

KAFKA#IOT



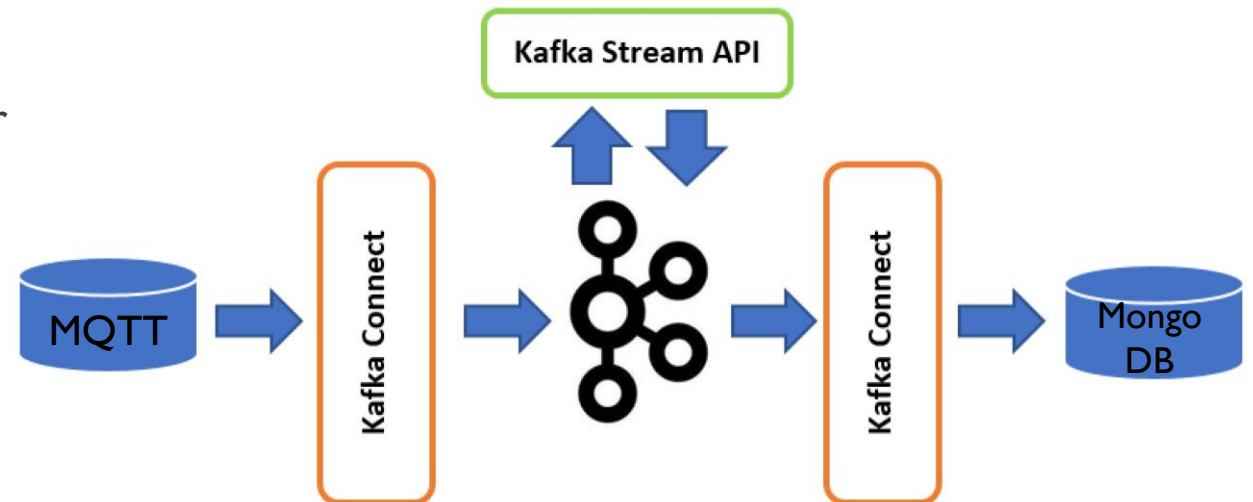
DATA ANALYTICS - A.Y. 2020-21
DIEGO DIOMEDI - LUCIA PASSERI

DESCRIPTION AND OBJECTIVE

Objective: perform data analysis on IoT data with Kafka Streams and ksqlDB.

Description:

- Data are provided by Filippetti Device Simulator
- Data are ingested in real time from MQTT broker and analysed with Kafka Streams and ksqlDB
- Output analytics was saved on an external system: MongoDB



TECHNOLOGIES

- Ubuntu 18.04
- Java 8
- Mqtt-spy
- Mosquitto
- Apache Kafka
- Confluent Platform
- KsqlDB
- MongoDB
- MongoDB Compass
- Git and Github



TECHNOLOGIES

Thanks to Mosquitto and mqtt-spy, we were able to read the messages published by the sensors.



Confluent Platform allows us to make a streaming analytics and filtering data that comes from mqtt broker.



We decided to use MongoDB as the source sink to save the data.



TECHNICAL IMPLEMENTATION

What to install:

- Ubuntu 18.04
- Java 8
- JavaFX
- Mqtt-spy
- Mosquitto
- Filippetti Simulator
- Confluent Platform
- Connectors from confluent-hub

TECHNICAL IMPLEMENTATION

What to install:

- Ubuntu 18.04
- Java 8
- JavaFX
- Mqtt-spy
- Mosquitto
- Filippetti Simulator
- Confluent Platform
- Connectors from confluent-hub



Start with command:
`confluent local services start`

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can procede:

```
lucia@lucia-VirtualBox:~$ confluent local services start
The local commands are intended for a single-node development environment only,
NOT for production usage. https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /tmp/confluent.396829
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
```

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can procede:

- Browse <http://localhost:9021/clusters> and select your *cluster*

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can procede:

- Browse `http://localhost:9021/clusters` and select your *cluster*
- Add kafka *topic*

ALL TOPICS >

New topic

Topic name*

audit-log

Number of partitions* ⓘ

1

Create with defaults

Customize settings

Cancel

Topic Summary

name

audit-log

partitions

1

replication.factor

--

cluster

Kafka Raleigh

min.insync.replicas

--

cleanup.policy

--

retention.ms

604800000

retention.bytes

-1

max.message.bytes

1000012

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can proceed:

- Browse `http://localhost:9021/clusters` and select your *cluster*
- Add kafka *topic*
- Add *MQTT Source connector* and set appropriate configurations

Add Connector

01 Setup connection

02 Test and verify

```
{
  "name": "source-mqtt",
  "connector.class": "io.confluent.connect.mqtt.MqttSourceConnector",
  "tasks.max": "1",
  "mqtt.server.uri": "tcp://127.0.0.1:1883",
  "kafka.topic": "mqtt-source-1",
  "mqtt.topics": "#"
}
```

Launch

Back

[Download connector config file](#)

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can proceed:

- Browse <http://localhost:9021/clusters> and select your *cluster*
- Add kafka *topic*
- Add *MQTT Source connector* and set appropriate configurations
- Create KSQL *Streams* and *Tables*

The screenshot shows the 'Create Stream' form in the KSQL UI. The form is organized into several sections:

- Topic:** A text input field containing 'pageviews' with a lock icon to its right.
- Stream name:** A text input field containing 'PAGEVIEWS'.
- Query type:** A dropdown menu with 'Stream' selected.
- Encoding:** A dropdown menu with 'AVRO' selected.
- Key:** A dropdown menu.
- Timestamp:** A dropdown menu.
- Field(s) you'd like to include in your Stream:** A section containing three rows of field definitions:
 - Field name: 'viewtime', Field type: 'BIGINT', with a trash icon to the right.
 - Field name: 'userid', Field type: 'VARCHAR', with a trash icon to the right.
 - Field name: 'pageid', Field type: 'VARCHAR', with a trash icon to the right.
- + Add another field:** A link to add more fields.
- Buttons:** 'Save Stream' and 'Back' buttons at the bottom.

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can procede:

- Browse `http://localhost:9021/clusters` and select your *cluster*
- Add kafka *topic*
- Add *MQTT Source connector* and set appropriate configurations
- Create KSQL *Streams* and *Tables*
- Write *queries* in KSQL EDITOR page and run them:
 1. Non-persistent query: `SELECT column FROM stream EMIT CHANGES;`
 2. Persistent query: `CREATE STREAM name AS SELECT column FROM stream WHERE column='FAMALE';`

TECHNICAL IMPLEMENTATION

Once all services are [UP] we can proceed:

- Browse `http://localhost:9021/clusters` and select your *cluster*
- Add *kafka topic*
- Add *MQTT Source connector* and set appropriate configurations
- Create *KSQL Streams and Tables*
- Write *queries* in KSQL EDITOR page and run them:
 1. Non-persistent query: `SELECT column FROM stream EMIT CHANGES;`
 2. Persistent query: `CREATE STREAM name AS SELECT column FROM stream WHERE column='FEMALE';`
- Add *MongoDB Sink connector* and set appropriate configurations to save data into database

01 Setup connection

02 Test and verify

```
{
  "value.converter.schema.registry.url": "http://localhost:8081",
  "key.converter.schema.registry.url": "http://localhost:8081",
  "schemas.enable": "false",
  "key.converter.schemas.enable": "false",
  "value.converter.schemas.enable": "false",
  "name": "MongoSinkConnectorConnector_0",
  "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
  "tasks.max": "1",
  "key.converter": "org.apache.kafka.connect.storage.StringConverter",
  "value.converter": "org.apache.kafka.connect.json.JsonConverter",
  "topics": [
    "default_ksql_processing_log"
  ],
  "connection.uri": "mongodb+srv://dbuser:password123!@cluster0.nagab.mongodb.net/test",
  "database": "da",
  "collection": "q1"
}
```

Launch

Back

[Download connector config file](#)

SIMULATOR DATA FORMAT

It is the data format for each sensor.

```
{
  "t": timestamp in secondi
  "tz": timestamp in HHMMDDtHHMMSS
  "uuid": identificativo univoco del messaggio
  "cuid":
  "ref": "jzp://edv#0503.0000" identificativo fisico del device
  "type": "presence" tipo di messaggio del sensore
  "cat": "0610"
  "sn": "integer(0, 255)",
  "m": [
    {
      "t": "nowTimestamp()",
      "tz": "now()",
      "k": "device_temperature" tipo di misura: temperatura del device
      "v": "double(0, 40)", valore della misura
      "u": "C" unità di misura
    },
  ],
}
```

QUERIES

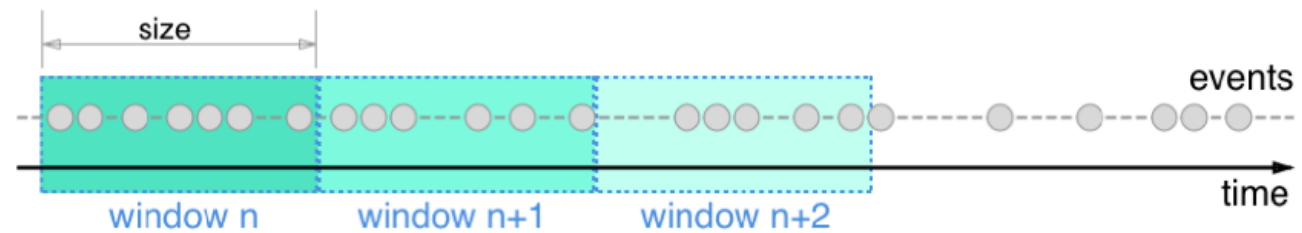
- [BASE]

```
CREATE STREAM base_stream (ref varchar KEY, type varchar, m array<struct<k varchar, v double>>)
WITH (kafka_topic='mqtt', value_format='JSON_SR');
```

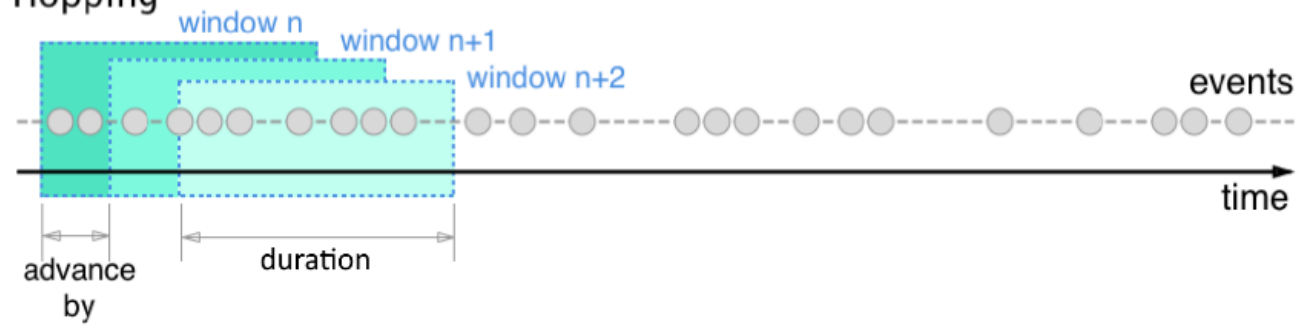
```
CREATE STREAM exploded_base AS
SELECT ref, type, EXPLODE(m)->k AS name, EXPLODE(m)->v AS value
FROM base_stream
EMIT CHANGES;
```

WINDOW TIME

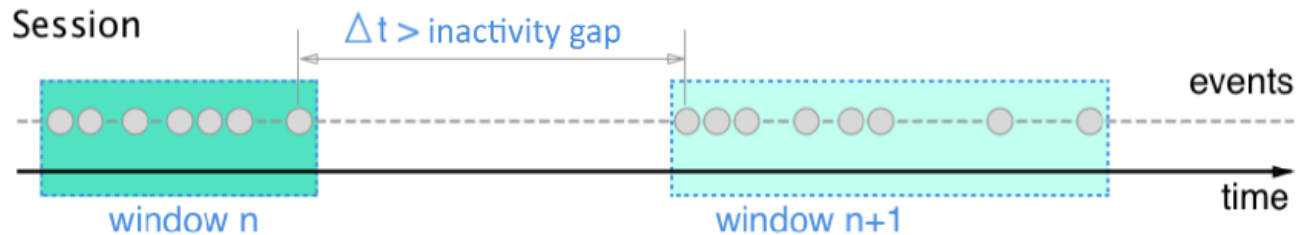
Tumbling



Hopping



Session



QUERIES

- [QUESTION A] – numero messaggi letti in una finestra temporale

```
CREATE TABLE questiona WITH (value_format='JSON') AS
SELECT I, TIMESTAMPTOSTRING(WINDOWSTART, 'yyyy-MM-dd HH:mm:ss', 'Europe/London') AS start_ts,
count(*) AS count
FROM base_stream WINDOW TUMBLING (SIZE 60 SECONDS)
GROUP BY I
EMIT CHANGES;
```

QUERIES

- [QUESTION B] – numero messaggi letti e categorizzati per tipologia di device (type) in una finestra temporale

```
CREATE TABLE questionb WITH (value_format='JSON') AS
SELECT type, TIMESTAMPTOSTRING(WINDOWSTART, 'yyyy-MM-dd HH:mm:ss', 'Europe/London') AS start_ts,
count(*) AS count
FROM base_stream WINDOW TUMBLING (SIZE 60 SECONDS)
GROUP BY type
EMIT CHANGES;
```

QUERIES

- [QUESTION C] – numero messaggi letti e categorizzati per ID di device (ref) in una finestra temporale

```
CREATE TABLE questionc WITH (value_format='JSON') AS
SELECT ref, TIMESTAMPTOSTRING(WINDOWSTART, 'yyyy-MM-dd HH:mm:ss', 'Europe/London') AS start_ts,
count(*) AS count
FROM base_stream WINDOW TUMBLING (SIZE 60 SECONDS)
GROUP BY ref
EMIT CHANGES;
```

QUERIES

- [QUESTION D] – calcolo min, max, avg per ciascuna misura in una finestra di messaggi (vedi i campi k e v negli array 'm' del json)

```
CREATE TABLE questiond WITH (value_format='JSON') AS
SELECT ref, name, MIN(value) AS min, MAX(value) AS max, AVG(value) AS average
FROM exploded_base WINDOW TUMBLING (SIZE 5 MINUTES)
WHERE value > 0
GROUP BY ref, name
EMIT CHANGES;
```

QUERIES

- [QUESTION E] – generare un evento alla lettura di una specifica misura di un sensore con valore x (ad esempio se il pir riporta 1 nel campo presence.. di struttura della m)

```
CREATE STREAM questione WITH (value_format='JSON') AS  
SELECT ref, name, value  
FROM exploded_base  
WHERE name='device_temperature' AND value > 35;
```

QUERIES

- [QUESTION F] – generare un evento se una specifica misura di un sensore ha superato un valore x (ad esempio, se nella finestra ho registrato almeno 5 presenza del sensore pir...)

```
CREATE TABLE questionf WITH (value_format='JSON') AS
SELECT ref, name, value, count(*) AS count
FROM exploded_base WINDOW TUMBLING (SIZE 60 SECONDS)
WHERE name='device_temperature' AND value > 35
GROUP BY ref, name, value
HAVING count(*) > 5;
```

QUERIES

- [QUESTION G] – generare un evento se una specifica misura di un sensore ha superato un valore Y ed un altro sensore ha come media un valore Y nella stessa finestra (ad esempio, se nella finestra ho registrato almeno 5 presenza del sensore pir ed il valore di lux è mediamente K - lo date come predefinito all'avvio del job.....)

```
CREATE TABLE questiong WITH (value_format='JSON') AS
SELECT ref, avg(value) AS average
FROM exploded_base WINDOW TUMBLING (SIZE 60 SECONDS)
WHERE name='device_temperature'
GROUP BY ref
HAVING (count(*) > 2 AND avg(value) > 25)
EMIT CHANGES;
```

ACHIEVED RESULTS

We observed in *Confluent Platform Editor page* that all the queries produced the expected results (in two views).

The image shows two side-by-side screenshots of the Confluent Platform interface. Both screenshots display a 'Data structure' view for a 'STREAM'. The left screenshot shows a table with columns 'PAGEID', 'Page_39', 'Page_35', and 'Page_90'. The right screenshot shows a table with columns 'PAGEID', 'Page_39', 'Page_35', and 'Page_90'. Both screenshots also show a 'Messages/sec' and 'Total message bytes' section.

The results obtained are made persistent thanks to the saving action in *MongoDB*.

Repo: <https://github.com/LuciaPasseri/Kafka>

The image shows a screenshot of the MongoDB interface. The 'Documents' tab is selected, showing a list of documents. The filter is set to '{ field: 'value' }'. The results show three documents, each with a '_id' field and a 'device_temperature' field. The documents are displayed in a table format.

FUTURE WORKS

- Work with data provided by the *Filippetti sensors* (in this project we worked with the Simulator)
- Realization of new *queries* to obtain other different results
- Use *ElasticSearch* as Sink connector to save the data



THANKS FOR THE ATTENTION!

