University of Messina
Engineering Department

Master's Degree Course in
Engineering and Computer Science

Industrial IoT Project

IoT-based Smart Intrusion Detection and Notification System for Industry 4.0

Professor:
Giovanni Merlino

Realized by:
Davide Giuseppe Allegra
Cateno Gabriele D'Alì

Academic Year 2022/2023

Introduction

The following project, as the title suggests, aims to develop an intrusion detection system for the Smart Industry, utilizing IoT technologies, in order to ensure enhanced security within industrial environments. The system will continuously monitor industrial sensors, detecting intrusions and promptly sending notifications to operators through various IoT communication systems. Additionally, to provide an extra layer of prevention and security against Man-In-The-Middle (MITM) attacks, version 5 of the MQTT protocol has been employed, with authentication through certificates.

The project features are outlined below:

- *Intrusion Detection Sensors*: A motion sensor (HC-SR501 PIR motion sensor) has been installed on the Arduino Uno R3 board to detect intrusions within the company premises.
- *Alarms and Notifications*: Upon detecting a potential intrusion, the system will automatically trigger an alarm, also installed on the Arduino Uno R3 board, and then immediately send notifications to industry members.
- *MQTT Protocol (Message Queuing Telemetry Transport)*: The AWS IoT Core service from Amazon Web Services (AWS) has been configured to enable IoT communication among various devices.
- *Remote Control*: To allow remote control, an app named "TLS IoT Tool" has been utilized, leveraging certificate-based connection through the SSL/TLS protocol provided by Amazon's CA. Through this app, commands can be sent to the Arduino board via the MQTT protocol, such as enabling/disabling alarms, and more.
- *Data Storage*: For storing intrusion-related data, the InfluxDB time series database has been employed. This database is designed for efficient storage and optimized retrieval of time-varying data, including sensor metrics, server logs, and monitoring data.
- *System Activity Monitoring*: Subsequently, a dashboard (via a web page) has been implemented to visualize all system activities related to intrusions.

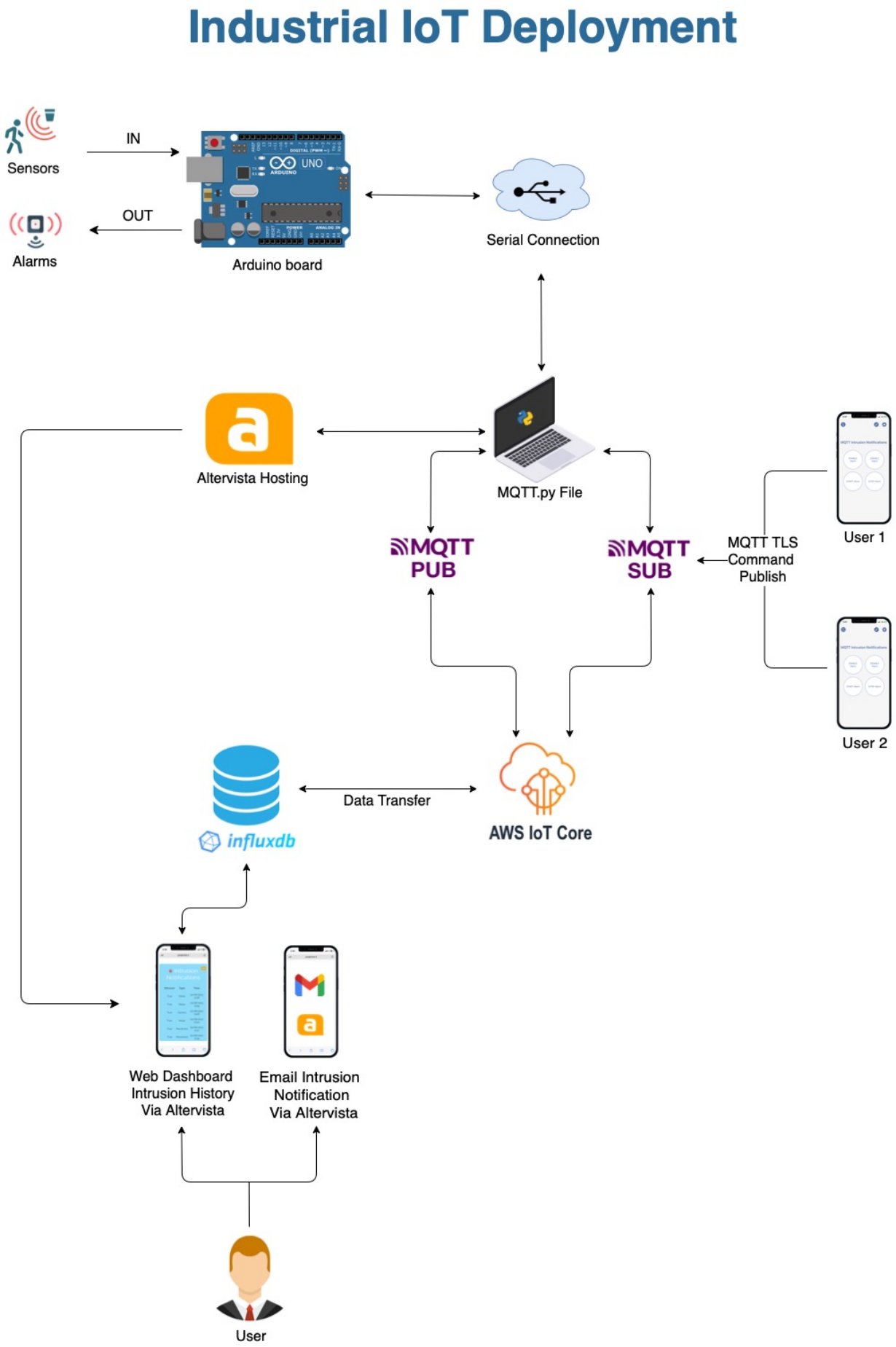
Technologies Used:

- Arduino IDE
- PyCharm
- Hosting Altervista
- Amazon Web Service (AWS)
- MQTT
- TLS IoT Tool
- InfluxDB Cloud
- Draw.io

Programming Languages Used:

- C
- Python
- Java Script
- InfluxQL/SQL

Below, a diagram summarizing the dynamics of the designed project is provided by means of the *Draw.io* platform.



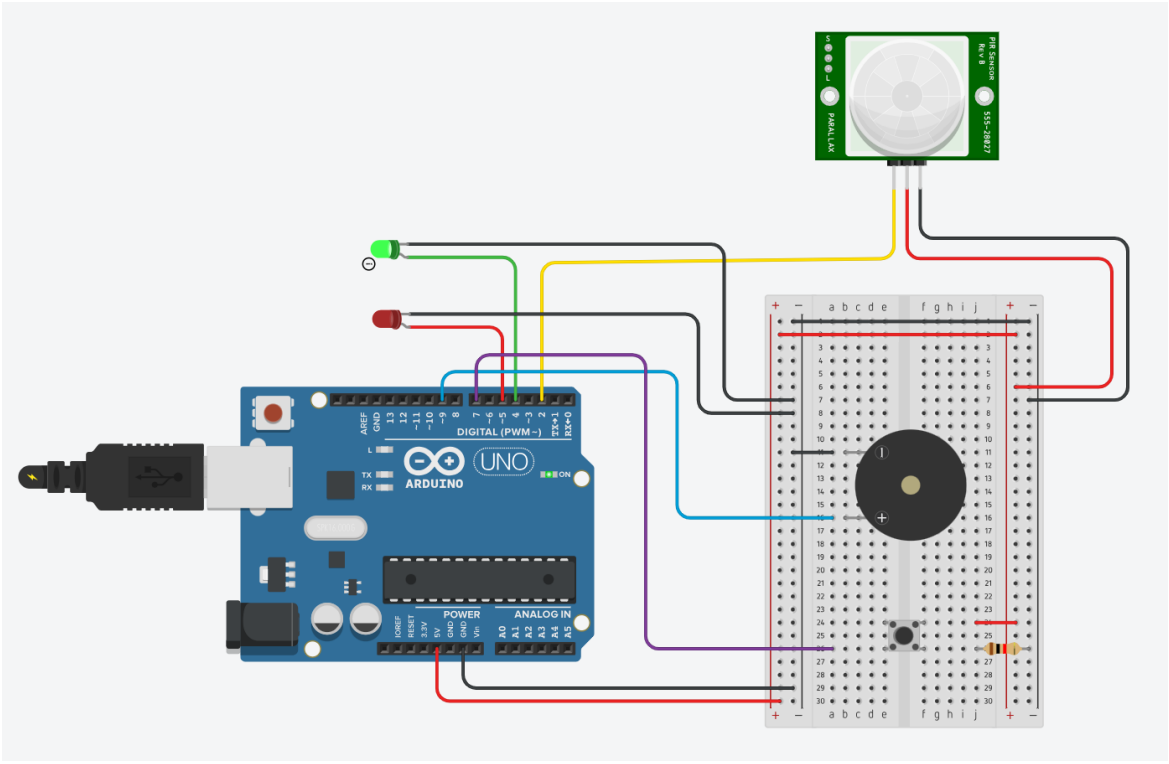
Arduino - What is?

Arduino is an open-source platform for hardware and software prototyping and development. It empowers users to create interactive projects, customized electronic devices and automated control systems. Arduino offers an integrated development environment (IDE) that makes easier the process of writing and uploading code onto these boards. The significance of Arduino in the realm of the Internet of Things (IoT) infers from its versatility and user-friendliness. Through the combination of accessible hardware and intuitive software, developers can create network-connected devices in a straightforward and dynamic manner. Arduino boards can be integrated with sensors, actuators, and other components to gather data from the surrounding environment and interact with it. This data can then be transmitted across the network for analysis, monitoring, or control by remote devices.

Arduino is composed of sensors, which are devices designed to detect various physical or environmental quantities, such as temperature, light, pressure, motion, humidity and so forth. They convert this information into digital or analog signals (simulated through PWM) that can be interpreted and utilized by electronic devices. These data can then be processed and employed for decision-making, automating actions, monitoring environmental conditions, or creating interactions with users. Without the use of sensors, the Arduino experience would certainly be incomplete and less flexible.

Arduino Project Idea

For the following project a circuit was developed on the Arduino UNO R3 platform capable of detecting movements in the surrounding environment. The underlying concept is to design an IoT system capable of detecting intrusions in the domain of Industry 4.0, but which can also be effectively extended to domestic and home automation scenarios.
The system operates through the utilization of a PIR sensor for motion detection. The digital output of the sensor is a simple binary signal [0: no detection, 1: motion detected]. Subsequently, within the Arduino IDE, the system's digital value is controlled through logic port 2 to verify the system status. If the value equals "1," it signifies that motion has been detected, triggering an alarm through a buzzer (piezo) installed on the same Arduino board.



Arduino Sketch

Within the sketch loaded onto the Arduino board, there's a reference to the "*SoftwareSerial.h*" library, which facilitates software-based serial communications on digital pins other than the predefined ones for Arduino's hardware serial communication.

The line *SoftwareSerial mySerial(10, 11)* instantiates an object named mySerial using the SoftwareSerial class. This object will be responsible for managing software serial communication using Arduino's digital pins 10 and 11.

These two pins are utilized respectively for receiving (RX) and transmitting (TX) serial data. This configuration establishes software serial communication on specific digital pins.

As can be noticed from the provided code, three Boolean variables have been defined:

- *deviceIsReady*: it has been set to false because the status of system remains inactive untile the motion sensor loading il competed;
- *alarmEnabled*: it has been set to true because the system is active by default;
- *alarmTriggered*: it has been set to false because initially the alarm is in the idle state.

In this particular scenario, it was particularly helpful for lack of an "*ESP32*" internet module to communicate with the "*COM10*" port of the operating system, enabling serial communication through the Arduino board connected to the USB port.

If the system is enabled, every second the reception of messages from MQTT subscription system is checked.

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11); // RX, TX

bool deviceIsReady = false;
bool alarmEnabled = true;
bool alarmTriggered = false;

void loop()
{
    if(deviceIsReady)
    {
        if(alarmEnabled)
        {
            int motionValue = digitalRead(PIR);

            if(motionValue == HIGH)
            {
                if(alarmTriggered == false)
                {
                    mySerial.println(1);
                    Serial.println(1);
                    startAlarm();
                }
            }
            else
            {
                mySerial.println(0);
                Serial.println(0);
            }
        }

        MQTT_checkMessages();
        checkButtonStatus();
        delay(1000);
    }
    else
    {
        LedBlinking();
    }
}
```

MQTT

As already mentioned, an Arduino Uno along with a motion sensor to detect potential intrusions has been used. However, since Arduino lacks direct Internet connectivity, it requires the assistance of a computer to which it is connected through a USB port. Through this serial connection (*ser*), Arduino can communicate with the Python console on the computer by means of a bidirectional data exchange.

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Serial port config
serial_port = 'COM10'
baud_rate = 9600
ser = serial.Serial(serial_port, baud_rate)
```

What MQTT is and Its Utility in IoT

MQTT (Message Queuing Telemetry Transport) is a lightweight and efficient messaging protocol designed for transmitting data between devices with bandwidth and power constraints, common in IoT networks. It's based on a publish-subscribe model, where devices publish messages on specific topics, and other devices subscribe to receive those messages. This model makes MQTT ideal for sending data from distributed devices to a wide range of destinations, including cloud services like AWS which we will discuss later. Its strengths include:

- Efficiency: MQTT is designed to operate in networks with limited bandwidth and power resources, making it suitable for resource-constrained IoT devices.
- Scalability: MQTT's publish-subscribe model enables scalable communication among numerous devices, providing a solid foundation for complex IoT systems.
- Reliability: MQTT ensures reliable message delivery through its receipt confirmation model.

In the context of the Internet of Things (IoT), reliable and scalable communication between devices is crucial to ensure the flow of data and information among interconnected components. MQTT stands out as one of the most relevant and widely used communication protocols in the IoT ecosystem.

In order to enable Arduino to send data to a cloud platform like AWS (Amazon Web Service), it is necessary to leverage a cloud service such as AWS IoT, which is a cloud service offered by AWS that enables easy connectivity of IoT devices to the cloud and interaction with them.

This service offers the MQTT communication protocol to act as a bridge between Arduino and AWS. As a result, the data collected by Arduino can be transmitted to the AWS cloud service through the serial connection, and subsequently, via the MQTT protocol. This enables the processing, analysis, and visualization of the data collected by Arduino using the cloud computing services provided by AWS.

MQTT Configuration

Configuring MQTT involves a series of well-defined steps to ensure that devices can communicate effectively and securely. As mentioned, in the following project, an IoT system has been created that detects intrusions using a motion sensor and sends data to an AWS cloud service. MQTT requires the implementation of a client-server model, where devices are represented as clients and the MQTT broker acts as an intermediary to facilitate communication. The key configuration steps are outlined below:

- *Broker Configuration*: It is necessary to select and set up an MQTT broker. In the project, AWS IoT Core was used as the chosen broker, which acts as an intermediary for communication between devices. The configuration includes the broker's endpoint, credentials, and certificates required to establish a secure connection.

- *Client Configuration*: Each device participating in MQTT communication must be configured as an MQTT client. In the project, the Arduino device is configured as an MQTT client. A unique identifier (client ID) must be provided, and credentials and certificates necessary for the connection must be configured. The MQTT client connects to the broker using the configured endpoint, credentials, and certificates. The connection occurs through a secure protocol.

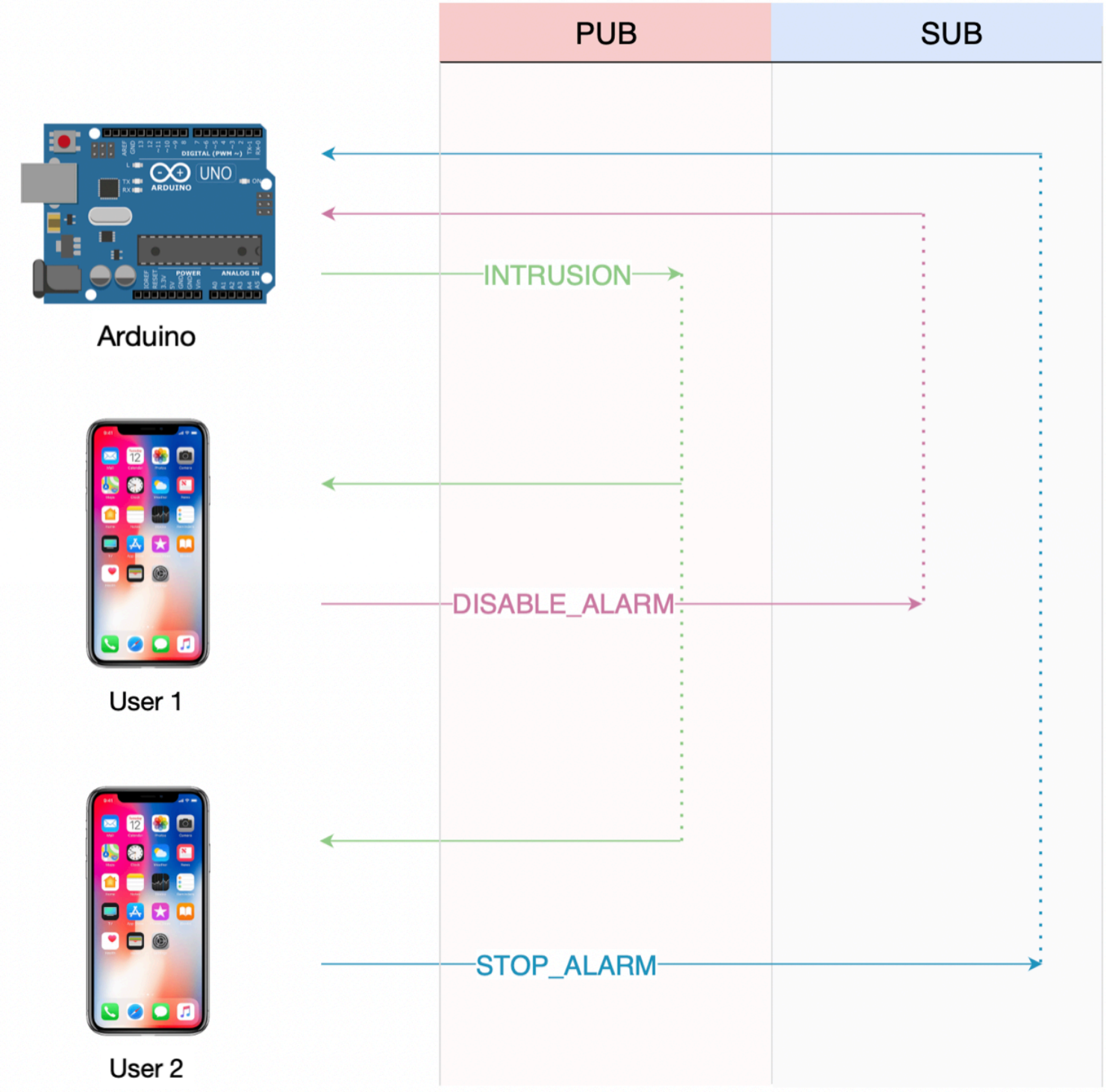
```
# Create a MQTT client
my_mqtt_client = AWSIoTMQTTClient(Config.client_id)
my_mqtt_client.configureEndpoint(Config.endpoint, 8883)
my_mqtt_client.configureCredentials(Config.root_ca, Config.private_key, Config.certificate)

# Broker AWS IoT connection
my_mqtt_client.connect()
print(">> MQTT broker connected")

# Topic subscription
my_mqtt_client.subscribe(sub_topic, 1, readMessages)
print(">> Client subscribed to ArduinoSensors topic")
```

MQTT topics are communication channels that allow clients to send and receive messages. In the project, two topics are defined: sub_topic and pub_topic:

- *Subscription Topic (SUB*): Arduino reads, while other devices publish.
- *Publishing Topic (PUB)*: Arduino publishes, while other devices read.

	PUB	SUB
Arduino	Write	Read
User 1	Read	Write
User 2	Read	Write

Presented below is a concise diagram depicting the communication among devices through MQTT's Publish-Subscribe (PUB-SUB) channels.



MQTT operates on a client-broker model, wherein the broker serves as an intermediary between clients. Clients publish messages on specific topics to which other clients can subscribe. The broker routes messages to the appropriate clients based on their subscribed topics. This enables effective and flexible communication between devices, even within resource-constrained networks.

Usage of Topics
In the MQTT.py file, the Arduino device publishes data to the pub_topic while simultaneously subscribing to the sub_topic to receive messages from other devices. When the Arduino device detects an alarm, it publishes an MQTT message containing intrusion-related data. This data is then sent to the AWS IoT Core broker, and devices subscribed to the pub_topic channel access the content.

```
# MQTT topic
sub_topic = "ArduinoSensors/sub"
pub_topic = "ArduinoSensors/pub"

# Topic Publication
my_mqtt_client.publish(pub_topic, json.dumps(message), 1)

# Topic subscription
my_mqtt_client.subscribe(sub_topic, 1, readMessages)
```

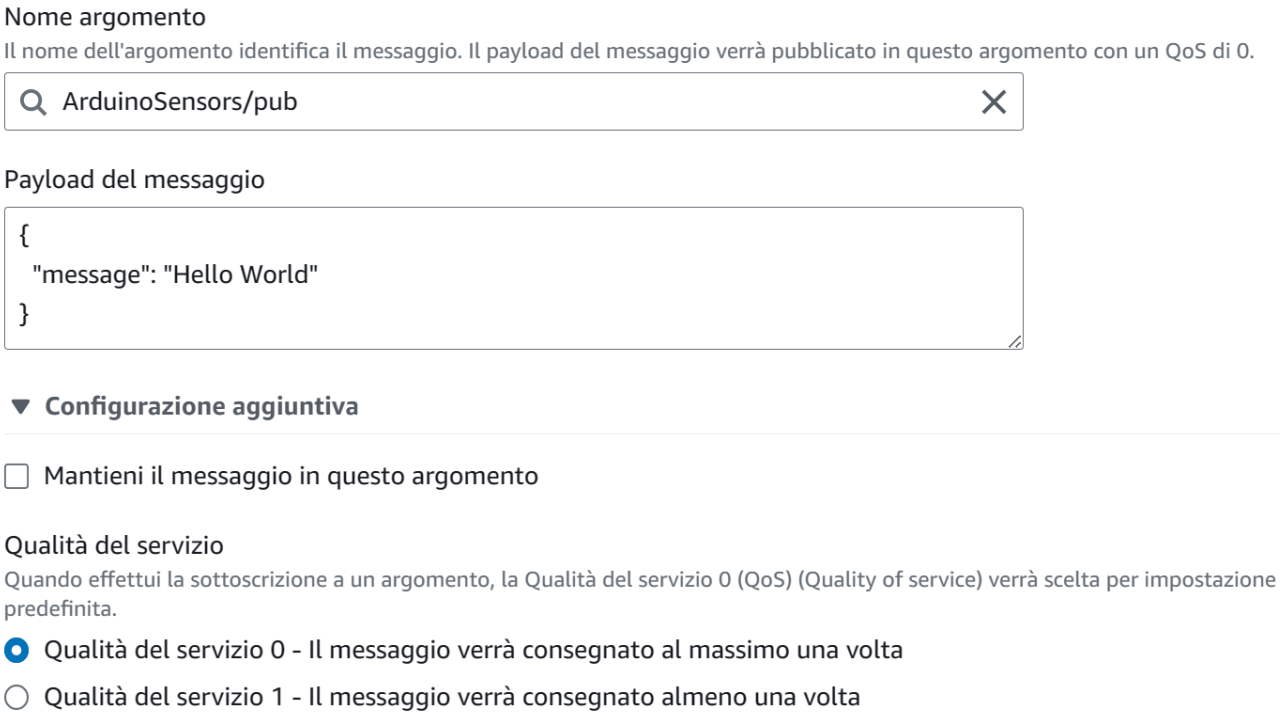
Quality of Service
Quality of Service (QoS) in MQTT refers to the level of message delivery assurance between the sender and the receiver. MQTT offers three levels of QoS:

- *QoS 0 (At most once)*: In this level, the message is sent from the sender to the MQTT broker only once, without any delivery confirmation. The broker does not keep track of message deliveries or receipts. This level offers the least delivery guarantee but is the most efficient.

- *QoS 1 (At least once)*: In this level, the message is sent from the sender to the MQTT broker, and the broker acknowledges the receipt. If the broker doesn't receive an acknowledgment from the receiver, it will resend the message. This level provides a higher delivery guarantee but might result in message duplications.

The "Retain" flag in MQTT is used to retain the last published message on a specific topic. When a message with the Retain flag is published on a topic, the broker stores it and automatically sends it to new clients subscribing to that topic. This is useful, for example, to provide new clients with the latest sensor reading when they connect.

Below is an example of a message being published on the PUB topic using QoS 0 and with retain disabled in the AWS MQTT test console:



Features of MQTT 5

MQTT version 5 is an evolution of MQTT version 3, introduced to bring new features and enhancements. Some of the main differences include:

- *Message Properties*: MQTT 5 introduces structured message properties, allowing inclusion of additional metadata such as timestamps, content types, session identifiers, and more, directly within messages.
- *Improved QoS 2*: MQTT 5 defines an improvement to QoS 2, making it more efficient compared to version 3.
- *Session Handling*: MQTT 5 provides more detailed control over session management, allowing clients to specify the level of support required from the broker.
- *Client Acknowledgment*: MQTT 5 introduces a client acknowledgment mechanism, enabling confirmation of message reception regardless of the QoS level.
- *Topic Handling Support*: MQTT 5 offers more flexibility in topic handling, including the ability to share topics between clients and brokers.
- *Broker Login with TLS X.509 Certificates*: Between MQTT version 3 and version 5, changes related to TLS X.509 certificates primarily involve the addition of new features and properties in the MQTT version 5 protocol, enabling more advanced certificate security management. The addition of these new properties and features in MQTT version 5 allows for more flexible and detailed management of X.509 certificates and security in MQTT messages overall.

Amazon Web Service - AWS

Amazon Web Services (AWS) is a cloud computing platform provided by Amazon. It offers a wide range of services and computing resources over the Internet; each resource on AWS is identified by an Amazon Resource Name (ARN), which uniquely identifies the object. AWS enables businesses, developers, and individuals to host applications, store data, and utilize scalable computing resources without investing in physical infrastructure.

AWS has become one of the leading global cloud service providers, offering a wide range of services including computing, storage, databases, analytics, machine learning, artificial intelligence, security, and more.

Advantages

- *Scalability*: AWS provides scalable resources, allowing you to increase or decrease computing resources based on real-time needs.
- *Flexibility*: AWS services are highly customizable, allowing you to choose only what you need and pay only for the resources you actually use.
- *Reliability*: AWS has a global presence with data centers in various regions, ensuring a high level of availability and redundancy.
- *Wide Range of Services*: AWS covers almost every aspect of cloud computing, allowing users to build and manage complex applications.
- *Security*: AWS offers advanced security measures to protect user data and applications, including access control, log monitoring, and more.

Disadvantages

- *Costs*: While AWS offers a free tier and usage-based payment flexibility, costs can escalate if resource usage isn't closely monitored.
- *Initial Complexity*: For those new to the cloud environment, learning AWS concepts and functionalities might take time and effort.
- *Internet Dependency*: Accessing AWS services requires a reliable internet connection. Lack of connectivity could hinder resource access.

Using AWS for IoT

AWS is particularly beneficial for the Internet of Things (IoT) as it provides a range of services that allow developers to effectively create, manage, and scale IoT applications. Some specific advantages include:

- *Elasticity*: AWS allows you to manage a large number of IoT devices, automatically scaling resources to handle load spikes.
- *Data Storage and Analytics:* AWS offers advanced storage and analytics services to manage and derive value from data generated by IoT devices.
- *Machine Learning and Predictive Analytics*: AWS provides advanced machine learning and analytics services for extracting insights from IoT data and making predictive decisions.
- *Security*: AWS offers advanced security tools to protect IoT data and devices from threats and unauthorized access.
- Global Connectivity: With its global data center presence, AWS enables reliable worldwide connectivity for IoT devices.

- *Locations*: The decision to opt for localization on AWS Frankfurt instead of near Italy was driven by the fact that suitable locations were not available in Italy during the design phase. Despite the geographical distance, Frankfurt provides reasonable communication latencies, whereas more distant locations could have had a greater impact on latency times. Furthermore, the choice to select different locations on AWS allows for greater flexibility and security for the installed applications.

Amazon AWS VS Microsoft Azure

In addition to Amazon AWS, there's another cloud service provided by Microsoft called Microsoft Azure, which is named Microsoft Azure. It offers IoT services that allow developers to create, manage, and scale IoT applications and devices. However, there are differences between the IoT services offered by the two platforms in terms of functionality, integration, ecosystem, pricing, and more.

Azure IoT is integrated with the Azure service ecosystem, including machine learning (Azure Machine Learning), data analytics (Azure Stream Analytics), resource management (Azure Resource Manager), and more.

Microsoft Azure IoT provides three services:

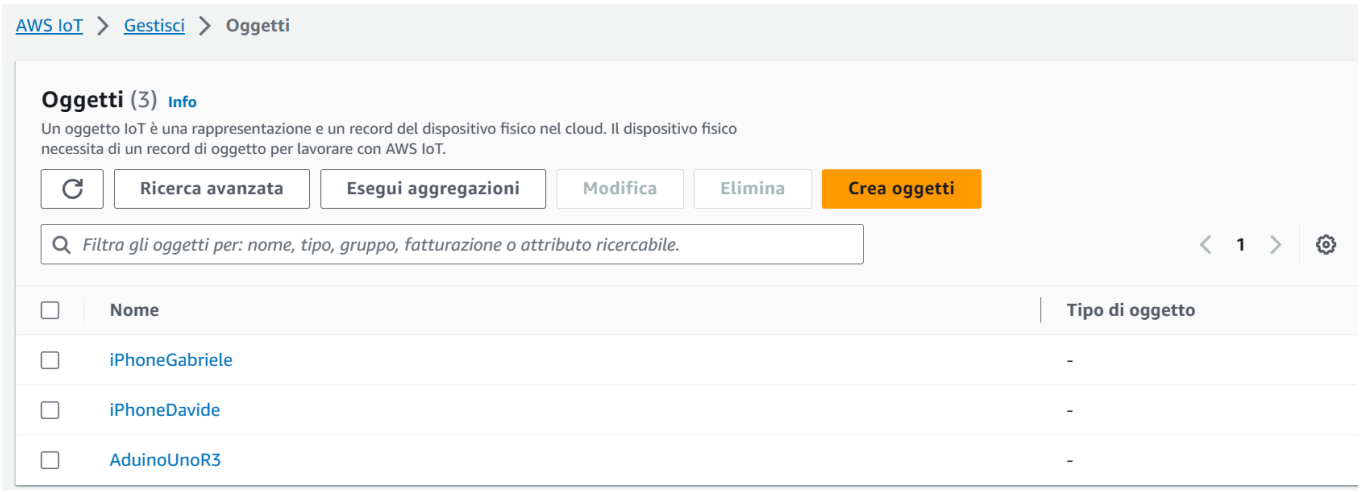
- *Azure IoT Hub*: This service offers connectivity and management features for IoT devices, supporting various communication protocols and enabling easy scaling to millions of devices.
- *Azure IoT Edge*: Similar to AWS IoT Greengrass, Azure IoT Edge allows for computation and analysis at the edge devices, improving latency and reducing dependence on cloud connection.
- *Azure IoT Central*: This service simplifies the creation and management of complex IoT solutions, providing an intuitive user interface for device and data configuration and monitoring.

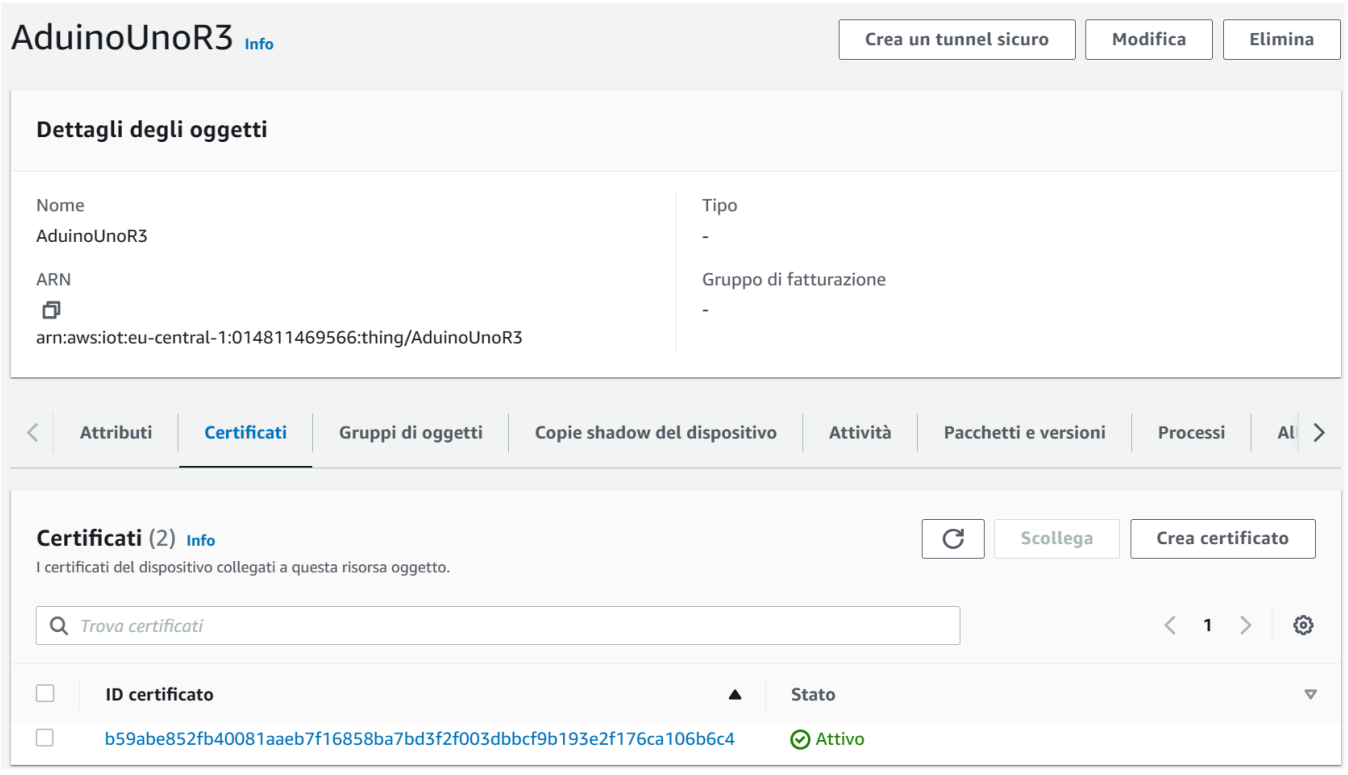
Both AWS IoT and Azure IoT offer a wide range of services to meet IoT application needs. The choice between the two will depend on user preferences, existing skills, specific project requirements, and other factors such as pricing and support. Both platforms are industry leaders in cloud computing and IoT, and the choice between them can depend on individual needs and priorities.

AWS Configuration

The configuration of AWS consists of three fundamental steps:

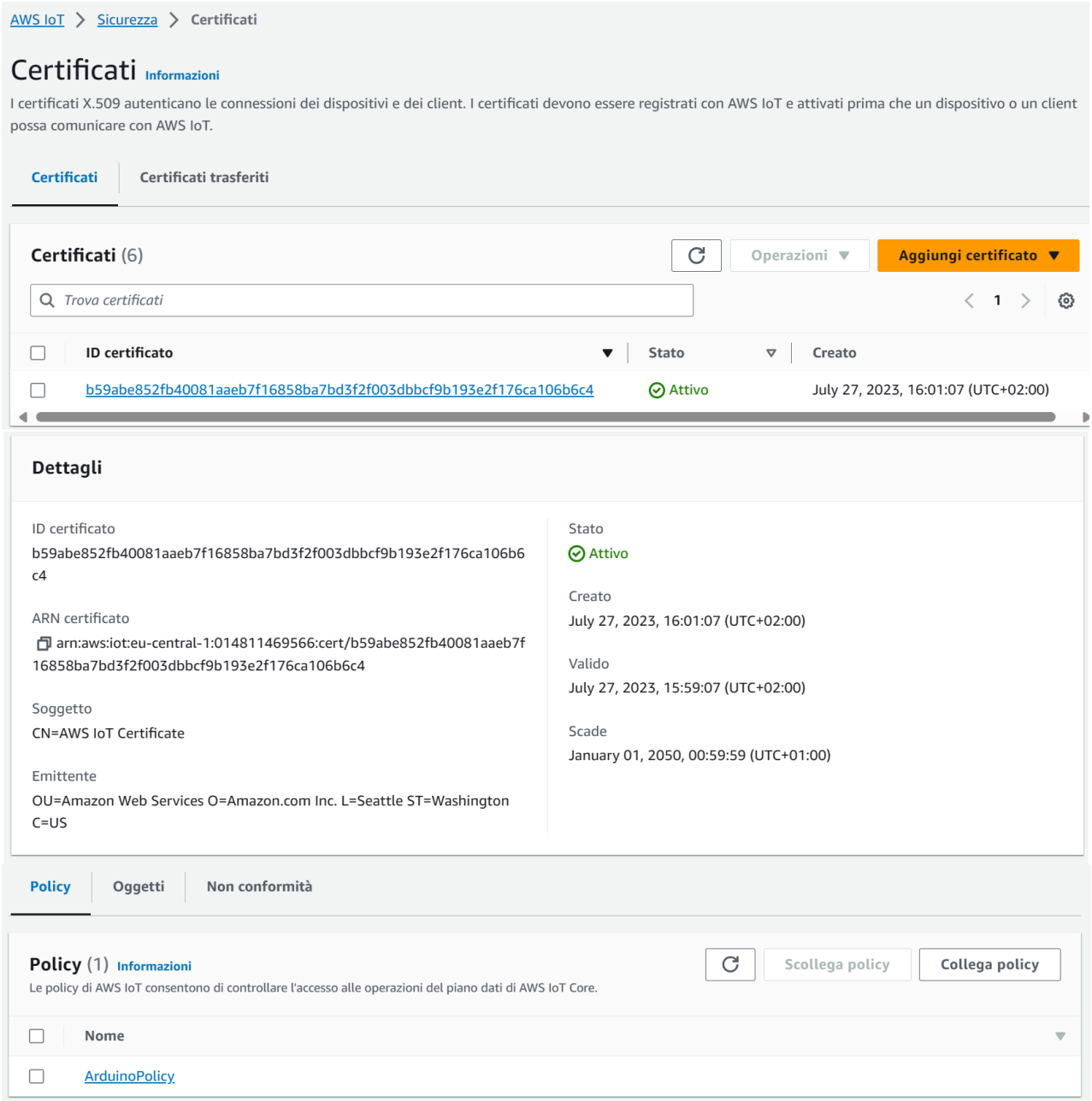
1. **Things**: In AWS, a "Thing" is a fundamental concept in the AWS IoT service. It represents a single physical or virtual entity that can be monitored and controlled through the cloud. Things can be physical devices, sensors, actuators, vehicles, industrial equipment, and more. Each Thing is associated with a unique identifier called the "Thing Name," to which a unique ARN code is linked.





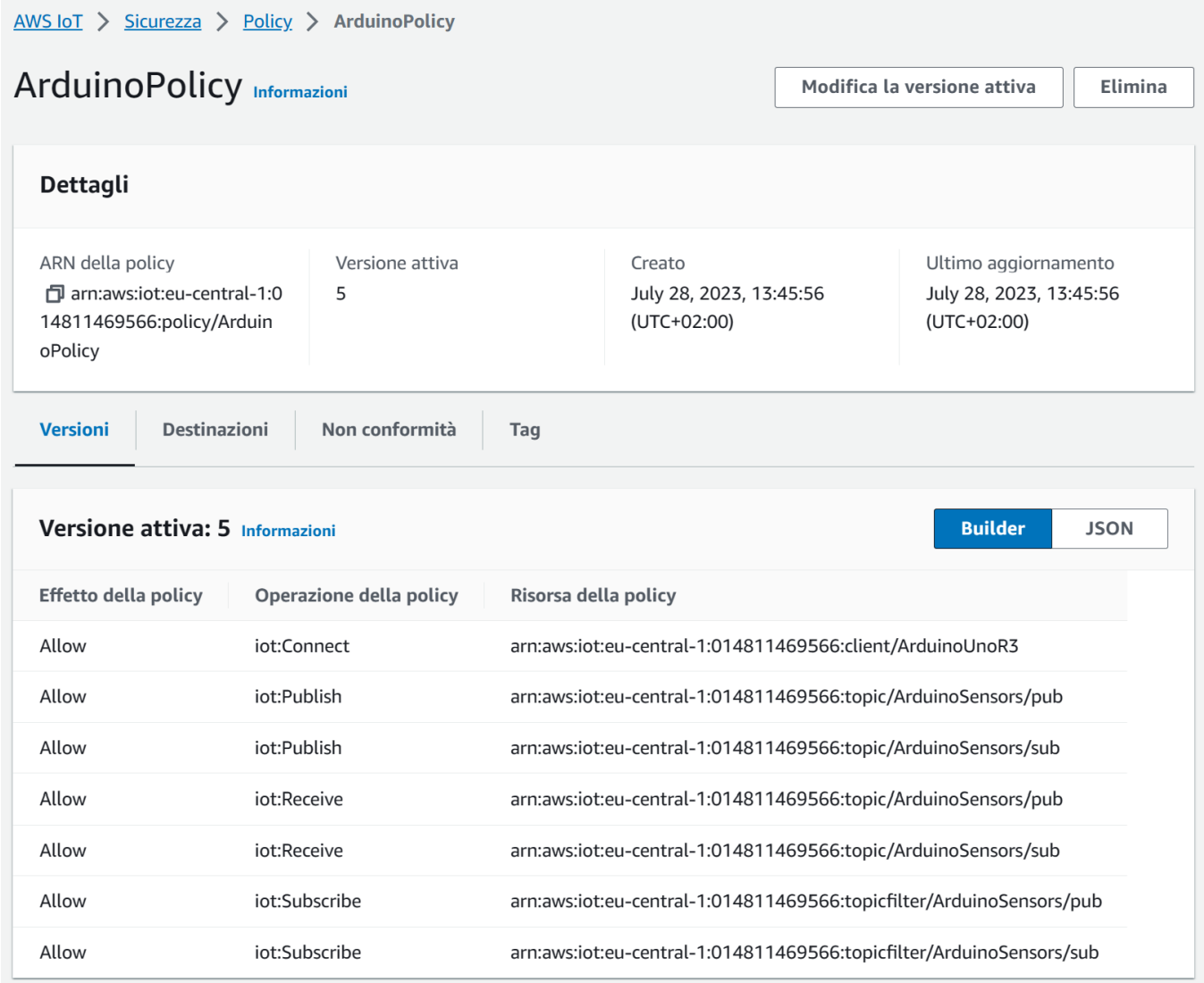
2. **Certificates**: Certificates in AWS IoT are used to establish the identity and security of Things and applications connecting to AWS IoT. They are used to authenticate and encrypt communications between the IoT device and the AWS IoT service.

    - *Device Certificate (X.509)*: Each Thing has its unique certificate, known as the device certificate. This certificate is installed on the IoT device to authenticate itself when connecting to AWS IoT.
    - *Private Key*: Each certificate is associated with a private key, used for data signing and decryption. The private key is securely stored on the device.
    - *CA Root Certificate*: AWS IoT uses a hierarchy of root certificates from Certificate Authorities (CAs) to establish trust in device certificates. The AWS IoT CA root certificate needs to be integrated into devices or client applications to allow them to communicate with the service.



3. **Policies**: Policies in AWS IoT define what actions a Thing can perform on which resources within the service. Policies enable control over access and permissions for Things and users.

    - *Policy Definition*: A policy defines the allowed or denied actions (such as publishing to an MQTT topic or subscribing to a topic) and the criteria associated with specific resources (e.g., MQTT topics) within AWS IoT.
    - *Linking to Things*: Policies are linked to objects (Things) through certificates. A device certificate must be associated with one or more policies to establish which actions the Thing can execute.
    - *Access Control:* Policies allow for fine-grained control over Things' access to resources. For instance, a policy can permit a device to publish data only to a specific topic and deny access to other topics.



TLS IoT Tool Application

In the context of the Internet of Things, communication security is of paramount importance to ensure the protection of data transmitted between connected devices. In the project, the "TLS IoT Tool" application was used, which enables remote control of the Arduino board. The primary objective revolves around establishing a fully safeguarded MQTTs communication framework to exercise authoritative control over an IoT endpoint. This endeavor encapsulates a concerted effort to create an intricate ecosystem that ensures the utmost security and seamless control over IoT devices, offering advanced authentication options while mitigating potential vulnerabilities. The central aim is to forge a comprehensive toolkit that encompasses the ensuing capabilities:

Open TLS IoT Tool

- Exclusive reliance on X.509 for both authentication and fortification of communications.
- Comprehensive integration of CA Root authentication mechanisms.
- Seamless integration of Face ID or Touch ID functionalities to enhance user authentication.
- Rigorous countermeasures implemented to thwart potential middle-man attacks.
- Elimination of the necessity to configure NAT port forwarding for the ultimate IoT device.

Below, you will find documentation on how to download, use, and configure the TLS IoT Tool application.

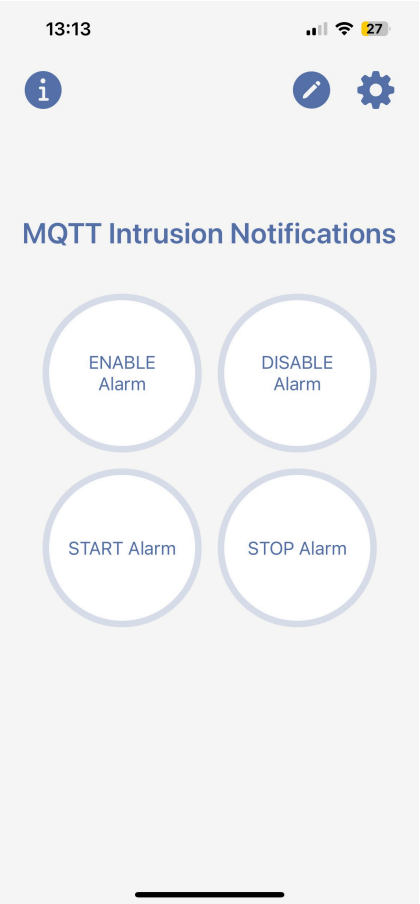
How to Download the Application

The "TLS IoT Tool" application is available for download on the app store of major mobile platforms, such as iOS and Android. Users can search for the app using the exact name or related keywords such as "IoT security tool" or "TLS communication." Once the application is found, users can download and install it on their device by following the standard instructions for downloading apps from the app store.

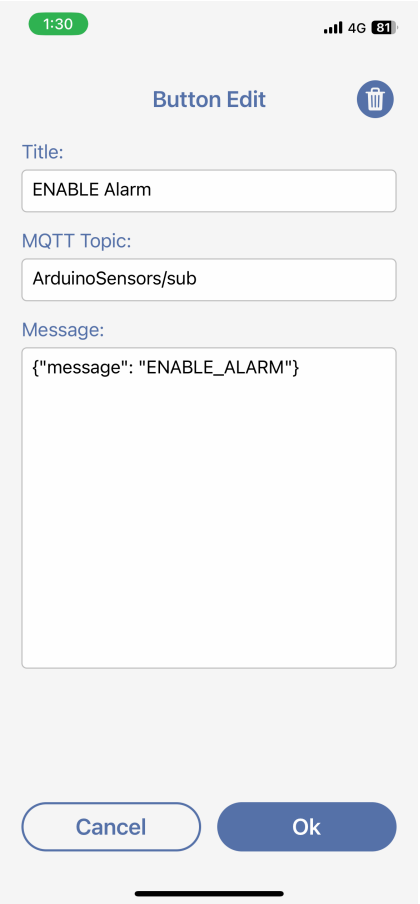
How to Configure the Application

The "TLS IoT Tool" application is designed to simplify the configuration and management of communication security protocols within IoT networks. It offers several key features:

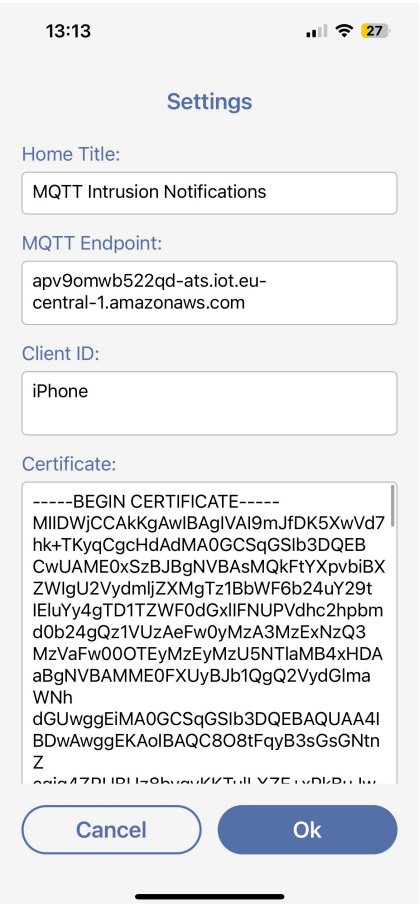
- *Key Management*: The app allows users to manage cryptographic key pairs (public and private) previously created by AWS, which are necessary for establishing secure connections.
- *Certificate Management:* The app supports the management of digital certificates previously created by AWS, which can be used to authenticate devices within the IoT network.
- *Configuration of TLS Parameters*: Users can specify various parameters related to the TLS protocol, such as protocol version, encryption algorithms, and key exchange algorithms. This enables them to tailor communication security to the specific needs of their IoT network.



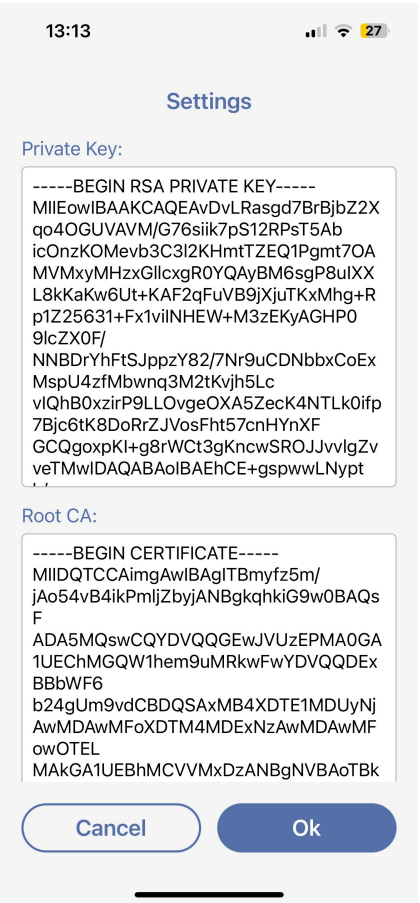
Homepage with the four MQTT commands



Example of JSON payload



AWS Parameters Configuration



Setting Certificate, Private Key and Root CA

Influx DB - What is?

InfluxDB Cloud functions as a rapid, adaptable, serverless real-time monitoring solution for gathering, retaining, handling, and presenting time series data. Time series data represents a sequence of data points systematically organized by time. Typically, these data points involve successive measurements derived from a common source, utilized for monitoring temporal changes. Tailored towards developers, InfluxDB Cloud operates as a cloud-native application hosted on AWS, capable of dynamic scalability in alignment with your workload. The data points commonly encompass consecutive measurements originating from the identical source and serve the purpose of tracking alterations across time. InfluxDB is crucial in the realm of IoT as it offers a dedicated solution for managing time series data. With its storage, analysis, and real-time monitoring capabilities, InfluxDB plays a role in converting raw data from IoT into meaningful and actionable insights, enabling more informed decision-making and enhancing operational efficiency.

Data Arrangement:

The structural framework of InfluxDB categorizes time series data into three essential components: buckets and measurements.

1. **Bucket**: A designated repository wherein time series data is stored, capable of housing multiple measurements
2. **Measurement**: A cohesive classification for time series data. All data points within a specific measurement share identical tags. A measurement comprises multiple tags and fields.
    - *Tags*: Pairs of key-value data with values that vary and undergo infrequent alteration. These tags serve the purpose of harboring metadata pertinent to each data point; examples include identifiers like host, location, and station.
    - *Fields*: Key-value pairs embodying data with fluctuating values over time. Illustrative instances encompass temperature, pressure, and stock price.
3. **Timestamp**: A chronological indicator linked to the data. When deposited and queried, all data aligns chronologically.

Documentation - Dependencies

In this section, documentation will be provided to analyze and explain each individual step required for the proper installation and configuration of InfluxDB. Following that, a few examples will be provided to make its usage more straightforward.
The chosen development environment is Python, therefore the initial mandatory step involves verifying the presence of Python3 on your system; the aforementioned framework will not be compatible with earlier Python versions.

Proceed to install the *influxdb3-python* module, which will serve as your tool for interacting with InfluxDB, enabling both data writing and querying. Execute the provided command within your terminal to carry out this installation.

```
pip install influxdb3-python
```

Eventually, the *pandas* library need to be installed, which will prove valuable in structuring our data output as we perform queries.

```
pip install pandas
```

Configuration

In this context, in a separate configuration file called Config.py file, we need to initiate some fundamental parameters which are essential for establishing the initial connection to InfluxDB, such as:

- **Token**: represents an authentication token used to access the InfluxDB Cloud APIs. This token identifies the authorized user or application to interact with the database. It's a secure form of access without needing to share sensitive credentials.
- **Org**: stands for the organization associated with the user or application. In InfluxDB databases, data is organized within different organizations, allowing better management and separation of data among different users or teams.
- **Host**: and server URL host ,signifies the URL of the InfluxDB Cloud server the client needs to establish a connection with. In this case, the URL is the one of the InfluxDB Cloud instance hosted in a specific geographical region (eu-central-1) within Amazon Web Services (AWS) cloud computing service.

```
# Config InfluxDB
token_influx = "H4cem9xbCj-7rT165wBxmHEDNebCg2Fq1bZ0LGY9cDlGGSP7zwKsTsEeb8Em7eNcPX6RD6VdgqXU9dwUCnx3-Q=="
org_influx = "University of Messina"
host_influx = "https://eu-central-1-1.aws.cloud2.influxdata.com"
```

Following this, in order to establish the client connection, an instance of *InfluxDBClient* is initialized by using the values from *Config.py* file for the InfluxDB host URL and authentication token. Also a *write_api* is instantiated using the *SYNCHRONOUS* write option. This indicates that write operations will be synchronous, meaning they will wait for a response before continuing.

```
client = InfluxDBClient(url=Config.host_influx, token=Config.token_influx)
write_api = client.write_api(write_options=SYNCHRONOUS)
```

Insert Data

In the MQTT.py file, within the *readArduinoValues()* function, the following code block saves the intrusion-related data detected by the Arduino sensor inside the *data_influx* dictionary, which is meant to store data for InfluxDB. Specifically it contains information about the intrusion ("Intrusion": "True") and the type of sensor used ("SensorType": “sensortype"). For academic purposes, the *random.choice()* function has been employed to randomly select an item from the *types* list. This approach assigns a distinct sensor type to each intrusion detected by the Arduino-implemented sensor, thereby having different values within the *SensorType* column.

InfluxDB is characterized by the *Point* objects which represents an individual data point within a time series. A time series is a sequence of data points indexed based on time, as a matter of fact the *Point* object is fundamental for storing data within InfluxDB, as it holds essential information about the data you want to store and analyze. Therefore within the for loop, which it has been created in order to iterate through the keys in the *data_influx* dictionary, a new InfluxDB *Point* object is created with the measurement name *ArduinoData*, which represents the logical grouping of similar data.

As previously mentioned a measurement contains multiple tags and fields, indeed:

- *.tag()* method adds a tag to the *Point* object. Tags are metadata that provide context to data points. In this case, the *Intrusion* tag is added with the value from the *data_influx* dictionary.
- *.field()* method adds a field to the *Point* object. Fields contain the actual data values. Here, the *SensorType* field is added with the value from the *data_influx* dictionary.

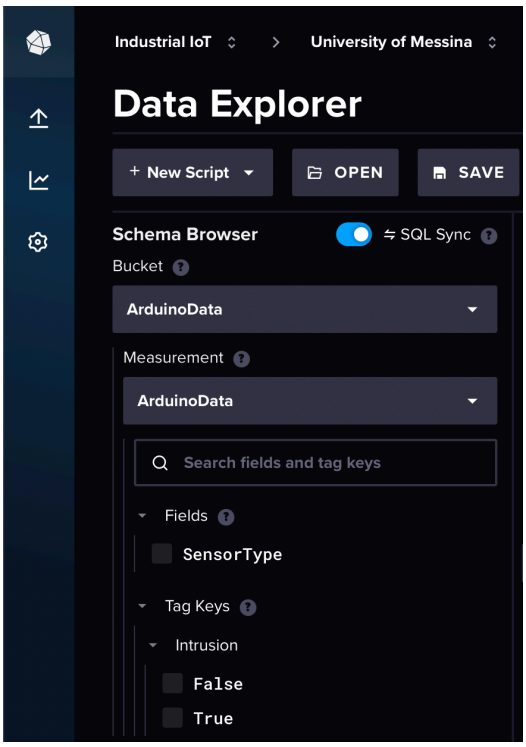
Finally, the *client.write()* method is called to write the *Point* object into the specified InfluxDB bucket, called *ArduinoData* as well, which is provided through the *Config.bucket* variable.

```python
# Simulating other sensors
types = ["Movement", "Noise", "Camera"]
sensortype = random.choice(types)

# Saving data into InfluxDB
data_influx = {
    "data": {
        "Intrusion": "True",
        "SensorType": sensortype
    }
}

for key in data_influx:
    point = (
        Point("ArduinoData")
        .tag("Intrusion", data_influx[key]["Intrusion"])
        .field("SensorType", data_influx[key]["SensorType"])
    )
client.write(database=Config.bucket, record=point)
```

Thus by means of the Data Explorer section, we can can have a graphical interface of the Bucket just created.

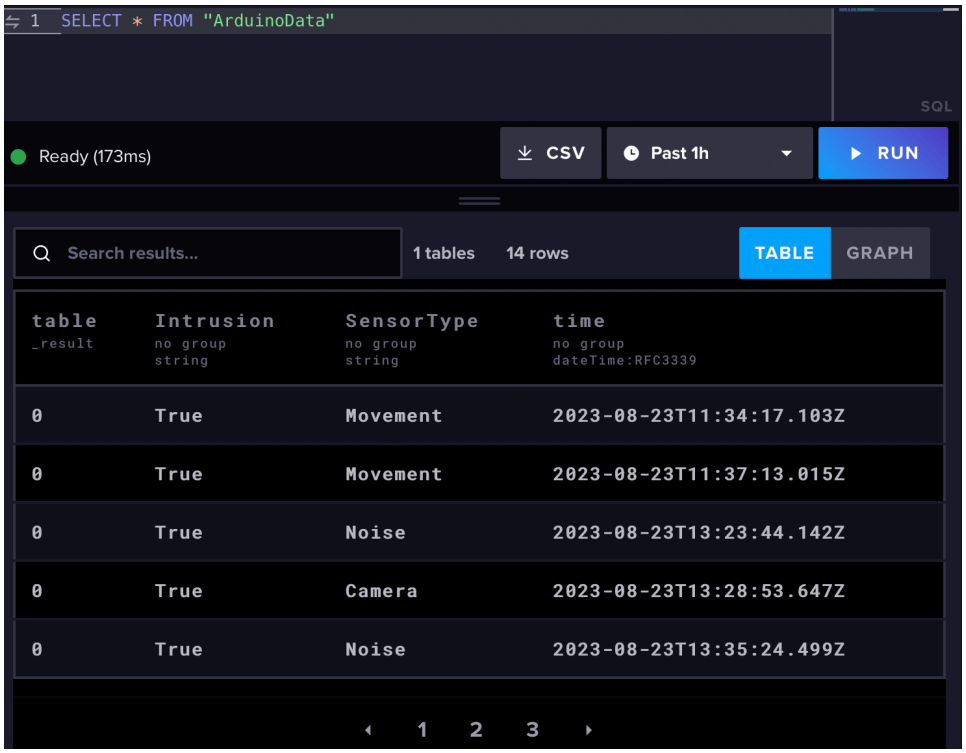
Within it a corresponding Measurement has been created, also named *ArduinoData*. Within this Measurement, a combination of Fields and Tags exists.

The Field is named *SensorType*, denoting a specific type of sensor reading. Concurrently, a Tag named *Intrusion* has been defined. This Tag has the capacity to adopt two Boolean values, namely *True* and *False*, serving as an indicator of the detection or absence of an intrusion event.



Data visualization through InfluxDB

In the Data Explorer section, there is also a segment where you can write SQL code to query the database. In fact, for instance, by writing SELECT * FROM “ArduinoData", is possible to display all the intrusion-related values contained within the "ArduinoData" Measurement, discriminating all the three columns of the database: Intrusion, SensorType a time.



Data visualization through Python

It is also possible to query the database by using Python. Specifically, the formatted strings (f-strings) has been used to create the *query* variable which retrieves data from the specified *Config.bucket* within a certain time range, filtered based on the measurement name, and sorted in descending order by time.

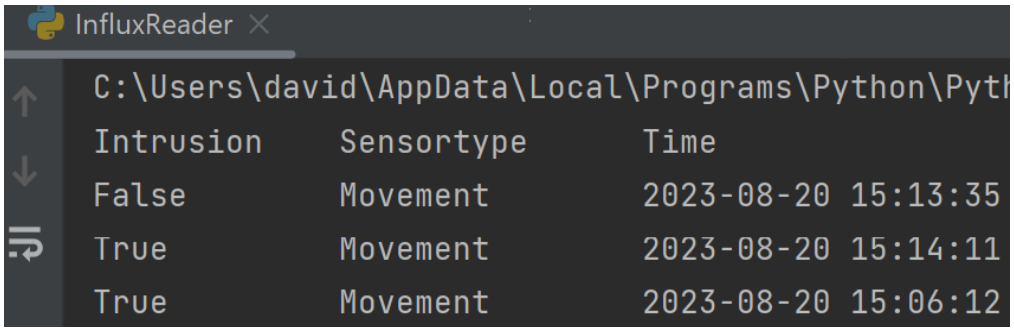
Once the query has been wrote it has to be executed by means of.

- *client.query_api()* which retrieves the query API object from the InfluxDB client instance
- *.query(query=query, org=Config.org_influx)* which executes the query using the query API object, passing the query string and the organization from Config.py file.

Eventually the query result is assigned to the *result* variable.

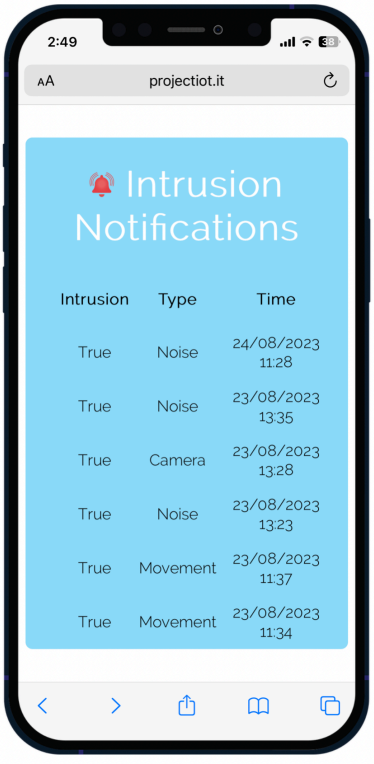
```
query = f'from(bucket:"{Config.bucket}") ▷ range(start: -30d) ▷ filter(fn: (r) => ...
    ... r["_measurement"] == "{Config.bucket}") ▷ sort(columns: ["_time"], desc: true)'
result = client.query_api().query(query=query, org=Config.org_influx)
```

Below is the Python console displaying the results obtained from the previously written query:



Data visualization through Altervista Hosting

Moreover, we have translated the Python code into JavaScript to develop a graphical dashboard using HTML hosted on Altervista. This facilitates remote access for viewing the state of the system. Similarly, in the JavaScript context, we have successfully achieved the intended outcome: a table is dynamically generated to display the results retrieved from the InfluxDB query. Notably, this is accomplished while utilizing the access token credentials provided by the service. As seen in the image depicted, it has been decided, for design purposes, to display only the data related to intrusions (*True*). However, even so, the boolean value *False* could prove useful to enable future implementations related to the proper working condition of the system.



Conclusions and Future Work

In conclusion, the project has represented a significant step towards the implementation of advanced security solutions in the context of Industry 4.0. Through the use of IoT technologies, a system for intrusion detection and notification has been developed, aiming to provide a higher level of protection for critical industrial environments. The main objective of this project was to demonstrate how IoT technologies can be effectively integrated to monitor industrial spaces and promptly detect unauthorized intrusions.

Throughout the development and implementation of the system, several challenging aspects were addressed, including the selection of appropriate sensors and the integration of real-time notification mechanisms. The results obtained from experimental trials have demonstrated the effectiveness of the system in detecting simulated intrusions and providing timely alerts to security personnel. This underscores the potential of IoT solutions in enhancing security in advanced industrial settings.

Despite the achieved successes, it is important to note that the proposed system could be further improved and optimized. Ongoing considerations are required to address challenges such as reducing false positives, expanding monitoring capabilities on a larger scale, and integrating with other pre-existing security systems. Additionally, the security of the IoT system itself must be taken into account, with special attention to vulnerabilities and cybersecurity threats that could compromise the integrity of the system.

Some of the directions that could be explored include:

- *Addition of detection algorithms*: Developing more sophisticated machine learning algorithms for intrusion detection to further reduce false positives and improve adaptability to new attack patterns.
- *Integration with artificial intelligence*: Exploring the use of machine learning techniques to enhance detection accuracy and enable a deeper understanding of intrusion patterns.
- *Expansion of sensor network*: Scaling up the system to cover a larger monitoring area by integrating a wider variety of sensors to detect not only physical intrusions but also other threats such as fires or chemical leaks.
- *System security*: Investing in the development of robust protection and authentication mechanisms for the IoT system itself to prevent cyberattacks and ensure data integrity.
- *Integration with business systems*: Exploring ways to integrate the data and alerts generated by the intrusion detection system with other business systems, such as human resource management or emergency management systems.

The current project serves as just a starting point in the quest for advanced IoT-based industrial security solutions. The ongoing evolution of technologies and threats will require continuous efforts to develop increasingly sophisticated and effective systems to ensure the protection of industrial environments in the era of Industry 4.0.