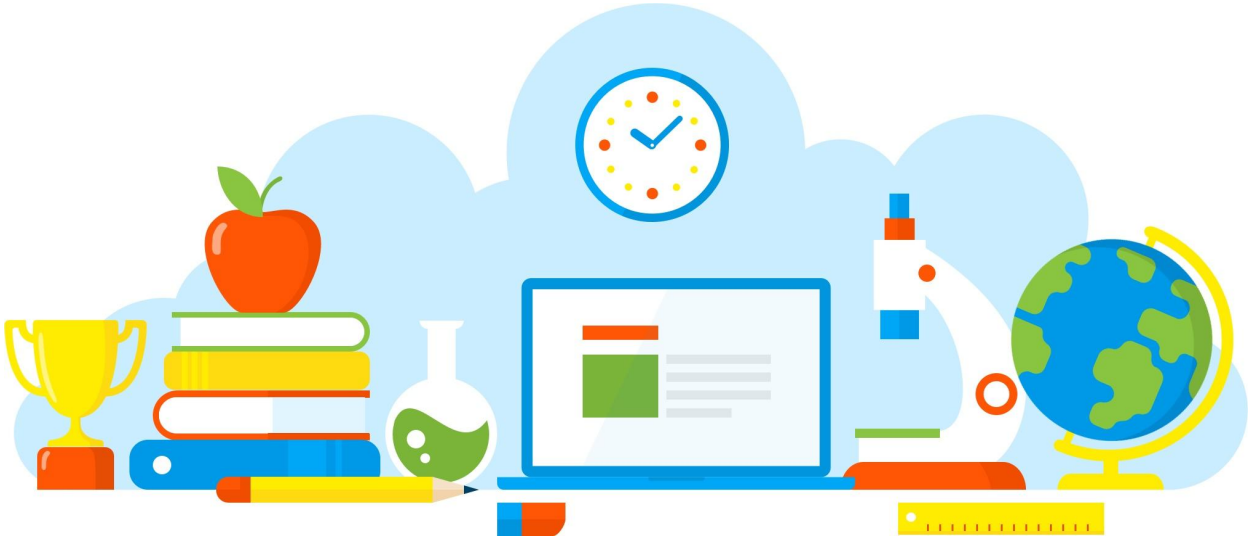


EBT User Manual

Version: 1.0

Mahsa Hadian & Fares Hamouda
Prof. Marios Fokaefs

07/2022

Table of contents

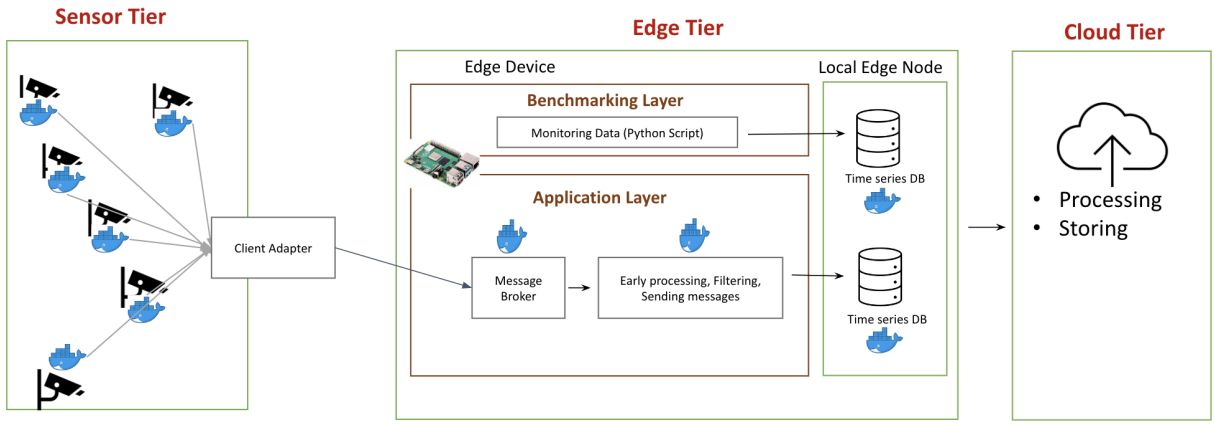
Overview

The EdgeBenchmarkTool (EBT) is an end-to-end benchmark tool for IoT applications deployed on edge networks to evaluate the overall performance and energy consumption of edge devices.

EBT is designed to be flexible and extensible to support different application architectures. Also, EBT can be used for capacity, load and stress testing, among others, particularly focusing on low capacity edge devices and edge servers.

I.Architecture

EBT assumes the architecture below for the intended application. It is assumed that there exists three tiers: a sensor tier, an edge tier and a cloud tier. Next, we describe each one of them in detail



1. Framework Tiers:
    a. Sensor Tier:

        This is a simulated tier where Docker *containers* are used to simulate sensors that send data to the edge application through a *client adapter* .

        At the current version, the containers simulate cameras that send frames of a video of varying sizes between 141KB and 148KB every 1/5/10 seconds. The client adapter provides a simple API to receive data from the containers-sensors as post requests. Subsequently, the client adapter forwards the data to a message broker, like Mosquitto or Kafka.

        It is possible to replace the Docker image to simulate different and multiple sensors at the same time. The adapter can also be modified to connect to different message-oriented middleware.

b. Edge Tier

The edge tier corresponds to the application, which is usually responsible for receiving, analyzing and up to a certain degree storing local data. The purpose of the edge application is to process data as close as possible to the point of production to reduce latency and potentially increase accuracy.

At the current implementation, the Edge Tier of EBT consists of a physical edge device and a VM acting as the local edge node.

- **Edge device:**

  The edge devices can be lightweight, like Raspberry PI(RPI) and the sensors can pair with them via WiFi, Bluetooth, or through the local area network.

  Even when simulated, the edge device should be placed near the sensors.

  1. **components :**
     - **Application Layer :**

       represents the user's application composed of the message broker (mosquitto, kafka ..) responsible for collecting data from the sensors and the processing and filtering component (spark streaming , kafka streams , a simple script ..).

       NB: All these components need to be running on docker containers and no other container should be running on the edge device

     - **Benchmarking Layer :**

       This is the layer responsible for collecting the resource consumption metrics of the Application Layer.

- **Local Edge Node:**

  The local edge node can be a VM on the edge side but shouldn't be as near to the sensors as the edge device.

  1. **components :**

     Measurements Database :

     a Time Series Database , InfluxDB, in which the benchmarking Layer saves the resource consumption metrics.

     Application Database :

     any type of database, this is related to the Application Layer in which data is saved after the filtering and processing.

c. Cloud Tier

The cloud tier handles data storage and larger scale computations.

In the current implementation of EBT, a VM located in the Digital Research Alliance of Canada (formerly ComputeCanada) research cloud[1] is used.

II. Used technologies & Repository File System Description

/DB Side:

this the code to be deployed on the DB Side.

- /yourexample: in this folder you will find a dockercompose.yml file in which there is a Time series Database used for saving benchmarking Data. You need to add another database according to your application Layer (check the steps here).
- /mongoDB: in this example the application layer requires a mongoDB Database
- /influxDB: in this example the application layer requires an InfluxDB Database

/Edge Side:

[1] https://alliancecan.ca/en

this is the code to be deployed on the edge device

- /Processing: code responsible for processing and filtering the data sent from sensors.
    - /simple script : an example of processing using a python script and saving in a mongoDB database.
    - /spark: an example of processing using spark and saving in an InfluxDB database.
    - /YourExample : you can here put your own code for processing.
- /monitoring: code responsible for monitoring the edge using docker stats.

/Sensor Side:

this is the code to be deployed on the sensor side

- Client Adapter:

  code responsible for collecting data from the Datagenerator and adapting sending it according to the message broker.

    - ☐ / MQTT Adapter: in this folder the adapter subscribes to an MQTT broker and sends data using publish/subscribe protocol.
    - ☐ /YourAdapter: the user can create his/her own adapter , the code in the folder will facilitate the task.
- Data Generator:

  code responsible for generating data simulating cameras with docker containers.

  NB: All these folders are started up using docker compose up

III. Configuration

1. Sensor Side
    a. Data Generator

       you need to edit the conf file to route the data from the data generator to the client Adapter.

       Sensor Side/DataGenerator/datagenerator-variables.env

CLIENT_ADAPTER_IP	the IP of the Adapter you are using to send data to the edge device , the IP should be the same as the Sensor Side Machine.
CLIENT_ADAPTER_PORT	the PORT on what the Adapter you are using to send data is listening.

b. Client Adapter

if you are using the MQTT Adapter

you need to edit the conf file to route the data from the Client Adapter to the Edge side.

`Sensor Side/ClientAdapter/MQTTAdapter/sensor-variables.env`

MQTT_SERVER_IP	the message broker IP on the edge device , this should be the same IP of the edge device (e.g. tcp://1.1.1.1)
MQTT_SERVER_PORT	the message broker PORT on the edge device
MQTT_SERVER_TOPIC	the topic that the MQTT Adapter is publishing data to .

if you are creating another adapter

you can user the folder

`Sensor Side/ClientAdapter/YourAdapter` to define your own adapter

you can use the file

`Sensor Side/ClientAdapter/YourAdapter/sensor-variables.env`

to define all your env variables and then use them inside your code using

```
os.getenv('the name of the variable')
```

2. Edge Side

if you are using simple script as a processor

you need to use the folder Edge Side/Processing/simple script/

and edit the conf file to route data from the python script to the database

Edge Side/Processing/simple script/simple-script-edge-variables.env

MONGODB_DATABASE_NAME	the name of the MongoDB database
MONGODB_COLLECTION_NAME	the name of the MongoDB collection
MONGODB_DATABASE_IP	the IP of the MongoDB database (this should be the Local edge node IP)
MONGODB_DATABASE_PORT	the PORT of the MongoDB database
MONGODB_DATABASE_USERNAME	the username used to authenticate to the MongoDB database
MONGODB_DATABASE_PASSWORD	the password used to authenticate to the MongoDB database
MQTT_SERVER_IP	the message broker IP on the edge device , this should be the same IP of the edge device(e.g. 1.1.1.1)
MQTT_SERVER_PORT	the message broker PORT on the edge device
MQTT_TOPIC	the topic that script is subscribing to

if you are using spark as a processor

you need to use the folder Edge Side/Processing/spark/

and edit the conf file to route data from the spark to the database

Edge Side/Processing/simple script/spark-edge-variables.env

INFLUXDB_PROTOCOL	the protocol used to connect to the influxDB database (e.g. http/https)
INFLUXDB_IP	the IP of the InfluxDB database (this should be the Local edge node IP)
INFLUXDB_PORT	the PORT of the InfluxDB database
INFLUXDB_TOKEN	the TOKEN used to authenticate to the InfluxDB database
INFLUXDB_ORG	the name of InfluxDB organization
INFLUXDB_BUCKET	the name of InfluxDB Bucket
INFLUXDB_WRITE_TIMEOUT	the timeout of writing in the InfluxDB with seconds(e.g. 60)
MQTT_SERVER_IP	the message broker IP on the edge device , this should be the same IP of the edge device(e.g. tcp://1.1.1.1)
MQTT_SERVER_PORT	the message broker PORT on the edge device
MQTT_TOPIC	the topic that script is subscribing to
MASTER_IP	the IP of the spark master (this should be the edge device)
MASTER_PORT	the port on which the spark master is listening
SPARK_EXECUTOR_MEMORY	Amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).
SPARK_DRIVER_MEMORY	Amount of memory to use for the driver process, i.e. where SparkContext is initialized, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).

SPARK_MEMORY_FRACTION	Fraction of (heap space - 300MB) used for execution and storage. The lower this is, the more frequently spills and cached data eviction occur. The purpose of this config is to set aside memory for internal metadata, user data structures, and imprecise size estimation in the case of sparse, unusually large records.
SPARK_MEMORY_STORAGEFRACTION	Amount of storage memory immune to eviction, expressed as a fraction of the size of the region set aside by spark.memory.fraction. The higher this is, the less working memory may be available to execution and tasks may spill to disk more often.

NB : in our case for a 4gb (RAM) Raspberry Pi these are the values we've chosen for spark memory when running spark cluster with 1 worker

SPARK_EXECUTOR_MEMORY="1g"

SPARK_DRIVER_MEMORY="2g"

SPARK_MEMORY_FRACTION="0.9"

SPARK_MEMORY_STORAGEFRACTION="0.2"

Mosquitto

you need to edit the mosquitto.conf file

listener	1883
allow_anonymous	true
max_queued_messages	0
max_connections	-1
log_type	all
max_inflight_messages	0

IV. How to use EBT

1. Minimum Requirement

To run these experiments we need **2 virtual machines**, **1 edge device**, a powerMeter, and a computer to measure the energy of the edge device.

A. Sensor Side & Db Side

2 VMS

OS	Ubuntu 18.04
CPU	2 vCPUs
Memory	8GB RAM

B. Edge Side

a. Edge device

OS	incomplete
CPU	incomplete
Memory	4GB RAM

b. Power Meter

to retrieve the energy metrics

c. computer

to install the software compatible with the power meter and extract the energy data.

OS	at least WIN 7

2. Prerequisite

Step 1, 2& 3 need to be executed on the 2 VMS (Sensor Side & DB side) and the edge device

a. Step 1 : Install docker

a.1 Follow officiel documentation, install docker on ubuntu, or follow steps on 1.2 subsection.

a.2 Manual installation

```
$ sudo apt update

$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -

$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu bionic stable"

$ sudo apt update

$ sudo apt install docker-ce
```

1.3 Run docker without sudo

```
$ sudo usermod -aG docker ${USER}

$ su - ${USER}
```

b. Step 2: Install docker compose

b.1 Follow officiel documentation, install docker compose on ubuntu, or follow steps on 1.2 subsection.

b.2 Manual installation

```
$ sudo curl -L

https://github.com/docker/compose/releases/download/1.21.2/do

cker-compose-`uname -s`-`uname -m` -o

/usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose
```

c. Step 3: Clone the repo

```
git clone https://github.com/polytechnique-ease/EdgeBenchmark
```

d. Step 4 : access the right folders

d.1 on Sensor Side

```
cd Sensor\ Side/
```

d.2 on Edge Side

```
cd Edge\ Side/
```

d.3 on DB Side

```
cd DB\ Side/
```

e. Step 5 : install the power meter software on the computer

you can follow the manual and install the software from this link

https://drive.google.com/drive/u/0/folders/1mMV2qPtrhsiYuaeDA7boKEAcR4J_iGno

3. Get Started

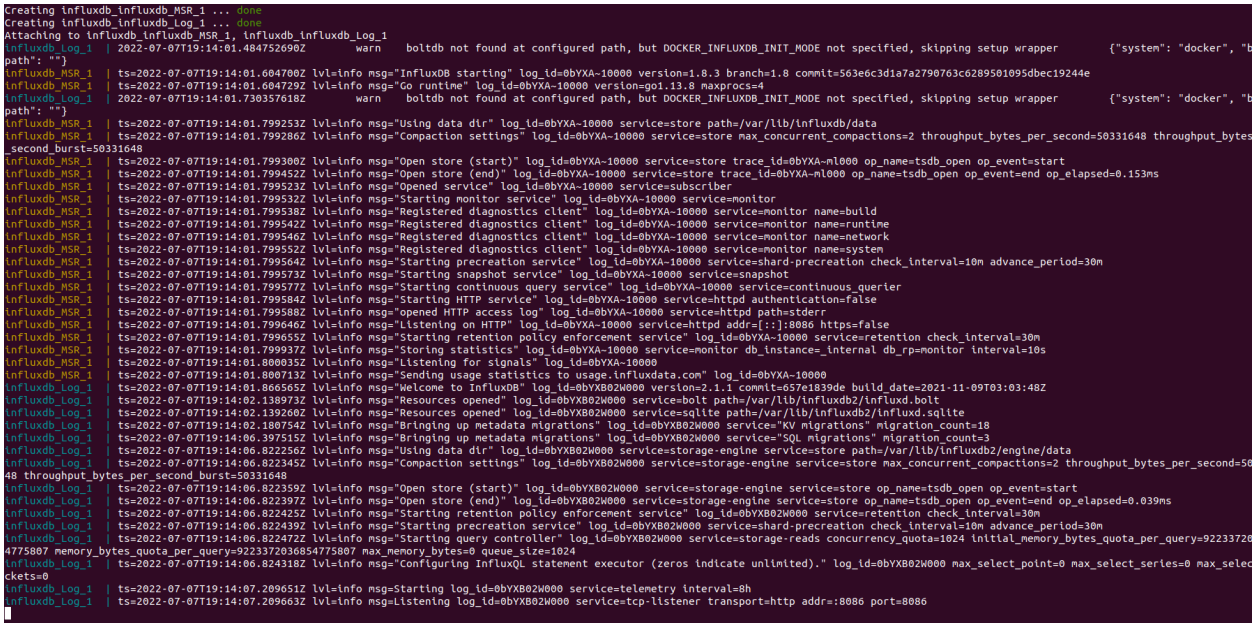
3.1 Local Edge node / DB side Machine

you are now inside the DB Side folder in the Sensors VM

A. OPTION 1 : SPARK (processing) + InfluxDB(SAVING DATA)

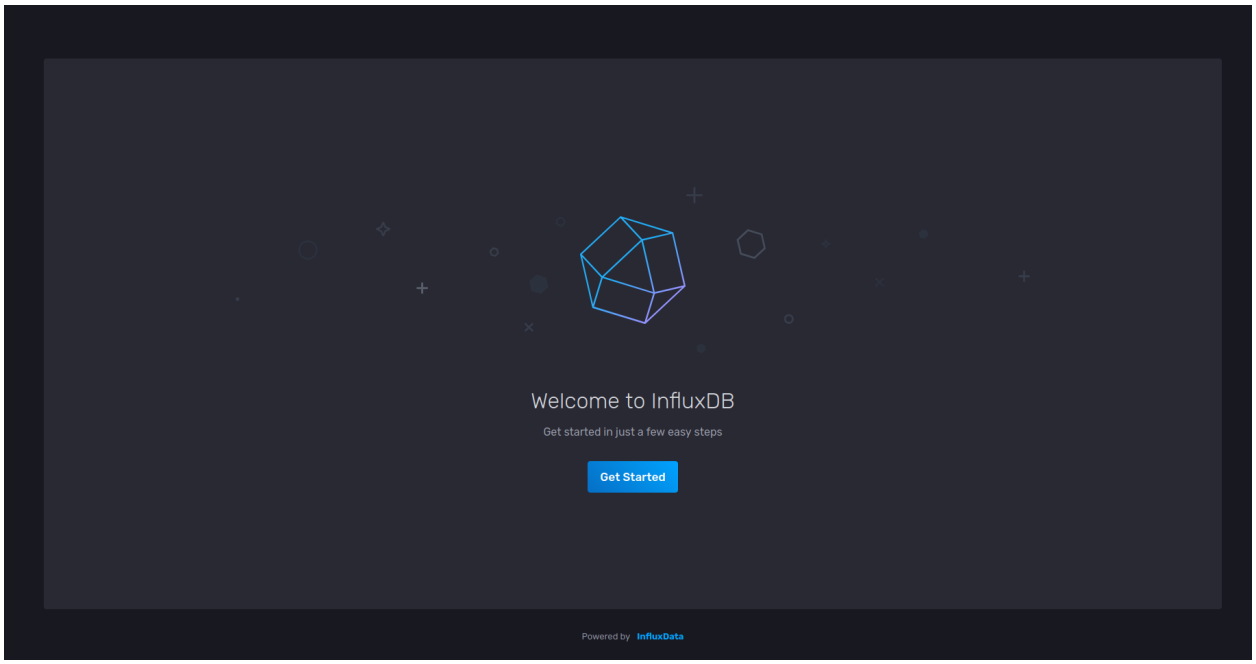
1.
```
cd influxDB/
```
2.
```
docker-compose up
```

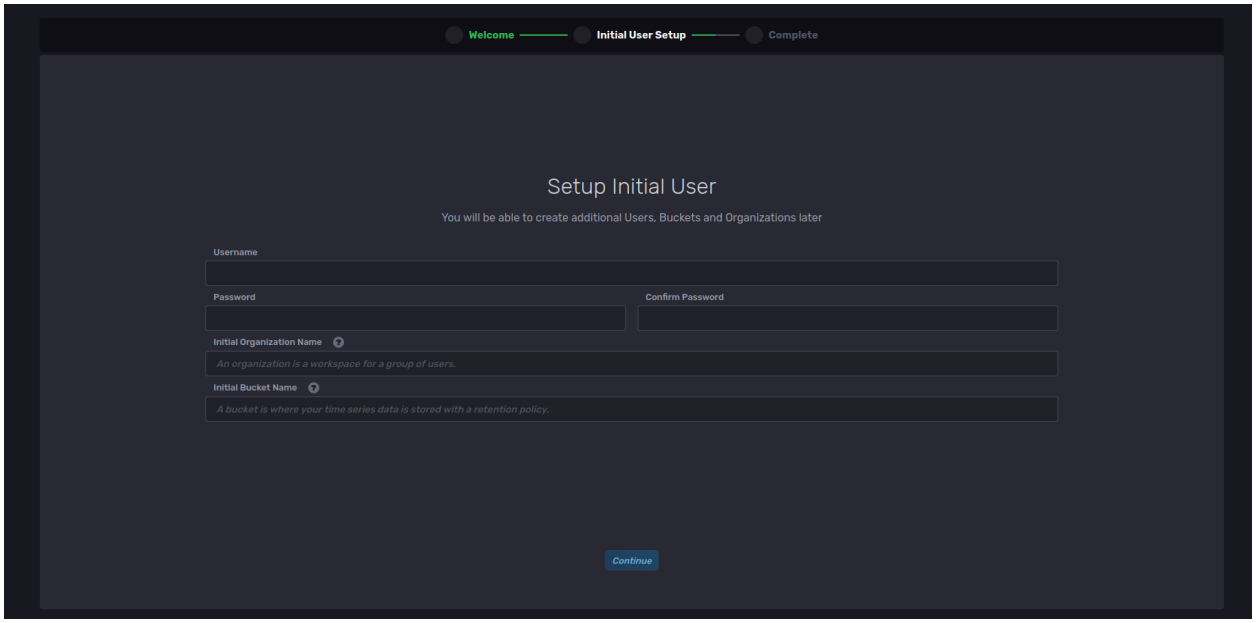
result :



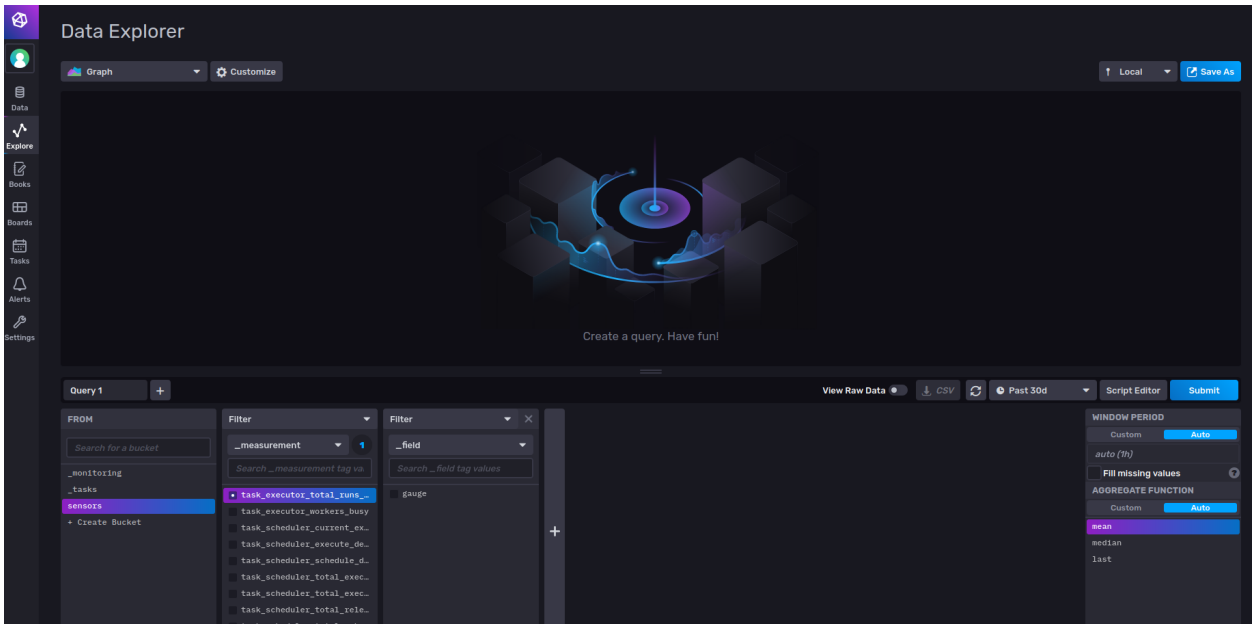
3. open the browser and open localhost:8086

result :





4. fill the form above and save the **Organization name** and the **bucket name** we'll need them later
5. choose explore from the left panel



6. that's it the database is ready

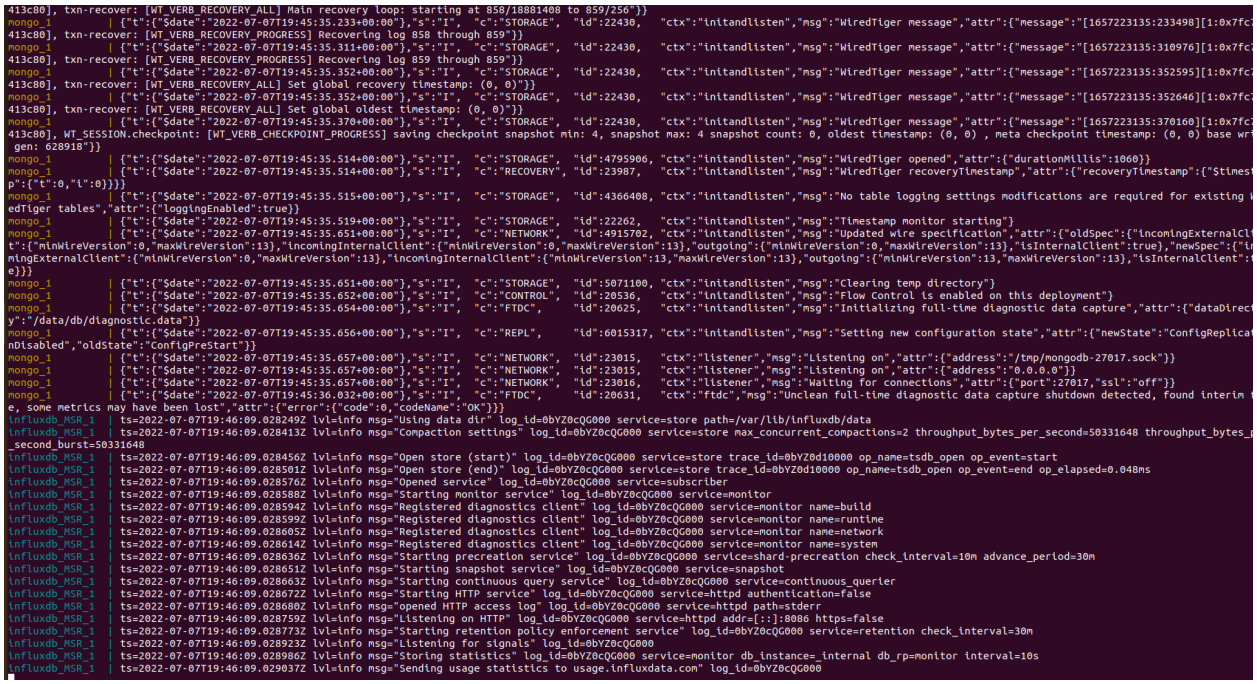
B. OPTION 2 : PYTHON SCRIPT (FILTERING) + MONGODB (SAVING DATA)

1. `cd mongodb`
2. you can change these values in the docker compose file

```
MONGO_INITDB_ROOT_USERNAME: root
MONGO_INITDB_ROOT_PASSWORD: example
```

3. `docker-compose up`

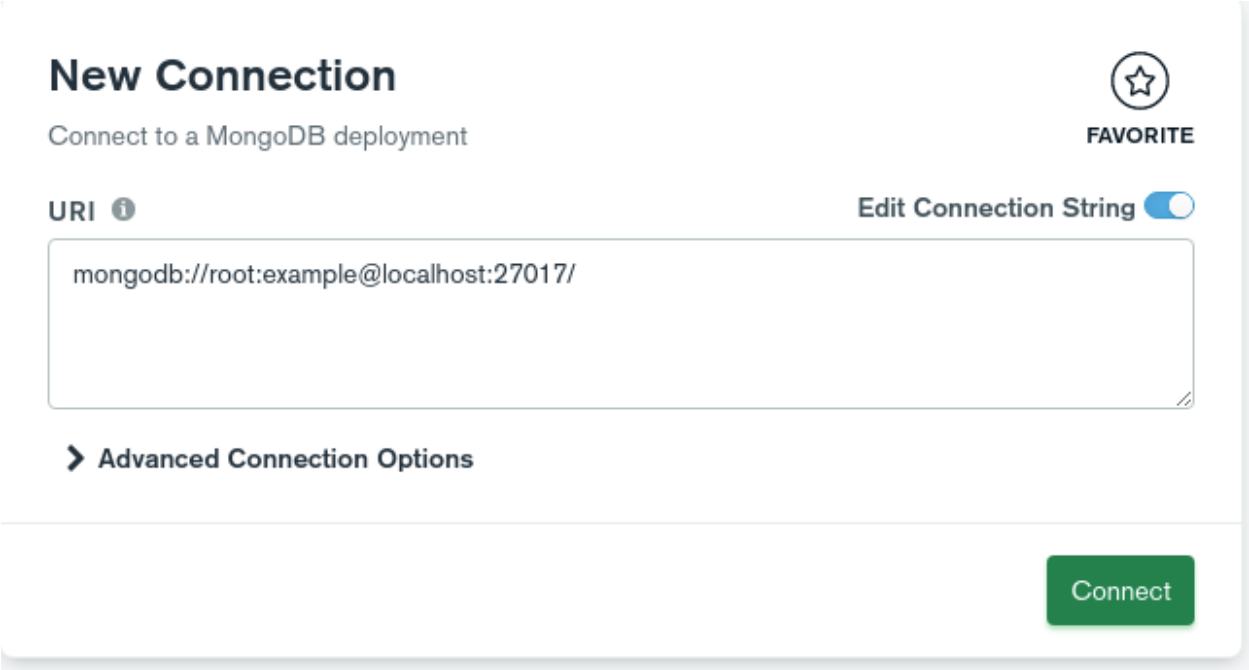
result :



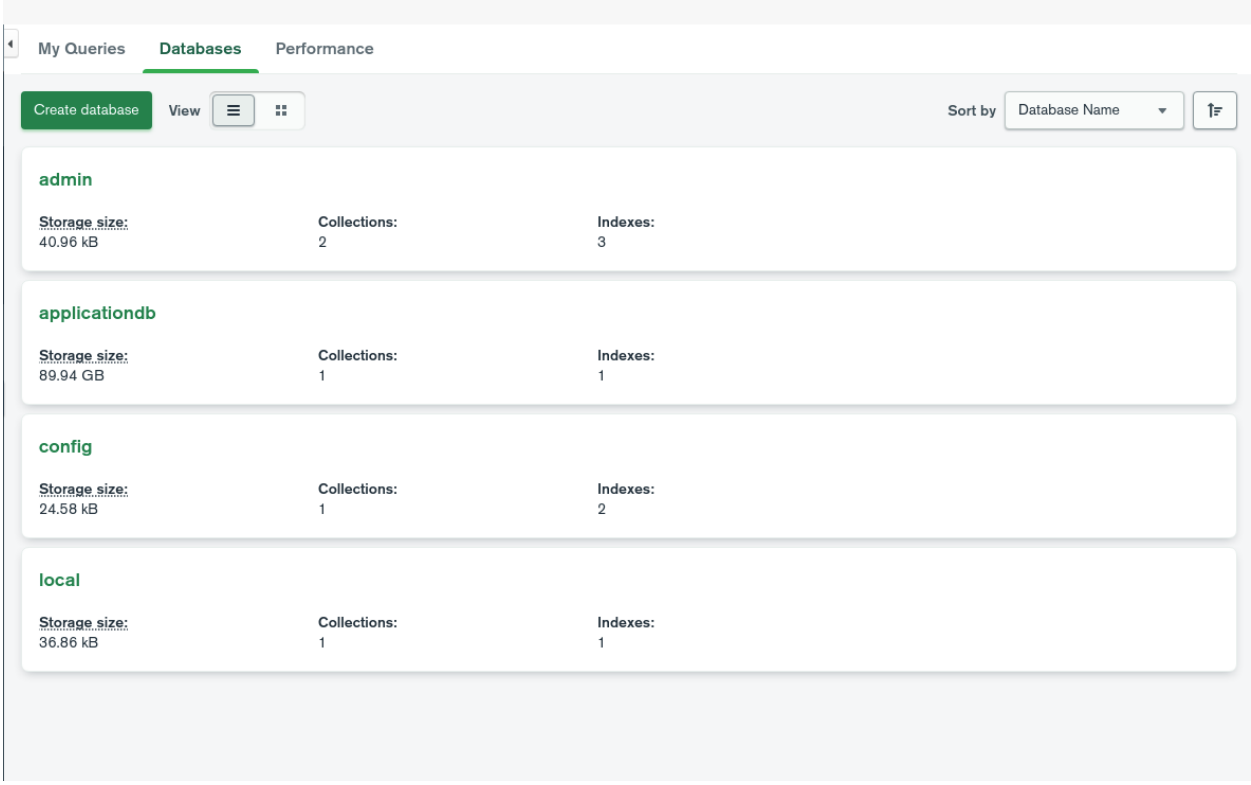
4. follow this link to install MongoDB compass to manage the DB

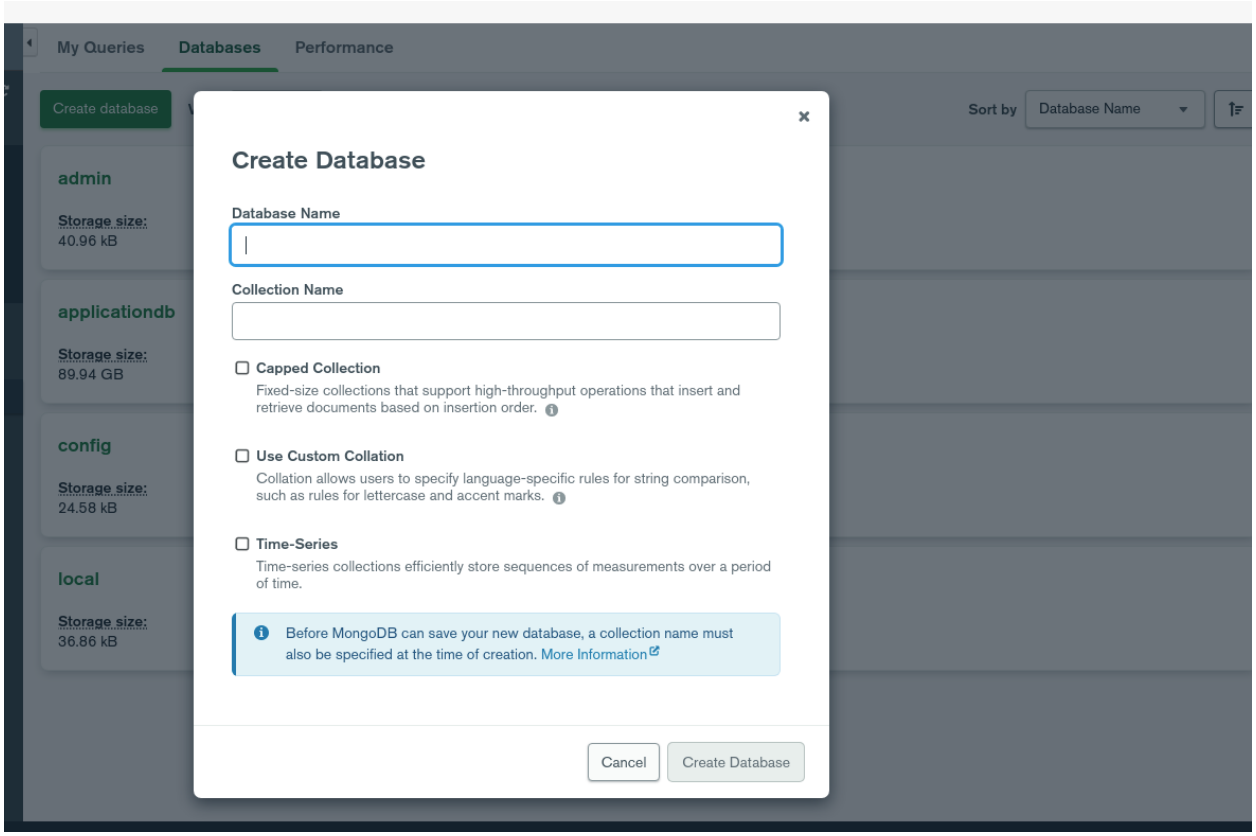
   https://www.mongodb.com/docs/compass/current/install/

5. create a new connection



6. create a new database





save the **database name** and **collection name** we'll need them later

7. that's it the database is ready

C. OPTION 3 : Your own application

1.
```
cd yourexample/
```

2. this service is required in docker compose it's for the benchmarking data

```
influxdb_MSR:
  image: influxdb:1.8.3
  restart: always
  ports:
    - "8087:8086"
  volumes:
    - ./influxdb_MSR:/var/lib/influxdb
```

3. add your database to the docker compose file as a service.
4. `docker-compose up`

3.2 Edge device

you are now inside the Edge Side folder in the Edge Device

A. OPTION 1 : SPARK (processing) + InfluxDB(SAVING DATA)

1. `cd Edge\ Side/Processing/spark`
2. `cd spark-master/`
3. `docker build -t yourrepo/spark-master .`

   in our case the repo name is mahsahadian so that's the name we'll be using in docker compose

4. `cd ../spark-worker/`
5. `docker build -t yourrepo/spark-worker .`
6. `cd ../spark-job/maven-spark-processing/`
7. **This is optional :** you can edit the maven project but after editing you need to install maven on the Edge device and execute this command.

```
mvn install
```

result :

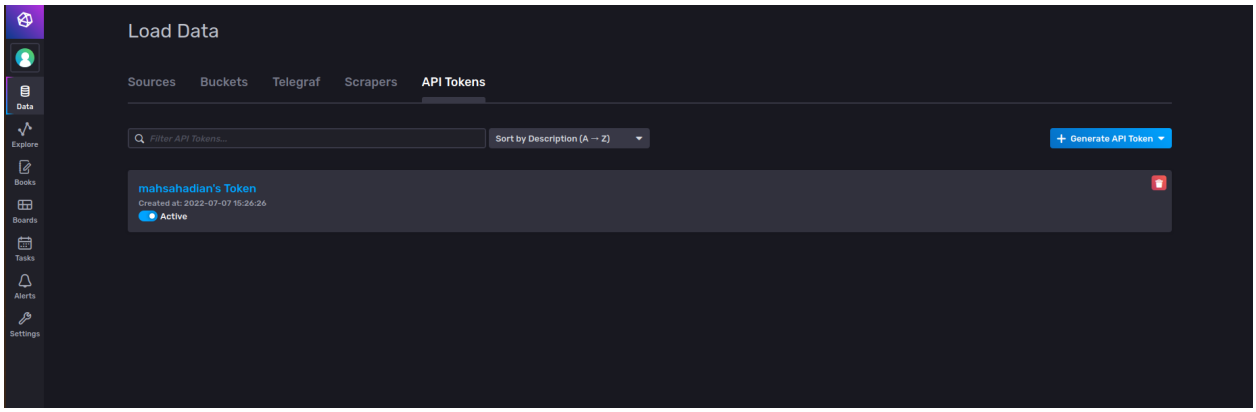
```
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ maven-spark-processing ---
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ maven-spark-processi
[INFO] Installing /home/pi/Documents/NewRepo/EdgeBenchmark/Edge Side/spark/spark-job/
spark-processing/1.0-SNAPSHOT/maven-spark-processing-1.0-SNAPSHOT.jar
[INFO] Installing /home/pi/Documents/NewRepo/EdgeBenchmark/Edge Side/spark/spark-job/
rk-processing-1.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.763 s
[INFO] Finished at: 2022-07-07T17:28:33-04:00
[INFO] ------------------------------------------------------------------------
```

8. `cd ..`
9. `docker build -t yourrepo/spark-job .`
10. **Configuration :** now we need to edit the spark-edge-variables.env

A.

```
#The full HTTP or UDP URL for your InfluxDB instance.
INFLUXDB_PROTOCOL='http'
INFLUXDB_IP='****'
INFLUXDB_PORT=8086
INFLUXDB_TOKEN="***"
INFLUXDB_ORG="polymtl"
INFLUXDB_BUCKET="sensors1"
INFLUXDB_WRITE_TIMEOUT="60"
```

you need to change the InfluxDB_IP with the Edge device's IP and the port as configured in DB side docker compose. All the other variables need to be changed according to your Database Configuration(organization name, Bucket name…).



the token needs to be taken from the InfluxDB UI in the photo above.

B.

```
#Broker URL for the MQTT server
MQTT_SERVER_IP=tcp://**
MQTT_SERVER_PORT=1883
MQTT_TOPIC="sensors"
```

the MQTT SERVER IP is the same as the edge device's IP (don't forget to add tcp://)

the MQTT SERVER PORT is configured in the mosquitto.conf file

we'll be choosing "sensors" as a topic (if you change it you need to change it on the Sensor side also)

C.

```
#Spark
MASTER_IP=**
MASTER_PORT=7077
SPARK_EXECUTOR_MEMORY="500m"
SPARK_DRIVER_MEMORY="2g"
SPARK_MEMORY_FRACTION="0.9"
SPARK_MEMORY_STORAGEFRACTION="0.2"
```

the master IP is the same as the edge device's IP without tcp://

for 4 gb as ram to launch spark with 1 worker this is the spark conf we advise you to choose

D.

```
# This is a Mosquitto configuration file that create
# that allows unauthenticated access.

listener 1883
allow_anonymous true

max_queued_messages 0
#persistence true
#persistence_location /
max_connections -1
log_type all
max_inflight_messages 0
#log_dest file /hi
```

this is the configuration you need to have on mosquitto.conf

listener is the same MQTT SERVER PORT in spark-edge-variables.env

11. `$ cd ..`
12. `docker-compose up`

if you want to run spark with more than 1 worker you need to run this command

`docker-compose up --scale sparkworker=2` we chose 2 in this case

B. OPTION 2 : PYTHON SCRIPT (FILTERING) + MONGODB (SAVING DATA)

1. `cd Edge\ Side/Processing/simple\ script/`
2.

```
# This is a Mosquitto configuration file that create
# that allows unauthenticated access.

listener 1883
allow_anonymous true

max_queued_messages 0
#persistence true
#persistence_location /
max_connections -1
log_type all
max_inflight_messages 0
#log_dest file /hi
```

this is the configuration you need to have on mosquitto.conf

listener is the same MQTT SERVER PORT in simple-script-edge-variables.env

3.
```
MONGODB_DATABASE_NAME="applicationdb"
MONGODB_COLLECTION_NAME="sensors"
MONGODB_DATABASE_IP='132.207.170.25'
MONGODB_DATABASE_PORT=27017

MONGODB_DATABASE_USERNAME="root"
MONGODB_DATABASE_PASSWORD="example"
```

all these values can be taken from the Database Side

the username and the password are the same from the docker compose on the DB side

4. `docker build -t yourrepo/pythonprocessor .`

yourrepo/pythonprocessor should be the same name in the docker compose file

5. `docker-compose up`

C. OPTION 3 : Your own application

1. `cd Edge\ Side/Processing/YourExample`
2. your application needs to be running in docker containers so you only need to run your application using docker compose or only with docker run

   you can use mosquitto as a message broker which is already in the docker compose file or you can change it with another message broker

3.3 Sensor Side Machine

you are now inside the Sensor Side folder in the Local Edge Node(Db) VM.

We have 2 main components:

- Data generator: generating data as images and sending them as post requests to the client adapter.
- Client Adapter: accepting requests from the Data Generator, and sending the data to the edge side according to the message broker.

A. **Client Adapter:**

   a. **you are Using MQTTAdapter:**

      1. **configuration:** you need to edit the file

         /ClientAdapter/MQTTAdapter/sensor-variables.env

MQTT SERVER IP is the same as the edge device's IP (without tcp://)

MQTT SERVER PORT is configured in the mosquitto.conf file.

2. `cd MQTTAdapter/`
3. `docker-compose build`
4. `docker-compose up`

```
Starting mqttadapter_app_1 ... done
Attaching to mqttadapter_app_1
```

b. you are developing your Own Adapter:

1. ```
   cd YourAdapter/
   ```
2. open YourAdapter.py

   you have 2 sections to edit :

   - In the run function before initiating the handler, you can put any code you want to be run only once before receiving requests.
   - if you want to initiate variables and pass them to the post function, the variable client and b are exemples.

     pass them to the function partial after the variable YourAdapter(which is the name of the class)

```
def run(server_class=HTTPServer, handler_class=YourAdapter, port=8088):
    client = "xx"
    b ="bb"
    handler = partial(YourAdapter,client,b)
    server_address = ('', port)
    httpd = server_class(server_address, handler)
    print('Server running at localhost:8088...')
    httpd.serve_forever()
```

and get them in the class constructor

```
def __init__(self,client,b, request, client_address, server):
    self.request = request
    self.client_address = client_address
    self.server = server
    self.client = client
    self.b = b
    print(client)
```

NB : Order of the variables is very important
```

In the function below you put all the code that you want it to be run on each post request:

```
def do_POST(self):
    ctype, pdict = cgi.parse_header(self.headers.get('content-type'))

    # refuse to receive non-json content
    if ctype != 'application/json':
        self.send_response(400)
        self.end_headers()
        return

    # read the message and convert it into a python dictionary
    length = int(self.headers.get('content-length'))

    message = self.rfile.read(length)

    # send the message back
    self._set_headers()
```

message is the variable containing your data.

B. Data Generator:

1. you need to choose the measurement name in the videogenerator.py

```
jsondata['measurement_name'] = "t_1w_111_1_10s"
```

2. each time you change the measurement name you need to run this command

```
docker-compose build
```

3. Now you can run any number of emulated sensors you want.

```
COMPOSE_HTTP_TIMEOUT=2000 docker-compose up --scale app=54
```

Here we are launching 54 sensors at the same time.

But before launching the sensors we need to go back to the edge device and launch the MONITORING TOOL

First, We need to be inside the edge device now

1. **Resource Consumption**
   a.
```
cd Edge\ Side/monitoring/
```
   b. you need to change the measurement name just like in the sensor Side and it should preferably be the same as the name in the sensor side

```
                    'net_tx': float(self.get_network_throughput(container_stats)['tx'])}
json_body = [
                {
                "measurement": "a_85_1wor",
                "tags": {
                "container_name": name,
                "short_id": cont.short_id
                },
                "fields": {
               'short_id': cont.short_id,
               'cpu_percent': float(self.get_cpu_percent(container_stats)),
               #'cpu_power': float(self.client.powerapi.formula.find_one({"target":cont.name},
               'memory': float(self.get_memory(container_stats)['memory']),
               'memory_limit': float(self.get_memory(container_stats)['memory_limit']),
               'memory_percent': float(self.get_memory(container_stats)['memory_percent']),
               'memory_utilization': float(self.get_memory(container_stats)['memory_utilization
               'disk_i': float(self.get_disk_io(container_stats)['disk_i']),
               'disk_o': float(self.get_disk_io(container_stats)['disk_o']),
               'net_rx': float(self.get_network_throughput(container_stats)['rx']),
               'net_tx': float(self.get_network_throughput(container_stats)['tx'])
                }
              }
              ]
```

the measurement name in this case is a_85_1wor

   c. now you just need to launch this command

```
pi@raspberrypi:~/Documents/NewRepo/EdgeBenchmark/Edge Side/monitoring $ python monitoring.py
Monitoring V2
Using: Docker remote API
{'date': '1657320615.3', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec
 ______________
Monitoring V2
Using: Docker remote API
{'date': '1657320616.38', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec
 ______________
Monitoring V2
Using: Docker remote API
{'date': '1657320617.39', 'nb_containers': 0}
Size of data = 140 bytes
Time to insert into the database 0.01 sec
```

Now resources metrics are being saved in the Database

2. **Energy Consumption**

   now using the power meter we'll collect the energy data

   a. we'll plug the power meter to the edge device
   b. open the software on the computer near the edge device to view the energy consumption
   c. configure the values **(this needs to be completed)**
   d. launch capturing energy data

=> Now everything is ready resource consumption and energy data is being saved in the database , we can launch the sensors with this command on the sensor side

COMPOSE_HTTP_TIMEOUT=2000 docker-compose up —scale app=50

```
COMPOSE_HTTP_TIMEOUT=2000 docker-compose up --scale app=54
```

When seeing that all data is saved on the DB side and the Sensor Side finishes sending data, we can stop the resource consumption and energy scripts.

VI. Results