## 

## 

## 

Reporte de Proyecto Kafka

Estudiante: Inti Luna Avilés

Programa: Ingeniería de datos.

DATAHACK

## 

## 

## 

## 

## 

## 

## 

**2024-06-06**

## 

[**Resumen 3**](#_94ket1p0le7s)

[**Deployment 4**](#_ugacmd7guehn)

[**Detalle de proyecto 5**](#_igzw5wamawgr)

[**Diseño 5**](#_8jt7qob8g8fw)

[Diagramas 5](#_y9exx7mho3jb)

[Servicios usados al hacer deploy 5](#_kngzgbjkdohn)

[Consideraciones 5](#_iuz4yc3jx2yv)

[Ficheros y descripción 7](#_oh5bu27olb0m)

[Estructura 7](#_kukh967gb338)

[Descripción 8](#_d37l9fcf87cs)

[Flujo General 8](#_8aux9vep1ceb)

[Datos de entrada 8](#_e4y4s3lfb4yc)

[**Resultados 9**](#_y7zufp7qn8vm)

[Webapp con Flask 9](#_3siq89gqd7on)

[Análisis de sentimiento 9](#_gzxqxqackg54)

[Oportunidades de Mejoras (no ejecutadas en v1) 10](#_pheqai85dbco)

[Recursos consultados 10](#_mgp1b6r76qfk)

## 

## 

## 

## Resumen

Se ha creado una *data pipeline* para procesar datos de opiniones de un producto (maletas) utilizando kafka donde se leen datos desde una fuente (csv), se realiza análisis de sentimiento , se agrega la información de stream, se guarda en base de datos (mongodb) y por último se consumen datos desde una web app para mostrar tabla y gráfico de barras. A manera de facilitar el despliegue, se ha logrado crear fichero bash (.sh) para correr los subprocesos de python y abrir el navegador automáticamente con la URL del despliegue.

## Deployment

Solo se tiene que ejecutar fichero bash en directorio proyecto/code

>./run\_kafka\_project.sh

Este fichero:

- Instala paquetes de python segun requirements.txt

- Levanta servicios a partir de docker-compose.yml

- Verifica que el servidor ksqldb esté listo para funcionar

- Corre procesos de pipeline en segundo plano

- Inicia web app con Flask

- abre el navegador con URL apropiada

Para detener hay que:

>docker compose down

#Detener los procesos en segundo plano

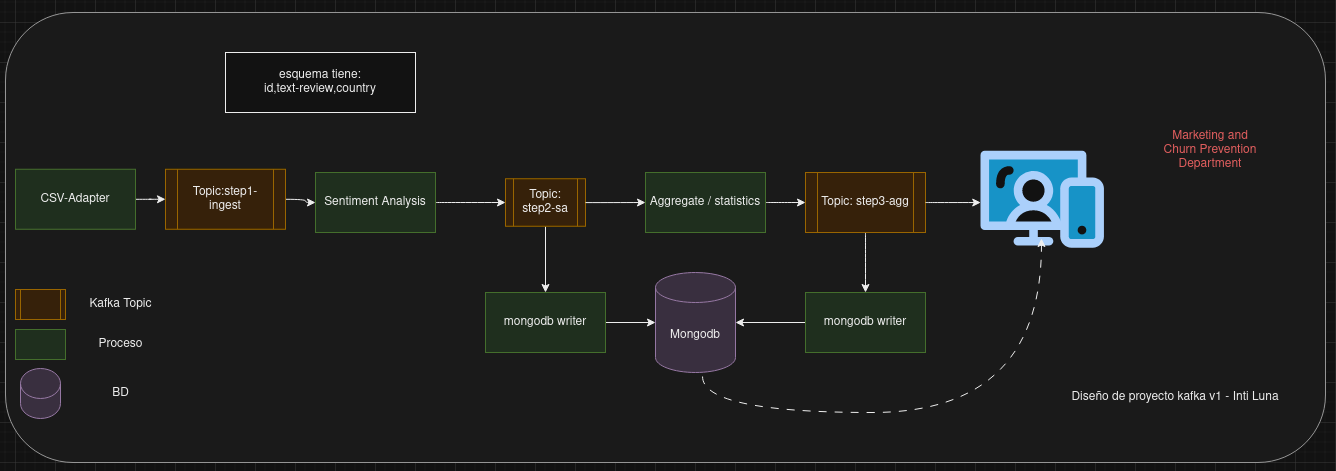
>./stop\_kafka\_project.sh

## 

## Detalle de proyecto

### Diseño

#### Diagramas

****

#### Servicios usados al hacer deploy

**Kafka broker**

**Ksqldb**

**Zookeeper**

**Control Center**

#### Consideraciones

**Source-Sink**

Se podría haber implementado kafka connect para leer datos de fichero csv y para escribir en base de datos (mongodb) con el plugin específico. Esta opción sería la recomendada para proyectos en producción por su fiabilidad sin embargo se requiere de más recursos. En este proyecto se decidió por un script específico dada la naturaleza del proyecto y la facilidad para ejecutarlo.

**Agregación**

Se ha utilizado ksqldb por su facilidad de construcción usando Ksql comparado con Kstreams usando java. Una herramienta muy útil para el desarrollo y prueba de las consultas fue la GUI de control center.

Otra manera de hacer agregaciones es hacer consultas directamente a la base de datos mongodb que contiene dos colecciones, los datos procesados de análisis de sentimiento junto a los datos originales y la colección “resumen” donde estan los datos agregados (en base a proceso de ksqldb).

**Servicio a End-User**

Se usó Flask porque me es familiar y porque es un framework minimalista y hay muchos plugins de extensiones.

### 

### Ficheros y descripción

#### Estructura

├── app.py

├── docker-compose.yml

├── flask.log

├── generate\_syntethic\_dataset.py

├── index\_steps.txt

├── input\_csv2.csv

├── \_\_pycache\_\_

│ └── app.cpython-311.pyc

├── requirements.txt

├── run\_kafka\_project.sh

├── static

│ ├── flujo\_v1.png

│ └── styles.css

├── step1.log

├── step1.py

├── step2.log

├── step2.py

├── step3\_db\_writer.log

├── step3\_db\_writer.py

├── step3\_ksqldb.log

├── step3\_ksqldb.py

├── step4\_save\_mongodb.log

├── step4\_save\_mongodb.py

├── stop\_kafka\_project.sh

├── templates

│ └── index.html

├── tests

│ ├── step3\_consumer.py

│ ├── step4\_consumer\_plot.py

│ ├── step4\_consumer.py

│ ├── test\_get\_mongodb\_data\_full.py

│ └── test\_get\_mongodb\_data\_resumen.py

└── verificar\_estado\_ksqldb\_server.py

#### Descripción

| **name** | **description** |
| --- | --- |
| generate\_syntethic\_dataset.py | Genera fichero sintético tipo opiniones buenas, malas y neutras. |
| [step1.py](http://step1.py) | ingesta data desde csv y crea topic "step1-ingest" |
| [step2.py](http://step2.py) | consume datos desde topic "step1-ingest", hace análisis de sentimiento y genera topic "step2-sa" |
| [step3\_db\_writer.py](http://step3_db_writer.py) | consume datos de "step2-sa" y guarda en Mongodb |
| [step3\_ksqldb.py](http://step3_ksqldb.py) | consume datos de "step2-sa" y genera tablas agregadas y stream |
| [step4\_save\_mongodb.py](http://step4_save_mongodb.py) | consume datos de "step3-agg" y guarda en Mongodb |

Adicionalmente se tiene scripts de pruebas en directorio tests. Estos scripts permiten hacer pruebas manuales y no se llegó a realizar unit tests automáticas con pytest por ejemplo.

### Flujo General

#### Datos de entrada

### 

### 

### Resultados

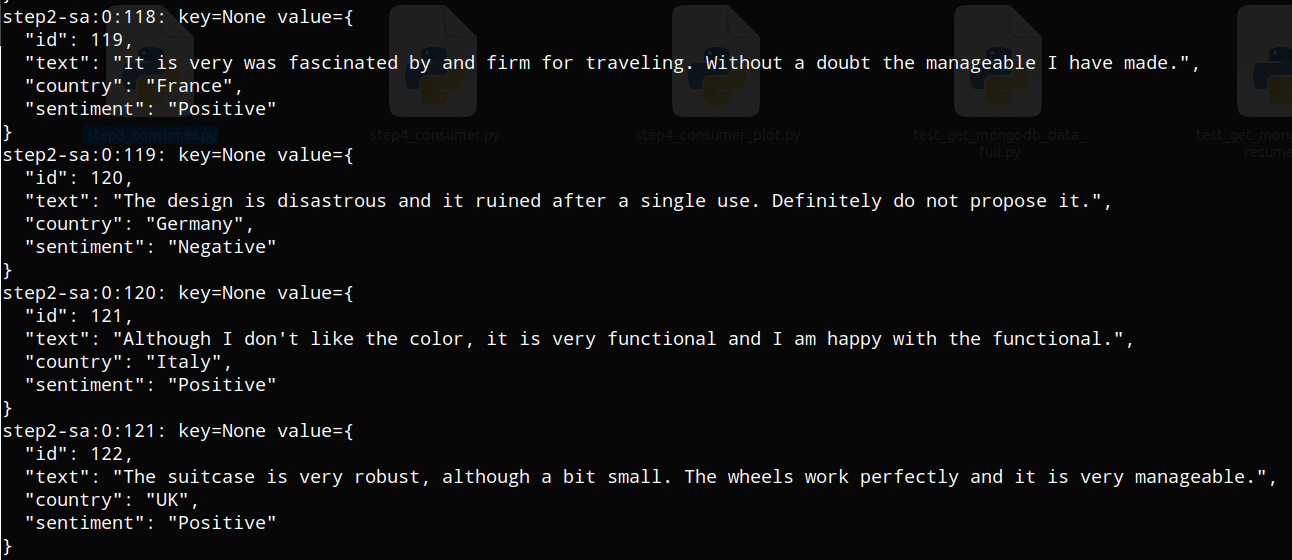
#### Webapp con Flask

(localhost:5000)

### 

#### Análisis de sentimiento

(Usando tests/step3\_consumer.py)



### 

### Oportunidades de Mejoras (no ejecutadas en v1)

* Integrar **Kafka Connect**
* Agregar datos según país.
* Encontrar otro mecanismo para generar plot de barras con python en Flask (actualmente se refresca automáticamente para generar nuevo plot).
* Implementar **Schema registry** que sería la herramienta a usar en un caso real para manejar las situaciones de cambio de esquemas en los tópicos.
* Integrar una aplicación de streaming de datos como *Reddit* o *Mastodon*
* Integrar diferentes flujos de proceso a partir de datos de topic, por ejemplo: filtrar datos según país para otros análisis y procesos (bajo la lógica que hay diferentes políticas de atención a clientes en cada región o país).
* Realizar pruebas de unit tests para mejorar la confiabilidad y mantenimiento del código.
* Agregar seguridad tanto a la web app como a cluster de kafka
* Dependiendo del caso de uso específico configurar brokers y topics para optimizar recursos, seguridad de operaciones y velocidad.

### Recursos consultados

**Bash**

<https://tecadmin.net/check-if-a-command-succeeded-in-bash/>

<https://unix.stackexchange.com/questions/74605/use-xdg-open-to-open-a-url-with-a-new-process>

<https://phoenixnap.com/kb/echo-command-linux>

**Kafka-Flask Integration**

<https://www.youtube.com/watch?v=hfi_ALPlsOQ>

<https://www.youtube.com/watch?v=MuLMxeI7Ytk&list=PL2UmzTIzxgL7Bq-mW--vtsM2YFF9GqhVB>

**Ksqldb**

<https://ksqldb.io/quickstart.html>

<https://developer.confluent.io/courses/ksqldb/intro/?_ga=2.105403331.1391371507.1717459641-1439070384.1715970528>

<https://docs.ksqldb.io/en/latest/developer-guide/api/>

<https://forum.confluent.io/t/using-ksqldb-rest-api-with-python/2732>

<https://github.com/bryanyang0528/ksql-python>