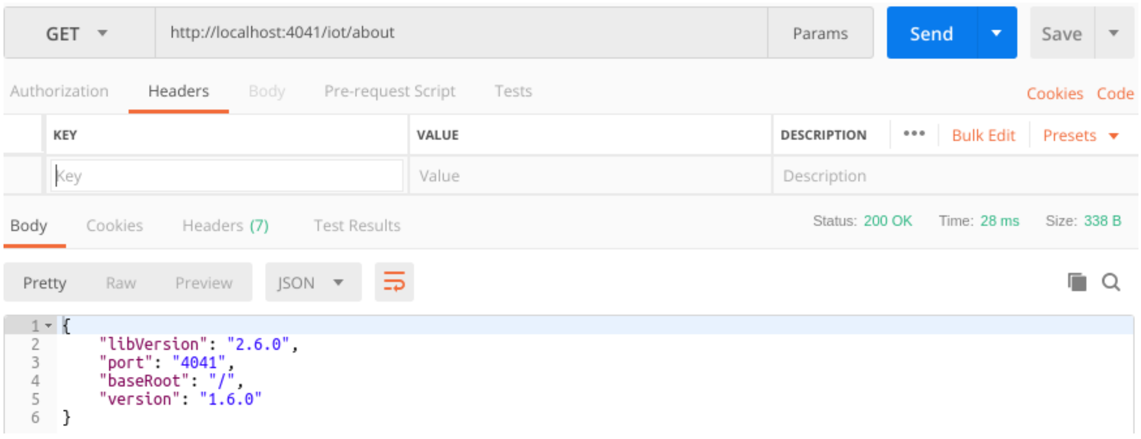
Status: 200 OK Time: 5 ms Size: 549 B

```
{
    "orion": {
        "version": "1.14.0",
        "uptime": "0 d, 0 h, 6 m, 48 s",
        "git_hash": "819e40ce43e194a352c906326fe2846f55a26d91",
        "compile_time": "Fri Jun 15 10:10:23 UTC 2018",
        "compiled_by": "root",
        "compiled_in": "f1c966d77090",
        "release_date": "Fri Jun 15 10:10:23 UTC 2018",
        "doc": "https://fiware-orion.readthedocs.org/en/1.14.0/"
    }
}
```

Figure 4.2 Orion Context Broker Health Check

4.2.2.2 IoT Agent Health Check

Figure 4.3 shows the request for verifying if the IoT Agent is running and the obtained response.



http://localhost:4041/iot/about

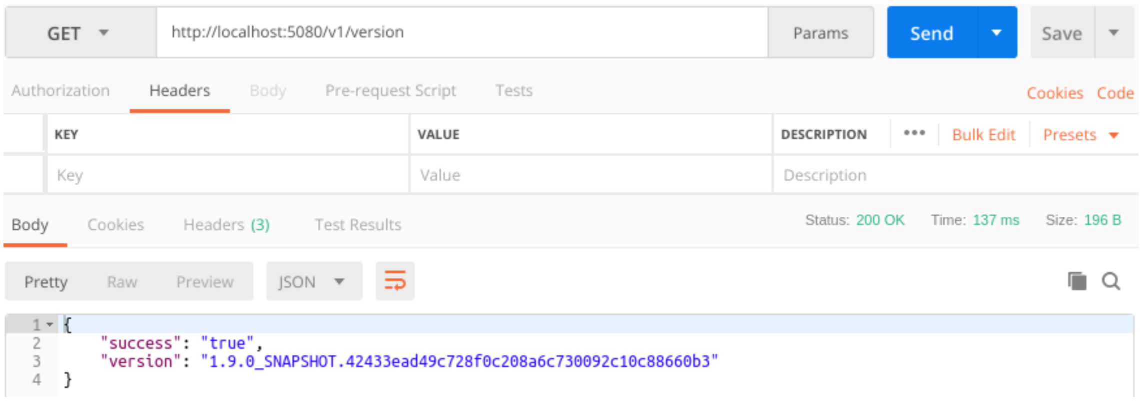
Status: 200 OK Time: 28 ms Size: 338 B

```
{
    "libVersion": "2.6.0",
    "port": "4041",
    "baseRoot": "/",
    "version": "1.6.0"
}
```

Figure 4.3 IoT Agent Context Broker Health Check

4.2.2.3 Cygnus Health Check

Figure 4.4 shows the request for verifying if Cygnus is running and the obtained response.

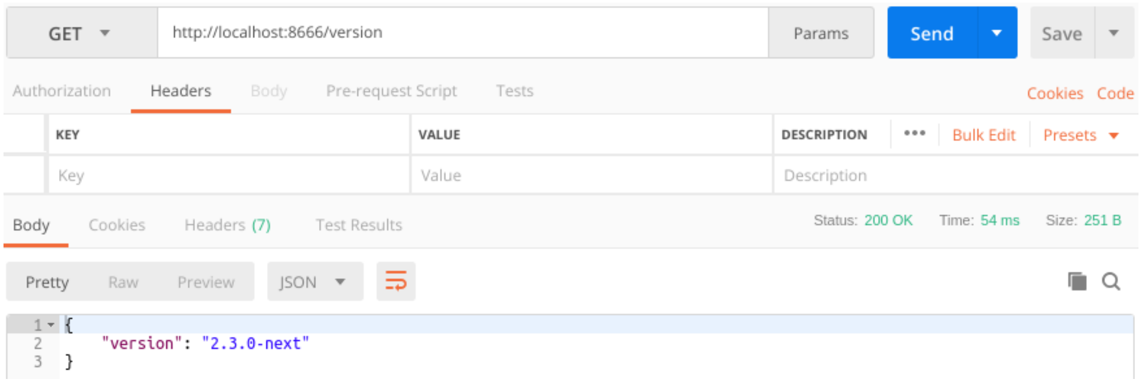


```
{
    "success": "true",
    "version": "1.9.0_SNAPSHOT.42433ead49c728f0c208a6c730092c10c88660b3"
}
```

Figure 4.4 Cygnus Health Check

4.2.2.4 STH-Comet Health Check

Figure 4.5 shows the request for verifying if STH-Comet is running and the obtained response.

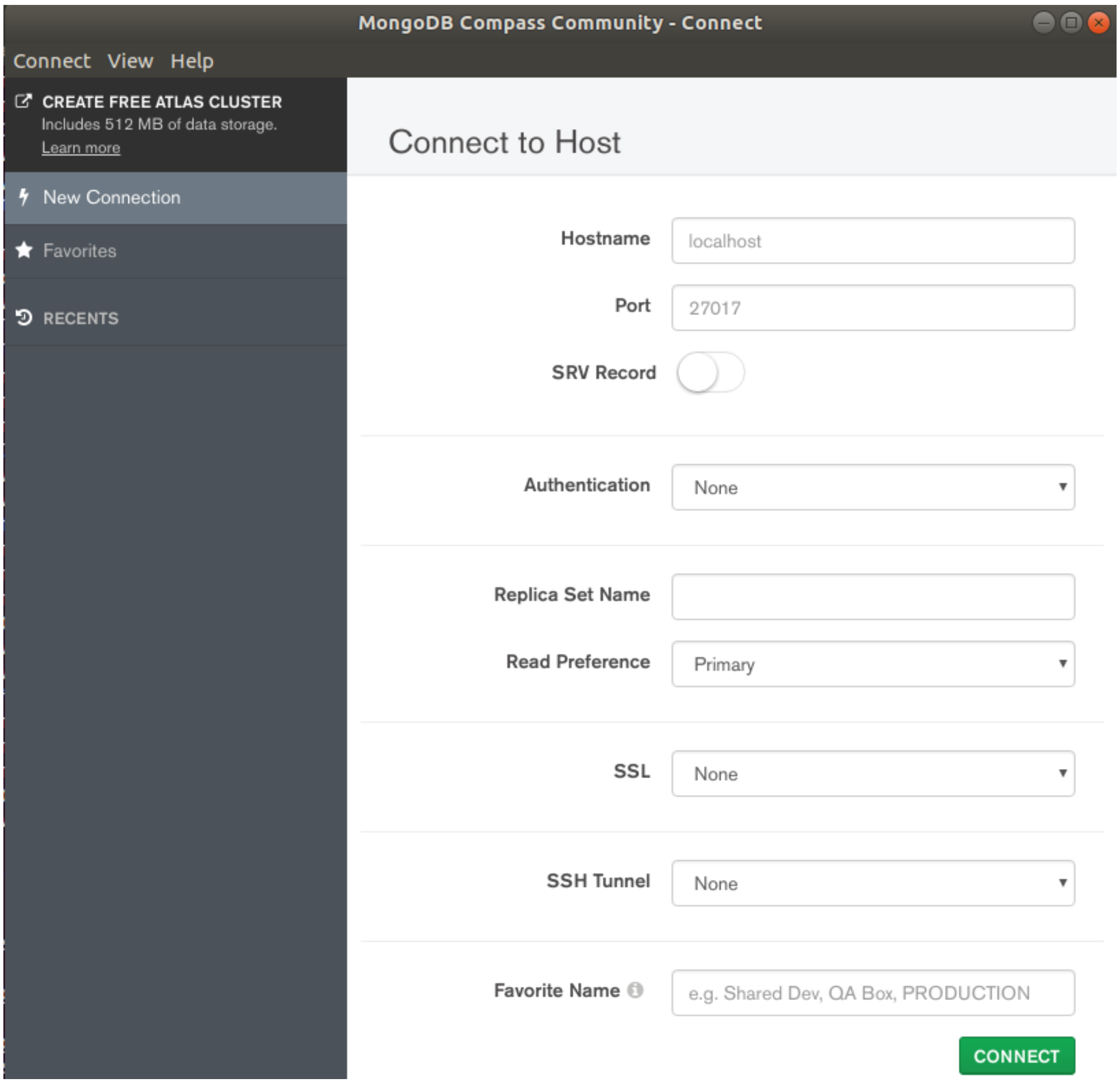


```
{
    "version": "2.3.0-next"
}
```

Figure 4.5 STH-Comet Health Check

4.2.2.5 Databases Created

Figure 4.6 displays the connection of Compass to the MongoDB cluster, to visualize the databases created by the FIWARE GEs.

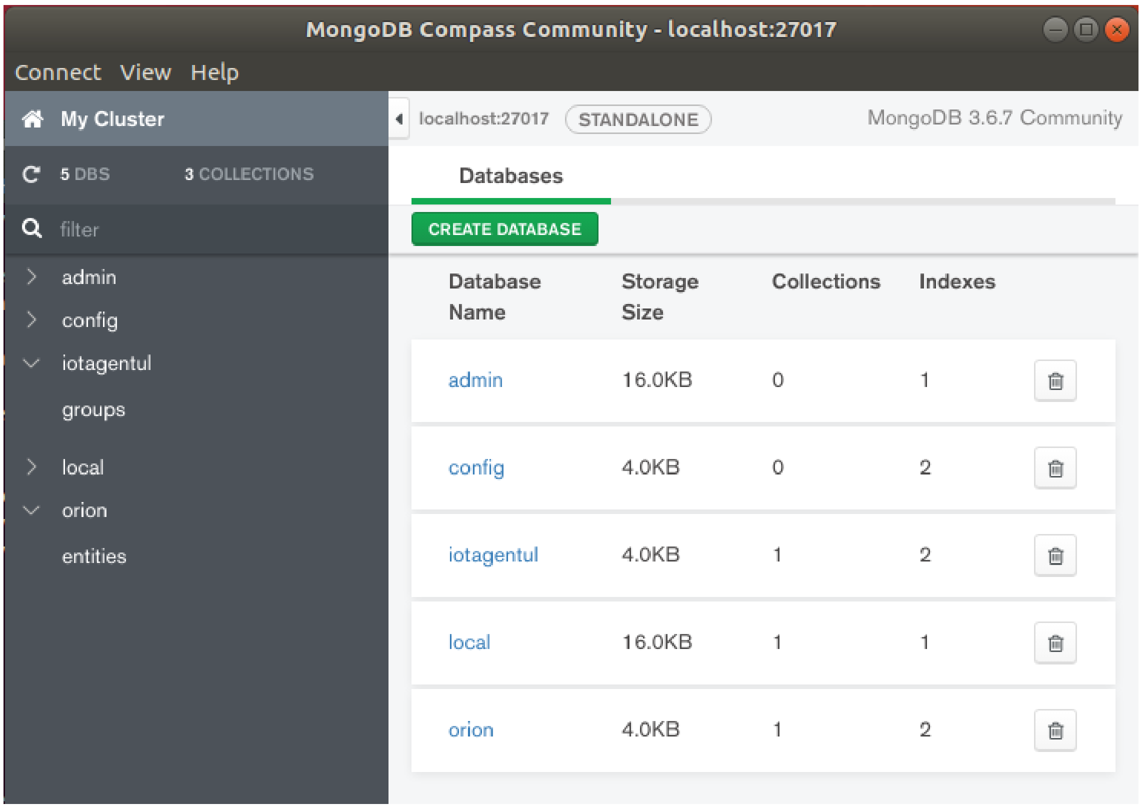

MongoDB Compass Community - Connect
Connect View Help
CREATE FREE ATLAS CLUSTER
Includes 512 MB of data storage.
Learn more
New Connection
Favorites
RECENTS
CONNECT

Connect to Host

Hostname: localhost
Port: 27017
SRV Record
Authentication: None
Replica Set Name:
Read Preference: Primary
SSL: None
SSH Tunnel: None
Favorite Name: e.g. Shared Dev, QA Box, PRODUCTION

Figure 4.6 Connection of Compass to the MongoDB Cluster (MongoDB Docker Container)

To simplify the interaction of the FIWARE components with the MongoDB cluster, and since the FIWARE components that implement security features were used in this system due to various motives, later explained, it was not defined a username and password for the MongoDB cluster. However, it is important to note that databases must always be protected by at least a username and password (there are other and better authentication methods) that also must not be the default ones.



Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	1
config	4.0KB	0	2
iotagentul	4.0KB	1	2
local	16.0KB	1	1
orion	4.0KB	1	2

Figure 4.7 MongoDB Databases

The only FIWARE GEs that automatically create a database at startup are the Orion Context Broker and the IoT Agent, as Cygnus and SHT-Comet must first be configured to receive data. As seen in Figure 4.7 both Orion and the IoT Agent are working as expected as both created the databases (expanded in the figure).

4.2.3 Context Data Management

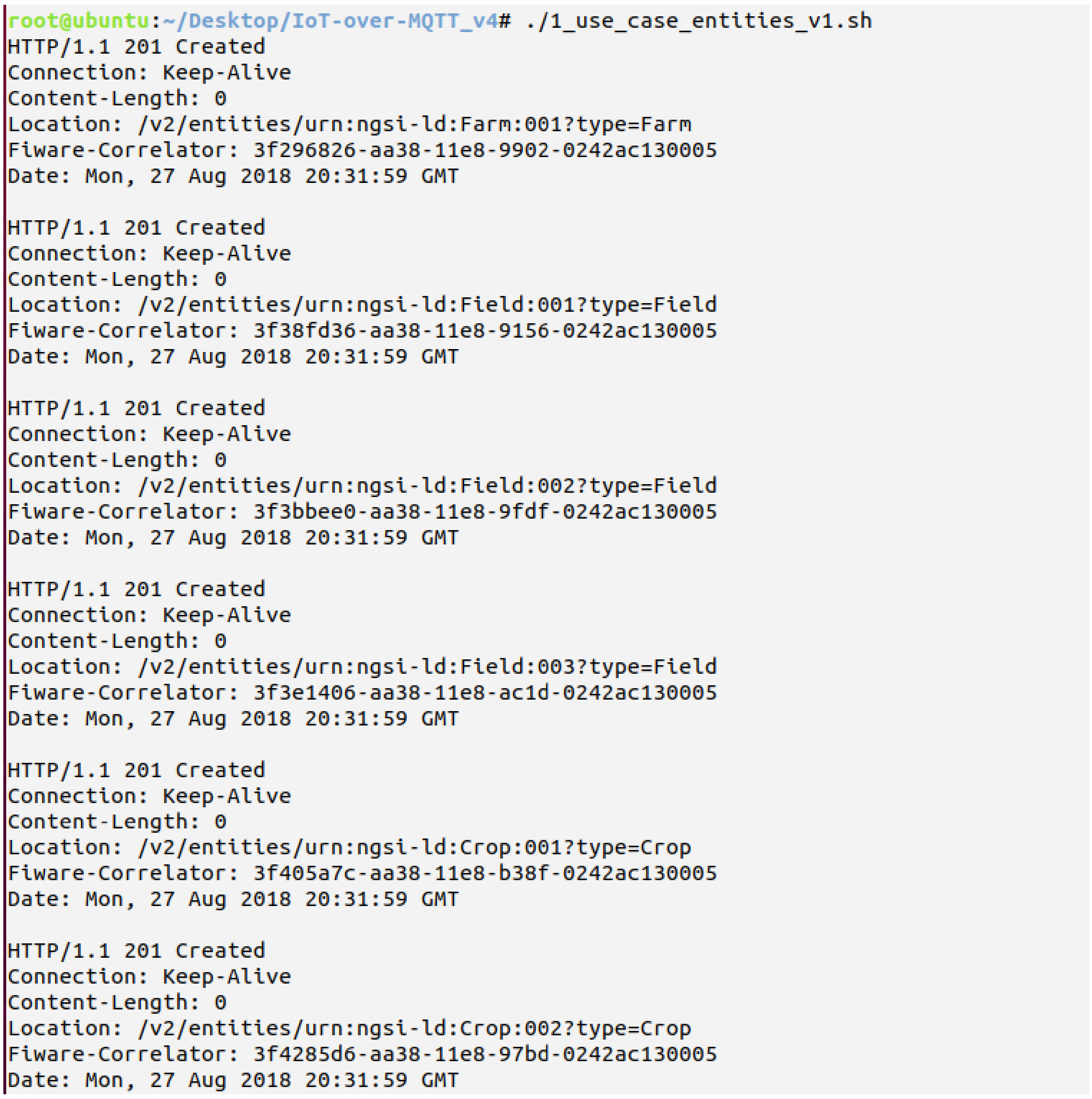
The following tests serve to verify that is possible to manage context information through Orion Context Broker as it is expected. As mentioned before cURL will be used to send information to Orion and Postman will be used to visualize said data in Orion. Through Compass it will be also possible visualize data written in the Orion MongoDB database.

The data here created and sent to Orion is based on the use case.

4.2.3.1 Entities Creation

For this test, a shell script named "1_use_case_entities_v1.sh", was created to execute the cURL commands that create the entities for the farm, for each field (field A, B and C), for each crop (apples, tomatoes and corn), and for each water source (well, tank and borehole)., in one go. The contents of the script are available in Annex I.

The script output (Figure 4.8), that depicts the output of each cURL command, shows that every entity was created successfully.



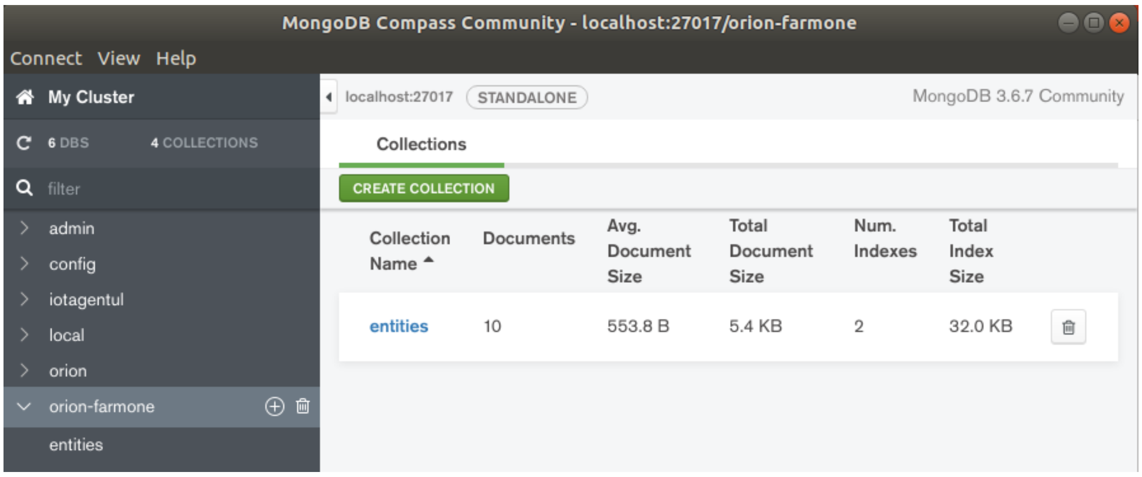
```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./1_use_case_entities_v1.sh
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Farm:001?type=Farm
Fiware-Correlator: 3f296826-aa38-11e8-9902-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Field:001?type=Field
Fiware-Correlator: 3f38fd36-aa38-11e8-9156-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Field:002?type=Field
Fiware-Correlator: 3f3bbee0-aa38-11e8-9fdf-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Field:003?type=Field
Fiware-Correlator: 3f3e1406-aa38-11e8-ac1d-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Crop:001?type=Crop
Fiware-Correlator: 3f405a7c-aa38-11e8-b38f-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Crop:002?type=Crop
Fiware-Correlator: 3f4285d6-aa38-11e8-97bd-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT
```

Figure 4.8 Output of the Commands That Created the Entities – Part 1

```
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Crop:003?type=Crop
Fiware-Correlator: 3f44c7d8-aa38-11e8-9721-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Well:001?type=Well
Fiware-Correlator: 3f470f16-aa38-11e8-a00e-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Tank:001?type=Tank
Fiware-Correlator: 3f495640-aa38-11e8-a84d-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/entities/urn:ngsi-ld:Borehole:001?type=Borehole
Fiware-Correlator: 3f4b77ae-aa38-11e8-8320-0242ac130005
Date: Mon, 27 Aug 2018 20:31:59 GMT
```

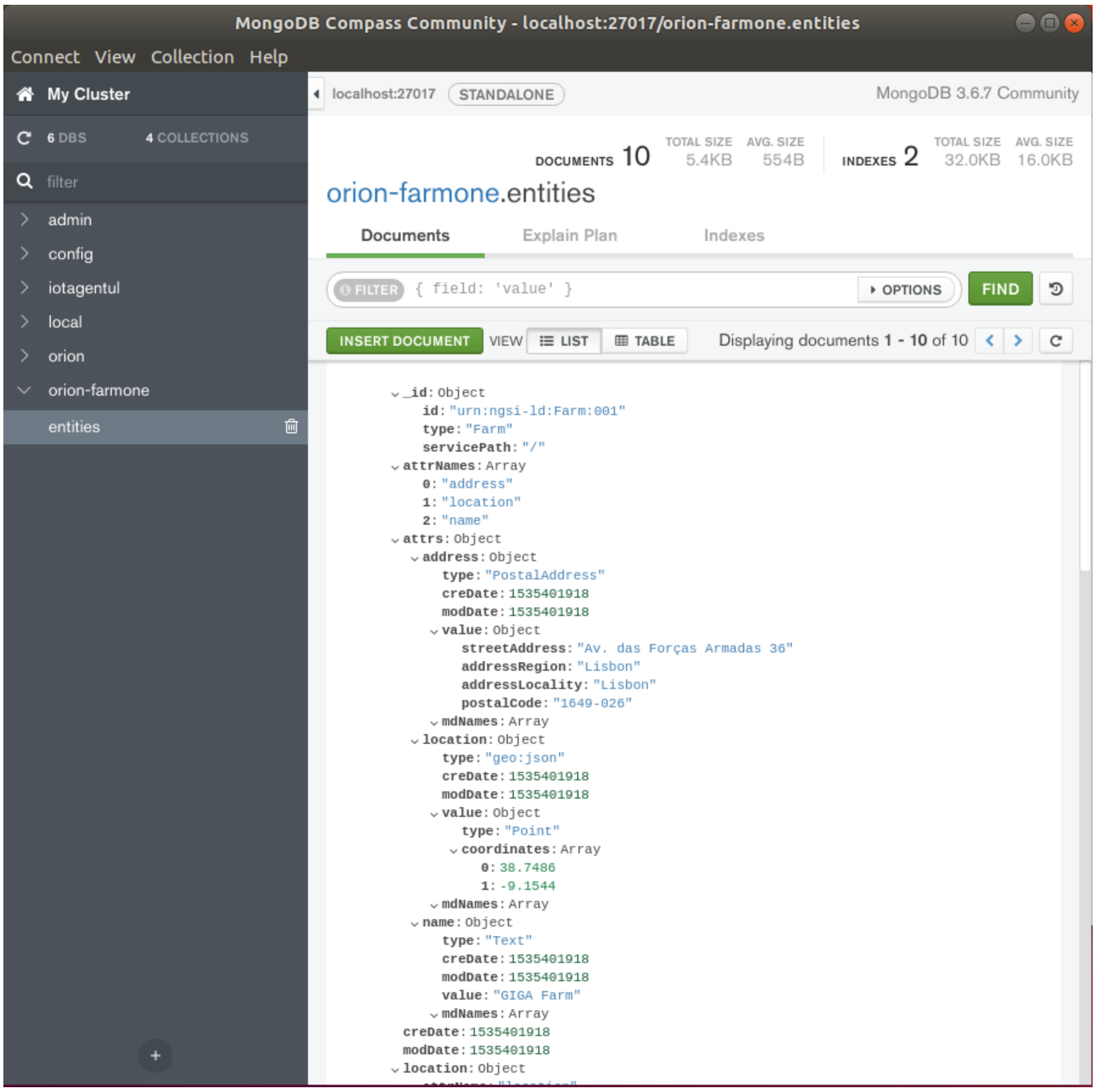
Figure 4.8 Output of the Commands That Created the Entities – Part 2

By updating the MongoDB GUI, Compass, it is possible to verify (Figure 4.9 and Figure 4.10) that a new database named "orion-farmone" (orion + fiware-service) was created. All commands sent to Orion must have the header that specifies the "fiware-service" otherwise nothing will be returned as the command is sent to the database name "orion" (considered the root) which is empty.


MongoDB Compass Community - localhost:27017/orion-farmone

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
entities	10	553.8 B	5.4 KB	2	32.0 KB

Figure 4.9 MongoDB Database with the Created Entities


MongoDB Compass Community - localhost:27017/orion-farmone.entities
Connect View Collection Help
My Cluster
6 DBS 4 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
entities
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 10 TOTAL SIZE 5.4KB AVG. SIZE 554B
INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB
orion-farmone.entities
Documents
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 10 of 10

```
_id: Object
  id: "urn:ngsi-ld:Farm:001"
  type: "Farm"
  servicePath: "/"
attrNames: Array
  0: "address"
  1: "location"
  2: "name"
attrs: Object
  address: Object
    type: "PostalAddress"
    creDate: 1535401918
    modDate: 1535401918
    value: Object
      streetAddress: "Av. das Forças Armadas 36"
      addressRegion: "Lisbon"
      addressLocality: "Lisbon"
      postalCode: "1649-026"
    mdNames: Array
  location: Object
    type: "geo:json"
    creDate: 1535401918
    modDate: 1535401918
    value: Object
      type: "Point"
      coordinates: Array
        0: 38.7486
        1: -9.1544
    mdNames: Array
  name: Object
    type: "Text"
    creDate: 1535401918
    modDate: 1535401918
    value: "GIGA Farm"
    mdNames: Array
creDate: 1535401918
modDate: 1535401918
location: Object
```

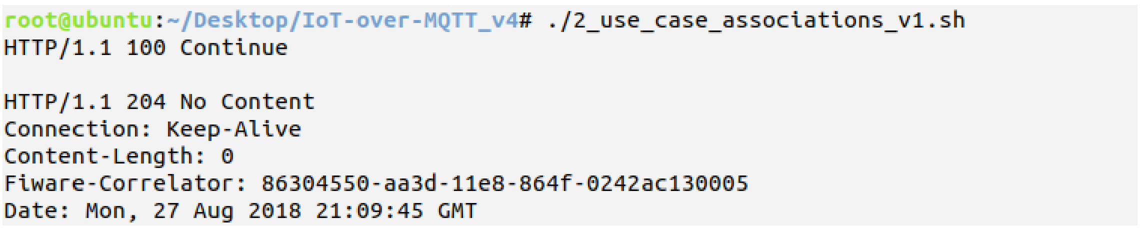
Figure 4.10 Visualization of an Entity Details

It was then concluded that the creation of entities is working as expected.

4.2.3.2 Entities Association

For this test, a shell script named "2_use_case_associations_v1.sh", was created to execute the cURL commands that do the associations of the previous created entities, in one go. The contents of the script are available in Annex J.

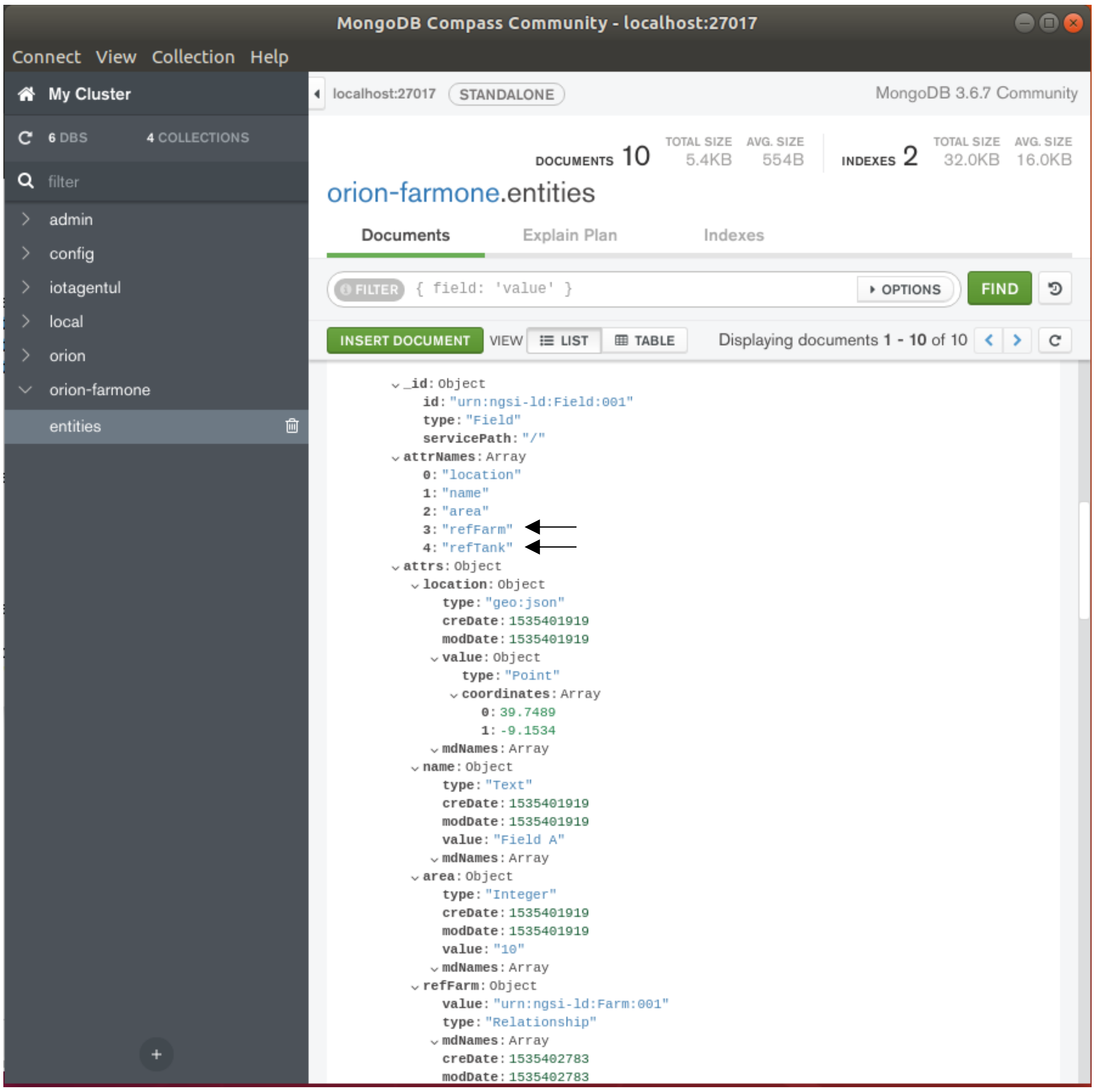
As the association attribute of each entity was not specified at the creation time but after, the association tasks done by the script are also considered entity modifications. The script output (Figure 4.11), that depicts the output of each cURL command, shows that every entity was created successfully.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./2_use_case_associations_v1.sh
HTTP/1.1 100 Continue

HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: 86304550-aa3d-11e8-864f-0242ac130005
Date: Mon, 27 Aug 2018 21:09:45 GMT
```

Figure 4.11 Output of the Command That Associated the Entities

By viewing the Field:001 entity using Compass (Figure 4.12) it is observed that two attributes ("refFarm" and "refTank") were added as expected.

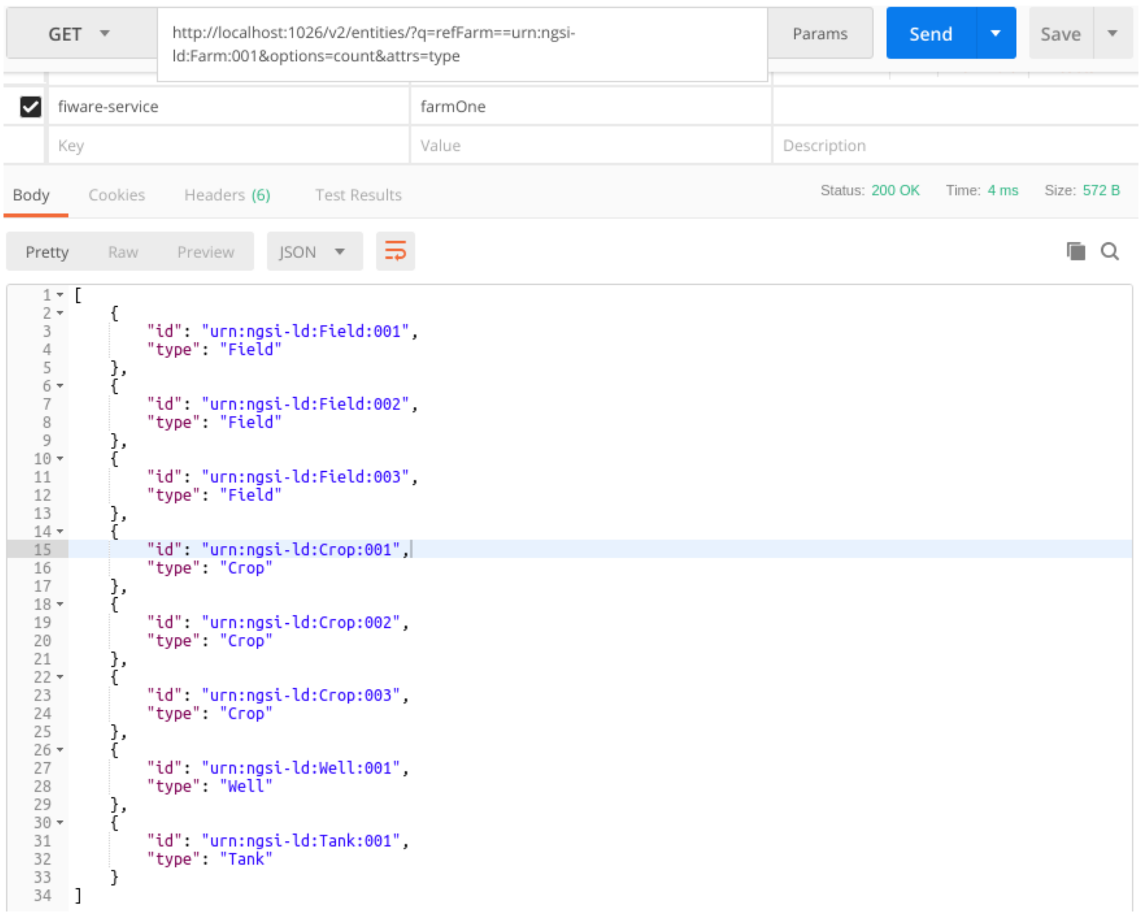


MongoDB Compass Community - localhost:27017
Connect View Collection Help
My Cluster
6 DBS 4 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
entities
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 10 TOTAL SIZE 5.4KB AVG. SIZE 554B
INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB
orion-farmone.entities
Documents Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE
Displaying documents 1 - 10 of 10

```
_id: Object
    id: "urn:ngsi-ld:Field:001"
    type: "Field"
    servicePath: "/"
attrNames: Array
    0: "location"
    1: "name"
    2: "area"
    3: "refFarm"
    4: "refTank"
attrs: Object
  location: Object
      type: "geo:json"
      creDate: 1535401919
      modDate: 1535401919
    value: Object
        type: "Point"
      coordinates: Array
          0: 39.7489
          1: -9.1534
    mdNames: Array
  name: Object
      type: "Text"
      creDate: 1535401919
      modDate: 1535401919
      value: "Field A"
    mdNames: Array
  area: Object
      type: "Integer"
      creDate: 1535401919
      modDate: 1535401919
      value: "10"
    mdNames: Array
  refFarm: Object
      value: "urn:ngsi-ld:Farm:001"
      type: "Relationship"
    mdNames: Array
      creDate: 1535402783
      modDate: 1535402783
```

Figure 4.12 Visualization of Entity Details

It is also possible to observe the associations by making a query asking for all entities related to an entity. In Figure 4.13 a query is made for all entities related to the Farm entity.



```
http://localhost:1026/v2/entities/?q=refFarm==urn:ngsi-ld:Farm:001&options=count&attrs=type
```

Key	Value	Description
fiware-service	farmOne	

Status: 200 OK Time: 4 ms Size: 572 B

```
[
    {
        "id": "urn:ngsi-ld:Field:001",
        "type": "Field"
    },
    {
        "id": "urn:ngsi-ld:Field:002",
        "type": "Field"
    },
    {
        "id": "urn:ngsi-ld:Field:003",
        "type": "Field"
    },
    {
        "id": "urn:ngsi-ld:Crop:001",
        "type": "Crop"
    },
    {
        "id": "urn:ngsi-ld:Crop:002",
        "type": "Crop"
    },
    {
        "id": "urn:ngsi-ld:Crop:003",
        "type": "Crop"
    },
    {
        "id": "urn:ngsi-ld:Well:001",
        "type": "Well"
    },
    {
        "id": "urn:ngsi-ld:Tank:001",
        "type": "Tank"
    }
]
```

Figure 4.13 Query for All Entities Associated with the Farm Entity

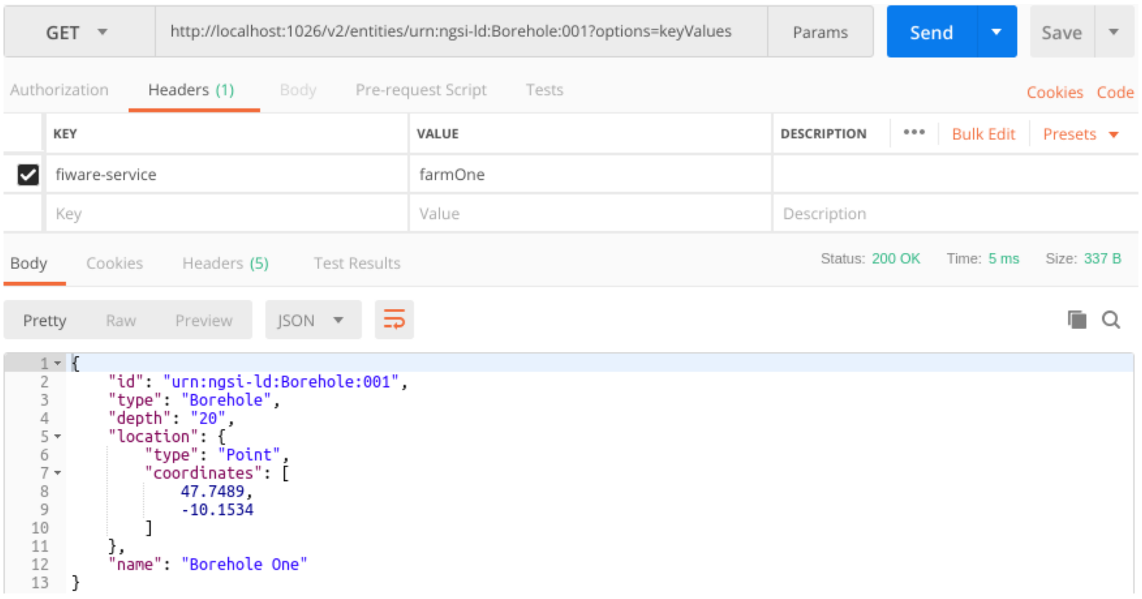
It was then concluded that the association and alteration of entities is working.

4.2.3.3 Entities Modification

Although it was already proved by the previous test that modifying entities is working a new test is performed.

For this test, a shell script named "3_use_case_entities_modification_(1)_v1.sh", was created to modify a single attribute of an entity, and a script named "3_use_case_entities_modification_(1)_v1.sh" to modify several attributes of an entity, which in in this test is the borehole entity, that was created specifically for this. The contents of the scripts are available in Annex K.

Figure 4.14 shows the Borehole entity before changes were made.


GET http://localhost:1026/v2/entities/urn:ngsi-ld:Borehole:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	farmOne	
Key	Value	Description

Status: 200 OK Time: 5 ms Size: 337 B

```
{
    "id": "urn:ngsi-ld:Borehole:001",
    "type": "Borehole",
    "depth": "20",
    "location": {
        "type": "Point",
        "coordinates": [
            47.7489,
            -10.1534
        ]
    },
    "name": "Borehole One"
}
```

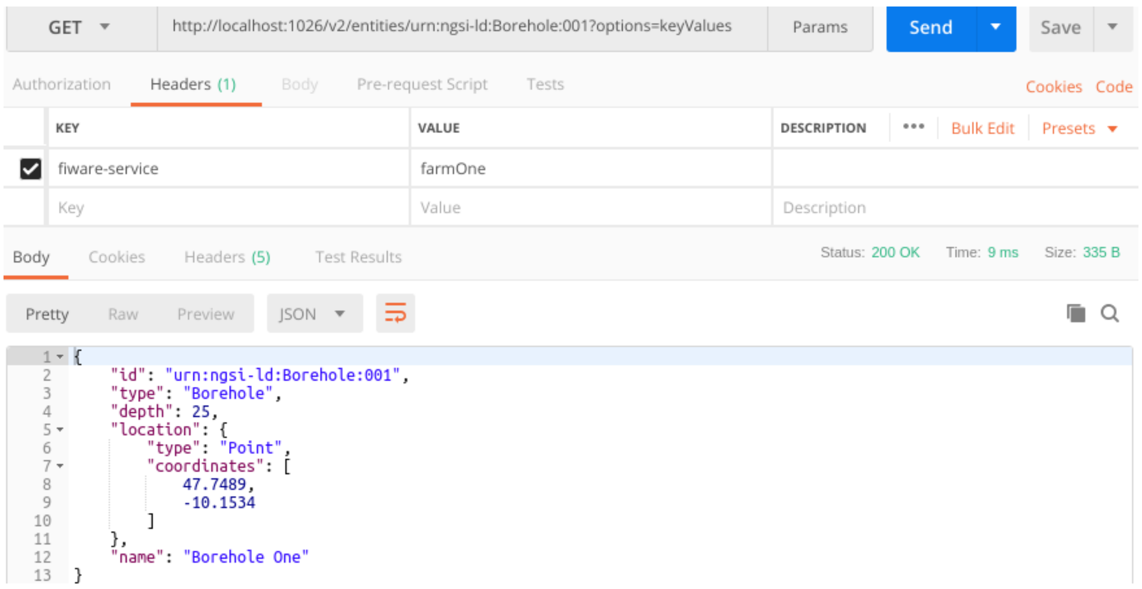
Figure 4.14 Borehole Entity Key Values Before Changes

Figure 4.15 shows the output of the execution of the first script

```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./3_use_case_entities_modification_\(1\)_v1.
sh
HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: 55b82226-aae7-11e8-b95a-0242ac130005
Date: Tue, 28 Aug 2018 17:25:18 GMT
```

Figure 4.15 Output of the Command That Modified an Entity (1)

Figure 4.16 shows that the depth attribute was altered from 20 to 25 as expected.


GET http://localhost:1026/v2/entities/urn:ngsi-ld:Borehole:001?options=keyValues

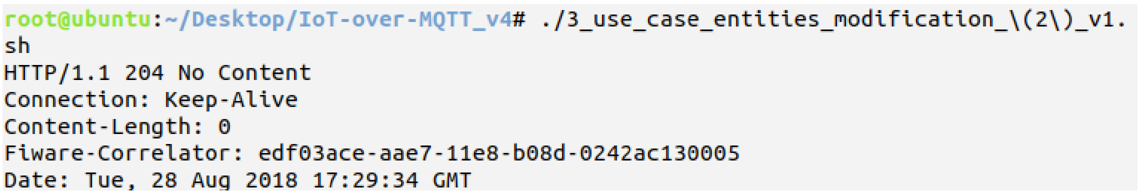
KEY	VALUE	DESCRIPTION
✓ fiware-service	farmOne	
Key	Value	Description

Status: 200 OK Time: 9 ms Size: 335 B

```
{
    "id": "urn:ngsi-ld:Borehole:001",
    "type": "Borehole",
    "depth": 25,
    "location": {
        "type": "Point",
        "coordinates": [
            47.7489,
            -10.1534
        ]
    },
    "name": "Borehole One"
}
```

Figure 4.16 Borehole Entity Key Values After Changes (First Script)

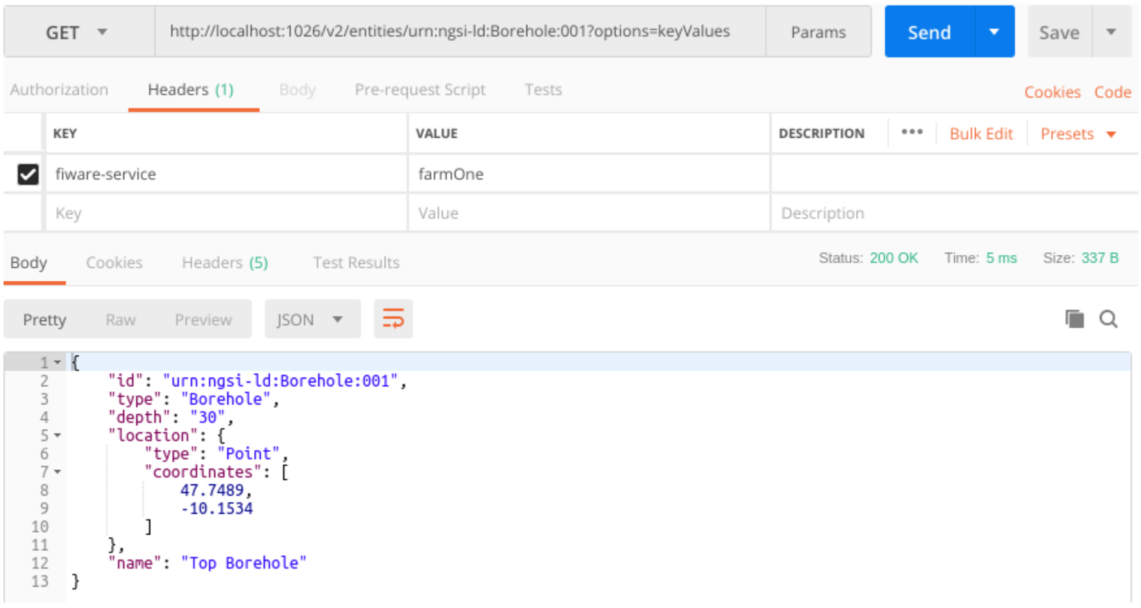
Figure 4.17 shows the output of the execution of the second script



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./3_use_case_entities_modification_\(2\)_v1.
sh
HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: edf03ace-aae7-11e8-b08d-0242ac130005
Date: Tue, 28 Aug 2018 17:29:34 GMT
```

Figure 4.17 Output of the Command That Modified an Entity (2)

Figure 4.18 shows that the depth attribute was altered from 25 to 30 and the name attribute from “Borehole One” to “Top Borehole” as expected.



GET http://localhost:1026/v2/entities/urn:ngsi-ld:Borehole:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	farmOne	
Key	Value	Description

Status: 200 OK Time: 5 ms Size: 337 B

```
{
    "id": "urn:ngsi-ld:Borehole:001",
    "type": "Borehole",
    "depth": "30",
    "location": {
        "type": "Point",
        "coordinates": [
            47.7489,
            -10.1534
        ]
    },
    "name": "Top Borehole"
}
```

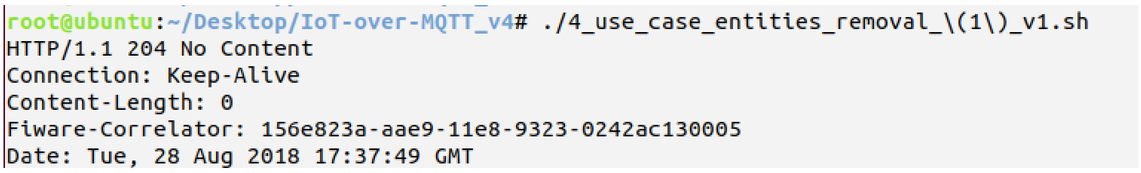
Figure 4.18 Borehole Entity Key Values After Changes (Second Script)

Therefore, it was concluded that the modification of entities is working.

4.2.3.4 Entities Removal

For this test, two shell scripts named “4_use_case_entities_removal_(1)_v1.sh” and “4_use_case_entities_removal_(2)_v1.sh”, were created to test the removal of an entity attribute and the removal of an entity. The contents of the scripts are available in Annex L.

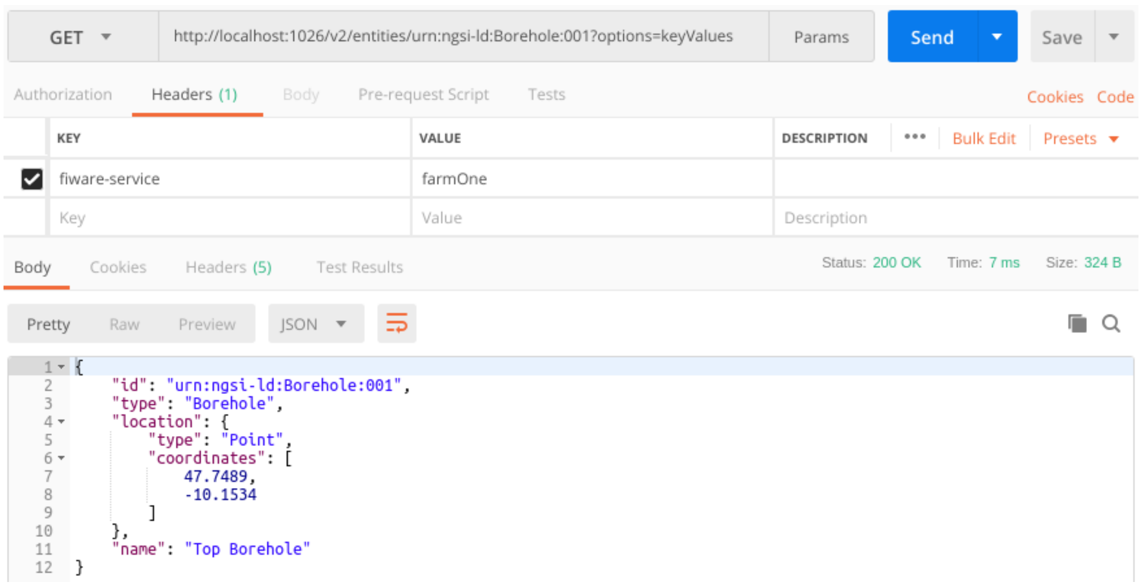
Figure 4.19 shows the output of the execution of the first script.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./4_use_case_entities_removal_\(1\)_v1.sh
HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: 156e823a-aae9-11e8-9323-0242ac130005
Date: Tue, 28 Aug 2018 17:37:49 GMT
```

Figure 4.19 Output of the Command That Removed an Entity Attribute

Figure 4.20 shows that the depth attribute was removed as expected.

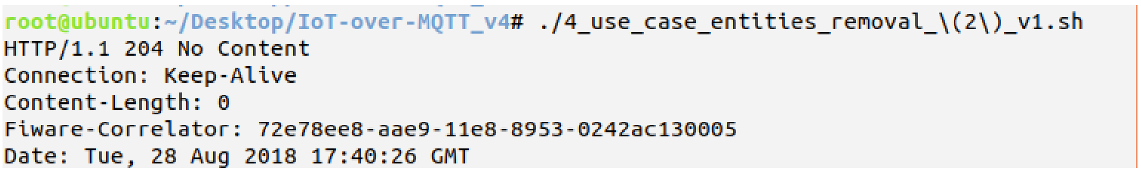


KEY	VALUE	DESCRIPTION
fiware-service	farmOne	
Key	Value	Description

```
{
    "id": "urn:ngsi-ld:Borehole:001",
    "type": "Borehole",
    "location": {
        "type": "Point",
        "coordinates": [
            47.7489,
            -10.1534
        ]
    },
    "name": "Top Borehole"
}
```

Figure 4.20 Borehole Entity Key Values After Removal of an Attribute (First Script)

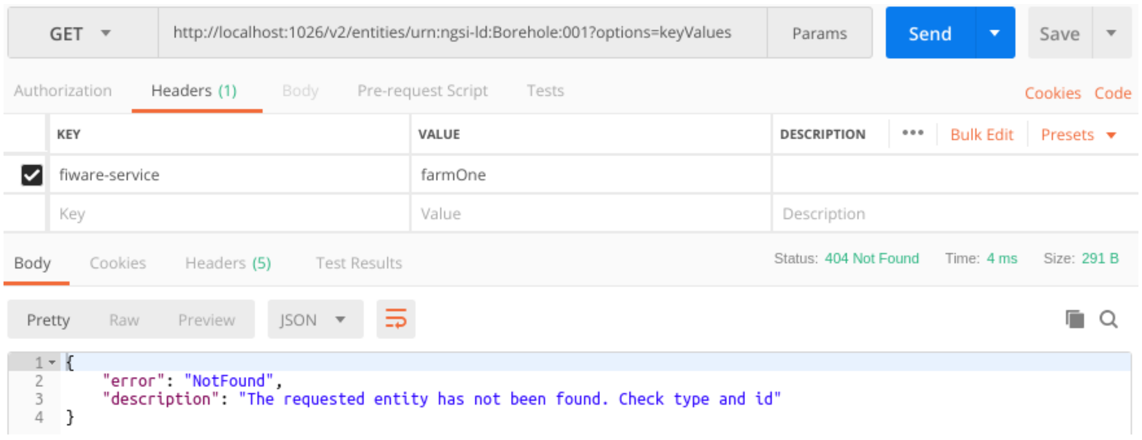
Figure 4.21 shows the output of the execution of the second script.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./4_use_case_entities_removal_\(2\)_v1.sh
HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: 72e78ee8-aae9-11e8-8953-0242ac130005
Date: Tue, 28 Aug 2018 17:40:26 GMT
```

Figure 4.21 Output of the Command That Removed an Entity (1)

Figure 4.22 shows that the entity was deleted as expected, as an error was returned.



GET http://localhost:1026/v2/entities/urn:ngsi-ld:Borehole:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	farmOne	
Key	Value	Description

Status: 404 Not Found Time: 4 ms Size: 291 B

```
{
    "error": "NotFound",
    "description": "The requested entity has not been found. Check type and id"
}
```

Figure 4.22 A Query for the Borehole Entity Returns a "Not Found" Error

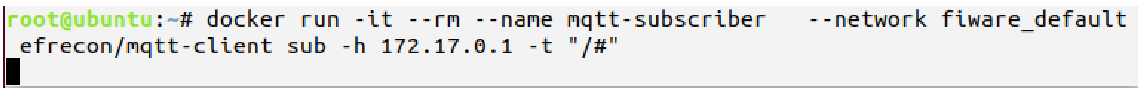
Therefore, it was concluded that the removal of attributes and entities was working.

4.2.4 Mosquitto MQTT Broker Health Check

To test if the Mosquitto MQTT Broker is working properly, a pair of dummy MQTT publisher/subscriber are used. The subscriber is configured to receive all messages independently of the topic sent by the publisher. If everything is working well as intended, then the subscriber will receive all messages.

The publisher and subscriber are used as Docker Containers and the Mosquitto MQTT Broker was already created when the system was initiated.

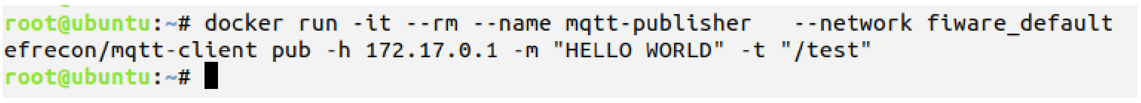
The subscriber (Figure 4.23) is initiated and stays waiting for messages.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber   --network fiware_default
 efrecon/mqtt-client sub -h 172.17.0.1 -t "/#"
```

Figure 4.23 Creation of the MQTT Subscriber

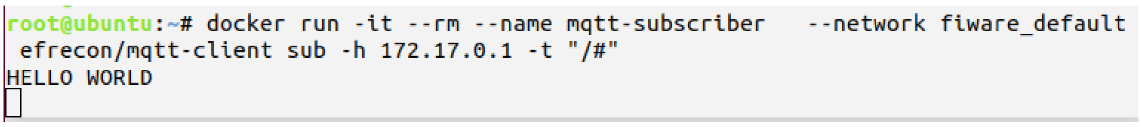
Then using a publisher (Figure 4.24) a message is sent.



```
root@ubuntu:~# docker run -it --rm --name mqtt-publisher   --network fiware_default
efrecon/mqtt-client pub -h 172.17.0.1 -m "HELLO WORLD" -t "/test"
root@ubuntu:~#
```

Figure 4.24 Creation of the MQTT Publisher and Sending of a Message

The message is received by the subscriber (Figure 4.25) proving that the Broker is working well.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber   --network fiware_default
 efrecon/mqtt-client sub -h 172.17.0.1 -t "/#"
HELLO WORLD
```

Figure 4.25 Message Received by the MQTT Publisher

4.2.5 IoT Devices Management

The following tests serve to verify that is possible to manage IoT Devices and associated services as it is expected. Like before, cURL will be used to send information and Postman will be used to visualize data. Through Compass it will be also possible visualize data written in the databases.

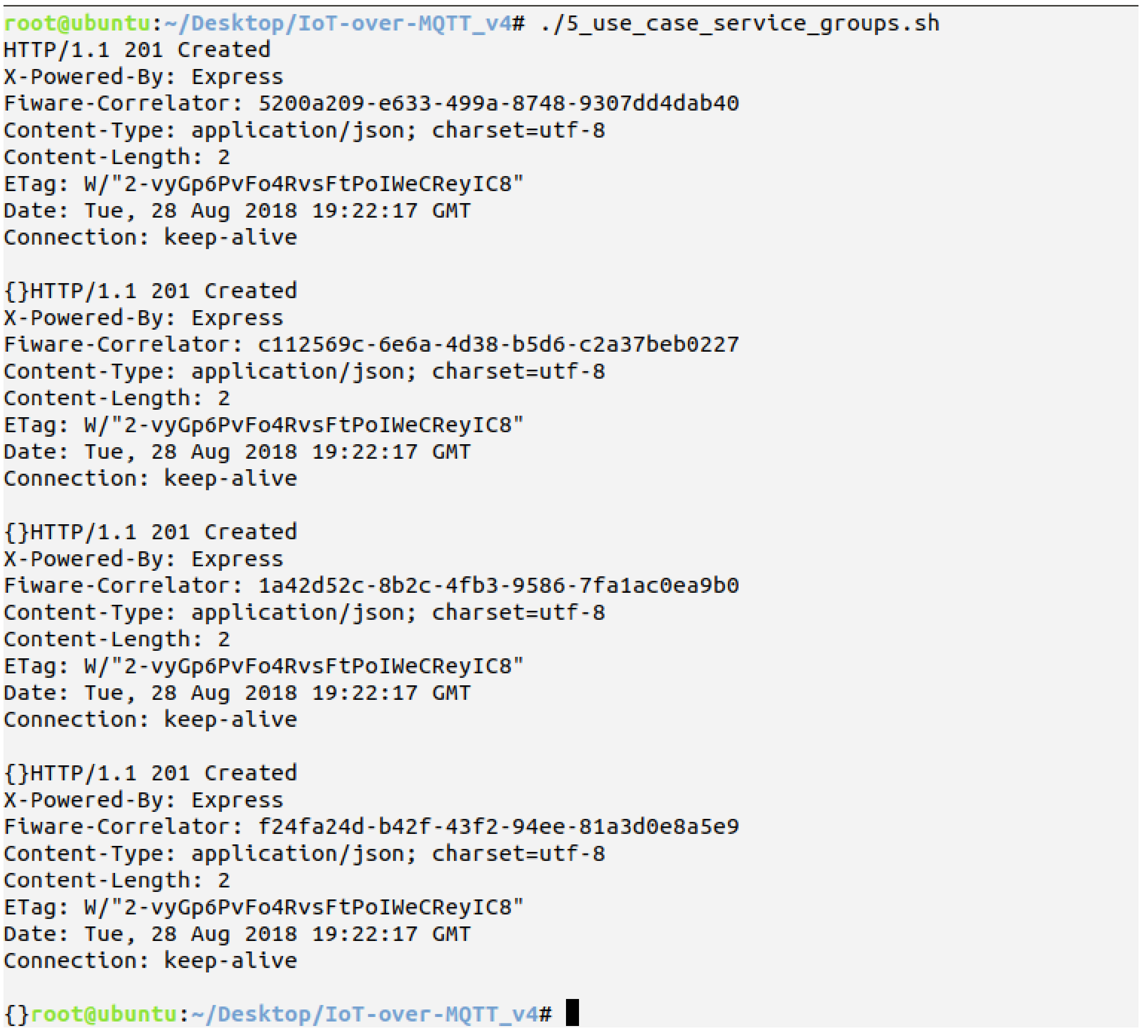
The data here created is based on the use case.

4.2.5.1 Service Group Provisioning

For this use case four "fiware-servicepaths" where created to differentiate between arrays of devices, two paths for the field A devices (sensors path: "/fieldA/sensors"; actuators path: "/fieldA/actuators") and two paths for the tank devices (sensors path: "/tank/sensors"; actuators path: "/tank/actuators"). Therefore, it is necessary to provision four corresponding service groups, each with a different API key otherwise an error will be returned. For this and the following tests a different "fiware-service" was used to easily observe what is created during the IoT devices set-up.

To provision the service groups in one go, a shell scrip named "5_use_case_service_groups.sh" was created, whose contents are available in Annex M.

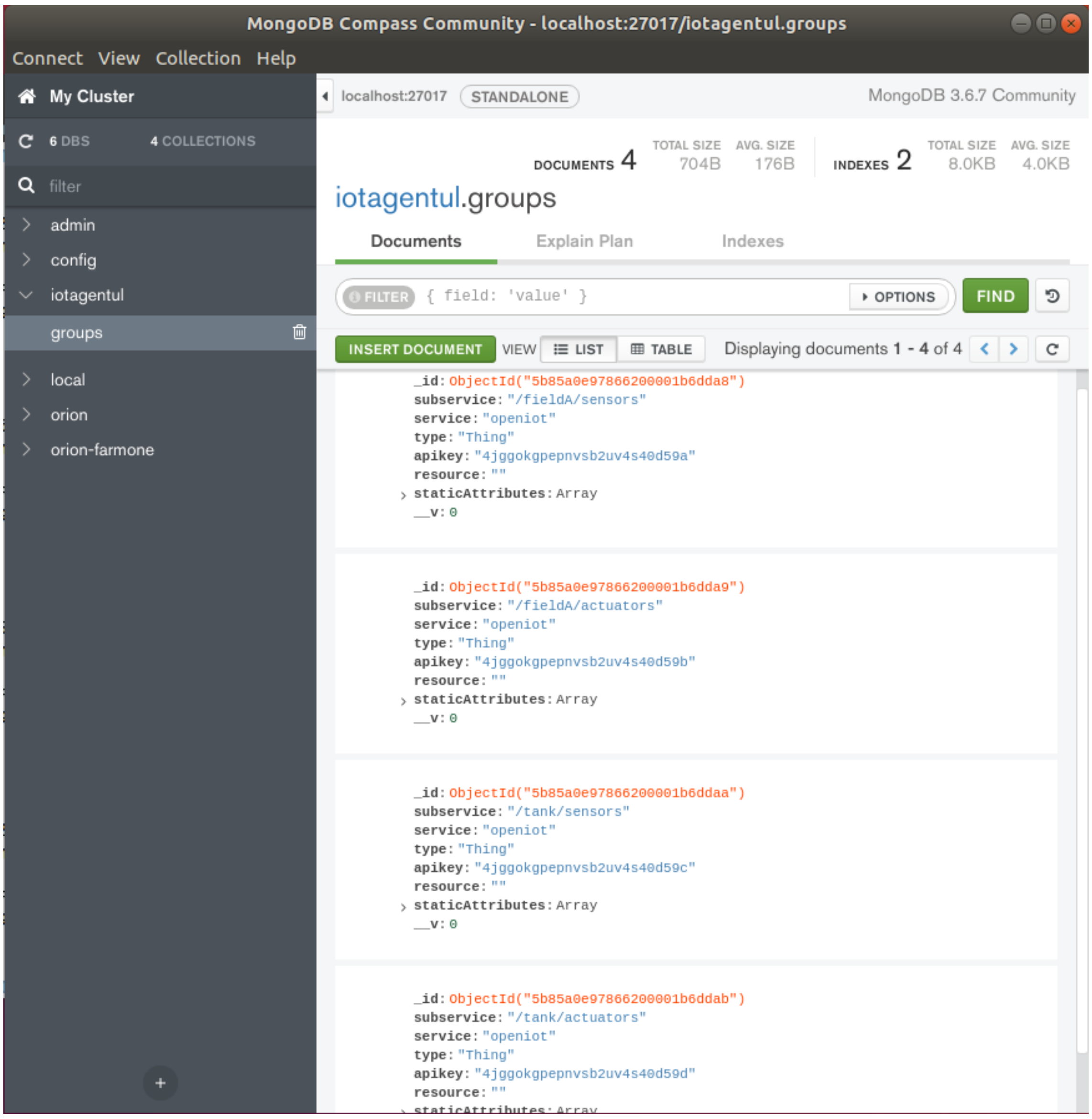
The output of the script execution (Figure 4.26) indicates that the service groups were created successfully.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./5_use_case_service_groups.sh
HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 5200a209-e633-499a-8748-9307dd4dab40
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:22:17 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: c112569c-6e6a-4d38-b5d6-c2a37beb0227
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:22:17 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 1a42d52c-8b2c-4fb3-9586-7fa1ac0ea9b0
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:22:17 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: f24fa24d-b42f-43f2-94ee-81a3d0e8a5e9
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:22:17 GMT
Connection: keep-alive

{}root@ubuntu:~/Desktop/IoT-over-MQTT_v4#
```

Figure 4.26 Output of the Commands That Created the Service Groups

In Figure 4.27 it is possible to observe the created service groups in the database associated to the IoT Agent.



```
  _id: ObjectId("5b85a0e97866200001b6dda8")
  subservice: "/fieldA/sensors"
  service: "openiot"
  type: "Thing"
  apikey: "4jggokgpepnvsb2uv4s40d59a"
  resource: ""
> staticAttributes: Array
  __v: 0

  _id: ObjectId("5b85a0e97866200001b6dda9")
  subservice: "/fieldA/actuators"
  service: "openiot"
  type: "Thing"
  apikey: "4jggokgpepnvsb2uv4s40d59b"
  resource: ""
> staticAttributes: Array
  __v: 0

  _id: ObjectId("5b85a0e97866200001b6ddaa")
  subservice: "/tank/sensors"
  service: "openiot"
  type: "Thing"
  apikey: "4jggokgpepnvsb2uv4s40d59c"
  resource: ""
> staticAttributes: Array
  __v: 0

  _id: ObjectId("5b85a0e97866200001b6ddab")
  subservice: "/tank/actuators"
  service: "openiot"
  type: "Thing"
  apikey: "4jggokgpepnvsb2uv4s40d59d"
  resource: ""
> staticAttributes: Array
```

Figure 4.27 Service Groups Created

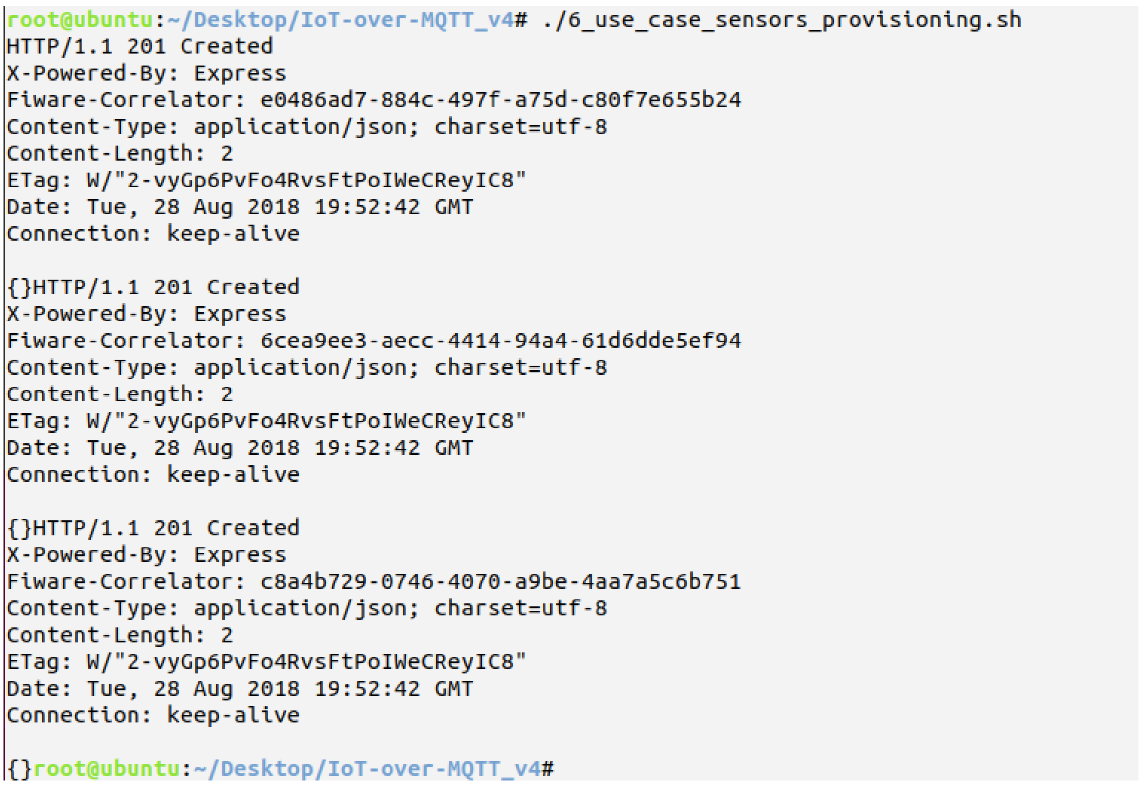
It was then concluded that the provisioning of service groups is working as expected.

4.2.5.2 Sensors Provisioning

As mentioned at the beginning of this chapter, a total of five IoT Devices are used, three on field A (a weather station, an earth-humidity sensor and a valve) and two on the tank (a water level sensor and a valve). In this test the sensors are provision in order for the system to be able to map the receiving data into the right context entities and therefore be able to know what is the received data. In the next test, the actuators are provision.

To provision the sensors in one go, a shell scrip named “6_use_case_sensors_provisioning.sh” was created, whose contents are available in Annex N.

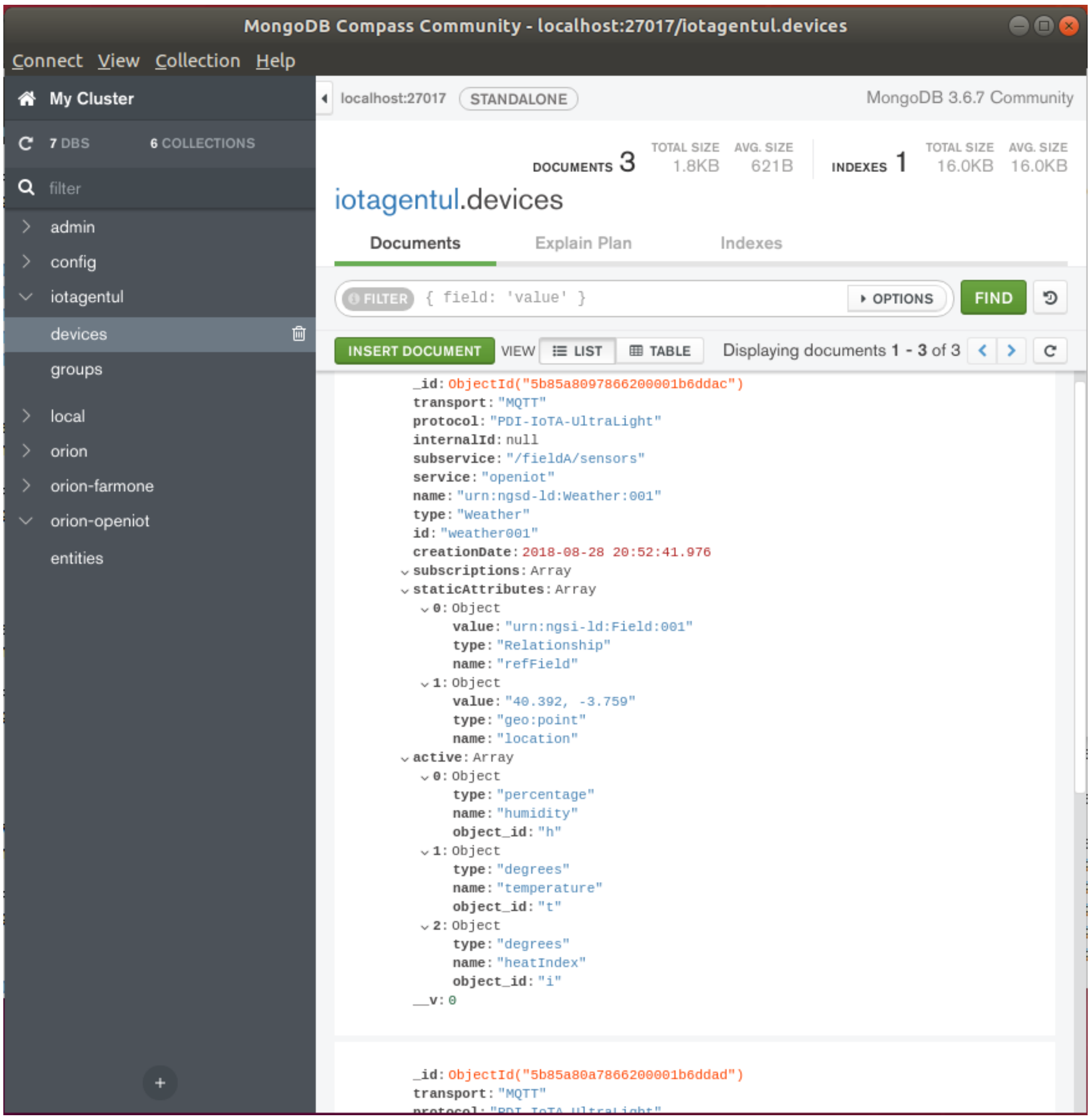
The output of the script execution (Figure 4.28) indicates that the provisioning of the sensors was done successfully. It is also possible to observe that is was possible to make an association of the earth-humidity sensor to an apple tree even though an entity for said tree was not created, proving that there is no data integrity in a MongoDB database.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./6_use_case_sensors_provisioning.sh
HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: e0486ad7-884c-497f-a75d-c80f7e655b24
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:52:42 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 6cea9ee3-aecc-4414-94a4-61d6dde5ef94
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:52:42 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: c8a4b729-0746-4070-a9be-4aa7a5c6b751
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 19:52:42 GMT
Connection: keep-alive

{}root@ubuntu:~/Desktop/IoT-over-MQTT_v4#
```

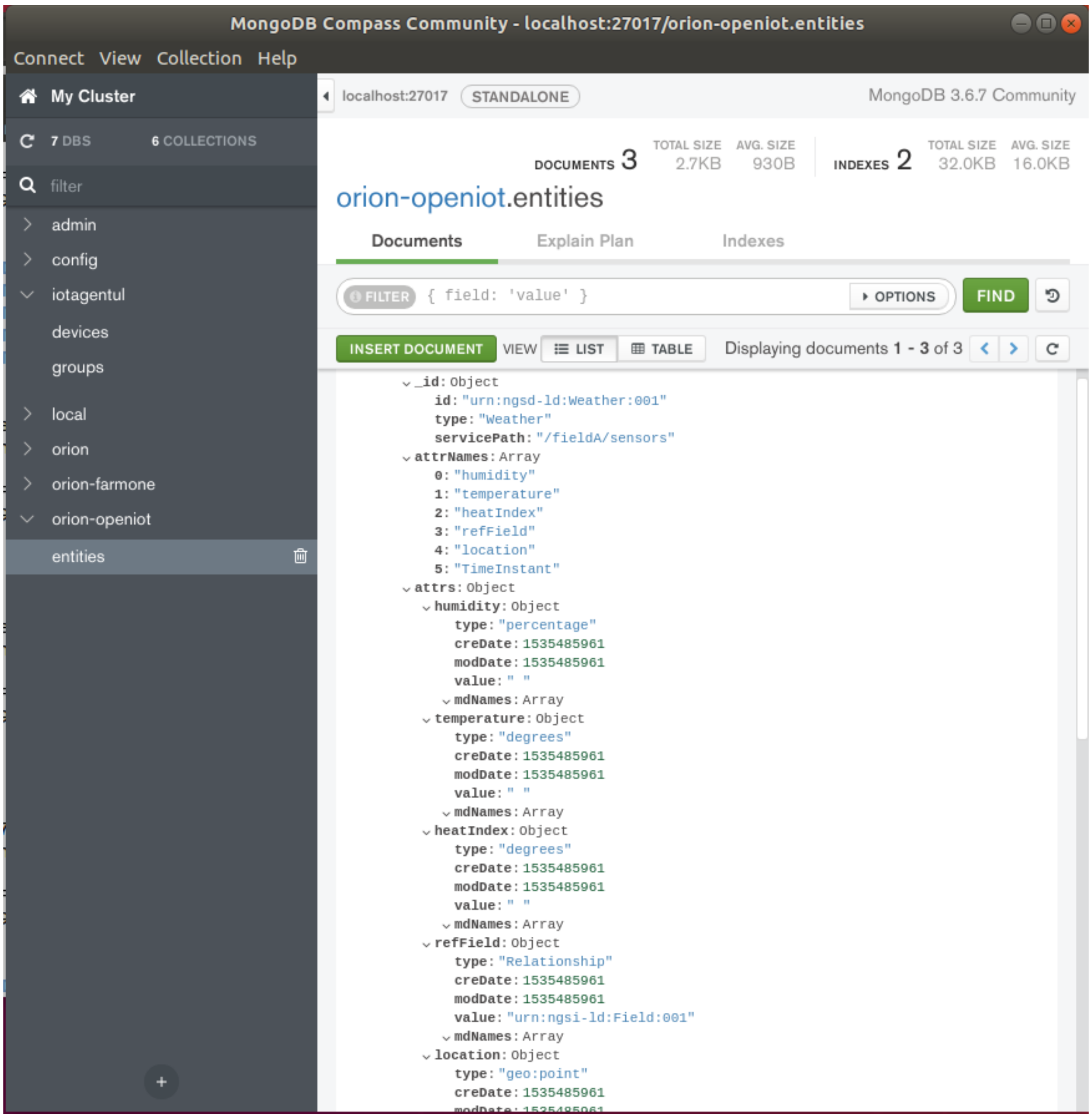
Figure 4.28 Output of the Commands That Provisioned the Sensors

In Figure 4.29 is observed that in the database managed by the IoT Agent a new directory for devices was created, and in Figure 4.30, that a new Orion database for the “fiware-service” used in this test was also created, which contains the devices entities. This are the entities that are queried when it is necessary to obtain information, and not the devices in the IoT Agent.


MongoDB Compass Community - localhost:27017/iotagentul.devices
Connect View Collection Help
My Cluster
7 DBS 6 COLLECTIONS
filter
admin
config
iotagentul
devices
groups
local
orion
orion-farmone
orion-openiot
entities
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 3 TOTAL SIZE 1.8KB AVG. SIZE 621B
INDEXES 1 TOTAL SIZE 16.0KB AVG. SIZE 16.0KB
iotagentul.devices
Documents
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 3 of 3

```
    _id: ObjectId("5b85a8097866200001b6ddac")
    transport: "MQTT"
    protocol: "PDI-IoTA-UltraLight"
    internalId: null
    subservice: "/fieldA/sensors"
    service: "openiot"
    name: "urn:ngsd-ld:Weather:001"
    type: "Weather"
    id: "weather001"
    creationDate: 2018-08-28 20:52:41.976
  ⌄ subscriptions: Array
  ⌄ staticAttributes: Array
     ⌄ 0: Object
          value: "urn:ngsi-ld:Field:001"
          type: "Relationship"
          name: "refField"
     ⌄ 1: Object
          value: "40.392, -3.759"
          type: "geo:point"
          name: "location"
  ⌄ active: Array
     ⌄ 0: Object
          type: "percentage"
          name: "humidity"
          object_id: "h"
     ⌄ 1: Object
          type: "degrees"
          name: "temperature"
          object_id: "t"
     ⌄ 2: Object
          type: "degrees"
          name: "heatIndex"
          object_id: "i"
    __v: 0

    _id: ObjectId("5b85a80a7866200001b6ddad")
    transport: "MQTT"
    protocol: "PDI-IoTA-UltraLight"
```

Figure 4.29 Provisioned Sensors in the IoT Agent


MongoDB Compass Community - localhost:27017/orion-openiot.entities
Connect View Collection Help
My Cluster
7 DBS 6 COLLECTIONS
filter
admin
config
iotagentul
devices
groups
local
orion
orion-farmone
orion-openiot
entities
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 3 TOTAL SIZE 2.7KB AVG. SIZE 930B
INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB
orion-openiot.entities
Documents
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 3 of 3

```
⌄_id: Object
    id: "urn:ngsd-ld:Weather:001"
    type: "Weather"
    servicePath: "/fieldA/sensors"
⌄attrNames: Array
    0: "humidity"
    1: "temperature"
    2: "heatIndex"
    3: "refField"
    4: "location"
    5: "TimeInstant"
⌄attrs: Object
  ⌄humidity: Object
      type: "percentage"
      creDate: 1535485961
      modDate: 1535485961
      value: " "
    ⌄mdNames: Array
  ⌄temperature: Object
      type: "degrees"
      creDate: 1535485961
      modDate: 1535485961
      value: " "
    ⌄mdNames: Array
  ⌄heatIndex: Object
      type: "degrees"
      creDate: 1535485961
      modDate: 1535485961
      value: " "
    ⌄mdNames: Array
  ⌄refField: Object
      type: "Relationship"
      creDate: 1535485961
      modDate: 1535485961
      value: "urn:ngsi-ld:Field:001"
    ⌄mdNames: Array
  ⌄location: Object
      type: "geo:point"
      creDate: 1535485961
      modDate: 1535485961
```

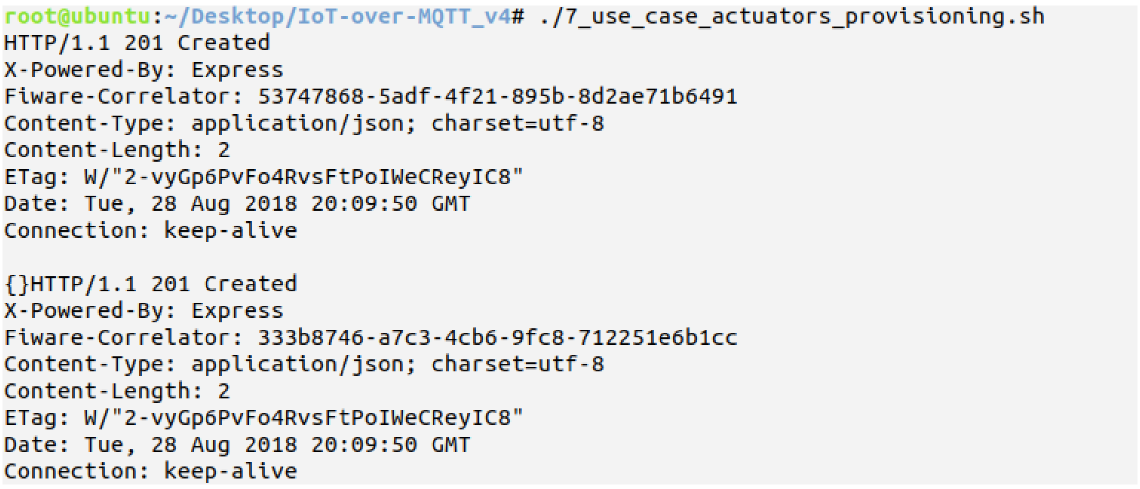
Figure 4.30 Provisioned Sensors Entities in Orion

4.2.5.3 Actuators Provisioning

In this test the actuators are provision in order for the system to be able to map the receiving data into the right context entities and therefore be able to know what is the received data, and also to know which commands are accepted by the actuators.

To provision the actuators in one go, a shell scrip named “7_use_case_ actuators_provisioning.sh” was created, whose contents are available in Annex O.

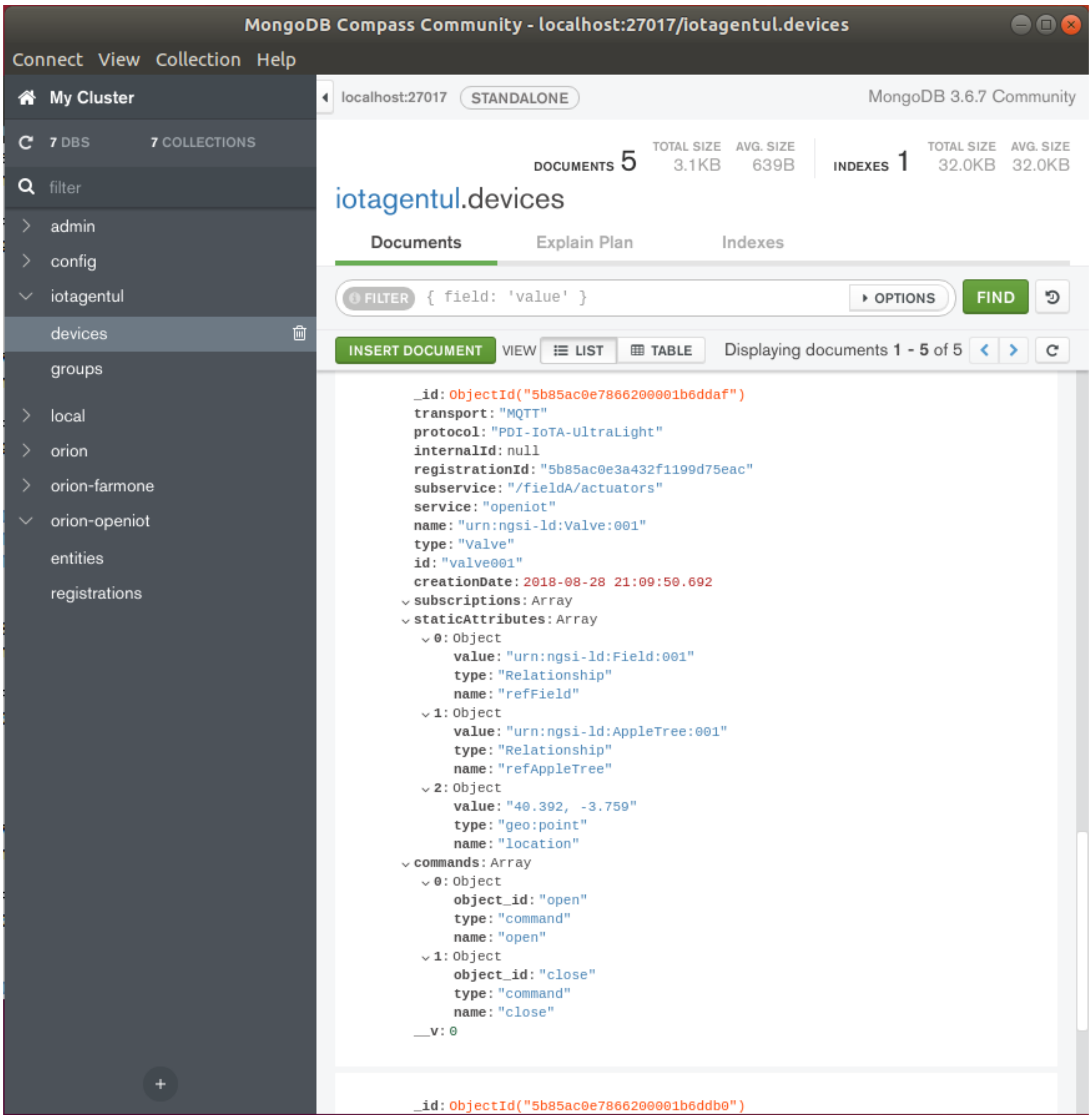
The output of the script execution, Figure 4.31, indicates that the provisioning of the actuators was done successfully.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./7_use_case_actuators_provisioning.sh
HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 53747868-5adf-4f21-895b-8d2ae71b6491
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 20:09:50 GMT
Connection: keep-alive

{}HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 333b8746-a7c3-4cb6-9fc8-712251e6b1cc
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Tue, 28 Aug 2018 20:09:50 GMT
Connection: keep-alive
```

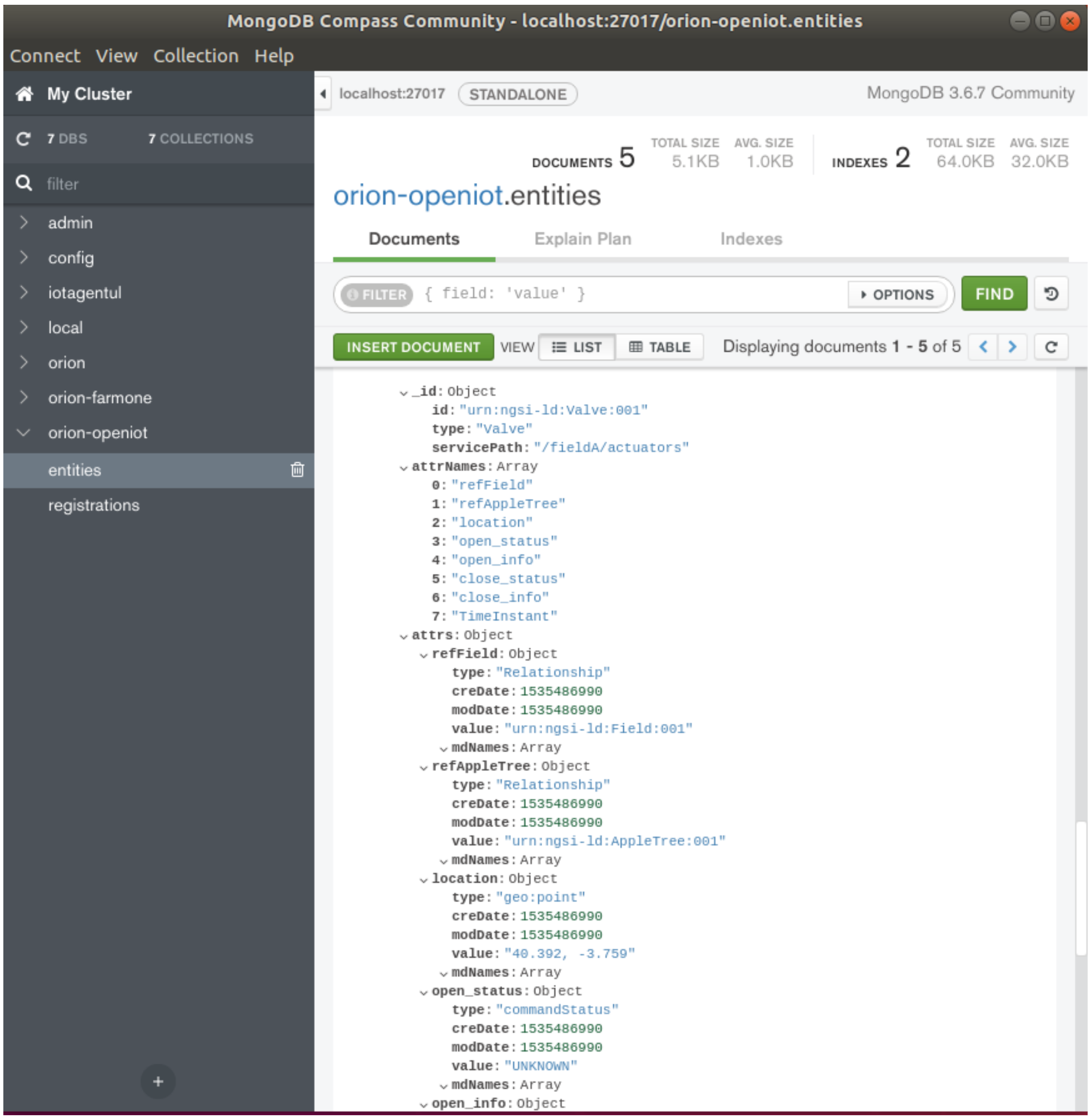
Figure 4.31 Output of the Commands That Provisioned the Actuators

In Figure 4.32 is observed that the actuators were added to the IoT Agent database and that, in Figure 4.33, the corresponding entities were also created in the Orion database. It is also observed, in Figure 4.34, that a new directory was created in the Orion database that contains information about the actuators commands.


MongoDB Compass Community - localhost:27017/iotagentul.devices
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
devices
groups
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 3.1KB AVG. SIZE 639B
INDEXES 1 TOTAL SIZE 32.0KB AVG. SIZE 32.0KB
iotagentul.devices
Documents Explain Plan Indexes
FILTER { field: 'value' }
OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE
Displaying documents 1 - 5 of 5

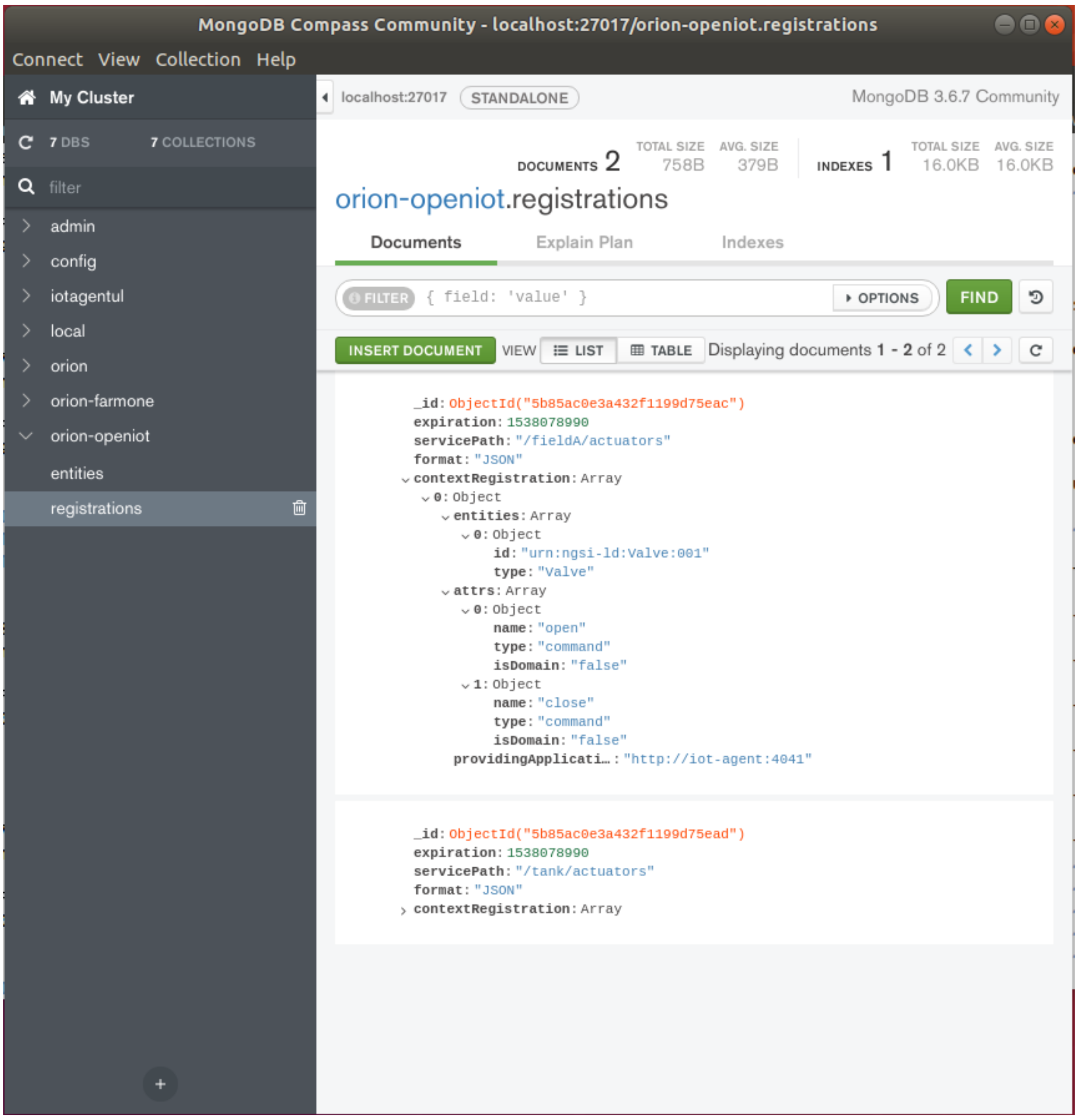
```
_id: ObjectId("5b85ac0e7866200001b6ddaf")
transport: "MQTT"
protocol: "PDI-IoTA-UltraLight"
internalId: null
registrationId: "5b85ac0e3a432f1199d75eac"
subservice: "/fieldA/actuators"
service: "openiot"
name: "urn:ngsi-ld:Valve:001"
type: "Valve"
id: "valve001"
creationDate: 2018-08-28 21:09:50.692
subscriptions: Array
staticAttributes: Array
  0: Object
     value: "urn:ngsi-ld:Field:001"
     type: "Relationship"
     name: "refField"
  1: Object
     value: "urn:ngsi-ld:AppleTree:001"
     type: "Relationship"
     name: "refAppleTree"
  2: Object
     value: "40.392, -3.759"
     type: "geo:point"
     name: "location"
commands: Array
  0: Object
     object_id: "open"
     type: "command"
     name: "open"
  1: Object
     object_id: "close"
     type: "command"
     name: "close"
__v: 0

_id: ObjectId("5b85ac0e7866200001b6ddb0")
```

Figure 4.32 Provisioned Actuators in the IoT Agent


MongoDB Compass Community - localhost:27017/orion-openiot.entities
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 5.1KB AVG. SIZE 1.0KB
INDEXES 2 TOTAL SIZE 64.0KB AVG. SIZE 32.0KB
orion-openiot.entities
Documents
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 5 of 5

```
_id: Object
    id: "urn:ngsi-ld:Valve:001"
    type: "Valve"
    servicePath: "/fieldA/actuators"
attrNames: Array
    0: "refField"
    1: "refAppleTree"
    2: "location"
    3: "open_status"
    4: "open_info"
    5: "close_status"
    6: "close_info"
    7: "TimeInstant"
attrs: Object
    refField: Object
        type: "Relationship"
        creDate: 1535486990
        modDate: 1535486990
        value: "urn:ngsi-ld:Field:001"
        mdNames: Array
    refAppleTree: Object
        type: "Relationship"
        creDate: 1535486990
        modDate: 1535486990
        value: "urn:ngsi-ld:AppleTree:001"
        mdNames: Array
    location: Object
        type: "geo:point"
        creDate: 1535486990
        modDate: 1535486990
        value: "40.392, -3.759"
        mdNames: Array
    open_status: Object
        type: "commandStatus"
        creDate: 1535486990
        modDate: 1535486990
        value: "UNKNOWN"
        mdNames: Array
    open_info: Object
```

Figure 4.33 Provisioned Actuators Entities in Orion


MongoDB Compass Community - localhost:27017/orion-openiot.registrations
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 2 TOTAL SIZE 758B AVG. SIZE 379B
INDEXES 1 TOTAL SIZE 16.0KB AVG. SIZE 16.0KB
orion-openiot.registrations
Documents Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 2 of 2

```
_id: ObjectId("5b85ac0e3a432f1199d75eac")
expiration: 1538078990
servicePath: "/fieldA/actuators"
format: "JSON"
contextRegistration: Array
  0: Object
    entities: Array
      0: Object
        id: "urn:ngsi-ld:Valve:001"
        type: "Valve"
    attrs: Array
      0: Object
        name: "open"
        type: "command"
        isDomain: "false"
      1: Object
        name: "close"
        type: "command"
        isDomain: "false"
    providingApplicati…: "http://iot-agent:4041"

_id: ObjectId("5b85ac0e3a432f1199d75ead")
expiration: 1538078990
servicePath: "/tank/actuators"
format: "JSON"
contextRegistration: Array
```

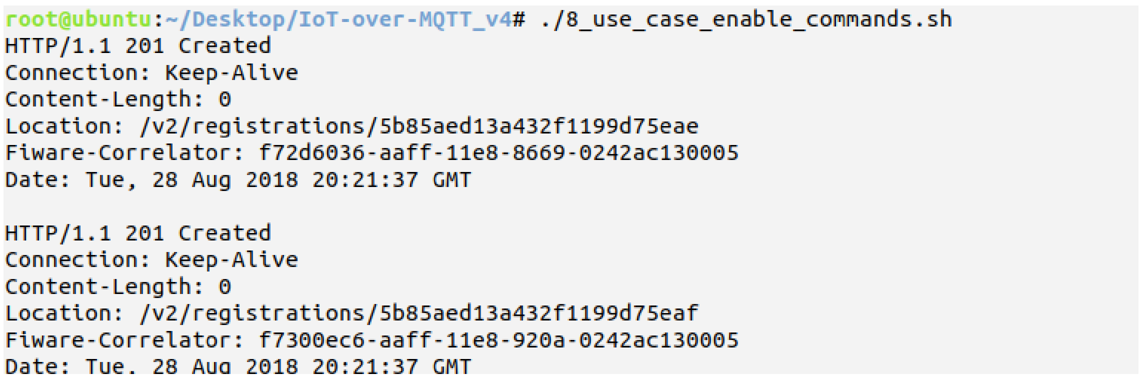
Figure 4.34 Directory with the Actuators Commands

Therefore, it was concluded that the provisioning of sensors and actuators is working properly.

4.2.5.4 Enabling Context Broker Commands

Although the actuators commands are already registered in the Orion database, it is necessary to inform the Orion Context Broker that the commands are available. For that a script named “8_use_case_enable_commands.sh” was created and whose contents are available in Annex P.

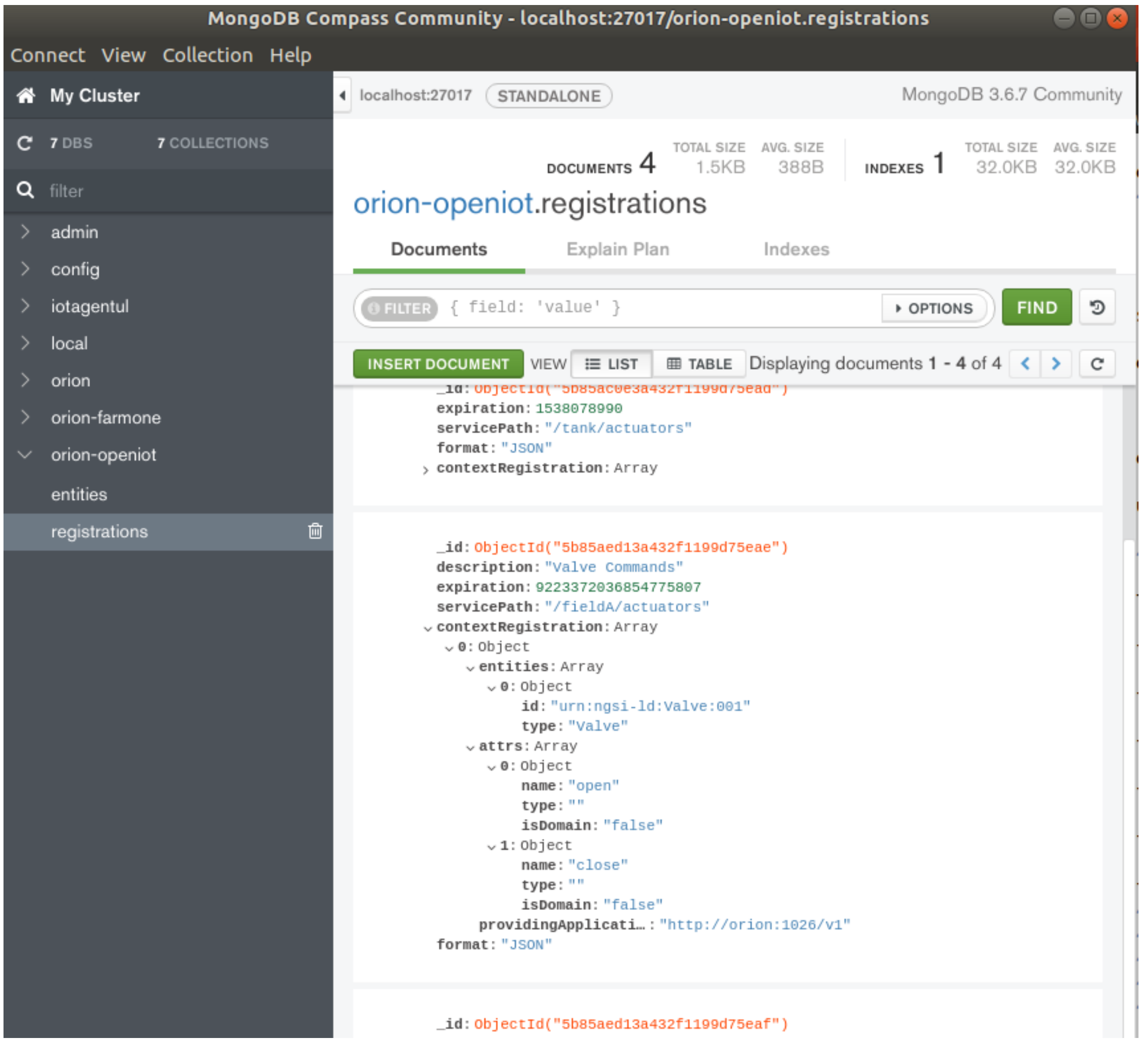
The output of the script execution (Figure 4.35) indicates that the commands were enabled successfully.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./8_use_case_enable_commands.sh
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/registrations/5b85aed13a432f1199d75eae
Fiware-Correlator: f72d6036-aaff-11e8-8669-0242ac130005
Date: Tue, 28 Aug 2018 20:21:37 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/registrations/5b85aed13a432f1199d75eaf
Fiware-Correlator: f7300ec6-aaff-11e8-920a-0242ac130005
Date: Tue, 28 Aug 2018 20:21:37 GMT
```

Figure 4.35 Output of the Commands That Enabled the Context Broker Commands for the Actuators

In Figure 4.36 is observed that the commands were added to Orion database.



MongoDB Compass Community - localhost:27017/orion-openiot.registrations
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 4 TOTAL SIZE 1.5KB AVG. SIZE 388B
INDEXES 1 TOTAL SIZE 32.0KB AVG. SIZE 32.0KB
orion-openiot.registrations
Documents Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 4 of 4

```
    _id: ObjectId("5b85ac0e3a432f1199d75ead")
    expiration: 1538078990
    servicePath: "/tank/actuators"
    format: "JSON"
  > contextRegistration: Array

    _id: ObjectId("5b85aed13a432f1199d75eae")
    description: "Valve Commands"
    expiration: 9223372036854775807
    servicePath: "/fieldA/actuators"
  v contextRegistration: Array
    v 0: Object
      v entities: Array
        v 0: Object
            id: "urn:ngsi-ld:Valve:001"
            type: "Valve"
      v attrs: Array
        v 0: Object
            name: "open"
            type: ""
            isDomain: "false"
        v 1: Object
            name: "close"
            type: ""
            isDomain: "false"
        providingApplicati…: "http://orion:1026/v1"
    format: "JSON"

    _id: ObjectId("5b85aed13a432f1199d75eaf")
```

Figure 4.36 Commands Enabled in Orion

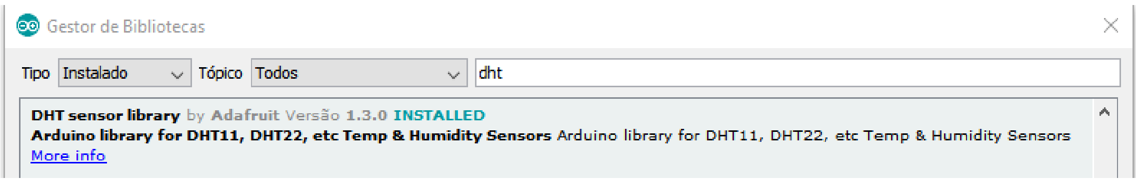
It was concluded that the commands activation is working as expected.

4.2.6 IoT Devices

The following tests serve to mainly verify if the used sensors and microcontrollers are working properly (actuators are simulated).

4.2.6.1 Test of DHT22 Sensor

To test if the DHT22 air temperature and humidity sensor is working properly it was used an already existing Arduino library (Figure 4.37) to process the sensor data and print it to the Arduino serial monitor.


Gestor de Bibliotecas

Tipo Instalado Tópico Todos dht

DHT sensor library by **Adafruit** Versão **1.3.0** **INSTALLED**
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
More info

Figure 4.37 Arduino Library Used for the DHT22 Sensor

In the Figure 4.38 can be observed the electrical schematics for connecting the sensor to the NodeMcu ESP8266 Devkit v1.0 board.

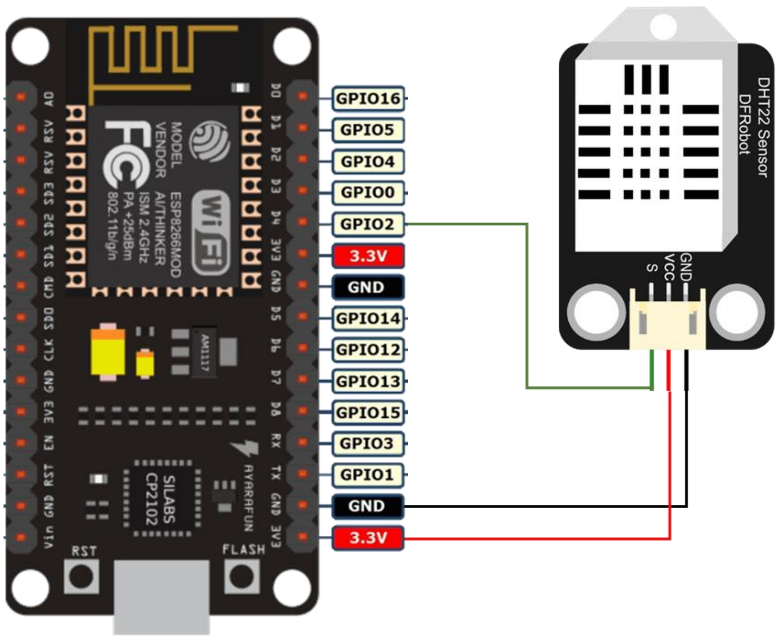


Figure 4.38 Electrical Schematics for Connecting the DHT22 Sensor to the NodeMcu ESP8266 Devkit v1.0 Board

Figure 4.39 shows the experimental montage of the circuit.

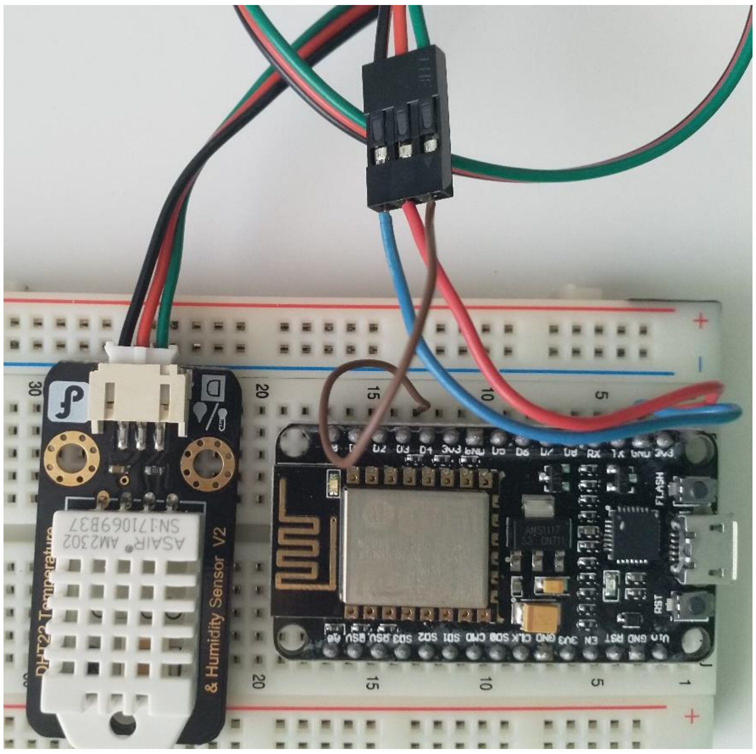
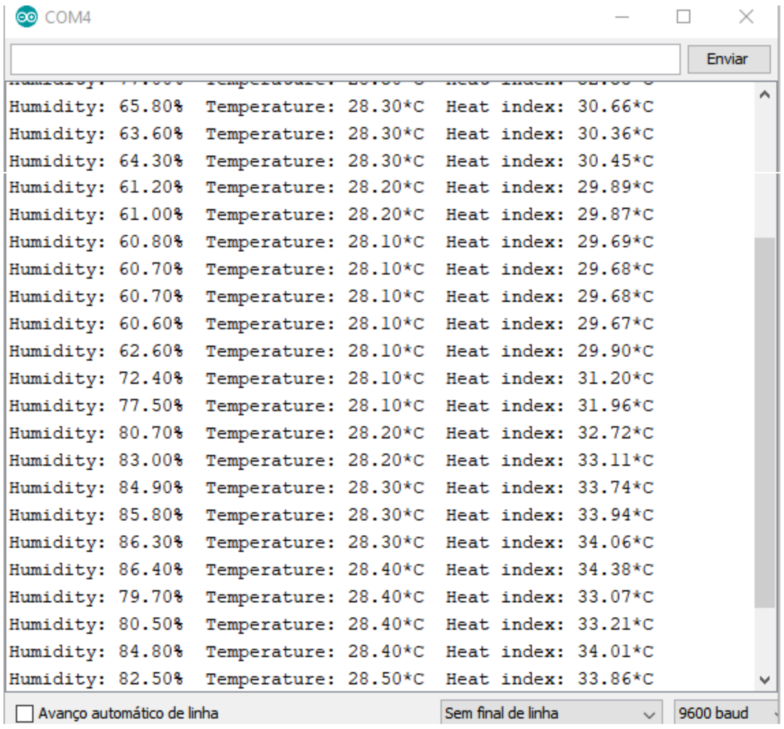


Figure 4.39 DHT22 Sensor + NodeMcu Circuit Montage

The code uploaded to the board (file name “esp8266_temp_hum_test.ino”) is available in Annex Q, which outputs the obtained data from the sensor (Figure 4.40).



```
Humidity: 65.80%  Temperature: 28.30*C  Heat index: 30.66*C
Humidity: 63.60%  Temperature: 28.30*C  Heat index: 30.36*C
Humidity: 64.30%  Temperature: 28.30*C  Heat index: 30.45*C
Humidity: 61.20%  Temperature: 28.20*C  Heat index: 29.89*C
Humidity: 61.00%  Temperature: 28.20*C  Heat index: 29.87*C
Humidity: 60.80%  Temperature: 28.10*C  Heat index: 29.69*C
Humidity: 60.70%  Temperature: 28.10*C  Heat index: 29.68*C
Humidity: 60.70%  Temperature: 28.10*C  Heat index: 29.68*C
Humidity: 60.60%  Temperature: 28.10*C  Heat index: 29.67*C
Humidity: 62.60%  Temperature: 28.10*C  Heat index: 29.90*C
Humidity: 72.40%  Temperature: 28.10*C  Heat index: 31.20*C
Humidity: 77.50%  Temperature: 28.10*C  Heat index: 31.96*C
Humidity: 80.70%  Temperature: 28.20*C  Heat index: 32.72*C
Humidity: 83.00%  Temperature: 28.20*C  Heat index: 33.11*C
Humidity: 84.90%  Temperature: 28.30*C  Heat index: 33.74*C
Humidity: 85.80%  Temperature: 28.30*C  Heat index: 33.94*C
Humidity: 86.30%  Temperature: 28.30*C  Heat index: 34.06*C
Humidity: 86.40%  Temperature: 28.40*C  Heat index: 34.38*C
Humidity: 79.70%  Temperature: 28.40*C  Heat index: 33.07*C
Humidity: 80.50%  Temperature: 28.40*C  Heat index: 33.21*C
Humidity: 84.80%  Temperature: 28.40*C  Heat index: 34.01*C
Humidity: 82.50%  Temperature: 28.50*C  Heat index: 33.86*C
```

Figure 4.40 Obtained Data from the DHT22 Sensor

4.2.6.2 Test of Ultrasonic Sensor

This sensor requires 5V to work however the board is only capable of supplying 3.3V, however, since the board is being powered through the USB port it is possible to draw 5V from the Vin pin to power the sensor.

In the Figure 4.41 can be observed the electrical schematics for connecting the sensor to the NodeMcu ESP8266 Devkit v1.0 board.

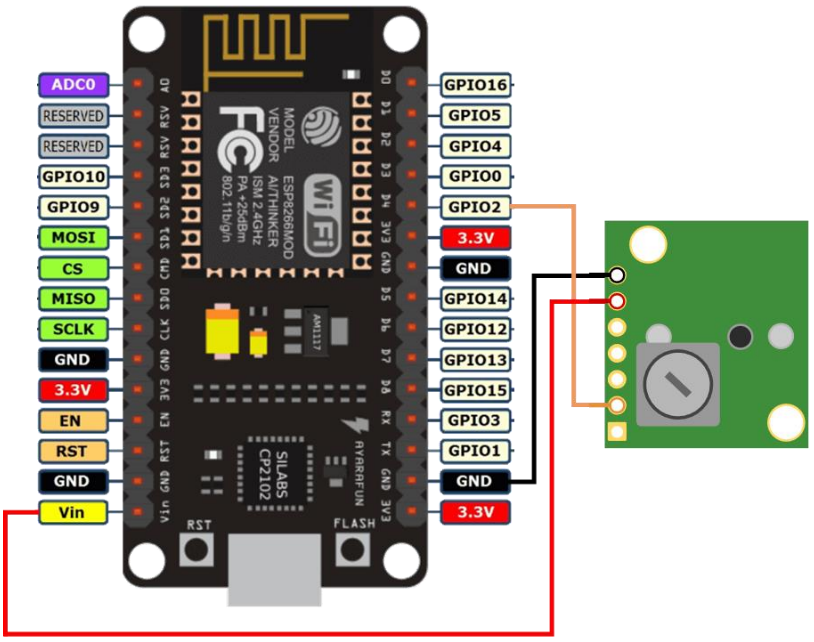


Figure 4.41 Electrical Schematics for Connecting the Ultrasonic Sensor to the NodeMcu ESP8266 Devkit v1.0 Board

Figure 4.42 shows the montage of the circuit.

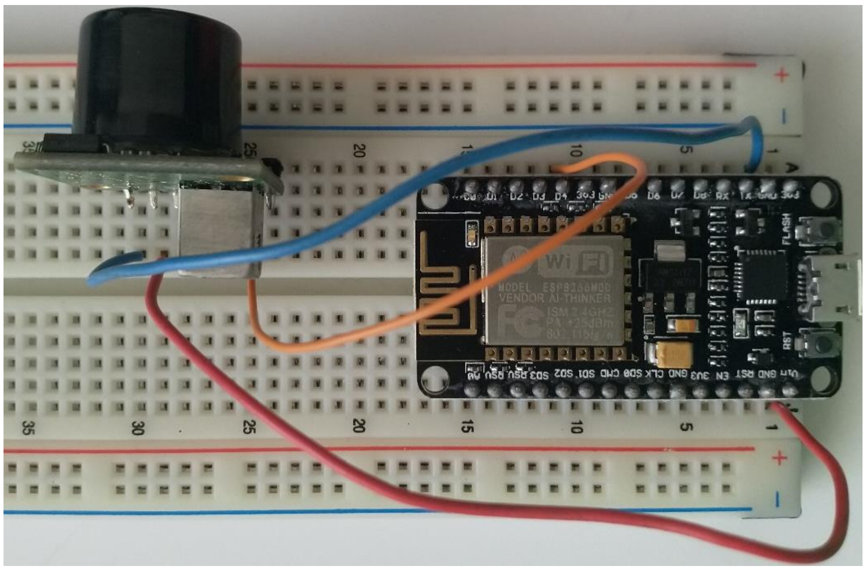
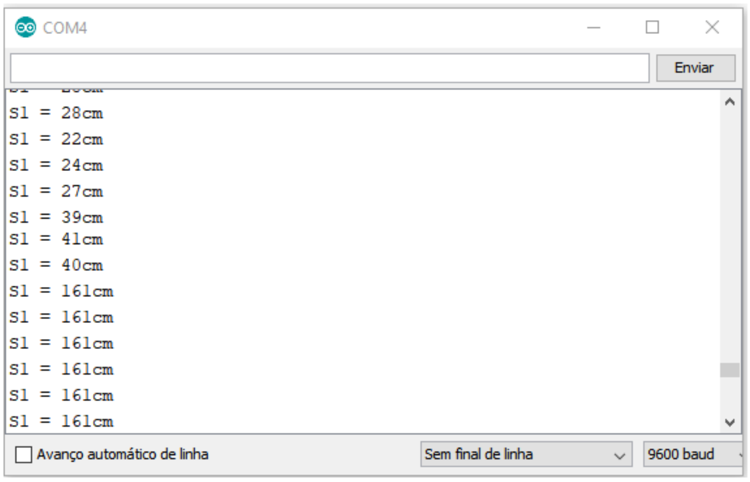


Figure 4.42 Ultrasonic Sensor + NodeMcu Circuit Montage

To test the sensor, it was used an already made example available in the manufacturer website [70], that had to be adapted to work with the board as it was originally for Arduino. The code uploaded to the board (file name “esp8266_ultrasonic_test.ino”) is available in Annex R.

Figure 4.43 shows the output of the obtained data from the sensor.



```
S1 = 28cm
S1 = 22cm
S1 = 24cm
S1 = 27cm
S1 = 39cm
S1 = 41cm
S1 = 40cm
S1 = 161cm
S1 = 161cm
S1 = 161cm
S1 = 161cm
S1 = 161cm
S1 = 161cm
```

Avanço automático de linha | Sem final de linha | 9600 baud

Figure 4.43 Obtained Data from the Ultrasonic Sensor

4.2.6.3 Test of Earth-Humidity Sensor

The Figure 4.44 shows the electrical schematics for connecting the sensor to the NodeMcu ESP8266 Devkit v1.0 board.

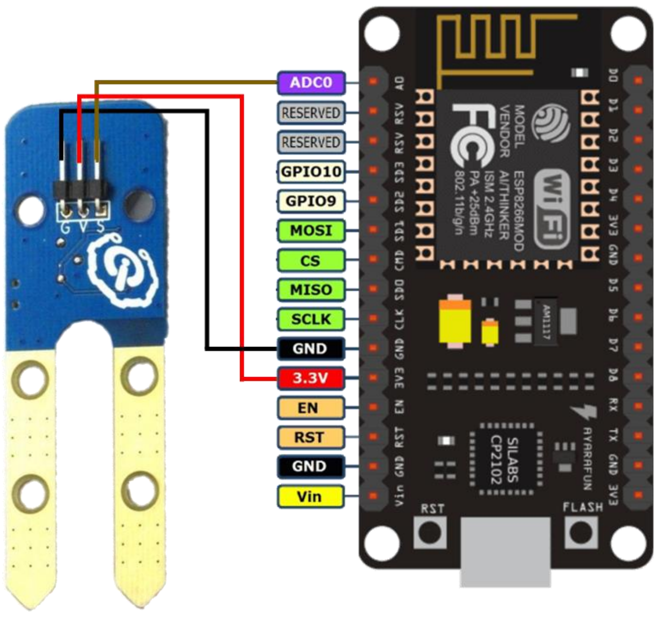


Figure 4.44 Electrical Schematics for Connecting the Earth-Humidity Sensor to the NodeMcu ESP8266 Devkit v1.0 Board

Figure 4.45 and Figure 4.46 show the montage of the circuit.

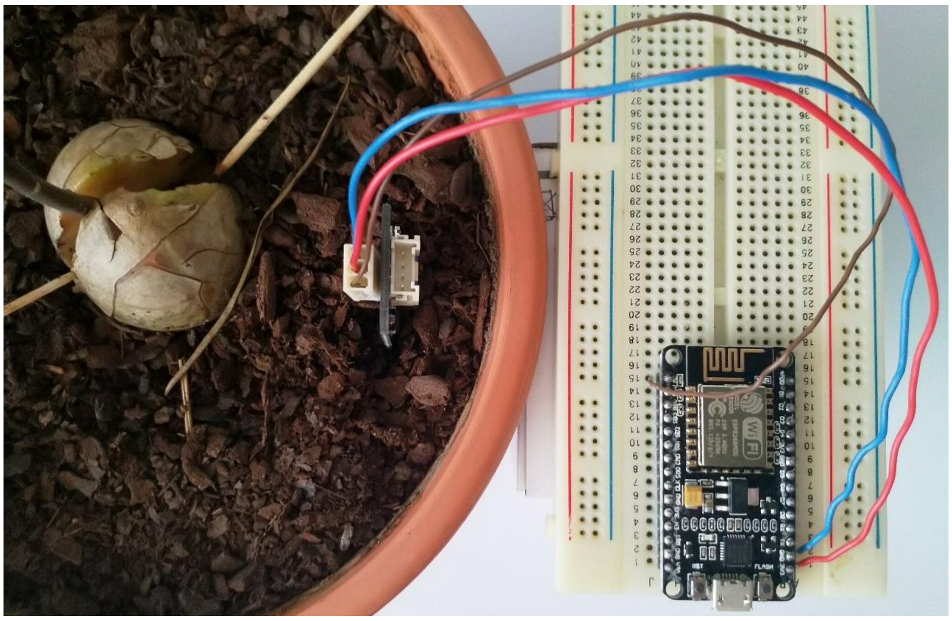


Figure 4.45 Earth-Humidity Sensor + NodeMcu Circuit Montage (1)

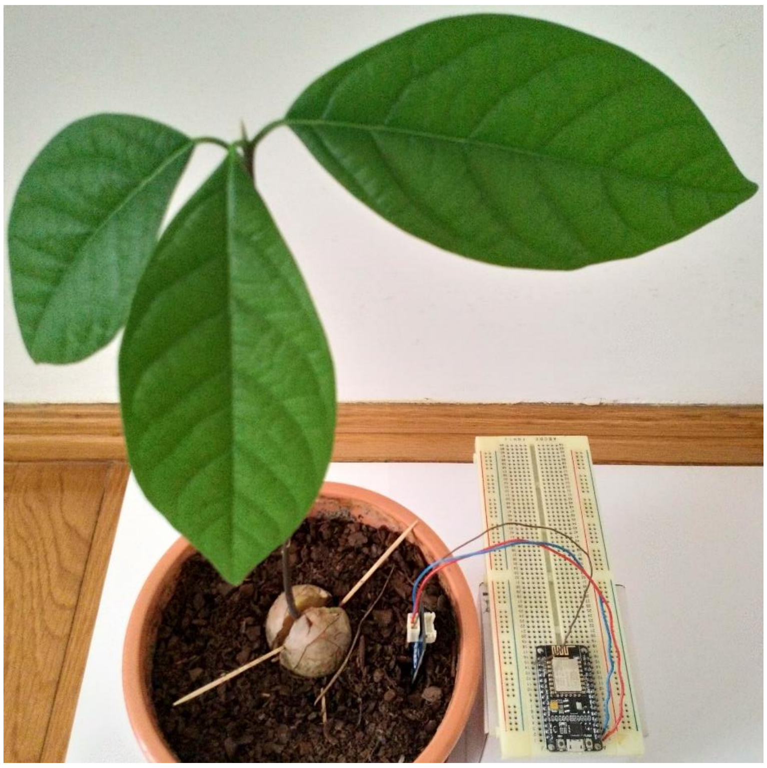
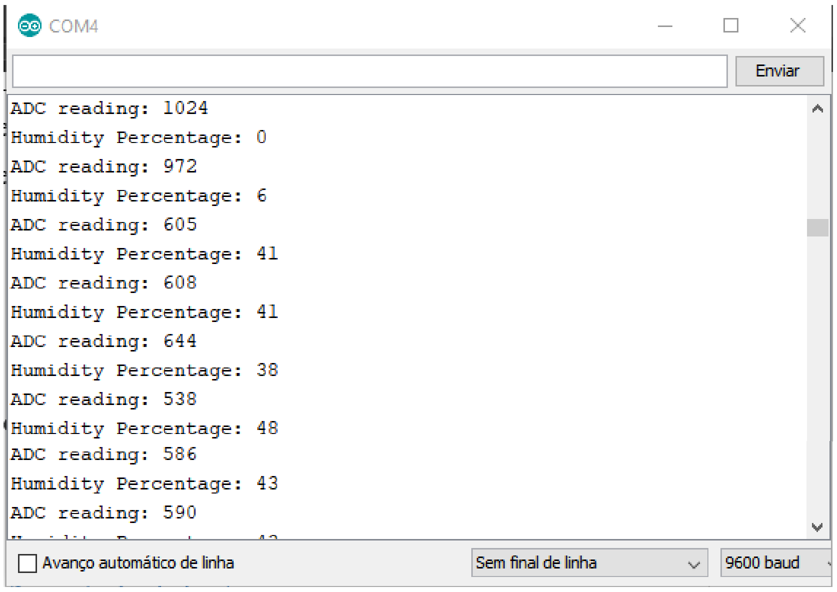


Figure 4.46 Earth-Humidity Sensor + NodeMcu Circuit Montage (2)

To test the sensor, it was used an already made example available in the manufacturer website [71], that had to be adapted to display better information, such as humidity in percentage. Both operation modes of the sensor, digital and analogic, were tested however only the analogic mode is useful for the use case as displays the values of the soil moisture, when the digital mode only returns “1” or “0” if the soil humidity is above or below a threshold defined by the potentiometer. The analog mode automatically implies that the ADC is used to convert the sensor analog output into digital values, between 0 and 1024, that are then converted to a percentage.

The code uploaded to the board (file name “esp8266_earth_humidity_test.ino”) is available in Annex S.

Figure 4.47 shows the output of the obtained data from the sensor. When the sensor probes were in the air the humidity percentage was 0 and when the probes were connected through a jump wire the humidity percentage was 100. The other values were obtained by placing the sensor in a vase.



```
ADC reading: 1024
Humidity Percentage: 0
ADC reading: 972
Humidity Percentage: 6
ADC reading: 605
Humidity Percentage: 41
ADC reading: 608
Humidity Percentage: 41
ADC reading: 644
Humidity Percentage: 38
ADC reading: 538
Humidity Percentage: 48
ADC reading: 586
Humidity Percentage: 43
ADC reading: 590
```

☐ Avanço automático de linha | Sem final de linha | 9600 baud

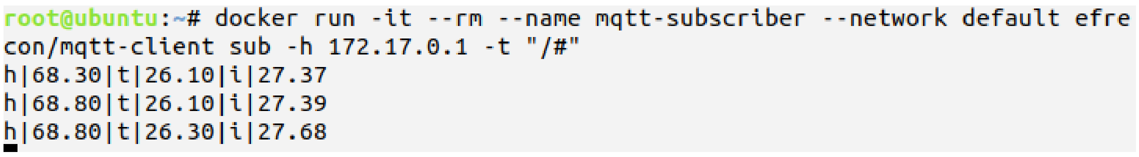
Figure 4.47 Obtained Data from the Earth-Humidity Sensor

Ideally the sensor probes should have been longer to obtain the humidity values of deeper soil, as the soil may appear to be dry at the surface but can be wet or moist at higher depths. For a single pant, multiple sensors with different probe lengths could also be used to obtain a global humidity reading of the soil.

4.2.6.4 Sending Measurements from DHT22 Sensor to the IoT System

The programs written for the IoT Devices are based on an existing Arduino MQTT library [72], which is one of the rare ones that support publishing messages with QoS (0, 1 and 2), being used QoS 2. The code uploaded to the board (file name "mqtt_esp8266_dht22_qos_v1.ino"), which collects the sensor data and transmits it to the system is available in Annex T.

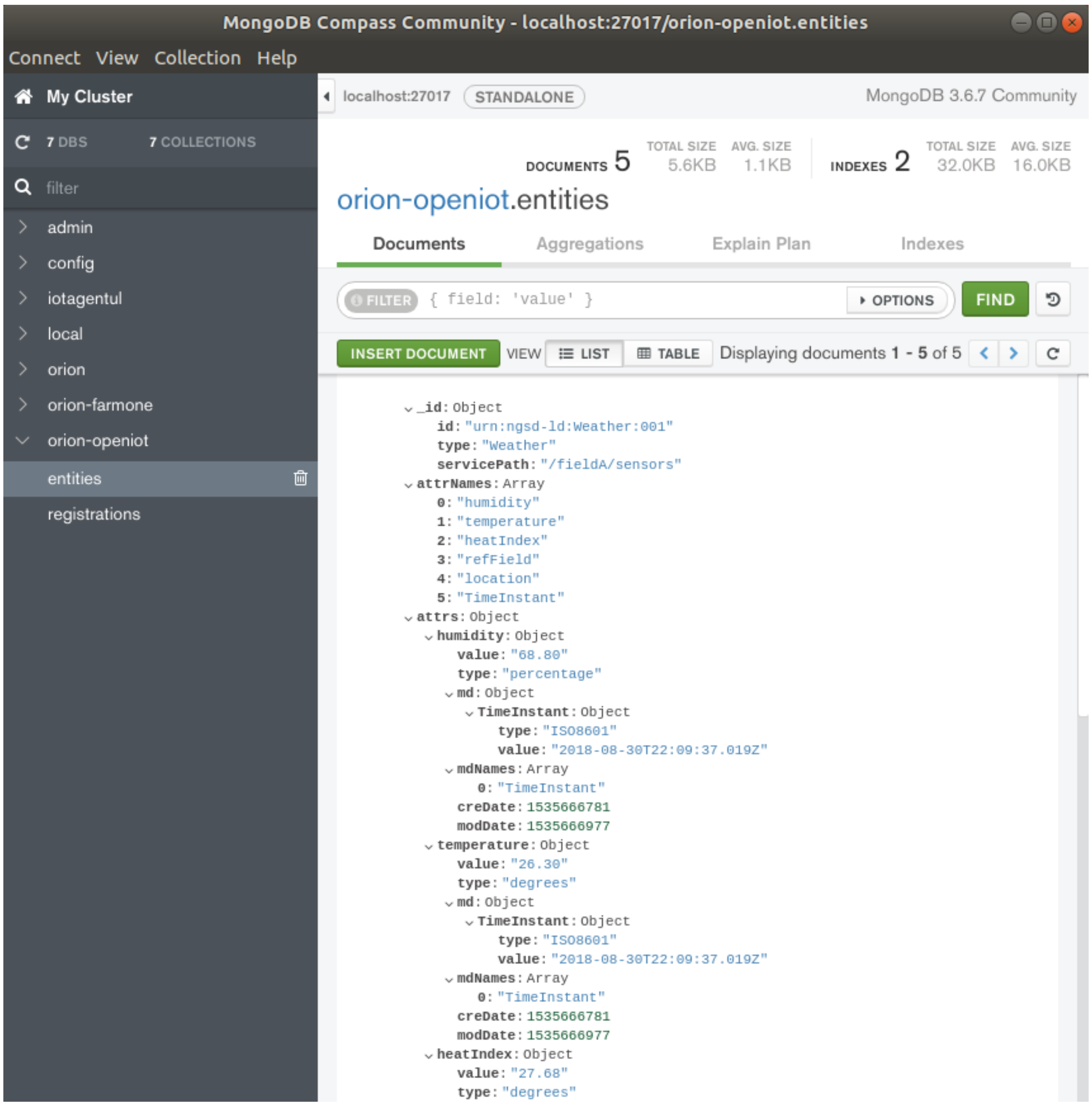
To visualize the message received by the Mosquitto MQTT Broker it was used the dummy MQTT subscriber (Figure 4.48) previously used when testing the Mosquitto Health.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber --network default efre
con/mqtt-client sub -h 172.17.0.1 -t "/#"
h|68.30|t|26.10|i|27.37
h|68.80|t|26.10|i|27.39
h|68.80|t|26.30|i|27.68
```

Figure 4.48 Messages Received by the MQTT Subscriber and Sent by the Weather Sensor

Figure 4.49 shows the received data from the sensor in the database.

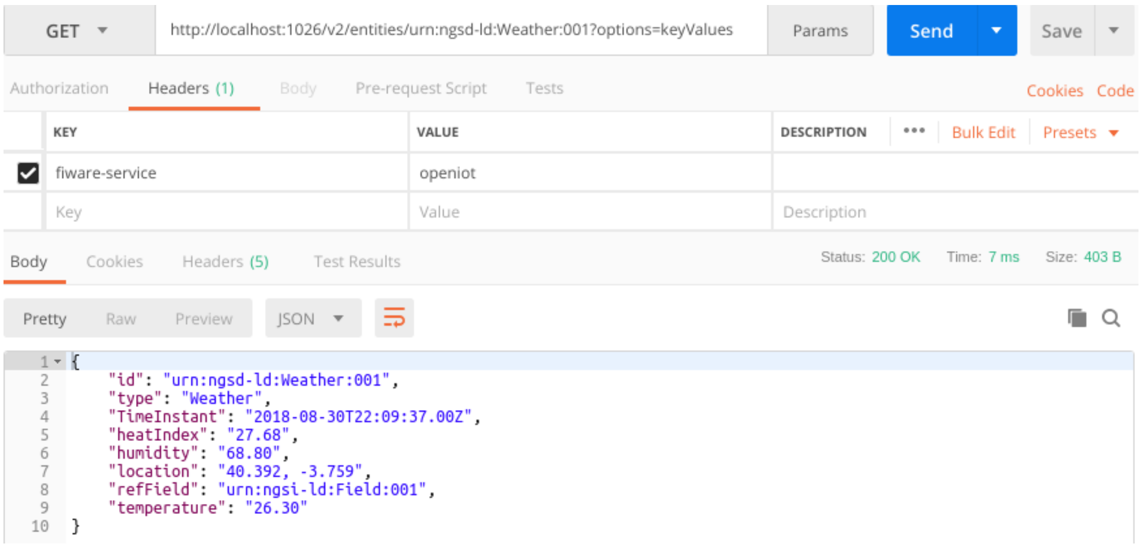


MongoDB Compass Community - localhost:27017/orion-openiot.entities
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 5.6KB AVG. SIZE 1.1KB
INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB
orion-openiot.entities
Documents Aggregations Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 5 of 5

```
_id: Object
  id: "urn:ngsd-ld:Weather:001"
  type: "Weather"
  servicePath: "/fieldA/sensors"
attrNames: Array
  0: "humidity"
  1: "temperature"
  2: "heatIndex"
  3: "refField"
  4: "location"
  5: "TimeInstant"
attrs: Object
  humidity: Object
    value: "68.80"
    type: "percentage"
    md: Object
      TimeInstant: Object
        type: "ISO8601"
        value: "2018-08-30T22:09:37.019Z"
    mdNames: Array
      0: "TimeInstant"
    creDate: 1535666781
    modDate: 1535666977
  temperature: Object
    value: "26.30"
    type: "degrees"
    md: Object
      TimeInstant: Object
        type: "ISO8601"
        value: "2018-08-30T22:09:37.019Z"
    mdNames: Array
      0: "TimeInstant"
    creDate: 1535666781
    modDate: 1535666977
  heatIndex: Object
    value: "27.68"
    type: "degrees"
```

Figure 4.49 Received Data in the Weather Sensor Entity

Figure 4.50 shows the results of making a query for the sensor data.



```
http://localhost:1026/v2/entities/urn:ngsd-ld:Weather:001?options=keyValues
```

KEY	VALUE	DESCRIPTION
☑ fiware-service	openiot	
Key	Value	Description

Status: 200 OK Time: 7 ms Size: 403 B

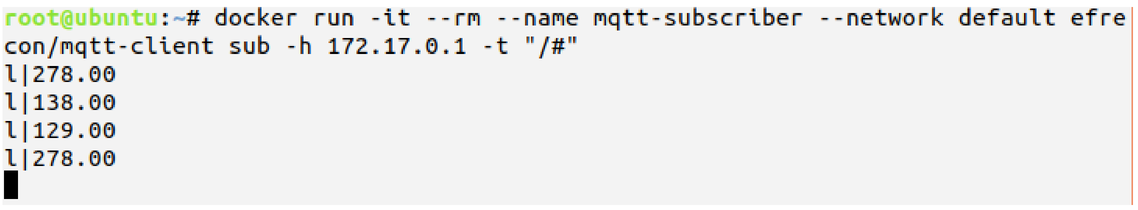
```
{
    "id": "urn:ngsd-ld:Weather:001",
    "type": "Weather",
    "TimeInstant": "2018-08-30T22:09:37.00Z",
    "heatIndex": "27.68",
    "humidity": "68.80",
    "location": "40.392, -3.759",
    "refField": "urn:ngsi-ld:Field:001",
    "temperature": "26.30"
}
```

Figure 4.50 Weather Sensor Key Values

4.2.6.5 Sending Measurements from Ultrasonic Sensor to the IoT System

The code uploaded to the board (file name “mqtt_esp8266_ultrasonic_qos_v1.ino”), which collects the sensor data and transmits it to the system is available in Annex U.

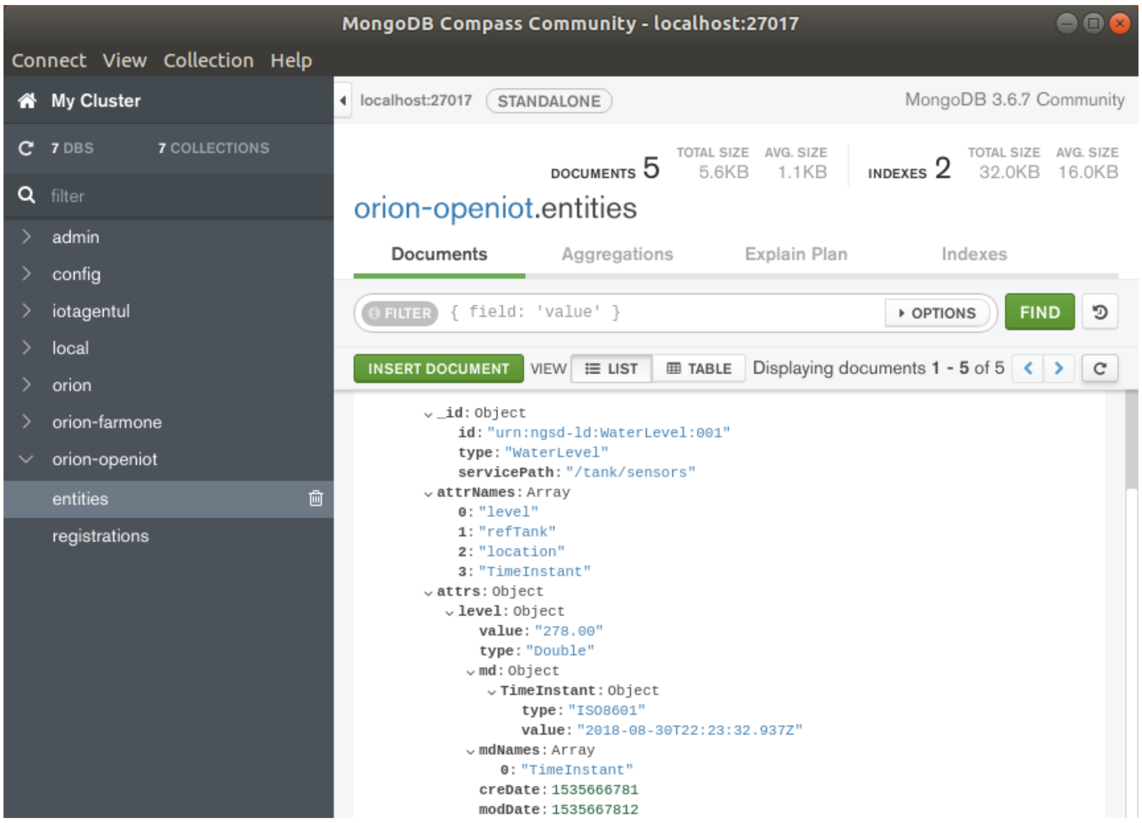
To visualize the message received by the Mosquitto MQTT Broker it was used the dummy MQTT subscriber (Figure 4.51) previously used when testing the Mosquitto Health.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber --network default efre
con/mqtt-client sub -h 172.17.0.1 -t "/#"
l|278.00
l|138.00
l|129.00
l|278.00
```

Figure 4.51 Messages Received by the MQTT Subscriber and Sent by the Water Level Sensor

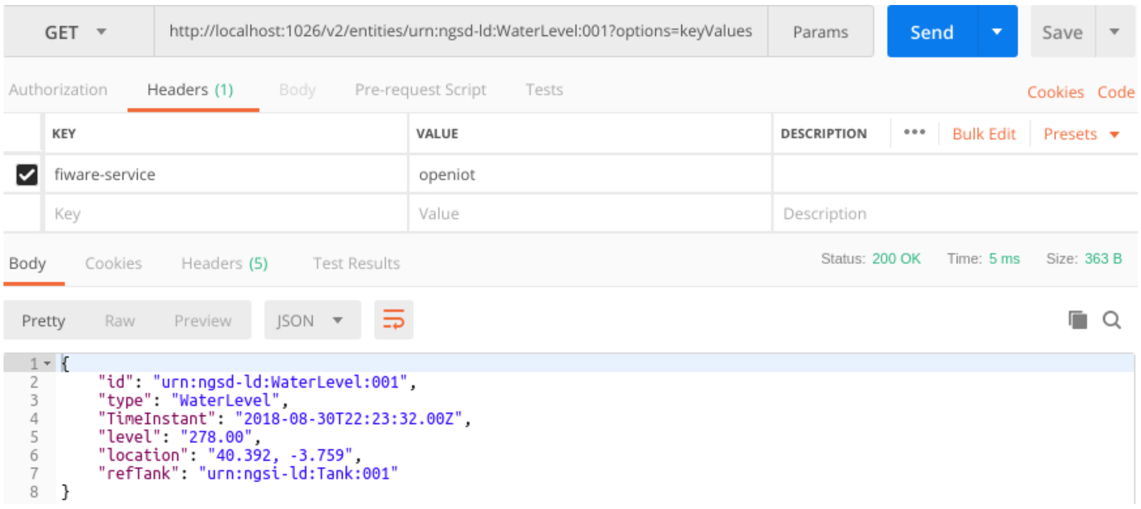
Figure 4.52 shows the received data from the sensor in the database.


MongoDB Compass Community - localhost:27017
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 5.6KB AVG. SIZE 1.1KB
INDEXES 2 TOTAL SIZE 32.0KB AVG. SIZE 16.0KB
orion-openiot.entities
Documents Aggregations Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 5 of 5

```
_id: Object
   id: "urn:ngsd-ld:WaterLevel:001"
   type: "WaterLevel"
   servicePath: "/tank/sensors"
attrNames: Array
   0: "level"
   1: "refTank"
   2: "location"
   3: "TimeInstant"
attrs: Object
   level: Object
      value: "278.00"
      type: "Double"
      md: Object
         TimeInstant: Object
            type: "ISO8601"
            value: "2018-08-30T22:23:32.937Z"
      mdNames: Array
         0: "TimeInstant"
      creDate: 1535666781
      modDate: 1535667812
```

Figure 4.52 Received Data in the Water Level Sensor Entity

Figure 4.53 shows the results of making a query for the sensor data.


GET
Params
Send
Save
Authorization Headers (1) Body Pre-request Script Tests
Cookies Code
Bulk Edit Presets
Body Cookies Headers (5) Test Results
Pretty Raw Preview JSON

```
http://localhost:1026/v2/entities/urn:ngsd-ld:WaterLevel:001?options=keyValues
```

	KEY	VALUE	DESCRIPTION
✓	fiware-service	openiot	
	Key	Value	Description

Status: 200 OK Time: 5 ms Size: 363 B

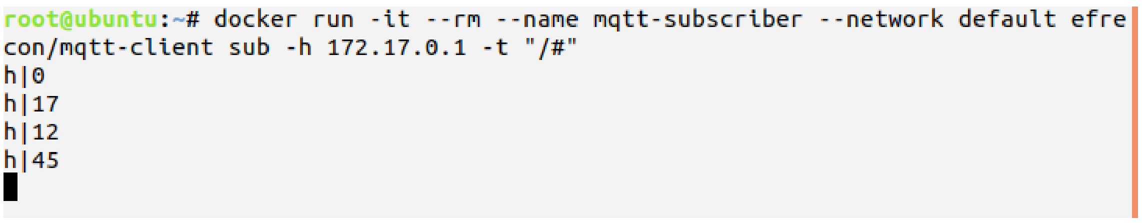
```
{
    "id": "urn:ngsd-ld:WaterLevel:001",
    "type": "WaterLevel",
    "TimeInstant": "2018-08-30T22:23:32.00Z",
    "level": "278.00",
    "location": "40.392, -3.759",
    "refTank": "urn:ngsi-ld:Tank:001"
}
```

Figure 4.53 Water Level Sensor Key Values

4.2.6.6 Sending Measurements from Earth-Humidity Sensor to the IoT System

The code uploaded to the board (file name “mqtt_esp8266_earth_humidity_qos_v1.ino”), which collects the sensor data and transmits it to the system is available in Annex V.

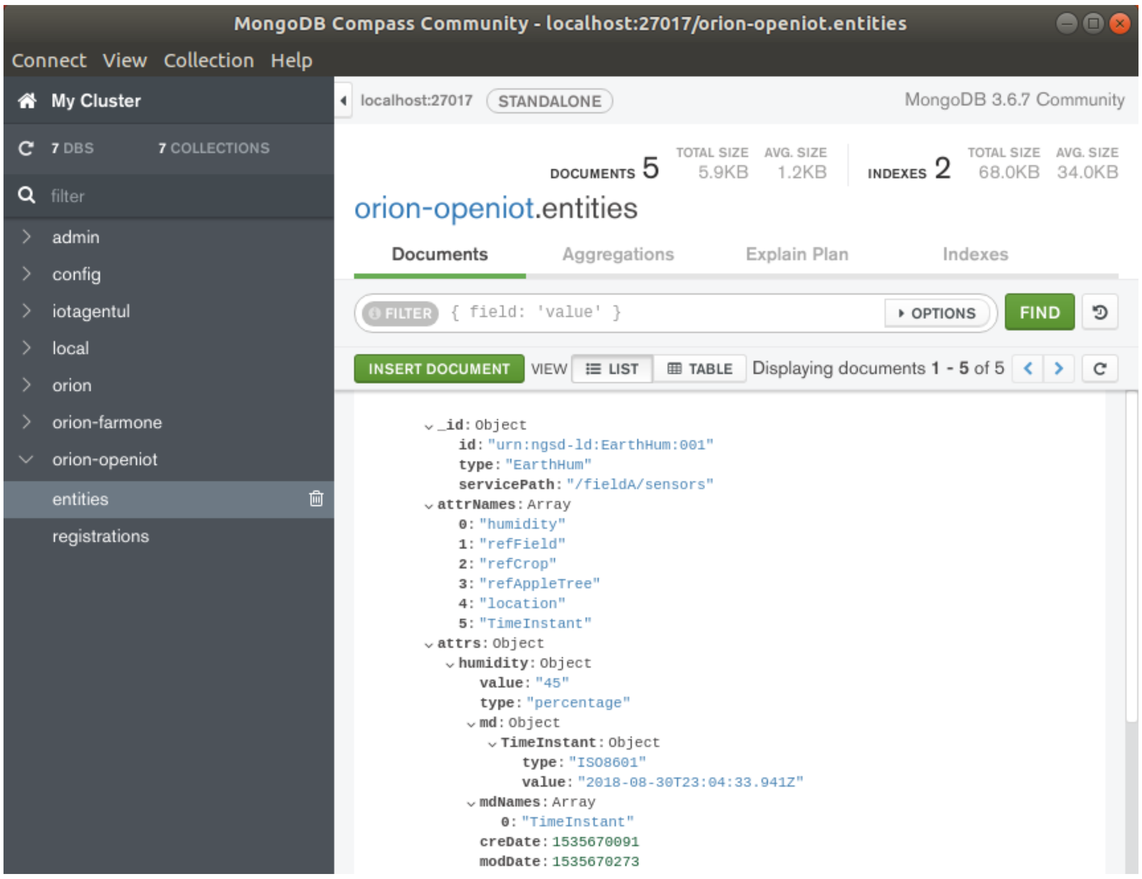
To visualize the message received by the Mosquitto MQTT Broker it was used the dummy MQTT subscriber (Figure 4.54) previously used when testing the Mosquitto Health.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber --network default efre
con/mqtt-client sub -h 172.17.0.1 -t "/#"
h|0
h|17
h|12
h|45
```

Figure 4.54 Messages Received by the MQTT Subscriber and Sent by the Earth-Humidity Sensor

Figure 4.55 shows the received data from the sensor in the database.

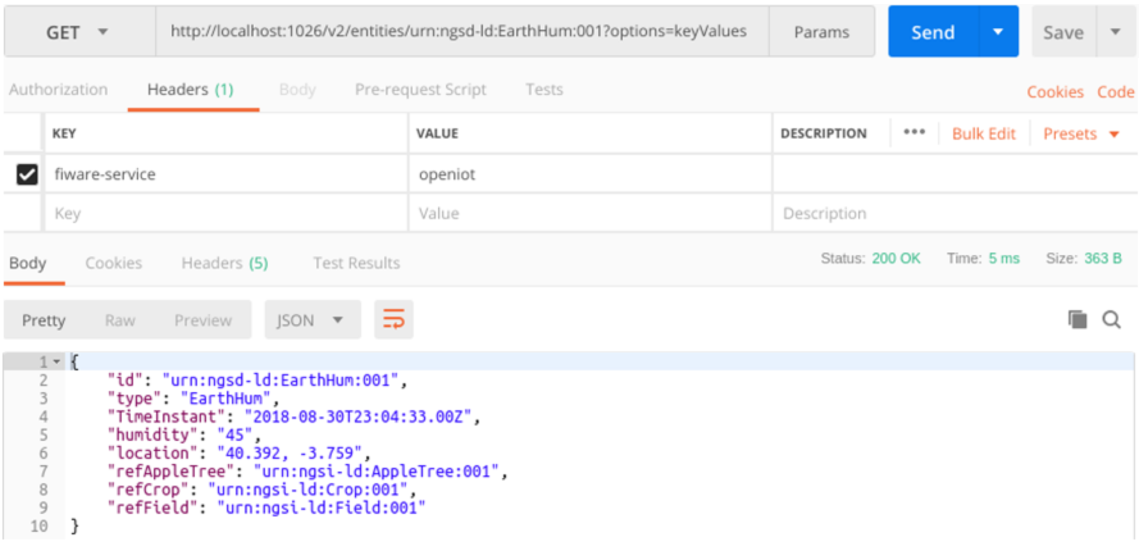


MongoDB Compass Community - localhost:27017/orion-openiot.entities
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 5.9KB AVG. SIZE 1.2KB
INDEXES 2 TOTAL SIZE 68.0KB AVG. SIZE 34.0KB
orion-openiot.entities
Documents Aggregations Explain Plan Indexes
FILTER { field: 'value' }
OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 5 of 5

```
_id: Object
  id: "urn:ngsd-ld:EarthHum:001"
  type: "EarthHum"
  servicePath: "/fieldA/sensors"
attrNames: Array
  0: "humidity"
  1: "refField"
  2: "refCrop"
  3: "refAppleTree"
  4: "location"
  5: "TimeInstant"
attrs: Object
  humidity: Object
    value: "45"
    type: "percentage"
    md: Object
      TimeInstant: Object
        type: "ISO8601"
        value: "2018-08-30T23:04:33.941Z"
    mdNames: Array
      0: "TimeInstant"
    creDate: 1535670091
    modDate: 1535670273
```

Figure 4.55 Received Data in the Earth-Humidity Sensor Entity

Figure 4.56 shows the results of making a query for the sensor data.



http://localhost:1026/v2/entities/urn:ngsd-ld:EarthHum:001?options=keyValues

KEY	VALUE	DESCRIPTION
fiware-service	openiot	
Key	Value	Description

Status: 200 OK Time: 5 ms Size: 363 B

```
{
    "id": "urn:ngsd-ld:EarthHum:001",
    "type": "EarthHum",
    "TimeInstant": "2018-08-30T23:04:33.00Z",
    "humidity": "45",
    "location": "40.392, -3.759",
    "refAppleTree": "urn:ngsi-ld:AppleTree:001",
    "refCrop": "urn:ngsi-ld:Crop:001",
    "refField": "urn:ngsi-ld:Field:001"
}
```

Figure 4.56 Earth-Humidity Sensor Key Values

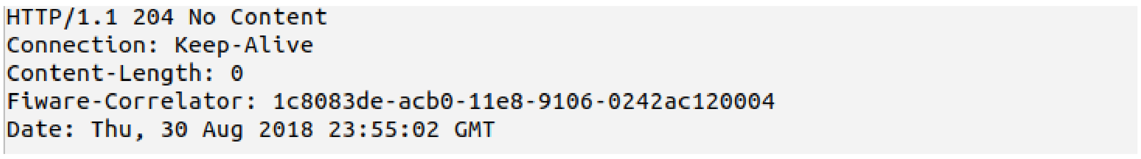
4.2.6.7 Sending Commands from System to IoT Devices (Actuators)

The code uploaded to the board (file names “mqtt_esp8266_valve001_v1.ino” and “mqtt_esp8266_valve002_v1.ino”) is available in Annex W. As the only difference between the two programs is the device Id in the MQTT topics, only of the programs is presented.

Below is the cURL command sent to the Orion to open the valve 001, which in this case turns ON the built in LED of the board (all the available commands are in file named “9_use_case_send_commands.sh” available in Annex X).

```
curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:001/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
  "open": {
      "type" : "command",
      "value" : ""
  }
}'
```

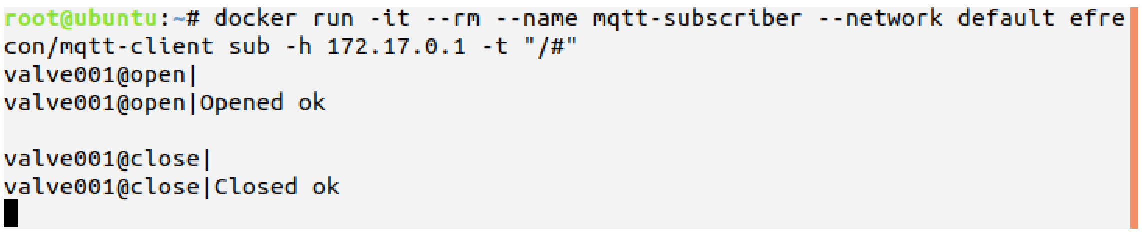
Figure 4.57 shows the output of the command.



```
HTTP/1.1 204 No Content
Connection: Keep-Alive
Content-Length: 0
Fiware-Correlator: 1c8083de-acb0-11e8-9106-0242ac120004
Date: Thu, 30 Aug 2018 23:55:02 GMT
```

Figure 4.57 Output of the Command Sent to the Actuator

To visualize the commands, sent by the Orion Context Broker and the actuator reply it was used the dummy subscriber (Figure 4.58) previously used when testing the Mosquitto Health.



```
root@ubuntu:~# docker run -it --rm --name mqtt-subscriber --network default efre
con/mqtt-client sub -h 172.17.0.1 -t "/#"
valve001@open|
valve001@open|Opened ok

valve001@close|
valve001@close|Closed ok
```

Figure 4.58 Messages Containing the Commands Sent to the Actuator and the Response Received

Figure 4.59 shows the results of the “open” command, which turned the board LED on simulating an open valve.

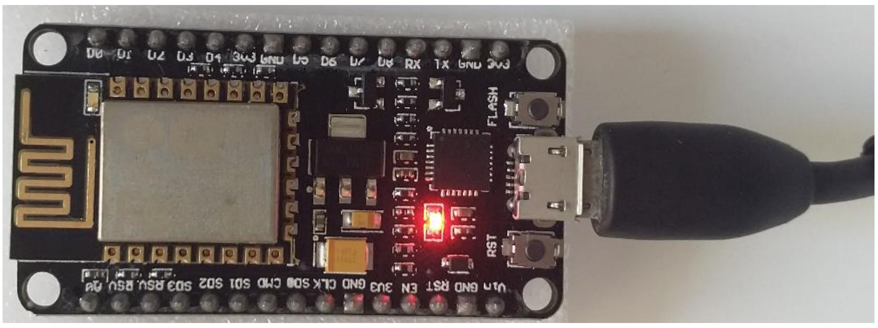
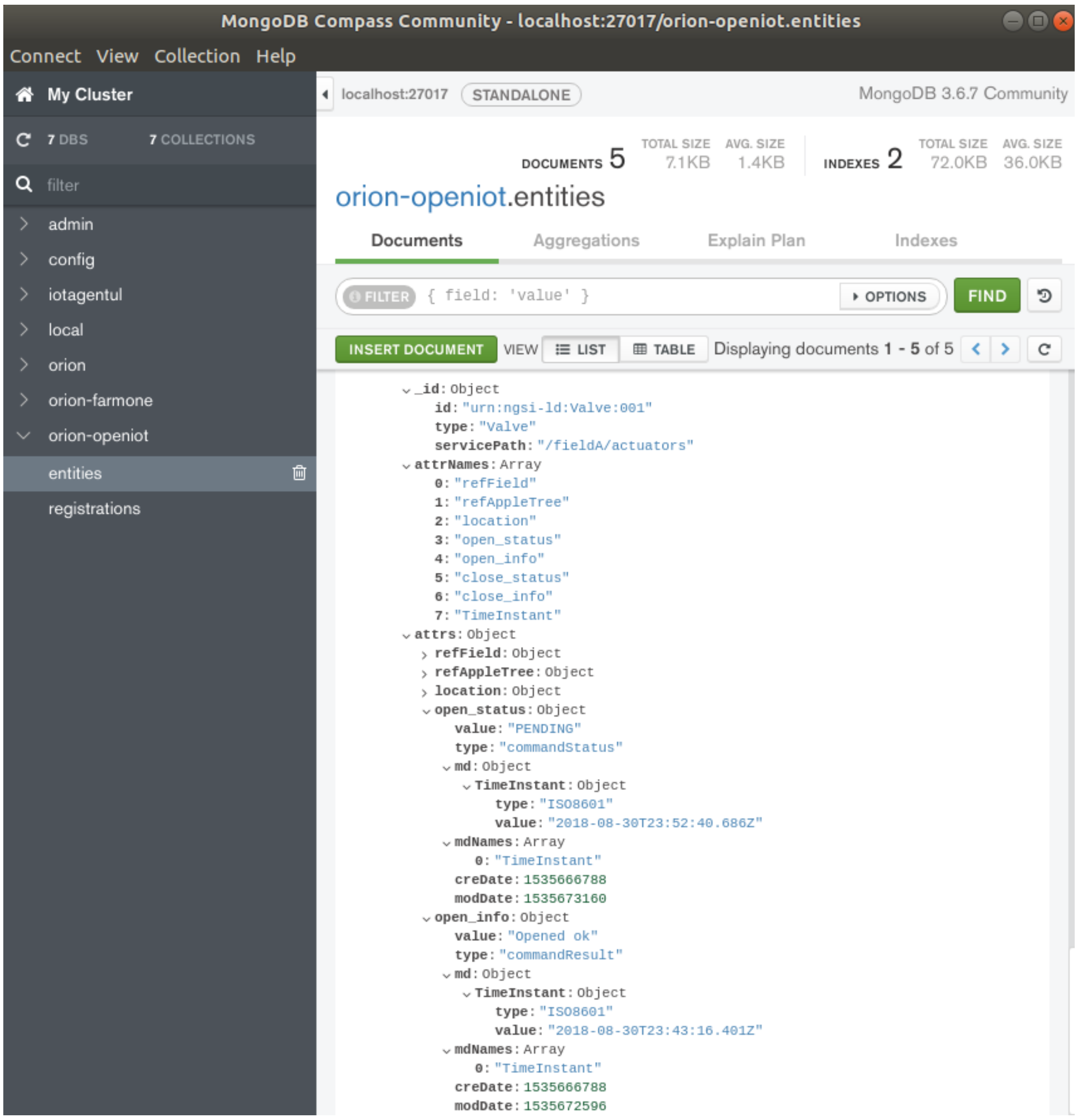


Figure 4.59 Simulation of an Open Valve (Actuator)

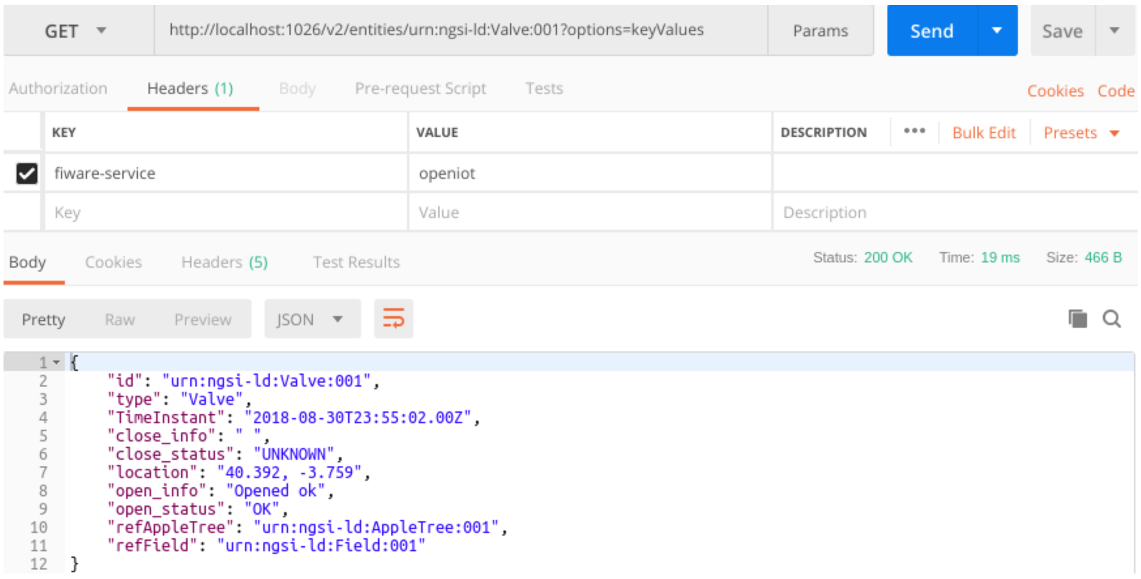
Figure 4.60 shows the information about the actuator after the command execution in the database.


MongoDB Compass Community - localhost:27017/orion-openiot.entities
Connect View Collection Help
My Cluster
7 DBS 7 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 5 TOTAL SIZE 7.1KB AVG. SIZE 1.4KB
INDEXES 2 TOTAL SIZE 72.0KB AVG. SIZE 36.0KB
orion-openiot.entities
Documents Aggregations Explain Plan Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 5 of 5

```
⌄_id: Object
   id: "urn:ngsi-ld:Valve:001"
   type: "Valve"
   servicePath: "/fieldA/actuators"
⌄attrNames: Array
   0: "refField"
   1: "refAppleTree"
   2: "location"
   3: "open_status"
   4: "open_info"
   5: "close_status"
   6: "close_info"
   7: "TimeInstant"
⌄attrs: Object
  › refField: Object
  › refAppleTree: Object
  › location: Object
  ⌄ open_status: Object
      value: "PENDING"
      type: "commandStatus"
    ⌄ md: Object
      ⌄ TimeInstant: Object
          type: "ISO8601"
          value: "2018-08-30T23:52:40.686Z"
    ⌄ mdNames: Array
        0: "TimeInstant"
      creDate: 1535666788
      modDate: 1535673160
  ⌄ open_info: Object
      value: "Opened ok"
      type: "commandResult"
    ⌄ md: Object
      ⌄ TimeInstant: Object
          type: "ISO8601"
          value: "2018-08-30T23:43:16.401Z"
    ⌄ mdNames: Array
        0: "TimeInstant"
      creDate: 1535666788
      modDate: 1535672596
```

Figure 4.60 State of the Valve Actuator in the Valve Actuator Entity

Figure 4.61 shows the results of making a query for the actuator information.



http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	openiot	
Key	Value	Description

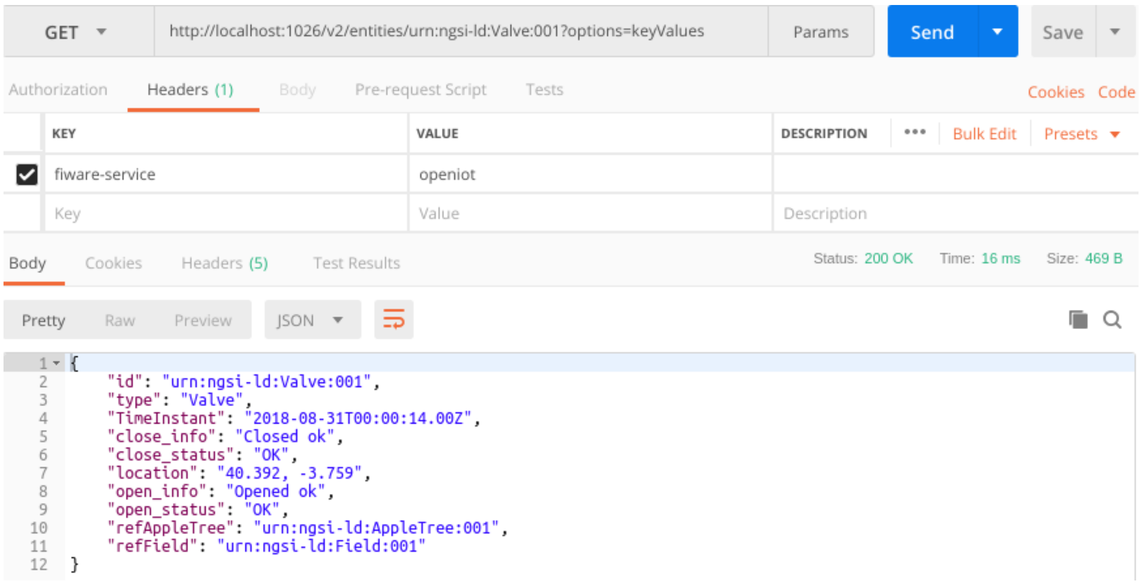
Status: 200 OK Time: 19 ms Size: 466 B

```
{
    "id": "urn:ngsi-ld:Valve:001",
    "type": "Valve",
    "TimeInstant": "2018-08-30T23:55:02.00Z",
    "close_info": " ",
    "close_status": "UNKNOWN",
    "location": "40.392, -3.759",
    "open_info": "Opened ok",
    "open_status": "OK",
    "refAppleTree": "urn:ngsi-ld:AppleTree:001",
    "refField": "urn:ngsi-ld:Field:001"
}
```

Figure 4.61 Valve Actuator Key Values After a “open” Command

The reply sent by the IoT Devices after executing the command serves to fill in the “_info” and “_status” attributes, which indicate that the command was executed with success.

If subsequently a command to close the valve was sent (Figure 4.62) then the valve attributes “close_info” and “close_status” would be updated as expected, however the attributes associated to the open command would remain as before therefore it would be necessary to delete the attributes to avoid misinformation.



http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	openiot	
Key	Value	Description

Status: 200 OK Time: 16 ms Size: 469 B

```
{
    "id": "urn:ngsi-ld:Valve:001",
    "type": "Valve",
    "TimeInstant": "2018-08-31T00:00:14.00Z",
    "close_info": "Closed ok",
    "close_status": "OK",
    "location": "40.392, -3.759",
    "open_info": "Opened ok",
    "open_status": "OK",
    "refAppleTree": "urn:ngsi-ld:AppleTree:001",
    "refField": "urn:ngsi-ld:Field:001"
}
```

Figure 4.62 Valve Actuator Key Values After a “open” and “close” Command

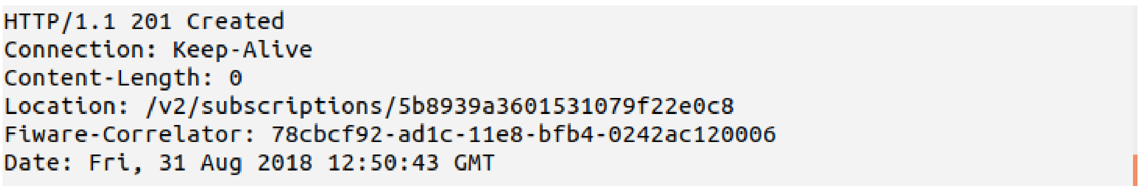
After completing the communication tests, it was determined that the everything is working as expected.

4.2.7 Subscriptions

To test the if the notifications functionality is working properly an echo server available in the Orion GitHub webpage [73] was used to visualize the notifications sent by the Orion Context Broker.

For this test it was created a subscription which would notify the user (echo server) whenever a weather sensor in field A (the individual device ID was not specified only the sensor type) returned a temperature above 30ºC. For that a script, named “10_use_case_subscriptions.sh”, containing the subscription was created and whose contents are available in Annex Y.

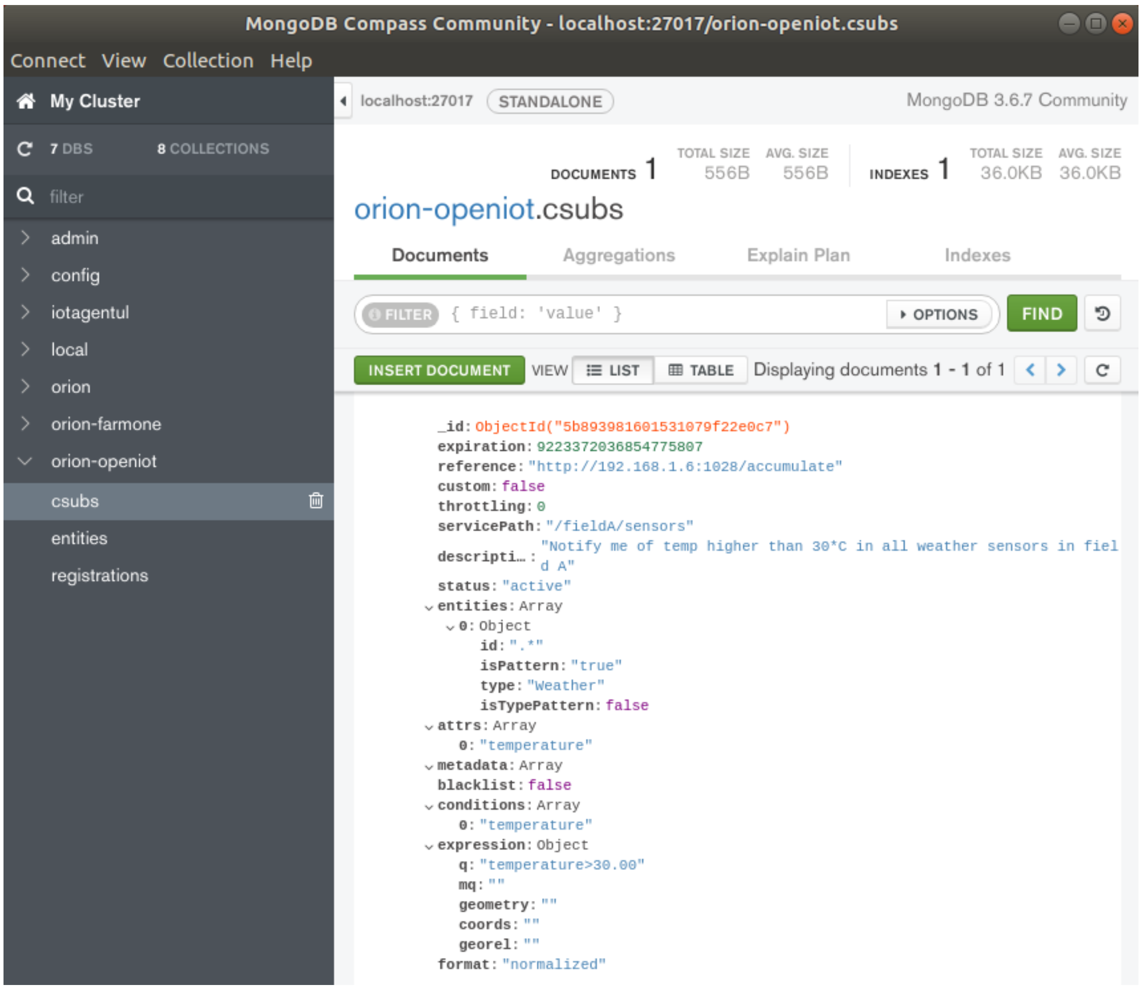
Figure 4.63 shows the command output.



```
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/subscriptions/5b8939a3601531079f22e0c8
Fiware-Correlator: 78cbcf92-ad1c-11e8-bfb4-0242ac120006
Date: Fri, 31 Aug 2018 12:50:43 GMT
```

Figure 4.63 Output of the Commands That Enabled Notifications

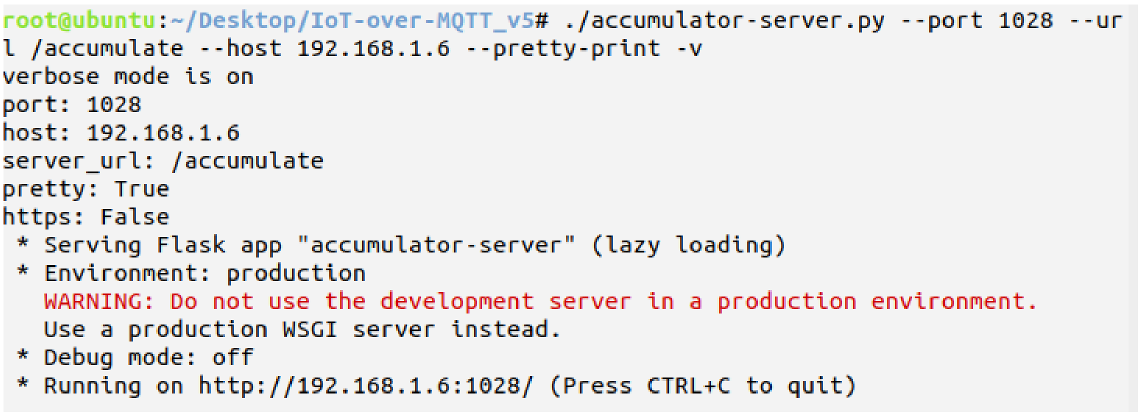
It is also observed, in Figure 4.64, that a new directory was created in the Orion database which contains information about the subscriptions.


MongoDB Compass Community - localhost:27017/orion-openiot.csubs
Connect View Collection Help
My Cluster
7 DBS 8 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
csubs
entities
registrations
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 1 TOTAL SIZE 556B AVG. SIZE 556B
INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB
orion-openiot.csubs
Documents Aggregations Explain Plan Indexes
FILTER { field: 'value' } OPTIONS FIND
INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 1 of 1

```
_id: ObjectId("5b893981601531079f22e0c7")
expiration: 9223372036854775807
reference: "http://192.168.1.6:1028/accumulate"
custom: false
throttling: 0
servicePath: "/fieldA/sensors"
descripti… : "Notify me of temp higher than 30°C in all weather sensors in fiel
             d A"
status: "active"
entities: Array
  0: Object
     id: ".*"
     isPattern: "true"
     type: "Weather"
     isTypePattern: false
attrs: Array
  0: "temperature"
metadata: Array
blacklist: false
conditions: Array
  0: "temperature"
expression: Object
  q: "temperature>30.00"
  mq: ""
  geometry: ""
  coords: ""
  georel: ""
format: "normalized"
```

Figure 4.64 Directory Containing Subscriptions

Figure 4.65 shows the echo server startup.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v5# ./accumulator-server.py --port 1028 --ur
l /accumulate --host 192.168.1.6 --pretty-print -v
verbose mode is on
port: 1028
host: 192.168.1.6
server_url: /accumulate
pretty: True
https: False
 * Serving Flask app "accumulator-server" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://192.168.1.6:1028/ (Press CTRL+C to quit)
```

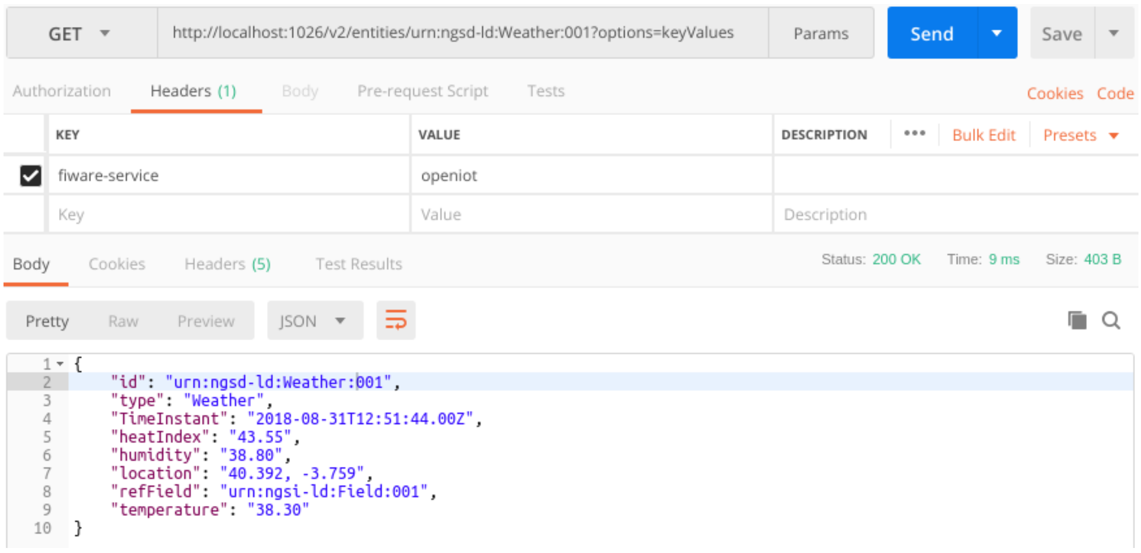
Figure 4.65 Startup of the Echo Server Used to Visualize Notification Sent by Orion

Figure 4.66 shows a notification in echo server, informing that the “temperature” attribute is over 30ºC as it was defined.

```
POST http://192.168.1.6:1028/accumulate
Fiware-Servicepath: /fieldA/sensors
Content-Length: 234
X-Auth-Token: null
User-Agent: orion/1.14.0 libcurl/7.29.0
Ngsiv2-Attrsformat: normalized
Host: 192.168.1.6:1028
Accept: application/json
Fiware-Service: openiot
Content-Type: application/json; charset=utf-8
Fiware-Correlator: 9d6b60f6-ad1c-11e8-9719-0242ac120006

{
    "data": [
        {
            "id": "urn:ngsd-ld:Weather:001",
            "temperature": {
                "metadata": {
                    "TimeInstant": {
                        "type": "ISO8601",
                        "value": "2018-08-31T12:51:44.651Z"
                    }
                },
                "type": "degrees",
                "value": "38.30"
            },
            "type": "Weather"
        }
    ],
    "subscriptionId": "5b8939a3601531079f22e0c8"
}
=======================================
```

Figure 4.66 Notification Received by the Echo Server

Figure 4.67 shows the results of a query for the Weather sensor key values.



http://localhost:1026/v2/entities/urn:ngsd-ld:Weather:001?options=keyValues

KEY	VALUE	DESCRIPTION
✓ fiware-service	openiot	
Key	Value	Description

Status: 200 OK Time: 9 ms Size: 403 B

```
{
    "id": "urn:ngsd-ld:Weather:001",
    "type": "Weather",
    "TimeInstant": "2018-08-31T12:51:44.00Z",
    "heatIndex": "43.55",
    "humidity": "38.80",
    "location": "40.392, -3.759",
    "refField": "urn:ngsi-ld:Field:001",
    "temperature": "38.30"
}
```

Figure 4.67 Weather Sensor Key Values

Some problems were encountered when using subscriptions with expressions, as not always the notification was sent. When not using expressions, the notifications work flawlessly.

The Orion Context Broker doesn't support rules, as such these must be implemented in the frontend part by handling the notifications sent by Orion and then sending commands for Orion to perform a given task.

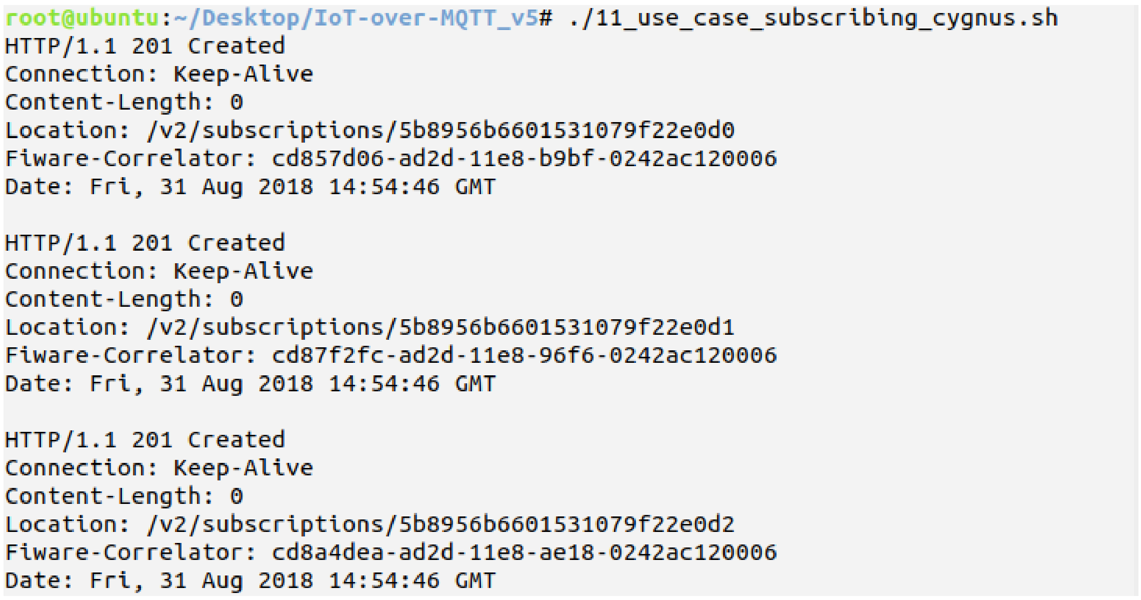
4.2.8 Data Persistence

The data persistence is done by the pair of FIWARE GEs, Cygnus and STH-Comet, which are configured to work in formal mode (done in the "docker-compose.yml" file), in which Cygnus is responsible of data collection and STH-Comet for reading the collected data.

For Cygnus to collect data it must before receive said data, which is done through subscriptions similar to the last test. However, here Orion uses subscriptions to notify Cygnus of changes in the context data.

To create the subscriptions that notify Cygnus of all sensor data changes, a shell script named "11_use_case_subscriptions_cygnus.sh" was created and whose contents are available in Annex Z.

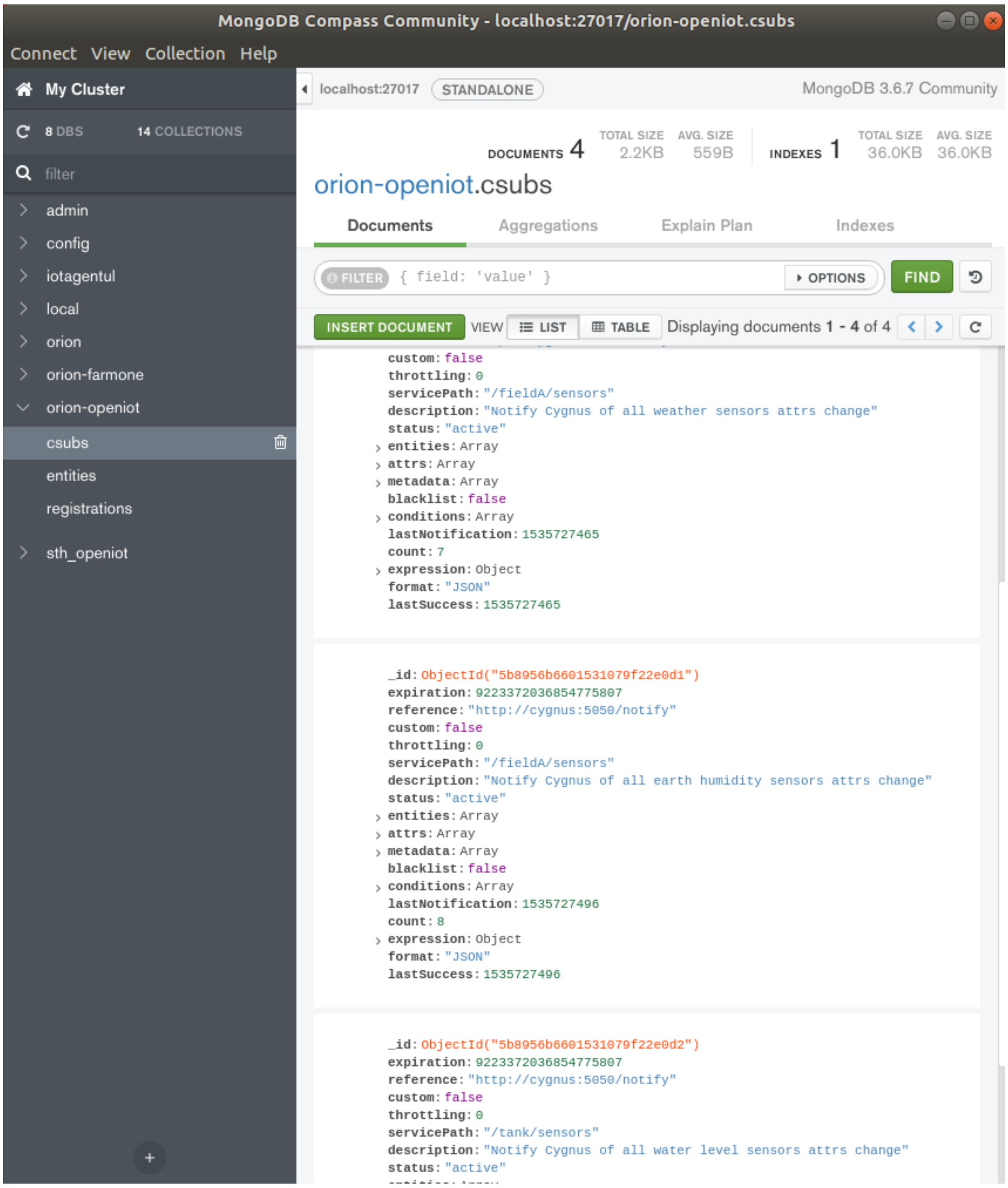
Figure 4.68 shows the commands output.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v5# ./11_use_case_subscribing_cygnus.sh
HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/subscriptions/5b8956b6601531079f22e0d0
Fiware-Correlator: cd857d06-ad2d-11e8-b9bf-0242ac120006
Date: Fri, 31 Aug 2018 14:54:46 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/subscriptions/5b8956b6601531079f22e0d1
Fiware-Correlator: cd87f2fc-ad2d-11e8-96f6-0242ac120006
Date: Fri, 31 Aug 2018 14:54:46 GMT

HTTP/1.1 201 Created
Connection: Keep-Alive
Content-Length: 0
Location: /v2/subscriptions/5b8956b6601531079f22e0d2
Fiware-Correlator: cd8a4dea-ad2d-11e8-ae18-0242ac120006
Date: Fri, 31 Aug 2018 14:54:46 GMT
```

Figure 4.68 Output of the Commands That Notified Cygnus of Data Alterations

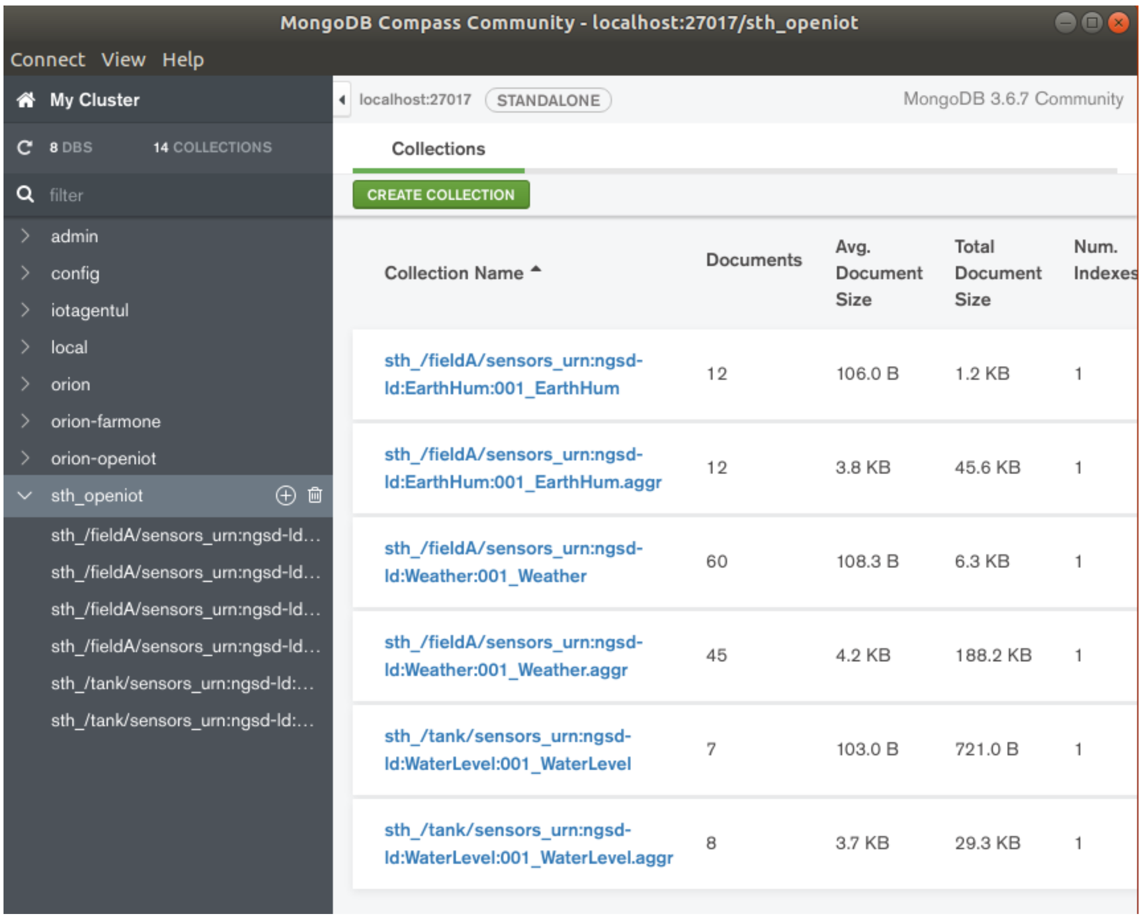
Figure 4.69 shows that after the commands execution, the subscriptions that notify Cygnus were added to the database.


MongoDB Compass Community - localhost:27017/orion-openiot.csubs
Connect View Collection Help
My Cluster
8 DBS 14 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
csubs
entities
registrations
sth_openiot
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 4 TOTAL SIZE 2.2KB AVG. SIZE 559B
INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB
orion-openiot.csubs
Documents
Aggregations
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 4 of 4

```
  custom: false
  throttling: 0
  servicePath: "/fieldA/sensors"
  description: "Notify Cygnus of all weather sensors attrs change"
  status: "active"
> entities: Array
> attrs: Array
> metadata: Array
  blacklist: false
> conditions: Array
  lastNotification: 1535727465
  count: 7
> expression: Object
  format: "JSON"
  lastSuccess: 1535727465

  _id: ObjectId("5b8956b6601531079f22e0d1")
  expiration: 9223372036854775807
  reference: "http://cygnus:5050/notify"
  custom: false
  throttling: 0
  servicePath: "/fieldA/sensors"
  description: "Notify Cygnus of all earth humidity sensors attrs change"
  status: "active"
> entities: Array
> attrs: Array
> metadata: Array
  blacklist: false
> conditions: Array
  lastNotification: 1535727496
  count: 8
> expression: Object
  format: "JSON"
  lastSuccess: 1535727496

  _id: ObjectId("5b8956b6601531079f22e0d2")
  expiration: 9223372036854775807
  reference: "http://cygnus:5050/notify"
  custom: false
  throttling: 0
  servicePath: "/tank/sensors"
  description: "Notify Cygnus of all water level sensors attrs change"
  status: "active"
```

Figure 4.69 Subscriptions that Notify Cygnus in the Orion Database

Figure 4.70 shows that when the subscriptions were created, a new database where the subscribed data is collected is created.

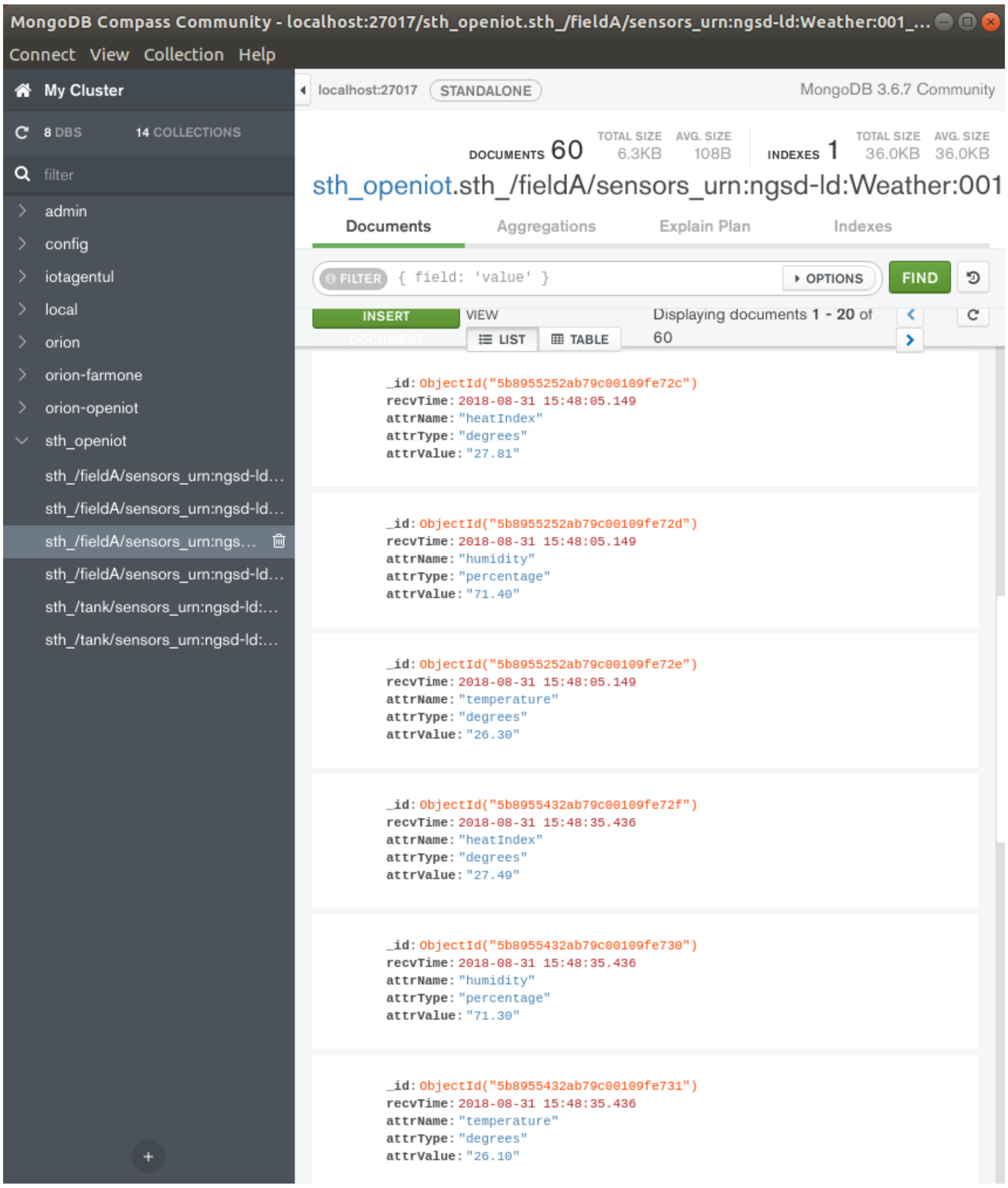


admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
sth_openiot
sth_/fieldA/sensors_urn:ngsd-Id…
sth_/fieldA/sensors_urn:ngsd-Id…
sth_/fieldA/sensors_urn:ngsd-Id…
sth_/fieldA/sensors_urn:ngsd-Id…
sth_/tank/sensors_urn:ngsd-Id:…
sth_/tank/sensors_urn:ngsd-Id:…

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes
sth_/fieldA/sensors_urn:ngsd-Id:EarthHum:001_EarthHum	12	106.0 B	1.2 KB	1
sth_/fieldA/sensors_urn:ngsd-Id:EarthHum:001_EarthHum.aggr	12	3.8 KB	45.6 KB	1
sth_/fieldA/sensors_urn:ngsd-Id:Weather:001_Weather	60	108.3 B	6.3 KB	1
sth_/fieldA/sensors_urn:ngsd-Id:Weather:001_Weather.aggr	45	4.2 KB	188.2 KB	1
sth_/tank/sensors_urn:ngsd-Id:WaterLevel:001_WaterLevel	7	103.0 B	721.0 B	1
sth_/tank/sensors_urn:ngsd-Id:WaterLevel:001_WaterLevel.aggr	8	3.7 KB	29.3 KB	1

Figure 4.70 Cygnus Database where Collected Data is Saved

Figure 4.71 shows some of the data collected for the weather sensor.


MongoDB Compass Community - localhost:27017/sth_openiot.sth_/fieldA/sensors_urn:ngsd-ld:Weather:001_...
Connect View Collection Help
My Cluster
8 DBS 14 COLLECTIONS
filter
admin
config
iotagentul
local
orion
orion-farmone
orion-openiot
sth_openiot
sth_/fieldA/sensors_urn:ngsd-ld...
sth_/fieldA/sensors_urn:ngsd-ld...
sth_/fieldA/sensors_urn:ngs...
sth_/fieldA/sensors_urn:ngsd-ld...
sth_/tank/sensors_urn:ngsd-ld:...
sth_/tank/sensors_urn:ngsd-ld:...
localhost:27017 STANDALONE
MongoDB 3.6.7 Community
DOCUMENTS 60 TOTAL SIZE 6.3KB AVG. SIZE 108B
INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB
sth_openiot.sth_/fieldA/sensors_urn:ngsd-ld:Weather:001
Documents
Aggregations
Explain Plan
Indexes
FILTER { field: 'value' }
OPTIONS
FIND
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 20 of 60

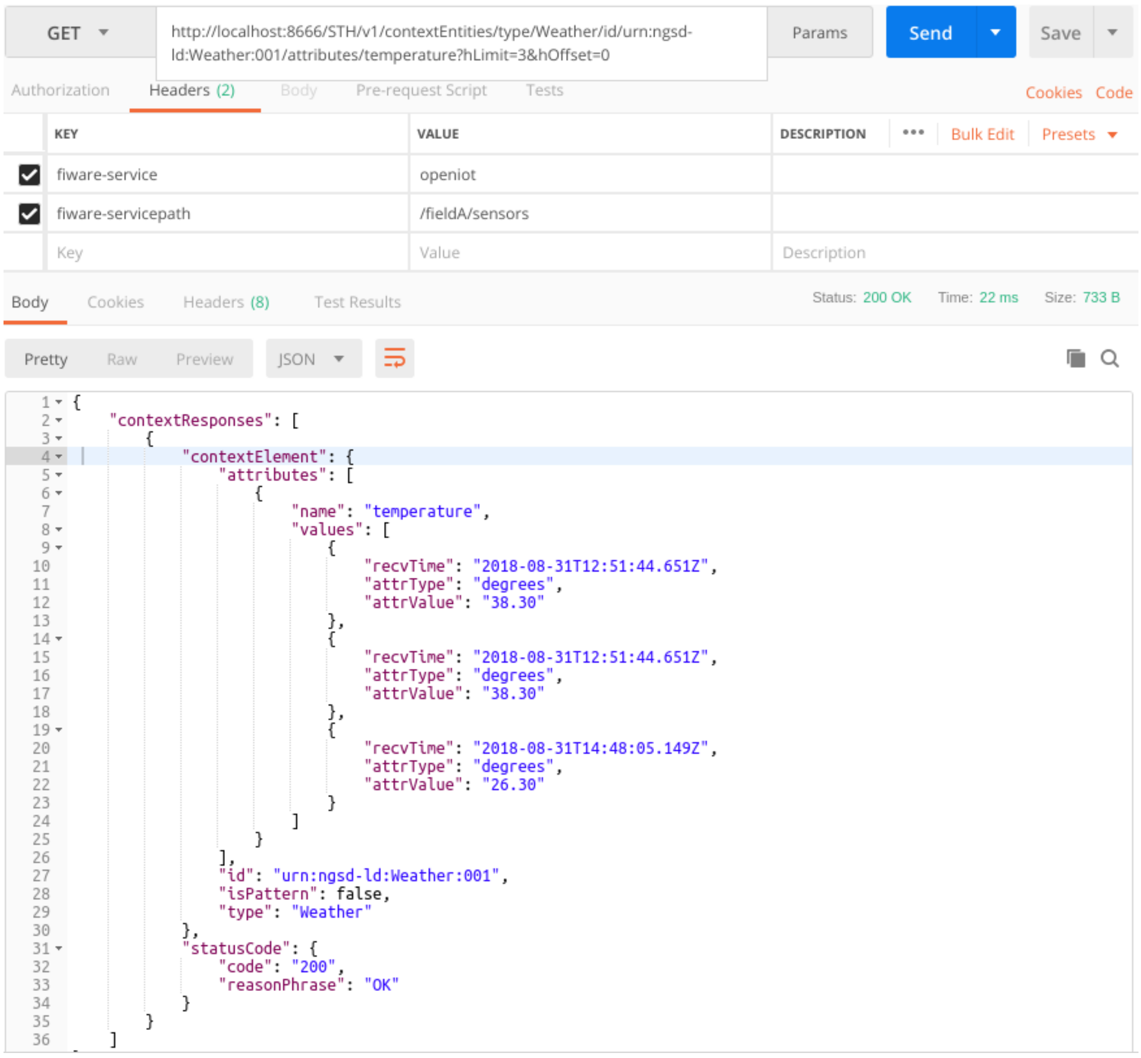
```
_id: ObjectId("5b8955252ab79c00109fe72c")
recvTime: 2018-08-31 15:48:05.149
attrName: "heatIndex"
attrType: "degrees"
attrValue: "27.81"

_id: ObjectId("5b8955252ab79c00109fe72d")
recvTime: 2018-08-31 15:48:05.149
attrName: "humidity"
attrType: "percentage"
attrValue: "71.40"

_id: ObjectId("5b8955252ab79c00109fe72e")
recvTime: 2018-08-31 15:48:05.149
attrName: "temperature"
attrType: "degrees"
attrValue: "26.30"

_id: ObjectId("5b8955432ab79c00109fe72f")
recvTime: 2018-08-31 15:48:35.436
attrName: "heatIndex"
attrType: "degrees"
attrValue: "27.49"

_id: ObjectId("5b8955432ab79c00109fe730")
recvTime: 2018-08-31 15:48:35.436
attrName: "humidity"
attrType: "percentage"
attrValue: "71.30"

_id: ObjectId("5b8955432ab79c00109fe731")
recvTime: 2018-08-31 15:48:35.436
attrName: "temperature"
attrType: "degrees"
attrValue: "26.10"
```

Figure 4.71 Historical Data of the Weather Sensor

It was then concluded that Cygnus is working properly as it can collect and save data.

4.2.9 Time-Series Data Queries

As the STH-Comet is only used to visualize the data collected by Cygnus, therefore to test if it is working properly it is necessary to make a data query. For this test (Figure 4.72) it was made a data query that listed the first 3 sampled temperature values of the weather sensor.



http://localhost:8666/STH/v1/contextEntities/type/Weather/id/urn:ngsd-ld:Weather:001/attributes/temperature?hLimit=3&hOffset=0

	KEY	VALUE	DESCRIPTION
✓	fiware-service	openiot	
✓	fiware-servicepath	/fieldA/sensors	
	Key	Value	Description

Status: 200 OK Time: 22 ms Size: 733 B

```
{
    "contextResponses": [
        {
            "contextElement": {
                "attributes": [
                    {
                        "name": "temperature",
                        "values": [
                            {
                                "recvTime": "2018-08-31T12:51:44.651Z",
                                "attrType": "degrees",
                                "attrValue": "38.30"
                            },
                            {
                                "recvTime": "2018-08-31T12:51:44.651Z",
                                "attrType": "degrees",
                                "attrValue": "38.30"
                            },
                            {
                                "recvTime": "2018-08-31T14:48:05.149Z",
                                "attrType": "degrees",
                                "attrValue": "26.30"
                            }
                        ]
                    }
                ],
                "id": "urn:ngsd-ld:Weather:001",
                "isPattern": false,
                "type": "Weather"
            },
            "statusCode": {
                "code": "200",
                "reasonPhrase": "OK"
            }
        }
    ]
```

Figure 4.72 Results of a Query for the First Three Collected Values of the Weather Sensor

It was then confirmed that the STH-Comet can return the historical data collected by Cygnus, therefore concluding the IoT System tests.

Intentionally Left Blank

Chapter 5 – Conclusions

5.1 Main Conclusions

With this dissertation it was demonstrated how to use the FIWARE Platform and its technologies to develop a modular universal IoT System able to communicate, control and collet data from IoT devices over a wireless environment.

In the beginning, during the investigation period, enormous difficulties were encountered due to poor documentation and lack of practical examples, it was possible to comprehend how to the Orion Context Broker Generic Enabler worker and how it was used, and what Generic Enablers would have to be used, nevertheless, progress was very slow. However, with the progressive release by FIWARE of a series of practical tutorials, starting in April and continuing during the following months, it was possible to understand how to the other necessary GEs worked and how to implement them.

Afterwards, the system was developed having been used the following FIWARE components or Generic Enablers: Orion Context Broker, IoT Agent for Ultralight 2.0 Protocol, Cygnus and STH-Comet; and also, a MongoDB database to store data, the Compass GUI to visualize data in the databases, and a Mosquitto MQTT Broker. The interaction with the system was done though cURL commands or by using the Postman program.

During the system testing phase, each component was tested by sending commands directly to them and by implementing a simple use case that simulated a real-world implementation of IoT devices with sensors and actuators, which proved that the system was working well as intended.

However, due to the time and effort spent on studying and understanding FIWARE and its components some of the desired features were not implemented, the GEs which implement security were not used (the tutorials were only released in August, and continue to be released), a GUI was also not developed, and other IoT communication protocols, in addition to MQTT, were also not implemented.

In spite of this, the student hopes that this dissertation can be useful as an introductory manual on FIWARE for other who needs to use the FIWARE Platform and its technologies or continue developing this project.

5.2 Future Work

Future work to be done in this project include implementing security, a GUI, other IoT protocols and respective IoT Agents, and replacing the GEs used for Context History (STH-Comet and Cygnus) for a new and recent GE.

Security:

- The use of the Keyrock Identity Management GE implements OAuth 2.0 authentication for users and devices, and user profile management, which is kept in a SQL database;
- The use of the Wilma Proxy GE serves as a Policy Enforcement Point as well as a proxy isolating the rest of the system from the frontend, only allowing authorized users to interact with the backend;
- The AuthZForce PDP/PAP GE serves as a Policy Decision Point and works in tandem with Wilma to secure the system.

GUI (frontend):

- The GUI which can be an application or website, can be developed using the appropriate technologies as an application or website, which the student never used and didn’t have time to learn how to. However, FIWARE also has a GE named Wirecloud which can be used to develop operational dashboards, but the problem of poor documentation and lack of practical examples prevented the use of this GE.

Other IoT protocols:

- The MQTT protocol was used in this project, however, MQTT is usually used in Wi-Fi environments were the connection to the Internet is good. If the system was used in a situation where the devices were located outdoors and distributed over a large area, like in the test use case, then it would be unrealistic to cover said area with Wi-Fi due to the large number of antennas needed and the work involved. That is where the use of protocols such as LoRaWAN which already has an IoT Agent available and has a coverage radius of kilometers can be used, although it also requires specific extra hardware that increases the costs of implementation considerably.

Replacement of the context history GEs [74]:

- The QuantumLeap Generic Enabler, made available in November, has the same functions as the STH-Comet GE, however, while the later doesn't yet support the NGSIv2 API, is tied to MongoDB and somewhat obsolete, QuantumLeap supports several time-series databases (e.g., InfluxDB, RethinkDB and Crate) and replaces both STH-Comet and Cygnus. In its current state it only supports Crate, as it is easy scalable, supports geo-queries natively, has a nice SQL-like querying and supports integration with visualization tools like Grafana

With the implementation of the security GEs and the replacement of the STH-Comet and Cygnus with QuantumLeap, the system architecture would evolve to be like the one shown in Figure 5.1. To this architecture other IoT Agents could also be added.

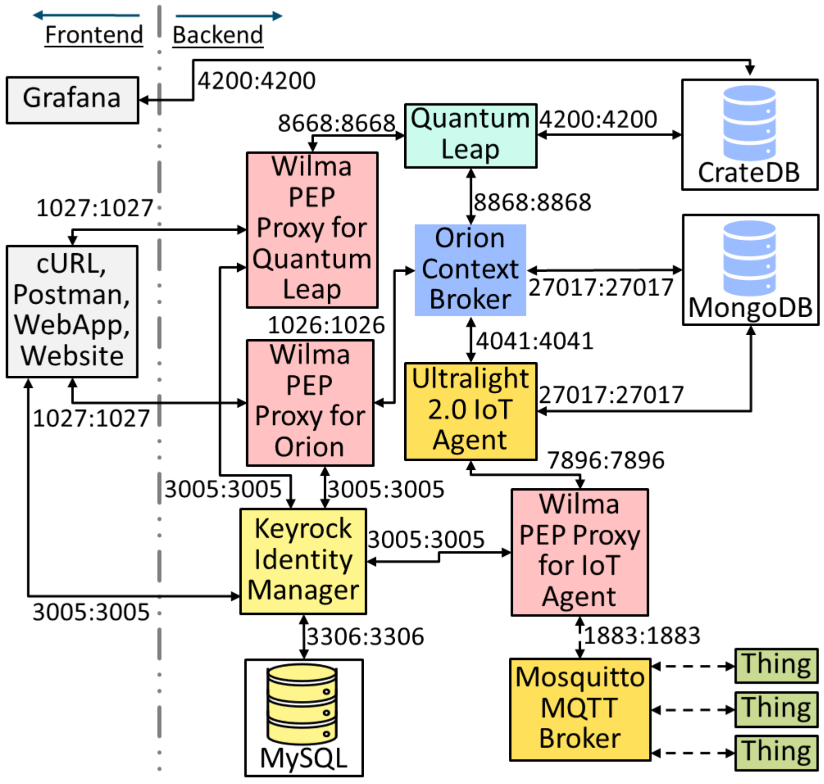


Figure 5.1 Future System Architecture Block Diagram

Annexes

Annex A – FIWARE Orion Context Broker Configuration File

```
# Copyright 2013 Telefonica Investigacion y Desarrollo, S.A.U
#
# This file is part of Orion Context Broker.
#
# Orion Context Broker is free software: you can redistribute it
and/or
# modify it under the terms of the GNU Affero General Public License
as
# published by the Free Software Foundation, either version 3 of the
# License, or (at your option) any later version.
#
# Orion Context Broker is distributed in the hope that it will be
useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Affero
# General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public
License
# along with Orion Context Broker. If not, see
http://www.gnu.org/licenses/.
#
# For those usages not covered by this license please contact with
# iot_support at tid dot es

#####
#
# Configuration file for orion-broker
#
#####

# BROKER_USER - Who to run orion-broker as. Note that you may need
to use root if you want
# to run Orion in a privileged port (<1024)
BROKER_USER=orion

# BROKER_PORT - the port/socket where orion-broker will listen for
connections
BROKER_PORT=1026

# BROKER_LOG_DIR - Where to log to
BROKER_LOG_DIR=/var/log/contextBroker

# BROKER_LOG_LEVEL - Log File Level
BROKER_LOG_LEVEL=WARN

# BROKER_PID_FILE - Where to store the pid for orion-broker
BROKER_PID_FILE=/var/run/contextBroker/contextBroker.pid

## Database configuration for orion-broker
BROKER_DATABASE_HOST=localhost
BROKER_DATABASE_NAME=orion
```

```
## Replica set configuration. Note that if you set this parameter,
the BROKER_DATABASE_HOST
## is interpreted as the list of host (or host:port) separated by
commas to use as
## replica set seed list (single element lists are also allowed). If
BROKER_DATABASE_RPL_SET
## parameter is unset, Orion CB assumes that the
BROKER_DATABASE_HOST is an stand-alone
## mongod instance
#BROKER_DATABASE_RPLSET=orion_rs

# Database authentication (not needed if MongoDB doesn't use --auth)
#BROKER_DATABASE_USER=orion
#BROKER_DATABASE_PASSWORD=orion

# Use the following variable if you need extra ops
#BROKER_EXTRA_OPS="-t 0-255"
# We need to start the CB with log append mode to not overwrite
previous logs and logrotate work correctly
BROKER_EXTRA_OPS="-multiservice -logAppend"

#### ADVANCED CONFIGURATION AREA ####

# The next environment variable generates, if it is defined as
'true', an archive in the
# path /var/cb_cores with the core file and the logs when the
process crash (the directory is
# automatically created if it doesn't previously exist). Use this
parameter for debug purposes.
# In addition, take into account the following:
#   * BROKER_USER is ignored and contextBroker will be started by
root user
#   * It is assumed bzip2 tool installed in the system
#   * Core generation requires CB to be launched with -fg and put in
background at shell level
#     (i.e. "with ending &"). Note this is different from regular CB
launch (i.e. not using -fg).
#     Check /etc/init.d/contextBroker for details
#   * The file names have the next format:
/var/cb_cores/CB_core_yyyymmdd_HHMMSS.tar.bz2, e.g.
/var/cb_cores/CB_core_220180108_151502.tar.bz2
#   * To avoid filling the disk only the last 8 cores and logs are
maintained.
#   * Have a look to the coredump_watcher.sh script (can be found at
Orion repo at https://github.com/telefonicaid/fiware-
orion/tree/master/scripts),
#     which can be used to send an email whenever a new core file is
generated
GENERATE_COREDUMP=false
```

Annex B – Mosquitto MQTT Broker Configuration File

```
# Config file for mosquitto
#
# See mosquitto.conf(5) for more information.
#
# Default values are shown, uncomment to change.
#
# Use the # character to indicate a comment, but only if it is the
# very first character on the line.

# =================================================================
# General configuration
# =================================================================

# Time in seconds to wait before resending an outgoing QoS=1 or
# QoS=2 message.
#retry_interval 20

# Time in seconds between updates of the $SYS tree.
# Set to 0 to disable the publishing of the $SYS tree.
#sys_interval 10

# Time in seconds between cleaning the internal message store of
# unreferenced messages. Lower values will result in lower memory
# usage but more processor time, higher values will have the
# opposite effect.
# Setting a value of 0 means the unreferenced messages will be
# disposed of as quickly as possible.
#store_clean_interval 10

# Write process id to a file. Default is a blank string which means
# a pid file shouldn't be written.
# This should be set to /var/run/mosquitto.pid if mosquitto is
# being run automatically on boot with an init script and
# start-stop-daemon or similar.
#pid_file

# When run as root, drop privileges to this user and its primary
# group.
# Leave blank to stay as root, but this is not recommended.
# If run as a non-root user, this setting has no effect.
# Note that on Windows this has no effect and so mosquitto should
# be started by the user you wish it to run as.
#user mosquitto

# The maximum number of QoS 1 and 2 messages currently inflight per
# client.
# This includes messages that are partway through handshakes and
# those that are being retried. Defaults to 20. Set to 0 for no
# maximum. Setting to 1 will guarantee in-order delivery of QoS 1
# and 2 messages.
#max_inflight_messages 20

# The maximum number of QoS 1 and 2 messages to hold in a queue
# above those that are currently in-flight.  Defaults to 100. Set
# to 0 for no maximum (not recommended).
# See also queue_qos0_messages.
#max_queued_messages 100
```

```
# Set to true to queue messages with QoS 0 when a persistent client
is
# disconnected. These messages are included in the limit imposed by
# max_queued_messages.
# Defaults to false.
# This is a non-standard option for the MQTT v3.1 spec but is
allowed in
# v3.1.1.
#queue_qos0_messages false

# This option sets the maximum publish payload size that the broker
will allow.
# Received messages that exceed this size will not be accepted by
the broker.
# The default value is 0, which means that all valid MQTT messages
are
# accepted. MQTT imposes a maximum payload size of 268435455 bytes.
#message_size_limit 0

# This option controls whether a client is allowed to connect with a
zero
# length client id or not. This option only affects clients using
MQTT v3.1.1
# and later. If set to false, clients connecting with a zero length
client id
# are disconnected. If set to true, clients will be allocated a
client id by
# the broker. This means it is only useful for clients with clean
session set
# to true.
#allow_zero_length_clientid true

# If allow_zero_length_clientid is true, this option allows you to
set a prefix
# to automatically generated client ids to aid visibility in logs.
#auto_id_prefix

# This option allows persistent clients (those with clean session
set to false)
# to be removed if they do not reconnect within a certain time
frame.
#
# This is a non-standard option in MQTT V3.1 but allowed in MQTT
v3.1.1.
#
# Badly designed clients may set clean session to false whilst using
a randomly
# generated client id. This leads to persistent clients that will
never
# reconnect. This option allows these clients to be removed.
#
# The expiration period should be an integer followed by one of h d
w m y for
# hour, day, week, month and year respectively. For example
#
# persistent_client_expiration 2m
# persistent_client_expiration 14d
# persistent_client_expiration 1y
#
# The default if not set is to never expire persistent clients.
#persistent_client_expiration
```

```
# If a client is subscribed to multiple subscriptions that overlap,
e.g. foo/#
# and foo/+/baz , then MQTT expects that when the broker receives a
message on
# a topic that matches both subscriptions, such as foo/bar/baz, then
the client
# should only receive the message once.
# Mosquitto keeps track of which clients a message has been sent to
in order to
# meet this requirement. The allow_duplicate_messages option allows
this
# behaviour to be disabled, which may be useful if you have a large
number of
# clients subscribed to the same set of topics and are very
concerned about
# minimising memory usage.
# It can be safely set to true if you know in advance that your
clients will
# never have overlapping subscriptions, otherwise your clients must
be able to
# correctly deal with duplicate messages even when then have QoS=2.
#allow_duplicate_messages false

# The MQTT specification requires that the QoS of a message
delivered to a
# subscriber is never upgraded to match the QoS of the subscription.
Enabling
# this option changes this behaviour. If upgrade_outgoing_qos is set
true,
# messages sent to a subscriber will always match the QoS of its
subscription.
# This is a non-standard option explicitly disallowed by the spec.
#upgrade_outgoing_qos false

# =================================================================
# Default listener
# =================================================================

# IP address/hostname to bind the default listener to. If not
# given, the default listener will not be bound to a specific
# address and so will be accessible to all network interfaces.
# bind_address ip-address/host name
#bind_address

# Port to use for the default listener.
#port 1883

# The maximum number of client connections to allow. This is
# a per listener setting.
# Default is -1, which means unlimited connections.
# Note that other process limits mean that unlimited connections
# are not really possible. Typically the default maximum number of
# connections possible is around 1024.
#max_connections -1

# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Websockets support is currently disabled by default at compile
time.
# Certificate based TLS may be used with websockets, except that
# only the cafile, certfile, keyfile and ciphers options are
supported.
#protocol mqtt

# When a listener is using the websockets protocol, it is possible
to serve
# http data as well. Set http_dir to a directory which contains the
files you
# wish to serve. If this option is not specified, then no normal
http
# connections will be possible.
#http_dir

# Set use_username_as_clientid to true to replace the clientid that
a client
# connected with with its username. This allows authentication to be
tied to
# the clientid, which means that it is possible to prevent one
client
# disconnecting another by using the same clientid.
# If a client connects with no username it will be disconnected as
not
# authorised when this option is set to true.
# Do not use in conjunction with clientid_prefixes.
# See also use_identity_as_username.
#use_username_as_clientid

# -----------------------------------------------------------------
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# The following options can be used to enable SSL/TLS support for
# this listener. Note that the recommended port for MQTT over TLS
# is 8883, but this must be set manually.
#
# See also the mosquitto-tls man page.

# At least one of cafile or capath must be defined. They both
# define methods of accessing the PEM encoded Certificate
# Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must
run
# "c_rehash <path to capath>" each time you add/remove a
certificate.
#cafile
#capath

# Path to the PEM encoded server certificate.
#certfile

# Path to the PEM encoded keyfile.
#keyfile

# This option defines the version of the TLS protocol to use for
this listener.
# The default value allows v1.2, v1.1 and v1.0, if they are all
supported by
```

```
# the version of openssl that the broker was compiled against. For
openssl >=
# 1.0.1 the valid values are tlsv1.2 tlsv1.1 and tlsv1. For openssl
< 1.0.1 the
# valid values are tlsv1.
#tls_version

# By default a TLS enabled listener will operate in a similar
fashion to a
# https enabled web server, in that the server has a certificate
signed by a CA
# and the client will verify that it is a trusted certificate. The
overall aim
# is encryption of the network traffic. By setting
require_certificate to true,
# the client must provide a valid certificate in order for the
network
# connection to proceed. This allows access to the broker to be
controlled
# outside of the mechanisms provided by MQTT.
#require_certificate false

# If require_certificate is true, you may set
use_identity_as_username to true
# to use the CN value from the client certificate as a username. If
this is
# true, the password_file option will not be used for this listener.
#use_identity_as_username false

# If you have require_certificate set to true, you can create a
certificate
# revocation list file to revoke access to particular client
certificates. If
# you have done this, use crlfile to point to the PEM encoded
revocation file.
#crlfile

# If you wish to control which encryption ciphers are used, use the
ciphers
# option. The list of available ciphers can be optained using the
"openssl
# ciphers" command and should be provided in the same format as the
output of
# that command.
# If unset defaults to
DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:@STRENGTH
#ciphers DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:@STRENGTH

# -----------------------------------------------------------------
# Pre-shared-key based SSL/TLS support
# -----------------------------------------------------------------
# The following options can be used to enable PSK based SSL/TLS
support for
# this listener. Note that the recommended port for MQTT over TLS is
8883, but
# this must be set manually.
#
# See also the mosquitto-tls man page and the "Certificate based
SSL/TLS
# support" section. Only one of certificate or PSK encryption
support can be
```

```
# enabled for any listener.

# The psk_hint option enables pre-shared-key support for this
listener and also
# acts as an identifier for this listener. The hint is sent to
clients and may
# be used locally to aid authentication. The hint is a free form
string that
# doesn't have much meaning in itself, so feel free to be creative.
# If this option is provided, see psk_file to define the pre-shared
keys to be
# used or create a security plugin to handle them.
#psk_hint

# Set use_identity_as_username to have the psk identity sent by the
client used
# as its username. Authentication will be carried out using the PSK
rather than
# the MQTT username/password and so password_file will not be used
for this
# listener.
#use_identity_as_username false

# When using PSK, the encryption ciphers used will be chosen from
the list of
# available PSK ciphers. If you want to control which ciphers are
available,
# use the "ciphers" option.  The list of available ciphers can be
optained
# using the "openssl ciphers" command and should be provided in the
same format
# as the output of that command.
#ciphers

# =================================================================
# Extra listeners
# =================================================================

# Listen on a port/ip address combination. By using this variable
# multiple times, mosquitto can listen on more than one port. If
# this variable is used and neither bind_address nor port given,
# then the default listener will not be started.
# The port number to listen on must be given. Optionally, an ip
# address or host name may be supplied as a second argument. In
# this case, mosquitto will attempt to bind the listener to that
# address and so restrict access to the associated network and
# interface. By default, mosquitto will listen on all interfaces.
# listener port-number [ip address/host name]
#listener

# The maximum number of client connections to allow. This is
# a per listener setting.
# Default is -1, which means unlimited connections.
# Note that other process limits mean that unlimited connections
# are not really possible. Typically the default maximum number of
# connections possible is around 1024.
#max_connections -1

# The listener can be restricted to operating within a topic
hierarchy using
```

```
# the mount_point option. This is achieved be prefixing the
mount_point string
# to all topics for any clients connected to this listener. This
prefixing only
# happens internally to the broker; the client will not see the
prefix.
#mount_point

# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Certificate based TLS may be used with websockets, except that
only the
# cafile, certfile, keyfile and ciphers options are supported.
#protocol mqtt

# When a listener is using the websockets protocol, it is possible
to serve
# http data as well. Set http_dir to a directory which contains the
files you
# wish to serve. If this option is not specified, then no normal
http
# connections will be possible.
#http_dir

# Set use_username_as_clientid to true to replace the clientid that
a client
# connected with with its username. This allows authentication to be
tied to
# the clientid, which means that it is possible to prevent one
client
# disconnecting another by using the same clientid.
# If a client connects with no username it will be disconnected as
not
# authorised when this option is set to true.
# Do not use in conjunction with clientid_prefixes.
# See also use_identity_as_username.
#use_username_as_clientid

# -----------------------------------------------------------------
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# The following options can be used to enable certificate based
SSL/TLS support
# for this listener. Note that the recommended port for MQTT over
TLS is 8883,
# but this must be set manually.
#
# See also the mosquitto-tls man page and the "Pre-shared-key based
SSL/TLS
# support" section. Only one of certificate or PSK encryption
support can be
# enabled for any listener.

# At least one of cafile or capath must be defined to enable
certificate based
# TLS encryption. They both define methods of accessing the PEM
encoded
# Certificate Authority certificates that have signed your server
certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must
run
# "c_rehash <path to capath>" each time you add/remove a
certificate.
#cafile
#capath

# Path to the PEM encoded server certificate.
#certfile

# Path to the PEM encoded keyfile.
#keyfile

# By default an TLS enabled listener will operate in a similar
fashion to a
# https enabled web server, in that the server has a certificate
signed by a CA
# and the client will verify that it is a trusted certificate. The
overall aim
# is encryption of the network traffic. By setting
require_certificate to true,
# the client must provide a valid certificate in order for the
network
# connection to proceed. This allows access to the broker to be
controlled
# outside of the mechanisms provided by MQTT.
#require_certificate false

# If require_certificate is true, you may set
use_identity_as_username to true
# to use the CN value from the client certificate as a username. If
this is
# true, the password_file option will not be used for this listener.
#use_identity_as_username false

# If you have require_certificate set to true, you can create a
certificate
# revocation list file to revoke access to particular client
certificates. If
# you have done this, use crlfile to point to the PEM encoded
revocation file.
#crlfile

# If you wish to control which encryption ciphers are used, use the
ciphers
# option. The list of available ciphers can be optained using the
"openssl
# ciphers" command and should be provided in the same format as the
output of
# that command.
#ciphers

# -----------------------------------------------------------------
# Pre-shared-key based SSL/TLS support
# -----------------------------------------------------------------
# The following options can be used to enable PSK based SSL/TLS
support for
# this listener. Note that the recommended port for MQTT over TLS is
8883, but
```

```
# this must be set manually.
#
# See also the mosquitto-tls man page and the "Certificate based
SSL/TLS
# support" section. Only one of certificate or PSK encryption
support can be
# enabled for any listener.

# The psk_hint option enables pre-shared-key support for this
listener and also
# acts as an identifier for this listener. The hint is sent to
clients and may
# be used locally to aid authentication. The hint is a free form
string that
# doesn't have much meaning in itself, so feel free to be creative.
# If this option is provided, see psk_file to define the pre-shared
keys to be
# used or create a security plugin to handle them.
#psk_hint

# Set use_identity_as_username to have the psk identity sent by the
client used
# as its username. Authentication will be carried out using the PSK
rather than
# the MQTT username/password and so password_file will not be used
for this
# listener.
#use_identity_as_username false

# When using PSK, the encryption ciphers used will be chosen from
the list of
# available PSK ciphers. If you want to control which ciphers are
available,
# use the "ciphers" option.  The list of available ciphers can be
optained
# using the "openssl ciphers" command and should be provided in the
same format
# as the output of that command.
#ciphers

# =================================================================
# Persistence
# =================================================================

# If persistence is enabled, save the in-memory database to disk
# every autosave_interval seconds. If set to 0, the persistence
# database will only be written when mosquitto exits. See also
# autosave_on_changes.
# Note that writing of the persistence database can be forced by
# sending mosquitto a SIGUSR1 signal.
#autosave_interval 1800

# If true, mosquitto will count the number of subscription changes,
retained
# messages received and queued messages and if the total exceeds
# autosave_interval then the in-memory database will be saved to
disk.
# If false, mosquitto will save the in-memory database to disk by
treating
# autosave_interval as a time in seconds.
#autosave_on_changes false
```

```
# Save persistent message data to disk (true/false).
# This saves information about all messages, including
# subscriptions, currently in-flight messages and retained
# messages.
# retained_persistence is a synonym for this option.
#persistence false

# The filename to use for the persistent database, not including
# the path.
#persistence_file mosquitto.db

# Location for persistent database. Must include trailing /
# Default is an empty string (current directory).
# Set to e.g. /var/lib/mosquitto/ if running as a proper service on
Linux or
# similar.
#persistence_location

# =================================================================
# Logging
# =================================================================

# Places to log to. Use multiple log_dest lines for multiple
# logging destinations.
# Possible destinations are: stdout stderr syslog topic file
#
# stdout and stderr log to the console on the named output.
#
# syslog uses the userspace syslog facility which usually ends up
# in /var/log/messages or similar.
#
# topic logs to the broker topic '$SYS/broker/log/<severity>',
# where severity is one of D, E, W, N, I, M which are debug, error,
# warning, notice, information and message. Message type severity is
used by
# the subscribe/unsubscribe log_types and publishes log messages to
# $SYS/broker/log/M/susbcribe or $SYS/broker/log/M/unsubscribe.
#
# The file destination requires an additional parameter which is the
file to be
# logged to, e.g. "log_dest file /var/log/mosquitto.log". The file
will be
# closed and reopened when the broker receives a HUP signal. Only a
single file
# destination may be configured.
#
# Note that if the broker is running as a Windows service it will
default to
# "log_dest none" and neither stdout nor stderr logging is
available.
# Use "log_dest none" if you wish to disable logging.
#log_dest stderr

# If using syslog logging (not on Windows), messages will be logged
to the
# "daemon" facility by default. Use the log_facility option to
choose which of
# local0 to local7 to log to instead. The option value should be an
integer
# value, e.g. "log_facility 5" to use local5.
```

```
#log_facility

# Types of messages to log. Use multiple log_type lines for logging
# multiple types of messages.
# Possible types are: debug, error, warning, notice, information,
# none, subscribe, unsubscribe, websockets, all.
# Note that debug type messages are for decoding the
incoming/outgoing
# network packets. They are not logged in "topics".
#log_type error
#log_type warning
#log_type notice
#log_type information

# Change the websockets logging level. This is a global option, it
is not
# possible to set per listener. This is an integer that is
interpreted by
# libwebsockets as a bit mask for its lws_log_levels enum. See the
# libwebsockets documentation for more details. "log_type
websockets" must also
# be enabled.
#websockets_log_level 0

# If set to true, client connection and disconnection messages will
be included
# in the log.
#connection_messages true

# If set to true, add a timestamp value to each log message.
#log_timestamp true

# =================================================================
# Security
# =================================================================

# If set, only clients that have a matching prefix on their
# clientid will be allowed to connect to the broker. By default,
# all clients may connect.
# For example, setting "secure-" here would mean a client "secure-
# client" could connect but another with clientid "mqtt" couldn't.
#clientid_prefixes

# Boolean value that determines whether clients that connect
# without providing a username are allowed to connect. If set to
# false then a password file should be created (see the
# password_file option) to control authenticated client access.
# Defaults to true.
#allow_anonymous true

# In addition to the clientid_prefixes, allow_anonymous and TLS
# authentication options, username based authentication is also
# possible. The default support is described in "Default
# authentication and topic access control" below. The auth_plugin
# allows another authentication method to be used.
# Specify the path to the loadable plugin and see the
# "Authentication and topic access plugin options" section below.
#auth_plugin

# -----------------------------------------------------------------
# Default authentication and topic access control
# -----------------------------------------------------------------

# Control access to the broker using a password file. This file can
be
# generated using the mosquitto_passwd utility. If TLS support is
not compiled
# into mosquitto (it is recommended that TLS support should be
included) then
# plain text passwords are used, in which case the file should be a
text file
# with lines in the format:
# username:password
# The password (and colon) may be omitted if desired, although this
# offers very little in the way of security.
#
# See the TLS client require_certificate and
use_identity_as_username options
# for alternative authentication options.
#password_file

# Access may also be controlled using a pre-shared-key file. This
requires
# TLS-PSK support and a listener configured to use it. The file
should be text
# lines in the format:
# identity:key
# The key should be in hexadecimal format without a leading "0x".
#psk_file

# Control access to topics on the broker using an access control
list
# file. If this parameter is defined then only the topics listed
will
# have access.
# If the first character of a line of the ACL file is a # it is
treated as a
# comment.
# Topic access is added with lines of the format:
#
# topic [read|write|readwrite] <topic>
#
# The access type is controlled using "read", "write" or
"readwrite". This
# parameter is optional (unless <topic> contains a space character)
- if not
# given then the access is read/write.  <topic> can contain the + or
#
# wildcards as in subscriptions.
#
# The first set of topics are applied to anonymous clients, assuming
# allow_anonymous is true. User specific topic ACLs are added after
a
# user line as follows:
#
# user <username>
#
# The username referred to here is the same as in password_file. It
is
# not the clientid.
#
#
```

```
# If is also possible to define ACLs based on pattern substitution
within the
# topic. The patterns available for substition are:
#
# %c to match the client id of the client
# %u to match the username of the client
#
# The substitution pattern must be the only text for that level of
hierarchy.
#
# The form is the same as for the topic keyword, but using pattern
as the
# keyword.
# Pattern ACLs apply to all users even if the "user" keyword has
previously
# been given.
#
# If using bridges with usernames and ACLs, connection messages can
be allowed
# with the following pattern:
# pattern write $SYS/broker/connection/%c/state
#
# pattern [read|write|readwrite] <topic>
#
# Example:
#
# pattern write sensor/%u/data
#
#acl_file

# -----------------------------------------------------------------
# Authentication and topic access plugin options
# -----------------------------------------------------------------

# If the auth_plugin option above is used, define options to pass to
the
# plugin here as described by the plugin instructions. All options
named
# using the format auth_opt_* will be passed to the plugin, for
example:
#
# auth_opt_db_host
# auth_opt_db_port
# auth_opt_db_username
# auth_opt_db_password


# =================================================================
# Bridges
# =================================================================

# A bridge is a way of connecting multiple MQTT brokers together.
# Create a new bridge using the "connection" option as described
below. Set
# options for the bridges using the remaining parameters. You must
specify the
# address and at least one topic to subscribe to.
# Each connection must have a unique name.
# The address line may have multiple host address and ports
specified. See
```

```
# below in the round_robin description for more details on bridge
behaviour if
# multiple addresses are used.
# The direction that the topic will be shared can be chosen by
# specifying out, in or both, where the default value is out.
# The QoS level of the bridged communication can be specified with
the next
# topic option. The default QoS level is 0, to change the QoS the
topic
# direction must also be given.
# The local and remote prefix options allow a topic to be remapped
when it is
# bridged to/from the remote broker. This provides the ability to
place a topic
# tree in an appropriate location.
# For more details see the mosquitto.conf man page.
# Multiple topics can be specified per connection, but be careful
# not to create any loops.
# If you are using bridges with cleansession set to false (the
default), then
# you may get unexpected behaviour from incoming topics if you
change what
# topics you are subscribing to. This is because the remote broker
keeps the
# subscription for the old topic. If you have this problem, connect
your bridge
# with cleansession set to true, then reconnect with cleansession
set to false
# as normal.
#connection <name>
#address <host>[:<port>] [<host>[:<port>]]
#topic <topic> [[[out | in | both] qos-level] local-prefix remote-
prefix]

# Set the version of the MQTT protocol to use with for this bridge.
Can be one
# of mqttv31 or mqttv311. Defaults to mqttv31.
#bridge_protocol_version mqttv31

# If a bridge has topics that have "out" direction, the default
behaviour is to
# send an unsubscribe request to the remote broker on that topic.
This means
# that changing a topic direction from "in" to "out" will not keep
receiving
# incoming messages. Sending these unsubscribe requests is not
always
# desirable, setting bridge_attempt_unsubscribe to false will
disable sending
# the unsubscribe request.
#bridge_attempt_unsubscribe true

# If the bridge has more than one address given in the
address/addresses
# configuration, the round_robin option defines the behaviour of the
bridge on
# a failure of the bridge connection. If round_robin is false, the
default
# value, then the first address is treated as the main bridge
connection. If
```

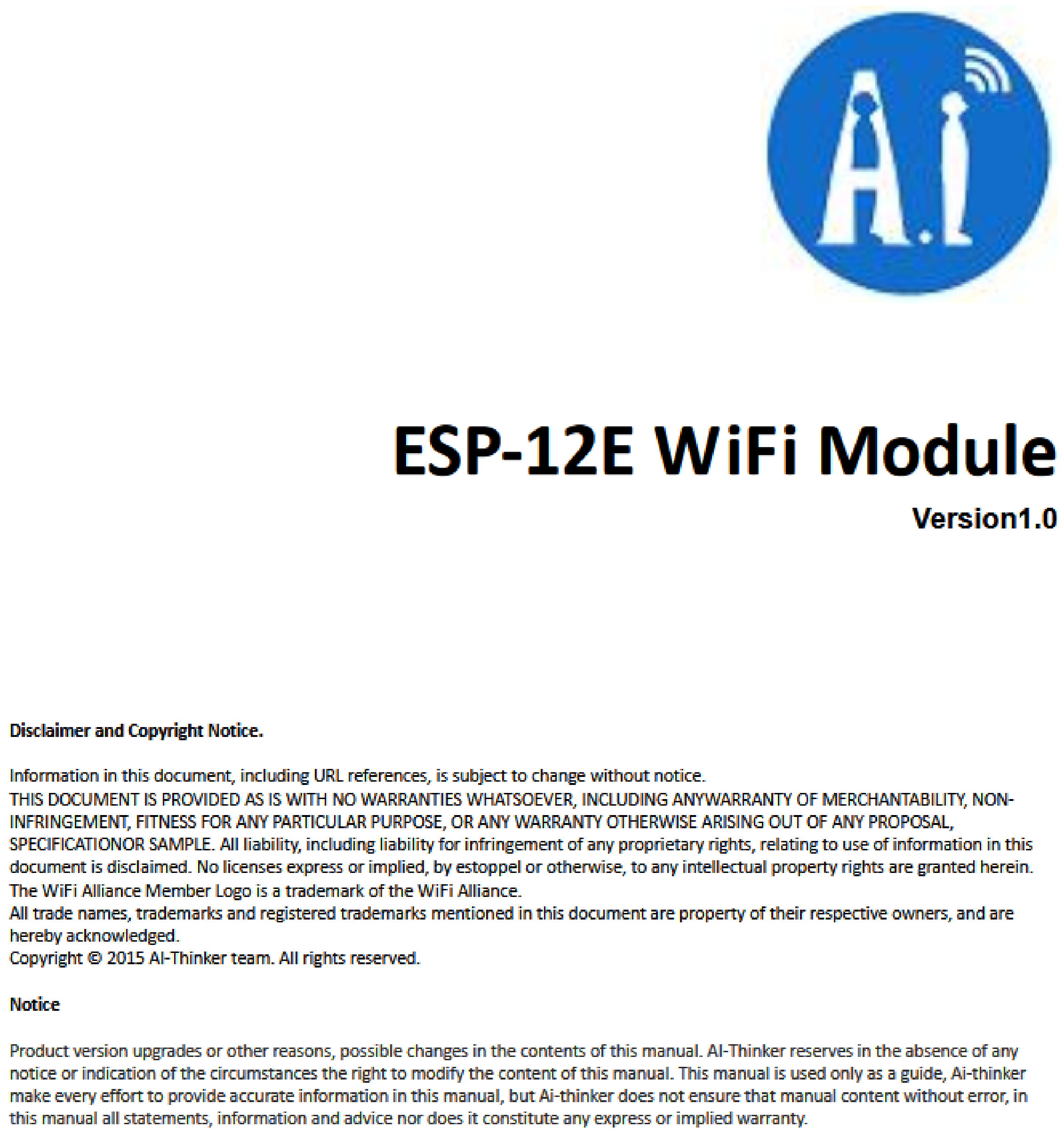
```
# the connection fails, the other secondary addresses will be
attempted in
# turn. Whilst connected to a secondary bridge, the bridge will
periodically
# attempt to reconnect to the main bridge until successful.
# If round_robin is true, then all addresses are treated as equals.
If a
# connection fails, the next address will be tried and if successful
will
# remain connected until it fails
#round_robin false

# Set the client id to use on the remote end of this bridge
connection. If not
# defined, this defaults to 'name.hostname' where name is the
connection name
# and hostname is the hostname of this computer.
# This replaces the old "clientid" option to avoid confusion.
"clientid"
# remains valid for the time being.
#remote_clientid

# Set the clientid to use on the local broker. If not defined, this
defaults to
# 'local.<clientid>'. If you are bridging a broker to itself, it is
important
# that local_clientid and clientid do not match.
#local_clientid

# Set the clean session variable for this bridge.
# When set to true, when the bridge disconnects for any reason, all
# messages and subscriptions will be cleaned up on the remote
# broker. Note that with cleansession set to true, there may be a
# significant amount of retained messages sent when the bridge
# reconnects after losing its connection.
# When set to false, the subscriptions and messages are kept on the
# remote broker, and delivered when the bridge reconnects.
#cleansession false

# If set to true, publish notification messages to the local and
remote brokers
# giving information about the state of the bridge connection.
Retained
# messages are published to the topic
$SYS/broker/connection/<clientid>/state
# unless the notification_topic option is used.
# If the message is 1 then the connection is active, or 0 if the
connection has
# failed.
#notifications true

# Choose the topic on which notification messages for this bridge
are
# published. If not set, messages are published on the topic
# $SYS/broker/connection/<clientid>/state
#notification_topic

# Set the keepalive interval for this bridge connection, in
# seconds.
#keepalive_interval 60
```

```
# Set the start type of the bridge. This controls how the bridge
starts and
# can be one of three types: automatic, lazy and once. Note that
RSMB provides
# a fourth start type "manual" which isn't currently supported by
mosquitto.
#
# "automatic" is the default start type and means that the bridge
connection
# will be started automatically when the broker starts and also
restarted
# after a short delay (30 seconds) if the connection fails.
#
# Bridges using the "lazy" start type will be started automatically
when the
# number of queued messages exceeds the number set with the
"threshold"
# parameter. It will be stopped automatically after the time set by
the
# "idle_timeout" parameter. Use this start type if you wish the
connection to
# only be active when it is needed.
#
# A bridge using the "once" start type will be started automatically
when the
# broker starts but will not be restarted if the connection fails.
#start_type automatic

# Set the amount of time a bridge using the automatic start type
will wait
# until attempting to reconnect.  Defaults to 30 seconds.
#restart_timeout 30

# Set the amount of time a bridge using the lazy start type must be
idle before
# it will be stopped. Defaults to 60 seconds.
#idle_timeout 60

# Set the number of messages that need to be queued for a bridge
with lazy
# start type to be restarted. Defaults to 10 messages.
# Must be less than max_queued_messages.
#threshold 10

# If try_private is set to true, the bridge will attempt to indicate
to the
# remote broker that it is a bridge not an ordinary client. If
successful, this
# means that loop detection will be more effective and that retained
messages
# will be propagated correctly. Not all brokers support this feature
so it may
# be necessary to set try_private to false if your bridge does not
connect
# properly.
#try_private true

# Set the username to use when connecting to a broker that requires
# authentication.
# This replaces the old "username" option to avoid confusion.
"username"
```

```
# remains valid for the time being.
#remote_username

# Set the password to use when connecting to a broker that requires
# authentication. This option is only used if remote_username is
also set.
# This replaces the old "password" option to avoid confusion.
"password"
# remains valid for the time being.
#remote_password

# -----------------------------------------------------------------
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# Either bridge_cafile or bridge_capath must be defined to enable
TLS support
# for this bridge.
# bridge_cafile defines the path to a file containing the
# Certificate Authority certificates that have signed the remote
broker
# certificate.
# bridge_capath defines a directory that will be searched for files
containing
# the CA certificates. For bridge_capath to work correctly, the
certificate
# files must have ".crt" as the file ending and you must run
"c_rehash <path to
# capath>" each time you add/remove a certificate.
#bridge_cafile
#bridge_capath

# Path to the PEM encoded client certificate, if required by the
remote broker.
#bridge_certfile

# Path to the PEM encoded client private key, if required by the
remote broker.
#bridge_keyfile

# When using certificate based encryption, bridge_insecure disables
# verification of the server hostname in the server certificate.
This can be
# useful when testing initial server configurations, but makes it
possible for
# a malicious third party to impersonate your server through DNS
spoofing, for
# example. Use this option in testing only. If you need to resort to
using this
# option in a production environment, your setup is at fault and
there is no
# point using encryption.
#bridge_insecure false

# -----------------------------------------------------------------
# PSK based SSL/TLS support
# -----------------------------------------------------------------
# Pre-shared-key encryption provides an alternative to certificate
based
# encryption. A bridge can be configured to use PSK with the
bridge_identity
```

```
# and bridge_psk options. These are the client PSK identity, and
pre-shared-key
# in hexadecimal format with no "0x". Only one of certificate and
PSK based
# encryption can be used on one
# bridge at once.
#bridge_identity
#bridge_psk


# =================================================================
# External config files
# =================================================================

# External configuration files may be included by using the
# include_dir option. This defines a directory that will be searched
# for config files. All files that end in '.conf' will be loaded as
# a configuration file. It is best to have this as the last option
# in the main file. This option will only be processed from the main
# configuration file. The directory specified must not contain the
# main configuration file.
#include_dir

# =================================================================
# rsmb options - unlikely to ever be supported
# =================================================================

#ffdc_output
#max_log_entries
#trace_level
#trace_output
```

Annex C – ESP8266-12E Datasheet



ESP-12E WiFi Module

Version1.0

1. Preambles

ESP-12E WiFi module is developed by Ai-thinker Team. core processor ESP8266 in smaller sizes of the module encapsulates Tensilica L106 integrates industry-leading ultra low power 32-bit MCU micro, with the 16-bit short mode, Clock speed support 80 MHz, 160 MHz, supports the RTOS, integrated Wi-Fi MAC/BB/RF/PA/LNA, on-board antenna.

The module supports standard IEEE802.11 b/g/n agreement, complete TCP/IP protocol stack. Users can use the add modules to an existing device networking, or building a separate network controller.

ESP8266 is high integration wireless SOCs, designed for space and power constrained mobile platform designers. It provides unsurpassed ability to embed Wi-Fi capabilities within other systems, or to function as a standalone application, with the lowest cost, and minimal space requirement.

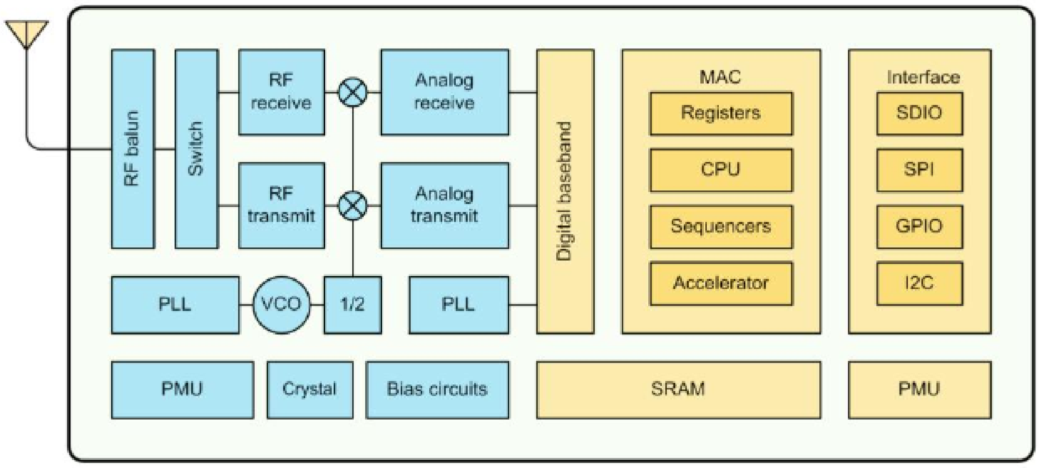


Figure 1 ESP8266EX Block Diagram

ESP8266EX offers a complete and self-contained Wi-Fi networking solution; it can be used to host the application or to offload Wi-Fi networking functions from another application processor.

When ESP8266EX hosts the application, it boots up directly from an external flash. In has integrated cache to improve the performance of the system in such applications.

Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any micro controllerbased design with simple connectivity (SPI/SDIO or I2C/UART interface).

ESP8266EX is among the most integrated WiFi chip in the industry; it integrates the antenna switches, RF balun, power amplifier, low noise receive amplifier, filters, power management modules, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area.

ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor, with on-chip SRAM, besides the Wi-Fi functionalities. ESP8266EX is often integrated with external sensors and other application specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

Espressif Systems' Smart Connectivity Platform (ESCP) demonstrates sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing. for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

1.1. Features

- 802.11 b/g/n
- Integrated low power 32-bit MCU
- Integrated 10-bit ADC
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- Integrated PLL, regulators, and power management units
- Supports antenna diversity
- Wi-Fi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- Support Smart Link Function for both Android and iOS devices
- Support Smart Link Function for both Android and iOS devices
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IRDA, PWM, GPIO
- STBC, 1x1 MIMO, 2x1 MIMO
- A-MPDU & A-MSDU aggregation and 0.4s guard interval

- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C

1.2. Parameters

Table 1 below describes the major parameters.

Table 1 Parameters

Categories	Items	Values
WiFi Paramters	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4GHz-2.5GHz (2400M-2483.5M)
Hardware Paramaters	Peripheral Bus	UART/HSPI/I2C/I2S/Ir Remote Contorl
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	16mm*24mm*3mm
	External Interface	N/A
Software Parameters	Wi-Fi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network) / download and write firmware via host
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

2. Pin Descriptions

There are altogether 22 pin counts, the definitions of which are described in Table 2 below。

Table 2 ESP-12E Pin design

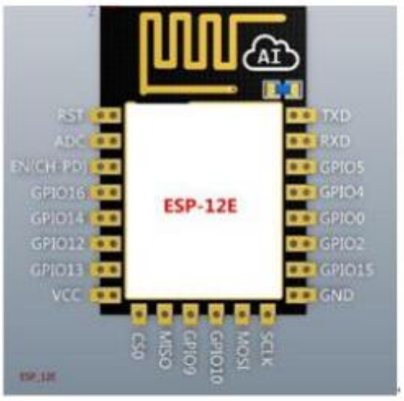


Table 3 Pin Descriptions

NO.	Pin Name	Function
1	RST	Reset the module
2	ADC	A/D Conversion result.Input voltage range 0-1v,scope:0-1024
3	EN	Chip enable pin.Active high
4	IO16	GPIO16; can be used to wake up the chipset from deep sleep mode.
5	IO14	GPIO14; HSPI_CLK
6	IO12	GPIO12; HSPI_MISO
7	IO13	GPIO13; HSPI_MOSI; UART0_CTS
8	VCC	3.3V power supply (VDD)
9	CS0	Chip selection
10	MISO	Salve output Main input

11	IO9	GPIO9
12	IO10	GBIO10
13	MOSI	Main output slave input
14	SCLK	Clock
15	GND	GND
16	IO15	GPIO15; MTDO; HSPICS; UART0_RTS
17	IO2	GPIO2; UART1_TXD
18	IO0	GPIO0
19	IO4	GPIO4
20	IO5	GPIO5
21	RXD	UART0_RXD; GPIO3
22	TXD	UART0_TXD; GPIO1

Table 4 Pin Mode

Mode	GPIO15	GPIO0	GPIO2
UART	Low	Low	High
Flash Boot	Low	High	High

Table 5 Receiver Sensitivity

Parameters	Min	Typical	Max	Unit
Input frequency	2412		2484	MHz
Input impedance		50		Ω
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
DSSS, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

3. Packaging and Dimension

The external size of the module is 16mm*24mm*3mm, as is illustrated in Figure 3 below. The type of flash integrated in this module is an SPI flash, the capacity of which is 4 MB, and the package size of which is SOP-210mil. The antenna applied on this module is a 3DBi PCB-on-board antenna.

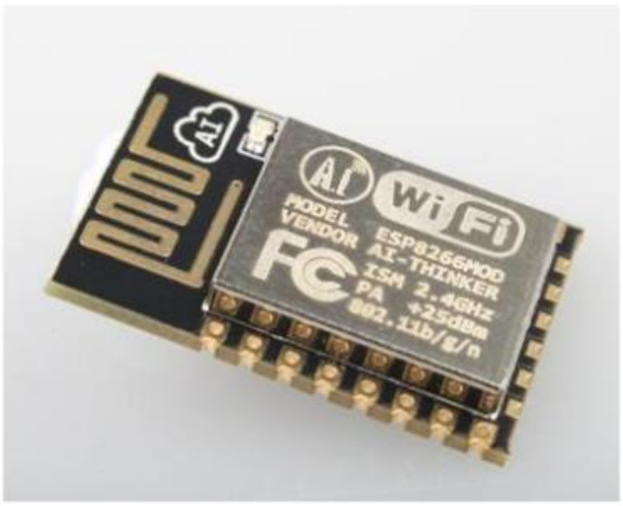


Figure 3 [Module Pin Counts, 22 pin, 16 mm *24 mm *3 mm]

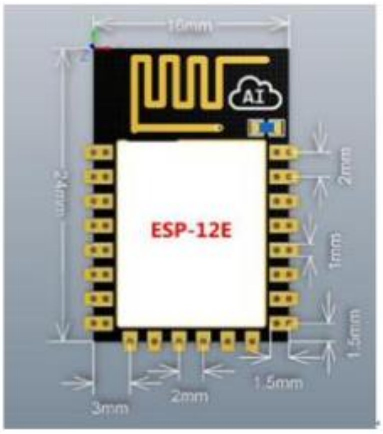


Figure 4 Top View of ESP-12E WiFi Module

Table 5 Dimension of ESP-12E WiFi Modul

Length	Width	Height	PAD Size(Bottom)	Pin Pitch
16 mm	24mm	3 mm	0.9 mm x 1.7 mm	2mm

4. Functional Descriptions

4.1. MCU

ESP8266EX is embedded with Tensilica L106 32-bit micro controller (MCU), which features extra low power consumption and 16-bit RSIC. The CPU clock speed is 80MHz. It can also reach a maximum value of 160MHz. ESP8266EX is often integrated with external sensors and other specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

4.2. Memory Organization

4.2.1. Internal SRAM and ROM

ESP8266EX WiFi SoC is embedded with memory controller, including SRAM and ROM. MCU can visit the memory units through iBus, dBus, and AHB interfaces. All memory units can be visited upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK provided, SRAM space that is available to users is assigned as below:

▪RAM size < 36kB, that is to say, when ESP8266EX is working under the station mode and is connected to the router, programmable space accessible to user in heap and data section is around 36kB.)

▪ There is no programmable ROM in the SoC, therefore, user program must be stored in an external SPI flash.

4.2.2. External SPI Flash

This module is mounted with an 4 MB external SPI flash to store user programs. If larger definable storage space is required, a SPI flash with larger memory size is preferred. Theoretically speaking, up to 16 MB memory capacity can be supported.

Suggested SPI Flash memory capacity:

▪OTA is disabled: the minimum flash memory that can be supported is 512 kB;

▪OTA is enabled: the minimum flash memory that can be supported is 1 MB.

Several SPI modes can be supported, including Standard SPI, Dual SPI, and Quad SPI.

Therefore, please choose the correct SPI mode when you are downloading into the flash, otherwise firmwares/programs that you downloaded may not work in the right way.

4.3. Crystal

Currently, the frequency of crystal oscillators supported include 40MHz, 26MHz and 24MHz. The accuracy of crystal oscillators applied should be ±10PPM, and the operating temperature range should be between -20°C and 85°C.

When using the downloading tools, please remember to select the right crystal oscillator type. In circuit design, capacitors C1 and C2, which are connected to the earth, are added to the input and output terminals of the crystal oscillator respectively. The values of the two capacitors can be flexible, ranging from 6pF to 22pF, however, the specific capacitive values of C1 and C2 depend on further testing and adjustment on the overall performance of the whole circuit. Normally, the capacitive values of C1 and C2 are within 10pF if the crystal oscillator frequency is 26MHz, while the values of C1 and C2 are 10pF<C1, C2<22pF if the crystal oscillator frequency is 40MHz.

4.4. Interfaces

Table 6 Descriptions of Interfaces

Interface	Pin Name	Description

HSPI	IO12(MISO)
	IO13(MOSI)
	IO14(CLK)
	IO15(CS)
		SPI Flash 2, display screen, and MCU can be connected using HSPI interface.

PWM	IO12(R)
	IO15(G)
	IO13(B)
		Currently the PWM interface has four channels, but users can extend the channels according to their own needs. PWM interface can be used to control LED lights, buzzers, relays, electronic machines, and so on.

IR Remote Control	IO14(IR_T)
	IO5(IR_R)
		The functionality of Infrared remote control interface can be implemented via software programming. NEC coding, modulation, and demodulation are used by this interface. The frequency of modulated carrier signal is 38KHz.

ADC	TOUT	ESP8266EX integrates a 10-bit analog ADC. It can be used to test the power-supply voltage of VDD3P3 (Pin3 and Pin4) and the input power voltage of TOUT (Pin 6). However, these two functions cannot be used simultaneously. This interface is typically used in sensor products.

I2C	IO14(SCL)
	IO2(SDA)
		I2C interface can be used to connect external sensor products and display screens, etc.

Interface	Pin Name	Description
UART	**UART0:** TXD (U0TXD) RXD (U0RXD) IO15 (RTS) IO13 (CTS) **UART1:** IO2(TXD)	Devices with UART interfaces can be connected with the module. Downloading: U0TXD+U0RXD or GPIO2+U0RXD Communicating: UART0: U0TXD, U0RXD, MTDO (U0RTS), MTCK (U0CTS) Debugging: UART1_TXD (GPIO2) can be used to print debugging information.
		By default, UART0 will output some printed information when the device is powered on and is booting up. If this issue exerts influence on some specific applications, users can exchange the inner pins of UART when initializing, that is to say, exchange U0TXD, U0RXD with U0RTS, U0CTS.

I2S	**I2S Input:**
	IO12 (I2SI_DATA) ;
	IO13 (I2SI_BCK);
	IO14 (I2SI_WS);
	I2S Output:
	IO15 (I2SO_BCK);
	IO3 (I2SO_DATA);
	IO2 (I2SO_WS).
		I2S interface is mainly used for collecting, processing, and transmission of audio data.

4.5. Absolute Maximum Ratings

Table 7 Absolute Maximum Ratings

Rating	Condition	Value	Unit
Storage Temperature		-40 to 125	°C
Maximum Soldering Temperature		260	°C
Supply Voltage	IPC/JEDEC J-STD-020	+3.0 to +3.6	V

4.6. Recommended Operating Conditions

Table 8 Recommended Operating Conditions

Operating Condition	Symbol	Min	Typ	Max	Unit
Operating Temperature		-40	20	125	°C
Supply voltage	VDD	3.0	3.3	3.6	V

4.7. Digital Terminal Characteristics

Table 9 Digital Terminal Characteristics

Terminals	Symbol	Min	Typ	Max	Unit
Input logic level low	V_{IL}	-0.3		0.25VDD	V
Input logic level high	V_{IH}	0.75VDD		VDD+0.3	V
Output logic level low	V_{OL}	N		0.1VDD	V
Output logic level high	V_{OH}	0.8VDD		N	V

Note: Test conditions: VDD = 3.3V, Temperature = 20 ℃, if nothing special is stated.

5. RF Performance

Description	Min.	Typ.	Max	Unit
Input frequency	2400		2483.5	MHz
Input impedance		50		ohm
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
CCK, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

Table 10 RF Performance

6. Power Consumption

Parameters	Min	Typical	Max	Unit
Tx802.11b, CCK 11Mbps, P OUT=+17dBm		170		mA
Tx 802.11g, OFDM 54Mbps, P OUT =+15dBm		140		mA
Tx 802.11n, MCS7, P OUT =+13dBm		120		mA
Rx 802.11b, 1024 bytes packet length , -80dBm		50		mA
Rx 802.11g, 1024 bytes packet length, -70dBm		56		mA
Rx 802.11n, 1024 bytes packet length, -65dBm		56		mA
Modem-Sleep①		15		mA
Light-Sleep②		0.9		mA
Deep-Sleep③		10		uA

Table 11 Power Consumption

❶Modem-Sleep requires the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it saves power to shut down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission. E.g. in DTIM3, to maintain a sleep 300mswake 3ms cycle to receive AP's Beacon packages, the current is about 15mA.

❷ During Light-Sleep, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power according to the 802.11 standard (U-APSD). E.g. in DTIM3, to maintain a sleep 300ms-wake 3ms cycle to receive AP's Beacon packages, the current is about 0.9mA.

❸ Deep-Sleep does not require Wi-Fi connection to be maintained. For application with long time lags between data transmission, e.g. a temperature sensor that checks the temperature every 100s ,sleep 300s and waking up to connect to the AP (taking about 0.3~1s), the overall average current is less than 1mA.

7. Reflow Profile

Table 12 Instructions

T_S max to T_L (Ramp-up Rate)	3°C/second max
Preheat Temperature Min.(T_S Min.) Temperature Typical.(T_S Typ.) Temperature Min.(T_S Max.) Time(T_S)	 150°C 175°C 200°C 60~180 seconds
Ramp-up rate (T_L to T_P)	3°C/second max
Time Maintained Above: --Temperature(T_L)/Time(T_L)	 217°C/60~150 seconds
Peak Temperature(T_P)	260°C max. for 10 seconds
Target Peak Temperature (T_P Target)	260°C +0/-5°C
Time within 5°C of actual peak(t_P)	20~40 seconds
T_S max to T_L (Ramp-down Rate)	6°C/second max
Tune 25°C to Peak Temperature (t)	8 minutes max

8. Schematics

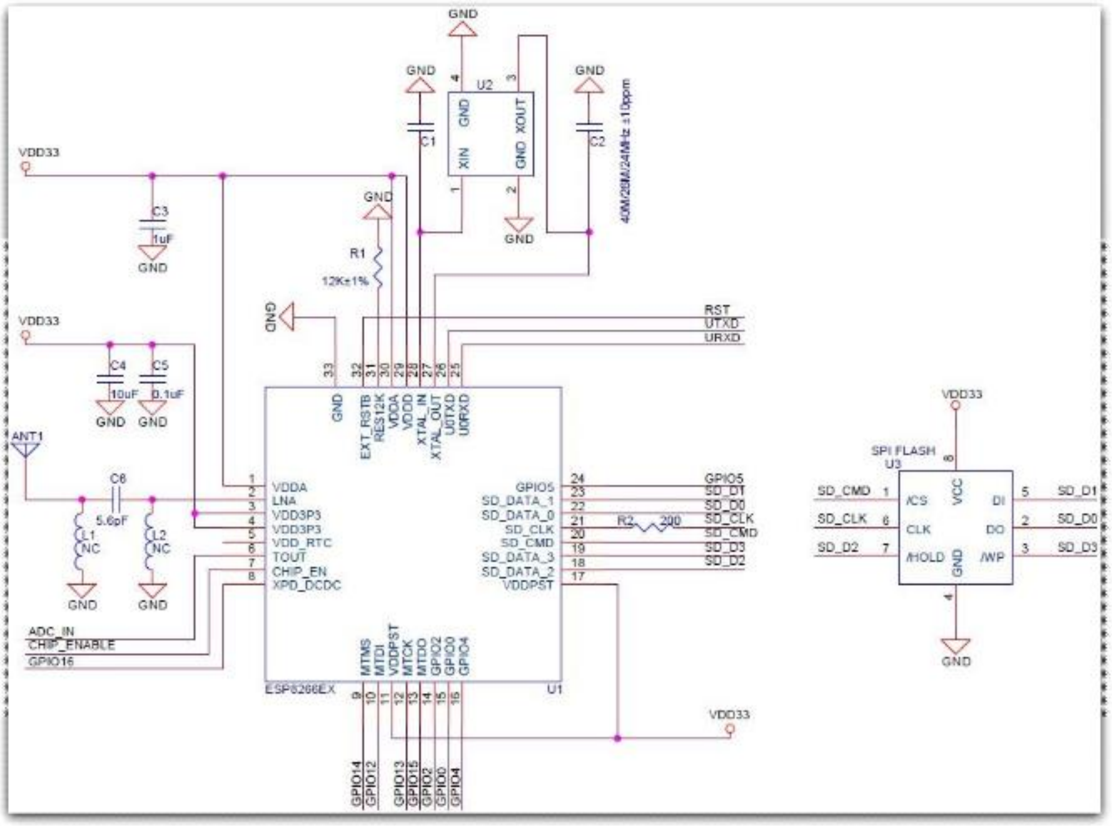
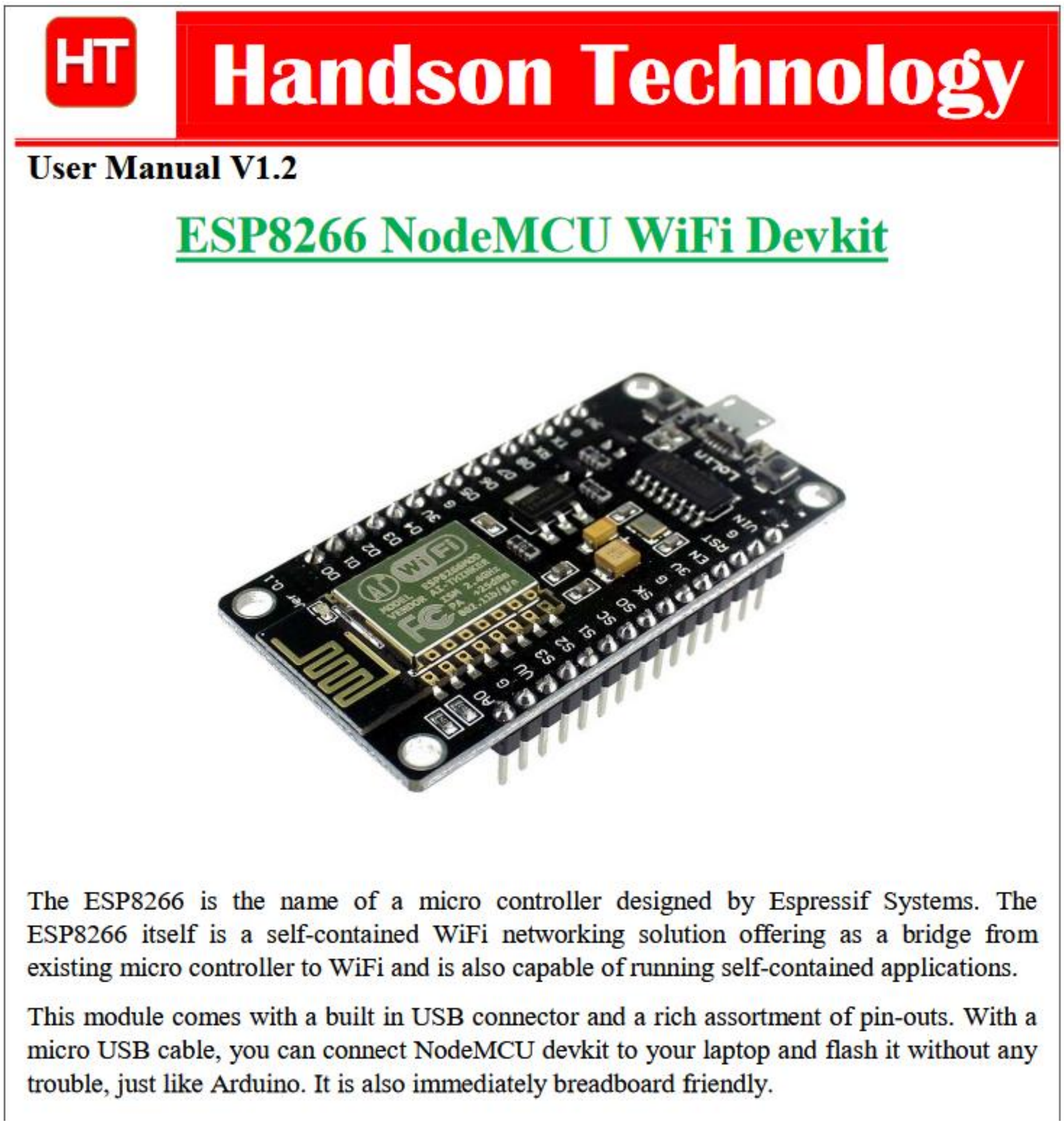


Figure 4 Schematics of Esp-12E WiFi Module

Annex D – NodeMcu Devkit v1.0 ESP8266 Wi-Fi Module ESP-12E User Manual



Handson Technology

User Manual V1.2

ESP8266 NodeMCU WiFi Devkit

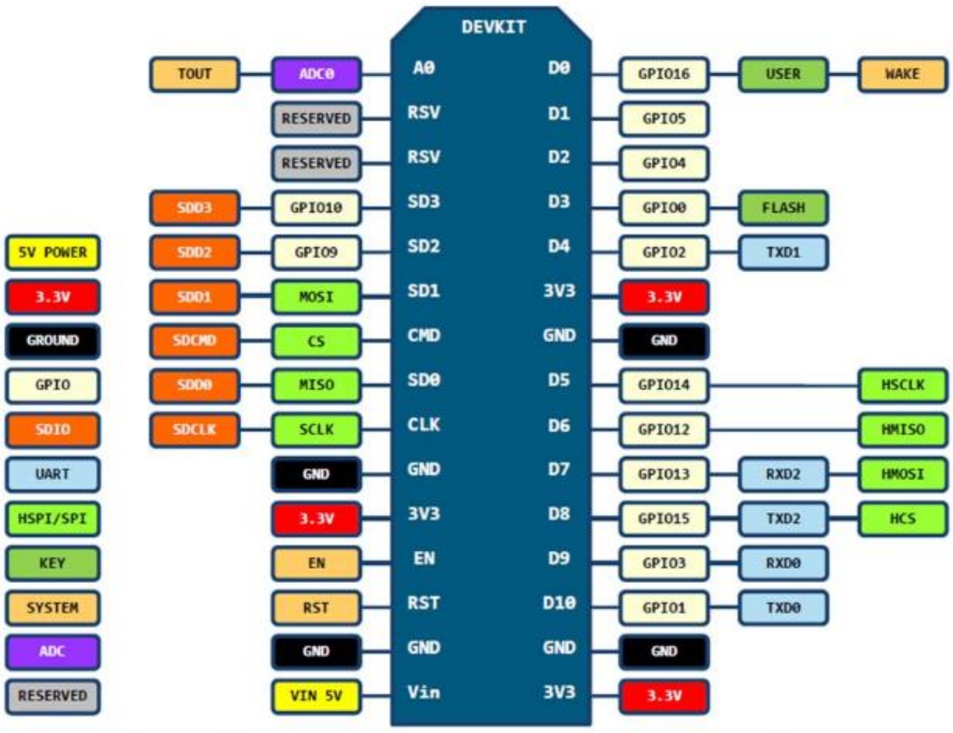
The ESP8266 is the name of a micro controller designed by Espressif Systems. The ESP8266 itself is a self-contained WiFi networking solution offering as a bridge from existing micro controller to WiFi and is also capable of running self-contained applications.

This module comes with a built in USB connector and a rich assortment of pin-outs. With a micro USB cable, you can connect NodeMCU devkit to your laptop and flash it without any trouble, just like Arduino. It is also immediately breadboard friendly.

1. Specification:

- Voltage:3.3V.
- Wi-Fi Direct (P2P), soft-AP.
- Current consumption: 10uA~170mA.
- Flash memory attachable: 16MB max (512K normal).
- Integrated TCP/IP protocol stack.
- Processor: Tensilica L106 32-bit.
- Processor speed: 80~160MHz.
- RAM: 32K + 80K.
- GPIOs: 17 (multiplexed with other functions).
- Analog to Digital: 1 input with 1024 step resolution.
- +19.5dBm output power in 802.11b mode
- 802.11 support: b/g/n.
- Maximum concurrent TCP connections: 5.

2. Pin Definition:



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

3. Using Arduino IDE

The most basic way to use the ESP8266 module is to use serial commands, as the chip is basically a WiFi/Serial transceiver. However, this is not convenient. What we recommend is using the very cool Arduino ESP8266 project, which is a modified version of the Arduino IDE that you need to install on your computer. This makes it very convenient to use the ESP8266 chip as we will be using the well-known Arduino IDE. Following the below step to install ESP8266 library to work in Arduino IDE environment.

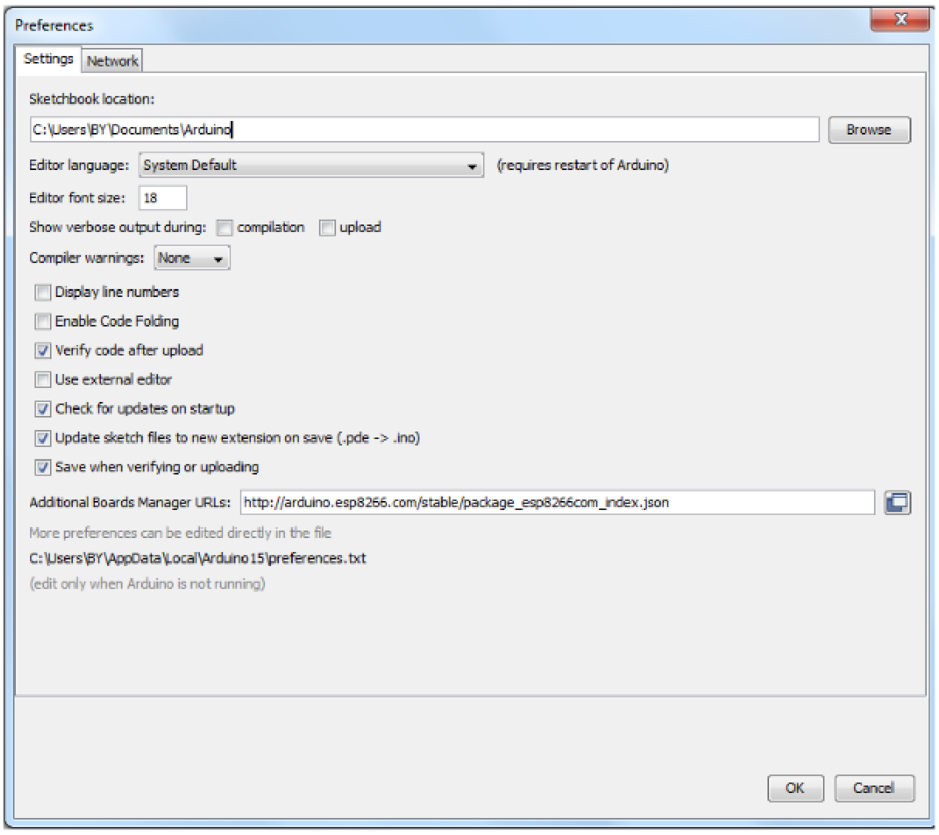
3.1 Install the Arduino IDE 1.6.4 or greater

Download Arduino IDE from Arduino.cc (1.6.4 or greater) - don't use 1.6.2 or lower version! You can use your existing IDE if you have already installed it.

You can also try downloading the ready-to-go package from the ESP8266-Arduino project, if the proxy is giving you problems.

3.2 Install the ESP8266 Board Package

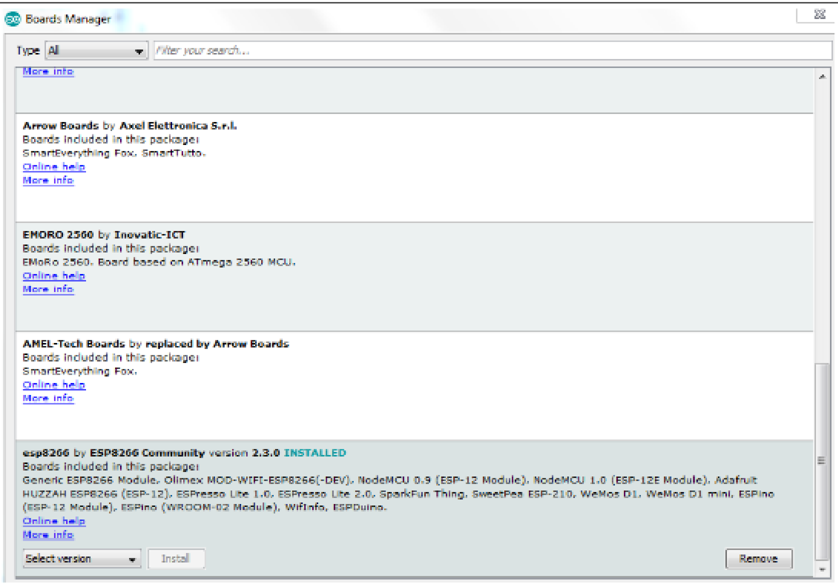
Enter ***http://arduino.esp8266.com/stable/package_esp8266com_index.json*** into *Additional Board Manager URLs* field in the Arduino v1.6.4+ preferences.



Sketchbook location: C:\Users\BY\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 18
Show verbose output during: ☐ compilation ☐ upload
Compiler warnings: None
☐ Display line numbers
☐ Enable Code Folding
☑ Verify code after upload
☐ Use external editor
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading
Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json
More preferences can be edited directly in the file
C:\Users\BY\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Click 'File' -> 'Preferences' to access this panel.

Next, use the Board manager to install the ESP8266 package.



Boards Manager

Type All | Filter your search...

More info

Arrow Boards by **Axel Elettronica S.r.l.**
Boards included in this package:
SmartEverything Fox, SmartTutto.
Online help
More info

EMORO 2560 by **Inovatic-ICT**
Boards included in this package:
EMoRo 2560, Board based on ATmega 2560 MCU.
Online help
More info

AMEL-Tech Boards by **replaced by Arrow Boards**
Boards included in this package:
SmartEverything Fox.
Online help
More info

esp8266 by **ESP8266 Community** version **2.3.0** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Olimex MOD-WIFI-ESP8266(-DEV), NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Adafruit HUZZAH ESP8266 (ESP-12), ESPresso Lite 1.0, ESPresso Lite 2.0, SparkFun Thing, SweetPea ESP-210, WeMos D1, WeMos D1 mini, ESPino (ESP-12 Module), ESPino (WROOM-02 Module), WifInfo, ESPDuino.
Online help
More info

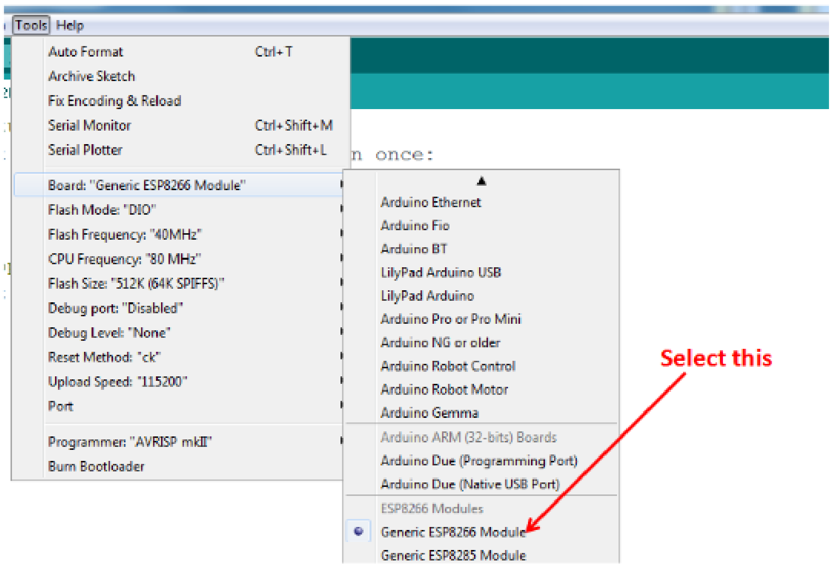
Select version | Install | Remove

Click 'Tools' -> 'Board:' -> 'Board Manager...' to access this panel.

Scroll down to ' esp8266 by ESP8266 Community ' and click "Install" button to install the ESP8266 library package. Once installation completed, close and re-open Arduino IDE for ESP8266 library to take effect.

3.3 Setup ESP8266 Support

When you've restarted Arduino IDE, select 'Generic ESP8266 Module' from the 'Tools' -> 'Board:' dropdown menu.

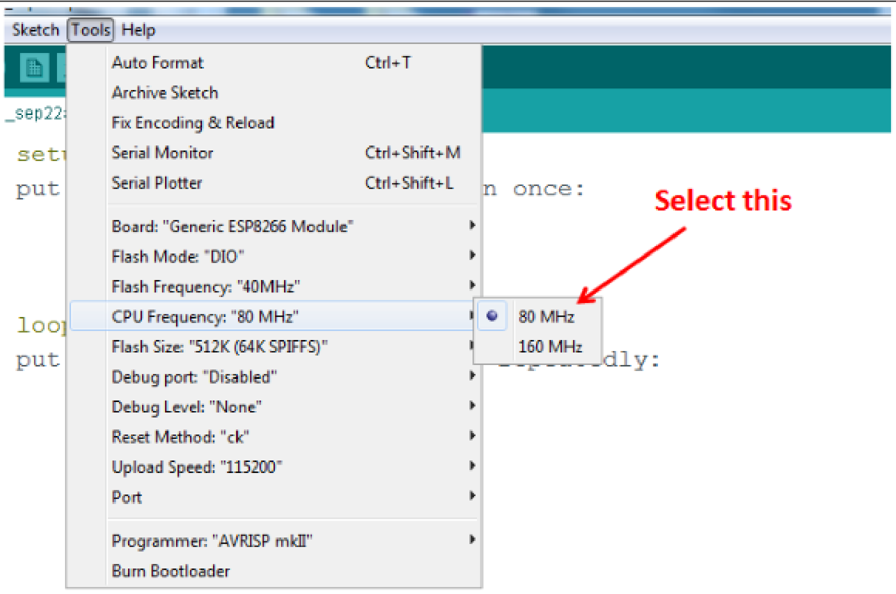


Tools Help

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port
Programmer: "AVRISP mkII"
Burn Bootloader

Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control
Arduino Robot Motor
Arduino Gemma
Arduino ARM (32-bits) Boards
Arduino Due (Programming Port)
Arduino Due (Native USB Port)
ESP8266 Modules
● Generic ESP8266 Module
Generic ESP8285 Module

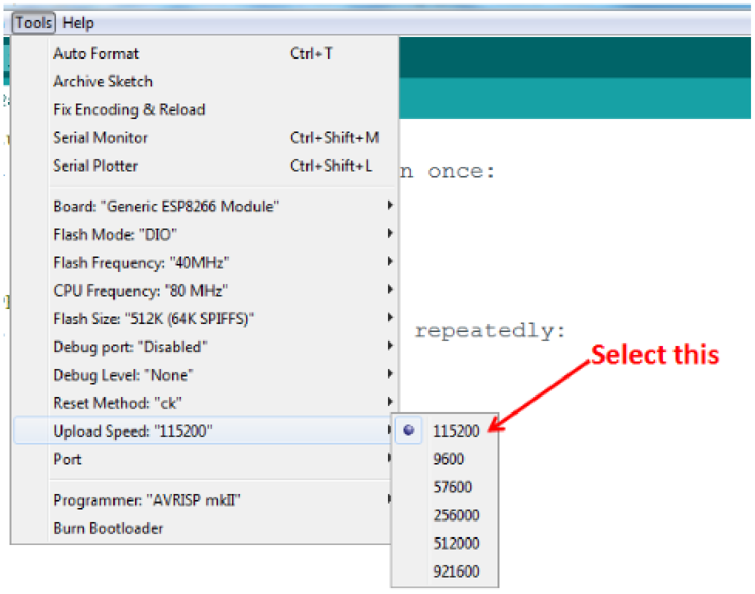
Select 80 MHz as the CPU frequency (you can try 160 MHz overclock later)


Sketch Tools Help

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port
Programmer: "AVRISP mkII"
Burn Bootloader

● 80 MHz
160 MHz

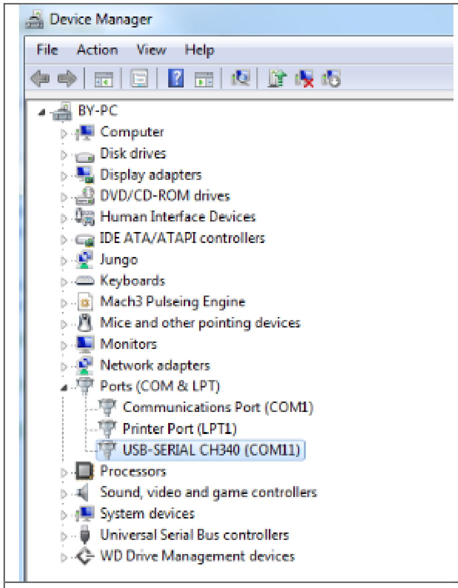
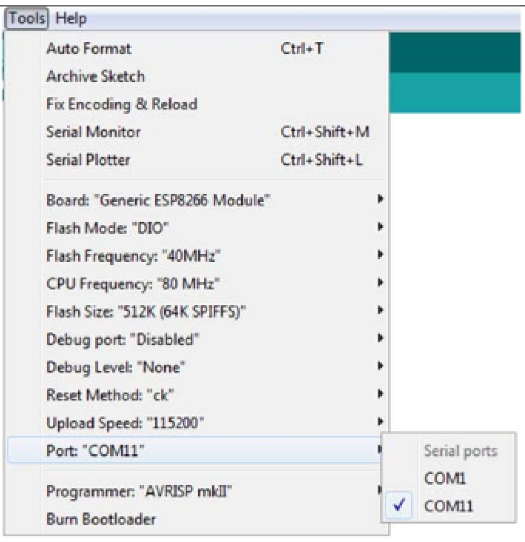
Select '115200' baud upload speed is a good place to start - later on you can try higher speeds but 115200 is a good safe place to start.


Tools Help

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port
Programmer: "AVRISP mkII"
Burn Bootloader

● 115200
9600
57600
256000
512000
921600

Go to your Windows 'Device Manager' to find out which Com Port 'USB-Serial CH340' is assigned to. Select the matching COM/serial port for your CH340 USB-Serial interface.

	
Find out which Com Port is assign for CH340	Select the correct Com Port as indicated on ‘Device Manager”

Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port: "COM11"
Programmer: "AVRISP mkII"

Note: if this is your first time using CH340 " USB-to-Serial " interface, please install the driver first before proceed the above Com Port setting. The CH340 driver can be download from the below site:

https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers

3.4 Blink Test

We'll begin with the simple blink test.

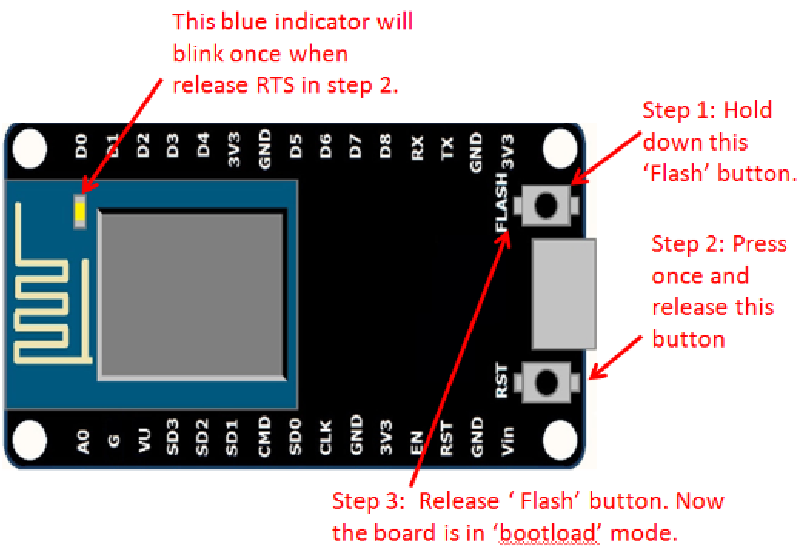
Enter this into the sketch window (and save since you'll have to). Connect a LED as shown in Figure3-1.

```
void setup() {
  pinMode(5, OUTPUT);   // GPIO05, Digital Pin D1
}

void loop() {
  digitalWrite(5, HIGH);
  delay(900);
  digitalWrite(5, LOW);
  delay(500);
}
```

Now you'll need to put the board into bootload mode. You'll have to do this before each upload. There is no timeout for bootload mode, so you don't have to rush!

- Hold down the ‘Flash’ button.
- While holding down ‘ Flash’, press the ‘RST’ button.
- Release ‘RST’, then release ‘Flash’

- When you release the 'RST' button, the blue indication will blink once, this means its ready to bootload.


This blue indicator will blink once when release RTS in step 2.
Step 1: Hold down this 'Flash' button.
Step 2: Press once and release this button
Step 3: Release ' Flash' button. Now the board is in 'bootload' mode.
D0
D1
D2
D3
D4
3V3
GND
D5
D6
D7
D8
RX
TX
GND
3V3
FLASH
RST
A0
G
VU
SD3
SD2
SD1
CMD
SD0
CLK
GND
3V3
EN
RST
GND
Vin

Once the ESP board is in bootload mode, upload the sketch via the IDE, Figure 3-2.

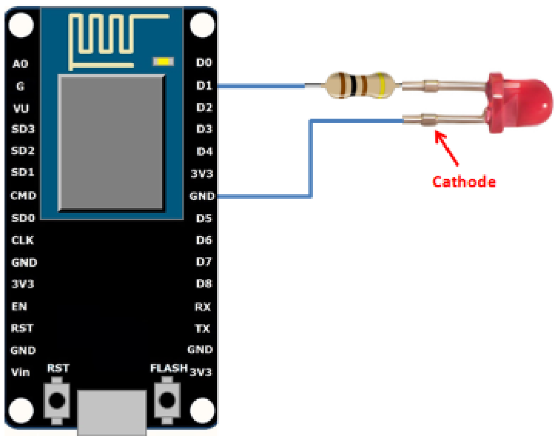


Figure3-1: Connection diagram for the blinking test

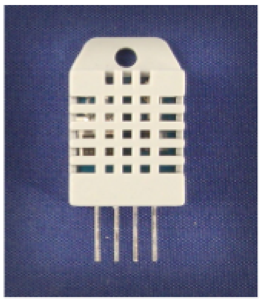
Annex E – DHT22 Datasheet

Aosong(Guangzhou) Electronics Co.,Ltd

Tell: +86-020-36380552, +86-020-36042809 Fax: +86-020-36380562
http://www.aosong.com
Email: thomasliu198518@yahoo.com.cn sales@aosong.com
Address: No.56, Renhe Road, Renhe Town, Baiyun District, Guangzhou, China

Digital-output relative humidity & temperature sensor/module

AM2303


Capacitive-type humidity and temperature module/sensor

1. Feature & Application:

* Full range temperature compensated * Relative humidity and temperature measurement
* Calibrated digital signal *Outstanding long-term stability *Extra components not needed
* Long transmission distance * Low power consumption *4 pins packaged and fully interchangeable

2. Description:

AM2303 output calibrated digital signal. It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip computer.

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.
Small size & low consumption & long transmission distance(20m) enable AM2303 to be suited in all kinds of harsh application occasions.

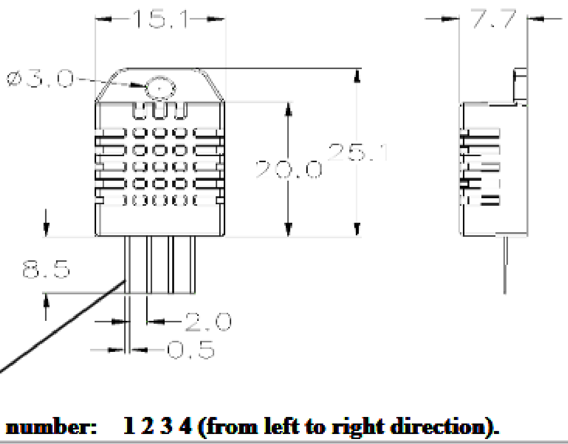
Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

Model	AM2303
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer humidity capacitor & DS18B20 for detecting temperature
Measuring range	humidity 0-100%RH; temperature -40~125Celsius

Accuracy	humidity +-2%RH(Max +-5%RH);	temperature +-0.2Celsius
Resolution or sensitivity	humidity 0.1%RH;	temperature 0.1Celsius
Repeatability	humidity +-1%RH;	temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH	
Long-term Stability	+-0.5%RH/year	
Sensing period	Average: 2s	
Interchangeability	fully interchangeable	

4. Dimensions: (unit----mm)



Pin sequence number: 1 2 3 4 (from left to right direction).

Pin	Function
1	VDD—power supply
2	DATA--signal
3	NULL
4	GND

5. Operating specifications:

(1) Power and Pins

Power's voltage should be 3.3-6V DC. When power is supplied to sensor, don't send any instruction to the sensor within one second to pass unstable status. One capacitor valued 100nF can be added between VDD and GND for wave filtering.

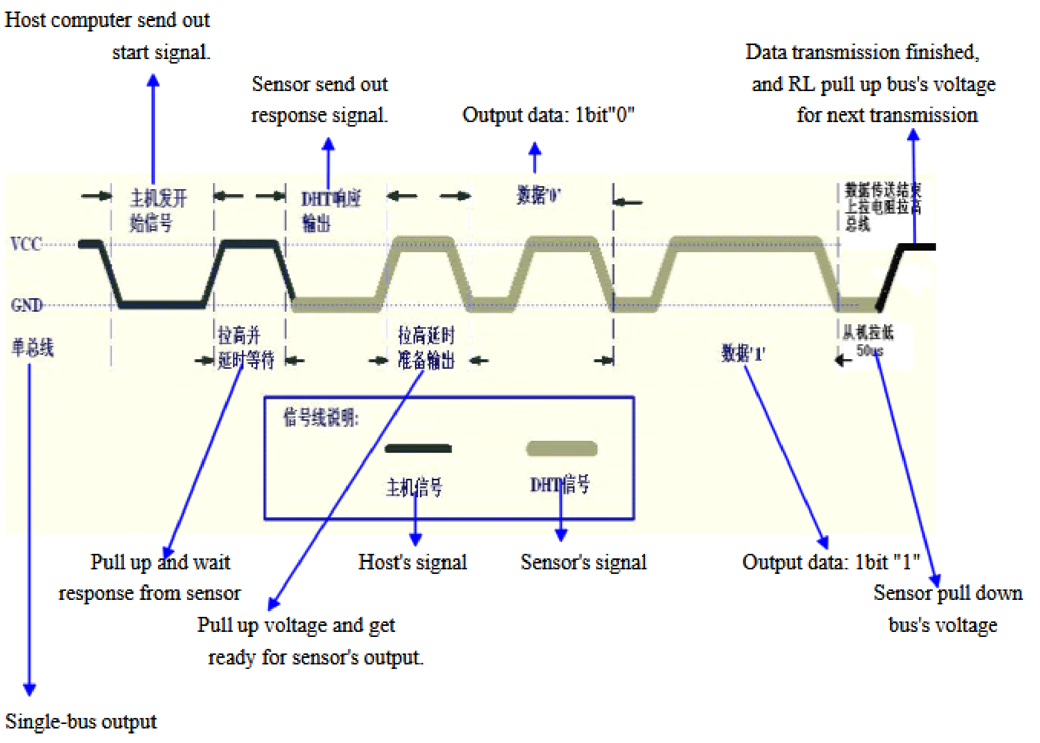
(2) Communication and signal

Single-bus data is used for communication between MCU and AM2303, it costs 5mS for single time communication.

Data is comprised of integral and decimal part, the following is the formula for data.
AM2303 send out higher data bit firstly!
DATA=8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data+8 bit check-sum
If the data transmission is right, check-sum should be the last 8 bit of "8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data".

When MCU send start signal, AM2303 change from low-power-consumption-mode to running-mode. When MCU finishs sending the start signal, AM2303 will send response signal of 40-bit data that reflect the relative humidity and temperature information to MCU. Without start signal from MCU, AM2303 will not give response signal to MCU. One start signal for one time's response data that reflect the relative humidity and temperature information from AM2303. AM2303 will change to low-power-consumption-mode when data collecting finish if it don't receive start signal from MCU again.

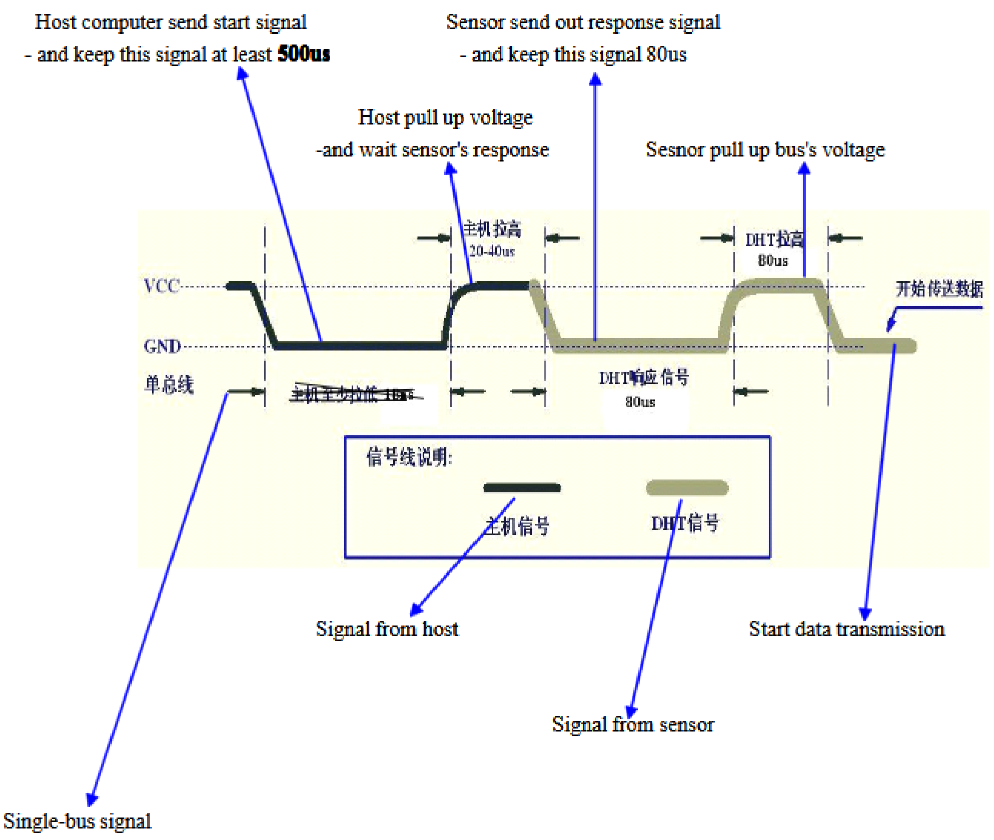
1) Check bellow picture for overall communication process:


Host computer send out start signal.
Sensor send out response signal.
Output data: 1bit"0"
Data transmission finished, and RL pull up bus's voltage for next transmission
主机发开始信号
DHT响应输出
数据'0'
数据传送结束上拉电阻拉高总线
VCC
GND
单总线
拉高并延时等待
拉高延时准备输出
数据'1'
从机拉低 50us
信号线说明:
主机信号
DHT信号
Pull up and wait response from sensor
Host's signal
Sensor's signal
Output data: 1bit "1"
Sensor pull down bus's voltage
Pull up voltage and get ready for sensor's output.
Single-bus output

2) Step 1: MCU send out start signal to AM2303

Data-bus's free status is high voltage level. When communication between MCU and AM2303 begin, program of MCU will transform data-bus's voltage level from high to low level and this process must beyond at least 18ms to ensure AM2303 could detect MCU's signal, then MCU will wait 20-40us for AM2303's response.

Check bellow picture for step 1:

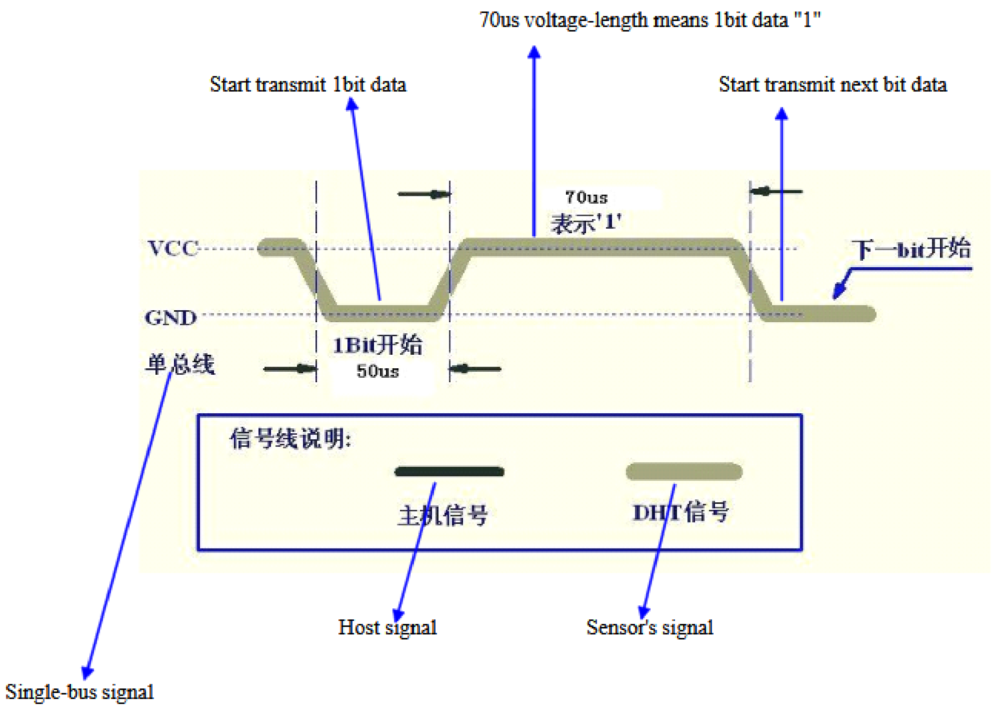


Step 2: AM2303 send response signal to MCU

When AM2303 detect the start signal, AM2303 will send out low-voltage-level signal and this signal last 80us as response signal, then program of AM2303 transform data-bus's voltage level from low to high level and last 80us for AM2303's preparation to send data.

Check bellow picture for step 2:

Tell: +86-020-36380552, +86-020-36042809 Fax: +86-020-36380562
http://www.aosong.com
Email: thomasliu198518@yahoo.com.cn sales@aosong.com
Address: No.56, Renhe Road, Renhe Town, Baiyun District, Guangzhou, China



If signal from AM2303 is always high-voltage-level, it means AM2303 is not working properly, please check the electrical connection status.

6. Electrical Characteristics:

Item	Condition	Min	Typical	Max	Unit
Power supply	DC	3.3	5	5.5	V
Current supply	Measuring	1.3	1.5	2.1	mA
	Average	0.5	0.8	1.1	mA
Collecting period	Second	1.7		2	Second

*Collecting period should be : >1.7 second.

7. Attentions of application:

(1) Operating and storage conditions

We don't recommend the applying RH-range beyond the range stated in this specification. The DHT11 sensor can recover after working in non-normal operating condition to calibrated status, but will accelerate sensors' aging.

(2) Attentions to chemical materials

Vapor from chemical materials may interfere AM2303's sensitive-elements and debase AM2303's sensitivity.

(3) Disposal when (1) & (2) happens

Step one: Keep the AM2303 sensor at condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two: After step one, keep the AM2303 sensor at condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Attention to temperature's affection

Relative humidity strongly depend on temperature, that is why we use temperature compensation technology to ensure accurate measurement of RH. But it's still be much better to keep the sensor at same temperature when sensing.

AM2303 should be mounted at the place as far as possible from parts that may cause change to temperature.

(5) Attentions to light

Long time exposure to strong light and ultraviolet may debase AM2303's performance.

(6) Attentions to connection wires

The connection wires' quality will effect communication's quality and distance, high quality shielding-wire is recommended.

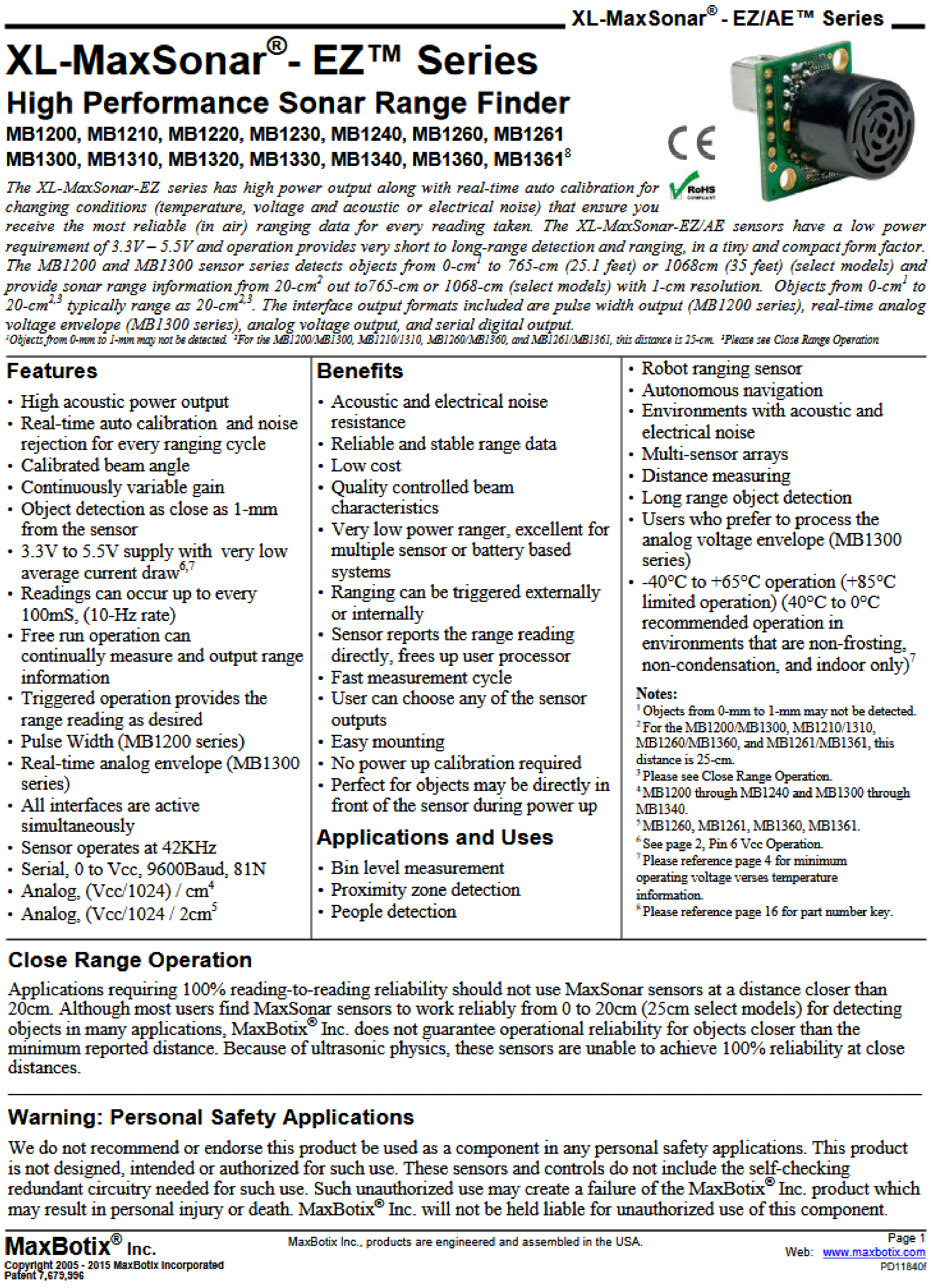
(7) Other attentions

* Welding temperature should be bellow 260Celsius.

* Avoid using the sensor under dew condition.

* Don't use this product in safety or emergency stop devices or any other occasion that failure of AM2303 may cause personal injury.

Annex F – XL-MaxSonar-EZ MB1260 Sensor Datasheet



XL-MaxSonar®- EZ™ Series

High Performance Sonar Range Finder

MB1200, MB1210, MB1220, MB1230, MB1240, MB1260, MB1261
MB1300, MB1310, MB1320, MB1330, MB1340, MB1360, MB1361[8]

The XL-MaxSonar-EZ series has high power output along with real-time auto calibration for changing conditions (temperature, voltage and acoustic or electrical noise) that ensure you receive the most reliable (in air) ranging data for every reading taken. The XL-MaxSonar-EZ/AE sensors have a low power requirement of 3.3V – 5.5V and operation provides very short to long-range detection and ranging, in a tiny and compact form factor. The MB1200 and MB1300 sensor series detects objects from 0-cm[1] to 765-cm (25.1 feet) or 1068cm (35 feet) (select models) and provide sonar range information from 20-cm[2] out to765-cm or 1068-cm (select models) with 1-cm resolution. Objects from 0-cm[1] to 20-cm[2,3] typically range as 20-cm[2,3]. The interface output formats included are pulse width output (MB1200 series), real-time analog voltage envelope (MB1300 series), analog voltage output, and serial digital output.

[1]Objects from 0-mm to 1-mm may not be detected. [2]For the MB1200/MB1300, MB1210/1310, MB1260/MB1360, and MB1261/MB1361, this distance is 25-cm. [3]Please see Close Range Operation

Features

- High acoustic power output
- Real-time auto calibration and noise rejection for every ranging cycle
- Calibrated beam angle
- Continuously variable gain
- Object detection as close as 1-mm from the sensor
- 3.3V to 5.5V supply with very low average current draw[6,7]
- Readings can occur up to every 100mS, (10-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- Pulse Width (MB1200 series)
- Real-time analog envelope (MB1300 series)
- All interfaces are active simultaneously
- Sensor operates at 42KHz
- Serial, 0 to Vcc, 9600Baud, 81N
- Analog, (Vcc/1024) / cm[4]
- Analog, (Vcc/1024 / 2cm[5]

Benefits

- Acoustic and electrical noise resistance
- Reliable and stable range data
- Low cost
- Quality controlled beam characteristics
- Very low power ranger, excellent for multiple sensor or battery based systems
- Ranging can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the sensor outputs
- Easy mounting
- No power up calibration required
- Perfect for objects may be directly in front of the sensor during power up

Applications and Uses

- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Environments with acoustic and electrical noise
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Users who prefer to process the analog voltage envelope (MB1300 series)
- -40°C to +65°C operation (+85°C limited operation) (40°C to 0°C recommended operation in environments that are non-frosting, non-condensation, and indoor only)[7]

Notes:
[1] Objects from 0-mm to 1-mm may not be detected.
[2] For the MB1200/MB1300, MB1210/1310, MB1260/MB1360, and MB1261/MB1361, this distance is 25-cm.
[3] Please see Close Range Operation.
[4] MB1200 through MB1240 and MB1300 through MB1340.
[5] MB1260, MB1261, MB1360, MB1361.
[6] See page 2, Pin 6 Vcc Operation.
[7] Please reference page 4 for minimum operating voltage verses temperature information.
[8] Please reference page 16 for part number key.

Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 20cm. Although most users find MaxSonar sensors to work reliably from 0 to 20cm (25cm select models) for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out

Pin 1-BW-Leave open (or high) for serial output on the Pin 5 output. When Pin 1 is held low the Pin 5 output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2-PW– For the MB1200 (EZ) sensor series, this pin outputs a pulse width representation of range. To calculate distance, use the scale factor of 58uS per cm.

For the MB1300 (AE) sensor series, this pin outputs the analog voltage envelope of the acoustic wave form. The output allows the user to process the raw waveform of the sensor.

Pin 3-AN– For the 7.6 meter sensors (all sensors except for MB1260, MB1261, MB1360, and MB1361), this pin outputs analog voltage with a scaling factor of (Vcc/1024) per cm. A supply of 5V yields ~4.9mV/cm., and 3.3V yields ~3.2mV/cm. Hardware limits the maximum reported range on this output to ~700cm at 5V and ~600cm at 3.3V. The output is buffered and corresponds to the most recent range data.

For the 10 meter sensors (MB1260, MB1261, MB1360, MB1361), this pin outputs analog voltage with a scaling factor of (Vcc/1024) per 2 cm. A supply of 5V yields ~4.9mV/2cm., and 3.3V yields ~3.2mV/2cm. The output is buffered and corresponds to the most recent range data.

Pin 4-RX– This pin is internally pulled high. The XL-MaxSonar-EZ sensors will continually measure range and output if the pin is left unconnected or held high. If held low the will stop ranging. Bring high 20uS or more for range reading.

Pin 5-TX- When Pin 1 is open or held high, the Pin 5 output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in centimeters up to a maximum of 765, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232.

When Pin 1 is held low, the Pin 5 output sends a single pulse, suitable for low noise chaining (no serial data).

Pin 6-+5V- Vcc – Operates on 3.3V - 5V. The average (and peak) current draw for 3.3V operation is 2.1mA (50mA peak) and at 5V operation is 3.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit. Please reference page 4 for minimum operating voltage verses temperature information.

Pin 7-GND- Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.

Product Release Notes

For all MB1260/MB1360 sensors sold after Feb 20, 2013, the minimum reported distance is 25cm.
For all MB1261/MB1361 sensors sold after Feb 20, 2013, the minimum reported distance is 25cm.
For all MB1200/MB1300 sensors sold after Oct 01, 2013, the minimum reported distance is 25cm.
For all MB1210/MB1310 sensors sold after Oct 01, 2013, the minimum reported distance is 25cm.

Sensor Minimum Distance

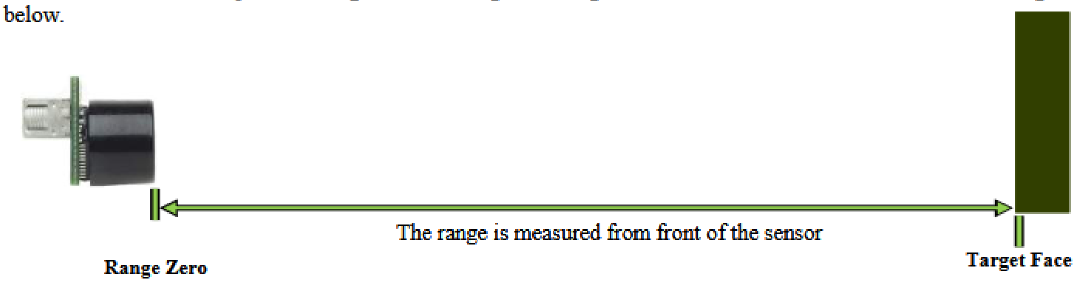
The sensor minimum reported distance is 20cm[1] (7.87 inches). However, the XL-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 20cm[1] will typically range as 20cm[1].

Sensor Operation from 6-inches to 20-inches

Because of acoustic phase effects in the near field, objects between 20cm and 50cm may experience acoustic phase cancellation of the returning waveform resulting in inaccuracies. These effects become less prevalent as the target distance increases, and has not been observed past 50cm. For this reason, industrial users that require the highest sensor accuracy are encouraged to mount the XL-MaxSonar-EZ from objects that are farther than 50cm.

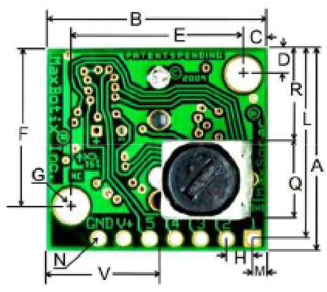
Range "0" Location

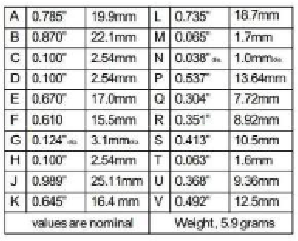
The XL-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.



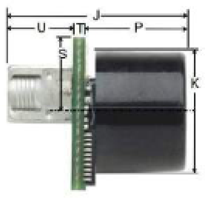
In general, the XL-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Mechanical Dimensions





A	0.785"	19.9mm	L	0.735"	18.7mm
B	0.870"	22.1mm	M	0.065"	1.7mm
C	0.100"	2.54mm	N	0.038" dia.	1.0mm dia.
D	0.100"	2.54mm	P	0.537"	13.64mm
E	0.670"	17.0mm	Q	0.304"	7.72mm
F	0.610	15.5mm	R	0.351"	8.92mm
G	0.124" dia.	3.1mm dia.	S	0.413"	10.5mm
H	0.100"	2.54mm	T	0.063"	1.6mm
J	0.989"	25.11mm	U	0.368"	9.36mm
K	0.645"	16.4 mm	V	0.492"	12.5mm
values are nominal			Weight, 5.9 grams		



Real-Time Auto Calibration

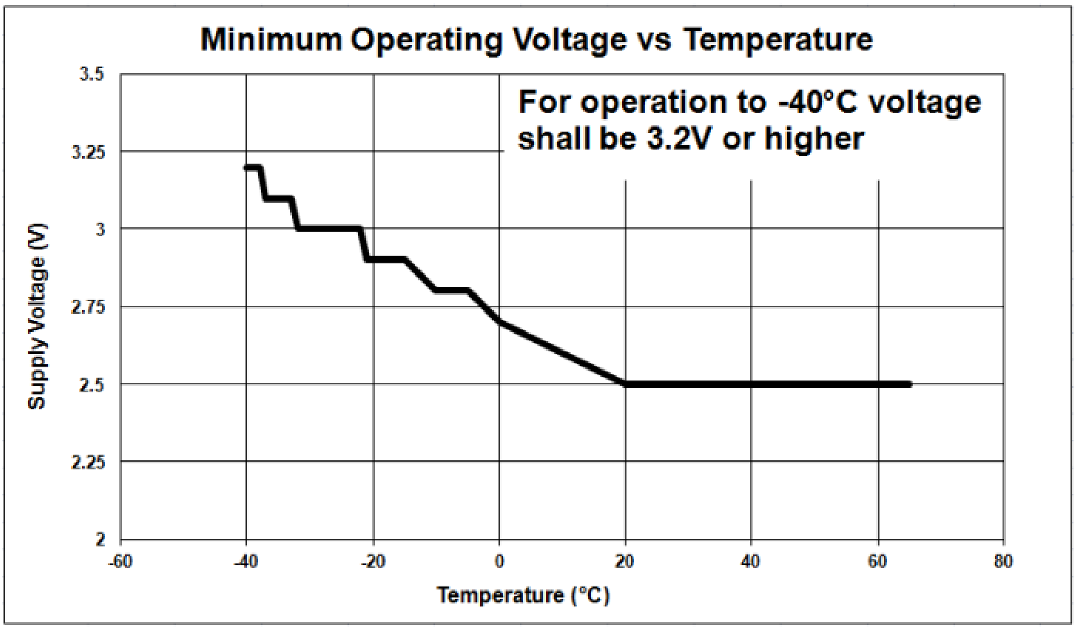
Each time before the XL-MaxSonar sensor takes a range reading it calibrates itself. The sensor then uses this data to range to objects. If the temperature, humidity, or applied voltage changes during operation, the sensor will continue to function normally. The sensor does not apply compensation for the speed of sound change versus temperature to any range readings.

Temperature Compensation

The speed of sound in air increases about 0.6 meters per second, per degree centigrade. The XL-MaxSonar-EZ sensors are not equipped with an internal temperature compensation. If temperature compensation is desired, contact MaxBotix and request the temperature compensation formula PDF. This will allow users to compensation for speed of sound changes.

Voltage vs Temperature

The graph below shows minimum operating voltage of the sensor verses temperature.



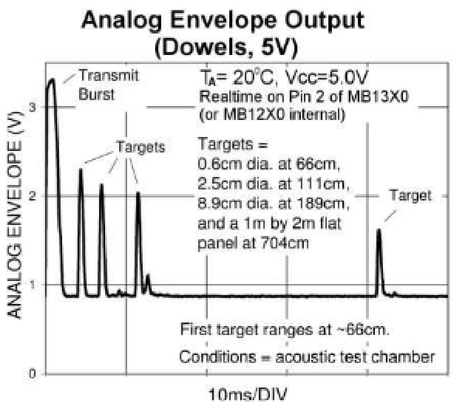
Real-time Noise Rejection

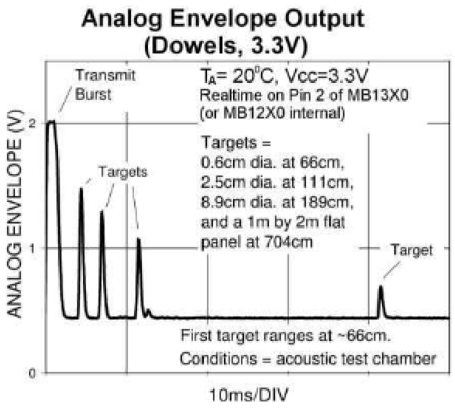
While the XL-MaxSonar® is designed to operate in the presence of noise, best operation is obtained when noise strength is low and desired signal strength is high. Hence, the user is encouraged to mount the sensor in such a way that minimizes outside acoustic noise pickup. In addition, keep the DC power to the sensor free of noise. This will let the sensor deal with noise issues outside of the users direct control (in general, the sensor will still function well even if these things are ignored). Users are encouraged to test the sensor in their application to verify usability.

For every ranging cycle, individual filtering for that specific cycle is applied. In general, noise from regularly occurring periodic noise sources such as motors, fans, vibration, etc., will not falsely be detected as an object. This holds true even if the periodic noise increases or decreases (such as might occur in engine throttling or an increase/decrease of wind movement over the sensor). Even so, it is possible for sharp non-periodic noise sources to cause false target detection. In addition, *(because of dynamic range and signal to noise physics,) as the noise level increases, at first only small targets might be missed, but if noise increases to very high levels, it is likely that even large targets will be missed.

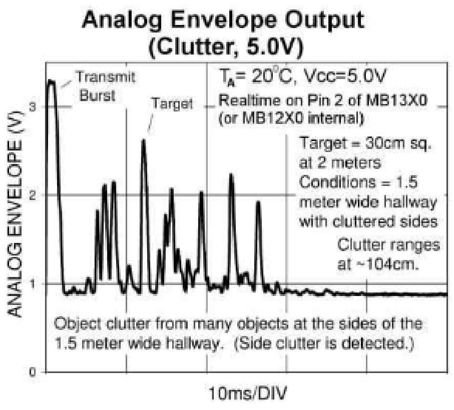
*In high noise environments, if needed, use 5V power to keep acoustic signal power high. In addition, a high acoustic noise environment may use some of the dynamic range of the sensor, so consider a part with less gain such as the MB1220/MB1320 MB1230/MB1330 or MB1240/MB1340. For applications with large targets, consider a part with ultra clutter rejection like the MB7369.

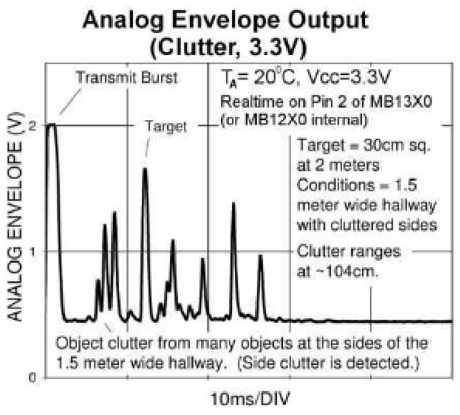
Typical Performance to Target





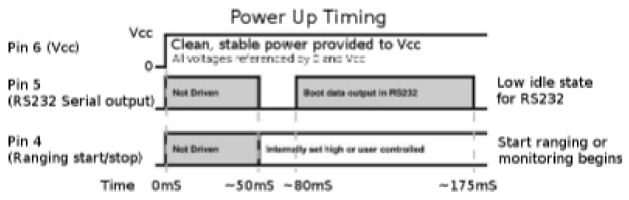
Typical Performance in clutter



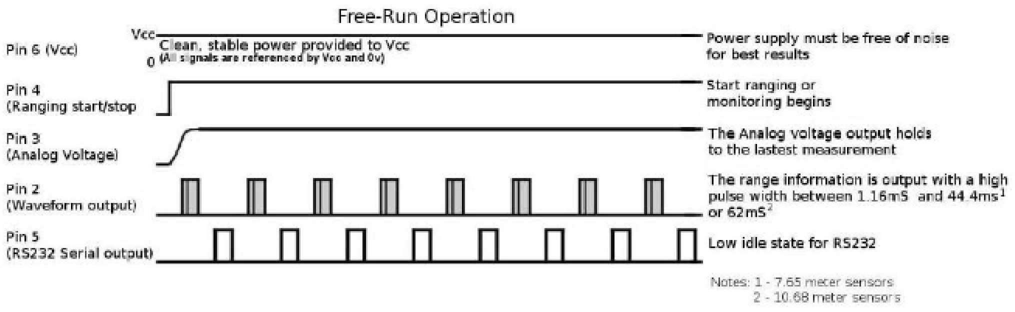


Sensor Timing Diagrams

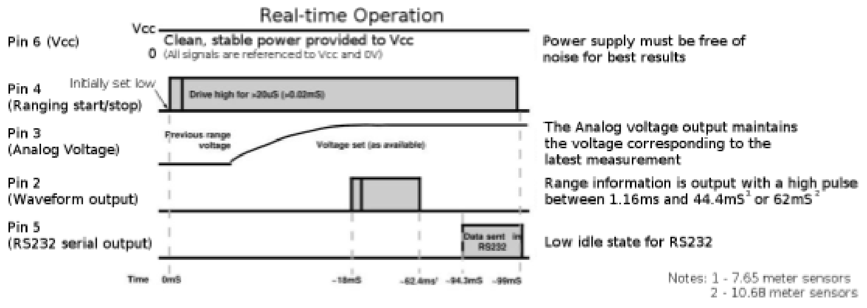
Power Up Timing


Power Up Timing
Pin 6 (Vcc)
Vcc
0
Clean, stable power provided to Vcc
All voltages referenced by 0 and Vcc
Pin 5
(RS232 Serial output)
Not Driven
Boot data output in RS232
Low idle state
for RS232
Pin 4
(Ranging start/stop)
Not Driven
Internally set high or user controlled
Start ranging or
monitoring begins
Time 0mS ~50mS ~80mS ~175mS

Sensor Free-Run Timing


Free-Run Operation
Pin 6 (Vcc)
Vcc
0
Clean, stable power provided to Vcc
(All signals are referenced by Vcc and 0v)
Power supply must be free of noise
for best results
Pin 4
(Ranging start/stop
Start ranging or
monitoring begins
Pin 3
(Analog Voltage)
The Analog voltage output holds
to the lastest measurement
Pin 2
(Waveform output)
The range information is output with a high
pulse width between 1.16mS and 44.4ms1
or 62mS2
Pin 5
(RS232 Serial output)
Low idle state for RS232
Notes: 1 - 7.65 meter sensors
2 - 10.68 meter sensors

Real-Time Operation


Real-time Operation
Pin 6 (Vcc)
Vcc
0
Clean, stable power provided to Vcc
(All signals are referenced to Vcc and 0V)
Power supply must be free of
noise for best results
Pin 4
(Ranging start/stop)
Initially set low
Drive high for >20uS (>0.02mS)
Pin 3
(Analog Voltage)
Previous range voltage
Voltage set (as available)
The Analog voltage output maintains
the voltage corresponding to the
latest measurement
Pin 2
(Waveform output)
Range information is output with a high pulse
between 1.16ms and 44.4mS1 or 62mS2
Pin 5
(RS232 serial output)
Data sent in RS232
Low idle state for RS232
Time 0mS ~18mS ~62.4mS1 ~94.3mS ~99mS
Notes: 1 - 7.65 meter sensors
2 - 10.68 meter sensors

Timing Description

175mS after power-up, the XL-MaxSonar is ready to begin ranging. If Pin-4 is left open or held high (20uS or greater), the sensor will take a range reading. The XL-MaxSonar checks the Pin-4 at the end of every cycle. Range data can be acquired once every 99mS. Each 99mS period starts by Pin-4 being high or open, after which the XL-MaxSonar calibrates and calculates for 20.5mS, and after which, twenty 42KHz waves are sent.

At this point, for the MB1260, the pulse width (PW) Pin-2 is set high and until an object is detected after which the pin is set low. If no target is detected the PW pin will be held high for up to 44.4mS[1] (i.e. 58uS * 765cm) or 62.0mS[2] (i.e. 58uS * 1068cm). (For the most accurate range data, use the PW output.)

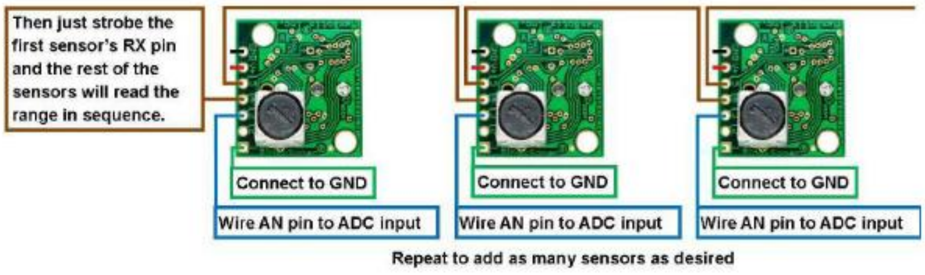
For the MB1300 sensor series, The analog envelope output, Pin-2, will show the real-time signal return information of the analog waveform.

For both parts, the remainder of the 99mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level, (and allowing the high acoustic power to dissipate). During the last 4.7mS, the serial data is sent.

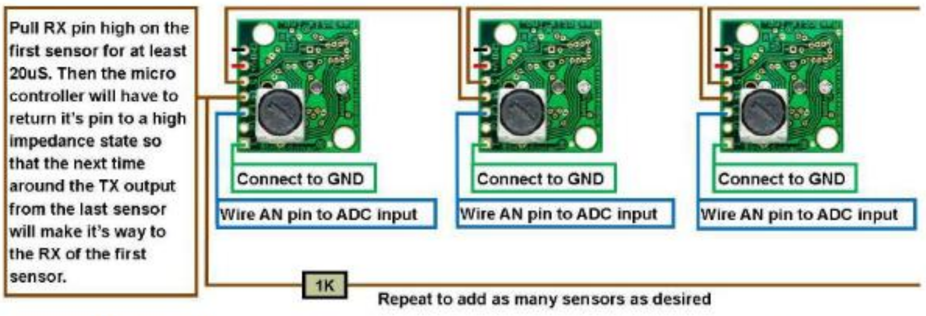
Using Multiple Sensors in a single system

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered a solution to this problem for the XL-MaxSonar-EZ sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

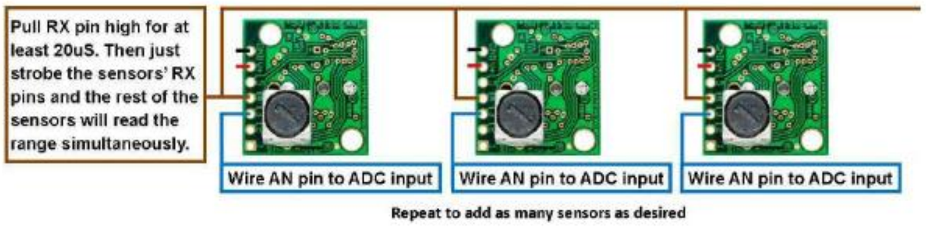
The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.


Then just strobe the first sensor's RX pin and the rest of the sensors will read the range in sequence.
Connect to GND
Connect to GND
Connect to GND
Wire AN pin to ADC input
Wire AN pin to ADC input
Wire AN pin to ADC input
Repeat to add as many sensors as desired

The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.


Pull RX pin high on the first sensor for at least 20uS. Then the micro controller will have to return it's pin to a high impedance state so that the next time around the TX output from the last sensor will make it's way to the RX of the first sensor.
Connect to GND
Connect to GND
Connect to GND
Wire AN pin to ADC input
Wire AN pin to ADC input
Wire AN pin to ADC input
1K
Repeat to add as many sensors as desired

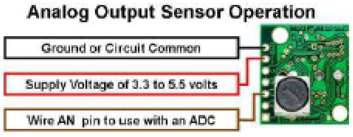
The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are conned together and triggered at the same time causing all the sensor to take a range reading at the same time. Once the range reading is complete, the sensors stop ranging until triggered next time. Below is a diagram on how to set this up.


Pull RX pin high for at least 20uS. Then just strobe the sensors' RX pins and the rest of the sensors will read the range simultaneously.
Wire AN pin to ADC input
Wire AN pin to ADC input
Wire AN pin to ADC input
Repeat to add as many sensors as desired

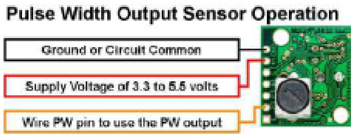
Independent Sensor Operation

The XL-MaxSonar-EZ sensors have the capability to operating independently when the user desires. When using the XL-MaxSonar-EZ sensors in single or independent sensor operation, it is easiest to allow the sensor to free-run. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The XL-MaxSonar-EZ sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width, and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs when operating in a single or independent sensor operating environment.

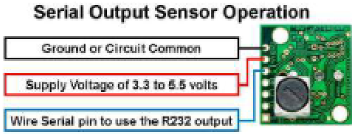
Analog Output Sensor Operation


Ground or Circuit Common
Supply Voltage of 3.3 to 5.5 volts
Wire AN pin to use with an ADC

Pulse Width Output Sensor Operation


Ground or Circuit Common
Supply Voltage of 3.3 to 5.5 volts
Wire PW pin to use the PW output

Serial Output Sensor Operation


Ground or Circuit Common
Supply Voltage of 3.3 to 5.5 volts
Wire Serial pin to use the R232 output

Selecting a XL-MaxSonar-EZ/AE

Different applications require different sensors. The XL-MaxSonar-EZ/AE product line offers varied sensitivity to allow you to select the best sensor to meet your needs.

The XL-MaxSonar-EZ Sensors At a Glance

People Detection Wide Beam High Sensitivity		**Best Balance**	**Large Targets Narrow Beam Noise Tolerance**	
MB1200/MB1300	**MB1210/MB1310**	**MB1220/MB1320**	**MB1230/MB1330**	**MB1240/MB1340**
MB1260/MB1261	**MB1261/MB1361**			

The diagram above shows how each product balances sensitivity and noise tolerance. This does not effect the maximum range, pin outputs, or other operations of the sensor. To view how each sensor will function to different sized targets reference the XL-MaxSonar-EZ Beam Patterns.

Beam Characteristics

Background Information Regarding our Beam Patterns

Each XL-MaxSonar-EZ sensor has a calibrated beam pattern. Each sensor is matched to provide the approximate detection pattern shown in this datasheet. This allows end users to select the part number that matches their given sensing application. Each part number has a consistent field of detection so additional units of the same part number will have similar beam patterns. The beam plots are provided to help identify an estimated detection zone for an application based on the acoustic properties of a target versus the plotted beam patterns.
Each beam pattern is a 2D representation of the detection area of the sensor. The beam pattern is actually shaped like a 3D cone (having the same detection pattern both vertically and horizontally). Detection patterns for dowels are used to show the beam pattern of each sensor. Dowels are long cylindered targets of a given diameter. The dowels provide consistent target detection characteristics for a given size target which allows easy comparison of one MaxSonar sensor to another MaxSonar sensor.

People Sensing: For users that desire to detect people, the detection area to the 1-inch diameter dowel, in general, represents the area that the sensor will reliably detect people.

For each part number, the four patterns (A, B, C, and D) represent the detection zone for a given target size. Each beam pattern shown is determined by the sensor's part number and target size.
The actual beam angle changes over the full range. Use the beam pattern for a specific target at any given distance to calculate the beam angle for that target at the specific distance. Generally, smaller targets are detected over a narrower beam angle and a shorter distance. Larger targets are detected over a wider beam angle and a longer range.

MB1260/MB1360: XL-MaxSonar-EZ/AEL0

The XL-MaxSonar-EZL/AEL0 has the same gain and sensitive as the MB1200/MB1300. This sensor features a longer range of 1068cm to large targets. This sensor is recommended for long range measurement to large targets.

Note: Firmware rev 1.6b and newer have a 25cm minimum reported distance. This applies to all sensors sold after February 20, 2013. All sensors sold before this date have a 20cm minimum reported distance.

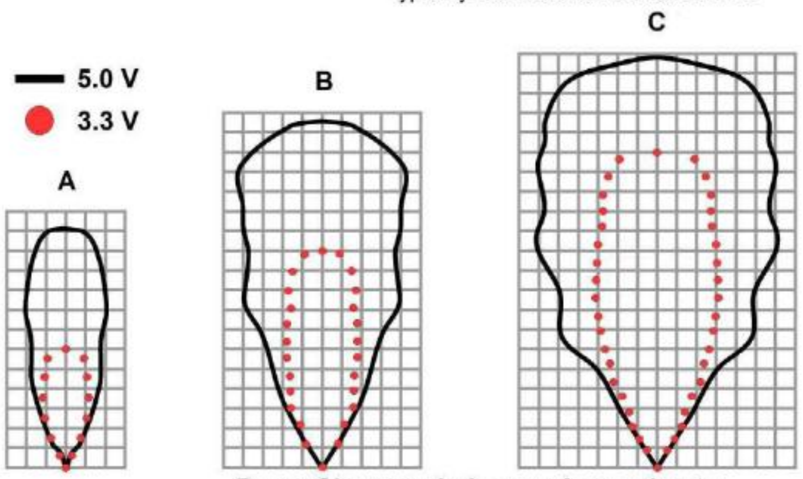
MB1260-MB1360

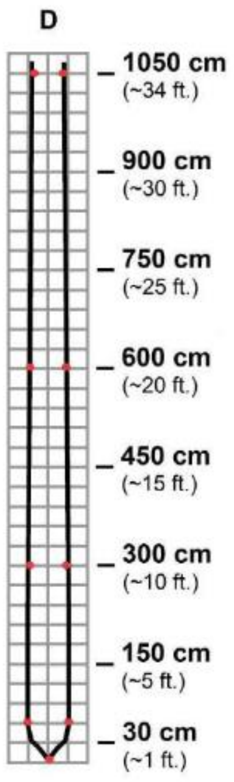
XL-MaxSonar®-EZ/AEL0™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel
B 2.54-cm (1-inch) diameter dowel
C 8.89-cm (3.5-inch) diameter dowel
D 11-inch wide board moved left to right with the board parallel to the front sensor face. This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.





Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

MB1260/MB1360
Features and Benefits

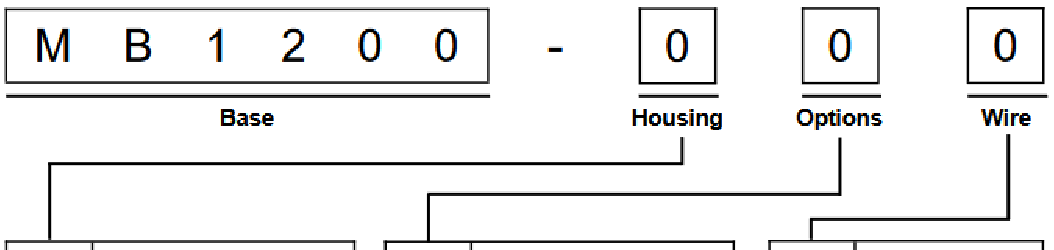
- Shares same beam pattern with MB1200/MB1300
- Maximum range of 1068cm to large targets
- Low power consumption
- Easy to use interface
- Can detect people to approximately 18feet
- 3.3v to 5v operational voltage

MB1260/MB1360
Applications and Uses

- Great for people detection
- Security
- Motion detection
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance
- Long range detection

Part Numbers

All part numbers are a combination of a six-character base followed by a dash and a three-digit product code. Please review the following table for more information on the three-digit product code.



0	Not Applicable
1	3/4" NPS WR
2	3/4" NPS WRC
3	Ultra Compact
4	Ultra Compact Flush Mount
5	1" NPS
6	1" BSPP
7	30MM 1.5
8	Extended Horn

0	No Options (Bagged)
1	F-Option
2	P-Option
3	F-Option and P-Option
4	No Options (Trayed)
5	TTL (Bagged)
6	TTL (Trayed)

0	No Wire
1	Wire Attached

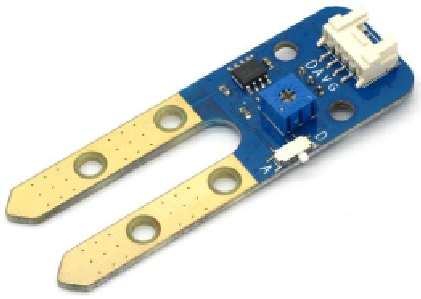
The following table displays all of the active and valid part numbers for this product.

Active Part Numbers for MB1200, MB1210, MB1220, MB1230, MB1240, MB1260 and MB1261						
MB1200-000	MB1210-000	MB1220-000	MB1230-000	MB1240-000	MB1260-000	MB1261-000
MB1200-040	MB1210-040	MB1220-040	MB1230-040	MB1240-040	MB1260-040	MB1261-040
Active Part Numbers for MB1300, MB1310, MB1320, MB1330, MB1340, MB1360 and MB1361						
MB1300-000	MB1310-000	MB1320-000	MB1330-000	MB1340-000	MB1360-000	MB1361-000
MB1300-040	MB1310-040	MB1320-040	MB1330-040	MB1340-040	MB1360-040	MB1361-040

Annex G – Earth-Humidity Sensor Datasheet

Soil Moisture Sensor

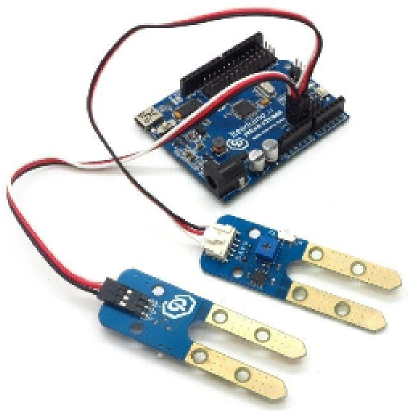
Overview



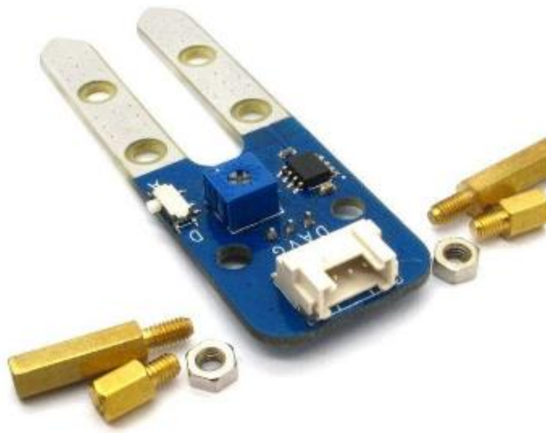
What is an electronic brick? An electronic brick is an electronic module which can be assembled like Lego bricks simply by plugging in and pulling out. Compared to traditional universal boards and circuit modules assembled with various electronic components, electronic brick has standardized interfaces, plug and play, simplifying construction of prototype circuit on one' s own. There are many types of electronic bricks, and we provide more than twenty types with different functions including buttons, sensors, Bluetooth modules, etc, whose functions cover from sensor to motor drive, from Ethernet to wireless communication via Bluetooth, and so on. We will continue to add more types to meet the various needs of different projects.
Electronic brick of soil moisture sensor is mainly used to detect the moisture content in the soil. The control board can get the moisture value or threshold in the soil via analog or digital pins.

Features

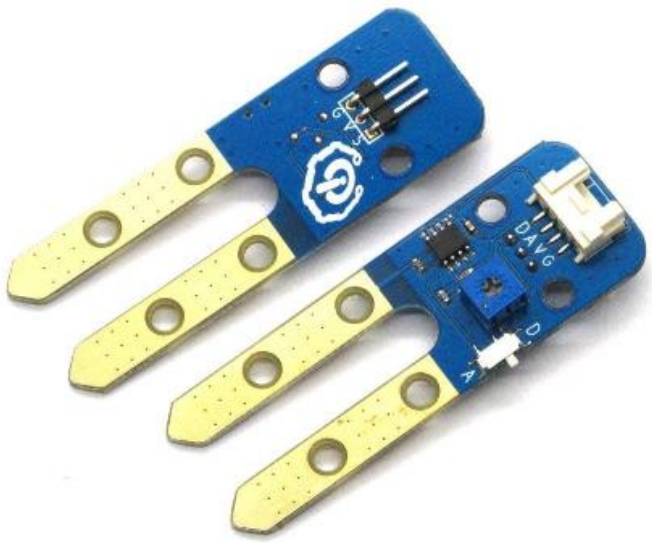
1. Plug and play, easy to use. Compatible with the mainstream 2.54 interfaces and 4-Pin Grove interfaces in the market.



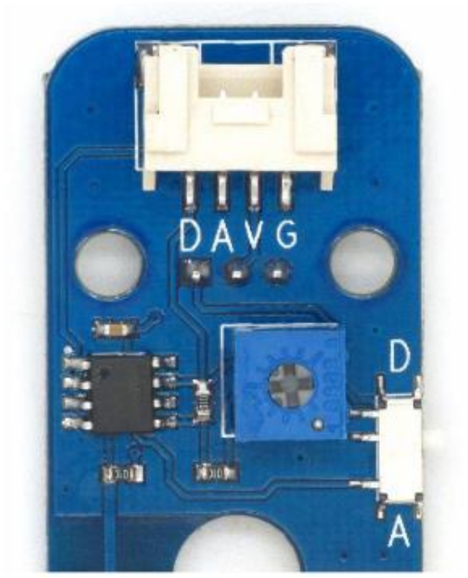
2. With use of M4 standard fixed holes, compatible with M4-standard kits such as Lego and Makeblock.



3. With switch to shift between analog and digital output. Able to read the specific soil moisture information (analog) or the over-wet or over-dry soil information according to the threshold (digital). The adjustable potentiometer is used to set the moisture threshold.



4. With hysteresis comparator circuit for more stable digital output voltage.



Specifications

PCB size	71.65mm X 24.00mm X 1.6mm
Working voltage	3.3or 5V DC
Operating voltage	3.3 or 5V DC
Compatible interfaces	2.54 3-pin interface and 4-pin Grove interface(1)(2)

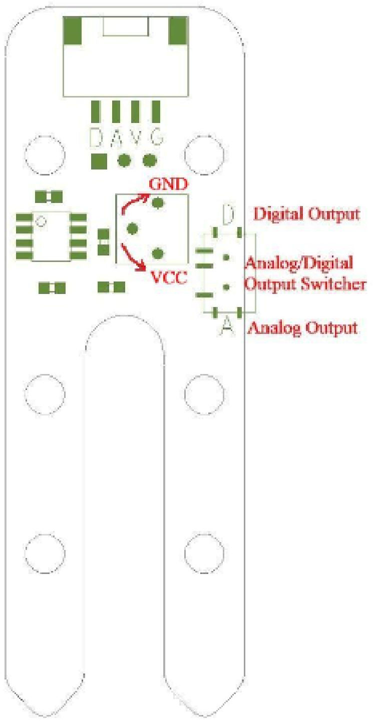
Note 1 : D for digital output port, A for analog output port, S for analog/digital output port (defined according to the switch), V and G for voltage at the common collector and ground respectively

Note 2 : When setting as analog output, output range is 0-3.3V or 0-5V according to the working voltage; when setting as digital output, output is 0/3.3V or 0/5V according to the working voltage.

Electrical characteristics

Parameter	Min.	Typical	Max.	Unit
Working voltage	2.1	5	5.5	VDC
Analog output voltage (VCC=5V)	0	Vout	5	V
Digital output voltage (VCC=5V)	0	-	5	V
Working current (VCC=5V)	-	5	-	mA
Threshold hysteresis ΔUth	-	VCC*0.09	-	V

Hardware

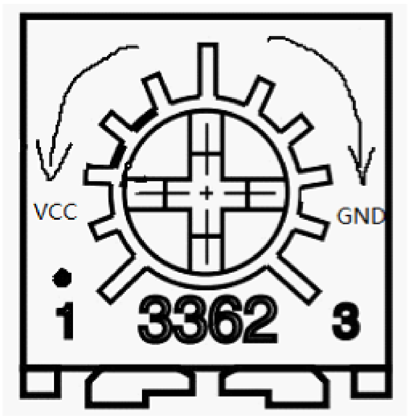


Top view

Switch and indicator

1. Regulating of threshold voltage:

The threshold voltage is a voltage for comparison. When the soil moisture value read by the sensor is above the threshold value, a low level (0V) will be digitally output; when the soil moisture value read by the sensor is below the threshold value, a high level (3.3V or 5V) will be digitally output. In this way, the digital pin can be used directly to read the current soil moisture value to see if it is above the threshold or not. The threshold voltage can be regulated by simply twisting the potentiometer which is shown in Figure 1., and it increases by rotating to left side and decreases by rotating to right side.



2. Switch to shift between analog and digital output
   For 4-pin Grove interface, this switch makes no sense.
   For S pin in 2.54mm 3-pin interface, it outputs analog signals when switch is pushed to A terminal and digital signals when pushed to D terminal. When there is analog signal output, it can read the specific soil moisture value; when there is digital signal output, it can only indicate whether the soil moisture value is above threshold value or not.

DEMO

1. Push the snap switch of sensor to analog output, connect S port to A0 port of Arduino board, and we will use the following program to read the analog value of soil and send it to computer for display via the serial port.

```
int ASignal = A0;
void setup() {
  Serial.begin(9600);
}


void loop() {
  int sensorValue = analogRead(ASignal);
  Serial.println(sensorValue);
}
```

2. Push the snap switch of sensor to digital output, connect S port to D2 port of Arduino board, and we will use the following program to read the digital value of soil and send the threshold to computer for display .

```
int DSIGNAL = 2;
void setup()
{
  Serial.begin(9600);
  pinMode(DSIGNAL, INPUT);
}

void loop()
{
  int DsignalState = digitalRead(DSIGNAL);
  Serial.println(DsignalState);
  delay(100);
}
```

Revision history

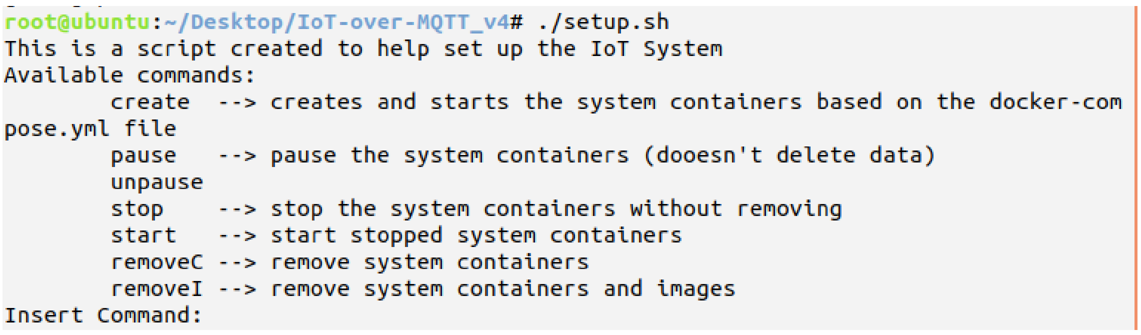
Version	Description	Date	Written by	Audited by
v1.0	Initial edition	12th, April, 2013	Stan Lee	Tse Zhe

Annex H – System Set-Up

To simplify the IoT System set-up a simple shell script was written, named "setup.sh", which only requires the user to insert well defined commands to manage the System and the underlying Docker Images and Containers without the need to know Docker commands.

```
echo "This is a script created to help set up the IoT System
Available commands:
      create      --> creates and starts the system containers based
on the docker-compose.yml file
      pause       --> pause the system containers (dooesn't delete
data)
      unpause
      stop        --> stop the system containers without removing
      start       --> start stopped system containers
      removeC     --> remove system containers
      removeI     --> remove system containers and images"

echo -n "Insert Command: "
read x

command="$x"
case "${command}" in
      "help")
        echo "usage: commands
[create|pause|unpause|stop|start|remove]"
        ;;
      "create")
            echo "downloading images, creating containers, starting
containers"
            docker-compose -p iotSystem up -d --remove-orphans
            sleep 2
      ;;
      "pause")
            echo "pausing containers"
            docker-compose -p iotSystem pause
            ;;
      "unpause")
            echo "unpausing containers"
            docker-compose -p iotSystem unpause
            ;;
      "stop")
            echo "stopping containers"
            docker-compose -p iotSystem stop
            ;;
      "start")
            echo "starting containers"
            docker-compose -p iotSystem start
            ;;
      "removeC")
            echo "removing containers"
            docker-compose -p iotSystem down
            ;;
      "removeI")
            echo "removing containers and images"
            docker-compose -p iotSystem down --rmi all
            ;;
      *)
            echo "Command not Found."
            Echo "usage: commands
[create|pause|unpause|stop|start|remove]"
            exit 127;
            ;;
esac
```

When the set-up shell script is run, the following output is presented, Figure H.1, asking the user to insert one of the commands available.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-com
pose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command:
```

Figure H.2 Set-Up Shell Script Output

The “create” command, runs the appropriate Docker-Compose command to pull the necessary Images, create, configure and start the Containers as defined in the “docker-compose.yml” file presented in the previous section.

The “pause” command, pauses the IoT System Containers, which can be unpaused by using the “unpause” command.

The “stop” command, stops the IoT System Containers, which can be started again, by using the “start” command.

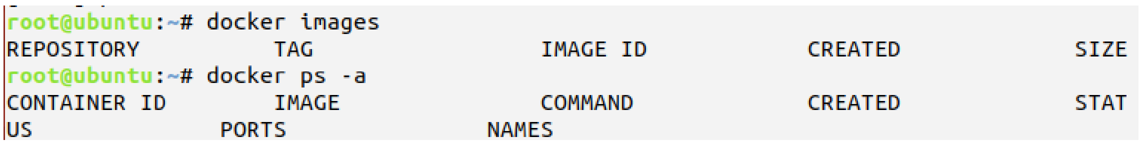
The “removeC” command, removes the IoT System Containers and all data is lost.

The “removeI” commands, removes the IoT System Containers and the Docker Images used to create them, allowing to restart from zero.

If none of the above commands is inserted, an error is returned, and the script exits.

“create” Command

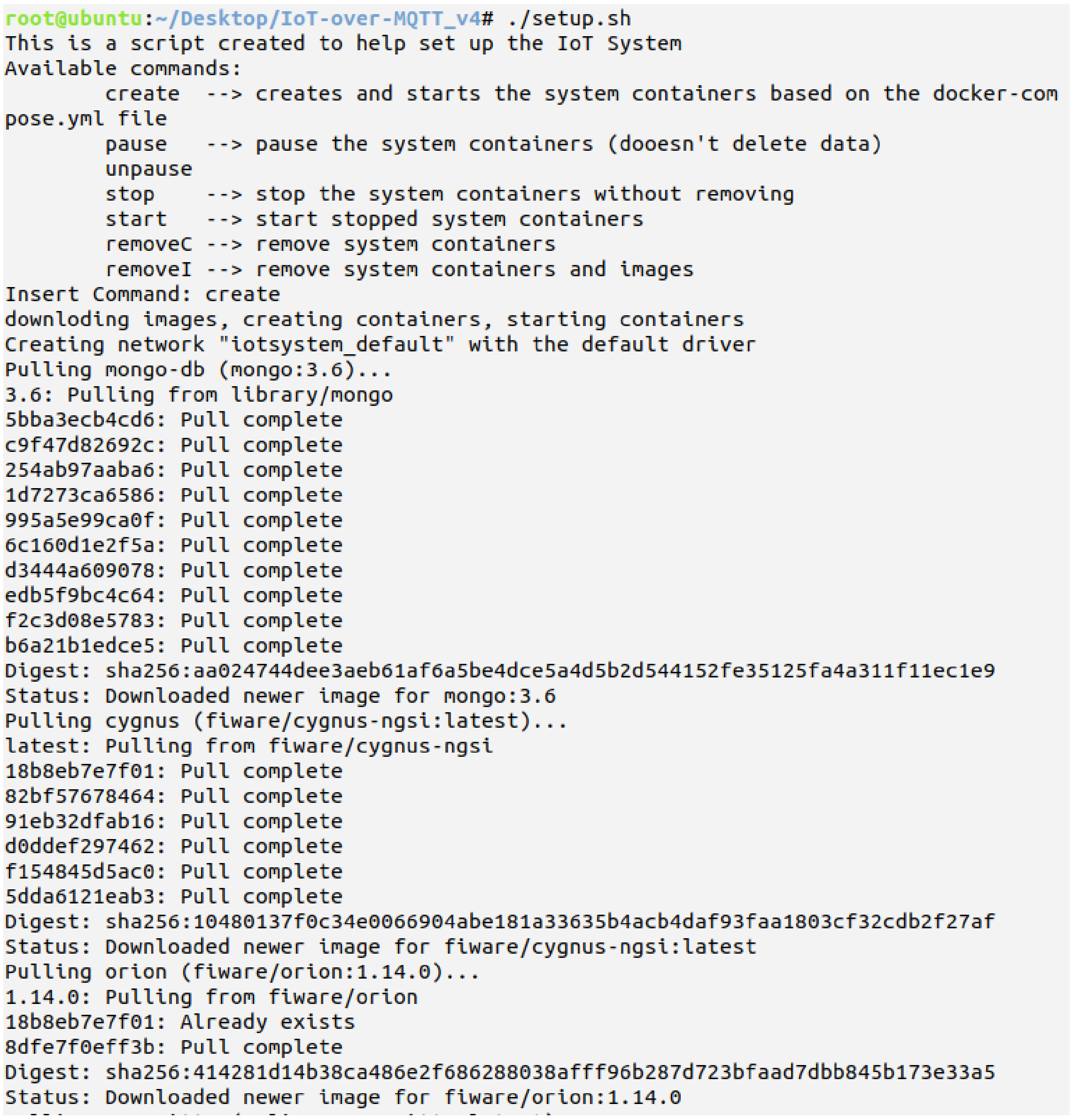
Figure H.2 shows the Docker Images and Containers before the script execution, where it is observed that there aren’t any Images and Containers.



```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STAT
US                PORTS               NAMES
```

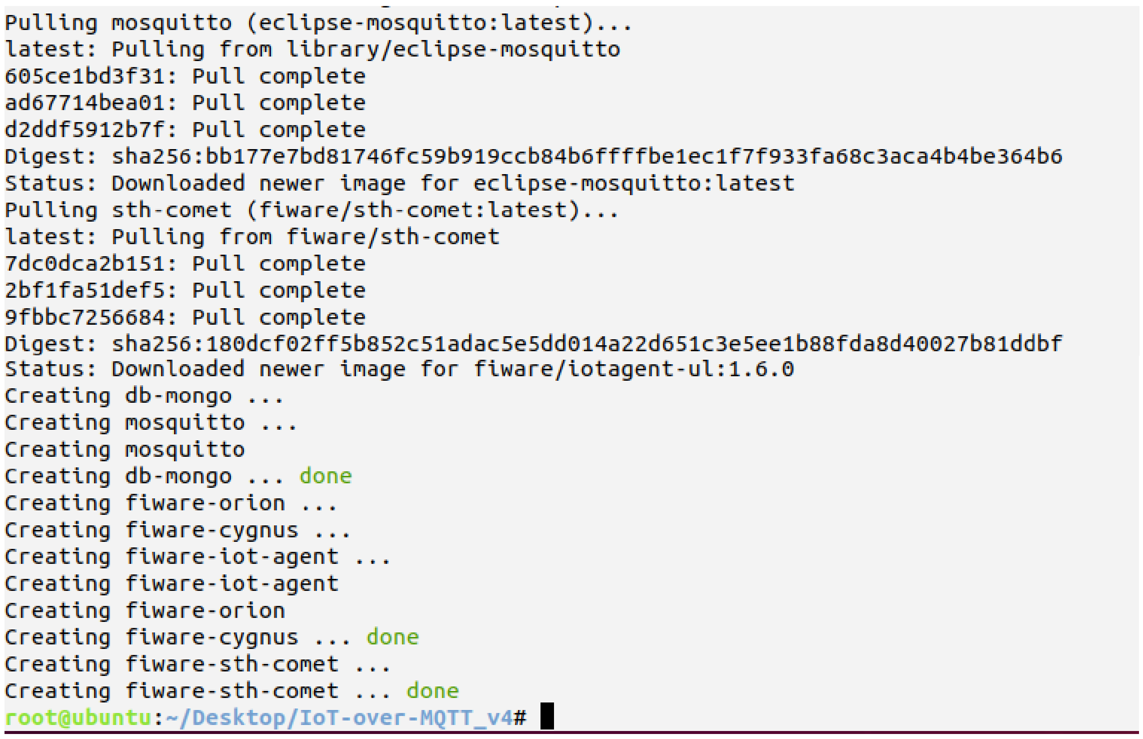
Figure H.3 Docker Images and Containers Before the Script Execution

Figure H.3 displays the script execution output, where it is possible to visualize the downloading of the Images and the creation of the Containers.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-com
pose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: create
downloding images, creating containers, starting containers
Creating network "iotsystem_default" with the default driver
Pulling mongo-db (mongo:3.6)...
3.6: Pulling from library/mongo
5bba3ecb4cd6: Pull complete
c9f47d82692c: Pull complete
254ab97aaba6: Pull complete
1d7273ca6586: Pull complete
995a5e99ca0f: Pull complete
6c160d1e2f5a: Pull complete
d3444a609078: Pull complete
edb5f9bc4c64: Pull complete
f2c3d08e5783: Pull complete
b6a21b1edce5: Pull complete
Digest: sha256:aa024744dee3aeb61af6a5be4dce5a4d5b2d544152fe35125fa4a311f11ec1e9
Status: Downloaded newer image for mongo:3.6
Pulling cygnus (fiware/cygnus-ngsi:latest)...
latest: Pulling from fiware/cygnus-ngsi
18b8eb7e7f01: Pull complete
82bf57678464: Pull complete
91eb32dfab16: Pull complete
d0ddef297462: Pull complete
f154845d5ac0: Pull complete
5dda6121eab3: Pull complete
Digest: sha256:10480137f0c34e0066904abe181a33635b4acb4daf93faa1803cf32cdb2f27af
Status: Downloaded newer image for fiware/cygnus-ngsi:latest
Pulling orion (fiware/orion:1.14.0)...
1.14.0: Pulling from fiware/orion
18b8eb7e7f01: Already exists
8dfe7f0eff3b: Pull complete
Digest: sha256:414281d14b38ca486e2f686288038afff96b287d723bfaad7dbb845b173e33a5
Status: Downloaded newer image for fiware/orion:1.14.0
```

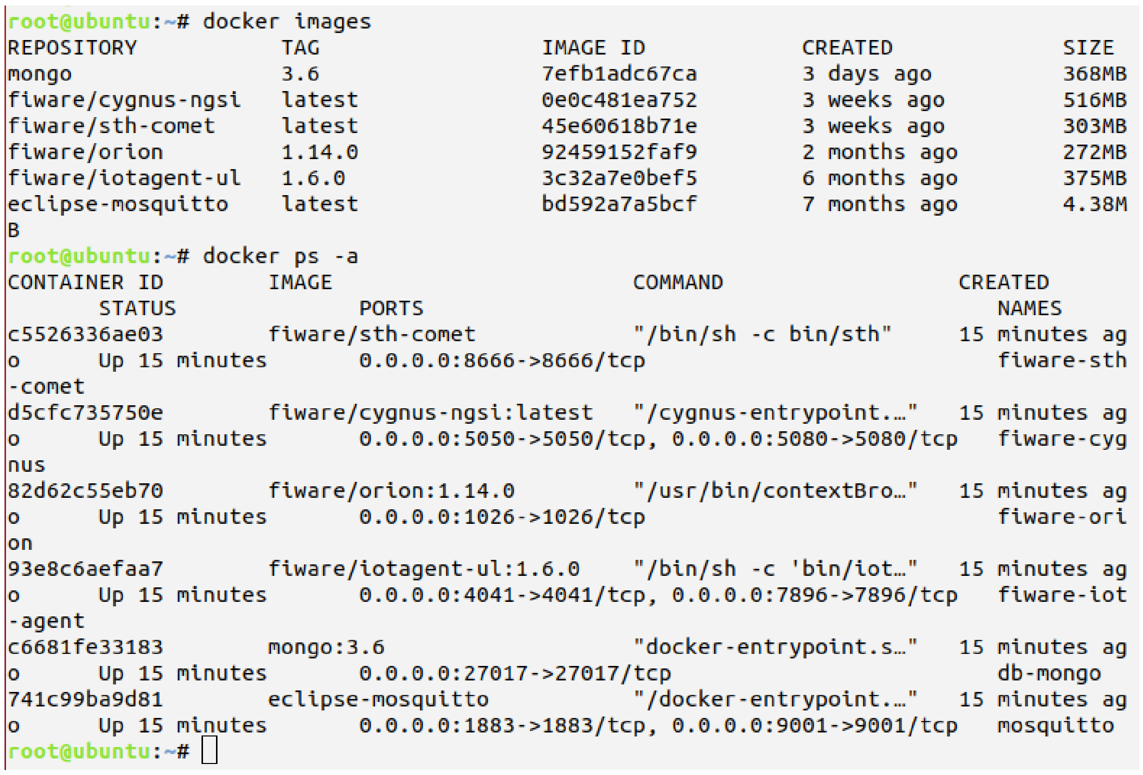
Figure H.3 "create" Command Output – Part 1



```
Pulling mosquitto (eclipse-mosquitto:latest)...
latest: Pulling from library/eclipse-mosquitto
605ce1bd3f31: Pull complete
ad67714bea01: Pull complete
d2ddf5912b7f: Pull complete
Digest: sha256:bb177e7bd81746fc59b919ccb84b6ffffbe1ec1f7f933fa68c3aca4b4be364b6
Status: Downloaded newer image for eclipse-mosquitto:latest
Pulling sth-comet (fiware/sth-comet:latest)...
latest: Pulling from fiware/sth-comet
7dc0dca2b151: Pull complete
2bf1fa51def5: Pull complete
9fbbc7256684: Pull complete
Digest: sha256:180dcf02ff5b852c51adac5e5dd014a22d651c3e5ee1b88fda8d40027b81ddbf
Status: Downloaded newer image for fiware/iotagent-ul:1.6.0
Creating db-mongo ...
Creating mosquitto ...
Creating mosquitto
Creating db-mongo ... done
Creating fiware-orion ...
Creating fiware-cygnus ...
Creating fiware-iot-agent ...
Creating fiware-iot-agent
Creating fiware-orion
Creating fiware-cygnus ... done
Creating fiware-sth-comet ...
Creating fiware-sth-comet ... done
root@ubuntu:~/Desktop/IoT-over-MQTT_v4#
```

Figure H.4 "create" Command Output – Part 2

Figure H.4 shows the Docker Images and Containers after the script execution, confirming that the Images where downloaded and that the Containers where created and are running.

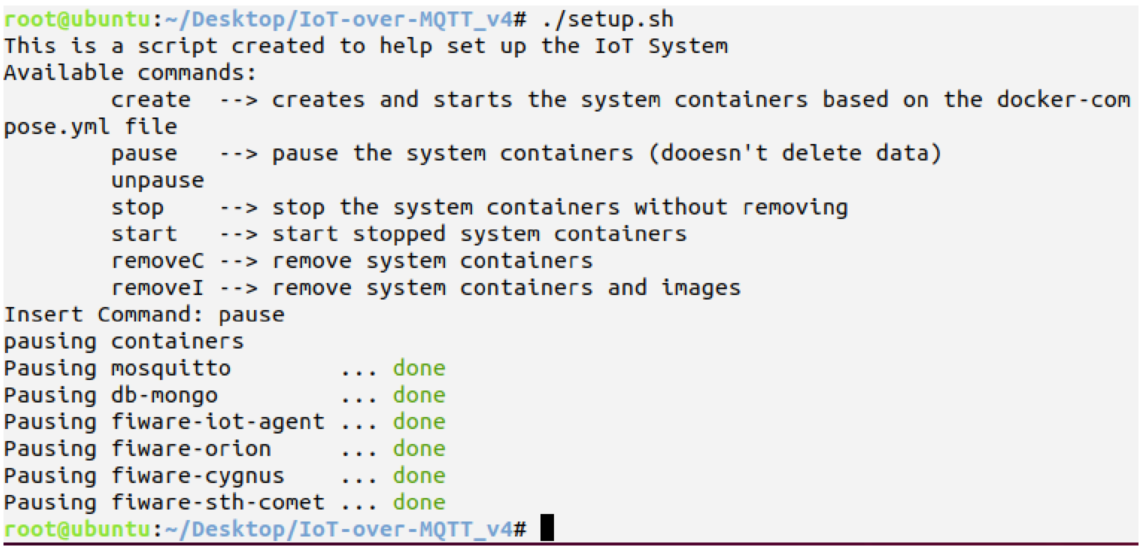


```
root@ubuntu:~# docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
mongo                3.6                 7efb1adc67ca        3 days ago          368MB
fiware/cygnus-ngsi   latest              0e0c481ea752        3 weeks ago         516MB
fiware/sth-comet     latest              45e60618b71e        3 weeks ago         303MB
fiware/orion         1.14.0              92459152faf9        2 months ago        272MB
fiware/iotagent-ul   1.6.0               3c32a7e0bef5        6 months ago        375MB
eclipse-mosquitto    latest              bd592a7a5bcf        7 months ago        4.38M
B
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE                        COMMAND                  CREATED
       STATUS              PORTS                                            NAMES
c5526336ae03        fiware/sth-comet             "/bin/sh -c bin/sth"     15 minutes ag
o      Up 15 minutes       0.0.0.0:8666->8666/tcp                           fiware-sth
-comet
d5cfc735750e        fiware/cygnus-ngsi:latest    "/cygnus-entrypoint.…"   15 minutes ag
o      Up 15 minutes       0.0.0.0:5050->5050/tcp, 0.0.0.0:5080->5080/tcp   fiware-cyg
nus
82d62c55eb70        fiware/orion:1.14.0          "/usr/bin/contextBro…"   15 minutes ag
o      Up 15 minutes       0.0.0.0:1026->1026/tcp                           fiware-ori
on
93e8c6aefaa7        fiware/iotagent-ul:1.6.0     "/bin/sh -c 'bin/iot…"   15 minutes ag
o      Up 15 minutes       0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp   fiware-iot
-agent
c6681fe33183        mongo:3.6                    "docker-entrypoint.s…"   15 minutes ag
o      Up 15 minutes       0.0.0.0:27017->27017/tcp                         db-mongo
741c99ba9d81        eclipse-mosquitto            "/docker-entrypoint.…"   15 minutes ag
o      Up 15 minutes       0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp   mosquitto
root@ubuntu:~#
```

Figure H.5 Docker Images and Containers After the Script Execution

“pause” Command

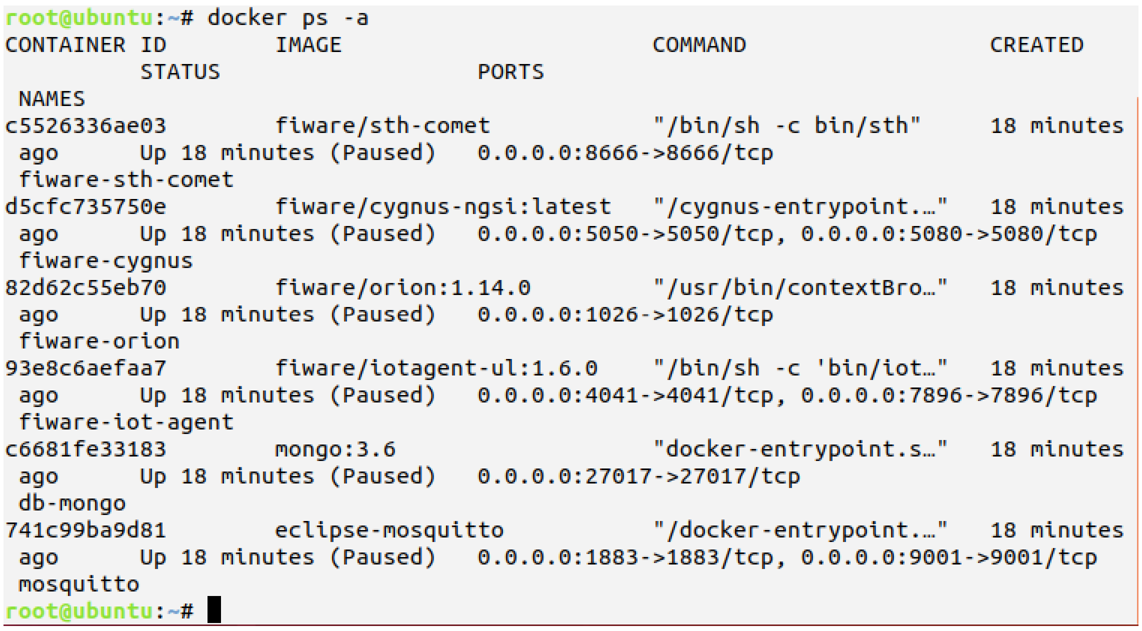
Figure H.5 displays the script execution output, where it is observed that the Containers where paused.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-com
pose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: pause
pausing containers
Pausing mosquitto        ... done
Pausing db-mongo         ... done
Pausing fiware-iot-agent ... done
Pausing fiware-orion     ... done
Pausing fiware-cygnus    ... done
Pausing fiware-sth-comet ... done
root@ubuntu:~/Desktop/IoT-over-MQTT_v4#
```

Figure H.6 "pause" Command Output

Figure H.6 shows the Docker Containers after the script execution, confirming that the Containers where indeed paused.

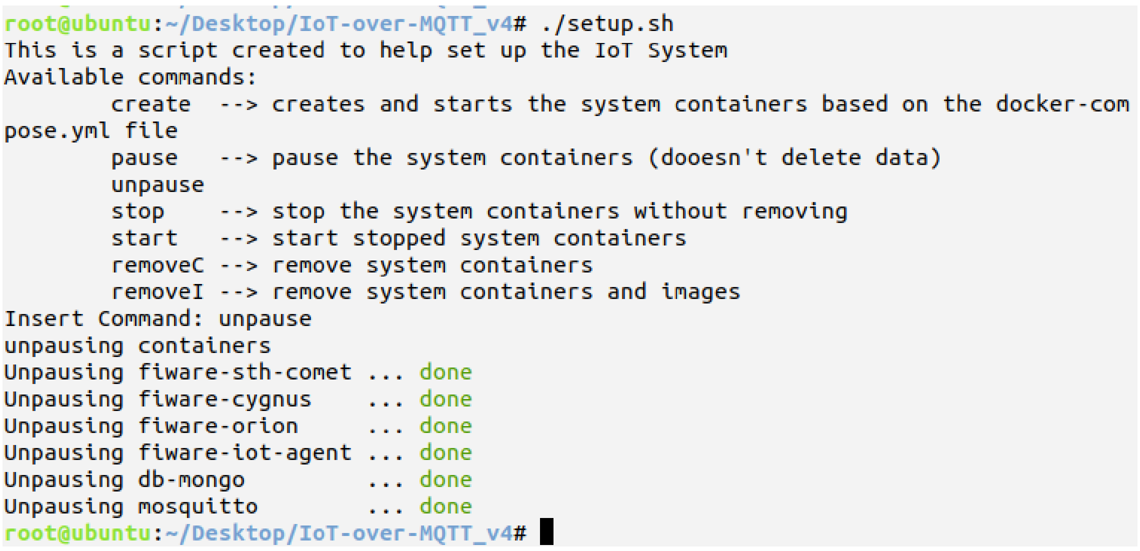


```
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE                       COMMAND                  CREATED
          STATUS                      PORTS
 NAMES
c5526336ae03        fiware/sth-comet            "/bin/sh -c bin/sth"     18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:8666->8666/tcp
 fiware-sth-comet
d5cfc735750e        fiware/cygnus-ngsi:latest   "/cygnus-entrypoint.…"   18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:5050->5050/tcp, 0.0.0.0:5080->5080/tcp
 fiware-cygnus
82d62c55eb70        fiware/orion:1.14.0         "/usr/bin/contextBro…"   18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:1026->1026/tcp
 fiware-orion
93e8c6aefaa7        fiware/iotagent-ul:1.6.0    "/bin/sh -c 'bin/iot…"   18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp
 fiware-iot-agent
c6681fe33183        mongo:3.6                   "docker-entrypoint.s…"   18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:27017->27017/tcp
 db-mongo
741c99ba9d81        eclipse-mosquitto           "/docker-entrypoint.…"   18 minutes
 ago       Up 18 minutes (Paused)    0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp
 mosquitto
root@ubuntu:~#
```

Figure H.7 Docker Containers After “pause” Command

“unpause” Command

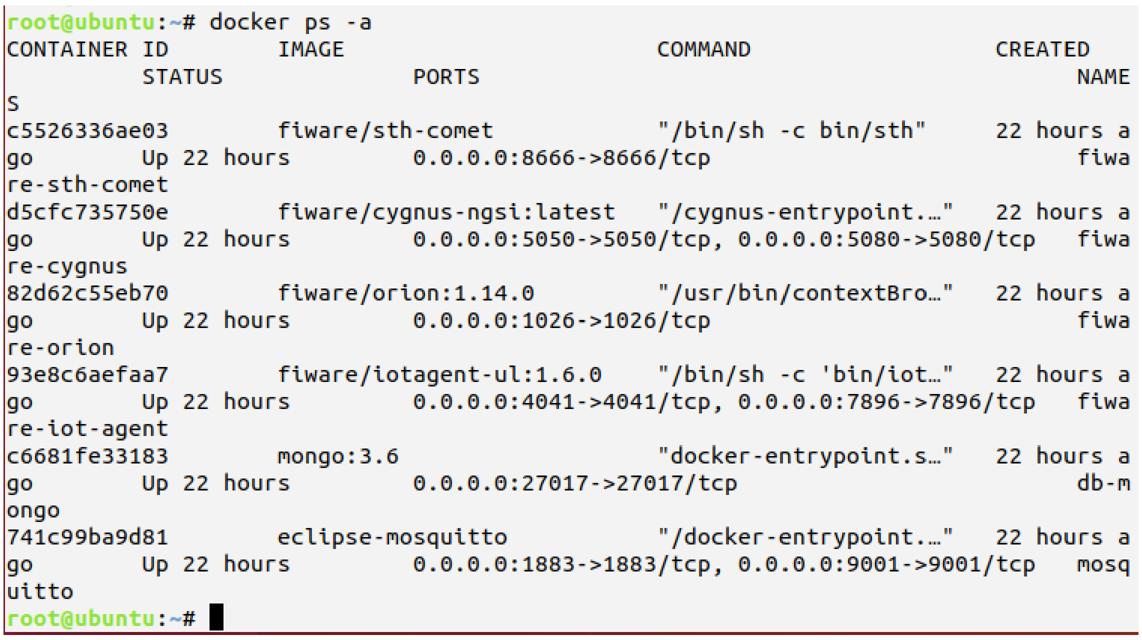
Figure H.7 displays the script execution output, where it is observed that the Containers where unpaused.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-com
pose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: unpause
unpausing containers
Unpausing fiware-sth-comet ... done
Unpausing fiware-cygnus    ... done
Unpausing fiware-orion     ... done
Unpausing fiware-iot-agent ... done
Unpausing db-mongo         ... done
Unpausing mosquitto        ... done
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# 
```

Figure H.8 "unpause" Command Output

Figure H.8 shows the Docker Containers after the script execution, confirming that the Containers where indeed unpaused.

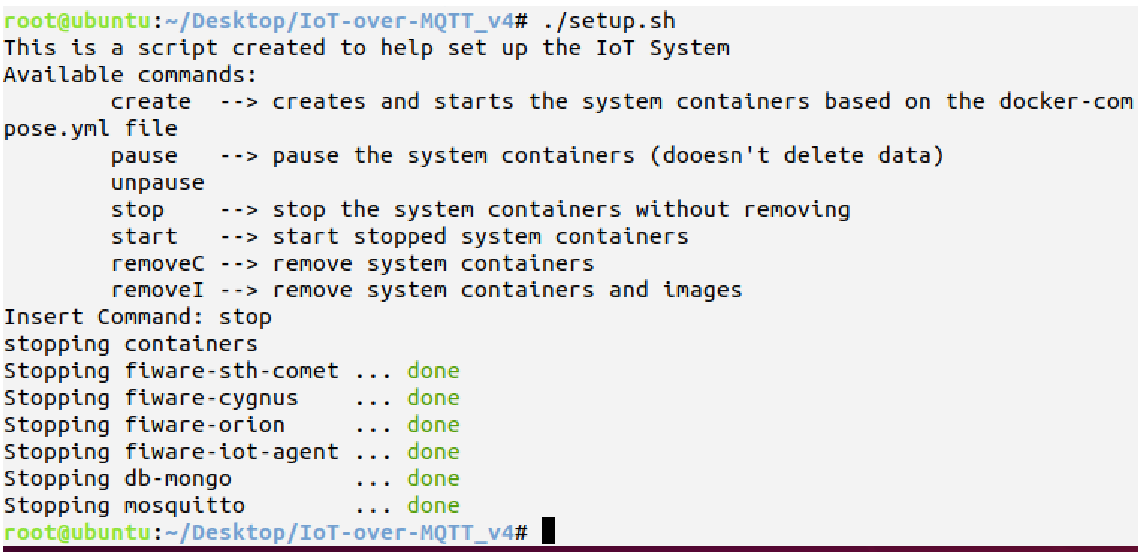


```
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED
          STATUS              PORTS                                            NAME
S
c5526336ae03        fiware/sth-comet               "/bin/sh -c bin/sth"     22 hours a
go        Up 22 hours         0.0.0.0:8666->8666/tcp                           fiwa
re-sth-comet
d5cfc735750e        fiware/cygnus-ngsi:latest      "/cygnus-entrypoint.…"   22 hours a
go        Up 22 hours         0.0.0.0:5050->5050/tcp, 0.0.0.0:5080->5080/tcp   fiwa
re-cygnus
82d62c55eb70        fiware/orion:1.14.0            "/usr/bin/contextBro…"   22 hours a
go        Up 22 hours         0.0.0.0:1026->1026/tcp                           fiwa
re-orion
93e8c6aefaa7        fiware/iotagent-ul:1.6.0       "/bin/sh -c 'bin/iot…"   22 hours a
go        Up 22 hours         0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp   fiwa
re-iot-agent
c6681fe33183        mongo:3.6                      "docker-entrypoint.s…"   22 hours a
go        Up 22 hours         0.0.0.0:27017->27017/tcp                         db-m
ongo
741c99ba9d81        eclipse-mosquitto              "/docker-entrypoint.…"   22 hours a
go        Up 22 hours         0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp   mosq
uitto
root@ubuntu:~# 
```

Figure H.9 Docker Container After "unpause" Command

“stop” Command

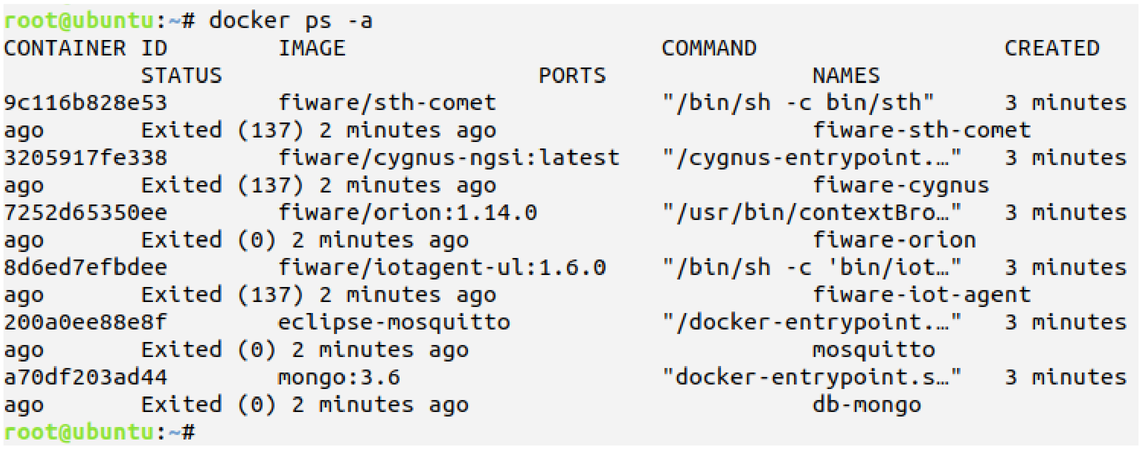
Figure H.9 displays the script execution output, where it is observed that the Containers where stopped.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-com
pose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: stop
stopping containers
Stopping fiware-sth-comet ... done
Stopping fiware-cygnus    ... done
Stopping fiware-orion     ... done
Stopping fiware-iot-agent ... done
Stopping db-mongo         ... done
Stopping mosquitto        ... done
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# 
```

Figure H.10 "stop" Command Output

Figure H.10 shows the Docker Containers after the script execution, confirming that the Containers where indeed stopped.

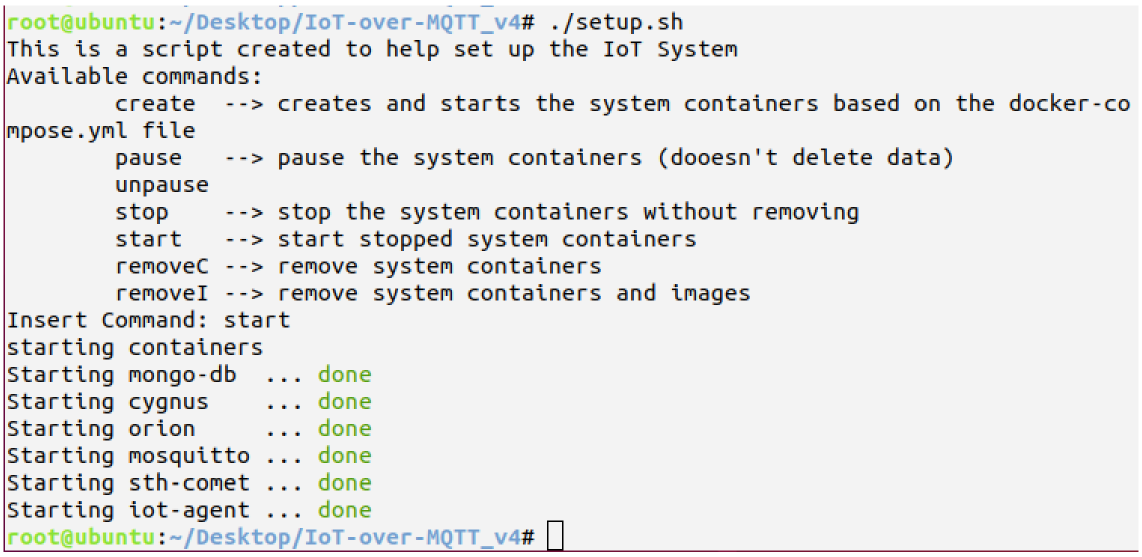


```
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE                         COMMAND                  CREATED
          STATUS                      PORTS               NAMES
9c116b828e53        fiware/sth-comet              "/bin/sh -c bin/sth"     3 minutes
ago       Exited (137) 2 minutes ago                      fiware-sth-comet
3205917fe338        fiware/cygnus-ngsi:latest     "/cygnus-entrypoint.…"   3 minutes
ago       Exited (137) 2 minutes ago                      fiware-cygnus
7252d65350ee        fiware/orion:1.14.0           "/usr/bin/contextBro…"   3 minutes
ago       Exited (0) 2 minutes ago                        fiware-orion
8d6ed7efbdee        fiware/iotagent-ul:1.6.0      "/bin/sh -c 'bin/iot…"   3 minutes
ago       Exited (137) 2 minutes ago                      fiware-iot-agent
200a0ee88e8f        eclipse-mosquitto             "/docker-entrypoint.…"   3 minutes
ago       Exited (0) 2 minutes ago                        mosquitto
a70df203ad44        mongo:3.6                     "docker-entrypoint.s…"   3 minutes
ago       Exited (0) 2 minutes ago                        db-mongo
root@ubuntu:~#
```

Figure H.11 Docker Containers After "stop" Command

“start” Command

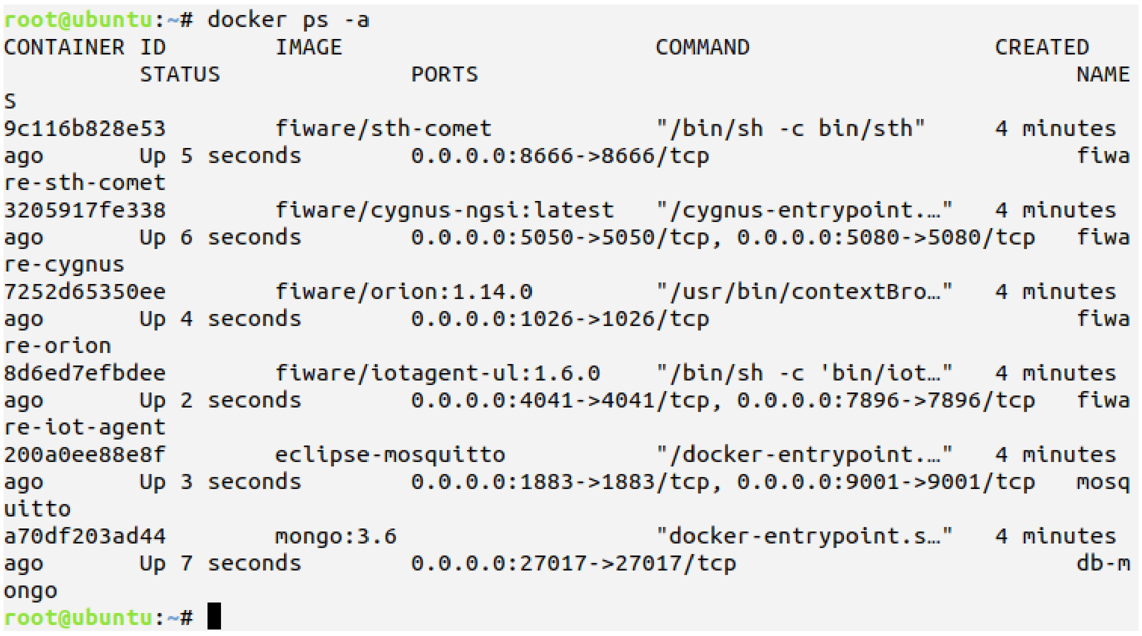
Figure H.11 displays the script execution output, where it is observed that the stopped Containers where started.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-co
mpose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: start
starting containers
Starting mongo-db  ... done
Starting cygnus    ... done
Starting orion     ... done
Starting mosquitto ... done
Starting sth-comet ... done
Starting iot-agent ... done
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# 
```

Figure H.12 "start" Command Output

Figure H.12 shows the Docker Containers after the script execution, confirming that the Containers where indeed started.

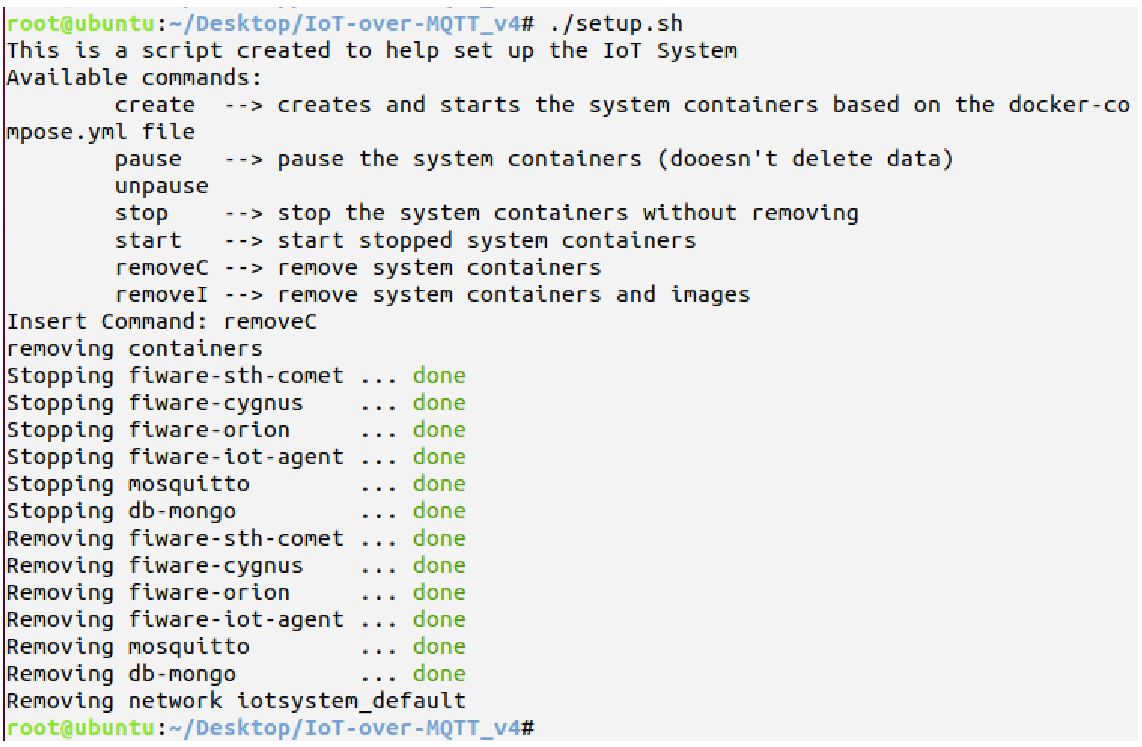


```
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE                        COMMAND                  CREATED
          STATUS              PORTS                                            NAME
S
9c116b828e53        fiware/sth-comet             "/bin/sh -c bin/sth"     4 minutes
ago         Up 5 seconds        0.0.0.0:8666->8666/tcp                           fiwa
re-sth-comet
3205917fe338        fiware/cygnus-ngsi:latest    "/cygnus-entrypoint.…"   4 minutes
ago         Up 6 seconds        0.0.0.0:5050->5050/tcp, 0.0.0.0:5080->5080/tcp   fiwa
re-cygnus
7252d65350ee        fiware/orion:1.14.0          "/usr/bin/contextBro…"   4 minutes
ago         Up 4 seconds        0.0.0.0:1026->1026/tcp                           fiwa
re-orion
8d6ed7efbdee        fiware/iotagent-ul:1.6.0     "/bin/sh -c 'bin/iot…"   4 minutes
ago         Up 2 seconds        0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp   fiwa
re-iot-agent
200a0ee88e8f        eclipse-mosquitto            "/docker-entrypoint.…"   4 minutes
ago         Up 3 seconds        0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp   mosq
uitto
a70df203ad44        mongo:3.6                    "docker-entrypoint.s…"   4 minutes
ago         Up 7 seconds        0.0.0.0:27017->27017/tcp                         db-m
ongo
root@ubuntu:~# 
```

Figure H.13 Docker Containers After "start" Command

“removeC” Command

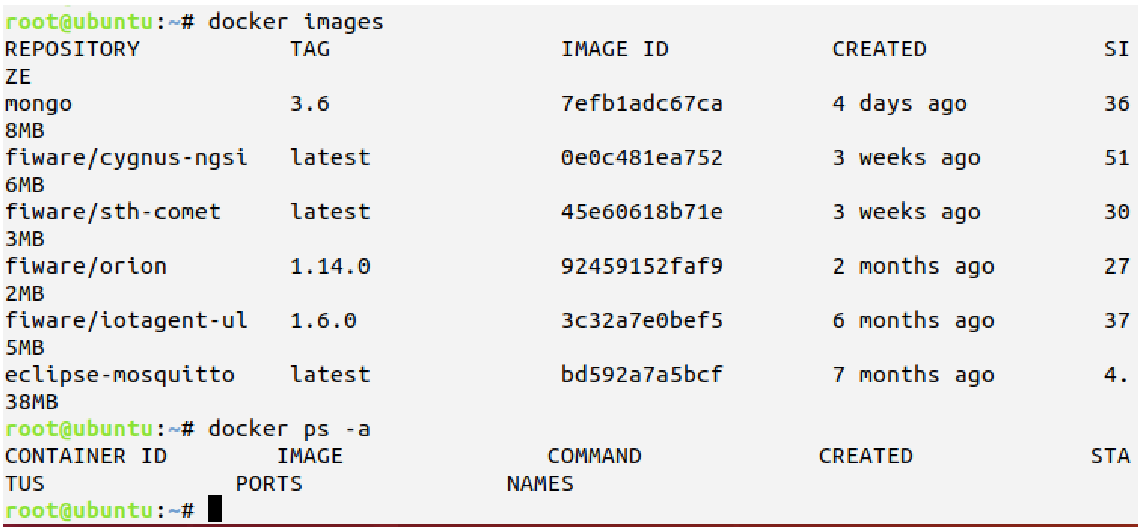
Figure H.13 displays the script execution output, where it is observed that the Containers where stopped and then removed.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-co
mpose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: removeC
removing containers
Stopping fiware-sth-comet ... done
Stopping fiware-cygnus    ... done
Stopping fiware-orion     ... done
Stopping fiware-iot-agent ... done
Stopping mosquitto        ... done
Stopping db-mongo         ... done
Removing fiware-sth-comet ... done
Removing fiware-cygnus    ... done
Removing fiware-orion     ... done
Removing fiware-iot-agent ... done
Removing mosquitto        ... done
Removing db-mongo         ... done
Removing network iotsystem_default
root@ubuntu:~/Desktop/IoT-over-MQTT_v4#
```

Figure H.14 "removeC" Command Output

Figure H.14 shows the Docker Images and Containers after the script execution, confirming that the Containers where indeed removed and that the Images remained.

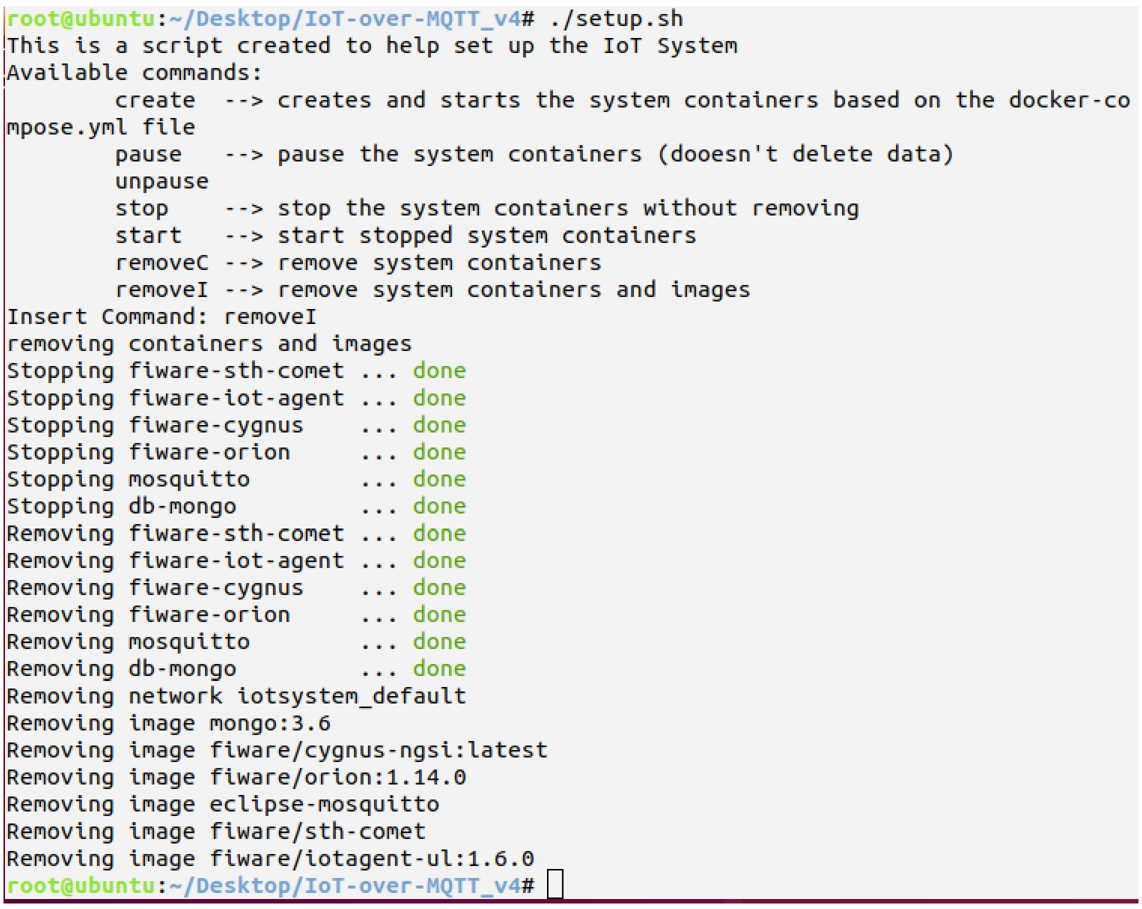


```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SI
ZE
mongo               3.6                 7efb1adc67ca        4 days ago          36
8MB
fiware/cygnus-ngsi  latest              0e0c481ea752        3 weeks ago         51
6MB
fiware/sth-comet    latest              45e60618b71e        3 weeks ago         30
3MB
fiware/orion        1.14.0              92459152faf9        2 months ago        27
2MB
fiware/iotagent-ul  1.6.0               3c32a7e0bef5        6 months ago        37
5MB
eclipse-mosquitto   latest              bd592a7a5bcf        7 months ago        4.
38MB
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STA
TUS                 PORTS               NAMES
root@ubuntu:~# 
```

Figure H.15 Docker Images and Containers After "removeC" Command

“removeI” Command

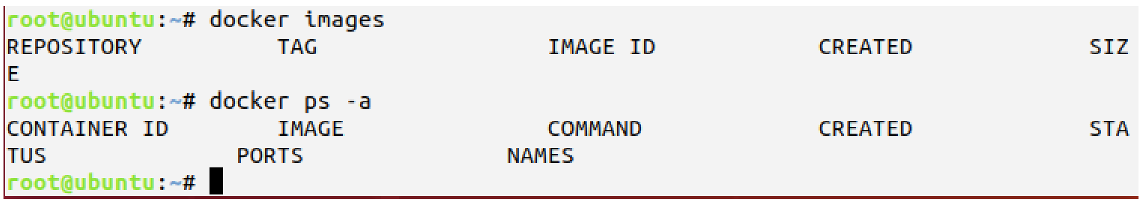
Figure H.15 displays the script execution output, where it is observed that the Containers where stopped and then removed, and the Images were also removed.



```
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# ./setup.sh
This is a script created to help set up the IoT System
Available commands:
        create  --> creates and starts the system containers based on the docker-co
mpose.yml file
        pause   --> pause the system containers (dooesn't delete data)
        unpause
        stop    --> stop the system containers without removing
        start   --> start stopped system containers
        removeC --> remove system containers
        removeI --> remove system containers and images
Insert Command: removeI
removing containers and images
Stopping fiware-sth-comet ... done
Stopping fiware-iot-agent ... done
Stopping fiware-cygnus    ... done
Stopping fiware-orion     ... done
Stopping mosquitto        ... done
Stopping db-mongo         ... done
Removing fiware-sth-comet ... done
Removing fiware-iot-agent ... done
Removing fiware-cygnus    ... done
Removing fiware-orion     ... done
Removing mosquitto        ... done
Removing db-mongo         ... done
Removing network iotsystem_default
Removing image mongo:3.6
Removing image fiware/cygnus-ngsi:latest
Removing image fiware/orion:1.14.0
Removing image eclipse-mosquitto
Removing image fiware/sth-comet
Removing image fiware/iotagent-ul:1.6.0
root@ubuntu:~/Desktop/IoT-over-MQTT_v4# 
```

Figure H.16 "removeI" Command Output

Figure H.16 shows the Docker Images and Containers after the script execution, confirming that the Images and Containers where indeed removed.



```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZ
E
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STA
TUS              PORTS               NAMES
root@ubuntu:~# 
```

Figure H.17 Docker Images and Containers After "removeI" Command

Annex I – Entities Creation Script

```
# Farm001

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Farm:001",
    "type": "Farm",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Av. das Forças Armadas 36",
            "addressRegion": "Lisbon",
            "addressLocality": "Lisbon",
            "postalCode": "1649-026"
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [38.7486, -9.1544]
        }
    },
    "name": {
        "type": "Text",
        "value": "GIGA Farm"
    }
}'

# Fields

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Field:001",
    "type": "Field",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [39.7489, -9.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Field A"
    },
    "area": {
        "type": "Integer",
        "value": "10"
    }
}'
```

```
curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Field:002",
    "type": "Field",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [39.7489, -9.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Field B"
    },
    "area": {
        "type": "Integer",
        "value": "5"
    }
}'

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Field:003",
    "type": "Field",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [39.7489, -9.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Field C"
    },
    "area": {
        "type": "Integer",
        "value": "5"
    }
}'

# Crops

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Crop:001",
    "type": "Crop",
    "name": {
        "type": "Text",
        "value": "Apples"
    }
}'

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Crop:002",
    "type": "Crop",
    "name": {
        "type": "Text",
        "value": "Tomatoes"
    }
}'

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Crop:003",
    "type": "Crop",
    "name": {
        "type": "Text",
        "value": "Corn"
    }
}'

# Well

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Well:001",
    "type": "Well",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [39.7489, -9.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Well One"
    },
    "depth": {
        "type": "Integer",
        "value": "10"
    }
}'
```

```
# Tank

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Tank:001",
    "type": "Tank",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [45.7489, -9.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Tank One"
    },
    "depth": {
        "type": "Integer",
        "value": "3"
    }
}'

# Borehole

curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '
{
    "id": "urn:ngsi-ld:Borehole:001",
    "type": "Borehole",
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [47.7489, -10.1534]
        }
    },
    "name": {
        "type": "Text",
        "value": "Borehole One"
    },
    "depth": {
        "type": "Integer",
        "value": "20"
    }
}'
```

Annex J – Entities Association Script

```
# Associations

curl -iX POST \
  'http://localhost:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: farmOne' \
  -d '{
  "actionType":"APPEND",
  "entities":[
    {
      "id":"urn:ngsi-ld:Field:001", "type":"Field",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refTank": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Tank:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Field:002", "type":"Field",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refWell": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Well:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Field:003", "type":"Field",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refwell": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Well:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Crop:001", "type":"Crop",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refField": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Field:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Crop:002", "type":"Crop",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refField": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Field:002"
      }
    },
    {
      "id":"urn:ngsi-ld:Crop:003", "type":"Crop",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      },
      "refField": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Field:003"
      }
    },
    {
      "id":"urn:ngsi-ld:Well:001", "type":"Well",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      }
    },
    {
      "id":"urn:ngsi-ld:Tank:001", "type":"Tank",
      "refFarm": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:Farm:001"
      }
    }
  ]
}'
```

Annex K – Entities Modification Scripts

```
# Borehole - Original Entity

#curl -iX POST \
#  'http://localhost:1026/v2/entities' \
#  -H 'Content-Type: application/json' \
#  -H 'fiware-service: farmOne' \
#  -d '
#{
#    "id": "urn:ngsi-ld:Borehole:001",
#    "type": "Borehole",
#    "location": {
#        "type": "geo:json",
#        "value": {
#             "type": "Point",
#             "coordinates": [47.7489, -10.1534]
#        }
#    },
#    "name": {
#        "type": "Text",
#        "value": "Borehole One"
#    },
#    "depth": {
#        "type": "Integer",
#         "value": "20"
#     }
#}'
```

Script “3_use_case_entities_modification_(1)_v1.sh”

```
#Overwrite a single attribute value (depth)

curl -iX PUT \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Borehole:001/attrs/depth/value' \
  --header 'Content-Type: text/plain' \
  --header 'fiware-service: farmOne' \
  --data 25
```

Script “3_use_case_entities_modification_(2)_v1.sh”

```
#Overwrite multiple attributes

curl -iX PATCH \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Borehole:001/attrs' \
  --header 'Content-Type: application/json' \
  --header 'fiware-service: farmOne' \
  --data ' {
      "name":{"type":"Text", "value": "Top Borehole"},
      "depth":{"type":"Integer", "value": "30"}
}'
```

Annex L – Entities Removal Scripts

Script “4_use_case_entities_removal_(1)_v1.sh”

```
# Borehole - remove attribute

curl -iX DELETE 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Borehole:001/attrs/depth' \
  -H 'fiware-service: farmOne'
```

Script “4_use_case_entities_removal_(2)_v1.sh”

```
# Borehole - remove entity

curl -iX DELETE 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Borehole:001' \
  -H 'fiware-service: farmOne'
```

Annex M – Service Group Provisioning Script

```
#field A service groups

curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59a",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'

curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59b",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'

#tank service groups

curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/sensors' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59c",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'

curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/actuators' \
  -d '{
```

```
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59d",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'
```

Annex N – Sensors Provisioning Script

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
 "devices": [
   {
     "device_id":   "weather001",
     "entity_name": "urn:ngsd-ld:Weather:001",
     "entity_type": "Weather",
     "protocol":    "PDI-IoTA-UltraLight",
     "transport":   "MQTT",
     "timezone":    "Europe/Lisbon",
     "attributes": [
       { "object_id": "h", "name": "humidity", "type": "percentage"
},
       { "object_id": "t", "name": "temperature", "type": "degrees"
},
       { "object_id": "i", "name": "heatIndex", "type": "degrees" }
     ],
     "static_attributes": [
       {"name":"refField", "type": "Relationship", "value":
"urn:ngsi-ld:Field:001"},
       {"name":"location", "type": "geo:point", "value": "40.392, -
3.759"}
     ]
   }
 ]
}'

curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
 "devices": [
   {
     "device_id":   "earthHum001",
     "entity_name": "urn:ngsd-ld:EarthHum:001",
     "entity_type": "EarthHum",
     "protocol":    "PDI-IoTA-UltraLight",
     "transport":   "MQTT",
     "timezone":    "Europe/Lisbon",
     "attributes": [
       {"object_id": "h", "name": "humidity", "type": "percentage" }
     ],
     "static_attributes": [
       {"name":"refField", "type": "Relationship", "value":
"urn:ngsi-ld:Field:001"},
       {"name":"refCrop", "type": "Relationship","value": "urn:ngsi-
ld:Crop:001"},
       {"name":"refAppleTree", "type": "Relationship","value":
"urn:ngsi-ld:AppleTree:001"},
       {"name":"location", "type": "geo:point", "value": "40.392, -
3.759"}
     ]
   }
 ]
}'

curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/sensors' \
  -d '{
 "devices": [
   {
     "device_id":   "waterLevel001",
     "entity_name": "urn:ngsd-ld:WaterLevel:001",
     "entity_type": "WaterLevel",
     "protocol":    "PDI-IoTA-UltraLight",
     "transport":   "MQTT",
     "timezone":    "Europe/Lisbon",
     "attributes": [
       { "object_id": "l", "name": "level", "type": "Double" }
     ],
     "static_attributes": [
       { "name":"refTank", "type": "Relationship", "value":
"urn:ngsi-ld:Tank:001"},
       { "name":"location", "type": "geo:point", "value": "40.392, -
3.759"}
     ]
   }
 ]
}'
```

Annex O – Actuators Provisioning Script

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
  "devices": [
    {
      "device_id": "valve001",
      "entity_name": "urn:ngsi-ld:Valve:001",
      "entity_type": "Valve",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "MQTT",
      "timezone": "Europe/Lisbon",
      "commands": [
        { "name": "open", "type": "command" },
        { "name": "close", "type": "command" }
       ],
       "static_attributes": [
         {"name":"refField", "type": "Relationship","value":
"urn:ngsi-ld:Field:001"},
         {"name":"refAppleTree", "type": "Relationship","value":
"urn:ngsi-ld:AppleTree:001"},
         {"name":"location", "type": "geo:point", "value": "40.392,
-3.759"}
      ]
    }
  ]
}'

curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/actuators' \
  -d '{
  "devices": [
    {
      "device_id": "valve002",
      "entity_name": "urn:ngsi-ld:Valve:002",
      "entity_type": "Valve",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "MQTT",
      "timezone": "Europe/Lisbon",
      "commands": [
        { "name": "open", "type": "command" },
        { "name": "close", "type": "command" }
       ],
       "static_attributes": [
         {"name":"refTank", "type": "Relationship","value":
"urn:ngsi-ld:Tank:001"},
         { "name":"location", "type": "geo:point", "value": "40.392,
-3.759"}
      ]
    }
  ]
}'
```

Annex P – Enabling Context Broker Commands Script

```
curl -iX POST \
  'http://localhost:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
  "description": "Valve Commands",
  "dataProvided": {
    "entities": [
      { "id": "urn:ngsi-ld:Valve:001", "type": "Valve" }
    ],
    "attrs": ["open", "close"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'

curl -iX POST \
  'http://localhost:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/actuators' \
  -d '{
  "description": "Valve Commands",
  "dataProvided": {
    "entities": [
      { "id": "urn:ngsi-ld:Valve:002", "type": "Valve" }
    ],
    "attrs": ["open", "close"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'
```

Annex Q – Code for Testing the DHT22 Sensor

```
// Example testing sketch for various DHT humidity/temperature
sensors

// Written by ladyada, public domain

#include "DHT.h"
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define DHTPIN 2     // what digital pin we're connected to

// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due
connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the
sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third
parameter to
// tweak the timings for faster processors.  This parameter is no
longer needed
// as the current DHT reading algorithm adjusts itself to work on
faster procs.
```

```
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHT22 test!");

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very
slow sensor)

  float h = dht.readHumidity();

  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();

  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print("%  ");

  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print("*C  ");

  Serial.print("Heat index: ");
  Serial.print(hic);
  Serial.println("*C  ");
}
```

Annex R – Code for Testing the Ultrasonic Sensor

```
/*
Test code for the Arduino Uno
Written by Tom Bonar for testing
Sensors being used for this code are the MB12X0 from MaxBotix
*/
const int pwmPin = 2; //GPIO2 (PWM pin 4)

long sensor1, cm;

void setup () {
  Serial.begin(9600);
  pinMode(pwmPin, INPUT);
}

void read_sensor(){
  sensor1 = pulseIn(pwmPin, HIGH);
  cm = sensor1/58;
}

void loop () {
  read_sensor();
  printall();
  delay(1000);
}

void printall(){
  Serial.print("S1");
  Serial.print(" = ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();
}
```

Annex S – Code for Testing the Earth-Humidity Sensor

```
//sensor dgital output to read the soil mosture, returns 1 above a
threshould defined by the potenciometer, otherwise returns 0

//int DSIGNAL = 2; //GPIO2 --> pin D4
//
//void setup() {
//  Serial.begin(9600);
//  pinMode(DSIGNAL, INPUT);
//}
//
//void loop() {
//  int DsignalState = digitalRead(DSIGNAL);
//  Serial.println(DsignalState);
//  delay(1000);
//}

//sensor analog output to read the soil mosture

int ASignal = A0; //ADC pin

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(ASignal);
  Serial.print("ADC reading: ");
  Serial.println(sensorValue);
  int valuePercentage = map(sensorValue, 0, 1024, 100, 0); //convert
the ADC values to percentage
  Serial.print("Humidity Percentage: ");
  Serial.println(valuePercentage);
  delay(1000);
}
```

Annex T – Code for Sensing Measurements from DHT22 Sensor to the IoT Sensor

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

const char *ssid = "*****";   // cannot be longer than 32 characters!
const char *pass = "*****";   //

// Update these with values suitable for your network.
IPAddress server(192, 168, 1, 6);

#define deviceId "weather001"
#define outTopic "/4jggokgpepnvsb2uv4s40d59a/weather001/attrs" //
/apikey/deviceID/attrs

#define DHTPIN 2     // what digital pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor.

WiFiClient wclient;
PubSubClient client(wclient, server);


void setup() {
  // Setup console
  //Serial.begin(115200);
  delay(10);
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    //Serial.print("Connecting to ");
    //Serial.print(ssid);
    //Serial.println("...");
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED)
      return;
    //Serial.println("WiFi connected");
  }

  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      if (client.connect(deviceId)) {
        //client.set_callback(callback);
      }
    }

    if (client.connected())
      client.loop();

    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    // Compute heat index in Celsius (isFahreheit = false)
    float hic = dht.computeHeatIndex(t, h, false);
    String ul = "h|" + String(h) + "|t|" + String(t) + "|i|" +
String(hic);

    //Serial.print("Publish message: ");
    //Serial.println(ul);

    //client.publish(outTopic, ul); //publish with QoS 0
    //client.publish(MQTT::Publish(outTopic, ul).set_qos(1));
//publish with QoS 1
    client.publish(MQTT::Publish(outTopic, ul).set_qos(2));
//publish with QoS 2

    delay(30000); //delay 30 seconds between messages transmissions
  }
}


void callback(const MQTT::Publish& pub) { //only when the device
susbcribes to messages (actuators)
  // handle message arrived
}
```

Annex U – Code for Sensing Measurements from Ultrasonic Sensor to the IoT System

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char *ssid = "*****";   // cannot be longer than 32 characters!
const char *pass = "*****";   //

// Update these with values suitable for your network.
IPAddress server(192, 168, 1, 6);

#define deviceId "waterLevel001"
#define outTopic "/4jggokgpepnvsb2uv4s40d59c/waterLevel001/attrs" //
/apikey/deviceID/attrs

const int pwmPin = 2; //GPIO2 (PWM pin 4)
const int tankDepth = 322; //tank is 3 meters deep + 22 cm of
minimum reading distance of the sensor (virtual 0)
long sensor1, cm;

WiFiClient wclient;
PubSubClient client(wclient, server);


void setup() {
  // Setup console
  //Serial.begin(115200);
  delay(10);
  pinMode(pwmPin, INPUT);
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    //Serial.print("Connecting to ");
    //Serial.print(ssid);
    //Serial.println("...");
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED)
      return;
    //Serial.println("WiFi connected");
  }

  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      if (client.connect(deviceId)) {
        //client.set_callback(callback);
      }
    }

    if (client.connected())
      client.loop();

    read_sensor();
    //printall();

    double waterLevel = tankDepth - cm;
    String ul = "l|" + String(waterLevel);

    //Serial.print("Publish message: ");
    //Serial.println(ul);

    //client.publish(outTopic, ul); //publish with QoS 0
    //client.publish(MQTT::Publish(outTopic, ul).set_qos(1));
//publish with QoS 1
    client.publish(MQTT::Publish(outTopic, ul).set_qos(2));
//publish with QoS 2

    delay(30000); //delay 30 seconds between messages transmissions
  }
}

void read_sensor() {
  sensor1 = pulseIn(pwmPin, HIGH);
  cm = sensor1 / 58;
}

//void printall() {
//  Serial.print("S1");
//  Serial.print(" = ");
//  Serial.print(cm);
//  Serial.print("cm");
//  Serial.println();
//}

void callback(const MQTT::Publish& pub) { //only when the device
susbcribes to messages (actuators)
  // handle message arrived
}
```

Annex V – Code for Sensing Measurements from Earth-Humidity Sensor to the IoT System

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char *ssid = "*****";   // cannot be longer than 32 characters!
const char *pass = "*****";   //

// Update these with values suitable for your network.
IPAddress server(192, 168, 1, 6);

#define deviceId "EarthHum001"
#define outTopic "/4jggokgpepnvsb2uv4s40d59a/EarthHum001/attrs" //
/apikey/deviceID/attrs

WiFiClient wclient;
PubSubClient client(wclient, server);

int ASignal = A0; //ADC pin

void setup() {
  // Setup console
  //Serial.begin(115200);
  delay(10);
}


void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    //Serial.print("Connecting to ");
    //Serial.print(ssid);
    //Serial.println("...");
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED)
      return;
    //Serial.println("WiFi connected");
  }

  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      if (client.connect(deviceId)) {
        //client.set_callback(callback);
      }
    }

    if (client.connected())
      client.loop();

    int sensorValue = analogRead(ASignal);
    //Serial.print("ADC reading: ");
    //Serial.println(sensorValue);
    int valuePercentage = map(sensorValue, 0, 1024, 100, 0);
//convert the ADC values to percentage

    String ul = "h|" + String(valuePercentage);

    //Serial.print("Publish message: ");
    //Serial.println(ul);

    //client.publish(outTopic, ul); //publish with QoS 0
    //client.publish(MQTT::Publish(outTopic, ul).set_qos(1));
//publish with QoS 1
    client.publish(MQTT::Publish(outTopic, ul).set_qos(2));
//publish with QoS 2

    delay(30000); //delay 30 seconds between messages transmissions
  }
}

void callback(const MQTT::Publish& pub) { //only when the device
susbcribes to messages (actuators)
  // handle message arrived
}
```

Annex W – Code for Receiving Commands from the IoT System

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char *ssid = "*****";   // cannot be longer than 32 characters!
const char *pass = "*****";   //

// Update these with values suitable for your network.
IPAddress server(192, 168, 1, 6);

WiFiClient wclient;
PubSubClient client(wclient, server);

#define deviceId "valve001"

#define outTopic "/4jggokgpepnvsb2uv4s40d59b/valve001/attrs"
#define inTopic "/4jggokgpepnvsb2uv4s40d59b/valve001/attrs"


void setup() {
  // Setup console
  //Serial.begin(115200);
  delay(10);
  pinMode(BUILTIN_LED, OUTPUT); // Initialize the BUILTIN_LED pin as
an output
  digitalWrite(BUILTIN_LED, HIGH); //turn LED OFF
}


void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    //Serial.print("Connecting to ");
    //Serial.print(ssid);
    //Serial.println("...");
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED)
      return;

    //Serial.println("WiFi connected");
  }
  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      if (client.connect(deviceId)) {
        client.set_callback(callback);
        client.subscribe(inTopic);
      }
    }

    if (client.connected())
      client.loop();
  }
}


// Callback function | process the received message
void callback(const MQTT::Publish& pub) {
  // In order to republish this payload, a copy must be made
  // as the orignal payload buffer will be overwritten whilst
  // constructing the PUBLISH packet.

  //Serial.print("Message Received: ");
  //Serial.println(pub.payload_string());

  String msg = pub.payload_string();

  if (String(msg) == "valve001@open|") { //execute command
    digitalWrite(BUILTIN_LED, LOW); //LED ON

    String reply = String(msg) + "Opened ok";
    client.publish(MQTT::Publish(outTopic, reply).set_qos(2));
//reply
  }
  if (String(msg) == "valve001@close|") { //execute command
    digitalWrite(BUILTIN_LED, HIGH); //LED OFF

    String reply = String(msg) + "Closed ok";
    client.publish(MQTT::Publish(outTopic, reply).set_qos(2));
//reply
  }
}
```

Annex X – Available Commands to Control Actuators

```
#valve001 (fieldA) commands

curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:001/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
  "open": {
      "type" : "command",
      "value" : ""
  }
}'

curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:001/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/actuators' \
  -d '{
  "close": {
      "type" : "command",
      "value" : ""
  }
}'


#valve002 (tank) commands

curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:002/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/actuators' \
  -d '{
  "open": {
      "type" : "command",
      "value" : ""
  }
}'

curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Valve:002/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/actuators' \
  -d '{
  "close": {
      "type" : "command",
      "value" : ""
  }
}'
```

Annex Y – Subscriptions Script

```
curl -iX POST \
  'http://localhost:1026/v2/subscriptions' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
  "description": "Notify me of temp higher than 30*C in all weather
sensors in field A",
  "subject": {
    "entities": [{"idPattern": ".*","type": "Weather"}],
    "condition": {
      "attrs": ["temperature"],
      "expression": {
        "q": "temperature>30"
      }
    }
  },
  "notification": {
    "http": {
      "url": "http://192.168.1.6:1028/accumulate"
    },
    "attrs": [
      "temperature"
    ]
  }
}'
```

Annex Z – Data Persistence Script

```
# weather sensors

curl -iX POST \
  'http://localhost:1026/v2/subscriptions' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
  "description": "Notify Cygnus of all weather sensors attrs
change",
  "subject": {
    "entities": [
      {
        "idPattern": "Weather.*"
      }
    ],
    "condition": {
      "attrs": [
        "humidity",
        "temperature",
        "heatIndex"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://cygnus:5050/notify"
    },
    "attrs": [
      "humidity",
      "temperature",
      "heatIndex"
    ],
    "attrsFormat": "legacy"
  }
}'

# earth humidity sensors

curl -iX POST \
  'http://localhost:1026/v2/subscriptions' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /fieldA/sensors' \
  -d '{
  "description": "Notify Cygnus of all earth humidity sensors attrs
change",
  "subject": {
    "entities": [
      {
        "idPattern": "EarthHum.*"
      }
    ],
    "condition": {
      "attrs": [
        "humidity"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://cygnus:5050/notify"
    },
    "attrs": [
      "humidity"
    ],
    "attrsFormat": "legacy"
  }
}'

# waterlevel sensors

curl -iX POST \
  'http://localhost:1026/v2/subscriptions' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /tank/sensors' \
  -d '{
  "description": "Notify Cygnus of all water level sensors attrs
change",
  "subject": {
    "entities": [
      {
        "idPattern": "WaterLevel.*"
      }
    ],
    "condition": {
      "attrs": [
        "level"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://cygnus:5050/notify"
    },
    "attrs": [
      "level"
    ],
    "attrsFormat": "legacy"
  }
}'
```

Annex AA – Paper

Universal Internet of Things System Powered by FIWARE

Diogo Lopes
Department of Information Science and Technology
ISCTE-University Institute of Lisbon
Lisbon, Portugal
Diogo_Rodrigues@iscte-iul.pt

Pedro Sebastião
Department of Information Science and Technology
ISCTE-University Institute of Lisbon
Instituto de Telecomunicações
Lisbon, Portugal
Pedro.Sebastiao@iscte-iul.pt

Abstract— **Internet of Things has grown exponentially in recent years and will continue to grow for some time, with ever more IoT devices available, there are also increasingly systems and platforms that use and support these devices, thus providing the possibility to view information by these collected or control them through a graphical interface, which can be a website or an application.**

Due to the expansion of the Internet market of Things resulting from a wide variety of devices and systems from different manufacturers it is difficult to find systems that are compatible with all or several devices from different manufacturers, since many use proprietary communication protocols. This dissertation aims at the development of a universal IoT system using the FIWARE platform, promoted by the European Commission, which allows the use of the modular components that make up this platform to develop the intended universal system.

A set of microcontrollers coupled to various sensors and actuators will be used to test the system and to verify the proper functioning of the same and each FIWARE component used, which will communicate with the system transmitting the collected data or receiving commands in the case of the actuators.

Keywords— Internet of Things; FIWARE; Microcontroller; Sensor; Actuator

I. INTRODUCTION (*HEADING 1*)

The Internet of Things is becoming more and more popular, transition from only being know and used in the industry to the general people, which are becoming ever more dependent on the new IoT devices and services that come out almost every day. For the general people these devices can transform their way of live, making tasks easier or even provide constant health monitoring, and all appears to just work like a miracle, it is possible to connect these IoT devices to a computer or smartphone and control them from there and visualize the data collected by them. However, in reality, things are more complex than that, it is necessary to have a whole system behind these devices and most of all they all have to be able to communicate with each other, which can be done directly or through the Internet.

To capitalize on the growth of the Internet of Things, the European Commission promoted and created FIWARE, an opensource smart solution platform, with the aim of bringing the benefits of the Internet of Things and the Internet to everyone. This is done by allowing everyone to use the FIWARE technologies to develop new smart solutions, easing the creation of new and innovative services and products before inexistent. Nowadays FIWARE is an independent foundation its community, being general people or enterprises, is growing year after year making FIWARE increasingly know and adopted by new people and business.

A. Objectives

The main objective of this project is to develop an Universal Internet of Things System Powered by FIWARE, which as the name indicates it implies the implementation of the available FIWARE technologies and some other complementary technologies as necessary, to create an IoT System which can be used with an array of different devices, sensors and actuators.

B. Document Sctucture

This article has five more chapters:

- FIWARE, a Bit of History;
- Powered by FIWARE;
- Universal IoT System Powered by FIWARE;
- Future Work: Security Implementation;
- Conclusions.

II. FIWARE, A BIT OF HISTORY

In 2011, the Internet had almost two billion users, The European Commission launched a €300 million Future Internet Public Private Partnership (FI-PPP) with the objective of increasing and sharing the social and economic benefits of the future Internet with consumers, citizens, private and public sectors [1]. The FI-PPP developed FIWARE, which combined the best existing technologies to create an opensource platform of components that could be used to develop smart applications [1].

In autumn 2016, four big companies, Atos, Engineering, Orange and Telefonica launched the FIWARE Foundation, an open body within the FIWARE Community, with the intent of promoting, augmenting, protecting and validating the FIWARE brand and its technologies (FIWARE Platform) [1] [2].

III. POWERED BY FIWARE

The FIWARE Platform is a curated framework of opensource components, named Generic Enablers (GEs), which can be combined with other third-party platform components to hasten the development of smart solutions [3].

In every smart solution it is essential to gather and manage context information, process it and inform external actors, allowing them to actuate and so change or enrich the current context. The FIWARE Context Broker component is the core constituent of any "Powered by FIWARE" solution, as it enables the system to update and access the current state of context [3].

As the core, the Context Broker is in turn surrounded by additional components, as shown in Figure 1, which can supply context data from various sources (e.g., a Customer Relationship Management (CRM) system, social networks, mobile apps, IoT sensors), support to data processing, analysis and visualization, or adding support to data access control, publication or monetization [3].

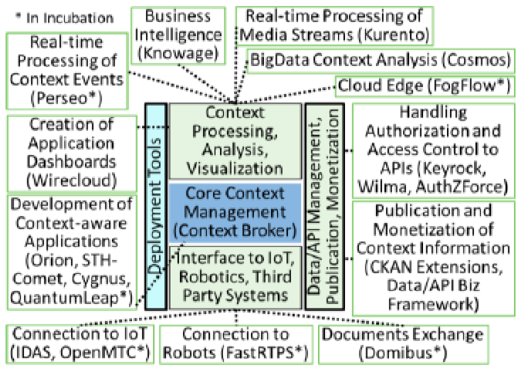

* In Incubation
Business Intelligence (Knowage)
Real-time Processing of Media Streams (Kurento)
Real-time Processing of Context Events (Perseo*)
BigData Context Analysis (Cosmos)
Cloud Edge (FogFlow*)
Creation of Application Dashboards (Wirecloud)
Context Processing, Analysis, Visualization
Handling Authorization and Access Control to APIs (Keyrock, Wilma, AuthZForce)
Deployment Tools
Core Context Management (Context Broker)
Data/API Management, Publication, Monetization
Development of Context-aware Applications (Orion, STH-Comet, Cygnus, QuantumLeap*)
Interface to IoT, Robotics, Third Party Systems
Publication and Monetization of Context Information (CKAN Extensions, Data/API Biz Framework)
Connection to IoT (IDAS, OpenMTC*)
Connection to Robots (FastRTPS*)
Documents Exchange (Domibus*)

Fig. 1. *FIWARE Generic Enablers (Based on Sources: [6] [7])*

All communications between applications (frontend) or platform components and the Context Broker (together form the backend) are done with the use of the FIWARE NGSIv2 RESTful API [4], a simple and powerful open standard that in the future will align with the ETSI NGSI-LD [5] specifications that are based and an evolution of the former, and are currently available for public review [3].

The open standard characteristic of the FIWARE NGSI API allows developers to port their applications across different "Powered by FIWARE" platforms and a guarantee of a stable framework for future development [3]. Also, additional functionalities can easily be added to a solution by using FIWARE or third-party components that comply to the FIWARE NGSI API. Since all components comply to the same API, integration is simplified as all components use the same standard interface, eliminating vendor lock-in [3]. The use of FIWARE also allows for rearchitecting solutions according to the user or business needs, as all FIWARE architectures are modular due to being made up of independent components [3].

IV. Universal IoT System Powered by FIWARE

As the name implies, a universal IoT system must be able to communicate with an array of devices regardless of the communication protocol used by these devices. The use of the FIWARE enables this, as the protocols used by the devices are converted by the IDAS component to the NGSI API which is used internally and common to all FIWARE components.

However, while the universality is the focus of this work, the core of every FIWARE powered system is the context management, which is done by using the Orion Context Broker, and extended using three other components [6] [17]:

- The **Orion Context Broker Generic Enabler**, which as the core component, allows the management of context information in a highly decentralized and large-scale manner, and provides the FIWARE NGSIv2 RESTful API, enabling updates, queries or subscriptions to changes on context information saved on a MongoDB database. However, as this GE only holds the latest information about the current context, the following components are used to save context information permanently, allowing the visualization of a context history;
- The **STH-Comet Generic Enabler** enables storing a Short-Term History of context data (typically months) on MongoDB;
- The **Cygnus Generic Enabler** enables managing the history of context, created as a stream of data which can be injected into several data sinks, including some of the most popular databases like PostgreSQL, MySQL, MongoDB or AWS DynamoDB as well as BigData platforms like Hadoop, Storm, Spark or Flink;
- The **QuantumLeap Generic Enabler**, has the same functions as the STH-Comet GE, however, while the later doesn´t yet support the NGSIv2 API, is tied to MongoDB and somewhat obsolete, QuantumLeap supports several time-series databases (e.g., InfluxDB, RethinkDB and Crate). In its current state it only supports Crate, as it is easy scalable, supports geo-queries natively, has a nice SQL-like querying and supports integration with visualization tools like Grafana.

As mentioned at the beginning of this chapter, the connection with the devices for collecting information or trigger actuations in response to context updates is done using the IDAS component [6]:

- The **Backend Device Management - IDAS Generic Enabler** offers a wide range of IoT Agents which make it easier to interface with devices using the most widely used IoT protocols:
  - Lightweight Machine-to-Machine (LWM2M) over Constrained Application Protocol (CoAP);
  - JavaScript Object Notation (JSON) over Hypertext Transfer Protocol/ Message Queuing Telemetry Transport (HTTP/MQTT);
  - Ultralight 2.0 over HTTP/MQTT;
  - Open Platform Communications-Unified Architecture (OPC-UA);
  - Long Range Wide Area Network (LoRaWAN).

  It is also possible to develop custom IoT Agents for specific protocols using the tools available for developers.

As all FIWARE components are available as Docker Images, it is possible to set-up a simple system architecture, as exemplified in Figure 2. This system implements the

Orion Context Broker, Cygnus, STH-Comet and the Ultralight 2.0 IoT Agent Generic Enablers; a MongoDB and a Mosquitto MQTT broker to handle MQTT communications. Although this example only uses one type of IoT Agent, multiple Agents could be implemented at the same time, and the same goes for the databases connected and used by Cygnus, which can also use multiple databases at the same time.

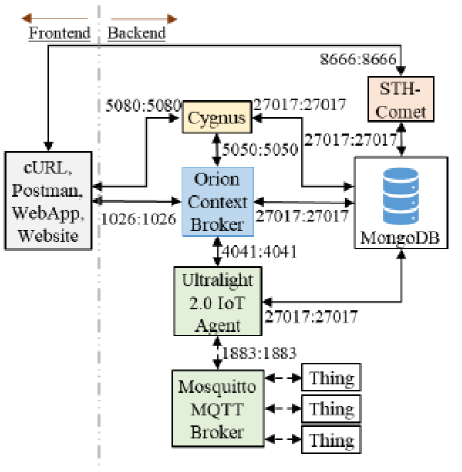


Fig. 2. System Architecture Block Diagram

The Orion Context Broker, the core of the system, allows the management of context information (creation, removal, updates, queries, relationships, registrations and subscriptions) and its availability [8].

The IoT Agent for UltraLight 2.0 provides a North Port NGSI interface used to interact with the Orion, and a South Port to interact with the native protocol of the attached devices, converting the protocols [9]. In this case the IoT Agent converts NGSI requests to UltraLight 2.0 MQTT topics for the Mosquitto MQTT Broker and vice-versa.

The Mosquitto MQTT Broker acts as the central communication point between the IoT Agent and the IoT devices, passing topics between them as necessary [10].

Since the Orion Context Broker only holds the most recent context information, and rather than overload Orion with the task of keeping the context history, this task was delegated to Cygnus, STH-Comet and QuantumLeap [11].

Cygnus, persists context data into one or several databases, creating a historical view of the context data to which is subscribed to [11].

While both Cygnus and STH-Comet can be used to keep a record of context information changes, Cygnus is only capable of saving such changes into several types of databases and not perform queries, whereas STH-Comet can only save changes to a MongoDB database it can also retrieve time-based data aggregations [12].

STH-Comet can be configured to work in the following operation modes [12]:

- Minimal mode, simpler and easier to set-up, STH-Comet is responsible for data collection and interpretation;
- Formal mode, more flexible and future proof, the collection of data is done by Cygnus and the STH-Comet only reads data from an existing database (used in this system).

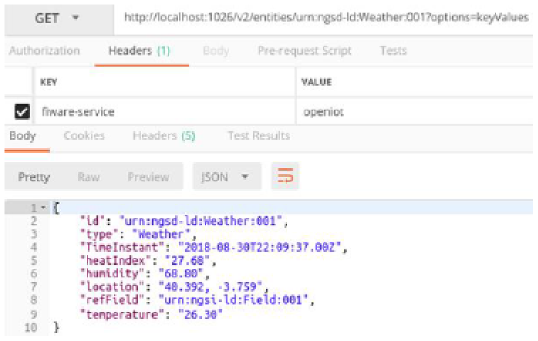
At the time this architecture was designed the QuantumLeap Generic Enabler was in incubation and not yet available, therefore it was not implement nor tested, and will not be further elaborated beyond the initial explanation written at the beginning of this chapter.

The MongoDB database is used by Orion to hold context information, such as entities, subscriptions and registrations; by Cygnus to hold time-based historical context data and by STH-Comet to read said data (formal mode); and by the IoT Agent to hold information about devices, such as API Keys, Ids and other attributes [10].

For testing proposes several devices, composed of a microcontroller (NodeMcu Devkit v1.0 ESP8266 Wi-Fi Module ESP-12E) like the one in Figure 3 connected to a generic sensor (air temperature, air humidity or ultrasonic) or actuator (simulated by turning a LED on or off), were connected to the system, using the UltraLight 2.0 Protocol over MQTT to communicate to the system.

Fig. 3. NodeMcu Devkit v1.0 (front)

After configuring the system (every component), registering the devices in it, and program the devices by using the Arduino IDE and using available libraries for MQTT and the sensors it was possible to send data from the sensors to the system, visualize the data by querying the Orion Context Broker (Figure 4, request sent by using Postman), obtain the context history by querying the STH-Comet and control the actuators by sending commands to Orion.



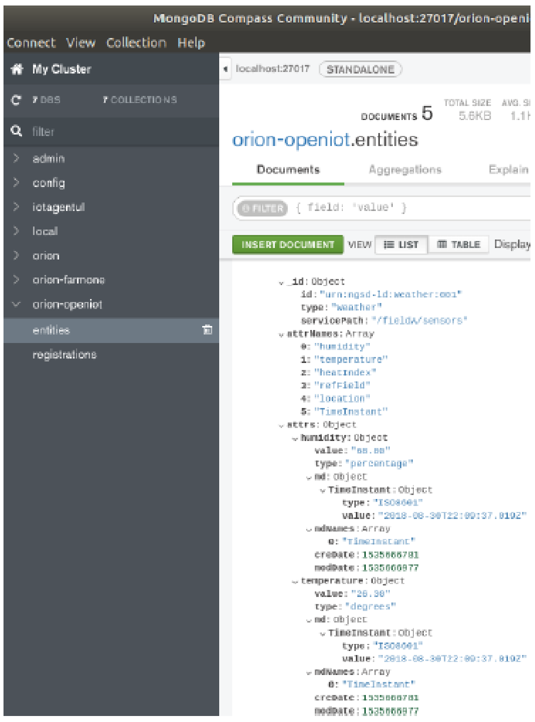
GET http://localhost:1026/v2/entities/urn:ngsd-ld:Weather:001?options=keyValues

KEY	VALUE
✓ fiware-service	openiot

```
{
    "id": "urn:ngsd-ld:Weather:001",
    "type": "Weather",
    "TimeInstant": "2018-08-30T22:09:37.002",
    "heatIndex": "27.68",
    "humidity": "68.80",
    "location": "40.392, -3.759",
    "refField": "urn:ngsi-ld:Field:001",
    "temperature": "26.30"
}
```

Fig. 4. Weather (Air Temperature and Humidity) Sensor Key Values

And also visualize data in all the databases used by each component through the use of the MongoDB GUI, Compass, as shown in Figure 5.



```
_id: Object
    id: "urn:ngsd-ld:weather:001"
    type: "weather"
    servicePath: "/fieldA/sensors"
attrNames: Array
    0: "humidity"
    1: "temperature"
    2: "heatIndex"
    3: "refField"
    4: "location"
    5: "TimeInstant"
attrs: Object
  humidity: Object
      value: "[illegible]"
      type: "percentage"
    md: Object
      TimeInstant: Object
          type: "ISO8601"
          value: "2018-08-30T22:09:37.019Z"
    mdNames: Array
        0: "TimeInstant"
      creDate: 1535666781
      modDate: 1535666977
  temperature: Object
      value: "26.30"
      type: "degrees"
    md: Object
      TimeInstant: Object
          type: "ISO8601"
          value: "2018-08-30T22:09:37.019Z"
    mdNames: Array
        0: "TimeInstant"
      creDate: 1535666781
      modDate: 1535666977
```

Fig. 5. Received Data in the Weather Sensor Entity

Although the system works properly if implement as shown before and on the devices part MQTT has security features built in and only devices which know the API Key can successful communicate with the system, if kept as it is, anyone would be able to access the system it as no security and access control are employed.

V. Security Implementation

Due to the modularity of the FIWARE platform it is always possible to add to a system architecture new Generic Enablers to improve said system or add new functionalities. In this case, security features are added to the system by implementing the following GEs [6]:

- The **Identity Management - Keyrock Generic Enabler** provides secure and private OAuth2 authentication of users and devices, management of user profiles, safekeeping of personal data, Single Sign-On (SSO) and Identity Federation over several administration domains;

- The **PEP-Proxy - Wilma Generic Enabler** enables proxy functions within OAuth2 authentication schemas and applies Policy Enforcement Point (PEP) functions within an eXtensible Access Control Markup Language (XACML) schema;

- The **Authorization Policy Decision Point (PDP) – AuthZForce** Generic Enabler enforces Policy Decision Point/Policy Authorization Point (PDP/PAP) functions within an access XACML schema.

Figure 6, shows the iteration of previous system architecture when the security related GEs are implemented.

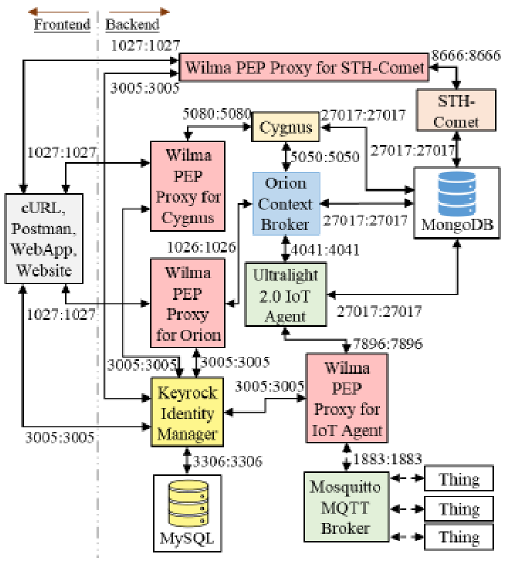


Fig. 6. System Architecture with Security GEs Block Diagram

Keyrock is used to create and manage accounts, roles and permissions for humans and non-human objects like other GEs (e.g., IoT Agent, Wilma PEP Proxy, Devices), which are stored in a MySQL database. This GE is also responsible for authenticating all accounts returning the user access token if successful, which is then used in the requests header [13] [14].

The Wilma PEP Proxy is used to challenge the rights of every user, checking with Keyrock every request before forwarding it to the protected GE. The scrutiny of the PEP Proxy depends on the level of security configured [13] [15] [16]:

- Level 1, checks if the token included in the request header corresponded to an authenticated user;

- Level 2, checks if the token included in the request header corresponded to an authenticated user and if the user roles allow it to access the specified request;

- Level 3, checks if the token included in the request header corresponded to an authenticated user and other advanced parameters like the request headers and body.

Although security is extremely important, these GEs are still being studied therefore are not yet implement nor tested, being the integration of these and the replacement of STH-Comet and Cygnus with QuantumLeap the next phase of this work.

VI. Conclusions

With this article it was demonstrated how to use the FIWARE platform and its technologies to develop a modular universal IoT System able to communicate, control and collet data from IoT devices over a wireless environment.

In the beginning, during the investigation period, enormous difficulties were encountered due to poor documentation and lack of practical examples, it was possible to comprehend how to the Orion Context Broker Generic Enabler worker and how it was used, and what Generic Enablers would have to be used, nevertheless, progress was very slow. However, with the progressive release by FIWARE of a series of practical tutorials, starting in May and continuing during the following months, it was possible to understand how the necessary GEs worked and how to implement them.

Afterwards, the system was developed having been used the following FIWARE Generic Enablers: Orion Context Broker, IoT Agent for Ultralight 2.0 Protocol, Cygnus and STH-Comet; and also, a MongoDB database to store data, the Compass GUI to visualize data in the databases, and a Mosquitto MQTT Broker. The interaction with the system was done though cURL commands or by using the Postman program.

During the system testing phase, each component was tested by sending commands directly to them and by implementing an array of IoT devices with sensors and actuators, which proved that the system was working well as intended.

However, due to the time and effort spent on studying and understanding FIWARE and its components some of the desired features were not yet implemented, the GEs which implement security were not used, a GUI was also not developed, as it is necessary to give the system a focus theme, and other IoT communication protocols, in addition to MQTT, were also not implemented.

Despite this, the base work is done and will continue to be improved and incremented.

A. *Future Work*

As mentioned above, security, a GUI and other IoT protocols were not implemented in this project.

Security:

- The use of the Keyrock Identity Management GE implements OAuth 2.0 authentication for users and devices, and user profile management, which is kept in a SQL database;
- The use of the Wilma Proxy GE serves as a Policy Enforcement Point as well as a proxy isolating the rest of the system from the frontend, only allowing authorized users to interact with the backend;
- The AuthZForce PDP/PAP GE serves as a Policy Decision Point and works in tandem with Wilma to secure the system.

GUI (frontend):

- The GUI which can be an application or website, can be developed using the appropriate technologies as an application or website, which the student never used and didn't have time to learn how to. However, FIWARE also has a GE named Wirecloud which can be used to develop operational dashboards, but the problem of lack poor documentation and practical examples prevented the use of this GE.

Other IoT protocols:

- The MQTT protocol was used in this project, however, MQTT is usually used in Wi-Fi environments were the connection to the Internet is good. If the system was used in a situation where the devices were located outdoor and distributed over a large area, like in the test use case, then it would be unrealistic to cover said area with Wi-Fi due to the large number of antennas needed and the work involved. That is where the use of protocols such as LoRA which already has an IoT Agent available and has a coverage radius of kilometers can be used, although it also requires specific extra hardware that increases the costs of implementation considerably.

References

[1] "FIWARE moves on: From research and innovation to setting European standards and business success", Digital Single Market, 2017. [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/fiware-moves-research-and-innovation-setting-european-standards-and-business-success. [Accessed: 30-Jun-2018]

[2] "After the Open Day: from the FI-PPP to the FIWARE Foundation - FIWARE", FIWARE, 2017. [Online]. Available: https://www.fiware.org/2017/03/09/after-the-open-day-from-the-fi-ppp-to-the-fiware-foundation/. [Accessed: 30-Jun-2018]

[3] "Developers - FIWARE", FIWARE. [Online]. Available: https://www.fiware .org/developers/. [Accessed: 30-Jun-2018]

[4] J. Hierro, M. Reyes, K. Zangelin, I. León, C. Brox, A. Navarro, M. Capdevielle, G. Privat, S. Gómez and M. Bauer, "FIWARE-NGSI v2 Specification", Fiware.github.io, 2018. [Online]. Available: http://fiware.github.io/specifications/ngsiv2/stable/. [Accessed: 01-Jul-2018]

[5] ETSI, "Context Information Management (CIM); Application Programming Interface (API)", ETSI, 2018 [Online]. Available: https://docbox.etsi.org/ISG/CIM /Open/ISG_CIM_NGSI-LD_API_Draft_for_public_review.pdf. [Accessed: 01-Jul- 2018]

[6] "Developers Catalogue - FIWARE", FIWARE. [Online]. Available: https://www.fiware.org/developers/catalogue/. [Accessed: 01-Jul-2018]

[7] U. Ahle, "FIWARE Global Summit - FIWARE Today and Tomorrow", 2018.

[8] "Home - Fiware-Orion", Fiware-orion.readthedocs.io. [Online]. Available: https://fiware-orion.readthedocs.io/en/latest/. [Accessed: 12-Aug-2018]

[9] "IoT Agent - Step-by-Step", Fiware-tutorials.readthedocs.io, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/iot-agent/index.html. [Accessed: 16-Aug-2018]

[10] "IoT over MQTT - Step-by-Step", Fiware-tutorials.readthedocs.io, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/iot-over-mqtt /index.html. [Accessed: 16-Aug-2018]

[11] "Persisting Context - Step-by-Step", Fiware-tutorials.readthedocs.io, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/historic-context /index.html. [Accessed: 18-Aug-2018]

[12] "Short Term History - Step-by-Step", Fiware-tutorials.readthedocs.io, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/short-term-history /index.html. [Accessed: 18-Aug-2018]

[13] "Administrating Users - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/identity-management/index.html. [Accessed: 20- Nov- 2018].

[14] "Managing Roles and Permissions - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/roles-permissions/index.html. [Accessed: 20- Nov- 2018].

[15] "Securing Application Access - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-

tutorials.readthedocs.io/en/latest/securing-access/index.html. [Accessed: 20- Nov- 2018].

[16] "ging/fiware-pep-proxy", *GitHub*, 2018. [Online]. Available: https://github.com/ging/fiware-pep-proxy/blob/master/doc/user_guide.md#level-1-authentication. [Accessed: 20- Nov- 2018].

[17] "Home - QuantumLeap", *Smartsdk.github.io*, 2018. [Online]. Available: https://smartsdk.github.io/ngsi-timeseries-api/. [Accessed: 20- Nov- 2018].

Intentionally Left Blank

References

[1] K. Rose, S. Eldridge and L. Chapin, *The Internet Of Things: An Overview*. The Internet Society, 2015 [Online]. Available: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf. [Accessed: 15-Jun-2018]

[2] "Internet Toaster, John Romkey, Simon Hackett", *Livinginternet.com*. [Online]. Available: https://www.livinginternet.com/i/ia_myths_toast.htm. [Accessed: 15-Jun-2018]

[3] "Internet of Things At-a-Glance", Cisco, 2016 [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf. [Accessed: 15-Jun-2018]

[4] H. Tschofenig, ARM Ltd., J. Arkko, D. Thaler and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC Editor, 2015 [Online]. Available: https://www.rfc-editor.org/rfc/rfc7452.txt. [Accessed: 16-Jun-2018]

[5] C. Marsan, "IAB Releases Guidelines for Internet-of-Things Developers", *IETF Jounal*, 2015 [Online]. Available: https://www.ietfjournal.org/iab-releases-guidelines-for-internet-of-things-developers/. [Accessed: 16-Jun-2018]

[6] "Facebook scandal 'hit 87 million users'", *BBC News*, 2018. [Online]. Available: https://www.bbc.com/news/technology-43649018. [Accessed: 16-Jun-2018]

[7] P. Paganini, "Using Unsecured IoT Devices, DDoS Attacks Doubled in the First Half of 2017", *Security Affairs*, 2017. [Online]. Available: https://securityaffairs.co /wordpress/65827/hacking/iot-devices-ddos-attacks.html. [Accessed: 16-Jun-2018]

[8] C. Matyszczyk, "Samsung's warning: Our Smart TVs record your living room chatter", *CNET*, 2015. [Online]. Available: https://www.cnet.com/news/samsungs-warning-our-smart-tvs-record-your-living-room-chatter/. [Accessed: 16-Jun-2018]

[9] M. Turck, "Growing Pains: The 2018 Internet of Things Landscape", *Matt Turck*, 2018. [Online]. Available: http://mattturck.com/iot2018/. [Accessed: 17-Jun-2018]

[10] "FIWARE moves on: From research and innovation to setting European standards and business success", *Digital Single Market*, 2017. [Online]. Available:

https://ec.europa.eu/digital-single-market/en/news/fiware-moves-research-and-innovation-setting-european-standards-and-business-success. [Accessed: 30-Jun-2018]

[11] "After the Open Day: from the FI-PPP to the FIWARE Foundation - FIWARE", *FIWARE*, 2017. [Online]. Available: https://www.fiware.org/2017/03/09/after-the-open-day-from-the-fi-ppp-to-the-fiware-foundation/. [Accessed: 30-Jun-2018]

[12] "What is FIWARE? - FIWARE", *FIWARE*, 2011. [Online]. Available: https://www.fiware.org/2011/05/17/what-is-fiware/. [Accessed: 30-Jun-2018]

[13] "About Us - FIWARE", *FIWARE*. [Online]. Available: https://www.fiware .org/about-us/. [Accessed: 30-Jun-2018]

[14] "Developers - FIWARE", *FIWARE*. [Online]. Available: https://www.fiware .org/developers/. [Accessed: 30-Jun-2018]

[15] "Smart Industry - FIWARE", *FIWARE*. [Online]. Available: https://www.fiware .org /community /smart-industry/. [Accessed: 30-Jun-2018]

[16] J. Hierro, M. Reyes, K. Zangelin, I. León, C. Brox, A. Navarro, M. Capdevielle, G. Privat, S. Gómez and M. Bauer, "FIWARE-NGSI v2 Specification", *Fiware.github.io*, 2018. [Online]. Available: http://fiware.github.io/specifications/ngsiv2/stable/. [Accessed: 01-Jul-2018]

[17] ETSI, "Context Information Management (CIM); Application Programming Interface (API)", ETSI, 2018 [Online]. Available: https://docbox.etsi.org/ISG/CIM /Open/ISG_CIM_NGSI-LD_API_Draft_for_public_review.pdf. [Accessed: 01-Jul-2018]

[18] "Developers Catalogue - FIWARE", *FIWARE*. [Online]. Available: https://www.fiware.org/developers/catalogue/. [Accessed: 01-Jul-2018]

[19] U. Ahle, "FIWARE Global Summit - FIWARE Today and Tomorrow", 2018.

[20] Engineering Group and FIWARE Foundation, "Engineering Group and FIWARE Foundation announce Knowage as new FIWARE generic enabler for Business Intelligence and Data Analytics on Context Data", 2017 [Online]. Available: https://www.knowage-suite.com/site/wp-content/uploads/2017/06/PR_Knowage_ FIWARE.v2-revised-5.06.2017.pdf. [Accessed: 01- ul-2018]

[21] "Fiware-iotagent-ul - API Walkthrough & Development intro", *Fiware-iotagent-ul.readthedocs.io*. [Online]. Available: https://fiware-iotagent-ul.readthedocs.io/en/latest /usermanual/index.html#user-programmers-manual. [Accessed: 28-Jul-2018]

[22] "IoT over MQTT - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: http://fiware-tutorials.readthedocs.io/en/latest/iot-over-mqtt/index .html. [Accessed: 28-Jul-2018]

[23] "MQTT Essentials Part 6: Quality of Service 0, 1 & 2", *HiveMQ*, 2015. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service -levels. [Accessed: 28-Jul-2018]

[24] "Docker overview", *Docker Documentation*. [Online]. Available: https://docs.docker.com/engine/docker-overview/. [Accessed: 30-Jul-2018]

[25] "Get Started, Part 1: Orientation and setup", *Docker Documentation*. [Online]. Available: https://docs.docker.com/get-started/. [Accessed: 30-Jul-2018]

[26] "Overview of Docker Compose", *Docker Documentation*. [Online]. Available: https://docs.docker.com/compose/overview/. [Accessed: 30-Jul-2018]

[27] "MongoDB Quick Guide", *www.tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm. [Accessed: 30-Jul- 2018]

[28] "What Is MongoDB?", *MongoDB*. [Online]. Available: https://www.mongodb .com/ what-is-mongodb. [Accessed: 30-Jul-2018]

[29] "Getting Started — MongoDB Manual", *Docs.mongodb.com*. [Online]. Available: https://docs.mongodb.com/manual/tutorial/getting-started/. [Accessed: 30-Jul- 2018]

[30] L. Pires, *Microcontroladores*. 2010.

[31] L. Pires, *Sensores e Transdutores*. 2009.

[32] "Resistência LDR 3,4mm - sensor de luz", *Electronicaembajadores.com*, 2018. [Online]. Available: https://www.electronicaembajadores.com/pt/Productos/Detalle /SSLDR34/sensores/sensores-de-brilho-cor-/resistencia-ldr-3-4mm-sensor-de-luz. [Accessed: 10-Aug-2018]

[33] "Switch", 2015 [Online]. Available: https://upload.wikimedia.org/wikiversity /en/7/7b/4.Switch.wiki.20150330.pdf. [Accessed: 10-Aug-2018]

[34] M. Inácio, "Sensores e Atuadores (2)", 2009.

[35] "Motor de engranajes DC RS Pro, Con escobillas, 3 V, 1,5 → 3 V dc, 50 gcm, 2 - 2.300 rpm, 1,6 W", *Pt.rs-online.com*. [Online]. Available: https://pt.rs-online.com/web/p/motores-dc-con-caja-reductora/2389844/. [Accessed: 10- Aug- 2018]

[36] "LED, Díodo Emissor de Luz", *Eletronica PT*. [Online]. Available: https://www.electronica-pt.com/led. [Accessed: 10-Aug-2018]

[37] "FIWARE Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/index.html. [Accessed: 12-Aug- 2018]

[38] "Fiware/tutorials.Step-by-Step", *GitHub*, 2018. [Online]. Available: https://github.com/Fiware/tutorials.Step-by-Step. [Accessed: 12-Aug-2018]

[39] "Postman", *Postman*. [Online]. Available: https://www.getpostman.com/. [Accessed: 12-Aug-2018]

[40] "Home - Fiware-Orion", *Fiware-orion.readthedocs.io*. [Online]. Available: https://fiware-orion.readthedocs.io/en/latest/. [Accessed: 12-Aug-2018]

[41] "Entity Relationships - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/entity-relationships /index.html. [Accessed: 12-Aug-2018]

[42] "Guidelines - Fiware-DataModels", *Fiware-datamodels.readthedocs.io*. [Online]. Available: http://fiware-datamodels.readthedocs.io/en/latest/guidelines/index.html. [Accessed: 12-Aug-2018]

[43] "Home - schema.org", *Schema.org*. [Online]. Available: http://schema.org/. [Accessed: 12-Aug-2018]

[44] "GeoJSON", *Geojson.org*. [Online]. Available: http://geojson.org/. [Accessed: 12-Aug-2018]

[45] "Entity Relationships - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/entity-relationships /index.html. [Accessed: 15-Aug-2018]

[46] "CRUD Operations - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/crud-operations /index.html. [Accessed: 15-Aug-2018]

[47] "Subscriptions - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/subscriptions/index.html. [Accessed: 15-Aug-2018]

[48] "IoT Agent - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/iot-agent/index.html. [Accessed: 16-Aug-2018]

[49] "Eclipse Mosquitto an open source MQTT broker", *Eclipse Mosquitto*, 2018. [Online]. Available: https://mosquitto.org/. [Accessed: 16-Aug-2018]

[50] "IoT over MQTT - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/iot-over-mqtt /index.html. [Accessed: 16-Aug-2018]

[51] "Persisting Context - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/historic-context /index.html. [Accessed: 18-Aug-2018]

[52] "Short Term History - Step-by-Step", *Fiware-tutorials.readthedocs.io*, 2018. [Online]. Available: https://fiware-tutorials.readthedocs.io/en/latest/short-term-history /index.html. [Accessed: 18-Aug 2018]

[53] "Running Orion from command line - Fiware-Orion", *Fiware-orion.readthedocs.io*. [Online]. Available: https://fiware-orion.readthedocs.io/en /master/admin/cli/index.html#command-line-options. [Accessed: 18-Aug-2018]

[54] "Running Orion as system service - Fiware-Orion", *Fiware-orion.readthedocs.io*. [Online]. Available: https://fiware-orion.readthedocs.io/en/master/admin/running /#configuration-file. [Accessed: 18-Aug-2018]

[55] "telefonicaid/fiware-orion", *GitHub*. [Online]. Available: https://github.com /telefonicaid/fiware-orion/blob/master/etc/config/contextBroker. [Accessed: 18-Aug-2018]

[56] "telefonicaid/iotagent-ul Installation & Administration Guide", *GitHub*. [Online]. Available: https://github.com/telefonicaid/iotagent-ul/blob/master/docs/installationguide .md #installation. [Accessed: 18-Aug-2018]

[57] "telefonicaid/fiware-cygnus cygnus-ngsi docker", *GitHub*. [Online]. Available: https://github.com/telefonicaid/fiware-cygnus/blob/master/doc/cygnus-ngsi/installation _and_administration_guide/install_with_docker.md. [Accessed: 20-Aug-2018]

[58] "telefonicaid/fiware-sth-comet", *GitHub*. [Online]. Available: https://github.com /telefonicaid/fiware-sth-comet/blob/master/rpm/EXAMPLES/sth_default.conf. [Accessed: 20-Aug-2018]

[59] "NodeMcu -- An open-source firmware based on ESP8266 wifi-soc.", *Nodemcu.com*. [Online]. Available: http://www.nodemcu.com/index_en.html. [Accessed: 22-Aug-2018]

[60] J. Alves, "ESP8266", *jpralves.net*, 2016. [Online]. Available: https://jpralves.net/post/2016/11/15/esp8266.html. [Accessed: 22-Aug-2018]

[61] "nodemcu/nodemcu-devkit-v1.0", *GitHub*. [Online]. Available: https://github.com /nodemcu/nodemcu-devkit-v1.0. [Accessed: 22-Aug-2018]

[62] AI - Thinker, *ESP - 12E WiFi Module Version1.0*. AI - Thinker, 2015 [Online]. Available: https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs .pdf. [Accessed: 22-Aug-2018]

[63] "Nodemcu Pwm With Arduino Ide", *Electronicwings.com*. [Online]. Available: http://www.electronicwings.com/nodemcu/nodemcu-pwm-with-arduino-ide. [Accessed: 22-Aug-2018]

[64] Handsontec, *User Manual ESP8266 NodeMCU WiFi Devkit*. Handsontec [Online]. Available: http://www.handsontec.com/pdf_learn/esp8266-V10.pdf. [Accessed: 22-Aug-2018]

[65] Aosong (Guangzhou) Electronics, *Digital-output relative humidity & temperature sensor/module AM2303*. Aosong (Guangzhou) Electronics [Online]. Available:

https://www.electroschematics.com/wp-content/uploads/2015/02/DHT22-datasheet.pdf. [Accessed: 22-Aug-2018]

[66] "Sensor de Umidade e Temperatura AM2302 DHT22", *FilipeFlop*. [Online]. Available: https://www.filipeflop.com/produto/sensor-de-umidade-e-temperatura-am 2302-dht22/. [Accessed: 22-Aug-2018]

[67] "DHT22 Temperature and humidity module SKU:SEN0137", *Dfrobot.com*, 2016. [Online]. Available: https://www.dfrobot.com/wiki/index.php/DHT22_Temperature_ and_humidity _module_SKU:SEN0137. [Accessed: 22-Aug-2018]

[68] MaxBotix, *XL-MaxSonar-EZ Series High Performance Sonar Range Finder MB1200, MB1210, MB1220, MB1230, MB1240, MB1260, MB1261, MB1300, MB1310, MB1320, MB1330, MB1340, MB1360, MB1361*. MaxBotix [Online]. Available: https://www.maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf. [Accessed: 23-Aug-2018]

[69] S. Lee, *Soil Moisture Sensor*, 1st ed. ITEAD STUDIO, 2013 [Online]. Available: http://ftp://imall.iteadstudio.com/Electronic_Brick/IM121017001/DS_IM121017001.pd f. [Accessed: 23-Aug-2018]

[70] "How to Use an Ultrasonic Sensor with Arduino [With Code Examples]", *MaxBotix Inc.*. [Online]. Available: https://www.maxbotix.com/Arduino-Ultrasonic-Sensors-085/. [Accessed: 23-Aug-2018]

[71] "Moisture Sensor", *Itead.cc*, 2014. [Online]. Available: https://www.itead.cc /wiki/Moisture_Sensor. [Accessed: 23-Aug-2018]

[72] "Imroy/pubsubclient A client library for the ESP8266 that provides support for MQTT", *GitHub*. [Online]. Available: https://github.com/Imroy/pubsubclient. [Accessed: 23-Aug-2018]

[73] "telefonicaid/fiware-orion", *GitHub*. [Online]. Available: https://github.com /telefonicaid/fiware-orion/blob/master/scripts/accumulator-server.py. [Accessed: 23-Aug-2018]

[74] "Home - QuantumLeap", *Smartsdk.github.io*, 2018. [Online]. Available: https://smartsdk.github.io/ngsi-timeseries-api/. [Accessed: 20- Nov- 2018]