




DMPA: hirugarren entrega

---

## *Memoria*

---

Aimar Barrena  
Eneko Zabaleta



<b>Soluzioaren azalpen zehatza.....</b>	<b>2</b>
<b>Soluzioa osatzen duten fitxategiak .....</b>	<b>2</b>
Mosquitto (karpeta) .....	2
mysql-init (karpeta).....	2
connect-mqtt-source.json .....	3
Docker-compose.yml .....	3
Dockerfile.....	3
Requirements.txt .....	3
sender.py .....	3
<b>Erabilitako teknologiak eta tresnak .....</b>	<b>3</b>
<b>Hasieraketarako beharrezkoak diren pausuak.....</b>	<b>3</b>
<b>Soluzioaren funtzionamendua erakusten duten pantaila argazkiak.....</b>	<b>5</b>

## Soluzioaren azalpen zehatza

---

Lanaren helburua Twitterreko erabiltzaileen inguruko lau fitxategitan dauden datuak erabiliz datuen integrazio eta analisirako pipeline bat osatzea da. Datu hauek iturri desberdinetatik jasoko dira (JSON eta CSV), streaming testuinguruan prozesatuko dira eta azkenik bistaratu ahal izango dira. Datuen bidalketa, bilketa, eraldaketa eta bistaratzea egiteko hainbat teknologia erabiltzen dira: MQTT, Kafka, Flink, MySQL, Elasticsearch eta Kibana.

user\_tweets.json fitxategiko txioak Python bidez irakurri eta MQTT protokoloa erabiliz Mosquitto zerbitzarira bidaltzen dira streaming fluxu bat simulatuz. Script hau Docker edukiontzi batean exekutatzen da, eta bertan MQTT mezularitza protokoloa erabiltzen duen Paho liburutegia erabiltzen da. Mosquitto zerbitzariak mezu horiek jasotzen ditu, eta MQTT → Kafka bridge edo Kafka Connect erabiliz Kafka zerbitzura bidaltzen dira, streaming ingurunean lan egiteko.

mbti\_labels.csv fitxategian dauden datuak MySQL kontainerrean automatikoki kargatzen dira, kontainerren hasieraketarekin batera exekutatzen den init.sql script baten bidez. Script honek taula bat sortzen du eta CSV-ko datuak bertara sartzen ditu. Taula honetan erabiltzaile bakoitzaren MBTI pertsonalitate profilari buruzko informazioa dago.

Flink SQL bezeroaren bidez, Kafka eta MySQL zerbitzuekin konektatu eta bi taula sortzen dira. Lehenengo taulan (personalities) MySQL-tik MBTI informazioa jasotzen da: erabiltzailearen id-a, bere pertsonalitate motakoa eta identifikatzaile osagarri bat. Bigarren taulan (tweets) Kafka-n jasotako txioak daude: erabiltzailearen id-a, txioaren testua eta mezuaren prozesatze denbora (proctime). Bi taula hauek erabiliz, Flink-en SQL bidez hirugarren taula bat sortzen da, count\_per\_personality, non pertsonalitate motako txioen kopurua kalkulatzen den.

Hirugarren taulako datuak Elasticsearch-en gordetzen dira. Datu horiek Kibanaren bitartez bistaratzen dira. Kibanan index bat sortzen da Elasticsearch-en gordetako datuekin eta, horren bidez, grafiko bat sor daiteke pertsonalitate bakoitzak zenbat txio bidaltzen dituen ikusteko.

## Soluzioa osatzen duten fitxategiak

---

### Mosquitto (karpeta)

---

MQTT zerbitzari (broker) gisa Mosquitto erabiltzeko beharrezkoak diren fitxategi guztiak biltzen dituen direktorioa da. Hau da, konfigurazioa, datuak eta log-ak gordetzen dira hemen.

#### Config (karpeta) -- mosquitto.conf

Mosquitto-ren konfigurazio nagusia definitzen duen fitxategia da. Zerbitzariaren jokabidea zehazten du: zer portutan entzungo duen, zertarako baimenak dauden, eta abar.

#### Data (karpeta) -- mosquitto.db

Mosquitto-ren egoera gordetzen duen datu base fitxategia da. Gaien egoera, bezeroak, eta abar grabatzen dira.

#### Log (karpeta) -- mosquitto.log

Mosquitto-ren jardura guztia (konexioak, mezuak, erroreak...) jasotzen dituen log fitxategia da.

### mysql-init (karpeta)

---

## [init.sql](#)

MySQL datu basearen egitura edo hasierako datuak definitzen dituen SQL script bat da. Adibidez, taulak sortzea edo datu batzuk txertatzea.

## [mbti\\_labels.csv](#)

CSV formatuko fitxategia da, eta MBTI (pertsonalitate motak) etiketa edo etiketak jasotzen ditu. Baliteke datu analisi edo ML helburuetarako erabiltzea.

## [connect-mqtt-source.json](#)

---

JSON formatuko konfigurazio fitxategia da. MQTT iturburua konektatzeko informazioa daukana: zer gaiei entzun, nora bidali, eta abar.

## [Docker-compose.yml](#)

---

Docker-compose.yml fitxategia, guztietatik, funtsezkoena da. Azken finean, Docker kontainerren zerbitzuak kudeatzeko balio du. Hau da, erabiliko digun tresna guztiak elkar-komunikatzeko eta elkar-lanean aritzeko konfiguratzeko ditu.

## [Dockerfile](#)

---

Python script-a erabili ahal izateko (sender.py) docker kontainer bat sortzeko balio duen fitxategia da hau. Honi esker, txioak Kafka-ra bidaliko dira.

## [Requirements.txt](#)

---

Python proiektu honetan beharrezko diren paketeak zerrendatzen dituen fitxategia da. Dockerfile-ek hau erabiltzen du ingurunea prestatzeko (pip install -r requirements.txt).

## [sender.py](#)

---

Python script honek MQTT bidez jasotako datuak hartu, prozesatu, eta MySQL datu basera bidaltzeko ardura duen kodea da. Sistema osoaren datu-fluxua bideratzen du.

## **Erabilitako teknologiak eta tresnak**

---

Erabilitako teknologiak hauek dira: Python, streaming datuak bidaltzeko script-ak sortzeko; Mosquitto, MQTT mezuak jasotzeko broker modura; Kafka, datu horiek streaming bidez banatzeko; MySQL, datu egituratuak gordetzeko; Apache Flink, Kafka eta MySQL-en arteko datuen integrazioa eta prozesamendua egiteko; Elasticsearch, datu prozesatuak biltegiratzeko; eta Kibana, azken datuak bistaratzen dituen tresna gisa. Docker eta docker-compose ere funtsezkoak dira zerbitzu guztiak edukiontzi bidez abiarazi eta koordinatzeko. Kafka Connect erabiltzen da MQTT bidez bidalitako mezuak Kafka-ra igarotzeko, eta Paho liburutegia Python-en bidez MQTT mezularitzarako.

## **Hasieraketarako beharrezkoak diren pausuak**

---

Proiektua abiarazteko, lehenik eta behin proiektuaren direktorio nagusia prestatzen da eta bertan datuak izeneko karpeta sortzen da JSON eta CSV fitxategiekin. Ondoren, docker-compose.yml fitxategia definitzen da Mosquitto, Kafka (eta Zookeeper), MySQL (init.sql bidez hasieratuta), Flink cluster, Elasticsearch eta Kibana zerbitzuak kontainer moduan exekutatzeko.

Python script-a edukiontzi batean exekutatu ahal izateko, Dockerfile bat eta requirements.txt fitxategia prestatu behar dira. Gero, sender.py script-a moldatzen da user\_tweets.json fitxategitik ausaz aukeratutako txioak irakurri eta Paho liburutegia erabiliz MQTT bidez bidaltzeko. MQTT eta Kafka arteko zubia konfiguratzen da Kafka Connect bidez. Ondoren, Flink SQL bezeroa erabilita, MySQL-eko datuak eta Kafka-ko txioak gordetzen dituzten bi taula sortzen dira. Bi horiek erabiliz, count\_per\_personality izeneko taula bat sortzen da SQL kontsulta baten bidez, eta emaitzak Elasticsearch-era bidaltzen dira. Azkenik, Kibanan index bat sortzen da eta horren gainean pertsonalitate bakoitzaren txio kopurua bistaratzen duen grafikoa eraikitzen da.

Lehenengo komandoa, `curl -d @connect-mqtt-source.json -H "Content-Type: application/json" -X POST http://localhost:8083/connectors, connect-mqtt-source.json` fitxategia Kafka Connect-era bidaltzeko erabiltzen da. Fitxategi honek konfigurazioa dauka Mosquitto MQTT zerbitzari batetik mezuak jasotzeko eta connect-custom izeneko Kafka topic batera bidaltzeko. Horrela, MQTT bidez jasotako datuak automatikoki Kafka-ra igarotzen dira.

Ondoren,

```
docker exec kafka kafka-console-consumer
--bootstrap-server kafka:9092
--topic connect-custom
--from-beginning
```

komandoak Kafka topic-eko mezuak kontsolan ikusteko balio du. Horrela ikus daiteke ea MQTT mezuak ondo iristen ari diren Kafka-ra. `--from-beginning` erabilita, topic-aren hasieratik irakurtzen ditu mezu guztiak.

Gero, MySQL edukiontzira sartzen gara `docker exec -it mysql mysql -u root -p` erabiliz ziurtatzeko datuak ondo kargatu direla. Pasahitz gisa `root` sartzen da. Behin barruan, `USE mbti;` komandoa exekutatzen da mbti datu-basea aukeratzeko eta ondoren `SELECT * FROM mbti_labels LIMIT 10;` taulako lehen 10 erregistroak kontsultatzeko.

Jarraian, Flink-eko jobmanager edukiontzira sartzen gara `docker exec -it jobmanager bash` erabiliz, eta SQL bezeroa irekitzen dugu `/opt/flink/bin/sql-client.sh -l /opt/flink/lib` komandoarekin. Honek konektoreetarako liburutegiak kargatzen ditu.

SQL bezeroan, lehenik pertsonalitateak izeneko taula sortzen da, MySQL-eko mbti\_labels taulari lotua. Taula honek id, mbti\_personality eta pers\_id zutabeak izango ditu. JDBC konektorea erabilita, Flink-ek MySQL-etik datuak irakurri ahal izango ditu.

```
CREATE TABLE pertsonalitateak (
  id BIGINT,
  mbti_personality STRING,
  pers_id INT
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://mysql:3306/mbti',
  'table-name' = 'mbti_labels',
  'username' = 'root',
  'password' = 'root'
);
```

Ondoren, `SELECT * FROM pertsonalitateak LIMIT 10;` kontsulta exekutatzen da datuak zuzen irakurtzen direla egiaztatzeko.

Jarraian, txioak izeneko beste taula bat sortzen da, Kafka-ko connect-custom topic-ean oinarrituta. Honek user\_id, text eta proctime (prozesatze-denbora) zutabeak izango ditu. Kafka konektorea erabilita, Flink-ek streaming bidezko txioak irakurriko ditu.

```
CREATE TABLE txioak (
  user_id BIGINT,
```

```

text STRING,
proctime AS PROCTIME()
) WITH (
'connector' = 'kafka',
'topic' = 'connect-custom',
'properties.bootstrap.servers' = 'kafka:9092',
'format' = 'json',
'scan.startup.mode' = 'earliest-offset'
);

```

`SELECT * FROM txioak LIMIT 10;` erabilita, txioen edukia ikusten da.

Hirugarren taula bat sortzen da: `count_per_personality`. Honek `mbti_label`, `cnt` (kopurua) eta `mbti_index` izango ditu. Elasticsearch konektorea erabilita, Flink-ek taula honetako datuak Elasticsearch-en gordeko ditu, `count_per_personality` izeneko index batean.

```

CREATE TABLE count_per_personality (
mbti_label STRING,
cnt BIGINT,
mbti_index INT
) WITH (
'connector' = 'elasticsearch-7',
'hosts' = 'http://elasticsearch:9200',
'index' = 'count_per_personality',
'format' = 'json',
'sink.bulk-flush.max-actions' = '1'
);

```

Azkenik, hurrengo kontsulta exekutatzen da:

```

INSERT INTO count_per_personality
select mbti_personality, count(*) as cnt, pers_id
from txioak left join pertsonalitateak
on txioak.user_id = pertsonalitateak.id
group by (mbti_personality, pers_id);

```

Kontsulta honek txioak eta pertsonalitateak taulak elkartzen ditu erabiltzailearen IDaren bidez, eta pertsonalitate bakoitzeko zenbat txio dauden kontatzen du, emaitza Elasticsearch-era bidaliz.

Kibanan, index pattern bat sortzen da `count_per_personality` izenarekin. Gero, "Home" atalean "Visualize" aukera hautatzen da, eta "Create new visualization". Bertan, "Pie" grafikoa aukeratzen da eta sortutako indexa erabilita, pertsonalitate bakoitzeko txio kopurua bistaratzen da.

Horretarako "Metrics" atalean *Agregation: Sum* eta *Field: cnt* aukeratu behar dira. Bestalde "bucket" bat sortu behar da hurrengo konfigurazioarekin: *Agergation: Terms* , *Field: mbti\_label.keyword*, *Order by: Metric: Count*, eta *Size: 16*. Azkenik play botoi urdina sakatu behar da eta honek bistaraketa sortuko du.

## Soluzioaren funtzionamendua erakusten duten pantaila argazkiak

Grafiko honetan ikusi dezakegu Kibanak sortutako pertsonalitate desberdinen bistaraketa pie-chart baten moduan, non zati bakoitza pertsonalitate desberdinen kopurua adierazten duen

