DATU MASIBOEN PROZESAMENDURAKO AZPIEGITURAK

3.LANA

Andoni Sudupe eta Oier Ijurko

Aurkibidea

1. Sarrera

Lan honen helburu nagusia datu fluxuak prozesatzen dituen soluzio global bat lortzea da. Horretarako, ondorengo osagaiak elkarlanean jarriko ditugu:

1. MQTT, bai broker-ak eta baita bezeroak (igorlea eta harpideduna) ere.
2. Topic-etan antolatutako datu fluxuak biltegiratzeko Kafka erabili.
3. Datu fluxuak prozesatzeko Flink erabiltzea
4. Superset Impala-ra konektatu, bi taulen gainean kontsultak egin ahal izateko.
5. Elasticsearch (+Kibana) erabili datuak bistaratzeko grafikoki

2. Datuak

Erabili ditugun datuak, smartphone eta smartwatch aktibitatea eta biometrikak jasotzen dituen dataset batetik hartu ditugu. 51 pertsonei 3 minutuko 18 aktibitate egiteko eskatu zitzaien, eta mugikorra eta smartwatch-arekin giroskopioa eta azelerometroa erabiliz, hauen mugimenduak kontrolatu ziren. Datuak 20 Hz-ko maiztasunean gordeta daude eta denera 15,630,426 datu gordin.

Ezkerreko irudian aktibitate desberdinak agertzen dira eta eskuineko argazkian datu ilara bakoitzak izango duen formatua.

Activity	Code
Walking	A
Jogging	B
Stairs	C
Sitting	D
Standing	E
Typing	F
Brushing Teeth	G
Eating Soup	H
Eating Chips	I
Eating Pasta	J
Drinking from Cup	K
Eating Sandwich	L
Kicking (Soccer Ball)	M
Playing Catch w/Tennis Ball	O
Dribblinlg (Basketball)	P
Writing	Q
Clapping	R
Folding Clothes	S

Field name	Description
Subject-id	*Type: Symbolic numeric identifiifier.* Uniquely identifyies the subject. Range: 1600-1650.
Activity code	*Type: Symbolic single letter.* Identifies a specific activity as listed in Table 2. Range: A-S (no "N" value)
Timestamp	*Type: Integer.* Linux time
x	*Type Numeric: real.* Sensor value for x axis. May be positive or negative.
y	Same as x but for y axis
z	Same as x but for z axis

3. Erabili ditugun tresnak

Lan hau garatzeko hainbat tresna erabili ditugu. Hona hemen sarreran aipatutako helburua gauzatzeko beharrezkoak izango ditugun baliabideak:

3.1. Datuen jatorria: MQTT

MQTT, subscriber-publisher motako protokoloa da. Normalean, baliabide mugatuak (memoria edo banda-zabalera) dituzten gailuen artean mezuak trukatzeko erabiltzen
da protokolo hau. Bere abantailen artean, erabiltzeko oso erraza duen API bat duela dago.

3.2. Datuen biltegiratzea: Kafka

Apache Kafka gertaera sakabanatuko biltegia eta streaming bidez prozesatzeko open source plataforma bat da. Honen ideia, denbora errealeko datuekin lan egiteko sendoa, errendimendu handikoa eta latentzi baxukoa den plataforma bat eskaintzea da.

3.3. Datuen prozesamendua: Flink

Apache Flink fluxu-prozesamendurako erabiltzen den framework-a da. Eskala handiko datu-fluxuak prozesatzeko eta streaming aplikazioarekin prozesatutako datuei buruzko informazio analitikoa denbora errealean emateko erabiltzen da.

Lan honetan, Flink-eko SQL API-a erabiliko dugu taulak sortu eta Kafkan gordetako datuak ElasticSearch-era esportatzeko.

3.4. Datuak esportatu: ElasticSearch

Elasticsearch kode irekiko bilaketa- eta analisi-motor bat da, eta datuen azterketa errazten duena. Honen bitartez, kafkan gordeta ditugun datuak Kibana bezalako datu-bistaratzea ahalbidetuko duen aplikazioetara bidaltzeko ahalmena lortuko dugu.

3.5. Datuak bistaratu: Kibana

Elastic Stack-en gainean dagoen frontend-en aplikazioa da Kibana, eta datuak bistaratzeko eta Elasticsearch-en indexatutako datuak bilatzeko gaitasuna ematen du.

Honen bitartez, hasieran aipatutako datuak grafikoki bistaratzeko gai izango gara, dashboard bat sortuz lan honen amaieran.

4. Lana garatzeko prozesua

4.1. Compose eta Dockerfile

Egindako beste lanetan bezala, .yml fitxategietan adierazten dira lanean erabiliko diren kontainerrak. Kasu honetan, ondorengo kontainerrak izango ditugu: Mosquitto, Zookeeper, Kafka, Kafka-connect, ElasticSearch, Kibana, sql-client, jobmanager, taskmanager eta python kodea exekutatuko duen kontainer bat.

Mosquitto eta pythoneko kontainerra erabiliz, mqtt-client bat sortu eta broker eta portu bat erabiliz, deskargatutako datuak lortu ahal izango ditugu. Ondorengo atalean azalduko dugu hau zehaztasun handiagorekin.

Kafka eta Kafka-connect kontainerrekin, datuak gorde eta source eta sink-ak adierazteko gai izango gara. Modu honetan, MQTT-tik lortutako datuak gorde eta Flink-era eramateko gai izango gara. Horretarako, Kafka-connect-ek environment atalean dituen “connect” atalak erabiliko ditugu, baita deskargatutako jar-ak erabili ere.

Mosquitto eta Kafka arteko konexioa funtzionatu ahal izateko, json formatuan dagoen ondorengo connector-a erabili behar izan dugu:

```
{
    "name": "mqtt-source",
    "config": {
        "connector.class": "io.confluent.connect.mqtt.MqttSourceConnector",
        "tasks.max": 1,
        "mqtt.server.uri": "tcp://mosquitto:1883",
        "mqtt.topics": "smart",
        "kafka.topic": "smart",
        "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter",
        "confluent.topic.bootstrap.servers": "kafka:9092",
        "confluent.topic.replication.factor": 1
    }
}
```

Bertan, mosquitto-ren portua, Kafka-ren portua, topic-a eta bestelako informazioa adierazi dugu funtzionatu ahal izateko.

Zookeeper kontainerraren helburua zerbitzuen konfigurazioa eta sinkronizazioa ematea izango da. Normalean, Kafka, Zookeeper-rekin batera erabiltzen da.

Flink erabili ahal izateko, lehen aipatutako sql-client, jobmanager eta taskmanager kontainerrak erabiliko ditugu. Hauen bitartez, SQL API-a erabiliko dugu Kafkatik datuak hartu eta ondoren azalduko dugun elasticsearch-era esportatzeko. Datuak esportatzerakoan, Job berri bat sortuko da, dagokien Job ID-arekin.

Elasticsearch kontainerraren zeregina, Flink-en izango ditugun datuak esportatzea izango da, portu bat esleituz eta ondoren, Kibana bezalako aplikazio batean bistaratuz (honek ere bere kontainerra izango du, bere portuarekin).

Dockerfile-ari dagokionez, python script-ean client bat sortu ahal izateko erabiliko dugun paho.mqtt paketea instalatuko dugu, baita ondoren azalduko dugun .py-a exekutatu ere.

4.2. Python script-a

Datuak irakurri eta publikatzeko, python fitxategi bat sortuko dugu enuntziatuan emandako kodea abiapuntu hartuta. Hasieran, WISDM dataseta dagoen direktorio desberdinak definitzen dira, eta parametro batzuk irakurketa ongi egiteko.

```
import os, time
import random
from paho.mqtt import client as mqtt_client


rootpath = "/data/"
paths = ["raw/phone/accel/", "raw/phone/gyro/", "raw/watch/accel/", "raw/watch/gyro/"]
WAIT_TIME = 0.5
MAX_LINES = 1000000000 #connect mosquitto
i = 0
```

Ondoren, mqtt eta Kafka konektatzeko konektoreak martxan jartzeko komandoa eta sleep bat ditugu. Hau eskuz egin daiteke terminal batetik, baina horrela automatikoki egingo du. MQTT-kin publikatzeko beharrezko hasieraketak ere jarri ditugu, broker bat eta portu bat definitu ditugu bezeroa konektatzeko.

```
time.sleep(20)
#os.system('curl -d @/home/oier/Documents/kafka/config/connect-mqtt-source.json -H "Con
broker = 'mosquitto'
port = 1883
client = mqtt_client.Client()
client.connect(broker,port)
```

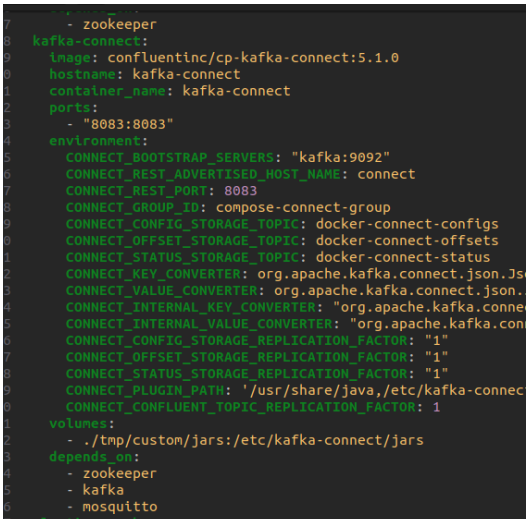
Honekin datuak irakurtzen hasi gaitezke. Fitxategi guztiak zeharkatuko ditugu eta fitxategiko ilara bakoitza hartu, json formatuko string bat sortu eta “smart” topikoan publikatuko dugu 0.5 segunduro . Honela, sorgailu programa bat sortu dugu, gure datuak stream prozesamendua egiteko moduan bidaliko ditu datuak broker batera smart gaiari lotuta.

```
for path in paths:
        data_files = sorted(os.listdir(rootpath+path))
        for data_file in data_files:
                with open(rootpath+path+data_file, "r") as f:
                        maxl = 0
                        while True:
                                try:
                                        line = f.readline()
                                except UnicodeDecodeError:
                                        continue
                                if not line or len(line) == 1:
                                        break
                                line = f.readline().split(",")
                                try:
                                        if line[-1][-1] == ";":
                                                line[-1] = line[-1][:-1]
                                        elif line[-1][-1] == "\n":
                                                line[-1] = line[-1][:-2]
                                except:
                                        continue
                                line = '{"usid":' + line[0] + ',"action": "' + line[1]
                                 #publish egin
                                time.sleep(WAIT_TIME)
                                result = client.publish("smart",line)
                                maxl += 1
                                if maxl > MAX_LINES:
                                        break
                                i += 1
```

Ondo publikatzen direla ikusteko, mosquitto bezero bat definitu dezakegu “smart” topikotik mezuak jasotzeko subscriber moduan, eta mezuak jasotzen diren ikusi.

4.3. Kafka connect eta datuak lortu

MQTT sorgailuak publikatutako datu fluxuak, Kafka kluster batean biltegiratuko dira. Hau egiteko, Kafka eta MQTT konektatzen dituen konektore bat definitu behar dugu, hau Kafka-connect izango da. Kontainer hau confluentinc/cp-Kafka-connect iruditik jaurtiko dugu, eta datu sarrera eta irteera egoki egiteko driverrak dituzten jar-ak dauden direktorioa definituko dugu. Honela, gure MQTT-Kafka konektorea gorde ahalko dugu eta Kafkan datuak jaso.



Komando honekin jar dezakegu martxan Kafka-mqtt, eta dena ondo badoa, lehenago azaldu den subscriberrean bezala, datuak agertzen dira. Honek esan nahi du, MQTT-tik Kafkara datuak ongi pasatzen direla.

4.4. Flink shell-eko deiak

Kafkan jasotako datua SQL bezala jaso eta tratatzeko Flink erabili dezakegu. “**`docker-compose exec sql-client ./sql-client.sh`**” eginez, Flinkeko Shell-era sar gaitezke. Bertan, gure datuen gainean SQL deiak egin ahal izango ditugu.

Kafkako datuak hemen sartu nahi baditugu, taula bat sortu beharko dugu, gure json moduko datuen zutabeak errespetatuz. Gure kasuan, taula honi LANA izena jarri diogu eta 6 zelai definituko dizkiogu, json-ak dituen bezala. Bestalde, WITH zatian, Kafkatik “smart” topikoan dauden datuak hartu nahi ditugula esango diogu, eta hauek json formatuan daudela.

```
Flink SQL> CREATE TABLE LANA (
>     usid BIGINT,
>     action STRING,
>     ts BIGINT,
>     x BIGINT,
>     y BIGINT,
>     z BIGINT
> ) WITH (
>     'connector' = 'kafka',  -- using kafka connector
>     'topic' = 'smart',  -- kafka topic
>     'scan.startup.mode' = 'earliest-offset',  -- reading from the beginning
>     'properties.bootstrap.servers' = 'kafka:9092',  -- kafka broker address
>     'format' = 'json'  -- the data format is json
> );
[INFO] Table has been created.
```

Lan honen azken zatia, datu hauek bistaratzea da, hau elasticsearchera esportatuta eta kibana erabilita lor dezakegu. Horretarako, interesgarriak zaizkigun datuak gordeko dituen taula bat sortu behar dugu eta hau elasticsearchera esportatu behar dugu.

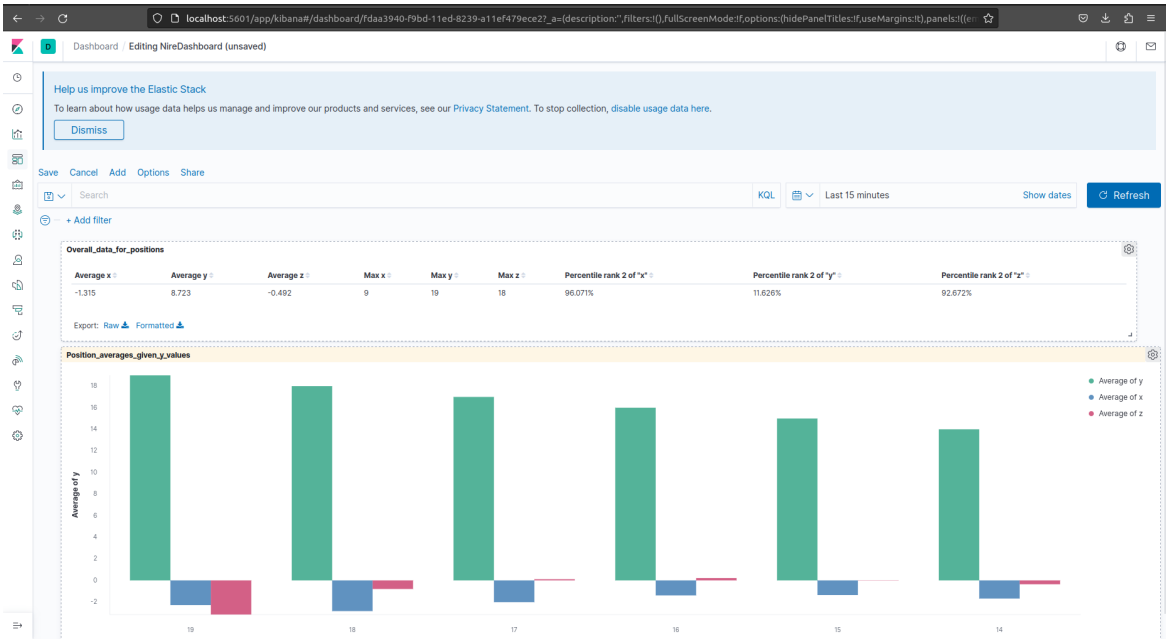
```
Flink SQL> CREATE TABLE datuak (x BIGINT, y BIGINT, z BIGINT) WITH ('connector' = 'elasticsearch-7',
>     'hosts' = 'http://elasticsearch:9200',
>     'index' = 'datuak');
[INFO] Table has been created.

Flink SQL> INSERT INTO datuak
> SELECT x,y,z
> FROM LANA;
[INFO] Submitting SQL update statement to the cluster...
[INFO] Table update statement has been successfully submitted to the cluster:
Job ID: b8f1a7e289fdeedbea9a1ba3161ec54f


Flink SQL>
```

Ondoren, taula honetara beste taulatik interesgarriak zaizkigun datuak txertatuko ditugu. Gure kasua, x, y eta z zatiak gordeko dituen taula bat dugu. Horrela, kibana bezalako aplikazio bat erabiliz, hauek bistaratu ahalko ditugu.

4.5. Kibanako Dashboard-a

Behin elasticsearch-era datuak esportatuta, datu horiek Kibanan bistaratzeko gai izango gara. Flink-eko shellean sortutako indizea zein den adieraziz Kibanan, datuen gainean deiak egingo ditugu.

Hona hemen sortutako dashboard-a gure datuekin:



Bertan, x, y eta z balioentzako batazbestekoak, balio maximoak eta bakoitzaren rank 2 pertzentilak atera ditugu. Bestalde, y balio desberdinetarako x eta z balioen fluktuazioa adierazten duen grafiko bat atera dugu ere bai.

5. Hobekuntza

5.1. Grafana erabiltzen saiatu

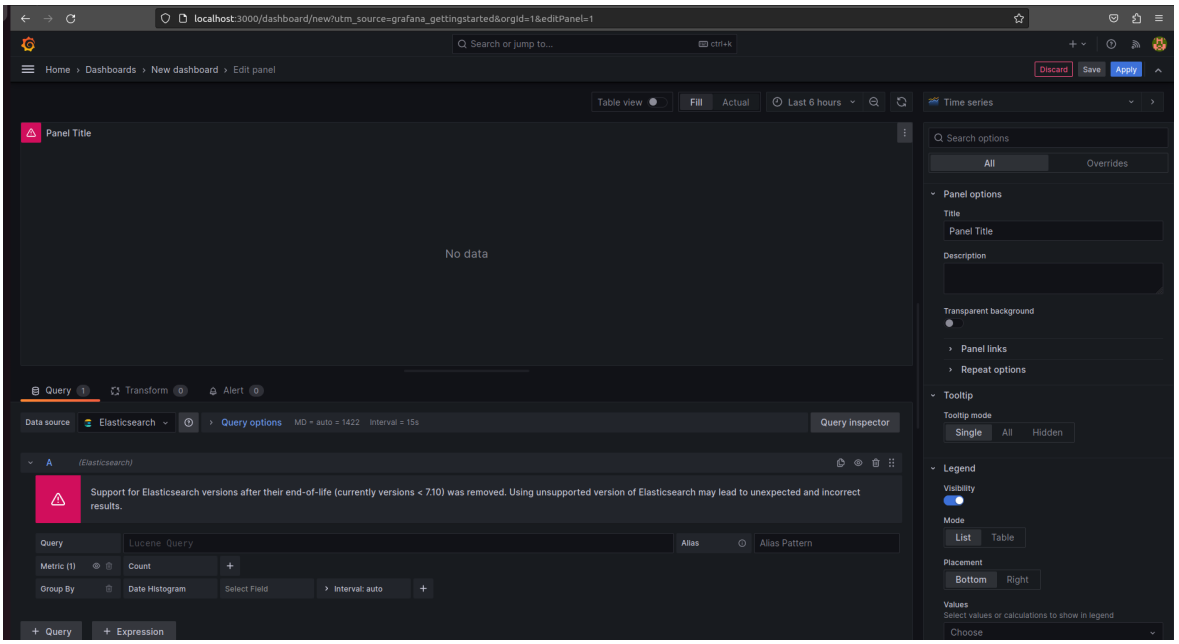
Elasticsearch-ekin erabili daitezkeen beste bistaratze-aplikazioak ikusiz, Grafanarekin aurkitu gara.

Kibana bezala, Grafanarentzat beste kontainer bat eginez docker compose fitxategian, eta bertan portu bat esleituko diogu sartu ahal izateko.

Kibanan ez bezala, Grafanan TimeStamp-a markatzen duen atributu bat izatea beharrezkoa zela ematen zuenez, lehenago egindako taulak (Flink-en) aldatu behar izan ditugu, ts atributua gehituz. Hona hemen erabilitako deia:

```
Flink SQL> INSERT INTO datuak
> SELECT x,y,z, HOUR(ts)
> FROM LANA;
[INFO] Submitting SQL update statement to the cluster...
[INFO] Table update statement has been successfully submitted to the cluster:
Job ID: 9ab8b50f12e9e842b9af537ac7a92c40
```

Taula egitea lortu badugu ere, ezin izan ditugu datu hauekin grafikoak lortu. Ondorengo irudian ikusten den bezala, erabiltzen ari garen elasticsearch-aren bertsioa ez dago sostendua Grafanarekin funtzionatzeko:



Bertsioarekin errore bat izan denez, txostenean sartzea erabaki dugu halere. Ez dugu bertsio desberdinekin probak egiteko denbora izan, baino Grafanarekin funtzionatu ahal izateko egin beharrezko pauso guztiak egin dira (Flink-eko taulak aldatu TimeStamp-a izan dadin, Compose fitxategian Grafana kontainerra gehitu...).

6. Ondorioak

Lan honetan, MQTT, Kafka, Flink, Elasticsearch, Kibana eta Grafana elkarlanean nola jartzen diren ikasi dugu, datu fluxuak prozesatzen dituen soluzio global bat lortuz. Broker-ak zer diren eta nola funtzionatzen duten ere ikasi dugu, teknologia desberdinen arteko konexioa nola egin ikasiz aldi berean.

Aipatzekoa da, compose.yml fitxategietatik bolumenak kenduta ditugunez, gure proiektuan docker up egiten den bakoitzean, Flink atalean sortutako taulak berriz ere sortu behar direla, Kafkarekin eta ElasticSearch-ekin berriro konektatuz.

7. Erreferentziak

- Gure proiektuaren github biltegia:
  https://github.com/oierIM/DMPA_MQTT-Kafka-Flink-ElasticSearch

- Kafka connect egiteko argibideak:
  https://www.baeldung.com/kafka-connect-mqtt-mongodb

- Kafka konektoreak lortzeko:
  https://www.confluent.io/hub/

- WISDM datasetaren informazioa:
  https://archive.ics.uci.edu/ml/datasets/WISDM+Smartphone+and+Smartwatch+Activity+and+Biometrics+Dataset+

- Flink erabiltzeko argibideak:
  https://flink.apache.org/2020/07/28/flink-sql-demo-building-an-end-to-end-streaming-application/