

**Đại học Bách Khoa Thành phố Hồ Chí Minh**  
**Khoa Khoa học và Kỹ Thuật Máy Tính**



**Dự án Phân tích dữ liệu lớn IOT dựa trên lượng sử dụng điện và nước**

**Môn học** : Dữ liệu lớn

**GVHD** : PGS.TS Thoại Nam

**Sinh viên thực hiện** : Trần Cao Bảo Ân - 2370297

Huỳnh Thanh Tâm - 2370193

***Thành phố Hồ Chí Minh ngày 29 tháng 05 năm 2024***

## Mục lục

<b>1. Giới thiệu.....</b>	<b>3</b>
<b>2. Hiện thực hệ thống.....</b>	<b>3</b>
<b>2.1. Công nghệ.....</b>	<b>3</b>
2.1.1. Dữ liệu.....	3
2.1.2. Apache Kafka.....	3
2.1.3. Apache Spark.....	3
2.1.4. MongoDB.....	3
2.1.5. Airflow.....	4
2.1.6. Metabase.....	4
<b>2.2. Sơ đồ dự án.....</b>	<b>4</b>
2.2.1. Luồng Batch.....	4
2.2.2. Luồng Stream.....	5
<b>2.3. Cách hoạt động.....</b>	<b>5</b>
2.3.1. Khởi tạo môi trường.....	5
2.3.1.1. Kafka, MongoDB và Metabase.....	5
2.3.1.2. Apache Spark.....	8
2.3.1.3 Airflow.....	9
2.3.2 Tạo dữ liệu mô phỏng.....	10
2.3.3 Xử lý dữ liệu.....	13
2.3.3.1. Luồng Stream.....	13
2.3.3.2. Luồng Batch.....	16
2.3.4 Trực quan dữ liệu.....	18
<b>3. Kết luận.....</b>	<b>19</b>

## **1. Giới thiệu**

Hiện thực dự án theo dõi lượng sử dụng điện và nước trong thành phố thông minh (smart city), cung cấp giải pháp theo dõi máy đo lượng nước và máy đo điện theo thời gian thực, để phát hiện lỗi phát sinh nếu có của từng máy đo, theo dõi lượng tiêu thụ điện và nước của từng hộ dân.

## **2. Hiện thực hệ thống**

### **2.1. Công nghệ**

#### **2.1.1. Dữ liệu**

Để có thể tạo ra lượng dữ liệu liên tục theo thời gian thực, nhóm sử dụng thư viện Faker để mô phỏng môi trường dữ liệu lớn thực tế và cho phép thử nghiệm hệ thống mà không cần sử dụng dữ liệu thực.

#### **2.1.2. Apache Kafka**

Kafka đóng vai trò là một message queue trong luồng xử lý dữ liệu, giúp cho dữ liệu không bị mất đi do tốc độ xử lý thông tin có thể khác so với tốc độ dữ liệu được tạo ra, và dữ liệu không cần thiết phải gửi thẳng tới nơi xử lý, mà nhiều nguồn tạo dữ liệu chỉ cần gửi về Kafka, giúp luồng xử lý dữ liệu và luồng thu thập dữ liệu được độc lập với nhau

#### **2.1.3. Apache Spark**

Spark là một công cụ xử lý dữ liệu lớn hiệu quả, Spark có thể phân tích dữ liệu theo bó (Batch) hoặc theo dòng (Stream) với thư viện Spark Streaming.

#### **2.1.4. MongoDB**

MongoDB là một cơ sở dữ liệu NoSQL được sử dụng để lưu trữ dữ liệu đã được xử lý bởi Spark. MongoDB là một cơ sở dữ liệu linh hoạt và có thể mở rộng, lý tưởng để lưu trữ lượng dữ liệu lớn được tạo ra bởi hệ thống giám sát thông minh.

### 2.1.5. Airflow

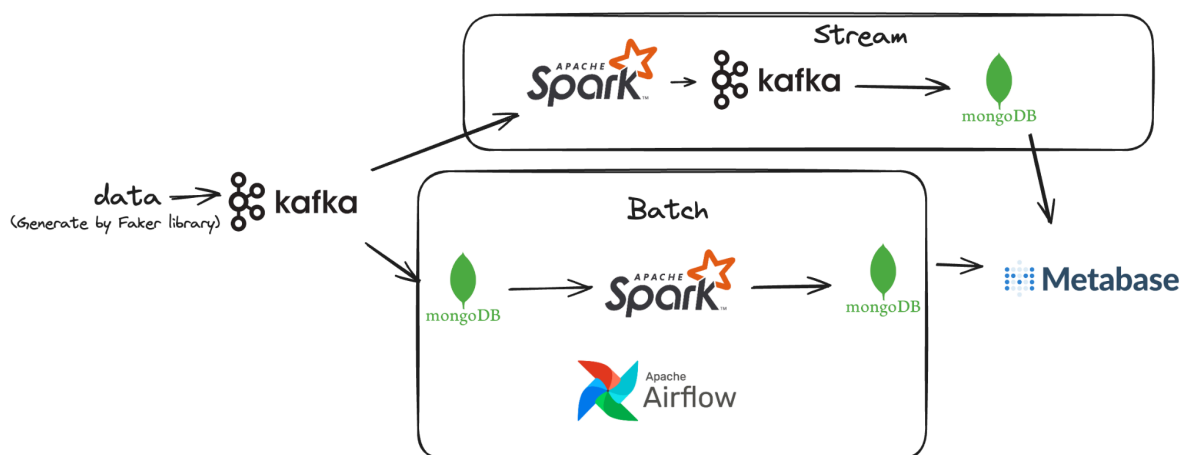
Apache Airflow là một nền tảng mã nguồn mở để lập lịch trình, quản lý và theo dõi các luồng công việc. Nó được sử dụng rộng rãi trong các hệ thống dữ liệu lớn để tự động hóa các tác vụ ETL (Extraction, Transformation, Load). Airflow sẽ được sử dụng để lên lịch cho Spark phân tích dữ liệu theo bố.

### 2.1.6. Metabase

Metabase là một công cụ trực quan hóa dữ liệu mã nguồn mở được sử dụng để hiển thị dữ liệu được lưu trữ trong MongoDB. Metabase cung cấp một giao diện người dùng trực quan cho phép người dùng dễ dàng truy vấn và khám phá dữ liệu. Metabase cho phép sử dụng các câu lệnh truy vấn native của MongoDB, giúp tận dụng được sự linh hoạt của các câu truy vấn Mongo.

Ngoài ra hệ thống còn sử dụng một số công nghệ như Docker để đóng gói hệ thống, Mongo Express để tương tác với MongoDB, ZooKeeper server để khởi chạy Kafka, cùng với ngôn ngữ Python để lập trình cho hệ thống.

## 2.2. Sơ đồ dự án



Dữ liệu được tạo ra sẽ được chuyển tới Kafka, từ đó dữ liệu sẽ được xử lý theo 2 luồng:

### 2.2.1. Luồng Batch

- Sử dụng đoạn code Python lấy dữ liệu từ Kafka sau đó lưu xuống MongoDB
- Airflow sẽ lên lịch cho Apache Spark lấy dữ liệu từ MongoDB theo từng ngày, Xử lý dữ liệu(tính toán lượng tiêu thụ trung bình trong ngày,..) Và lưu xuống MongoDB ở một Collection khác

### 2.2.2. Luồng Stream

- Spark Streaming sẽ liên tục lấy dữ liệu từ Kafka và xử lý (lọc ra máy nào đang bị lỗi,..) và đưa dữ liệu đến Kafka ở một topic khác.
- Một đoạn code Python sẽ lấy dữ liệu được đưa tới Kafka và lưu xuống MongoDB ở một Collection riêng.

Sau đó sử dụng Metabase lấy dữ liệu từ MongoDB để trực quan hóa dữ liệu, dữ liệu liên tục được làm mới (chỉnh sửa thời gian làm mới trong cài đặt của Metabase).

## 2.3. Cách hoạt động

### 2.3.1. Khởi tạo môi trường

#### 2.3.1.1. Kafka, MongoDB và Metabase

File docker-compose.yml

```
version: "3.8"

services:
  zoo1:
    image: confluentinc/cp-zookeeper:7.3.2
    hostname: zoo1
    container_name: zoo1
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
```

ZOOKEEPER\_SERVER\_ID: 1

ZOOKEEPER\_SERVERS: zoo1:2888:3888

kafka1:

image: confluentinc/cp-kafka:7.3.2

hostname: kafka1

container\_name: kafka1

ports:

- "9092:9092"

- "29092:29092"

- "9999:9999"

environment:

KAFKA\_ADVERTISED\_LISTENERS:

INTERNAL://kafka1:19092,EXTERNAL://{DOCKER\_HOST\_IP:-  
127.0.0.1}:9092,DOCKER://host.docker.internal:29092

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP:

INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,DOCKER:PLAINTEXT

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: INTERNAL

KAFKA\_ZOOKEEPER\_CONNECT: "zoo1:2181"

KAFKA\_BROKER\_ID: 1

KAFKA\_LOG4J\_LOGGERS:

"kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR:  
: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1

KAFKA\_JMX\_PORT: 9999

```
KAFKA_JMX_HOSTNAME:
${DOCKER_HOST_IP:-127.0.0.1}
KAFKA_AUTHORIZER_CLASS_NAME:
kafka.security.authorizer.AclAuthorizer
  KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
  depends_on:
    - zoo1
mongo:
  image: mongo:latest
  environment:
    - MONGO_INITDB_ROOT_USERNAME=admin
    - MONGO_INITDB_ROOT_PASSWORD=password
  ports:
    - "27017:27017"
  volumes:
    - mongo-data:/data/db

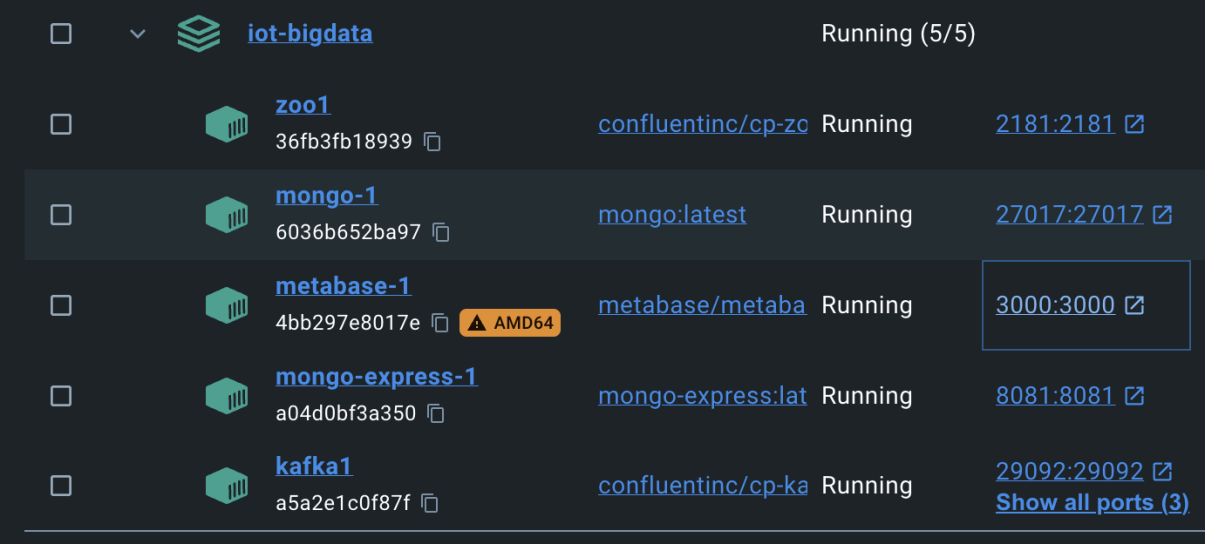
mongo-express:
  image: mongo-express:latest
  environment:
    - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
    - ME_CONFIG_MONGODB_ADMINPASSWORD=password
    - ME_CONFIG_MONGODB_SERVER=mongo
  ports:
    - "8081:8081"
  depends_on:
    - mongo
metabase:
  image: metabase/metabase:latest
  ports:
```

```
- "3000:3000"

volumes:
  - metabase-data:/metabase-data

volumes:
  mongo-data: {}
  metabase-data: {}
```

Sau khi chạy lệnh *docker-compose up*, Kafka, MongoDB, Mongo Express, Zookeeper và Metabase sẽ được khởi tạo.



The screenshot shows the Docker Desktop interface with a list of running containers under the 'iot-bigdata' project. The containers are: zoo1, mongo-1, metabase-1, mongo-express-1, and kafka1. The 'metabase-1' container is highlighted with a blue box around its port mapping '3000:3000'. A warning icon for 'AMD64' is visible next to the 'metabase-1' container name.


Container Name	Image	Status	Port Mapping
zoo1	confluentinc/cp-zo	Running	2181:2181
mongo-1	mongo:latest	Running	27017:27017
metabase-1	metabase/metaba	Running	3000:3000
mongo-express-1	mongo-express:lat	Running	8081:8081
kafka1	confluentinc/cp-ka	Running	29092:29092

Hình: Các môi trường được khởi tạo trong Docker

### 2.3.1.2. Apache Spark

Apache Spark sẽ được khởi tạo riêng, tải về theo đường dẫn <https://spark.apache.org/downloads.html>, Sau đó vào thư mục spark sbin và chạy câu lệnh *.start-master.sh*, web của Spark Master có thể truy cập ở đường dẫn localhost:8080.





3.5.1

Spark Master at spark://Trans-MacBook-Pro.local:7077

URL: spark://Trans-MacBook-Pro.local:7077  
 Alive Workers: 1  
 Cores in use: 8 Total, 0 Used  
 Memory in use: 15.0 GiB Total, 0.0 B Used  
 Resources in use:  
 Applications: 0 Running, 181 Completed  
 Drivers: 0 Running, 0 Completed  
 Status: ALIVE

Workers (1)
 

Worker id	Address	State	Cores	Memory	Resources
worker-20240518100847-192.168.8.138-64861	192.168.8.138:64861	ALIVE	8 (0 Used)	15.0 GiB (0.0 B Used)	

Running Applications (0)
 

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (181)
 

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20240518120255-0540	iot-batch	8	1024.0 MiB		2024/05/18 12:02:55	tranan	FINISHED	6 s
app-20240518120244-0539	iot-batch	8	1024.0 MiB		2024/05/18 12:02:44	tranan	FINISHED	6 s
app-20240518120231-0538	iot-batch	8	1024.0 MiB		2024/05/18 12:02:31	tranan	FINISHED	5 s
app-20240518120220-0537	iot-batch	8	1024.0 MiB		2024/05/18 12:02:20	tranan	FINISHED	5 s


Hình: Giao diện web của Spark Master sau khi khởi chạy

2.3.1.3 Airflow

Download Airflow theo đường dẫn

<https://airflow.apache.org/docs/apache-airflow/stable/start.html>

sau đó chạy lệnh airflow standalone, do port mặc định của airflow trùng với Spark Master nên phải đổi port mặc định trong file airflow.cfg



DAGs

Cluster Activity

Datasets

Security

Browse

Admin

Docs

08:44 UTC AU

DAGs
 

All 49 Active 1 Paused 48 Running 16 Failed 0

Filter DAGs by tag
 Search DAGs
 Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
dataset_consumes_1	airflow		Dataset		On s3://dag1/output_1.txt	
dataset_consumes_1_and_2	airflow		Dataset		0 of 2 datasets updated	
dataset_consumes_1_never_scheduled	airflow		Dataset		0 of 2 datasets updated	
dataset_consumes_unknown_never_scheduled	airflow		Dataset		0 of 2 datasets updated	
dataset_produces_1	airflow		@daily		2024-05-17, 00:00:00	
dataset_produces_2	airflow		None			
example_bash_operator	airflow	3	00***	2024-04-13, 00:00:00	2024-05-17, 00:00:00	

Hình: Giao diện của Airflow.

### 2.3.2 Tạo dữ liệu mô phỏng

```
# Kafka Configuration
bootstrap_servers = ['localhost:9092']
topic_name = 'iot-meter' # Name of your Kafka topic
producer = KafkaProducer(bootstrap_servers=bootstrap_servers)

# Meter IDs
electricity_meters = ["em1", "em2", "em3", "em4"]
water_meters = ["wm1", "wm2", "wm3", "wm4"]

def generate_electricity_data(meter_id):
    usage_kwh = fake_number = fake.random_int(min=-1, max=10)
    timestamp = datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ") # Adjusted
for current sim time
    return {
        "device_id": meter_id,
        "timestamp": timestamp,
        "usage_kwh": usage_kwh
    }

def generate_water_data(meter_id):
    usage = random.randint(5, 20)
    flow_rate = fake_number = fake.random_int(min=-1, max=10)
    timestamp = datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ") # Adjusted
for current sim time
    return {
        "device_id": meter_id,
        "timestamp": timestamp,
        "usage": usage,
```

```

        "flow_rate": flow_rate,
    }

while(True):
    for em in electricity_meters:
        data = generate_electricity_data(em)
        message = json.dumps(data).encode('utf-8')
        producer.send(topic_name, message)
        print(message)
    for wm in water_meters:
        data = generate_water_data(wm)
        message = json.dumps(data).encode('utf-8')
        producer.send(topic_name, message)
        print(message)
    time.sleep(1)# Kafka Configuration
bootstrap_servers = ['localhost:9092']
topic_name = 'iot-meter' # Name of your Kafka topic
producer = KafkaProducer(bootstrap_servers=bootstrap_servers)

# Meter IDs
electricity_meters = ["em1", "em2", "em3", "em4"]
water_meters = ["wm1", "wm2", "wm3", "wm4"]

def generate_electricity_data(meter_id):
    usage_kwh = fake_number = fake.random_int(min=-1, max=10)
    timestamp = datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ") # Adjusted
for current sim time
    return {
        "device_id": meter_id,
        "timestamp": timestamp,

```

```

        "usage_kwh": usage_kwh
    }

def generate_water_data(meter_id):
    usage = random.randint(5, 20)
    flow_rate = fake_number = fake.random_int(min=-1, max=10)
    timestamp = datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ") # Adjusted
for current sim time
    return {
        "device_id": meter_id,
        "timestamp": timestamp,
        "usage": usage,
        "flow_rate": flow_rate,
    }

while(True):
    for em in electricity_meters:
        data = generate_electricity_data(em)
        message = json.dumps(data).encode('utf-8')
        producer.send(topic_name, message)
        print(message)
    for wm in water_meters:
        data = generate_water_data(wm)
        message = json.dumps(data).encode('utf-8')
        producer.send(topic_name, message)
        print(message)
    time.sleep(1)

```

Sử dụng thư viện Faker, tạo dữ liệu mô phỏng với keyword “wm” là máy đo lượng nước (water meter), và “em” là máy đo điện (electricity meter), ở đây nhóm giả lập 5

máy đo cho mỗi loại, đánh số từ 1 tới 5 (“em1”, “em2”, “em3”... và “wm1”, “wm2”,...)

```
b'{"device_id": "wm1", "timestamp": "2024-05-27T15:57:52Z", "usage": 18, "flow_rate": 5}'
b'{"device_id": "wm2", "timestamp": "2024-05-27T15:57:52Z", "usage": 7, "flow_rate": 10}'
b'{"device_id": "wm3", "timestamp": "2024-05-27T15:57:52Z", "usage": 6, "flow_rate": 0}'
b'{"device_id": "wm4", "timestamp": "2024-05-27T15:57:52Z", "usage": 11, "flow_rate": 3}'
b'{"device_id": "em1", "timestamp": "2024-05-27T15:57:53Z", "usage_kwh": 9}'
b'{"device_id": "em2", "timestamp": "2024-05-27T15:57:53Z", "usage_kwh": -1}'
b'{"device_id": "em3", "timestamp": "2024-05-27T15:57:53Z", "usage_kwh": 5}'
b'{"device_id": "em4", "timestamp": "2024-05-27T15:57:53Z", "usage_kwh": 4}'
b'{"device_id": "wm1", "timestamp": "2024-05-27T15:57:53Z", "usage": 12, "flow_rate": 5}'
b'{"device_id": "wm2", "timestamp": "2024-05-27T15:57:53Z", "usage": 20, "flow_rate": 0}'
b'{"device_id": "wm3", "timestamp": "2024-05-27T15:57:53Z", "usage": 17, "flow_rate": 10}'
b'{"device_id": "wm4", "timestamp": "2024-05-27T15:57:53Z", "usage": 9, "flow_rate": 8}'
b'{"device_id": "em1", "timestamp": "2024-05-27T15:57:54Z", "usage_kwh": -1}'
b'{"device_id": "em2", "timestamp": "2024-05-27T15:57:54Z", "usage_kwh": 10}'
```

Hình: Dữ liệu được tạo ra

- Dữ liệu có giá trị -1 thể hiện cho việc máy đo bị lỗi.
- Dữ liệu sẽ được liên tục đưa đến Kafka.

### 2.3.3 Xử lý dữ liệu

#### 2.3.3.1. Luồng Stream

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, avg, count
from pyspark.sql.types import *

spark = SparkSession.builder.appName("Streaming").getOrCreate()

# Kafka Source Configuration
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "iot-meter") \
    .option("startingOffsets", "earliest") \
```

```

.load()

# Define a Unified Schema
schema = StructType([
    StructField("device_id", StringType()), # Assuming string for meter IDs
    StructField("timestamp", StringType()),
    StructField("usage", IntegerType()),
    StructField("usage_kwh", IntegerType()),
    StructField("flow_rate", DoubleType())
])

# Parse JSON and Filter
df = df.select(from_json(col("value").cast("string"), schema).alias("data"))

filtered_df = df.filter(
    (col("data.flow_rate") == -1) | (col("data.usage_kwh") == -1)
)

# Prepare for Kafka (Assuming you only want to output the invalid records)
output_df = filtered_df.selectExpr("to_json(struct(*)) AS value")

# Kafka Sink Configuration (same as before)
query = output_df.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("topic", "iot-abnormal") \
    .option("checkpointLocation",
"/Users/tranan/Desktop/HCMUT/BigData/iot-bigdata/stream-log") \
    .outputMode("update") \
    .start()
query.awaitTermination()

```

Đoạn code sử dụng Spark Streaming lấy dữ liệu từ Kafka, lọc ra dữ liệu nào bị lỗi (dữ liệu là -1) và đưa đến topic Kafka khác.

```
# Take data from spark to mongo
from kafka import KafkaConsumer
from pymongo import MongoClient
import json

# MongoDB Configuration
client = MongoClient("mongodb://admin:password@localhost:27017") # Adjust if
your MongoDB runs differently
db = client['iot']
collection = db['iot-abnormal']

# Kafka Configuration
bootstrap_servers = ['localhost:9092']
topic_name = 'iot-abnormal'

# Create a Kafka Consumer
consumer = KafkaConsumer(topic_name, bootstrap_servers=bootstrap_servers)

# Consume messages from the topic
for message in consumer:
    data = json.loads(message.value.decode('utf-8'))
    print(data['data'])
    collection.insert_one(data['data'])
```

Đoạn code python lấy dữ liệu đã được xử lý từ Spark, lưu vào MongoDB

Mongo Express Database: iot- Collection: iot-abnormal

### Viewing Collection: iot-abnormal

New Document New Index

Simple Advanced

Key Value String Find

Delete all 109 documents retrieved

1 2 3 4 5 > >>

_id	device_id	timestamp	usage	flow_rate	usage_kwh
66481ce51d2f2d5d0db04513	wm4	2024-05-18T10:13:40Z	13	-1	
66481ce61d2f2d5d0db04514	wm1	2024-05-18T10:13:41Z	9	-1	
66481ce71d2f2d5d0db04515	em1	2024-05-18T10:13:42Z			-1

Hình: Dữ liệu sau khi được xử lý theo luồng, sử dụng Mongo Express để kiểm tra.

### 2.3.3.2. Luồng Batch

Để chạy được luồng Batch, đầu tiên phải setup Airflow

```
import airflow
from datetime import datetime, timedelta
from airflow.operators.bash_operator import BashOperator

default_args = {
    'owner': 'jozimar',
    'start_date': datetime(2020, 11, 18),
    'retries': 10,
    'retry_delay': timedelta(hours=1)
}

with airflow.DAG('iot-batch',
                 default_args=default_args,
                 schedule_interval='* * * * *') as dag:
    task_elt_documento_pagar = BashOperator(
```



```
task_id='elt_documento_pagar_spark',  
bash_command="python  
/Users/tranan/Desktop/HCMUT/BigData/iot-bigdata/ETL/Batch.py",  
)
```

Đoạn code setup Airflow, nhằm mục đích mô phỏng, nhóm sẽ chạy Airflow mỗi phút một lần (thực tế sẽ chạy mỗi ngày vào cuối ngày)

```
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when, avg, current_date, to_date, lit  
import datetime  
from datetime import date  
  
appName = "iot-batch"  
master = "spark://Trans-MacBook-Pro.local:7077"  
# Create Spark session  
spark = SparkSession.builder \  
    .appName(appName) \  
    .master(master) \  
    .config("spark.mongodb.input.uri",  
"mongodb://admin:password@localhost:27017/iot.iot-meter?authSource=admin") \  
    .config("spark.mongodb.output.uri",  
"mongodb://admin:password@localhost:27017/iot.iot-meter?authSource=admin") \  
    .config('spark.jars.packages',  
'org.mongodb.spark:mongo-spark-connector_2.12:3.0.1') \  
    .getOrCreate()  
  
# Read data from MongoDB  
df = spark.read.format('mongo').load()  
# df.printSchema()  
df.show()
```

```

df = df.withColumn("meter_type", when(col("usage_kwh").isNotNull(),
"electricity") \
    .otherwise("water"))

# Filter for the current date
today = datetime.date.today()
# today = date(2024, 4, 17)
df_filtered = df.filter(to_date("timestamp") == today)

# Calculate average usage per meter type for the current date
df_averages = df_filtered.groupBy("meter_type").agg(
    avg(col("usage").cast("double")).alias("avg_usage"),
    avg(col("usage_kwh").cast("double")).alias("avg_usage_kwh"),
    current_date().alias("calculation_date")
    # lit(today).cast("date").alias("calculation_date")
)

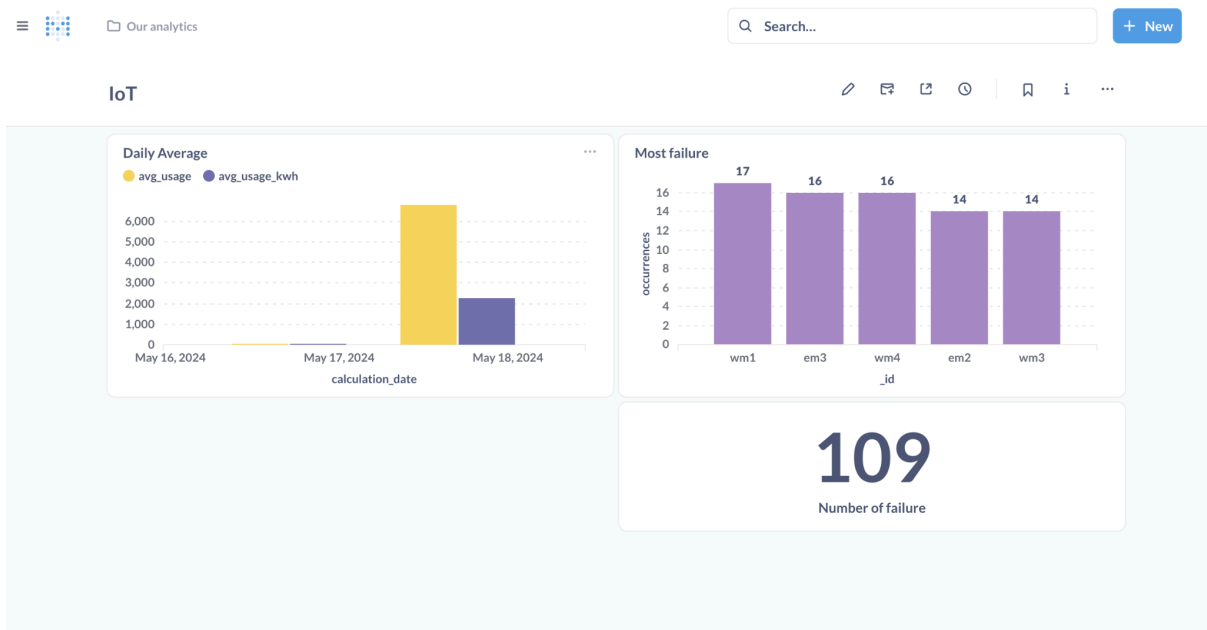
# Save the result DataFrame back to MongoDB
df_averages.write.format("mongo") \
    .mode("append") \
    .option("database", "iot") \
    .option("collection", "daily_averages") \
    .save()

```

Airflow sẽ submit đoạn code Spark này để xử lý, đoạn code trên sẽ tính lượng tiêu thụ trung bình trong ngày và lưu vào MongoDB.

### 2.3.4 Trực quan dữ liệu

Sử dụng Metabase lấy dữ liệu



Hình: Biểu đồ thể hiện dữ liệu

### 3. Kết luận

Project đã mô phỏng được cách thức hoạt động của một dự án theo dõi năng lượng tiêu thụ trong thành phố thông minh, xử lý một lượng dữ liệu lớn và theo thời gian thực.

Tuy nhiên vẫn còn một số điều có thể cải thiện:

- Các công nghệ khi khởi tạo chưa thống nhất (Airflow và Spark vẫn phải khởi tạo riêng) làm cho việc setup mất nhiều thời gian, nên đóng gói Airflow và Spark vào chung môi trường Docker với các công nghệ khác
- Hiện chỉ sử dụng một cluster Kafka, Mongo và Spark, thực tế các công nghệ này có thể khởi tạo nhiều cluster theo hướng phân tán, giúp cho việc xử lý dữ liệu lớn được hiệu quả hơn.

