



# EBT User Manual

*Version: 1.0*

Mahsa Hadian & Fares Hamouda  
Prof. Marios Fokaefs

07/2022

---

## Table of contents

### Table of contents

#### Overview

- I. **Architecture**
  - A. **Framework Components**
- II. **Used technologies & Folders Description**
- III. **Configuration**
- IV. **How to use EBT**
- V. **Results**

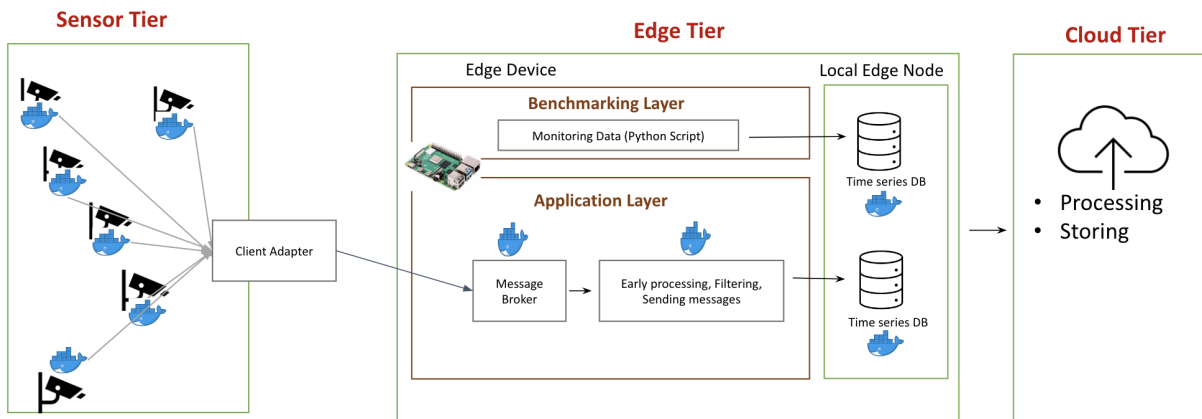
### Overview

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<sup>1</sup> 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:

---

<sup>1</sup> <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 use 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	<b>incomplete</b>
CPU	<b>incomplete</b>
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 official 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 official 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/1mMV2qPthrsiYuaeDA7boKEAcR4J\\_iGno](https://drive.google.com/drive/u/0/folders/1mMV2qPthrsiYuaeDA7boKEAcR4J_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)

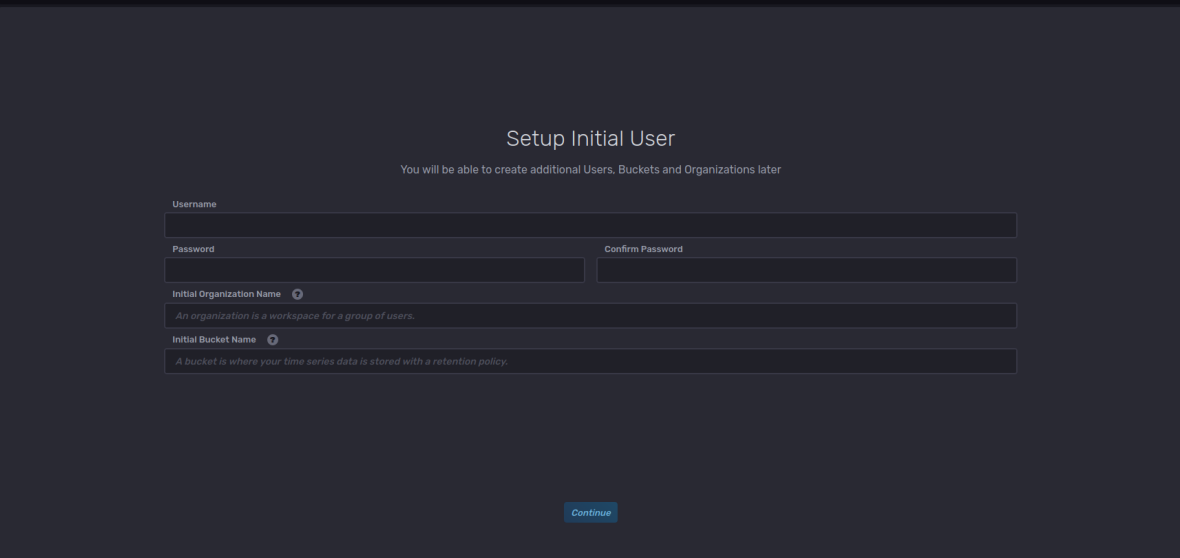
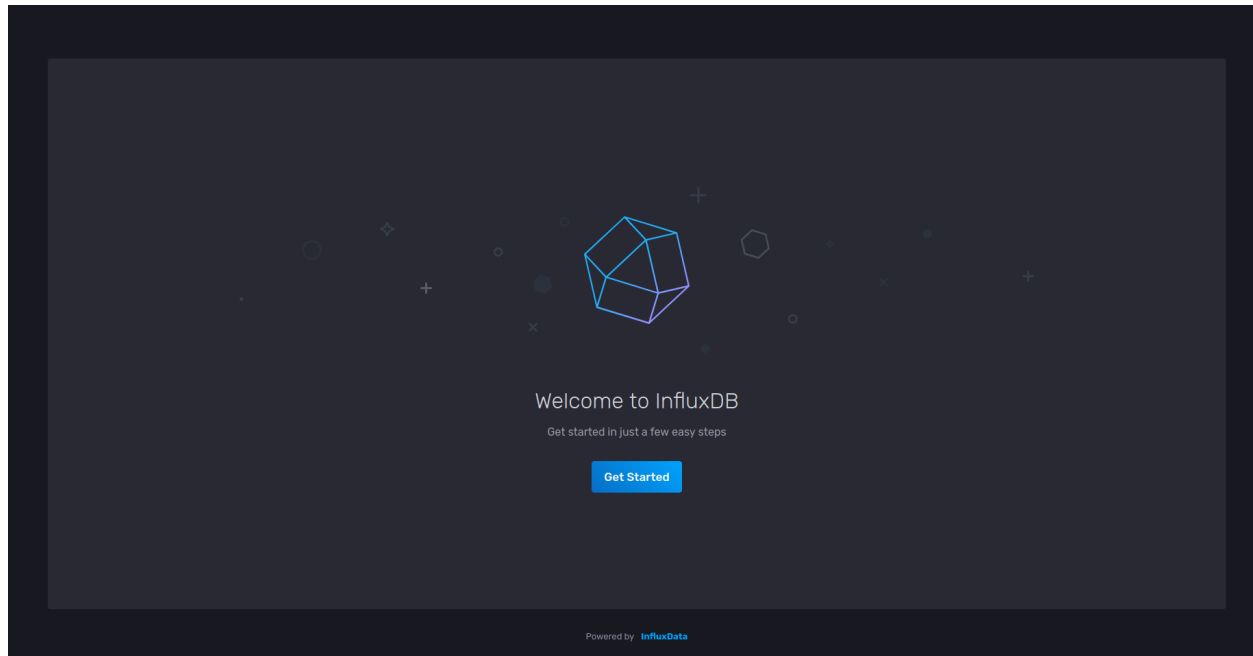
1. `cd influxDB/`
2. `docker-compose up`

result :

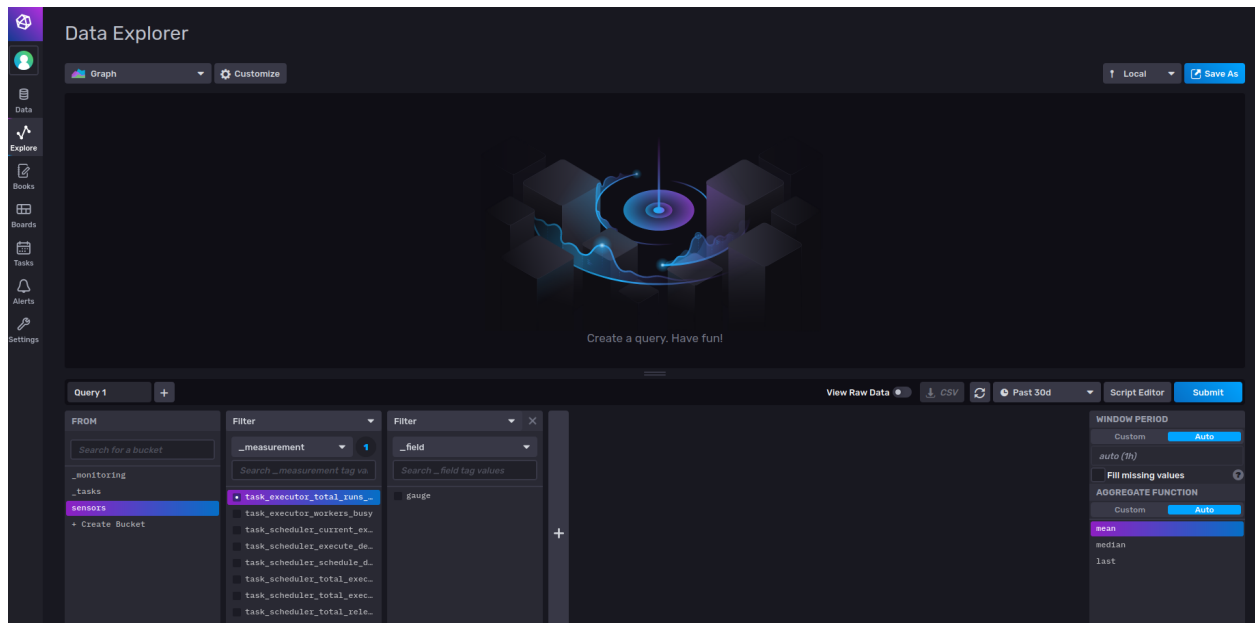
```
Creating influxdb_influxdb_MSR_1 ... done
Creating influxdb_influxdb_Log_1 ... done
Attaching to influxdb_influxdb_MSR_1, influxdb_influxdb_Log_1
influxdb_Log_1 | 2022-07-07T19:14:01.484752690Z warn boltdb not found at configured path, but DOCKER_INFLUXDB_INIT_MODE not specified, skipping setup wrapper ("system": "docker", "b
path": "")
influxdb_MSR_1 | ts=2022-07-07T19:14:01.604700Z lvl=info msg="InfluxDB starting" log_id=0bYXA-10000 version=1.8.3 branch=1.8 commit=5636c3d1a7a2790763c6289501095dbec19244e
influxdb_MSR_1 | ts=2022-07-07T19:14:01.604729Z lvl=info msg="Go runtime" log_id=0bYXA-10000 version=go1.13.8 maxprocs=4
influxdb_Log_1 | 2022-07-07T19:14:01.730357618Z warn boltdb not found at configured path, but DOCKER_INFLUXDB_INIT_MODE not specified, skipping setup wrapper ("system": "docker", "b
path": "")
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799253Z lvl=info msg="Using data dir" log_id=0bYXA-10000 service=store path=/var/lib/influxdb/data
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799286Z lvl=info msg="Compaction settings" log_id=0bYXA-10000 service=store max_concurrent_compactions=2 throughput_bytes_per_second=50331648 throughput_bytes
_per_second_burst=50331648
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799300Z lvl=info msg="Open store (start)" log_id=0bYXA-10000 service=store trace_id=0bYXA-m1000 op_name=tsdb_open op_event=start
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799452Z lvl=info msg="Open store (end)" log_id=0bYXA-10000 service=store trace_id=0bYXA-m1000 op_name=tsdb_open op_event=end op_elapsed=0.153ms
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799523Z lvl=info msg="Opened service" log_id=0bYXA-10000 service=subscriber
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799532Z lvl=info msg="Starting monitor service" log_id=0bYXA-10000 service=monitor
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799538Z lvl=info msg="Registered diagnostics client" log_id=0bYXA-10000 service=monitor name=build
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799542Z lvl=info msg="Registered diagnostics client" log_id=0bYXA-10000 service=monitor name=runtime
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799546Z lvl=info msg="Registered diagnostics client" log_id=0bYXA-10000 service=monitor name=network
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799552Z lvl=info msg="Registered diagnostics client" log_id=0bYXA-10000 service=monitor name=system
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799564Z lvl=info msg="Starting precreation service" log_id=0bYXA-10000 service=shard-precreation check_interval=10m advance_period=30m
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799573Z lvl=info msg="Starting snapshot service" log_id=0bYXA-10000 service=snapshot
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799577Z lvl=info msg="Starting continuous query service" log_id=0bYXA-10000 service=continuous_querier
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799584Z lvl=info msg="Starting HTTP service" log_id=0bYXA-10000 service=http authentication=false
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799588Z lvl=info msg="opened HTTP access log" log_id=0bYXA-10000 service=http path=stderr
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799646Z lvl=info msg="Listening on HTTP" log_id=0bYXA-10000 service=http addr=[::]:8086 https=false
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799655Z lvl=info msg="Starting retention policy enforcement service" log_id=0bYXA-10000 service=retention check_interval=30m
influxdb_MSR_1 | ts=2022-07-07T19:14:01.799937Z lvl=info msg="Storing statistics" log_id=0bYXA-10000 service=monitor db_instance=internal db_rp=monitor interval=10s
influxdb_MSR_1 | ts=2022-07-07T19:14:01.800035Z lvl=info msg="Listening for signals" log_id=0bYXA-10000
influxdb_MSR_1 | ts=2022-07-07T19:14:01.800713Z lvl=info msg="Sending usage statistics to usage.influxdata.com" log_id=0bYXA-10000
influxdb_Log_1 | ts=2022-07-07T19:14:01.860505Z lvl=info msg="Welcome to InfluxDB" log_id=0bYXB02W000 version=2.1.1 commit=657e1839de build_date=2021-11-09T03:03:48Z
influxdb_Log_1 | ts=2022-07-07T19:14:02.118917Z lvl=info msg="Resources opened" log_id=0bYXB02W000 service=sqlite path=/var/lib/influxdb2/influxd.sqlite
influxdb_Log_1 | ts=2022-07-07T19:14:02.139260Z lvl=info msg="Resources opened" log_id=0bYXB02W000 service=sqlite path=/var/lib/influxdb2/influxd.sqlite
influxdb_Log_1 | ts=2022-07-07T19:14:02.180754Z lvl=info msg="Bringing up metadata migrations" log_id=0bYXB02W000 service="KV migrations" migration_count=18
influxdb_Log_1 | ts=2022-07-07T19:14:06.397515Z lvl=info msg="Bringing up metadata migrations" log_id=0bYXB02W000 service="SQL migrations" migration_count=3
influxdb_Log_1 | ts=2022-07-07T19:14:06.822236Z lvl=info msg="Using data dir" log_id=0bYXB02W000 service=storage-engine service=store path=/var/lib/influxdb2/engine/data
influxdb_Log_1 | ts=2022-07-07T19:14:06.822345Z lvl=info msg="Compaction settings" log_id=0bYXB02W000 service=storage-engine service=store max_concurrent_compactions=2 throughput_bytes_per_second=50
48 throughput_bytes_per_second_burst=50331648
influxdb_Log_1 | ts=2022-07-07T19:14:06.822359Z lvl=info msg="Open store (start)" log_id=0bYXB02W000 service=storage-engine service=store op_name=tsdb_open op_event=start
influxdb_Log_1 | ts=2022-07-07T19:14:06.822397Z lvl=info msg="Open store (end)" log_id=0bYXB02W000 service=storage-engine service=store op_name=tsdb_open op_event=end op_elapsed=0.039ms
influxdb_Log_1 | ts=2022-07-07T19:14:06.822432Z lvl=info msg="Starting retention policy enforcement service" log_id=0bYXB02W000 service=retention check_interval=30m
influxdb_Log_1 | ts=2022-07-07T19:14:06.822439Z lvl=info msg="Starting precreation service" log_id=0bYXB02W000 service=shard-precreation check_interval=10m advance_period=30m
influxdb_Log_1 | ts=2022-07-07T19:14:06.822472Z lvl=info msg="Starting query controller" log_id=0bYXB02W000 service=storage-engine reads concurrency_quota=1024 initial_memory_bytes_quota_per_query=92233720
4775807 memory_bytes_quota_per_query=9223372036854775807 max_memory_bytes=0 queue_size=1024
influxdb_Log_1 | ts=2022-07-07T19:14:06.824318Z lvl=info msg="Configuring InfluxQL statement executor (zeros indicate unlimited)." log_id=0bYXB02W000 max_select_point=0 max_select_series=0 max_select
s=0
influxdb_Log_1 | ts=2022-07-07T19:14:07.209651Z lvl=info msg="Starting log_id=0bYXB02W000 service=telemetry interval=8h
influxdb_Log_1 | ts=2022-07-07T19:14:07.209663Z lvl=info msg="Listening log_id=0bYXB02W000 service=telemetry listener transport=http addr=:8086 port=8086
```

### 3. open the browser and open localhost:8086

result :

The image shows the "Setup Initial User" screen in InfluxDB. At the top, there is a progress bar with three steps: "Welcome" (completed), "Initial User Setup" (current), and "Complete". The main heading is "Setup Initial User", followed by the subtext "You will be able to create additional Users, Buckets and Organizations later". The form contains several input fields: "Username", "Password", "Confirm Password", "Initial Organization Name" (with a help icon and text "An organization is a workspace for a group of users."), and "Initial Bucket Name" (with a help icon and text "A bucket is where your time series data is stored with a retention policy."). A blue "Continue" button is at the bottom.

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 :

```

413c80, txn-recover: [WT_VERB_RECOVERY_ALL] Main recovery loop: starting at 859/18881408 to 859/250)})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.233+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WireTiger message","attr":{"message":"[1657223135:233498][1:0x7fc
413c80, txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 858 through 859})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.311+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WireTiger message","attr":{"message":"[1657223135:310976][1:0x7fc
413c80, txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 859 through 859})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.352+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WireTiger message","attr":{"message":"[1657223135:352595][1:0x7fc
413c80, txn-recover: [WT_VERB_RECOVERY_ALL] Set global recovery timestamp: (0, 0)})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.352+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WireTiger message","attr":{"message":"[1657223135:352646][1:0x7fc
413c80, txn-recover: [WT_VERB_RECOVERY_ALL] Set global oldest timestamp: (0, 0)})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.370+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WireTiger message","attr":{"message":"[1657223135:370160][1:0x7fc
413c80, WT_SESSION.checkpoint: [WT_VERB_CHECKPOINT_PROGRESS] saving checkpoint snapshot nIn: 4, snapshot max: 4 snapshot count: 0, oldest timestamp: (0, 0), meta checkpoint timestamp: (0, 0) base wr
gen: 628918)})
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.514+00:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WireTiger opened","attr":{"durationMllis":1060}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.514+00:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WireTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$time
p":{"t":{"$date":"2022-07-07T19:45:35.515+00:00"},"s":"I", "c":"STORAGE", "id":4366488, "ctx":"initandlisten","msg":"No table logging settings modifications are required for existing v
edTiger tables","attr":{"loggingEnabled":true}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.519+00:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Timestamp monitor starting"}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.651+00:00"},"s":"I", "c":"NETWORK", "id":4915782, "ctx":"initandlisten","msg":"Updated wire specification","attr":{"oldSpec":{"incomingExternalCl
t":{"minWireVersion":0, "maxWireVersion":13}, "incomingInternalClient":{"minWireVersion":0, "maxWireVersion":13}, "isInternalClient":true}, "newSpec":{"incomingExternalClient":{"minWireVersion":10, "maxWireVersion":13}, "incomingInternalClient":{"minWireVersion":13, "maxWireVersion":13}, "outgoing":{"minWireVersion":13, "maxWireVersion":13}, "isInternalClient":true}}}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.651+00:00"},"s":"I", "c":"STORAGE", "id":5071100, "ctx":"initandlisten","msg":"Clearing temp directory"}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.652+00:00"},"s":"I", "c":"CONTROL", "id":26536, "ctx":"initandlisten","msg":"Flow Control is enabled on this deployment"}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.654+00:00"},"s":"I", "c":"RECOVERY", "id":20625, "ctx":"initandlisten","msg":"Initializing full-time diagnostic data capture","attr":{"dataDirectory":"/data/db/diagnostic.data"}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.656+00:00"},"s":"I", "c":"REPL", "id":6015317, "ctx":"initandlisten","msg":"Setting new configuration state","attr":{"newState":"ConfigReplica
nabled","oldState":"ConfigPrestart"}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.657+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"/tmp/mongodb-27017.sock"}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.657+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"/0.0.0.0"}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:35.657+00:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"listener","msg":"Waiting for connections","attr":{"port":27017,"ssl":"off"}}
mongo_1 | [{"t":{"$date":"2022-07-07T19:45:36.032+00:00"},"s":"I", "c":"FTDC", "id":20631, "ctx":"ftdc","msg":"Unclean full-time diagnostic data capture shutdown detected, found interm
e, some metrics may have been lost","attr":{"error":{"code":0, "codeName":"OK"}}}
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028249Z lvl=info msg=Using data dir log_id=0bVZ0cQG000 service=store path=/var/lib/influxdb/data
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028413Z lvl=info msg=Compaction settings log_id=0bVZ0cQG000 service=store max_concurrent_compactions=2 throughput_bytes_per_second=50331648 throughput_bytes_s
_second_burst=50331648
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028456Z lvl=info msg=Open store (start) log_id=0bVZ0cQG000 service=store trace_id=0bVZ0d10000 op_name=tsdb_open op_event=start
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028501Z lvl=info msg=Open store (end) log_id=0bVZ0cQG000 service=store trace_id=0bVZ0d10000 op_name=tsdb_open op_event=end op_elapsed=0.048ms
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028576Z lvl=info msg=Opened service log_id=0bVZ0cQG000 service=subscriber
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028588Z lvl=info msg=Starting monitor service log_id=0bVZ0cQG000 service=monitor
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028594Z lvl=info msg=Registered diagnostics client log_id=0bVZ0cQG000 service=monitor name=build
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028599Z lvl=info msg=Registered diagnostics client log_id=0bVZ0cQG000 service=monitor name=runtime
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028605Z lvl=info msg=Registered diagnostics client log_id=0bVZ0cQG000 service=monitor name=network
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028612Z lvl=info msg=Registered diagnostics client log_id=0bVZ0cQG000 service=monitor name=system
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028636Z lvl=info msg=Starting precreation service log_id=0bVZ0cQG000 service=shard-precreation check_interval=10m advance_period=30m
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028651Z lvl=info msg=Starting snapshot service log_id=0bVZ0cQG000 service=snapshot
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028653Z lvl=info msg=Starting continuous query service log_id=0bVZ0cQG000 service=continuous_querier
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028672Z lvl=info msg=Starting HTTP service log_id=0bVZ0cQG000 service=http authentication=false
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028680Z lvl=info msg=opened HTTP access log log_id=0bVZ0cQG000 service=http path=stderr
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028759Z lvl=info msg=Listening on HTTP log_id=0bVZ0cQG000 service=http addr=[::]:8080 https=false
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028773Z lvl=info msg=Starting retention policy enforcement service log_id=0bVZ0cQG000 service=retention check_interval=30m
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028923Z lvl=info msg=Listening for signals log_id=0bVZ0cQG000
influxdb_MSR_1 | ts=2022-07-07T19:46:09.028986Z lvl=info msg=Storing statistics log_id=0bVZ0cQG000 service=monitor db_instances=internal db_rp=monitor interval=10s
influxdb_MSR_1 | ts=2022-07-07T19:46:09.029037Z lvl=info msg=Sending usage statistics to usage.influxdata.com log_id=0bVZ0cQG000

```

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

## New Connection

Connect to a MongoDB deployment

URI ⓘ Edit Connection String ☒

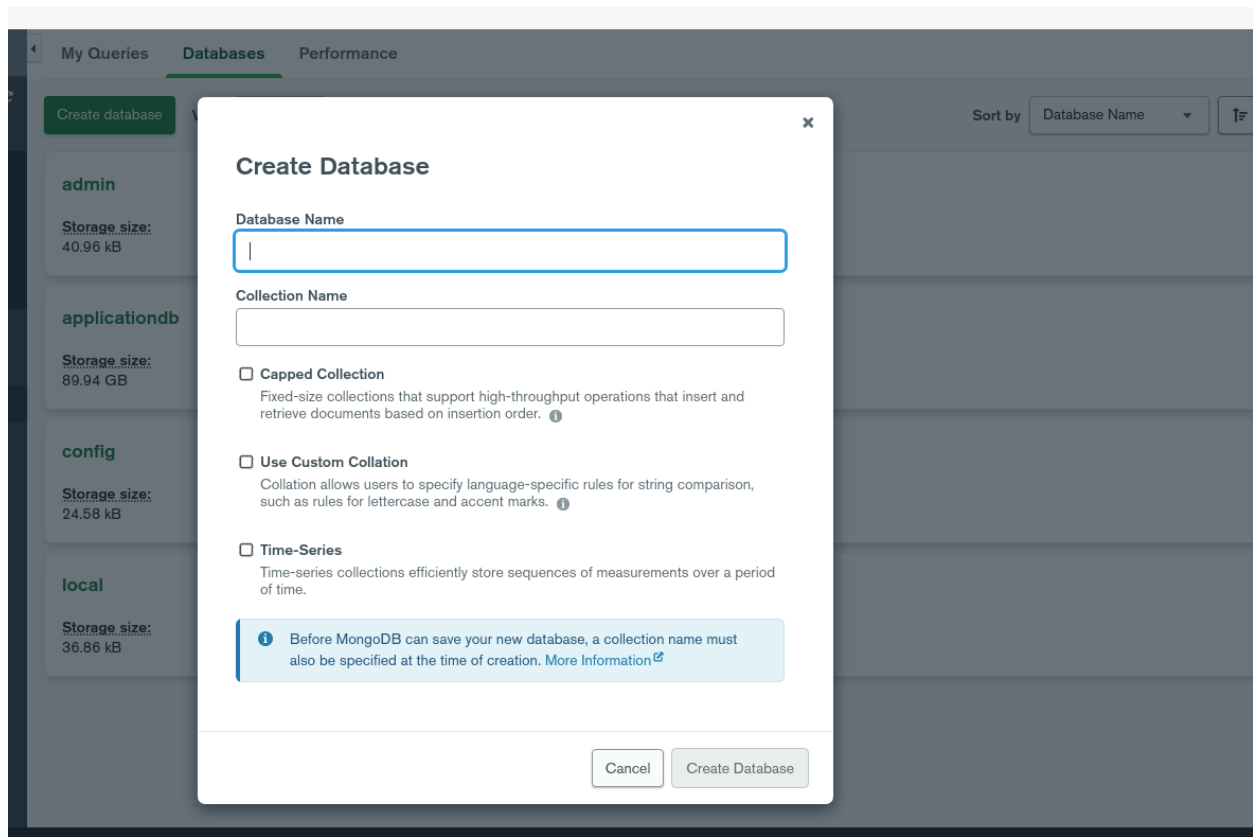
mongodb://root:example@localhost:27017/

➤ Advanced Connection Options

Connect

### 6. create a new database

My Queries	Databases	Performance
Create database	View	Sort by Database Name
admin		
Storage size:	Collections:	Indexes:
40.96 kB	2	3
applicationdb		
Storage size:	Collections:	Indexes:
89.94 GB	1	1
config		
Storage size:	Collections:	Indexes:
24.58 kB	1	2
local		
Storage size:	Collections:	Indexes:
36.86 kB	1	1



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] --- maven-jar-plugin:2.4:jar (default-jar) @ maven-spark-processing ---
[INFO] --- maven-install-plugin:2.4:install (default-install) @ maven-spark-processing ---
[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-spark-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 .
```

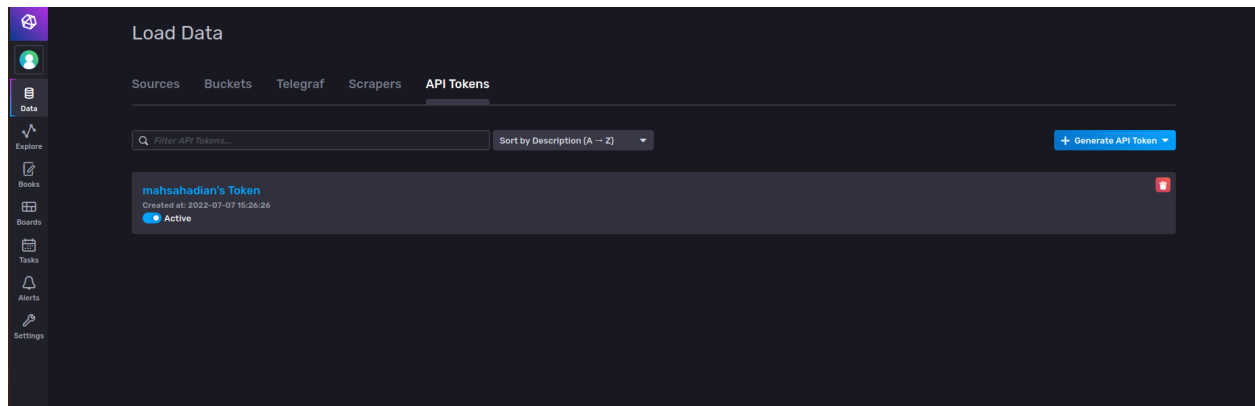
**10. Configuration :** now we need to edit the

```
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.

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

B.

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)

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

C.

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

```
# This is a Mosquitto configuration file that creates a broker
# 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
```

D.

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 creates a configuration
# 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

```
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"
```

3.

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 examples.

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

`cd Edge\ Side/monitoring/`

- a.
- 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

```
json_body = [
    {
        'net_tx': float(self.get_network_throughput(container_stats)['tx'])
        {
            "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