Internet of Things

IoT-Based Indoor Air Quality Monitoring System

Andrea Di Ubaldo, Riccardo Baratin

aa 2021/2022

Goals

- Implementation of an Air Quality monitoring system based on IoT environment using a micro-controller with temperature and gas sensors
- Multi-protocol communication between Edge-nodes and Proxy Server
- Storing sensor data on Influx Database
- Data forecasting using two different prediction models
- Visualization of stored data on Grafana Dashboards
- Implementation of Telegram Bots for monitoring data and alert notification

ARCHITECTURE

Architecture: micro-controller and sensor

It has been used an ESP-32 micro-controller with DHT-22 sensor for temperature and humidity measurements and MQ-2 sensor for gas concentration measurement.

Also, ESP-32 measures the Wi-Fi strength signal (RSS) and the AQI on gas concentration following this formula:

$$AQI(t) = \begin{cases} 1 \text{ if } avg \geq MAX\_GAS\_VALUE \\ 2 \text{ if } avg \leq MIN\_GAS\_VALUE \\ 0 \ otherwise \end{cases}$$

Architecture: Influx Database and Grafana Dashboard

Proxy Server and Telegram Bots communicate with an InfluxDB component using Flux query language.

InfluxManager component exposes the following capabilities:

- Writing new measurements distinguishing sensor data by prediction ones
- Querying buckets on data stored in a given interval

Grafana Dashboard is used to visualize sensor data and forecasting ones in Influx buckets

Architecture: protocols management

ESP-32 micro-controller communicates collected data to the Proxy Server via *MQTT* or *HTTP* protocol (chosen at run-time).

Notice that both protocols are TCP-base and ESP-32 can only maintain a single TCP session at a time.

The ESP-32 is capable of multiple simultaneous TCP connections, but to use more than one, more than one *WiFiClient* instance must be created.

MQTT protocol

The communication starting protocol.

It has been used to send collected data by the ESP-32 sensors to the Proxy Server and to manage all the capabilities.

Data transmission is related to a single topic, meanwhile the capabilities have multiple related topics.

HTTP protocol

HTTP is the second protocol implemented for the sensor data transmission.

When the protocol is running the edge-node will start sending data via HTTP post request to the Proxy Server.

HTTP protocol is only used to send measurement to the Proxy Server.

Architecture: Proxy Server

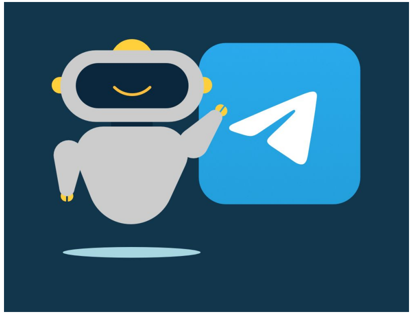
Proxy Server (gateway) exposes the following capabilities:

- Sensor data storing into InfluxDBs
- Hyper-parameters tuning to change sample frequency and min and max gas values
- Communication protocol switching
- Evaluation mode switching to get packages loss and latency stats
- Getting mean outdoor temperature values calling external API

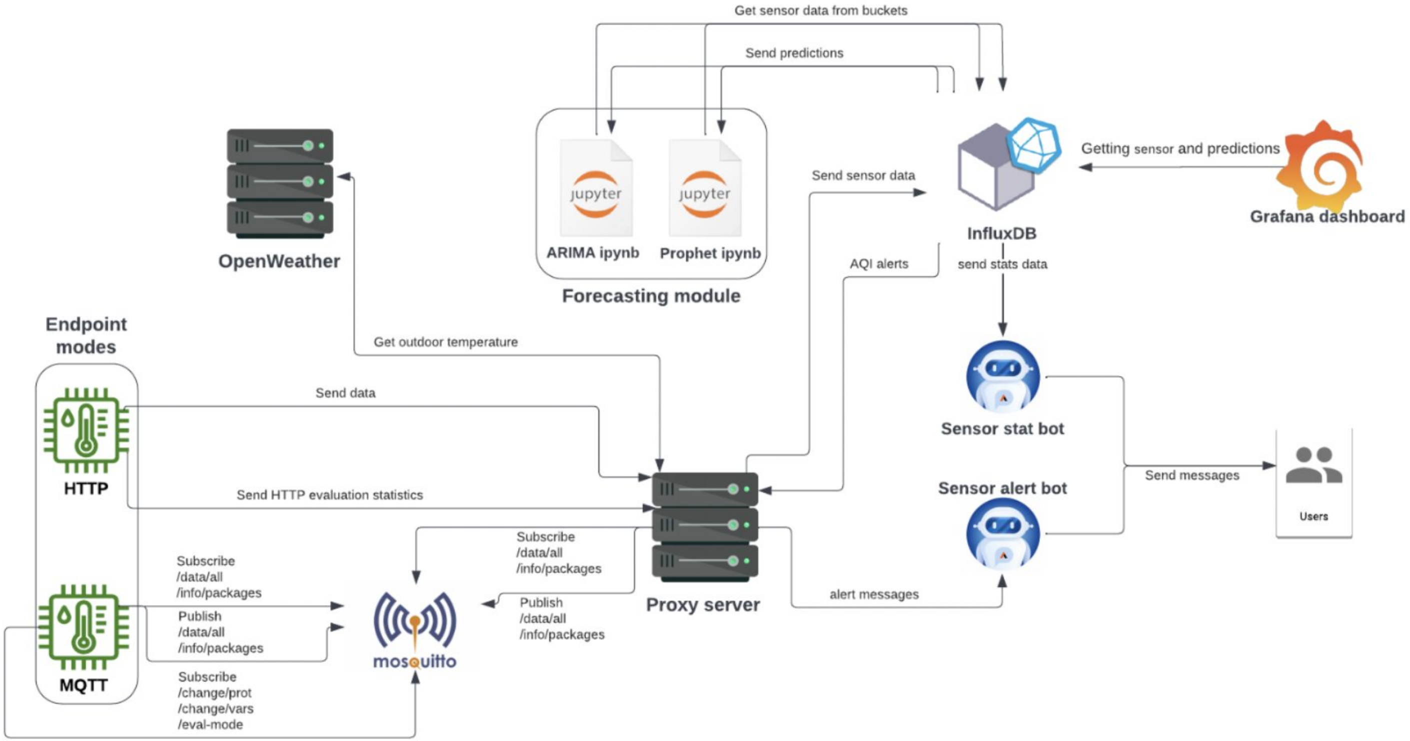
Architecture: Telegram Bots

Two Telegram Bots to managing different services:

- **gas_alert_bot**: manages gas alert notification sent by the Influx Alerts when the AQI value is equal to one or two
- **sensor_stat**: provides buckets mean values on users requests



Architecture: Network



IMPLEMENTATION

Implementation: ESP-32, DHT-22 and MQ-2 sensors

It has been used *AsyncMqttClient.h* library for MQTT communication and *FreeRTOS.h* and *timers.h* libraries for a better timing computation

ArduinoJson.h for data transmission in easy way

DHT.h for temperature and humidity measurements, analogical readings by ESP-32 pins and *WiFi.h* for WiFi strength signal estimation

```
---------------------------
Message arrived on topic: sensor/data/all
Temp: 18.70
Hum: 52.40
Gas: 93
WiFi RSS: -71
Protocol: MQTT
---------------------------
```

```
---------------------------
Temp: 18.40
Hum: 63.30
Gas: 112
WiFi RSS: -66
Protocol: HTTP
---------------------------
```

Implementation: Proxy server

Proxy server uses the following libraries:

- *express.js* and *http* for HTTP communication with devices and external services
- *mqtt* for MQTT communication with ESP-32 and broker
- *influxdb-client* for InfluxManager.js operations

Implementation: Forecast notebooks

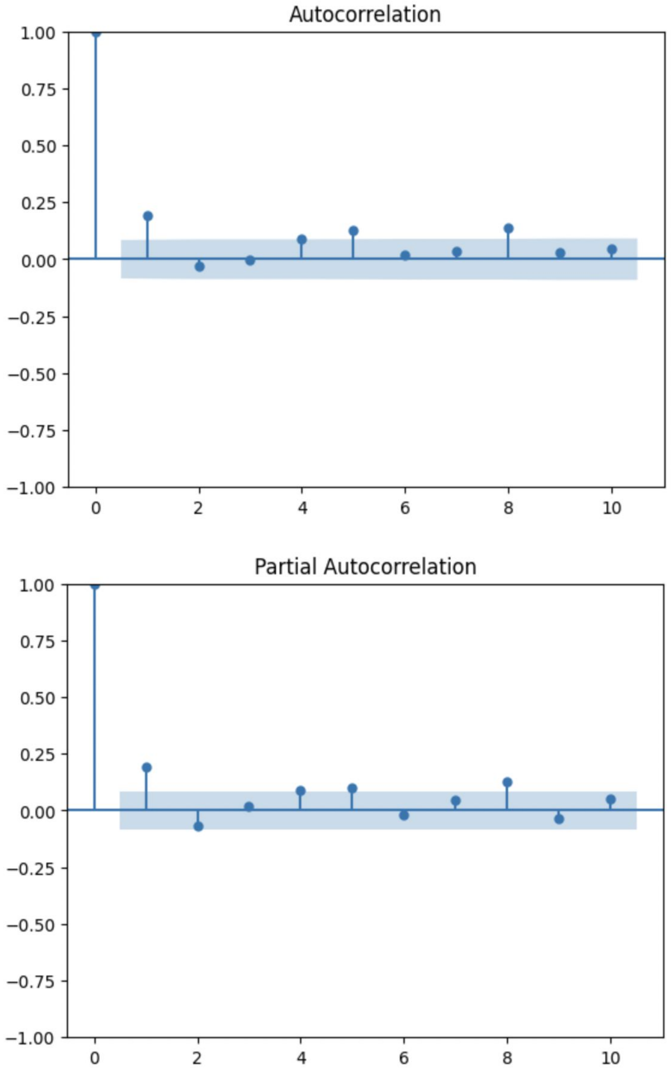
Forecasting model are implemented on jupyter notebooks: *ARIMA_Forecast.ipynb* and *prophet_forecast.ipynb.* In addition:

- It was used *InfluxDBClient* for data retrieval from Influx Database
- *Panda* library for data management
- *fbprophet* library for prophet building
- *statsmodels* library for ARIMA building

Forecasting models testing: ARIMA

Main steps:

- Dickey-Fuller test, which tests the null hypothesis that the series is Not-Stationary
- Autocorrelation and Partial Autocorrelation tests
- Evaluation of the model (RMSE and Confidence Interval)



Forecasting models testing: Prophet

Main steps:

- Fit the model on train dataset setting the right parameters
- Specify the forecasting period
- Evaluation of the model (RMSE and Confidence Interval)

```
m = Prophet(
    yearly_seasonality=False,
    weekly_seasonality=False,
    daily_seasonality=True,
    changepoint_range=1,
    changepoint_prior_scale=0.01
).fit(train)
```

```
future = m.make_future_dataframe(periods=test_interval,
            freq=DateOffset(minutes=1))
```

```
forecast = m.predict(future)
```

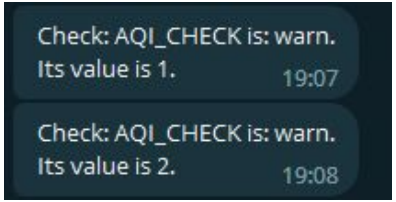
Implementation: Influx Database

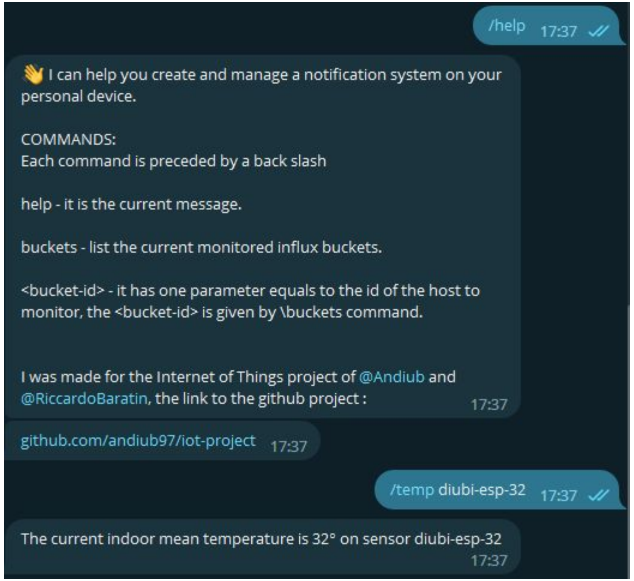
It was used Flux language to make queries on 6 buckets:

- **temperature**
- **out_temperature**
- **humidity**
- **gas**
- **aqi**
- ***rss***

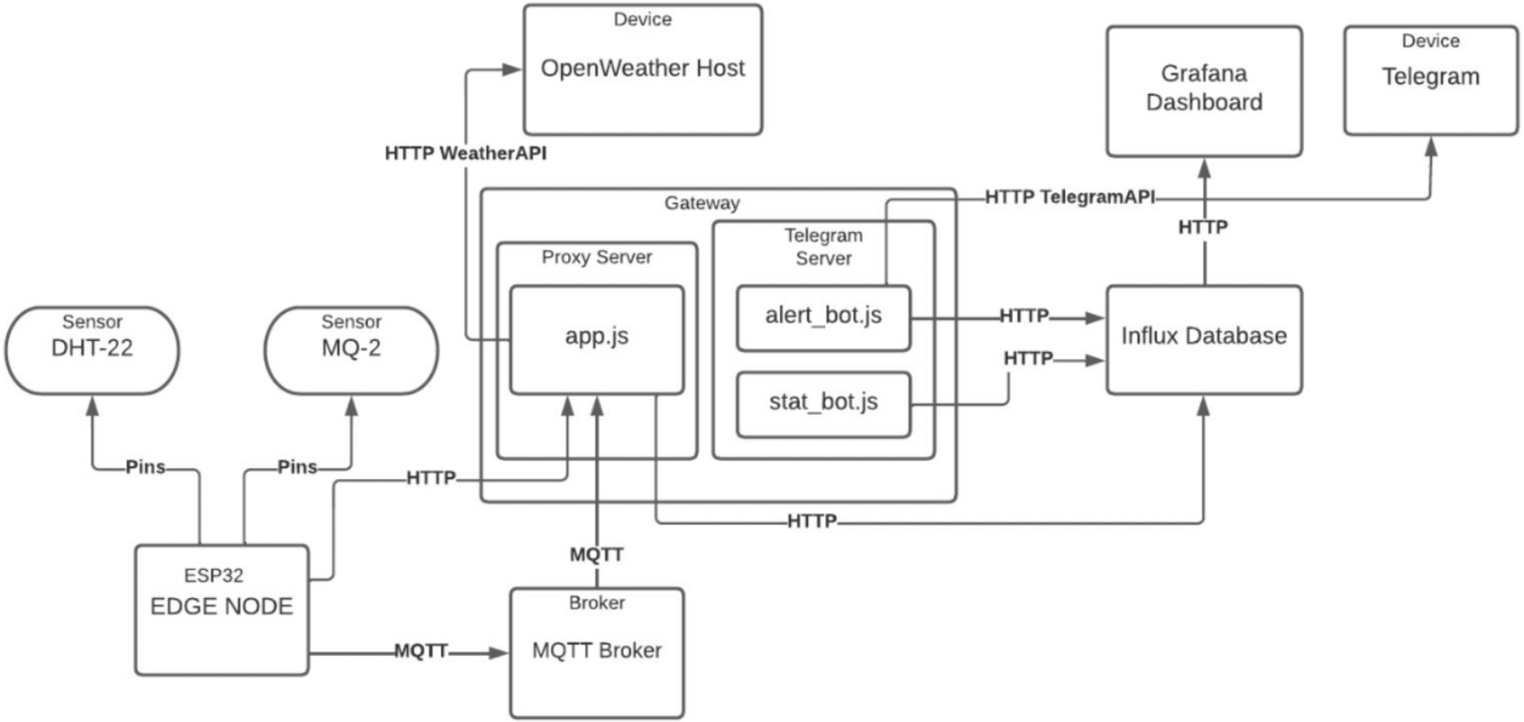
InfluxDBClient APIs and WritePoint object for writing data on buckets

Implementation: Telegram bots





Deployment Diagram



RESULTS

Results: Protocol metrics and evaluation

Protocol Metrics:

- Package Loss
- Connection Latency

Protocol	Latency	Package Loss
MQTT	10 ms(± 2)	0 %
HTTP	32 ms(± 4)	2 %

Consideration:

- MQTT seems to perform better than HTTP in terms of lost packets percentage and connection latency
- MQTT requires more time (maybe also power) than HTTP to establish a session.

Results: Forecasting metrics and evaluation

Forecasting Metrics:

- RMSE
- Prediction Mean
- Interval of Confidence

Two different forecasting model, ARIMA and Prophet, were trained by passing real data collected over two days.

The **goal**, for both models, was to predict values for the next few hours.

Results: ARIMA model vs Prophet model

ARIMA Results

Model Type	RMSE	Prediction Mean	Interval of Confidence
Temperature	0.196	19.19	[18.81, 19.57]
Humidity	0.565	52.09	[50.16, 54.02]
Gas	21.541	88	[52, 123]

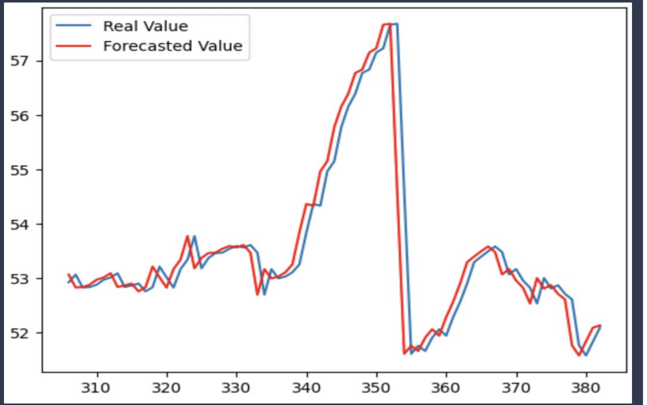
Prophet Results

Model Type	RMSE	Prediction Mean	Interval of Confidence
Temperature	0.603	19.56	[19.50, 19.62]
Humidity	2.488	54.89	[54.44, 55.34]
Gas	21.524	71	[69, 73]

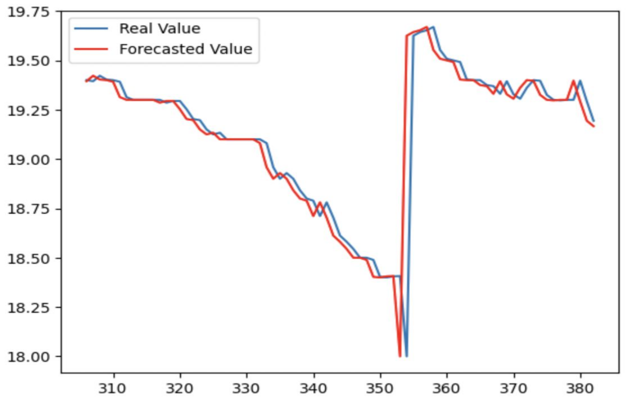
ARIMA model

Prophet model

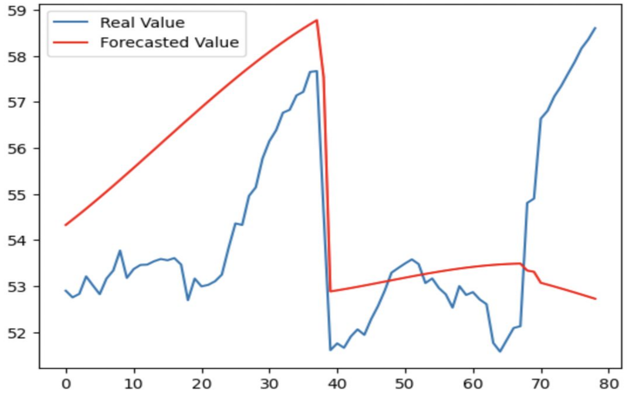
Humidity



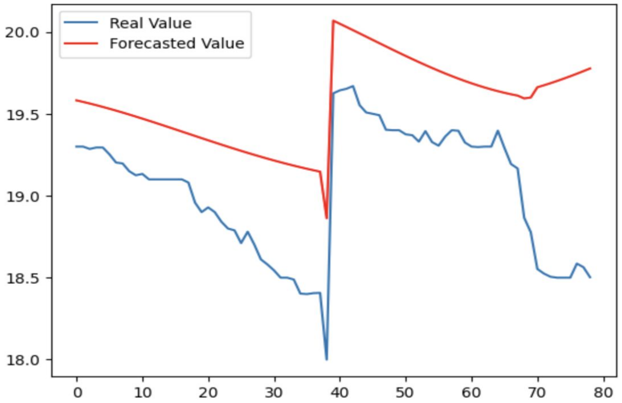
Temperature



Humidity



Temperature



Let's try the demo!