

# Eqlot



**Software Engineering for Internet of Things**

Beatrice Tomassi

(matr. 295351)

Lorenzo Rocchi

(matr. 294418)



# Table of Contents

Introduction.....	3
System Architecture & Description.....	4
1. Sensors .....	5
2. MQTT .....	6
3. Node-RED .....	7
4. Telegraf .....	8
5. InfluxDB.....	8
6. Grafana.....	8
Docker.....	11



## Introduction

EqIoT, short for “Earthquake Internet of Things”, is a smart system designed to keep a close eye on earthquakes around Campi Flegrei, Italy.

It uses sensors to feel ground movements and quickly alert a potential control centre whenever it senses a quake with a magnitude higher than 4.00 on the Richter’s scale. In this way, people in charge can monitor the situation and, eventually, warn authorities and law enforcement to keep everyone safe.

EqIoT operates with four sensor groups, each consisting of an accelerometer, a vertical geophone and a horizontal geophone, strategically positioned around Campi Flegrei, one at North, one at South, one at West, the last one at East.

Data collected are then transmitted to a database and visualized thanks to an intuitive dashboard that allows users to monitor real-time seismic activity in the area.

# System Architecture & Description

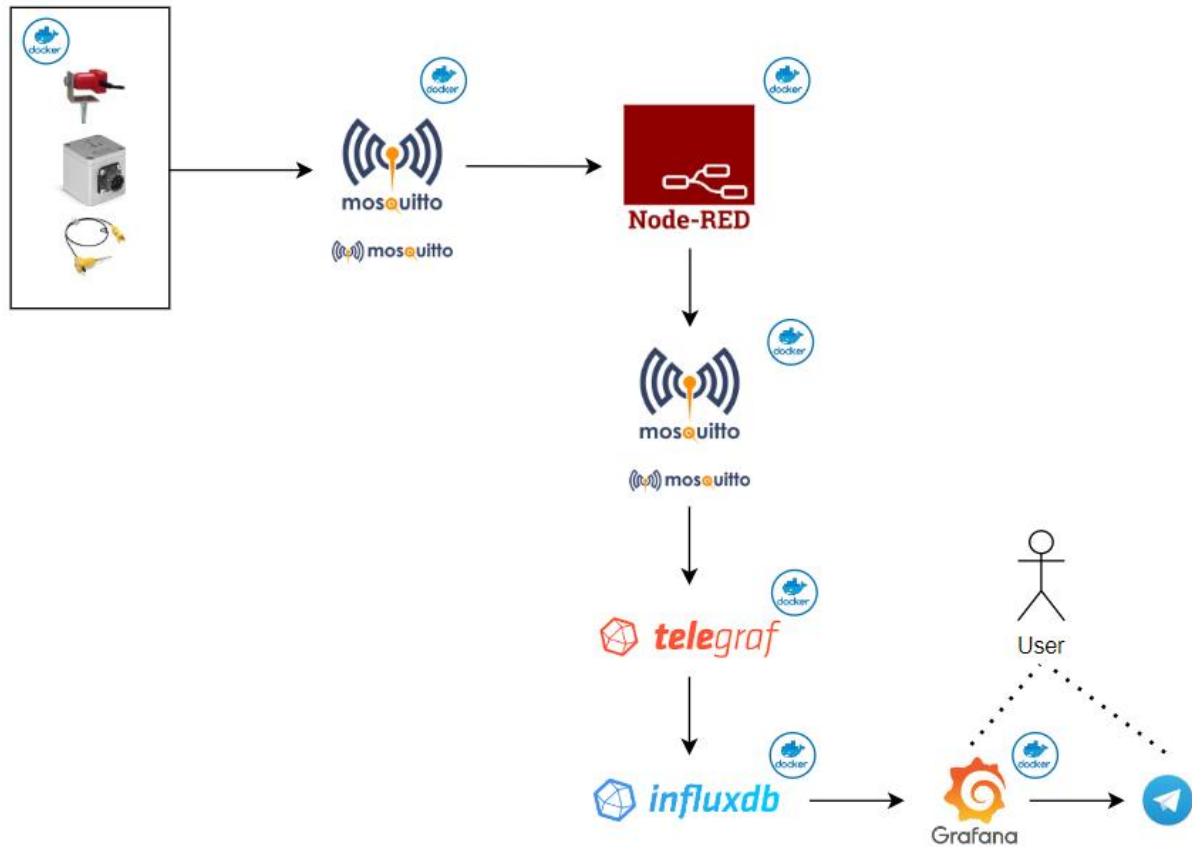


Figure 1 - System Architecture

The data flow in our system starts with the simulation of sensors through a Python script. The generated data is then transmitted to an MQTT broker and subsequently accessed through Node-RED. Here the data are aggregated, manipulated and forwarded to a new MQTT topic.

These data are then inserted into an InfluxDB bucket using a Telegraf configuration, which reads the data from the previously mentioned MQTT topic. At this stage, Grafana utilizes InfluxDB as a data source, enabling users to visualize the data through a user-friendly dashboard. Furthermore, it is Grafana that facilitates the sending of alarm notifications to users whenever an earthquake with a magnitude exceeding 4.0 is registered.

## 1. Sensors

As mentioned, the sensors simulated are of three kinds

- **Accelerometer**: provides data on the rate of change of velocity of ground movements, helping quantify the intensity of earthquakes;
- **Vertical Geophone**: captures vertical seismic waves, also called "Primary waves", responsible for the majority of damage;
- **Horizontal Geophone**: captures horizontal seismic waves, also called "Secondary waves", that typically follows primary waves causing additional ground shaking.

In python, the simulation of these sensors consists of four script: **accelerometro.py**, **geofonoX.py**, **geofonoY.py**, **sensors.py**.

The first three scripts contain methods for generating data using the random library. In the first case, a tuple of floats will be generated, while in the other two cases, a single float will be generated. However, in each case, there are two methods: one for generating "normal" data and one for generating "alarm" data.

In the "normal" data generation methods, the generated floats will be lower, representing typical or non-alarm conditions. Conversely, in the "alarm" data generation methods, the floats will be higher, representing abnormal or alarm conditions.

The **sensors.py** file not only simulates and sends data from accelerometer and geophone sensors to an MQTT broker but also configures the MQTT connection, defining its address and port.

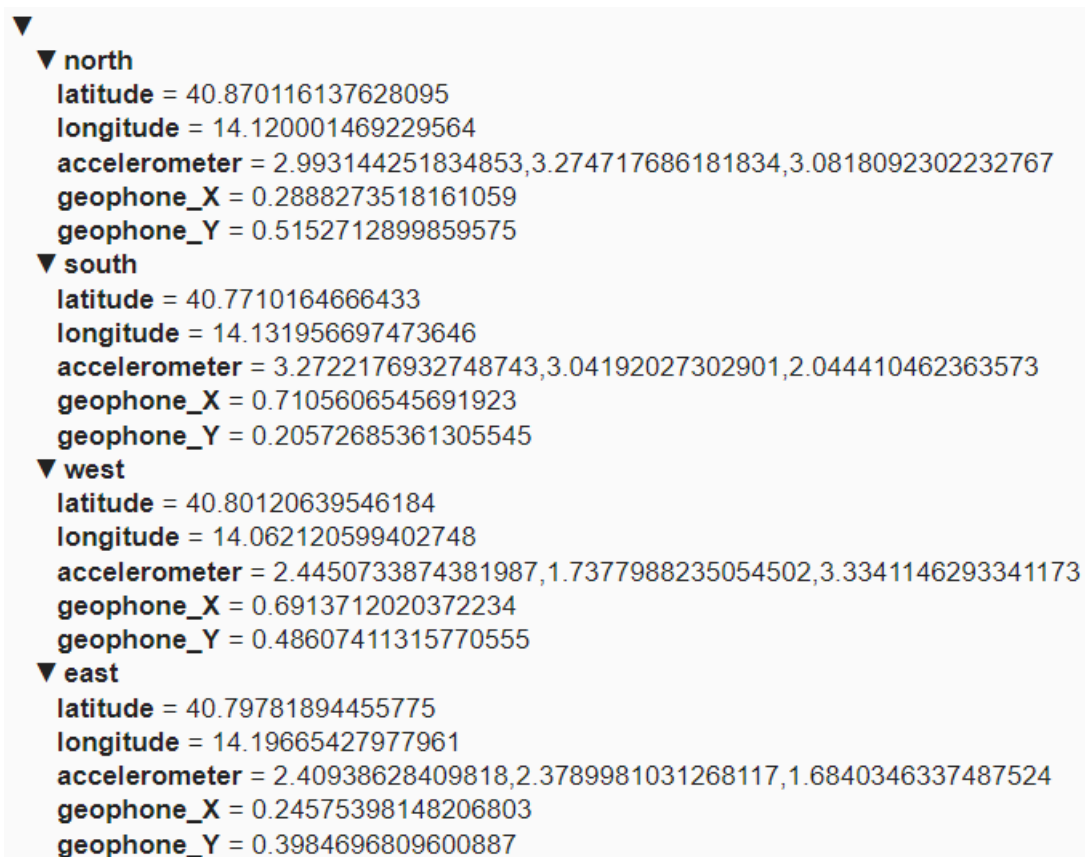
The program relies on a main loop, executed every 30 seconds. Within this loop, the locations defined within the *locations dictionary* are iterated through. This dictionary contains the coordinates (latitude and longitude) of four geographical positions: north, south, west, and east.

For each position, the *generate\_send\_Data()* function is called to generate and send normal data or alarm data, based on a random probability. It's important to note that if the function chooses to generate alarming data twice in a row for the same location, then the same data is sent again, this to ensure that no more than one alarming earthquake can be registered within 30 seconds.

Each sensor, for each location, has its own MQTT topic in the broker, following the notation: "*location/sensor*".

## 2. MQTT

The MQTT broker is executed within a Docker container running Mosquitto.



```
▼
▼ north
  latitude = 40.870116137628095
  longitude = 14.120001469229564
  accelerometer = 2.993144251834853,3.274717686181834,3.0818092302232767
  geophone_X = 0.2888273518161059
  geophone_Y = 0.5152712899859575
▼ south
  latitude = 40.7710164666433
  longitude = 14.131956697473646
  accelerometer = 3.2722176932748743,3.04192027302901,2.044410462363573
  geophone_X = 0.7105606545691923
  geophone_Y = 0.20572685361305545
▼ west
  latitude = 40.80120639546184
  longitude = 14.062120599402748
  accelerometer = 2.4450733874381987,1.7377988235054502,3.3341146293341173
  geophone_X = 0.6913712020372234
  geophone_Y = 0.48607411315770555
▼ east
  latitude = 40.79781894455775
  longitude = 14.19665427977961
  accelerometer = 2.40938628409818,2.3789981031268117,1.6840346337487524
  geophone_X = 0.24575398148206803
  geophone_Y = 0.3984696809600887
```

Figure 2 - Screenshot of MQTT Explorer

### 3. Node-RED

Node-RED collects, manipulates, aggregates, and forwards data to a new MQTT topic. Specifically, the flow starts with *MQTT-in* nodes, each dedicated to gathering data from a specific sensor. The positional data and the ones collected from the geophones remains untouched, while the tuple of floats originating from the accelerometer is utilized to compute earthquake magnitude through a *function* node. It's important to note that the performed calculation is a simplified abstraction rather than a precise seismic magnitude computation.

Subsequently, the data is aggregated into four JSON files, one for each location, using a *join* node. Each JSON undergoes formatting by a function node that sets the payload of the message, forwarded to a single MQTT out node representing the "*telegraf/SensorData*" topic.

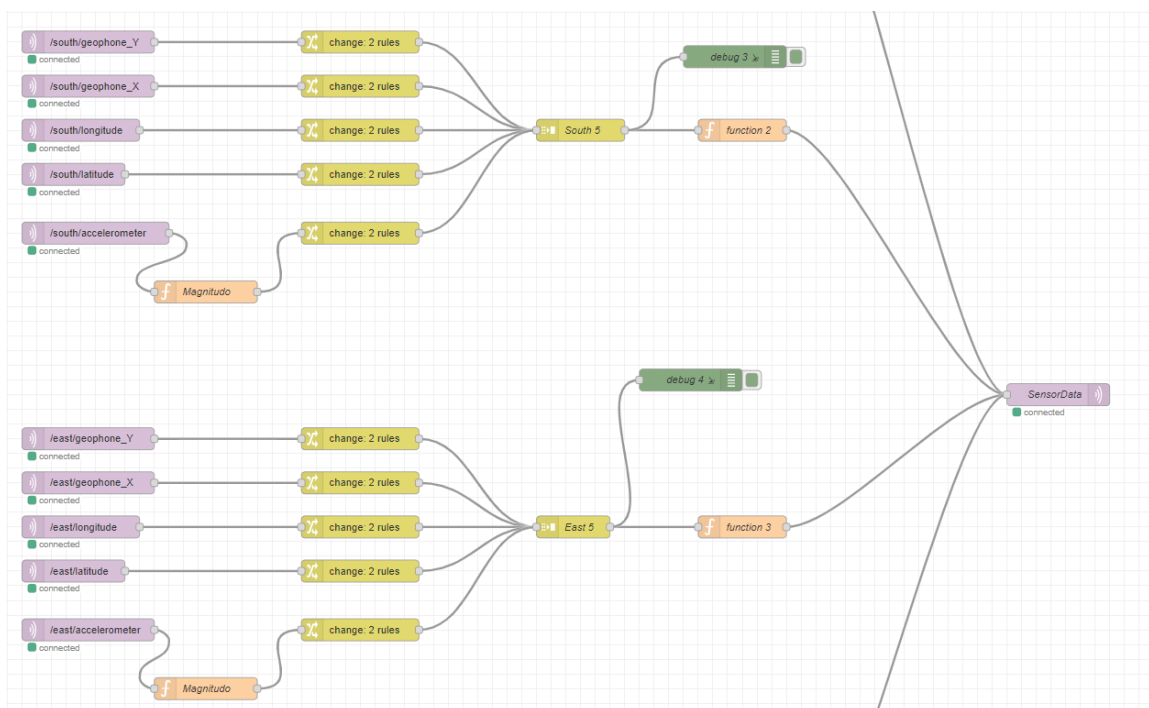


Figure 3 - Screenshot of a portion of the Node-RED flow



## 4. Telegraf

Telegraf is configured to forward the gathered data to InfluxDB's buckets using specific API tokens.

## 5. InfluxDB

Using InfluxDB, the system stores informations about the seismic events in the *EarthQuake* bucket.



Figure 4 - Screenshot of InfluxDB visualization

Each measurement is stored alongside its timestamp.

## 6. Grafana

Grafana allows a user-friendly visualization of the seismic activity. Our InfluxDB database is set up to be the data source for the dashboard. The dashboard holds a series of panels, each of which queries the database to gather data, using the Flux query language.

Here we provide an example of a query that gathers and formats magnitude and geophone data from the East group of sensors.



```

1  from(bucket: "EarthQuake")
2    |> range(start: -1h)
3    |> last()
4    |> filter(fn: (r) => r.location == "East")
5    |> group(columns: ["location"])
6    |> map(fn: (r) => ({
7      geophoneX: if r._field == "geophone_X" then r._value else 0.0,
8      geophoneY: if r._field == "geophone_Y" then r._value else 0.0,
9      magnitude: if r._field == "magnitude" then r._value else 0.0
10   }))
11   |> group(columns: ["_start"])
12   |> reduce(
13     identity: {geophoneX: 0.0, geophoneY: 0.0, magnitude: 0.0},
14     fn: (r, accumulator) => ({
15       geophoneX: if r.geophoneX != 0.0 then r.geophoneX else accumulator.geophoneX,
16       geophoneY: if r.geophoneY != 0.0 then r.geophoneY else accumulator.geophoneY,
17       magnitude: if r.magnitude != 0.0 then r.magnitude else accumulator.magnitude
18     })
19   )
20   |> map(fn: (r) => ({
21     _time: r._time,
22     geophoneX: r.geophoneX,
23     geophoneY: r.geophoneY,
24     magnitude: r.magnitude
25   })))

```

Figure 5 - Query Example

The dashboard contains an informed map, depicting the four sensor groups as markers. Size and colour of the markers depends on the last registered magnitude.

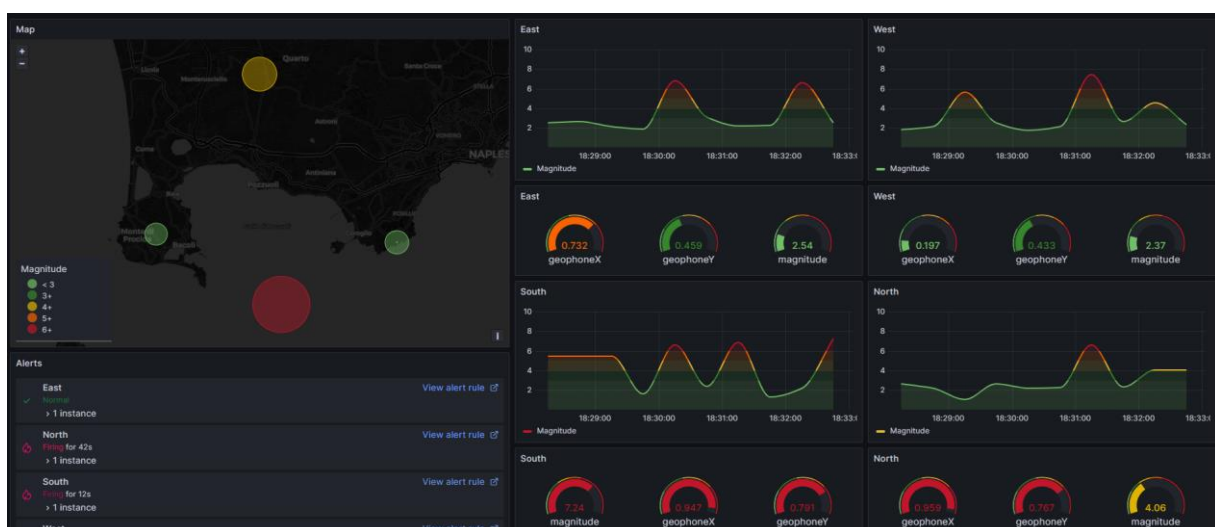


Figure 6 - Screenshot of Grafana Dashboard

For each group of sensors, the dashboard contains two panels: one dedicated to a time series representation of the oscillating value of the magnitude, the other showing values for magnitude, horizontal geophone and vertical geophone as gauges.

In Grafana is also possible to define alert rules. In particular, we defined alerts for earthquakes with a magnitude higher than 4.0.

When an alarming quake is registered, thanks to an appropriate configuration of a Telegram Channel, Grafana forward an alerting notification that follows the template shown in Figure 7.

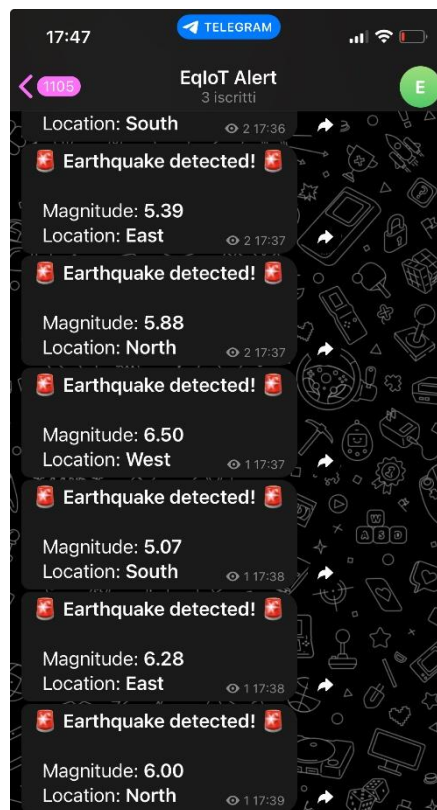


Figure 7 - Screenshot of Telegram

# Docker

All the system components, including the python environment representing the data generation, are completely containerized using docker. This enables portability and easy deployment on other machines.


















Name	Image	Status	↑ CPU (%)	Port(s)
 <a href="#">eqiot_earthquakemonitoring</a>		Running (6/6)	8.77%	
 <a href="#">eqiot-telegraf</a> 80ea5ce9964f 	<a href="#">telegraf:latest</a>	Running	0.27%	
 <a href="#">eqiot-mosquitto</a> 14bfe922aab6 	<a href="#">eclipse-mosquitto:latest</a>	Running	0.39%	<a href="#">1883:1883</a> 
 <a href="#">eqiot-python</a> 8b5f24e4afc0 	<a href="#">eqiot_earthquakemonitoring-python</a>	Running	0.44%	
 <a href="#">eqiot-grafana</a> 1117adeac238 	<a href="#">grafana/grafana-enterprise:latest</a>	Running	0.98%	<a href="#">3000:3000</a> 
 <a href="#">eqiot-influxdb</a> 130e727019b2 	<a href="#">influxdb:latest</a>	Running	3.26%	<a href="#">8086:8086</a> 
 <a href="#">eqiot-nodered</a> 0c43e56ca346 	<a href="#">eqiot_earthquakemonitoring-node-red</a>	Running	3.43%	<a href="#">1880:1880</a> 

Figure 8 - Docker environment in execution

All the steps needed to set the system up are contained in the README.md file [of the system's repository](#).

