

Häme University of Applied Sciences - HAMK
University Centre-Hämeenlinna
Smart Services Research Unit

IoT Centralization and Management Applying ThingsBoard Platform

Keywords: IoT, Smart Campus, WEB Platform

Project developed under the exchange program *PROPICIE 13th ed.*, in cooperation between the Brazilian institution *Federal Institute of Santa Catarina - IFSC* and the Finnish institution *Häme University of Applied Sciences - HAMK*.

Advisors:

Joni Kukkamäki (HAMK)

Robson Costa (IFSC)

Vilson Heck Júnior (IFSC)

Authors:

Eduardo Natan Bitencourt

Willian Pereira dos Anjos

Hämeenlinna – FI, August 2018

1. TABLE OF CONTENTS

Introduction	4
Objectives	6
Theoretical Approach	7
IoT Architectures	7
IoT Platforms	9
IoT protocols	9
MQTT	10
HTTP	11
CoAP	12
OPC-UA	13
Comparison	15
Microcontrollers	16
Telemetry data	18
Cloud platforms (Azure)	19
NoSQL databases	20
Cassandra database	20
Containers and Docker	22
Methodology and Research	25
IoT Platforms comparison survey	25
Thingsboard	27
Licenses	27
Installation (cloud/own server)	28
Architecture	29
Modules (protocols and databases)	30
External connection (in/out, APIs)	30
Analyzing/visualizing tools	31
Pros/Cons	32
Wemos D1 Mini Pro	32
DTH 11	33
PlatformIO	34
Experimental Procedures and Results	36
Installation of the Thingsboard Platform (Azure)	36
Using Docker	37
Configuration of Thingsboard (Creating users, devices and dashboards)	42
Getting temperature and humidity from devices	43
Using MQTT	44

Using HTTP	45
Monitoring and managing the Thingsboard database	45
Thingsboard Rest API	48
Embedding widget and dashboard to an external web page.	49
Rules, Alarms and Plugins. Firing alert Emails.	50
Thingsboard Gateway	52
MQTT External Broker	53
OPC-UA	54
Integrating Thingsboard with Yepzon Asset (industrial tracker)	55
Conclusion	58
Acknowledgments	60
References	61
Appendices	64

2. Introduction

The Internet of Things (IoT) is a communication paradigm that envisions a near future in which the objects of everyday life will be equipped with microcontrollers, transceivers for digital communication and suitable protocol stacks. It will enable these objects to communicate with the users, becoming an integral part of the internet [1].

In this context, the IoT technology shall be able to incorporate transparently and seamlessly a large number of different and heterogeneous end systems, while providing open access to selected subsets of data for the development of a plethora of digital services [1]. This paradigm indeed finds application in many different domains, such as home automation, industrial automation, medical aids, mobile healthcare, elderly assistance, intelligent energy management, smart grids, automotive, traffic management, and many others [2].

A survey by Gartner Inc., a market research agency, predicts that by 2020 there would be about 20.4 billion devices connected by this technology [3]. The increase is mainly driven by lower manufacturing costs of sensors, actuators and microcontrollers, as well as the availability of COTS (Commercial Off-the-Shelf).

As a consequence, cheap Internet of Things devices led to an increase in the range of projects available to developers. There is still no global standardization; therefore, developers have difficulties in defining an efficient way to transmit, store, manage and provide the data. These difficulties increase when one wants to manage the data of several applications and devices in an IoT environment or context. For this reason, an IoT platform could suit very well, one that supports efficient and proper devices management, implemented with multiple technologies, protocols, microcontrollers, sensors, actuators and belonging to different entities.

In this way, the use of a tool capable of centralizing the management of IoT environments, projects and software applications and also able to provide a good variety of visualization widgets is an attractive solution. There are several options already available on the market, however, which one is ideal for our context. An educational institution's context has some characteristics that need to be analyzed. First, requires versatility in support terms for a large number of devices and users, like teachers and

students. The tool needs to be adaptable to the diversity of communication protocols, as well as the liberty of distribution and open source code.

In short, an in-depth and comparative study will be necessary to choose the IoT platform tool more appropriate to the institutional context in the best possible way. As soon as the platform is chosen, a case study and testing with different devices, protocols, microcontrollers, and applications will be carried out to create hypotheses and define the advantages and disadvantages of this technology. Finally, implantation of the platform in a real-life context must be done, with real users, devices and projects.

3. Objectives

Main Objective:

Conduct a study to validate a centralized management infrastructure for IoT environments, projects and devices through an open source platform for educational and research purposes.

Specific Objectives:

- *Analyze the open source IoT platforms already available on the market, conduct a comparative study in order to identify its advantages and limitations in relation to the characteristics desired for the execution of the project.*
- *Study of IoT communication protocols: MQTT, HTTP, COAP, OPC-UA;*
- *Study of Devices, Microcontrollers, Sensors and Actuators used in IoT Projects;*
- *Study of NoSQL databases for IoT: Cassandra;*
- *Install the platform with NoSQL database Cassandra;*
- *Study of Containers and Docker;*
- *Install the platform with Docker;*
- *Configure the platform: Customers, Users, Rules, Plugins, Devices;*
- *Make a practical implementation testing the communication of the IoT platform and the protocols previously highlighted;*
- *Validation of real-time data visualization tools and widgets provided by the platform;*

4. Theoretical Approach

To understand the purpose and the project as a whole is necessary to know some theoretical concepts about technologies and approaches used in this paper.

First, we will dive into IoT architecture and software platforms for visualization and management of IoT environments and understand which problems they intend to solve. Furthermore, the main principles of the project are described, like IoT protocols, microcontrollers, NoSQL databases, among others.

4.1. IoT Architectures

IoT system architecture has the ability to perform interoperability among heterogeneous assets around us. Considering the aforementioned fact, the architecture of IoT should be flexible layered [4]. As there is no commonly accepted IoT architecture, authors proposed various architectures, few among them are SoA based architecture proposed by Atzori et al. and five-layered architecture, that can be seen in figure 1 [5].

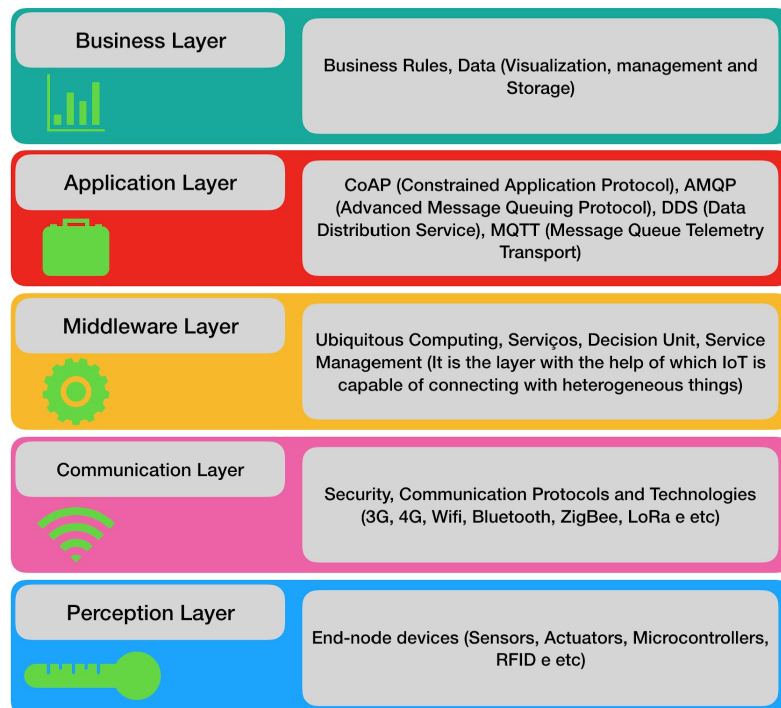


Figure 1. IoT five-layered architecture. [8]

A brief introduction to the five layers of IoT system architecture is as follows [5]:

- **Perception Layer:** The Perception layer is also known as 'Device Layer'. It consists of physical objects and sensor devices. The sensors can be RFID, 2D-barcode, or Infrared sensor depending upon objects identification method. This layer basically deals with the identification and collection of objects specific information by the sensor devices. Depending on the type of sensors, the information can be about location, temperature, orientation, motion, vibration, acceleration, humidity, chemical changes in the air and etc. The collected information is then passed to the Communication layer for its secure transmission to the information processing system.
- **Communication Layer:** The Communication layer can also be called 'Transmission Layer' or 'Network Layer'. This layer securely transfers the information from sensor devices to the information processing system. The transmission medium can be wired or wireless and technology can be 3G, UMTS, Wifi, Bluetooth, infrared, ZigBee, etc depending upon the sensor devices. Thus, the Network layer transfers information from the Perception layer to the Middleware layer.
- **Middleware Layer:** The devices over the IoT implement different types of services. Each device connects and communicates with only those other devices which implement the same service type. This layer is responsible for service management and has links to the database. It receives information from the Network layer and stores in the database. It performs information processing and ubiquitous computation and takes automatic decisions based on the results.
- **Application Layer:** This layer provides global management of the application based on the objects information processed in the Middleware layer. The applications implemented by IoT can be smart health, smart farming, smart home, smart city, intelligent transportation, etc. The Application layer incorporates CoAP, MQTT, HTTP protocols. These protocols are discussed in Section 4.3.

- **Business layer:** Also named as management layer manages all layers, activities, and service of IoT. It incorporates certain graphs, flowcharts, and models based on the data acquired from the application layer. This layer is capable of making effective decisions for big data analysis.

4.2. IoT Platforms

The Internet of Things (IoT) management software helps manage strategies involving the connectivity of smart devices, as well as smart packaging, and their impact on business. The internet of things refers to the wireless communication between devices and their ability to send, receive, and create data based on user activity and environmental factors. IoT management products help businesses monitor and take action on communication from and between registered devices, as well as control the devices from a remote interface on a desktop or mobile device when necessary. These products log and store data from connected smart devices that provide real-time insights to help businesses uncover trends and become more efficient. IT teams within various organizations use IoT software to centralize activity and analytics related to a smart device network, receive alerts when performance is interrupted, and export relevant information to other IT infrastructure or analytics programs [30].

To qualify a system as an Internet of Things Platform, software must:

- Sync with and monitor activity of smart devices;
- Provide tools for controlling, updating, and retrieving data from synced devices;
- Allow actions to be taken based on data received from devices;
- Provide dashboards and analytics for devices.

4.3. IoT protocols

It is a fact that IoT protocol world is complicated due to legacy protocols, emerging technologies, different layering methodologies and many use cases. There are hundreds of protocols and just a few standards to organize its layers and formats, as shown on Section 4.1 the five-layer architecture is a good approach for this paper's purpose.

In this project context, the most important layer is the application layer, where we can find the data protocols, since this paper focuses on data visualization\storage\management. The most common protocols used in this layer are: MQTT; CoAP; HTTP; OPC-UA* (OPC-UA is not a stand-alone protocol, more on that on Sections 4.2.4 and 4.2.5).

4.3.1. MQTT

Created in 1999 by Andy Stanford-Clark of IBM, it is described as “a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal for the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.” [6].

One important element of MQTT's architecture is the broker, according to the *HoveMQ* official Web site: “The broker is primarily responsible for receiving all messages, filtering them, deciding who is interested in it and then sending the message to all subscribed clients.” [7].

Generally speaking, the MQTT protocol uses a many-to-many paradigm and the broker decouples the publisher to the subscriber and acts as a message router. This protocol implements QoS, secure communication, persistence and so on. There is another version of this MQTT IoT protocol called MQTT-SN (or MQTT-S) that stands for Sensor network.

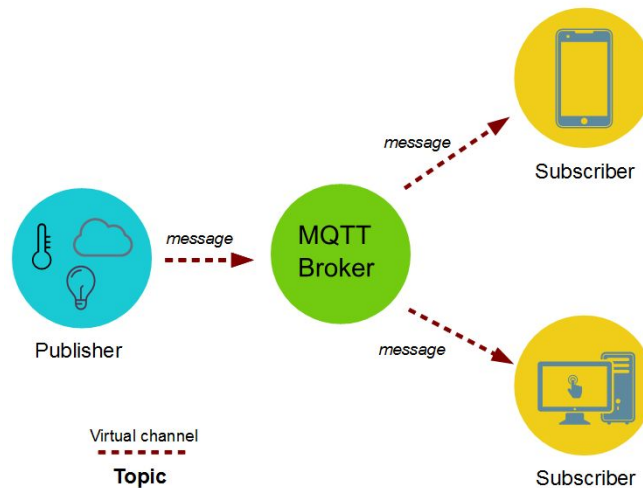


Figure 2. MQTT architecture. [8]

4.3.2. HTTP

The W3C Foundation, the official maintainer of HTTP, affirms “The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information and possible entity-body content.” [9].

This protocol, used as IoT protocol, is exploited to exchange data between applications/devices and to integrate applications belonging to different domains. HTTP uses client/server paradigm. HTTP is widely used not only in the IoT ecosystem, but in almost every web based application in the world.

For devices used in the IoT world, it is difficult to implement the old HTTP protocol due to various reasons. The main reason is the fact that IoT is made up of unseen devices that require very little interaction. IoT devices consume very little power and frequently have poor network connectivity. Accordingly, HTTP is too heavy to be a good fit for these devices. An HTTP request requires a minimum of nine TCP packets, even more when one considers packet loss from poor connectivity, and plain text headers can

get very verbose. And even with all this overhead HTTP doesn't guarantee message delivery [10]. The HTTP overhead also adds to IoT operating expenses. The current costs of wireless connectivity are exorbitant. Bandwidth conservation is especially important with enterprise customers that often have hundreds of thousands or millions of devices deployed.

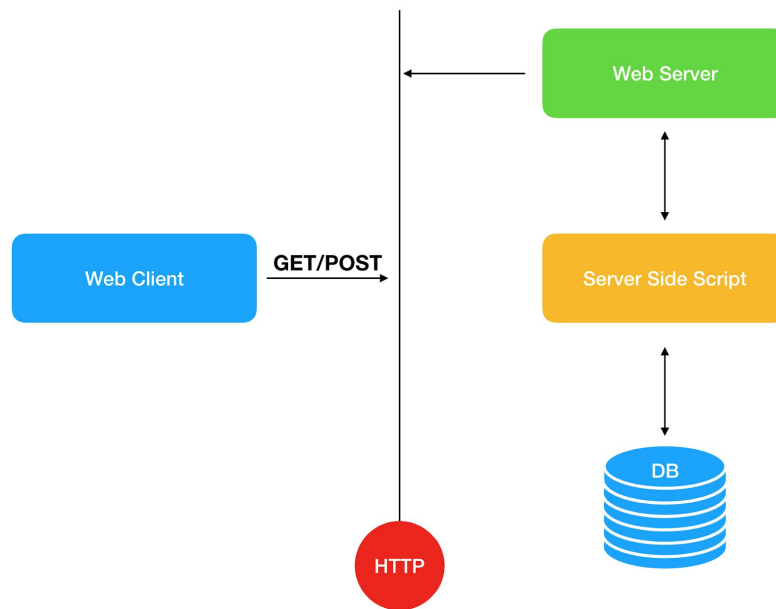


Figure 3. HTTP architecture.

4.3.3. CoAP

COAP stands for Constrained Application Protocol. This protocol was designed by IETF. It is an M2M protocol, defined by RFC7252, and it is a web transfer protocol. This protocol is very similar to HTTP and it uses the document transfer paradigm. Moreover, it uses the request/response model and supports built-in service discovery. Respect to HTTP protocol, COAP is lightweight with smaller packets. COAP uses UDP as the underlying network protocol. "COAP is basically a client-server IoT protocol where the client makes a request and the server sends back a response as it happens in HTTP. The methods used by COAP are the same used by HTTP" [11].

Conceptually, CoAP is compatible with HTTP except that it's designed for devices with constrained resources like sensors and microcontrollers that have restricted memory and bandwidth capabilities. It's designed to support RESTful architecture, so it

supports the four main methods (GET, POST, PUT and DELETE) and can identify resources by a URL [12].

Therefore, CoAP could be considered a non-official version of HTTP for IoT contexts. One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response Codes (see Figure 4) [13].

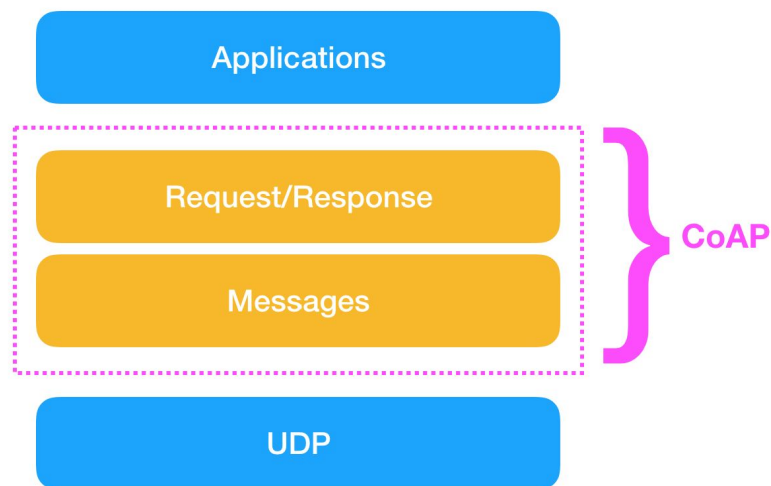


Figure 4. CoAP two-layer approach.

4.3.4. OPC-UA

OPC-UA is not a protocol, but a whole architecture where data protocol is only part of the topics addressed.

OPC Unified Architecture (OPC UA) is the current OPC Foundation technology for secure, reliable and interoperable transport of raw data and pre-processed information from the shop floor into production planning systems. Definition of OPC specifications started to simplify and to standardise data exchange between software applications in industrial environment [14]. To facilitate this information exchange OPC UA specifies the following: The information model to represent structure, behaviour and semantics; The message model to interact between applications; The communication model to transfer

the data between end-points; The conformance model to guarantee interoperability between systems.

As for its architecture, the OPC UA systems architecture models OPC UA Clients and Servers as interactive partners. Each system may contain multiple clients and servers. Each client may interact with one or more servers, and each server may interact concurrently with one or more clients. An application may combine Server and Client components to allow interaction with other Servers and Clients [14].

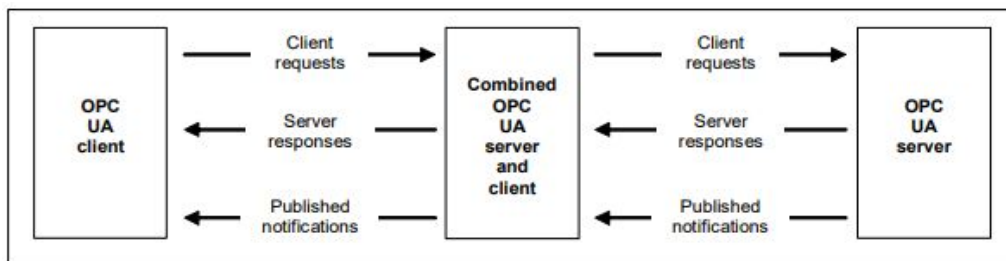


Figure 5. OPC-UA Server/Client Architecture. [14]

The client part of OPC-UA is the endpoints application. It uses the OPC UA Client API to send and receive OPC UA Service requests and responses to the OPC UA Server.

In Figure 6 an expanded architecture is shown making all the OPC-UA server components visible.

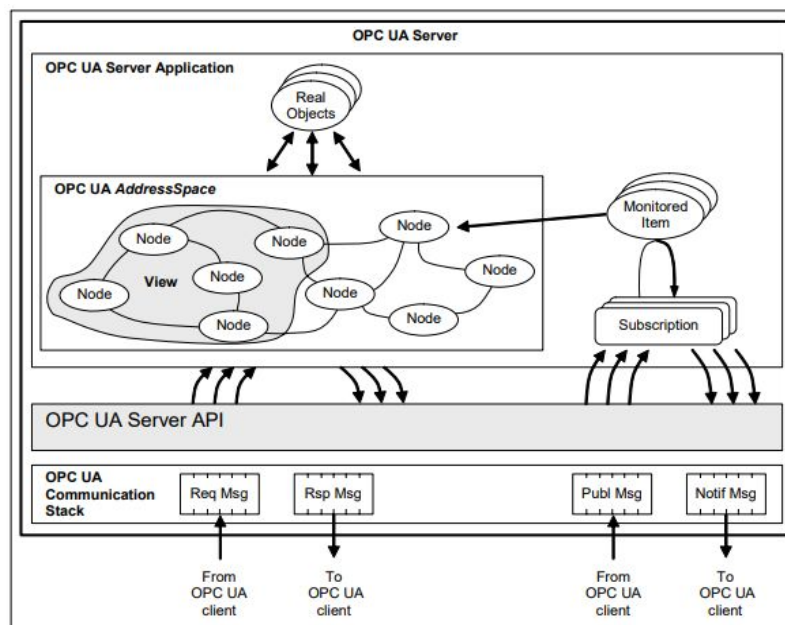


Figure 6. OPC-UA Server Architecture. [14]

- **Real Objects:** Are physical or software objects that are accessible by the OPC UA Server application or that it maintains internally. Examples include physical devices and diagnostics counters.
- **OPC UA Server application:** The OPC UA Server application is the code that implements the function of the Server. It uses the OPC UA Server API to send and receive OPC UA Messages from OPC UA Clients.
- **OPC UA AddressSpace:** AddressSpace is modeled as a set of Nodes accessible by clients using OPC UA Services (interfaces and methods). Nodes in AddressSpace are used to represent real objects, their definitions and their references to each other.
- **MonitoredItems:** Entities in the Server created by the Client that monitor AddressSpace Nodes and their real-world counterparts. When they detect a data change or an event/alarm occurrence, they generate a Notification that is transferred to the Client by a Subscription.
- **Subscriptions:** A Subscription is an endpoint in the Server that publishes Notifications to Clients. Clients control the rate at which publishing occurs by sending Publish Messages.

The OPC UA specifications are layered to isolate the core design from the underlying computing technology and network transport. This allows OPC UA to be mapped to future technologies as necessary, without negating the basic design [14].

For transportation and data formats, OPC-UA is able to use a couple of different technologies, and this creates some sort of freedom for developers to create whichever combination they need.

For data encoding: XML or UA Binary.

For transporting data: TCP or SOAP Web services over HTTP.

4.3.5. Comparison

Evidently, it is difficult to compare OPC-UA, which has a complete set of features including protocols with MQTT and CoAP, which are stand-alone protocols. Moreover, OPC-UA is restricted to be used in industrial contexts while MQTT or CoAP can be found

in several different contexts and applications. Also, since CoAP is an HTTP with a smaller overhead, this comparison will be restricted to CoAP and MQTT.

Protocol	Advantages	Disadvantages
MQTT	<ul style="list-style-type: none"> - Easy to implement; - Useful for connections with remote location; - Small code footprint; - Lightweight Asymmetric client-server relationship 	<ul style="list-style-type: none"> - No error-handling; - Hard to add extensions; - Basic message queuing implementations; - Doesn't address connection security
CoAP	<ul style="list-style-type: none"> - Multicast support; - Low overhead; - Minimizing complexity of mapping with HTTP; - Communication models flexibility; - Low latency 	<ul style="list-style-type: none"> - Doesn't enable communication level security; - Few existing libraries and solution support; - Small community

Table 1. MQTT and CoAP comparison.

4.4. Microcontrollers

A microcontroller is nothing less than a computer. Personal computers, mainframes or a microcontroller have several things in common like [15]:

- All computers have a CPU (central processing unit) that executes programs.
- The CPU loads the program from somewhere. On your desktop machine, the browser program is loaded from the hard disk.
- The computer has some RAM (random-access memory) where it can store "variables."
- The computer has some input and output devices so it can talk to people. On your desktop machine, the keyboard and mouse are input devices and the monitor and printer are output devices. A hard disk is an I/O device that handles both input and output.

The differences between a desktop computer and a microcontroller are that microcontrollers are special purpose computers, unlike general purpose computers. That is, microcontrollers do one thing well, they are specialized in specific tasks. Some characteristics that define microcontrollers are [15]:

- Microcontrollers are "embedded" inside some other device (often a consumer product) so that they can control the features or actions of the product. Another name for a microcontroller, therefore, is "embedded controller."
- Microcontrollers are dedicated to one task and run one specific program. The program is stored in ROM (read-only memory) and generally does not change.
- Microcontrollers are often low-power devices. A desktop computer is almost always plugged into a wall socket and might consume 50 watts of electricity. A battery-operated microcontroller might consume 50 milliwatts.
- A microcontroller has a dedicated input device and often (but not always) has a small LED or LCD display for output. A microcontroller also takes input from the device it is controlling and controls the device by sending signals to different components. For example, the microcontroller inside a TV takes input from the remote control and displays output on the TV screen. The controller controls the channel selector, the speaker system and certain adjustments on the picture tube electronics such as tint and brightness. The engine controller in a car takes input from sensors such as oxygen and knocks sensors and controls things like fuel mix and spark plug timing. A microwave oven controller takes input from a keypad, displays output on an LCD display and controls a relay that turns the microwave generator on and off.
- A microcontroller is often small and low cost. The components are chosen to minimize size and to be as inexpensive as possible.
- Microcontrollers are often, but not always, ruggedized in some way. The microcontroller controlling a car's engine, for example, has to work in temperature extremes that a normal computer generally cannot handle. A car's microcontroller in Alaska has to work fine in -30 degree F (-34 C) weather, while the same microcontroller in Nevada might be operating at 120 degrees F (49 C). When you

add the heat naturally generated by the engine, the temperature can go as high as 150 or 180 degrees F (65-80 C) in the engine compartment. On the other hand, a microcontroller embedded inside a VCR hasn't been ruggedized at all.

One common mistake for microcontrollers is to associate it with an Arduino or Raspberry Pi board. Arduino and Raspberry Pi are not microcontrollers, either. Both are still computers by definition but are breakout boards that can have a microcontroller. Concluding, a microcontroller is a small computer with low-memory and programmable input/output peripherals. We use them today because their low-powered and low memory makes them low-cost. Microcontrollers are part of the reason the Internet of Things is possible and prosperous today [15].

4.5. Telemetry data

Telemetry is the automatic recording and transmission of data from remote or inaccessible sources to an IT system in a different location for monitoring and analysis. Telemetry data may be relayed using radio, infrared, ultrasonic, GSM, satellite or cable, depending on the application (telemetry is not only used in software development, but also in meteorology, intelligence, medicine and other fields).

In the software development world, telemetry can offer insights into which features end users use most, detection of bugs and issues, and offering better visibility into performance without the need to solicit feedback directly from users.

In a general sense, telemetry works through sensors at the remote source which measures physical (such as precipitation, pressure or temperature) or electrical (such as current or voltage) data. This is converted to electrical voltages that are combined with timing data. They form a data stream that is transmitted over a wireless medium, wired or a combination of both.

At the remote receiver, the stream is disaggregated and the original data is displayed or processed based on the user's specifications.

In the context of software development, the concept of telemetry is often confused with logging. But logging is a tool used in the development process to diagnose errors and code flows, and it's focused on the internal structure of a website, app, or another

development project. Once the project is released, however, telemetry is what you are looking for to enable automatic collection of data from real-world use. Telemetry is what makes it possible to collect all that raw data that becomes valuable, actionable analytics.

One of the main benefits from telemetry is to be able to monitor the state of an object or environment when you can not be physically present. Telemetry is an incredibly valuable tool for ongoing performance monitoring and management. [16]

4.6. Cloud platforms (Azure)

Cloud is an abstract virtual environment where programs and data are stored. Cloud computing power is provided by data centers, which can contain systems for data storage and many servers that have the ability to manage almost any software, and customers pay flexibly according to the resources used, based on a monthly fee. Also in Cloud Computing users do not need to buy software or maintain expensive servers and devices for data storage, this leads to significant reduction of expenses, office space, and internal staff for IT support and increase of data security. This solution was made to users that need an efficient and adaptable infrastructure for the purpose of their business. This involves a non-stop running service, available from anywhere in the world. [17].

There are many options for cloud computing platforms available on the market. Windows Azure is the Microsoft solution that appeared in 2009 and contains the following components:

- Windows Azure: Provides a platform for running Windows applications and storing their data in the clouds. Windows Azure runs on a large number of servers in Microsoft data centers, which are connected into a unified whole by Windows Azure Fabric. Windows-based computing and storage services for cloud applications are built on top of this fabric.
- SQL Azure: Provides data services in a cloud based on SQL Server. Although the eventual goal of SQL Azure includes a range of data-oriented capabilities, including data synchronization and reporting, the first SQL Azure component to appear is SQL Azure Database, which provides a cloud-based database management system (DBMS).

- Windows Azure Platform AppFabric: Provides cloud services for connecting applications running on the cloud or on premises. The functions provided by AppFabric are called cloud-based infrastructure services, including the Service Bus and Access Control Service. [18].

4.7. NoSQL databases

NoSQL (not-only SQL) databases have emerged as a solution to storage scalability issues, parallelism, and management of large volumes of unstructured data. In general, NoSQL systems have the following characteristics [19]: (i) they are based on a nonrelational data model; (ii) they rely on distributed processing; (iii) high availability and scalability are main concerns; and (iv) some are schemaless and have the ability to handle both structured and unstructured data.

There are four main categories of NoSQL databases [20]:

- (i) Key-value stores: data is stored as key-pairs values. These systems are similar to dictionaries, where data is addressed by a single key. Values are isolated and independent from another, and relationships are handled by the application logic.
- (ii) Column family database: it defines the data structure as a predefined set of columns. The super columns and column family structures can be considered the database schema.
- (iii) Document-based storage: a document store uses the concept of key-value store. The documents are collections of attributes and values, where an attribute can be multivalued. Each document contains an ID key, which is unique within a collection and identifies document.
- (iv) Graph databases: graphs are used to represent schemas. A graph database works with three abstractions: node, relationships between nodes, and key value pairs that can attach to nodes and relationships.

4.7.1. Cassandra database

Cassandra Database System. Cassandra is a cloud-oriented database system, massively scalable, designed to store large amounts of data from multiple servers, while providing high availability and consistent data [21]. Cassandra is considered a hybrid NoSQL database, using characteristics of both key-value and column-oriented databases.

Cassandra's architecture is made of nodes, clusters, data centers and a partitioner. A node is a physical instance of Cassandra. Cassandra does not use master-slave architecture; rather, Cassandra uses peer-to-peer architecture, where all nodes are equal. A cluster is a group of nodes or even a single node. A group of clusters is a data center [20].

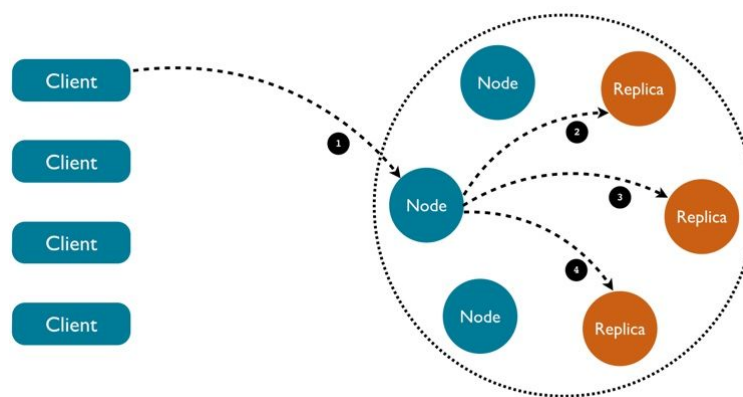


Figure 7. Cassandra diagram example.

For developers and administrators coming from the relational world, the Cassandra data model can be very difficult to understand initially. Some terms, such as “keyspace,” are completely new, and some, such as “column,” exist in both worlds but have different meanings [20].

Cassandra defines a column family as a logical division that associates similar data. For example, we might have a User column family, a Hotel column family, an Address Book column family, and so on. In this way, a column family is somewhat analogous to a table in the relational world. Putting this all together, we have the basic Cassandra data structures: the column, which is a name/value pair (and a client-supplied timestamp of when it was last updated), and a column family, which is a container for rows that have similar, but not identical, column sets. The row keys are a reference to an entity stored in the database, analogous to the basic row in SQL.

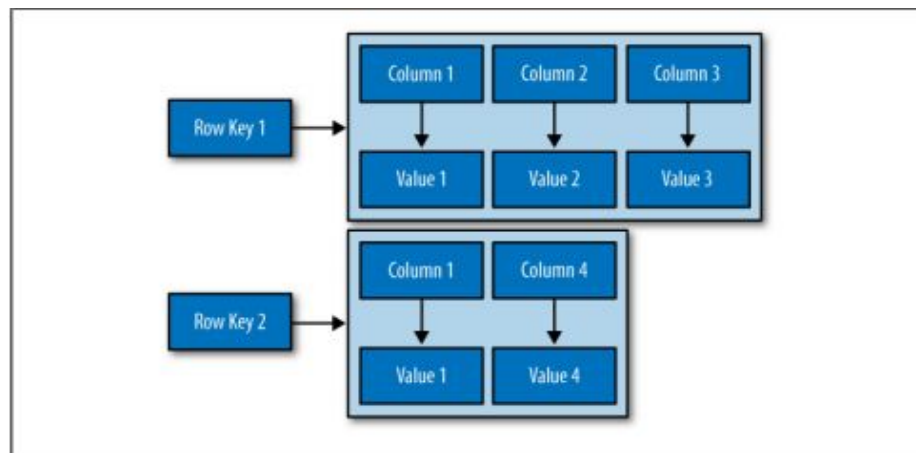


Figure 8. Cassandra column family example. [21]

4.8. Containers and Docker

Containers, or otherwise known as operating-system-level virtualization, are a lightweight approach to virtualization that only provides the bare minimum that an application requires to run and function as intended. In a way, they can be considered super minimalist virtual machines that are not running on hypervisor.

Items usually bundled into a container include:

- Application;
- Dependencies;
- Libraries;
- Binaries;
- Configuration files;

Containerizing an application enables it to run reliably in different environments by abstracting away the operating system and the physical infrastructure. Containerized applications are sharing the kernel of the host operating system with other containers and the shared part of the OS is read only. Inside a container, there is often a single executable service or microservice.

The idea of containers is not new; Linux-based operating systems had the technology available since the early 2000's. However, it needed Docker's appearance on the scene to really make it into the mainstream of application development. [22]

The docker is an open-source project for automating the deployment of applications as portable, self-sufficient containers that can run on any cloud or on-premises. The docker is also a company promoting and evolving this technology with a tight collaboration with clouds, Linux and Windows vendors.

Docker is becoming the standard unit of deployment and is emerging as the de-facto standard implementation for containers as it is being adopted by most software platforms and cloud vendors (Microsoft Azure, Amazon AWS, Google, etc.).

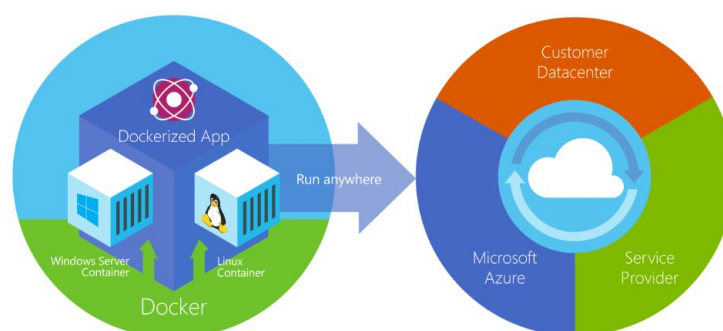


Figure 9. Docker deploys containers at all layers of the hybrid cloud.

With a container-oriented approach, you can eliminate most of the issues that arise when having inconsistent environmental setups and the problems that come with them. The bottom line is that when running an app or service inside a container you avoid issues that come with inconsistent environments.

Another important benefit when using containers is the ability to quickly instance any container. For instance, that allows to scale-up fast by instantly instantiating a specific short-term task in the form of a container. From an application point of view, instantiating an image (the container), should be treated in a similar way than instantiating a process (like a service or web app), although when running multiple instances of the same image across multiple host servers, you typically want each container (image instance) to run in a different host server/VM in different fault domains, for reliability.

In short and as the main takeaways, the main benefits provided by containers are isolation, portability, agility, scalability and control across the entire application lifecycle workflow. But the most important benefit is the isolation provided between Dev and Ops.

Following these characteristics, the container concept is an excellent option to use

in applications that need to run on more than one cloud, applications that use microservices and applications that benefit from DevOps. [23]

5. Methodology and Research

As this paper's main goal was the validation of an IoT platform as a tool to make IoT data visualization\storage\management easier and more efficiently, first a comparison study was executed in order to compare the pros and cons of several IoT platforms available in the market. In this manner, the tool that seemed most appropriate for the purpose of the project was chosen.

Subsequently, tests and validations were conducted with the platform, for that, a microcontroller with a DTH sensor was used in order to simulate a real IoT device. The objective was to test how the platform behaves, working directly with different protocols and data transmission methods.

Other aspects of the platform were validated as well, like data storage, visualization tools, integration with different databases and cloud services, alarm engines and user management between others, all of these tests are described in Section 6.

To be able to understand these tests it is necessary to learn how to use properly all of these technologies and approaches used during this project.

5.1. IoT Platforms comparison survey

To choose the ideal IoT Platform, it was necessary to compare the main platforms available on the market. The defined parameters to choose the platform was:

- Open Source;
- Support to install on cloud or own server;
- Variety of protocols;
- API support (External integrations);
- Analyzing and Visualization tools.

There is a limited quantity of Open Source IoT Platforms available on the market. The following table was created based on this study [24] to hint the main features of each open source platform found:

IoT Software Platform	Device management	Integration	Security	Protocols for data collection	Analytics	Support for visualizations	DB
-----------------------	-------------------	-------------	----------	-------------------------------	-----------	----------------------------	----

ThingsBoard	Yes	REST APIs, MQTT APIs	Industry standard encryption algorithms (SSL) and device credentials types (X.509 certificates and access tokens)	HTTP, MQTT, OPC-UA, CoAP and Node-red	Real time analytics(Apache Spark, Kafka)	Yes	PostgreSQL, Cassandra, HSQLDB
Kaa IoT Platform	Yes	Portable SDK available to integrate any particular platform, REST API	Link Encryption (SSL), RSA key 2048 bits, AES key 256 bits	MQTT, CoAP, XMPP, TCP, HTTP, WiFi, Ethernet, Zigbee	Real time IoT Data Analytics and Visualization with Kaa, Apache Cassandra and Apache Zeppelin	Yes (Doesn't have own dashboards)	MongoDB, Cassandra, Hadoop, Oracle NoSQL
WSO2	Yes	REST APIs	Link Encryption (SSL) and basic authentication	HTTP, WSO2 ESB, MQTT	Yes, WSO2 Data Analytics Server	Yes	Oracle, PostgreSQL, MySQL, or MS SQL
Site Where	Yes	REST API, Mule AnyPoint, and more	Link Encryption (SSL), Spring Security	MQTT, AMQP, Stomp, WebSockets, and direct socket connections	Real-time analytics (Apache Spark)	No	MongoDB, HBase, InfluxDB
Thing Speak	No	REST API, MQTT APIs	Basic Authentication	HTTP	MATLAB Analytics	No	MySQL
DeviceHive	*Unknown	REST API, MQTT APIs	Basic Authentication using JSON Web Tokens (JWT)	REST API, WebSockets or MQTT	Real-time analytics (Apache Spark)	Yes (Doesn't have own dashboards)	PostgreSQL, SAP Hana DB
Zetta	No	REST APIs	Basic Authentication	HTTP	Using Splunk	No	Unknown
Distributed Services Architecture (DSA)	No	REST APIs	Basic Authentication	HTTP	No	No	ETSDB – Embedded Time Series
Thingier.io	Yes	REST APIs	Link Encryption (SSL/TLS) and basic authentication	MQTT, CoAP and HTTP	Yes	No	MongoDB

Table 2. IoT platforms comparison.

From the nine platforms, only four have support for visualizations, ThingsBoard, Kaa IoT, WSO2, and DeviceHive. This support applies the capacity to see data on dashboards, an important prerequisite. Comparing the tools only ThingsBoard and WSO2

have your own dashboard creation and management, the other tools use external tools to be able to generate dashboards.

Comparing ThingsBoard and WSo2 is important to highlight the fact that the first one has the better adaptability to the main protocols of IoT, including OPC-UA.

ThingsBoard presented a better usability and features compared with other open source tools, so it was the IoT platform selected. Following this, we started with the study of the platform.

5.2. Thingsboard

The ThingsBoard official website describes it as an open-source IoT platform for data collection, processing, visualization, and device management. It enables device connectivity via industry standard IoT protocols - MQTT, CoAP and HTTP and supports both cloud and on-premises deployments. ThingsBoard combines scalability, fault-tolerance and performance [25].

5.2.1. Licenses

ThingsBoard has two versions, the Community Edition and the Professional Edition. The first one is open source and provides device management, data collection, processing and visualization.

	ThingsBoard Community Edition	ThingsBoard Professional Edition
Asset management & Data collection	✓	✓
End-user real-time dashboards	✓	✓
Data processing rules & alarms	✓	✓
Customizable rules, plugins, widgets	✓	✓
MQTT, HTTP, CoAP, OPC-UA transport	✓	✓
Integrations with BigData systems	✓	✓
NB-IoT, SigFox, LoRaWAN support	Basic	Advanced
Device, assets and customer groups	✗	✓
Multi-tenant configurable white-labeling	✗	✓
CSV/XLS data export	✗	✓
Platform Integrations	✗	✓

Figure 10. Comparison matrix between the two versions of thingsboard. [25]

5.2.2. Installation (cloud/own server)

ThingsBoard is designed to run and utilize the majority of hardware, from local Raspberry PI to powerful servers on the cloud. Ways to set up a ThingsBoard cluster include:

- Windows - install Thingsboard cluster on any pre-existing machines running Windows.
- Linux (Ubuntu & CentOS) - install ThingsBoard cluster on any pre-existing machines running Linux.
- Raspberry Pi 3 Model B (Raspbian Jessie) - install Cassandra and Thingboard server on a Raspberry Pi 3 Model B.
- Docker (Linux or Mac OS) - install a single-node ThingsBoard cluster on your Linux or Mac OS machine for development and testing.
- Docker (Windows) - install a single-node ThingsBoard cluster on your Windows machine for development and testing.

- AWC EC2 installation using AMIs - install a single-node ThingsBoard cluster using public AWS AMI.

5.2.3. Architecture

The general architecture of thingsboard is structured as the figure 11 shows:

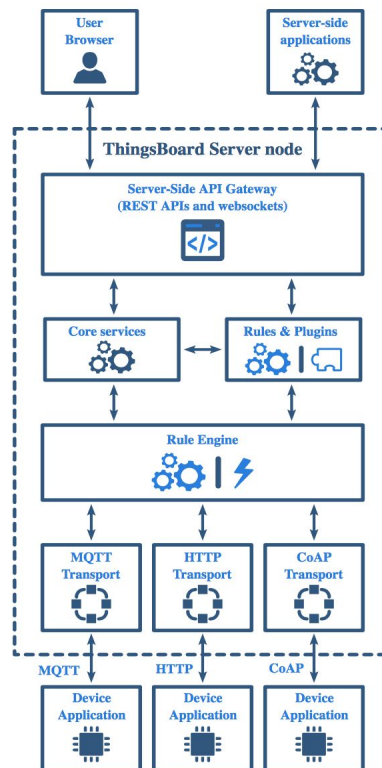


Figure 11. Thingsboard general architecture. [25]

- **Device Application**

ThingsBoard supports MQTT, CoAP and HTTP protocols for device connectivity. It is possible to plugin support of different protocols or customize existing implementations.

- **Rule Engine**

ThingsBoard Rule Engine allows to process messages from devices and trigger configurable processing modules called Plugins.

- **Core Services**

Management of entities.

- **Server-side API Gateway**

Websockets and REST API calls.

5.2.4. Modules (protocols and databases)

- **Protocols and Integrations:**

- **HTTP** : ThingsBoard server nodes act as an HTTP Server that supports both HTTP and HTTPS protocols.
- **MQTT** : ThingsBoard server nodes act as an MQTT Broker that supports QoS levels 0 (at most once) and 1 (at least once). It is possible to use an external MQTT broker (such as Mosquitto and HiveMQ) through the ThingsBoard IoT gateway.
- **OPC-UA** : ThingsBoard does not have native support and server nodes to integrate OPC-UA devices. The ThingsBoard IoT gateway can be used for this purpose.
- **CoAP**: CoAP protocol is UDP based, but similar to HTTP it uses the request/response model. CoAP observes options that allow users to subscribe to resources and receive notifications on resource changes. ThingsBoard server nodes act as a CoAP Server that supports both regular and observed requests.
- **Node-red**: ThingsBoard can be integrated with node-red as long as the user creates a custom function that will generate the proper format for sending via MQTT.

- **Databases:**

ThingsBoard is able to use SQL (PostgreSQL is recommended) or NoSQL (Cassandra) database layer. By default, ThingsBoard uses embedded HSQLDB instance which is very convenient for evaluation or development purposes.

5.2.5. External connection (in/out, APIs)

The ThingsBoard IoT Gateway is an open-source solution that allows you to integrate IoT devices connected to legacy and third-party systems with ThingsBoard.

As previously explained, it is possible to connect external OPC-UA devices, AWS cloud service or any type of external application through a MQTT Broker.

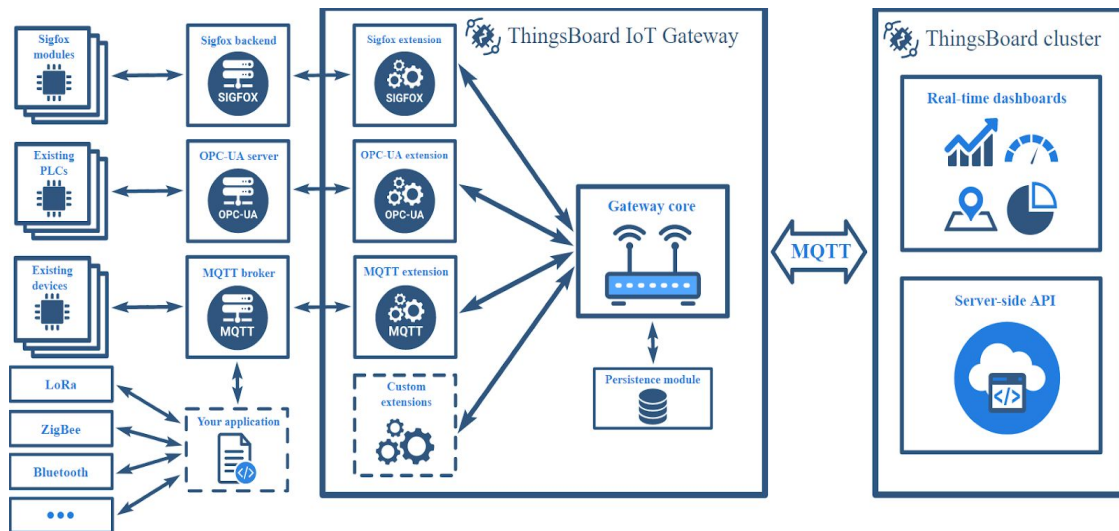


Figure 12. Thingsboard gateway architecture. [25]

- **Platform Integrations (PE) vs IoT Gateway (Free):**

Experienced ThingsBoard users may notice that functionality of Integrations features partially overlap with functionality of IoT Gateway. However, there are key differences between these two systems/features.

The first is that IoT Gateway is designed for local network deployments. On the other hand, Integrations are designed for server-to-server integration.

The second is that IoT Gateway is designed to support < 1000 devices, while Integrations are designed for high throughput, scalability and cluster deployments as part of the ThingsBoard server.

5.2.6. Analyzing/visualizing tools

ThingsBoard allows you to configure customizable IoT dashboards. Each IoT Dashboard may contain multiple dashboard widgets that visualize data from multiple IoT devices. Once IoT Dashboard is created, you may assign it to one of the customers of your IoT project.

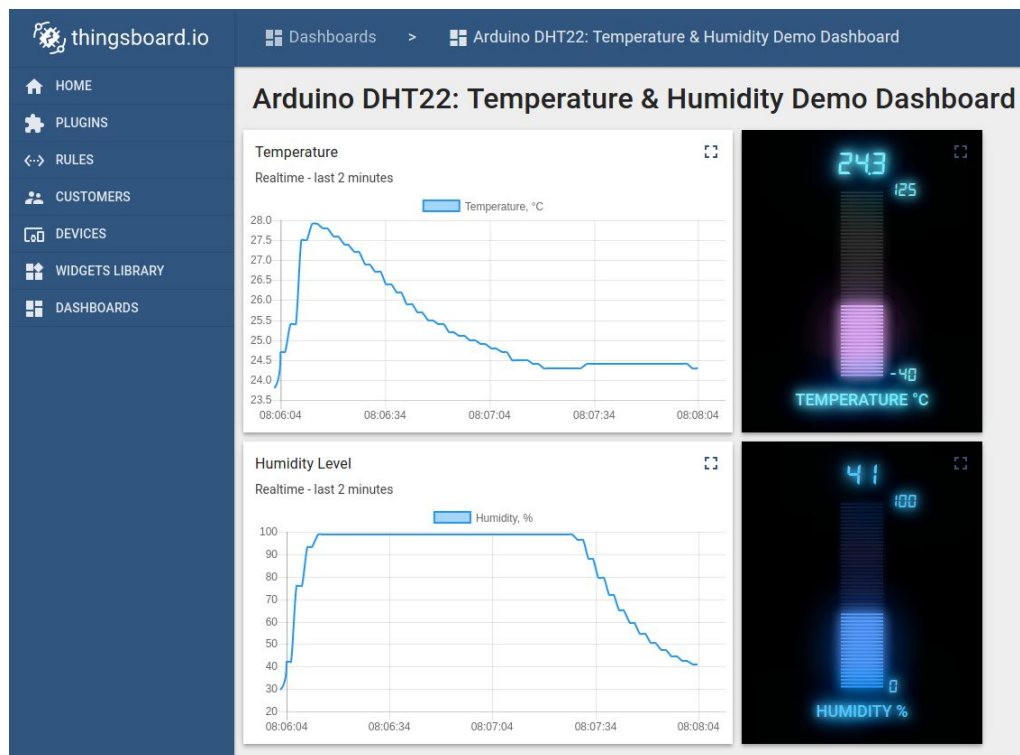


Figure 13. Thingsboard dashboard example. [25]

It is possible to customize visualization tools through the ThingsBoard Widget definitions or create a new type of visualization widget with HTML, CSS and Javascript.

5.2.7. Pros/Cons

Pros: Open source; Great documentation; Supports a good variety of protocols; Customizable dashboards and widgets; Great Users and Customers management; Multi Platform; Supports on-premises server or cloud service; Supports SQL and NoSQL databases; Mobile Friendly.

Cons: Most of Platform Integrations features are in the PE version (Azure, IBM Watson, AWS IoT and etc); Most of customizable features are in the PE version; Small Community; A lot of features still in development mode.

It is a great platform, the documentation is quite complete and it has a lot of features. Although, if we need to make often integrations with third-party systems and cloud services, the PE version should suit better.

5.3. Wemos D1 Mini Pro

For the tests and integration with ThingsBoard, the Wemos D1 Mini Pro board was used, with the board we could test multiple protocols, such as MQTT, CoAP and HTTP

and see how the ThingsBoard platform receives, manage, store and show the values received.

The Wemos D1 Mini Pro is a mini Wi-Fi board with 16MB flash, external antenna connector and built-in ceramic antenna based on ESP-8266EX. The Wemos D1 Mini Pro board is a suitable board for use in IoT (internet of things) projects and development of electronic projects in general. It has 11 digital I / O pins, 1 analog input pin, 16MB flash memory and supports interrupts, PWM, I2C and one-wire. Communication with the computer and power is done by the micro-USB connector [26].

ESP-8266EX offers a complete and self-contained Wi-Fi networking solution; it can be used to host the application or to offload WiFi networking functions from another application processor. When ESP8266EX hosts the application, it boots up directly from an external flash. It has integrated cache to improve the performance of the system in such applications [27].

Since Wemos is based on ESP-8266EX microcontrollers, Arduino libraries, tools and codes can be used without a problem.



Figure 14. Wemos D1 mini pro. [26]

5.4. DTH 11

As data examples, humidity and temperature were used, the data for these values were read by a DTH 11 sensor.

These sensors are very basic and slow, but are great for hobbyists who want to do some basic data logging. The DHT sensors are made of two parts, a capacitive humidity sensor and a thermistor. There is also a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity. The digital signal is fairly easy to read using any microcontroller [28].

Its specifications are: Ultra low cost ; 3 to 5V power and I/O ; 2.5mA max current use during conversion (while requesting data) ; Good for 20-80% humidity readings with 5% accuracy ; Good for 0-50°C temperature readings $\pm 2^\circ\text{C}$ accuracy ; No more than 1 Hz sampling rate (once every second) ; Body size 15.5mm x 12mm x 5.5mm 4 pins with 0.1" spacing.

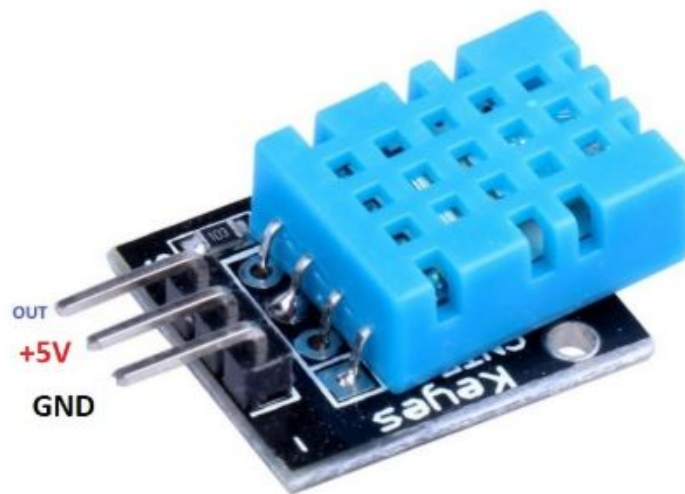


Figure 15.DTH 11 [28].

5.5. PlatformIO

PlatformIO is an open source ecosystem for IoT development. Cross-platform building system and unified debugger. Remote unit testing and firmware updates.

Some features such as autocomplete coding, management of libraries, debugging and customizable shortcuts, themes and styles make PlatformIO a better alternative for the usual Arduino IDE or any other IDE used to code firmwares or embedded software.

It supports the following Platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Freescale Kinetis, Intel ARC32, Lattice iCE40, Maxim 32, Microchip

PIC32, Nordic nRF51, Nordic nRF52, NXP LPC, Silicon Labs EFM32, ST STM32, Teensy, TI MSP430, TI Tiva, WIZNet W7500.

And the following Frameworks: Arduino, ARTIK SDK, CMSIS, Energia, ESP-IDF, libOpenCM3, mbed, Pumbaa, Simba, SPL, STM32Cube, WiringPi [29].

PlatformIO was used in this project to code for Wemos D1 board.

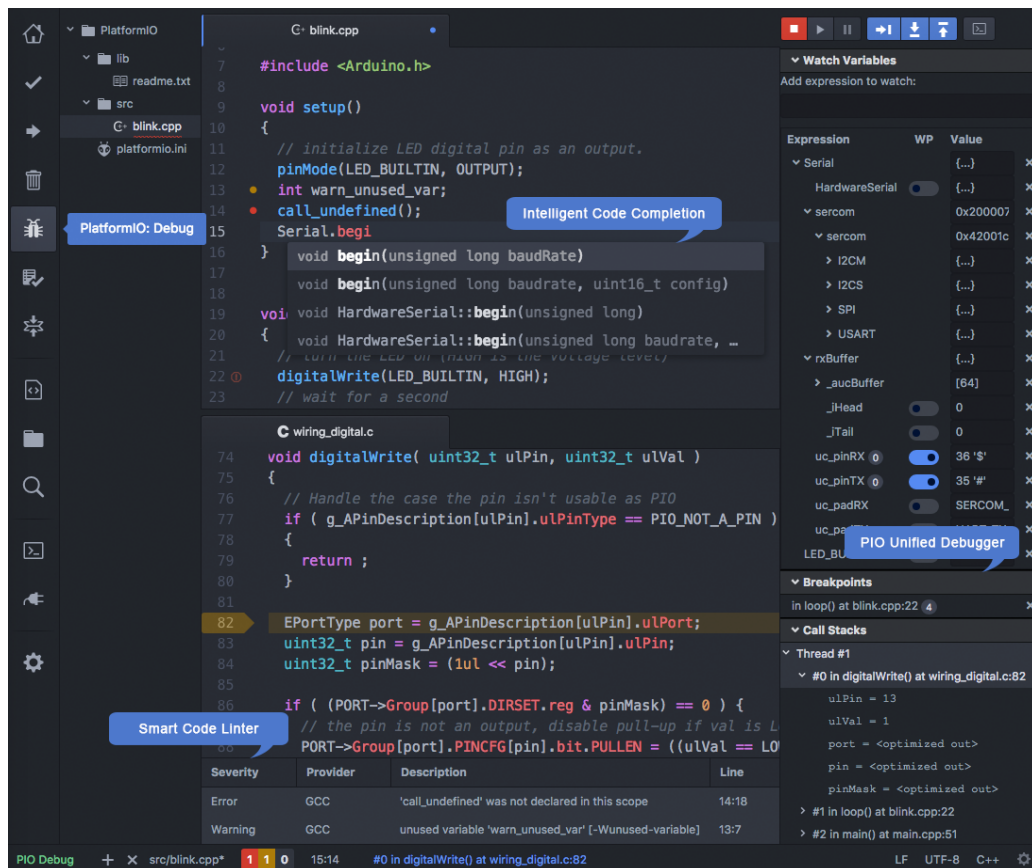


Figure 16. PlatformIO IDE [29].

6. Experimental Procedures and Results

All the tests, procedures, approaches and results executed in this project are going to be presented in this section. Since this paper proposes the validation, study and tests of an IoT platform it is difficult to separate the results from the procedures, once that our results are the success or failure of the tests itself.

6.1. Installation of the Thingsboard Platform (Azure)

An Ubuntu server 16.04.4 LTS virtual machine was installed in Azure Cloud Service, in this server the Thingsboard cluster was installed and configured.

First, we created a resource in Azure Portal, this resource was the ubuntu server, the Standard D1 v2 (1 vcpus, 3.5 GB memory) machine was chosen.

Virtual Machine Overview:

- System: Ubuntu Server 16.04.4 LTS
- Computer Name: ThingsBoardUbuntuVM
- Public IP: 52.178.138.16
- DNS name: thingsboardhamk.northeurope.cloudapp.azure.com
- Thingsboard Address:

thingsboardhamk.northeurope.cloudapp.azure.com:8080

- Cassandra-web: thingsboardhamk.northeurope.cloudapp.azure.com:3000

Once the virtual machine was up and running, we accessed it through SSH, using the admin user and password.

Once the server is properly configured, the Thingsboard cluster can be installed; for that, we need to install Cassandra, Thingsboard and Cassandra-Web.

In order to install Cassandra the following link was used <https://www.rosehosting.com/blog/how-to-install-apache-cassandra-on-ubuntu-16-04/>.

To install Thingsboard the <https://thingsboard.io/docs/user-guide/install/linux/> tutorial was used, during the installation we chose the external database configuration

option, with Cassandra. Note, before installing the platform, the `thingsboard/conf/thingsboard.yml` needs to be properly configured to work with Cassandra as the link shows. After that, the command `sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo` can run, this will install and start the Thingsboard platform on port 8080.

The Cassandra Web is an interface for Apache Cassandra with AngularJS and server-sent events. This tool can make interactions with the thingsboard database easier, since we will not be forced to use command lines. To install the tool, just follow the link <https://github.com/avalanche123/cassandra-web>. Note that the Cassandra Web may not work properly when dealing with a large amount of data if any issue appears, test accessing Cassandra through SSH.

6.1.1. Using Docker

As mentioned in Section 4.8 Docker is a project and application that provides the containerization concept, where it is possible to run applications as portable, self-sufficient and on any cloud or on-premises. The task consisted of using the Docker on Azure and installing ThingsBoard in this environment. Then after, it makes possible to generate a docker image with all the settings to possibly run it on any server.

It was identified in many forms to do it, as many problems too, which will be described below.

- **Installing Through Docker Community Edition (CE) for Azure:**

Tutorial Link:

<https://docs.docker.com/docker-for-azure/#docker-enterprise-edition-ee-for-azure>

Problem: To install the Docker CE for Azure within a Linux VM a Service Principal is needed. The Service Principal generates an AD App ID; AD App Secret; AD Tenant. All three pieces of informations are required for the setup of Docker CE.

Error: Running the Service Principal shows the Insufficient privileges to complete an operational error.

```

127.0.0.0/8
Live Restore Enabled: false

T-800:~ eduardobitencourt1$ docker pull docker4x/create-sp-azure:latest
latest: Pulling from docker4x/create-sp-azure
6f821164d5b7: Pull complete
a9c2ca1a583b: Pull complete
e370ebf47208: Pull complete
c773d92d70b7: Pull complete
34b0126a3b38: Pull complete
f0c4990ae9fd: Pull complete
Digest: sha256:08ca27cc5e8791e296e01f687a77f1591ae1e9d6984fc05cd7b884e85fd06c47
Status: Downloaded newer image for docker4x/create-sp-azure:latest
T-800:~ eduardobitencourt1$ docker run -ti docker4x/create-sp-azure spl
[info] Executing command login
[info] To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the
code C68WHGLWS to authenticate.
[info] Added subscription Älykkäät palvelut 772600 -Turo Nylund
[info] Setting subscription "Älykkäät palvelut 772600 -Turo Nylund" as default
+
[info] login command OK
The following subscriptions were retrieved from your Azure account
1) fa58dc79-671f-4192-99c7-e5c83eb21acb:Älykkäät_palvelut_772600_-Turo_Nylund
Please select the subscription option number to use for Docker swarm resources: 1
Using subscription fa58dc79-671f-4192-99c7-e5c83eb21acb
[info] Executing command account set
[info] Setting subscription to "Älykkäät palvelut 772600 -Turo Nylund" with id "fa58dc79-671f-4192-99c
7-e5c83eb21acb".
[info] Changes saved
[info] account set command OK
Creating AD application spl
error: {"odata.error":{"code":"Authorization_RequestDenied","message":{"lang":"en","value":"Insufficient
privileges to complete the operation."}}}
error: Error information has been recorded to /root/.azure/azure.err
error: ad app create command failed

Created AD application, APP_ID=
Cannot create application or determine application id.
T-800:~ eduardobitencourt1$ █

```

Figure 17. Docker Installation Error 1.

Workaround: Access to an Azure account with admin privileges.

- **Installing Through Azure Container Service:**

Tutorial Link:

<https://cloudify.co/2016/11/22/step-by-step-guide-deploying-docker-swarm-with-azure-container-service.html>

Error: When deploying the Azure Container service: The subscription is not registered to use namespace "Microsoft.ContainerService".

The screenshot shows the 'ThingsBoard - Deployments' interface. A table lists three deployments. The first deployment, 'microsoft.acs-20180412122423', is marked as 'Failed (Error details)'. The error details panel on the right provides the following information:

ERROR TYPE
At least one resource deployment operation failed. Please list deployment operations for details. Please see https://aka.ms/arm-debug for usage details. (Code: DeploymentFailed)

ERROR DETAILS

The subscription is not registered to use namespace 'Microsoft.ContainerService'. See https://aka.ms/rps-not-found for how to register subscriptions. (Code: MissingSubscriptionRegistration)

- The subscription is not registered to use namespace 'Microsoft.ContainerService'. See https://aka.ms/rps-not-found for how to register subscriptions. (Code: MissingSubscriptionRegistration, Target: Microsoft.ContainerService)

Figure 18. Docker Installation Error 2.

Cause: These errors show for one of three reasons: The resource provider has not been registered for your subscription; The API version not supported for the resource type; Location not supported for the resource type.

Workarounds:

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-register-provider-errors>

• Installing through Docker for Azure CE VM Resource

First try:

Steps:

1. It was installed by adding a new resource, searching as 'Docker CE'.
2. It was chosen as a virtual machine with HDD disc type with 3.5 GB of memory.

Error: "The resource operation completed with terminal provisioning state 'Failed'. (Code: ResourceDeploymentFailure)"

- OS Provisioning failed for VM 'thingsboarddocker' due to an internal error. (Code: OSProvisioningInternalError)".

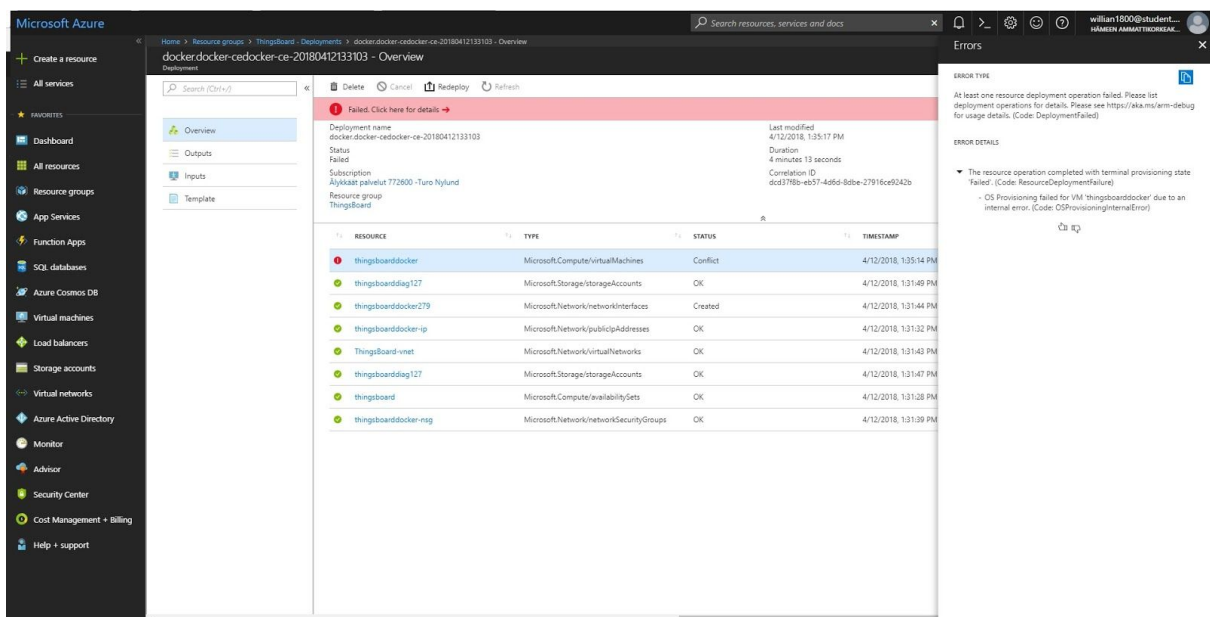


Figure 19. Docker Installation Error 3.

Second try:

Steps:

1. It was installed by adding a new resource, searchin as 'Docker CE'.
2. It was chosen as a virtual machine with HDD disc type with 4 GB of memory.

Error: Again "OSProvisioningInternalError" was a problem.

Error cause: "A provisioning failure happens when the OS image fails to load either due to incorrect preparatory steps or because of selecting the wrong settings during the image capture from the portal."

- **Alternative Approach 1: Installing a clean Ubuntu VM and deploying ThingsBoard Docker version in this environment:**

Creating the virtual machine:

<https://azure.microsoft.com/pt-br/resources/videos/create-linux-virtual-machine/>

Installing docker for ubuntu;

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

Installing docker compose for ubuntu:

<https://docs.docker.com/compose/install/>

Installing Thingsboard on ubuntu with docker:

<https://thingsboard.io/docs/user-guide/install/docker/>

Problem: It was not possible to access the ThingsBoard portal through a server inside the VM on Azure. Maybe an IP and port configuration is necessary to use it.

- **Alternative Approach 2: Installing a clean Ubuntu VM with a Ubuntu Docker environment where ThingsBoard was installed:**

Creating the virtual machine:

<https://azure.microsoft.com/pt-br/resources/videos/create-linux-virtual-machine/>

Installing docker for ubuntu:

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

Command line steps:

- `docker run -i -t -p 8080:8080 -p 1883:1883 -p 5683:5683/udp ubuntu:16.04 /bin/bash`

This line is responsible for creating a container with Ubuntu and connecting the mainly ThingsBoard ports with the container host ports. That will generate the container with an Ubuntu SO and open it to be able to prepare all the environment to support the ThingsBoard application within.

- `apt-get update`

To update the “new machine” hosted by Ubuntu VM.

- `apt-get install software-properties-common`

To be able to use the ‘add-apt-repository’ command.

- `add-apt-repository ppa:webupd8team/java`

Add a new repository to make it possible to install the Java JDK, a prerequisite for ThingsBoard installation.

- `apt-get update`
- `add-apt-repository ppa:webupd8team/java`

To install the Java JDK 8.

- Installing ThingsBoard for ubuntu:
<https://thingsboard.io/docs/user-guide/install/linux/>
- To commit and generate a docker image:

`docker commit 'CONTAINER ID' hamk/thingsboard-ubuntu:1.0 (example)`
Tutorial: <https://ropenscilabs.github.io/r-docker-tutorial/04-Dockerhub.html>

Observations: It was not possible to install Cassandra in this environment. It only worked installing without an external database installation.

6.2. Configuration of Thingsboard (Creating users, devices and dashboards)

Thingsboard has three levels of users: Admin, Tenant and Customer users, more on this here: <https://thingsboard.io/docs/user-guide/ui/users/>.

Since we installed using the option `--loadDemo`, a lot of demo data has already been created, like some demo accounts, devices, dashboards, rules, plugins and alarms.

In the first test a device was created, called DTH 11 DEMO DEVICE, once the device is created, we could get its access token and id, those attributes are going to be used to send the data to the platform (Section 6.3). After this dashboard was created, this dashboard shows the humidity and temperature charts and values, in real time visualization.

In this dashboard, a gpio (General Purpose Input/Output) control was also created in order to control the LED available in the device.

All these steps were based on this tutorial:

<https://thingsboard.io/docs/samples/esp8266/temperature/>.

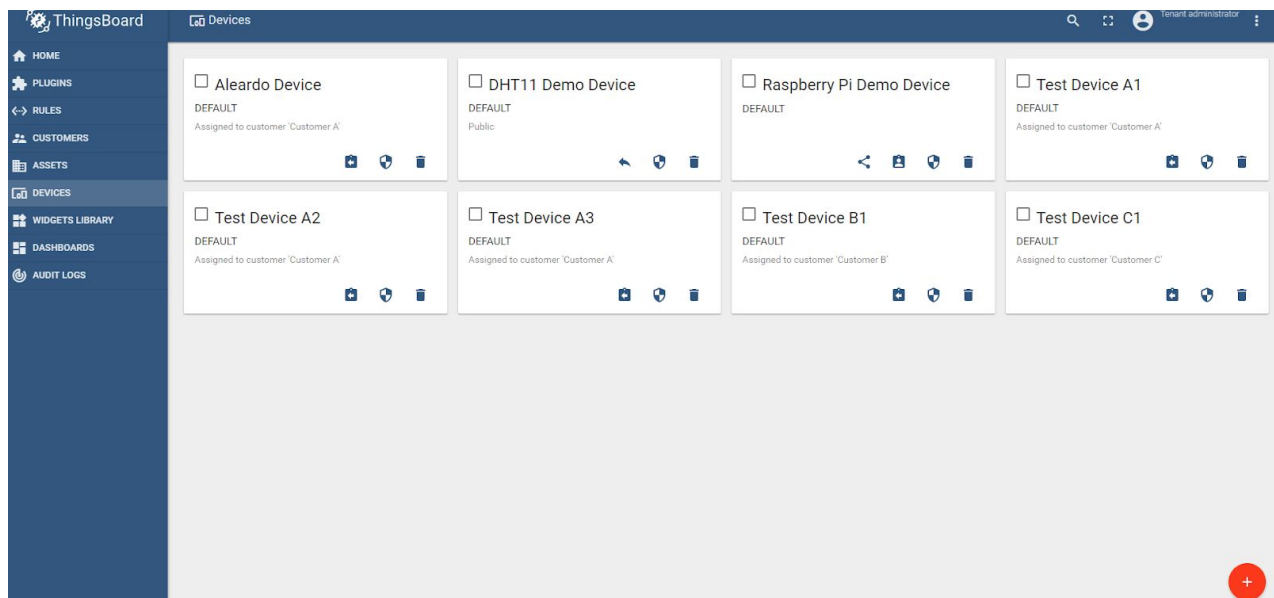


Figure 20. Thingsboard devices panel.

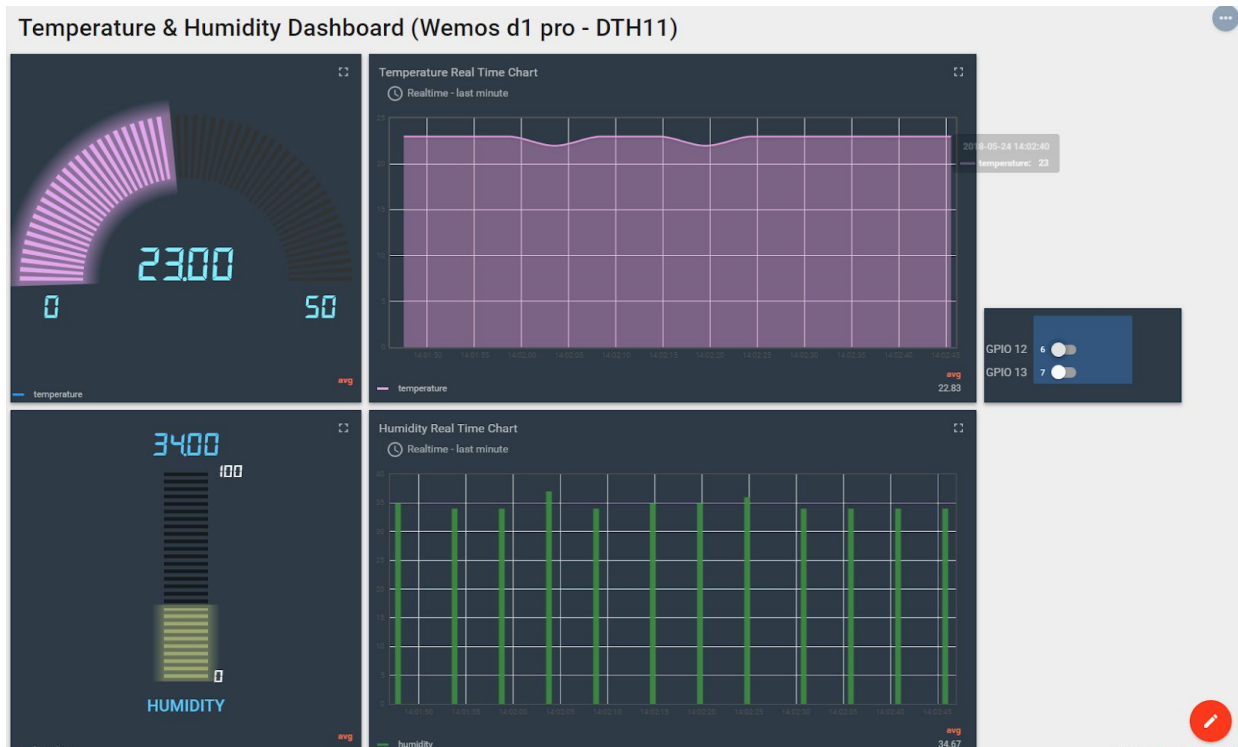


Figure 21. Thingsboard dashboards of the device DTH11.

6.3. Getting temperature and humidity from devices

The circuit was created with the board Wemos D1 (Section 5.3) and DTH sensor (Section 5.4). The purpose of this circuit was to simulate a real IoT device that would keep uploading data to the platform every second, this data was temperature and humidity. Also, a LED was used as an actuator, in order to validate the capability of sending commands from Thingsboard to devices, for example switching a LED on and off, the circuit schema can be seen in figure 22 and the real device in figure 23.

The program uploaded to the board was developed using platformIO IDE (Section 5.5).

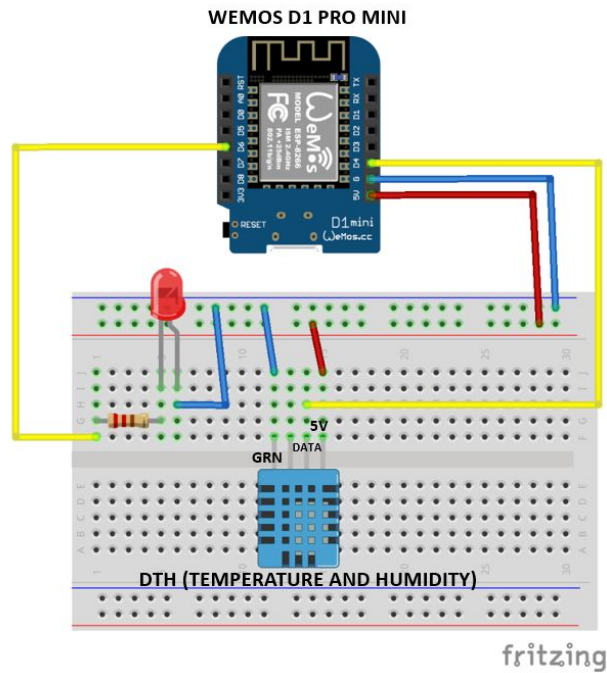


Figure 22. IoT device scheme that reads temperature and humidity.

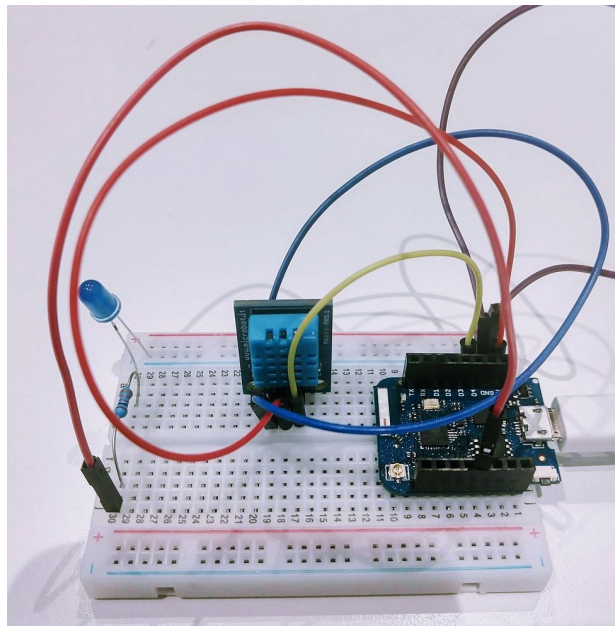


Figure 23. IoT device that reads temperature and humidity.

6.3.1. Using MQTT

As previously shown on Section 4.3.1, MQTT is an IoT protocol used in the application layer. To use MQTT with Wemos we first need to download and import some

Arduino Libraries which are going to be used in the project, this can be done easily with platformIO IDE. The libraries required are:

- **ArduinoJson:** Used to work with Json data format.
- **PubSubClient:** MQTT library that contains all the necessary methods and functions for publishing data to a subscriber through an MQTT broker.
- **ESP8266Wifi:** This is used to connect the Wemos board in some WIFI network.

To make the integration between the device and the Thingsboard platform, 3 attributes are important, the server address (address where thingsboard is running), the device ID (Id that can be copied from Thingsboard platform by just clicking into the device that was created in Section 6.2) and the device access Token (Define the device Token that we will use, copy the token from Thingsboard device panel).

The Wemos complete code for this project can be seen in the Appendices Section, this test was executed based on this Thingsboard tutorial:

<https://thingsboard.io/docs/samples/esp8266/temperature/>.

6.3.2. Using HTTP

As described in Section 4.3.2 HTTP is a general-purpose network protocol that can be used in IoT applications. HTTP protocol is TCP based and uses the request/response model. ThingsBoard server nodes act as an HTTP Server that supports both HTTP and HTTPS protocols.

The same device that was used with MQTT will be used with HTTP. Nothing on the Thingsboard platform changes, from the moment that the device was created is already listening for MQTT, HTTP or CoAP. What changes is the code running on the Wemos board, since it uses different Arduino libraries and methods in order to send data through HTTP, but it is very similar to MQTT. The code using HTTP can be found in the Appendices Section of this paper. More about Thingsboard working with HTTP: <https://thingsboard.io/docs/reference/http-api/>.

6.4. Monitoring and managing the Thingsboard database

Thingsboard does not allow database management through its graphical interface, so we need to connect directly with the database service that has been used with the platform. In this case, we are using NoSQL Cassandra for performance purposes. The Thingsboard Cassandra database can be accessed with two different ways:

- Alternative 1: The server and the Cassandra instance can be accessed through SSH. Command (mac or linux): `sudo ssh useradm@thingsboardserveraddress`. Once accessing the server, just use the command “`cqlsh`”. This will open the `cql` terminal, which you can use to make queries and operations in the Cassandra cluster.
- Alternative 2: There is an UI tool called Cassandra Web (<https://github.com/avalanche123/cassandra-web>), where `cql` commands and queries can be executed, with this tool we can see the tables, rows, keyspaces and the whole DB in an easier way. Note: The Cassandra Web may not work properly when dealing with a large amount of data if any issue appears, test accessing Cassandra through SSH (first alternative).

Some examples of queries are shown below:

- The `ts_kv_cf`: Table that stores all the telemetry data received from all devices.
- `entity_id`: Device id (can be found in thingsboard device panel).
- `key` = Attribute that indicates the name of the data.

Query a specific value from a specific device:

```
SELECT * FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' ALLOW
FILTERING;
```

Creating functions to get date/time intervals in cassandra:

Cassandra does not support arithmetic operations (version 3.11), so it is impossible to do something like “`SELECT * FROM thingsboard.ts_kv_cf WHERE ts <= now() - 7d`”, to be able to get for example the last 7 days of data, we need to create an UDF function in Cassandra, using Java.

First, turn on `enable_user_defined_functions: true` in the `conf/cassandra.yaml` file. Then create the function, for that, paste and execute the following query in the cql editor (SSH or Cassandra Web):

```
CREATE FUNCTION IF NOT EXISTS
thingsboard.timeAgoMinsOrDaysOrMonths(minutes int, days int, months int)
CALLED ON NULL INPUT RETURNS timestamp LANGUAGE java AS '
```

```
long now = System.currentTimeMillis();
if (minutes != null)
    return new Date(now - (minutes.intValue() * 60 * 1000));
else if (days != null)
    return new Date(now - (days.intValue() * 86400000));
else if (months != null)
    return new Date(now - (months.intValue() * 2629746000L));
else
    return new Date(now);
;
;
```

<https://stackoverflow.com/questions/46135576/how-to-do-arithmetics-with-datetime-in-cql?rq=1>

Note: The function has 3 parameters, you need to decide if you want to use an interval in minutes, days or months, once you decide the other parameters should be passed as null values. `timeAgoMinsOrDaysOrMonths(minutes,days,months)`.

This query returns the last 7 days of temperature data from a device:

```
SELECT * FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' and ts >=
timeAgoMinsOrDaysOrMonths(null,7,null) ALLOW FILTERING;
```

The same query, but it returns the last 1 month of temperature data from a device:

```
SELECT * FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' and ts >=
timeAgoMinsOrDaysOrMonths(null,null,1) ALLOW FILTERING;
```

The same query, but it returns the last 10 minutes of temperature data from a device:

```
SELECT * FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' and ts >=
timeAgoMinsOrDaysOrMonths(10,null,null) ALLOW FILTERING;
```

This new function allow us to execute clean routines in the database, for instance, if we want to erase the last seven days of data, we can execute:

```
DELETE FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' and partition IN
(1525132800000, 1522540800000,1525132800000) and entity_type = 'DEVICE'
and ts >= timeAgoMinsOrDaysOrMonths(null,7,null);
```

If we want to keep just the last 7 days of data and erase everything else:

```
DELETE FROM thingsboard.ts_kv_cf WHERE entity_id =
f35bcef0-3ef5-11e8-8fd0-cb746776d0c2 and key = 'temperature' and partition IN
(1525132800000, 1522540800000, 1525132800000) and entity_type = 'DEVICE'
and ts <= timeAgoMinsOrDaysOrMonths(null,7,null);
```

Note: the partition and the entity_type must be used to delete actions on Cassandra (because they form the primary key of the entity). To see all the partitions and be able to use them in the IN clause, you need to query the ts_kv_partitions_cf table. For that, use the Cassandra Web (serveraddress:3000), go to thingsboard/ts_kv_partitions_cf and then copy all the partitions of the values that you want to delete, for example, humidity and then put them inside the IN clause.

6.5. Thingsboard Rest API

Thingsboard can act as a web services RESTful and this allows us to make REST calls and get the data back from the platform, this would be great if we want to use all the data collected by the platform in a machine learning algorithm for example.

To make requests, some authentications are needed. Below the steps are described if one wants to use the Rest API functions in Thingsboard.

- Open thingsboard.yml file and change security.jwt.refreshTokenExpTime property to give a longer expiration time for the token (the default is just 15min, give it like 6 months).
- Use the tool Curl (<https://curl.haxx.se/>) to generate the JWT token.
- Use Thingsboard swagger to authorize the token that was generated (<http://serveraddress:8080/swagger-ui.html>).

Once those steps are finished, we are able to make some requests examples, for instance, fetch list of latest values for particular entity type and entity id using GET request:

```
curl -v -X GET
http://serveraddress:8080/api/plugins/telemetry/DEVICE/ac8e6020-ae99-11e6-b9b
d-2b15845ada4e/values/timeseries?keys=humidity,temperature \
--header "Content-Type:application/json" \
--header "Bearer X-Authorization: $JWT_TOKEN (The token generated)"
```

This will return the Json body with all the data requested. This request example was executed through curl tool, but other solutions can be applied as well, such as postman (<https://www.getpostman.com/>) or any other tool to make RESTful requests.

Link with the token generation and swagger authorization examples:

<https://thingsboard.io/docs/reference/rest-api/>

Link with REST requests examples:

<https://thingsboard.io/docs/user-guide/telemetry/>

6.6. Embedding widget and dashboard to an external web page.

It is possible to show dashboards from thingsboard into an external web page. The process consists of one embedded html (thingsboard dashboard) that will be displayed in an iframe tag (external page).

In order to do that, enter in Thingsboard with a Tenant account, go to devices and select the device related to the dashboard that is going to be shown in the HTML page and make the device public, by clicking on the icon right at the bottom of the device card. After that, do the same with the device's dashboard, this will generate a link, copy this link and paste inside an iframe tag, then the tag can be displayed anywhere inside an HTML page, the figure 24 shows the process.

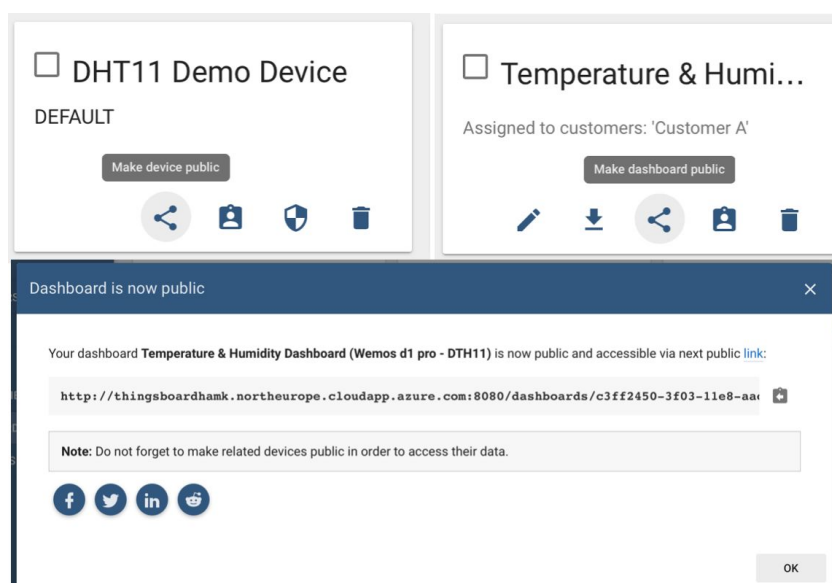


Figure 24. Making device and dashboard public in order to make them visible in an external web page.

6.7. Rules, Alarms and Plugins. Firing alert Emails.

In order to send Email as an alarm first an email plugin needs to be configured in Thingsboard, see more about this configuration here:

<https://thingsboard.io/docs/samples/alarms/mail/>. Although SendGrid was used as an email server, any other service or server can be used without any problem.

The rule engine allows you to process messages from devices with a configurable set of rules. This rule engine lets us create some very useful functions, for instance: send an email when a device attributes change; Create an alarm when telemetry value exceeds a certain threshold; Forward telemetry data to external RESTful server. And a lot more with other rules and plugins.

ThingsBoard Rule consists of three main components: Filters, Processor and Action. Depending on implementation, each component may require certain configuration before it can be used. In order to configure rules, you need to specify at least one filter and one action. Rule processors are optional, more on this here:

<https://thingsboard.io/docs/user-guide/rule-engine/>.

To create a rule that will fire the alarm, go to Rule panel, create a new one, and add the filters, you can filter data with attributes (server or client), telemetry data and message. For this example one filter was created, to filter the telemetry data from a specific device, in this case, the data was temperature, the filter is if the temperature reaches 27c/higher or 20c/lower, the alarm is fired.

After the filters, the alarm processor needs to be configured. In this case, it will contain the email format that will be sent.

The rule action needs to be chosen, where Send Mail Action is selected, this will use the email Plugin configured with SendGrid server. figure 25 shows these steps and configurations.

After this, click the Activate Rule button on the rule card and then go to the device that is wanted to hold this rule, enter in device settings and add a new relation, select the rule created and it is done; once the device receives a temperature out of the levels desired, the alarm is fired. This procedure was executed based on this tutorial:

<https://thingsboard.io/docs/samples/alarms/basic-rules/>.

The image displays three configuration windows and an email notification. The 'Filter' window shows a rule named 'Temperature filter' with a JavaScript condition: `typeof temperature !== 'undefined' && (temperature <= 20 || temperature >= 20)`. The 'Processor' window shows an 'Alarm Processor' with templates for the alarm ID and body. The 'Plugin action' window shows a 'Send Mail Action' configured with the sender 'eduardo.n.bitencourt@gmail.com' and recipients 'eduardo.n.bitencourt@gmail.com, anjos_willian@hotmail.com'. Below these, an email notification is shown with the subject '[2018-04-24 11:36] Room 10 Temperature alarm detected!' and the body '[2018-04-24 11:36:26] Room 10 HVAC WARNING!!! Temperature – 27.0 (°C)'.

Filter

Name* Temperature filter Type* Device Telemetry Filter

Filter*

```
1 C
2   typeof temperature !== 'undefined'
3   && (temperature <= 20 || temperature >= 20)
4 }
```

SAVE CANCEL

Processor

Name* Alarm Processor Type* (Deprecated) Alarm Deduplicatio...

Alarm id template*

[Date.get('yyyy-MM-dd HH:mm')] \$ss.get('Zoneld') Temperature alarm det

Alarm body template*

[Date.get('yyyy-MM-dd HH:mm:ss')] \$ss.get('Zoneld') HVAC
WARNING!!!
Temperature - \$temperature.valueAsString ('°C').

SAVE CANCEL

Plugin action

Name* Send Mail Action Type* Send Mail Action

Send flag (empty or 'isNewAlarm', 'isExistingAlarm', 'isClearedAlarm', 'isNewOrClearedAlarm')
isNewAlarm

From template*
eduardo.n.bitencourt@gmail.com

To template*
eduardo.n.bitencourt@gmail.com, anjos_willian@hotmail.com

CC template

BCC template

Subject template*
\$alarmId

Body template*
\$alarmBody

SAVE CANCEL

[2018-04-24 11:36] Room 10 Temperature alarm detected! Entrada x

eduardo.n.bitencourt@gmail.com por sendgrid.net
para anjos_willian

[2018-04-24 11:36:26] Room 10 HVAC WARNING!!! Temperature – 27.0 (°C).

Figure 25. Rule being configured and email alarm being received.

6.8. Thingsboard Gateway

The ThingsBoard IoT Gateway is an open-source solution that allows you to integrate IoT devices connected to legacy and third-party systems with ThingsBoard. The Gateway provides simple integration APIs and encapsulates common Thingsboard-related tasks: device provisioning, local data persistence and delivery, message converters/adapters, and more. As an application developer, you are able to choose Python, Go, C/C++, and other languages and connect to Thingsboard Gateway

through external MQTT broker or OPC-UA server. Devices that support other protocols may be connected to gateway by implementing custom extensions [31].

Thingsboard IoT Gateway provides following features:

- OPC-UA extension to collect data from devices that are connected to OPC-UA servers.
- MQTT extension to collect data that is published to external MQTT brokers.
- Persistence of collected data to guarantee data delivery in case of network and hardware failures.
- Automatic reconnecting to Thingsboard clusters.
- Simple yet powerful mapping of incoming data and messages in a unified format.

To test the ThingsBoard IoT Gateway it was installed in an Ubuntu VM following this tutorial <https://thingsboard.io/docs/iot-gateway/install/linux/>. The next steps were to create the gateway as a device on ThingsBoard WEB UI, provisioning a credential to be able to use it, configuring and testing the connection by an external MQTT broker and OPC-UA server. This steps were based on this tutorial to get start with ThingsBoard Gateway <https://thingsboard.io/docs/iot-gateway/getting-started/>.

6.8.1. MQTT External Broker

The Mosquitto MQTT broker was used to test the MQTT external broker connection through ThingsBoard IoT Gateway and ThingsBoard. Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers [32]. It was installed and configured based on this tutorial <http://www.steves-internet-guide.com/install-mosquitto-broker/>.

Both Mosquitto and ThingsBoard Gateway use the same port (1883) for MQTT service by default. So to test it in the same host was necessary to change the Mosquitto running port to 1884. The following command was used:

- `mosquitto -p 1884`

To configure the connection of ThingsBoard Gateway with MQTT Broker is possible to use a file configuration and a GUI configuration through the ThingsBoard portal. The file configuration was used in this test. So the next step was to edit the 'mqtt-config.json' on ThingsBoard Gateway installation folder to be able to listen to the right mqtt host and port. More details about the MQTT ThingsBoard extension configuration are on this link <https://thingsboard.io/docs/iot-gateway/mqtt/>. There, the mqtt broker port was changed from 1883 to 1884 as well.

After the Thingsboard Gateway service was restarted and a basic mapping example was created to simulate a device with sensors connected to this environment. All the steps and configurations are detailed through this link <https://thingsboard.io/docs/iot-gateway/getting-started/#step-8-connect-to-external-mqtt-broker>.

6.8.2. OPC-UA

For the OPC-UA example the KEPServerEX for Windows was installed on a Windows machine. KEPServerEX is a connectivity platform that provides a single source of industrial automation data to all of your applications. The platform design allows users to connect, manage, monitor and control diverse automation devices and software applications through a user interface [57]. After the installation of this server, the sample OPC-UA tag values of it were transformed into ThingsBoard attributes and telemetry. In that manner it was possible to visualize data using the ThingsBoard widgets. Those steps were done using the ThingsBoard documentation tutorial <https://thingsboard.io/docs/iot-gateway/getting-started/#step-9-connect-to-external-opc-ua-server>.

Observations: It was identified a problem to map all the telemetry device fields between the KEPServer and ThingsBoard, where only the 'Tag1' attribute was mapped.

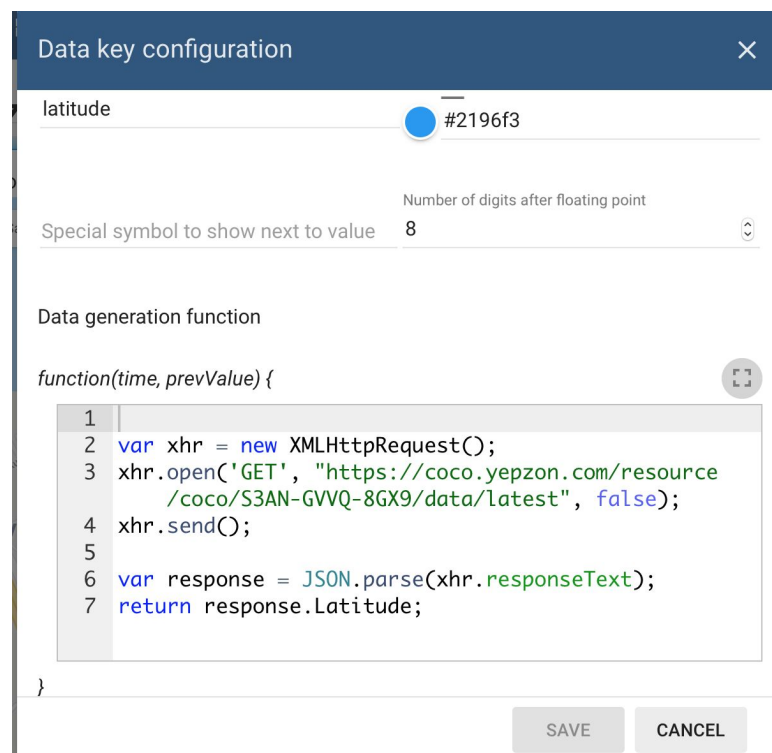
6.9. Integrating Thingsboard with Yepzon Asset (industrial tracker)

Yepzon Asset is an industrial tracker for cargo and asset management. It has its own platform for visualization and management of devices and data, but is limited when compared to Thingsboard, so a good approach is to integrate these two products.

Yepzon provides an integration API using RESTful methods, which can be used to get the data that the tracker sent to the platform.

Thingsboard provides map widgets for location, route and tracking visualization, so what needs to be done is get the data from Yepzon through their API and show the data in the map widgets.

In order to do that, the map/latest values widget needs to be created in one dashboard. When creating the Widget Thingsboard will ask for the datasource for the map, there are 2 options that can be chosen, from a device and from a javascript function, since there is no device for Yepzon because the data are pulled directly from their platform, the function option needs to choose. In this function the Yepzon API is going to be used for making the requests, each function will return one attribute from Yepzon's platform. Figure 26 shows the function used to pull latitude from the platform.



Data key configuration

latitude #2196f3

Number of digits after floating point: 8

Special symbol to show next to value:

Data generation function

```
function(time, prevValue) {
  1
  2 var xhr = new XMLHttpRequest();
  3 xhr.open('GET', "https://coco.yepzon.com/resource/coco/S3AN-GVVQ-8GX9/data/latest", false);
  4 xhr.send();
  5
  6 var response = JSON.parse(xhr.responseText);
  7 return response.Latitude;
}
```

SAVE CANCEL

Figure 26. Creating the javascript function to get Latitude from Yepzon Asset Device.

In order to show the last location of the tracker, latitude and longitude attributes are enough, but Yepzon provides temperature and humidity values as well, and these values can be shown in the map marker description, so to get these values, the same function can be used, just the return value needs to be changed to desirable attributes.

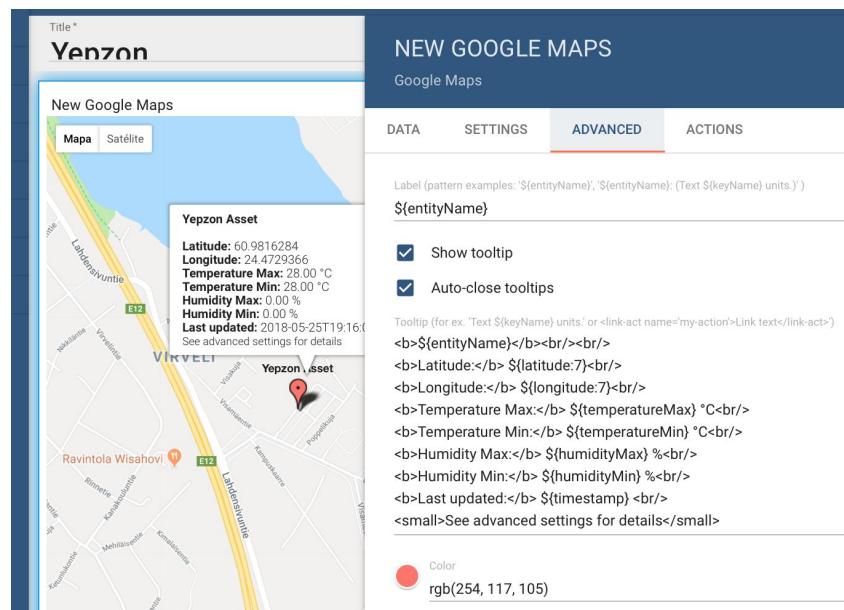


Figure 27. Yepzon data being displayed in thingsboard platform.

The full documentation of Yepzon API can be accessed here: <https://bit.ly/2GU5k1o>

Once the devices are all configured and integrated with the platform, they could be deployed to the bikes and the project could be validated and presented. To do this, a web page was created, containing information about the project and the Thingsboard widget map, to integrate the external page with the map, the method presented in Section 6.6. was used. Figure 27 shows the project being presented at an HAMK university event, in addition to the devices implanted on the bikes and the map of the Thingsboard platform being presented on a web page.



Figure 27. Yepzon project being presented.

7. Conclusion

Comparing the IoT Platforms, choosing the best option, studying, resuming your features and implanting some device solutions on it, made it possible to begin the validation of the ThingsBoard platform as a solution for an academic IoT environment. As shown it was possible to implement the tool on a cloud or on-premises server. There were a great variety of ways to deploy it, which showed that it is a tool with great adaptability.

The big quantity of IoT protocols that ThingsBoard supports and how easy it was to integrate with the developed solutions showed the great potential of this platform. The main integrations using HTTP, CoAP, and MQTT have worked well. Unfortunately, the OPC-UA implantation was not done in a real scenario, with a real OPC-UA device and server, but only the fact to support this and more other protocols through the ThingsBoard Gateway application maintains the ThingsBoard ahead of its rival platforms.

Around the data management the application worked well, storing data from devices with a NoSQL solution like Cassandra. The limitation here is that the platform does not have specific functions to manage the device's data, therefore some analytics and data management needs to be made independent from ThingsBoard.

The power of its visualization tools was proved as well. The platform has a vast quantity of widgets, which could be customized or new widgets could be created through HTML, CSS, and Javascript. It is possible to show the dashboards on a website through a responsive iFrame that works well on both computer and mobile screens.

Customization and the possibility of managing actions and events interconnected with the telemetry data of devices through the ThingsBoard Web UI showed it promissory. With it, it is possible to create rules, alarms and send emails based on specific data from a device.

About user and environment management, it was possible to use the platform to create users hierarchically that would help to implant the tool in employees, teachers, and students in academic contexts.

As described in Section 6.1.1, there were problems with the installation on Azure using Docker tool, more with Docker than the ThingsBoard platform, that required more

study. More special attention to integrating an OPC-UA real solution needs to be made as well. An adequate tool to do analytics work on specific data by the ThingsBoard DB needs to be seen. And to finalize, it needs to be implanted in a real server with many users, and devices, to validate the error handling, outliers and possible issues that can occur in some specific cases.

8. Acknowledgments

We would like to thank all the hospitality and support which we were received by our project coordinator Joni Kukkamäki, who helped us greatly with our introduction, adaptation, and follow-up both in the research project and in our stay in Hämeenlinna.

To Juha-Matti Torkkel for all the support and suggestions related to our research project.

To our project coordinator in Brazil Robson Costa for all the support, clarifications and ideas that helped us in the development of this paper.

To our another project coordinator in Brazil Vilson Heck Júnior for all the support with the beginning studies around this exchange program.

To all the Smart Services unit servers and colleagues who welcomed us with arms wide open and were always willing to help us with anything.

To HAMK and all its really welcoming structure, for all the support of laboratories and equipment needed for the project to be successfully developed.

And last but not least, to IFSC and this great project that gave us the opportunity to develop this exchange program.

9. References

- [1] ZANELLA, A. et al. Internet of things for smart cities. IEEE Internet of Things Journal, v. 1, n. 1, p. 22–32, Feb 2014. ISSN 2327-4662.
- [2] ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. Computer Networks, v. 54, n. 15, p. 2787 – 2805, 2010. ISSN 1389-1286. Available at: <<http://www.sciencedirect.com/science/article/pii/S1389128610001568>>.
- [3] GARTNER. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. 2017. Available at: <<https://www.gartner.com/newsroom/id/3598917>>.
- [4] KAUR, J.; KAUR, K. Internet of things: A review on technologies, architecture, challenges, applications, future trends. International Journal of Computer Network and Information Security(IJCNIS), v. 9, n. 4, p. 57–70, Apr 2017. ISSN 2074-9104.
- [5] KHAN, S. U.; ZAHEER, R.; KHAN, S. Future internet: The internet of things architecture, possible applications and key challenges. International Conference on Frontiers of Information Technology (FIT), v. 10, p. 257–260, 2012.
- [6] MQTT Specifications, 2014. [Online]. Available at: <<http://mqtt.org/documentation>>. [Accessed 24 April 2018].
- [7] MQTT Essentials Part 3: Client, Broker and Connection Establishment, 2015. [Online]. Available at: <<https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment>>. [Accessed 24 April 2018].
- [8] MQTT Protocol Tutorial: Step by step guide, Mosquitto and MQTT Security. [Online]. Available at: <<https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html>>. [Accessed 24 April 2018].
- [9] W3C. Hypertext Transfer Protocol – HTTP/1.1, 1999. [Online]. Available at: <<https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html#sec1>>. [Accessed 24 April 2018].
- [10] KELLOGG, T. Why HTTP won't work for IoT, 2014. [Online]. Available at: <<https://www.edn.com/electronics-blogs/eye-on-iot/4437056/Why-HTTP-Won-t-Work-for-IoT>>. [Accessed 7 May 2018].

- [11] IoT protocols List: IoT data and network protocols in 2018, 2018. [Online]. Available at: <<https://www.survivingwithandroid.com/2016/08/iot-protocols-list.html>>. [Accessed 24 April 2018].
- [12] MINERVA, R.; BIRU, A.; ROTONDI, D. Towards a definition of the internet of things (iot). IEEE Internet Initiative, n. 1, May 2015.
- [13] SHELBY, Z.; HARTKE, K.; BORMANN, C. Rfc7252 the constrained application protocol (coap). Internet Engineering Task Force (IETF), 2014. ISSN 2070-1721. Available at: <<https://tools.ietf.org/pdf/rfc7252.pdf>>.
- [14] OPC Foundation, "OPC Unified Architecture Specification", Parts 1–11, 2009. [Online]. Available at: <www.opcfoundation.org>. [Accessed 24 April 2018].
- [15] Brain, M. How Microcontrollers Work, 2000. [Online]. Available at: <<https://electronics.howstuffworks.com/microcontroller1.htm>>. [Accessed 2 May 2018].
- [16] STACKIFY. What Is Telemetry? How Telemetry Works, Benefits of Telemetry, Challenges, Tutorial, and More, 26 April 2017. [Online]. Available at: <<https://stackify.com/telemetry-tutorial/>>. [Accessed 03 May 2018].
- [17] G. Carutasui, M. A. Botezatui, C. Botezatu, M. Pirnau, "Cloud Computing and Windows Azure", Journal of Internet ECAI 2016 - International Conference – 8th Edition, pp.13, 2016. Available at: <https://www.researchgate.net/publication/305725698_Cloud_Computing_and_Windows_Azure>
- [18] J. Gong, P. Yue, H. Zhou, "Geoprocessing in the Microsoft Cloud computing platform - Azure", Commission IV, WG IV/4, 2010. Available at: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.9206&rep=rep1&type=pdf>>
- [19] R. Hecht and S. Jablonski, "NoSQL evaluation: a use case oriented survey," in Proceedings of the International Conference on Cloud and Service Computing (CSC '11), pp. 336–341, December 2011.
- [20] R. Aniceto, R. Xavier, V. Guimarães, F. Hondo, M. Holanda and S. Lifschitz, "Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency" International Journal of Genomics, Article 502795, May 2015.
- [21] E. Hewitt, "Cassandra: The Definitive Guide" O'Reilly Media, Inc. 2011.

- [22] Flow.ci. Introduction to Containers: Concept, Pros and Cons, Orchestration, Docker, and Other Alternatives, 2016 [Online]. Available at: <<https://medium.com/flow-ci/introduction-to-containers-concept-pros-and-cons-orchestration-docker-and-other-alternatives-9a2f1b61132c>>. [Accessed 26 April 2018].
- [23] Linthicum, D. Three apps in which Docker containers really shine, 2015 [Online]. Available at: <<https://searchitoperations.techtarget.com/tip/Three-apps-in-which-Docker-containers-really-shine>>. [Accessed 26 April 2018].
- [24] H2S Media Team. 9 Best & Top Open source IoT Platforms To Develop the IOT Projects, 2017 [Online]. Available at: <<https://www.how2shout.com/tools/best-opensource-iot-platforms-develop-iot-projects.html>>. [Accessed 26 April 2018].
- [25] ThingsBoard Documentation, 2017 [Online]. Available at: <<https://thingsboard.io/docs/>>. [Accessed 26 April 2018].
- [26] Wemos Documentation V1.1.0 (current version), 2018 [Online]. Available at: <https://wiki.wemos.cc/products:d1:d1_mini_pro> [Accessed 26 April 2018].
- [27] ESP8266EX Datasheet Version 4.3, Espressif Systems IOT Team, 2015, [Online]. Available at: <<http://img.filipeflop.com/files/download/Datasheet-ESP8266EX-v4.3.pdf>>. [Accessed 26 April 2018].
- [28] DHT11, DHT22 and AM2302 Sensors Adafruit Industries, 2018 [Online]. Available at: <<https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>>. [Accessed 26 April 2018].
- [29] PlatformIO IDE for Atom, 2014 [Online]. Available at: <<http://docs.platformio.org/en/latest/ide/atom.html>>. [Accessed 26 April 2018].
- [30] G2 Crowd. Best IoT Management Software, 2018. [Online]. Available at: <<https://www.g2crowd.com/categories/iot-management>>. [Accessed 26 April 2018].
- [31] ThingsBoard. What is ThingsBoard IoT Gateway? [Online]. Available at: <<https://thingsboard.io/docs/iot-gateway/what-is-iot-gateway/>>. [Accessed 27 May 2018].
- [32] Eclipse. Eclipse Mosquitto. [Online]. Available at: <<https://mosquitto.org/>>. [Accessed 27 May 2018].

10. Appendices

10.1. MQTT wemos code. The full PlatformIO project can be downloaded here <https://bit.ly/2si8V4c>

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>
#include "DHT.h"
#define WIFI_AP "AlykkaatPalvelut"
#define WIFI_PASSWORD "DigiHiki2017"
#define DHTPIN D4 // Digital PIN used to read DTH values
// Digital PINs used to receive controls from thingsboard (ex, Leds, Motors)
#define GPIO6_PIN 6
#define GPIO7_PIN 7
#define GPIO6 12
#define GPIO7 13
#define DHTTYPE DHT11 // Define the type of the DHT
DHT dht(DHTPIN, DHTTYPE); // Sets up the DHT ;
char deviceId[] = "f35bcef0-3ef5-11e8-8fd0-cb746776d0c2";
char thingsboardServer[] = "52.178.138.16"; // Define the thingsboard server
Address

int mqttPort = 1884; // The default is 1883 but in some cases it might be required
to change the port

int readingTries = 0; // Number of times that wemos will try to recollect the
data in case of a failure;

#define TOKEN "DHT11_DEMO_TOKEN" // Define the device Token which we will use
(copy the token from thingsboard device panel)

WiFiClient wifiClient;
PubSubClient client(wifiClient); // MQTT Client;
//Control variables

int status = WL_IDLE_STATUS;
unsigned long lastSend;

boolean gpioState[] = {false, false};

//This method will read the data from the sensor and will send it to thingsboard
server through MQTT

void getAndSendTemperatureAndHumidityData()
{
    //Serial.println("Collecting temperature data.");
```



```

// Reading temperature or humidity takes about 250 milliseconds!
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    readingTries += 1;
    return;
}

/* Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C ");    */
String temperature = String(t);
String humidity = String(h);
/* Just debug messages
Serial.print( "Sending temperature and humidity : [" );
Serial.print( temperature ); Serial.print( "," );
Serial.print( humidity );
Serial.print( "]"    -> " );*/
// Prepare a JSON payload string
String payload = "{";
payload += "\"temperature\":"; payload += temperature; payload += ",";
payload += "\"humidity\":"; payload += humidity;
payload += "}";

// Send payload via MQTT
char attributes[100];
payload.toCharArray( attributes, 100 );
client.publish( "v1/devices/me/telemetry", attributes );
}

//This method will read the status from the GPIO pins (HIGH -1 LOW - 0) and will
return a String with the values
String get_gpio_status() {
    // Prepare gpios JSON payload string
    StaticJsonBuffer<200> jsonBuffer;

```

```

JsonObject& data = jsonBuffer.createObject();
data[String(GPIO6_PIN)] = gpioState[0] ? true : false;
data[String(GPIO7_PIN)] = gpioState[1] ? true : false;
char payload[256];
data.printTo(payload, sizeof(payload));
String strPayload = String(payload);
Serial.print("Get gpio status: ");
Serial.println(strPayload);
return strPayload;
}

//This method will update the state of the GPIO pins
void set_gpio_status(int pin, boolean enabled) {
    if (pin == GPIO6_PIN) {
        // Output GPIOs state
        digitalWrite(GPIO6, enabled ? HIGH : LOW);
        // Update GPIOs state
        gpioState[0] = enabled;
    } else if (pin == GPIO7_PIN) {
        // Output GPIOs state
        digitalWrite(GPIO6, enabled ? HIGH : LOW);
        // Update GPIOs state
        gpioState[1] = enabled;
    }
}

// The callback for when a PUBLISH message is received from the server. (Method
used just in the GPIO control part)
void on_message(const char* topic, byte* payload, unsigned int length) {
    Serial.println("On message");
    char json[length + 1];
    strncpy(json, (char*)payload, length);
    json[length] = '\0';
    Serial.print("Topic: ");
    Serial.println(topic);
    Serial.print("Message: ");
    Serial.println(json);
    // Decode JSON request
    StaticJsonBuffer<200> jsonBuffer;
    JsonObject& data = jsonBuffer.parseObject((char*)json);
    if (!data.success())
    {

```

```

    Serial.println("parseObject() failed");
    return;
}

// Check request method
String methodName = String((const char*)data["method"]);
if (methodName.equals("getGpioStatus")) {
    // Reply with GPIO status
    String responseTopic = String(topic);
    responseTopic.replace("request", "response");
    client.publish(responseTopic.c_str(), get_gpio_status().c_str());
} else if (methodName.equals("setGpioStatus")) {
    // Update GPIO status and reply
    set_gpio_status(data["params"]["pin"], data["params"]["enabled"]);
    String responseTopic = String(topic);
    responseTopic.replace("request", "response");
    client.publish(responseTopic.c_str(), get_gpio_status().c_str());
    client.publish("v1/devices/me/attributes", get_gpio_status().c_str());
}
}

void InitWiFi()
{
    Serial.println("Connecting to AP ...");
    // attempt to connect to WiFi network

    WiFi.begin(WIFI_AP, WIFI_PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("Connected to AP");
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        status = WiFi.status();
        if (status != WL_CONNECTED) {
            WiFi.begin(WIFI_AP, WIFI_PASSWORD);
            while (WiFi.status() != WL_CONNECTED) {
                delay(500);
                Serial.print(".");
            }
        }
    }
}

```

```

    Serial.println("Connected to AP");
}
Serial.print("Connecting to ThingsBoard node ...");
// Attempt to connect (clientId, username, password)
if ( client.connect(deviceId, TOKEN, NULL) ) {
    Serial.println( "[DONE]" );

    client.subscribe("v1/devices/me/rpc/request/+");
    // Sending current GPIO status
    Serial.println("Sending current GPIO status ...");
    client.publish("v1/devices/me/attributes", get_gpio_status().c_str());
} else {
    Serial.print( "[FAILED] [ rc = " );
    Serial.print( client.state() );
    Serial.println( " : retrying in 5 seconds]" );
    // Wait 5 seconds before retrying
    delay( 5000 );
}
}
}

void setup() {
    Serial.begin(9600);

    Serial.println("DHTxx test!");
    pinMode(GPIO6, OUTPUT);
    pinMode(GPIO7, OUTPUT);
    dht.begin();
    delay(10);
    InitWiFi();
    client.setServer( thingsboardServer, mqttPort );
    client.setCallback(on_message);
    lastSend = 0;
}

void loop() {
    if ( !client.connected() ) {
        reconnect();
    }

    if(readingTries <= 15){
        if ( millis() - lastSend > 5000 ) { // Update and send only after 1 seconds

```

```

        getAndSendTemperatureAndHumidityData();
        lastSend = millis();
    }
}

client.loop();
}

```

10.2. HTTP wemos code. The full PlatformIO project can be downloaded here <https://bit.ly/2LC0MjJ>

```

#include <Arduino.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"

#define WIFI_AP "AlykkaatPalvelut"
#define WIFI_PASSWORD "DigiHiki2017"
#define DHTPIN D4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
HTTPClient http;

char                                thingsboardServer[] =
"http://192.168.1.102:8080/api/v1/DHT11_DEMO_TOKEN/telemetry";
#define TOKEN "DHT11_DEMO_TOKEN"
WiFiClient wifiClient;

int status = WL_IDLE_STATUS;
unsigned long lastSend;

void getAndSendTemperatureAndHumidityData()
{
    Serial.println("Collecting temperature data.");

    // Reading temperature or humidity takes about 250 milliseconds!
    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();

```

```

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
}

// Serial.print("Humidity: ");
// Serial.print(h);
// Serial.print(" %\t");
// Serial.print("Temperature: ");
// Serial.print(t);
// Serial.print(" *C ");

String temperature = String(t);
String humidity = String(h);

if(WiFi.status() == WL_CONNECTED) {
    // Just debug messages
    // Serial.print( "Sending temperature and humidity : [" );
    // Serial.print( temperature ); Serial.print( "," );
    // Serial.print( humidity );
    // Serial.print( "]" -> " );

    // Prepare a JSON payload string
    String payload = "{";
    payload += "\"temperature\":"; payload += temperature; payload += ",";
    payload += "\"humidity\":"; payload += humidity;
    payload += "}";

    // Send payload
    char attributes[100];
    payload.toCharArray( attributes, 100 );
    http.begin(thingsboardServer);
    http.addHeader("Content-Type", "application/json");
    int httpCode = http.POST(attributes);
    Serial.println(httpCode);    //Print HTTP return code
    Serial.println(payload);    //Print request response payload

    http.end();    //Close connection
}

// Serial.println( attributes );

```

```

}

void InitWiFi()
{
  Serial.println("Connecting to AP ...");
  // attempt to connect to WiFi network

  WiFi.begin(WIFI_AP, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}

void setup() {
  Serial.begin(9600);

  Serial.println("DHTxx test!");
  dht.begin();
  delay(10);
  InitWiFi();

  lastSend = 0;
}

void loop() {
  if ( millis() - lastSend > 1000 ) { // Update and send only after 1 seconds
    getAndSendTemperatureAndHumidityData();
    lastSend = millis();
  }
}

```