IoT Live Streaming using *Apache Kafka, Apache Flink, HBase and Grafana*

Natalia-Maria Grigoriadou
School of Electrical and Computer Engineering
NTUA
Athens, Greece
el18940@mail.ntua.gr

Orestis Zaras
School of Electrical and Computer Engineering
NTUA
Athens, Greece
el18207@mail.ntua.gr

Georgios Tsiakatara
School of Electrical and Computer Engineering
NTUA
Athens, Greece
el18130@mail.ntua.gr

***Abstract*— This report presents the process of implementing a prototype of an Internet of Things (IoT) livestream system, developed using open-source tools. The system is designed to detect data, process them, store them in a dataset and finally display them, allowing users to monitor events in real-time. The prototype consists of several layers, including Messaging Broker Layer, Live Streaming Layer, Data/Storage Layer, Presentation Layer. The system is built using open-source tools, including Apache Kafka, Apache Flink, HBase and Grafana. The report provides an overview of the system architecture, details the implementation process, and discusses the challenges and limitations encountered during the development process. Overall, this project demonstrates the feasibility of using open-source tools to develop IoT livestream systems, and provides a foundation for further research and development in this area.**

Keywords*— *Internet of Things (IoT), Livestream system, Open-source tools, Data processing, Data storage, Real-time monitoring, Messaging broker ,Live streaming, Presentation layer, Apache Kafka, Apache Flink, HBase, Grafana, Feasibility, Research and development.

I. INTRODUCTION

The Internet of Things (IoT) has revolutionized the way we interact with technology, giving us the ability to connect devices and share data in real-time. One of the most promising applications of IoT is in live data streaming, which provides real-time monitoring of events and data.

In our project, we were asked to use a variety of open-source tools to create a live streaming system which will be a prototype of an actual IoT system. The aim of this project is to simulate a system with sensors that provide energy data (about temperature, water consumption etc.) which after receiving and handle the data will display daily aggregated data (Sum, Average, etc.).

More specifically, we create a system with multiple layers, each for one of the following usages: creation of the data, messaging broker, live streaming, data storage, and data presentation. The open sources we used are Apache Kafka as a messaging Systems-Broker, Apache Flink as live streaming processing system, Hbase for storing the data (database) and Grafana for presenting the data and its aggregations. The system we created receives data in real-time through Kafka Message broker, processes them, transforms them and sends them at the database in specific format. At the end we present the data in Dashboards using Grafana and websockets.

II. SETUP

Below lies a thorough analysis of how to install and setup the system using the code from our public Github repository (https://github.com/Orestiszar/BigData22-33) aiming at facilitating the usage of our system .

A. Set-up the environment

Before using our system make sure that you have installed the libraries that are necessary. Note that our open-sources tools have been tested only on a Unix environment using python's version 3.8.10. So, to install the libraries use the pip command as following:

- pip install random
- pip install time
- pip install datetime
- pip install numpy
- pip install json
- pip install happybase
- pip install flask
- pip install socket
- pip install sys
- pip install confluent_kafka
- pip install socket
- pip install "apache-flink>=1.13.0,<1.14"

"confluent-kafka>=1.7.0,<1.8"

B. Install the open-sources tools

In order to use our system you have to download the open-source tools (Docker for Apache Kafka, Hbase, Grafana).

- For Docker: sudo snap install docker
- For hbase: Follow the instructions of the official reference guide of Hbase
- Grafana: Follow the instructions in the references.

C. Run the system

Since, you have already set-up the environment you are now ready to use the data streaming system. Firstly, in the folder flink_demo run in a terminal the following command:

sudo docker-compose up -d

This will set up the docker containers needed for Kafka. In order to ensure that the containers are active run the following command:

sudo docker ps

You should see two containers running

```
CONTAINER ID   IMAGE                             COMMAND                  CREATE
D      STATUS          PORTS                                                    
                                                                          NAM
ES
d0f309ad0d9c   confluentinc/cp-kafka:6.1.1       "/etc/confluent/dock…"   4 days
 ago   Up 39 minutes   0.0.0.0:9092->9092/tcp, :::9092->9092/tcp, 0.0.0.0:9101->
9101/tcp, :::9101->9101/tcp, 0.0.0.0:29092->29092/tcp, :::29092->29092/tcp   bro
ker
5005b53a182a   confluentinc/cp-zookeeper:6.1.1   "/etc/confluent/dock…"   4 days
 ago   Up 28 hours     2888/tcp, 0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 3888
/tcp                                                                      zoo
keeper
```

After that run the following commands:

- sudo docker exec -it broker kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic input
- sudo docker exec -it broker kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic input_late
- sudo docker exec -it broker kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic output_aggr
- sudo docker exec -it broker kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic output_raw

The Kafka environment is now set up.

Next we set up Hbase. In the hbase folder access /conf/hbase-site.xml and enter the following property:

```
<property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2183</value>
    <description>Property from ZooKeeper's config zoo.cfg.
    The port at which the clients will connect.
    </description>
  </property>
```

This is to ensure that the Zookeeper client needed for Hbase starts in a different port than the Zookeeper client needed for the Kafka broker.

Then, start Hbase using the file /bin/start-hbase.sh that you will find in the extracted folder of Hbase. In the same terminal run the following command:

./hbase-daemon.sh start thrift -p 9090 --infoport 9091

Hbase is now set up.

The next step is to set up the Grafana server.

To install Grafana run:

1. sudo apt-get install -y apt-transport-https
2. sudo apt-get install -y software-properties-common wget
3. sudo wget -q -O /usr/share/keyrings/grafana.key https://apt.grafana.com/gpg.key
4. echo "deb [signed-by=/usr/share/keyrings/grafana.key] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
5. sudo apt-get update
6. sudo apt-get install grafana.

If there are any problems visit the official Grafana documentation in the references.

After this, you should run the following commands in a terminal:

- sudo systemctl start grafana-server
- sudo grafana-cli plugins install grafana-simple-json-datasource
- sudo service grafana-server restart

From a browser connect to localhost:3000 and login using the credentials admin admin. Then you create a new datasource of the type simple json, using url localhost:5000. Save the datasource and go to dashboards and import the json file provided in the folder grafana_TEST.

After these steps, you can now run the following scripts in this order to access the system:

- flink_demo/delete_and_create_tables.py
- flink_demo/happybase-raw.py
- flink_demo/happybase-aggr.py
- flink_demo/processor.py
- flink_demo/DataCreation.py
- grafana_TEST/grafana-api.py

Visit the dashboard in localhost:3000 and change the time interval to 5 seconds.

Identify applicable funding agency here. If none, delete this text box.

III. Analysing each Layer

A. Data Creation Layer

In the first layer of our system we create the data through a python script (DataCreation.py). More specifically, we create data that simulate 10 sensors, 2 of temperature(TH1,TH2),2 for HVAC (HVAC1,HVAC2), 2 for the energy consumption of other electrical devices (MiAC1,MiAC2), 1 sensor for cumulative energy consumption(Etot), 1 for movement detection (MOV1), 1 for water consumption (W1) and 1 for cumulative water consumption (Wtot). The sensors TH1, TH2, HVAC1, HVAC2 and W1 produce data every 1 second that corresponds to a timestamp that differents from the previous by 15 minutes, while the other two produce increasing data every 96 seconds with a timestamp increased by 1 day. The following functions are called for the data creation:

```
def create_data_15min():
    data_dict = {}

    data_dict['TH1']=np.random.uniform(12,35)
    data_dict['TH2']=np.random.uniform(12,35)

    data_dict['HVAC1']=np.random.uniform(0,100)
    data_dict['HVAC2']=np.random.uniform(0,200)

    data_dict['MiAC1']=np.random.uniform(0,150)
    data_dict['MiAC2']=np.random.uniform(0,200)

    data_dict['W1']=np.random.uniform(0,1)

    return data_dict
```

```
def create_data_1day():
    global wtotenergy, etotenergy

    etot = np.random.uniform(-1000,1000)
    etotenergy += (2600*24 + etot)

    wtot = np.random.uniform(-10,10)
    wtotenergy += (100+wtot)
```

Only for the MOV1 sensor we produce data every 20 seconds that corresponds to a timestamp 2 days before and every 120 seconds produces data with a timestamp that corresponds to 10 days before. We forward these to the Kafka topic named ‘input_late’.

B. Message Broker Layer

The second layer we have in our system is a Messaging Broker Layer that sends the data we created to the next layer. In our case we use the Apache Kafka for this job. We created a Kafka topic named ‘input’ in which we send all the produced data from the sensors. The data is then read by the Apache Flink application in order to be aggregated.

We also have two Kafka topics that the Flink processor outputs data to. The first is ‘output_raw’, in which we simply dump the raw data. The second one is ‘output_aggr’ in which we forward the aggregated data in order to be processed.

C. Live Streaming Layer

The layer that follows the Apache Kafka is the Live Streaming Layer that processes the data that received from the message broker and calculates the aggregations metrics, using Apache Flink. The latter receives data from the ‘input’ Kafka topic and calculates some aggregation metrics. For the temperature sensors Flink computes the daily average, for the HVAC, MiAC and W1 computes the daily sum. We also used python scripts to calculate daily aggregations as well as the leak aggregations. In order to do that Flink sends both the raw and the aggregated data to 2 different Kafka topics. After calculating the aggregations we store both the raw and the aggregated data in Hbase tables.
For the aggregation process we used the flink SQL table api. To create the source table we run the following commands:

```
##############################################################
# Create Kafka Source Table with DDL
##############################################################
src_ddl = """
    CREATE TABLE sensor_data (
        m_name VARCHAR,
        m_value DOUBLE,
        m_timestamp TIMESTAMP(3),
        WATERMARK FOR m_timestamp AS m_timestamp
    ) WITH (
        'connector' = 'kafka',
        'topic' = 'input',
        'properties.bootstrap.servers' = 'localhost:9092',
        'properties.group.id' = 'sensor_data_group',
        'format' = 'json'
    )
"""

tbl_env.execute_sql(src_ddl)

# create and initiate loading of source Table
tbl = tbl_env.from_path('sensor_data')
```

We used similar code for the sink table, accordingly:

```
##############################################################
# Create The Kafka Sink Tables
##############################################################
sink_ddl = """
    CREATE TABLE daily_values (
        m_name VARCHAR,
        the_timestamp TIMESTAMP,
        window_daily_values DOUBLE
    ) WITH (
        'connector' = 'kafka',
        'topic' = 'output_aggr',
        'properties.bootstrap.servers' = 'localhost:9092',
        'format' = 'json'
    )
"""
tbl_env.execute_sql(sink_ddl)
```

We also present a SQL aggregation query:

```
sql_sensor_TH1 = """
    SELECT
      m_name,
      MAX(m_timestamp) as the_timestamp,
      AVG(m_value) AS window_daily_values
    FROM sensor_data
    WHERE m_name = 'TH1'
    GROUP BY
      TUMBLE(m_timestamp, INTERVAL '1' DAY),
      m_name
"""
```

For some of the more complicated aggregations we used some python scripts that will be explained in the sections below.

D. Hbase

Now, that we have processed the data and count the necessary aggregations we need to store them. For this we add a DataStorage layer. As mentioned before, we used Hbase for the storage, in which we store both raw and aggregated data in different tables for each sensor. This happens through python scripts using the happybase library, a python library that is specifically developed to interact with hbase.

We have a python script for each of the following purposes:

- Storing raw data – happybase-raw.py
- Storing aggregated data - happybase-aggr.py
- Storing late events – happybase-late.py

Each of those python scripts connects to its matching Kafka topic:

- output_raw
- output_aggr
- input_late

It then collects the data stored there by the Flink processor or the Data creation script and stores them in separate Hbase tables.

In the case of happybase-aggr.py, it also performs the daily aggregations.

E. GRAFANA

For the last layer of our project, we have a Presentation Layer. Using the open-source tool Grafana we provide charts for our data. In order to provide the data to the Grafana , due to the limitations of the hbase implementation, we created an API specifically to get the data from the hbase table we want to access each time and also created an API using Flask that is hosted in localhost:5000 and serves the data to the SimpleJSON datasource.

First things first, we will explain the implementation of database API. Database API implements only one function that is called from the Grafana API and returns an array of json data. The data are fetched from the table that is given as input in the parameters of the function, after connecting to the database. Also the only data we need to transfer to the Grafana API from the Database API are the value and the datetime of each Row from the table. The code of the main function and the connection function is shown below:

```
# gets raw and aggr data
def get_data(table_name):
    conn = connect_to_database()

    data = []
    try:
        table = conn.table(table_name)

        for key, val in table.scan():
            key = key.decode('utf-8').replace('cf:','')
            value = val[b'cf:value'].decode('utf-8')
            time = val[b'cf:datetime'].decode('utf-8')

            data.append(
                {
                "time":time,
                "value":value
                })
    except Exception as e:
        print(e)
        data = [{}]

    return(data)
```

Then we have the Grafana API. In order to be correctly connected to the Simple JSON data source, we must implement the web endpoints ‘/’, ‘/search’ and ‘/query’, which we choose to implement using flask.

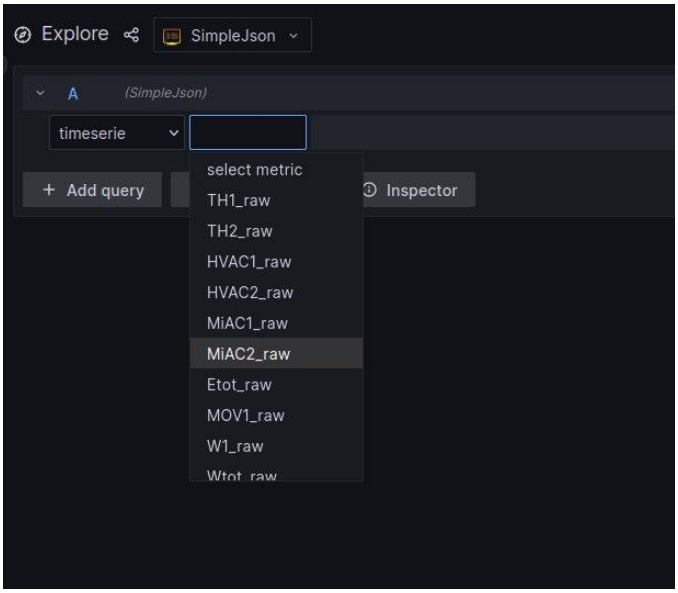
The ‘/’ path is the default path when the data source is connected to the API and responds with a simple ‘Hello World’.

Then we have the ‘/search’ that provides a list of targets that can be chosen in queries to the Simple JSON datasource. The targets are the names of the hbase tables.

The final endpoint is the ‘/query’ endpoint. This endpoint receives request data formatted as JSON that informs what is being queried from the Simple JSON datasource. The endpoint is getting the data from the database API that we explained before and returns a json object in the desired form to be

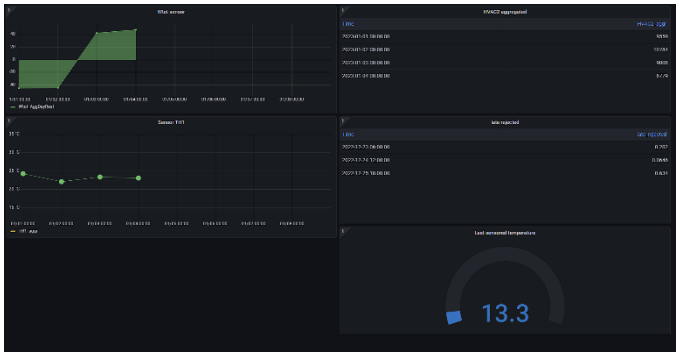
processed by Grafana. The data that is fetched is chosen by the target variable in the request header and it is basically the name of the hbase table we want to access.

The data are then displayed in the given dashboard but can also been seen in the explore section of Grafana by selecting the desired metric as seen below:



IV. RESULTS AND CONCLUSIONS

The result of our project can be portrayed through the following Grafana dashboard:



We can conclude that the tools used for this project can interact with each other to make the creation of a complete IoT application possible.

REFERENCES

[1] Hbase reference guide: https://hbase.apache.org/book.html. *(references)*

[2] https://grafana.com/grafana/plugins/grafana-simple-json-datasource

[3] https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/

[4] https://blog.jonathanmccall.com/2018/10/09/creating-a-grafana-datasource-using-flask-and-the-simplejson-plugin/

[5] https://nightlies.apache.org/flink/flink-docs-master/docs/dev/table/concepts/time_attributes/

[6] https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/datastream/kafka/

[7] https://nightlies.apache.org/flink/flink-docs-master/docs/dev/table/concepts/time_attributes/

[8] https://nightlies.apache.org/flink/flink-docs-release-1.16/

[9] https://github.com/Orestiszar/BigData22-33