SQL Presto Tech

University Of Camerino

UNICAM Università di Camerino 1336

Computer Science

Technologies for Big Data Management

A.Y. 2022/2023

Team

Avdil Mehemeti
avdil.mehmeti@studenti.unicam.it

Daniele Pelosi
daniele.pelosi@studenti.unicam.it

Kiran Jose Puthussery
kiranjose.puthussery@studenti.unicam.it

Supervisor

Massimo Callisto De Donato
dtmassimo.callisto@unicam.it

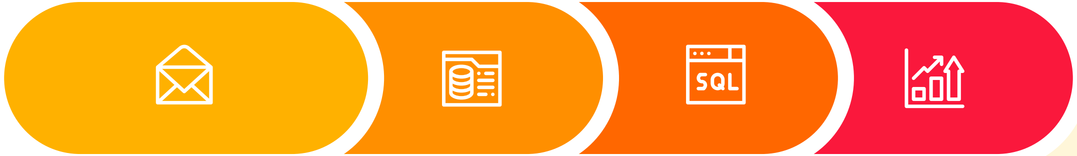
CONTENTS

1. Project Description

SQL Presto Tech

A **tool** that enables **Real-Time Data Analytics** on data from devices **IoT** devices using technologies such as Kafka, MongoDB and Presto

Objectives



Messages
Handle and publish messages in real-time on Kafka Topics

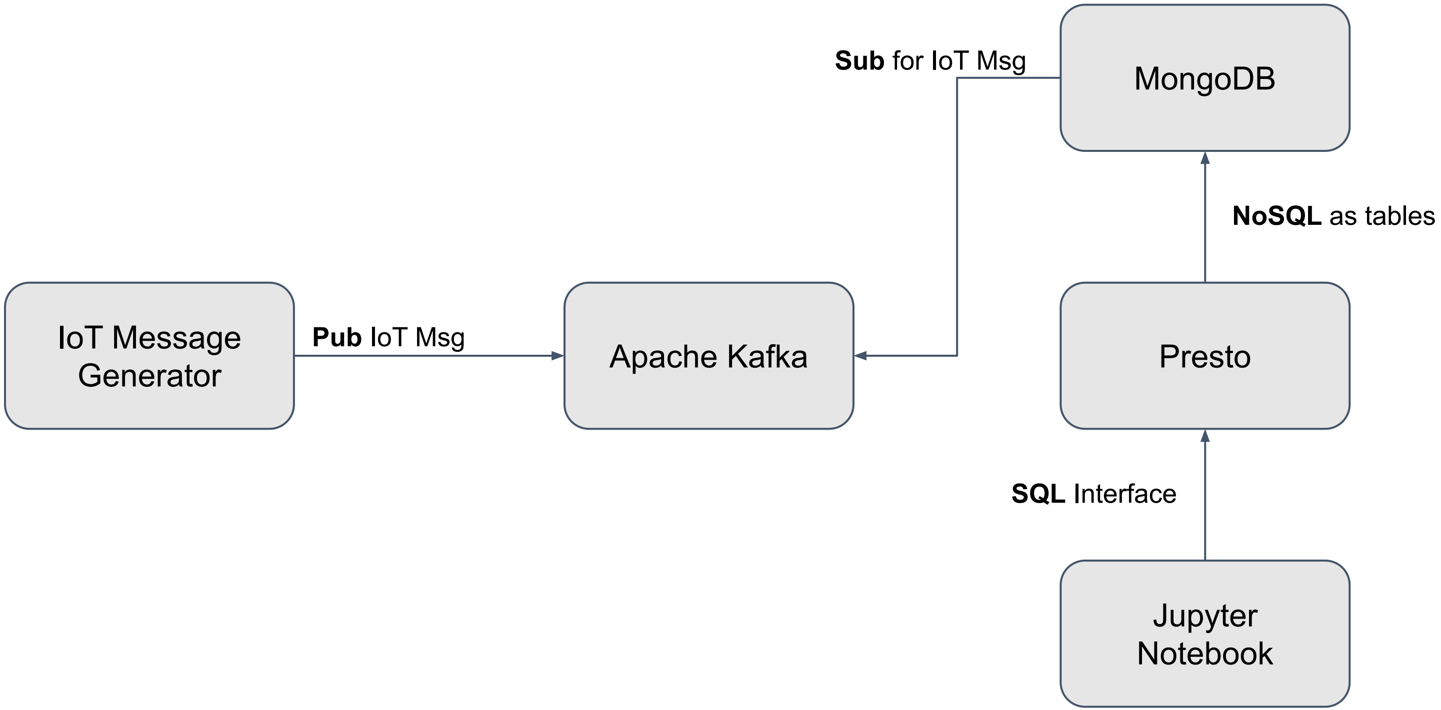
Storaging
Store IoT data in a NoSQL Database

Presto
Study and use Presto's SQL engine to perform queries on a NoSQL Database

Analytics
Create a tool for analytics

2. Methodologies & Technologies

General Architecture



Technologies

Apache Kafka

It is a distributed data streaming platform that can publish, subscribe to, store, and process streams of records in real time.

MongoDB

It is an open source NoSQL database that uses a non-relational, document-oriented data model to stores data objects

Presto

It is a distributed SQL query engine that is open-source and optimized for high-speed analytic queries of data of any size.

Jupyter Notebook

It is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and text.

3. Technical Implementation

Apache Kafka

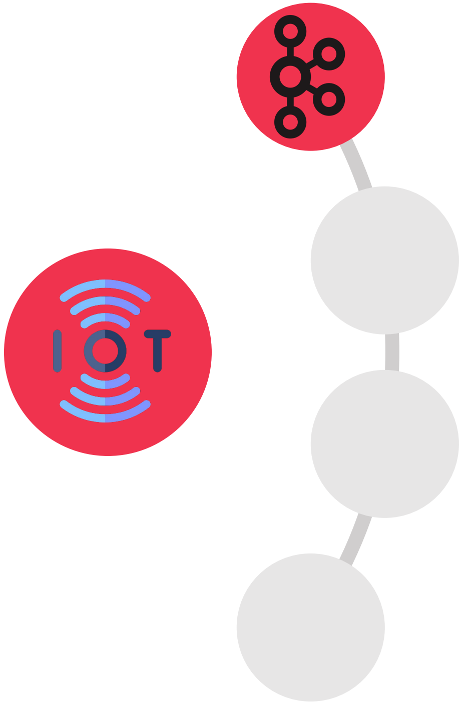
A distributed **publish-subscribe messaging system** used to stream messages that comes from the **iot-simulator**

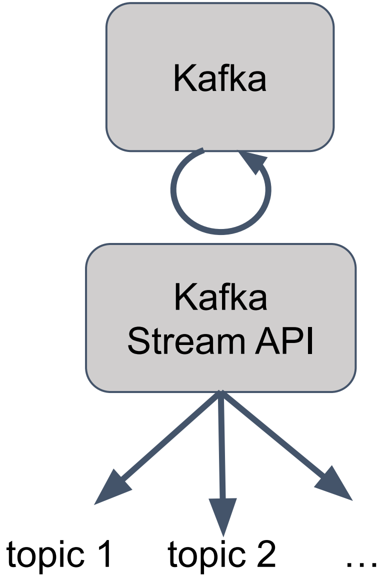
- In order to stream the messages is needed a **MQTT-Source connector** that connects to a MQTT broker and subscribes to the specified topics

```
{
  "name": "mqtt-source",
  "config": {
    "connector.class": "io.confluent.connect.mqtt.MqttSourceConnector",
    "mqtt.server.uri": "tcp://localhost:1883",
    "mqtt.topics": "#",
    "kafka.topic": "mqtt.echo",
    "value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter",
    "key.converter":"org.apache.kafka.connect.storage.StringConverter",
    "confluent.topic.bootstrap.servers": "localhost:9092",
  }
}
```

Apache Kafka

Kafka Stream API





- It **simplifies** the development of real-time streaming applications.

- With Kafka Streams API, is possible to perform various operations on the data streams, such as **filtering**, **transforming**, **aggregating**, and **joining**, to enable **real-time analytics** and processing.

Apache Kafka

Kafka Stream API

General Mapper

```java
@Override
public String apply(String value) {
    // Parse the message string into a JSON object
    JSONObject jsonObj = new JSONObject(value);

    // Remove the "measures" property from the JSON
    jsonObj.remove("m");

    // Convert the modified JSON object to a string
    String firstPart = jsonObj.toString();

    // Return the the message without the measures
    return firstPart;
```

Measures Mapper

```java
@Override
public Iterable<String> apply(String value) {
    // Parse the message string into a JSON object
    JSONObject jsonObj = new JSONObject(value);

    // Extract the value of the "uuid"
    this.uuidValue = jsonObj.optString("uuid", null);

    // Extract the measures part of the message
    this.measures = jsonObj.optString("m", null);

    // Convert the measures part into a JSONArray
    JSONArray jsonArray = new JSONArray(this.measures);

    // Create a list to store the transformed JSON objects
    List<String> jsonObjects = new ArrayList<>();

    // Iterate over each JSON object in the JSONArray
    for (int i = 0; i < jsonArray.length(); i++) {
        // Get the current JSON object
        JSONObject jsonObject = jsonArray.getJSONObject(i);

        // Add the UUID value to the JSON object
        jsonObject.put("uuid", uuidValue);

        // Convert the JSON object to a string & add it to list
        jsonObjects.add(jsonObject.toString());
    }
    // Return the list of transformed JSON objects
    return jsonObjects;
```

Kafka Stream App

```java
// Create a KStream that reads from the "mqtt.echo" topic
KStream<String, String> sourceStream = builder.stream("mqtt.echo");

// Create a ValueMapper to extract the first part of the message
ValueMapper<String, String> iotmessageMapper = new IoTMessageMapper();

// Create a ValueMapper to extract measures from the message as an Iterable
ValueMapper<String, Iterable<String>> measuresMapper = new MeasuresMessageMapper();

sourceStream
    .mapValues(iotmessageMapper)
    .to("mqtt.main", Produced.with(Serdes.String(), Serdes.String()));

sourceStream
    .flatMapValues(measuresMapper)
    .to("mqtt.measures", Produced.with(Serdes.String(), Serdes.String()));

// Create a KafkaStreams instance with the built StreamsBuilder and configuration
KafkaStreams streams = new KafkaStreams(builder.build(), config);

// Start the Kafka Streams application
streams.start();
```

Mongo DB

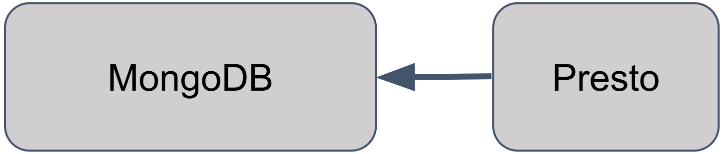
MongoDB is a document-oriented NoSQL database used to **store messages streamed by Kafka.**

- In order to address messages inside each mongodb collection is necessary to use the **MongoDB Sink Connector,** that reads data from Apache Kafka topic and writes data to MongoDB

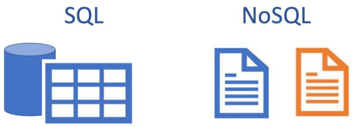
```
{
  "name": "mongodb-sink-2",
  "config": {
    "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
    "topics": "mqtt.measures",
    "connection.uri": "mongodb://localhost:27017",
    "database": "tbdmproject",
    "collection": "measures",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter"
  }
}
```

Presto

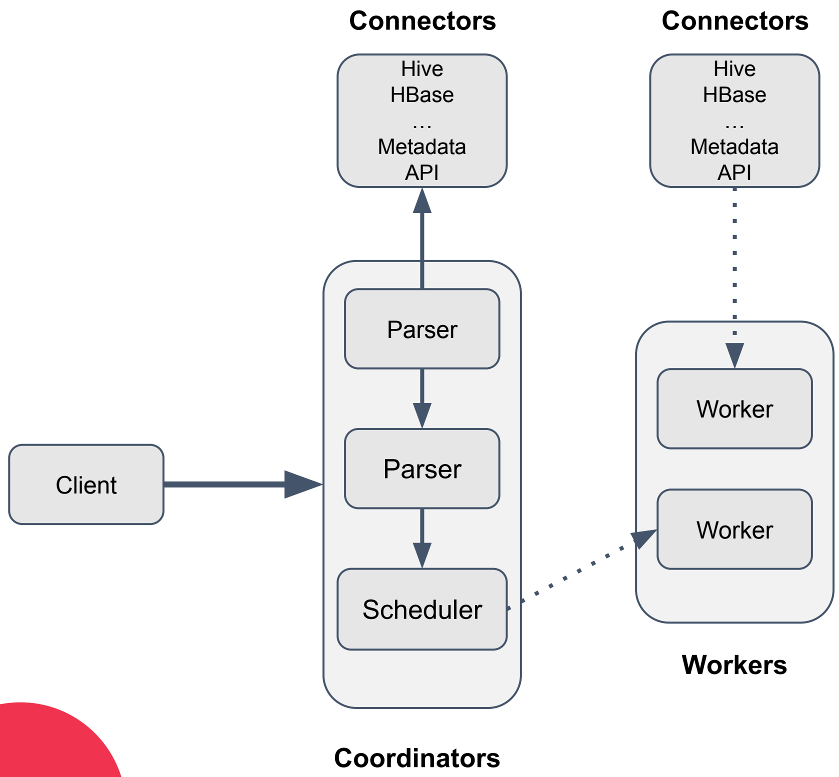
Presto is a distributed SQL query engine that supports non-relational sources.



- That is how Presto allows to **perform SQL computation over MongoDB** (NoSQL).



Presto
Architecture



- **Client:** (Presto CLI) submits SQL statements to a coordinator to get the result
- **Coordinator**: is a master daemon. It Parses the SQL queries then plans for the execution. Scheduler performs pipeline execution.
- **Workers**: The workers get actual data from the connector and delivers result to the client.
- **Connectors**: provides metadata and data for queries. The coordinator uses the connector to get metadata for building a query plan

Presto
Experienced Limitations

UPDATE operations not supported

Lack of community supports

Not optimal for large amount of queries at simultaneously

Jupyter Notebook

Jupyter Notebook is used to produce live code able to perform analytics

MongoDB ← Presto ← Jupyter

- In Jupyter is written the code for the connection between Presto and MongoDB that allow to write SQL queries to retrieve data

```
presto_conn = presto.connect(
    host='165.232.118.33',
    port=8090,
    catalog='mongodb',
    schema='tbdmproject'
)
presto_cur = presto_conn.cursor()
```

- In order produce chart and graphs are used libraries like pandas and plotly

4. Approaches

Approaches

In order to reach the final result, two different approaches have been identified and implemented.

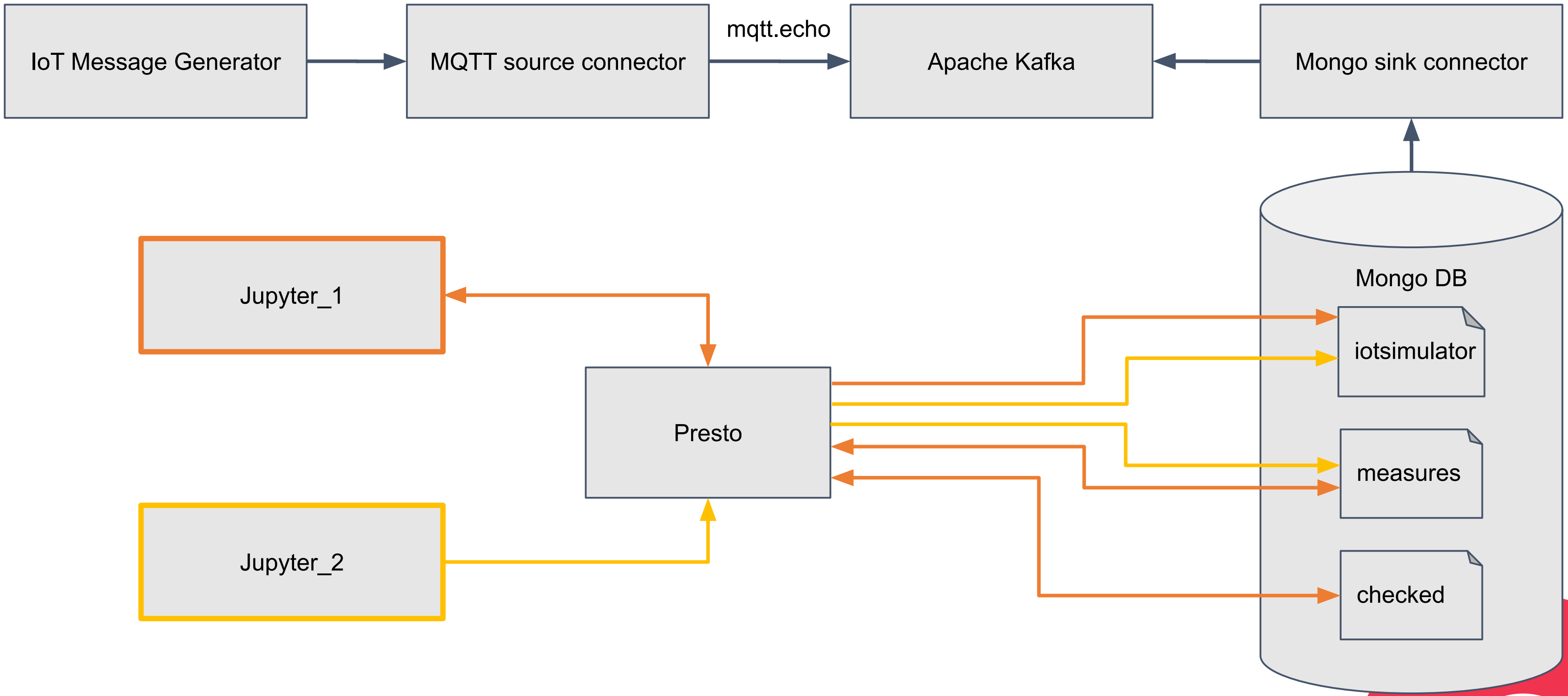
First Approach

Use of two different Jupyter Notebooks

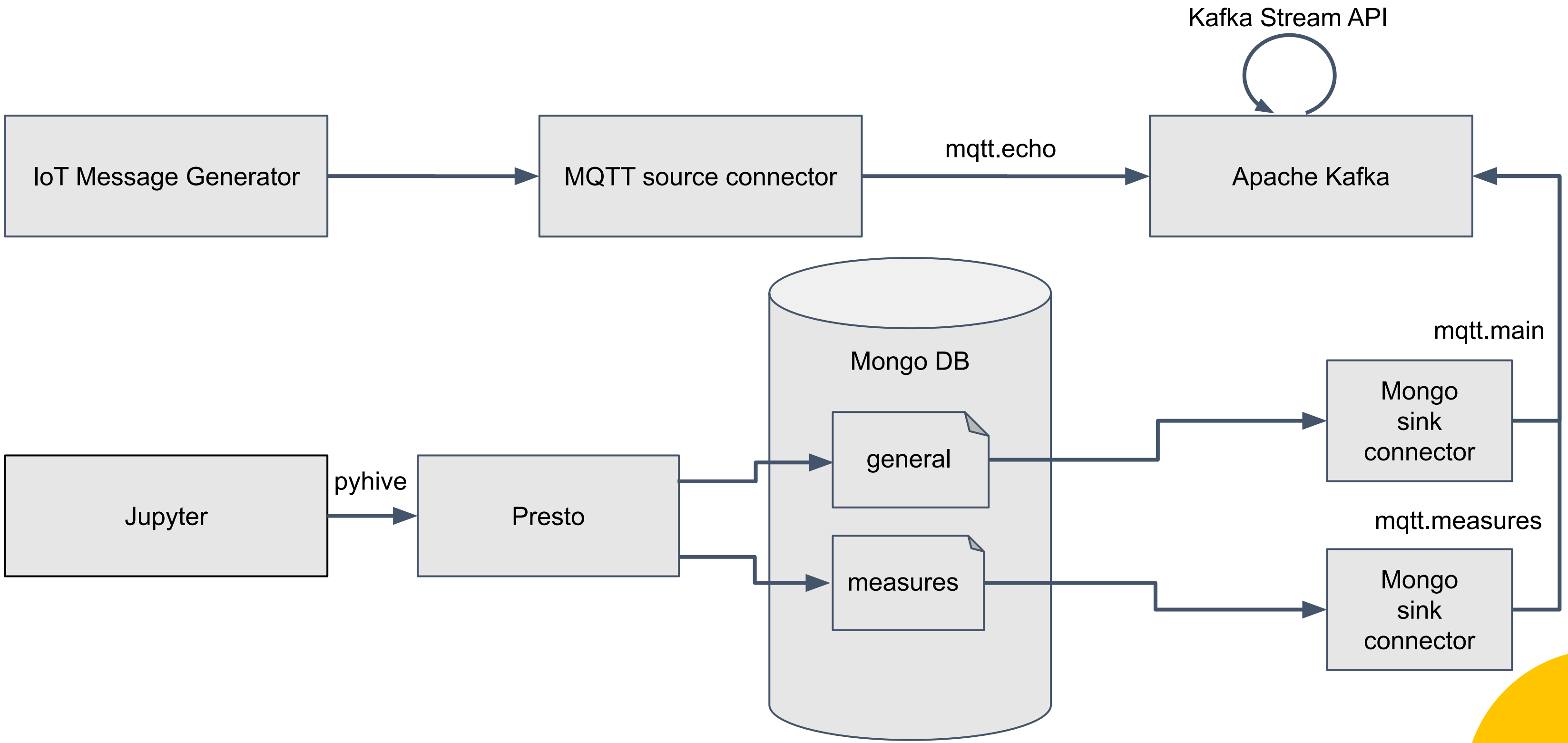
Second Approach

Use of Kafka Stream API

First Approach



Second Approach



Approaches Comparison

First Approach

- Slow

+ Easy to implement

- Need of performing large amount of queries

Second Approach

+ Able to handle data in real-time

+ Automatica Data flattening

- Lack of community support

5. Achieved Results

Performing queries

```
presto_cur.execute("select count(*),general.type FROM general INNER JOIN measures on general.uuid=measures.uuid GROUP BY general.type")
```

```
[(1065, 'environmental'),
 (350, 'presence'),
 (994, 'luxmeter'),
 (345, 'gasmeter')]
```

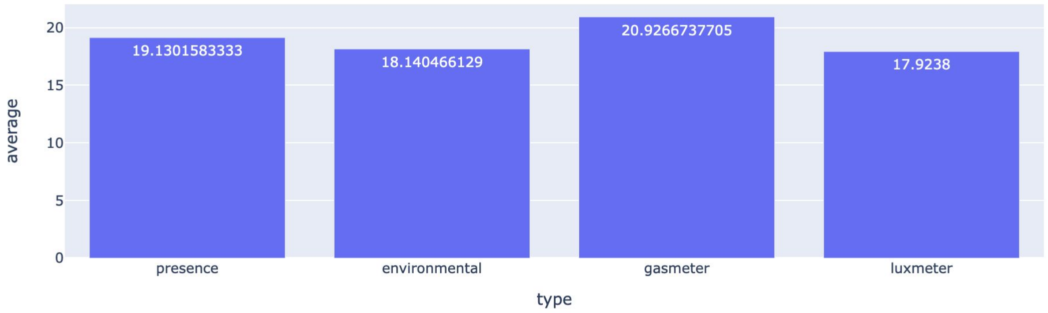
```
presto_cur.execute("select avg(measures.v),general.type FROM general INNER JOIN measures on general.uuid=measures.uuid WHERE measures.k='device_temperature' GROUP BY general.type")
```

```
[(19.60545, 'presence'),
 (21.523076811594215, 'gasmeter'),
 (17.631318309859164, 'environmental'),
 (17.960385915492957, 'luxmeter')]
```

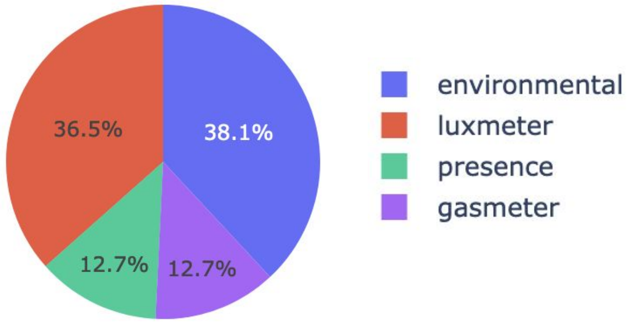
	cat	cuid	d	k	ref	sn	t	type
0	0620	d0e99fc3-d47a-45ec-90e1-27942e5a4bf3	None	adc_channel_01	jzp://edv#0501.0000	141	1685369982980	luxmeter
1	0620	d0e99fc3-d47a-45ec-90e1-27942e5a4bf3	None	pressure	jzp://edv#0501.0000	141	1685369982980	luxmeter
2	0620	d0e99fc3-d47a-45ec-90e1-27942e5a4bf3	jzp://coo#ffffffff00000500.0000	coordinator	jzp://edv#0501.0000	141	1685369982980	luxmeter

Building graphs

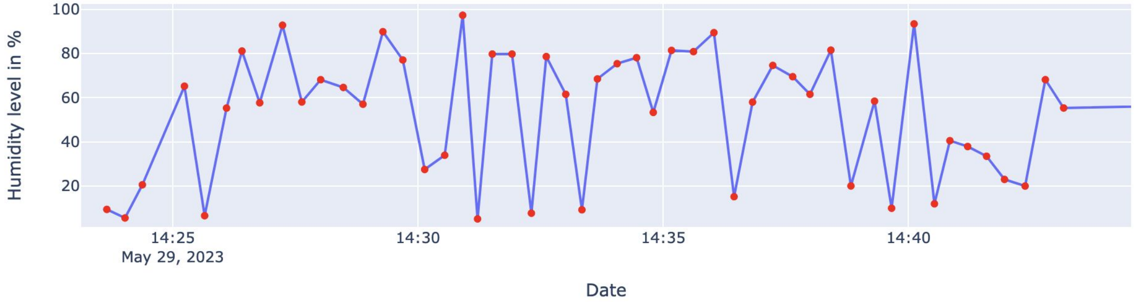
Average of the measured temperature by the Devices



Distribution of how many Measurements each type of device has done



Humidity in environmental devices



6. Conclusion

Future improvements

Implementing
Machine
learning algorithms

Building an actual
dashboard

Use of more
advanced data
analytics libraries

Thanks !