Infraestructura de Big Data

“EE2”

Grado en Ciencia de Datos e Inteligencia Artificial

03/05/2025

Integrantes:

- Néstor Rovira. nestor.rovira@alumnos.upm.es
- Boyuan Chen. b.chen@alumnos.upm.es
- Álvaro Huisman. alvaro.huisman@alumnos.upm.es
- Raffaele Cuzzaniti. raffaele.cuzzaniti@alumnos.upm.es
- Mario Fernández Blanco. m.fblanco@alumnos.upm.es

INDICE:

1. CARGA DE DATOS

En este trabajo se utilizaron dos conectores S3 Source Connector para extraer datos del bucket S3-MinIO y enviarlos al cluster de Kafka.

El primer conector, **source-safety**, se encargó de leer ficheros JSON relacionados con seguridad (fallos e incidentes), distribuyéndolos en los topics failures e incidents. Para separar los datos, se configuró una expresión regular en topic.regex.list, lo que permitió una clasificación automática. Se decidió no utilizar un esquema JSON formal, ya que se consideró que la validación adicional no aportaba ventajas prácticas relevantes en este caso.

```
CREATE SOURCE CONNECTOR "source-safety" WITH (
    "connector.class" = 'io.confluent.connect.s3.source.S3SourceConnector',
    "store.url" = 'https://izar.ls.fi.upm.es:30009/',
    "s3.bucket.name" = '4stations',
    "s3.region" = 'eu-west-2',
    "aws.access.key.id" = 'YA9JokyWUb2hFUbKYEEN',
    "aws.secret.access.key" = '0k2ornkQpVTqUrBbOEsEXOnBEEWgJf4AFQOU4O7Y',
    "format.class" = 'io.confluent.connect.s3.format.json.JsonFormat',
    "confluent.topic.bootstrap.servers" = 'kafka:9092',
    "confluent.topic.replication.factor" = '1',
    "value.converter" = 'org.apache.kafka.connect.json.JsonConverter',
    "value.converter.schemas.enable" = 'false',
    "topics.dir" = 'safety/',
    "topic.regex.list" = 'failures:failures/.*\\.json,incidents:incidents/.*\\.json',
    "tasks.max" = '1',
    "internal.kafka.reporter.bootstrap.servers" = 'kafka:9092'
);
```

El segundo conector, **source-traffic**, se utilizó para extraer datos de tráfico como llegadas, salidas, pasajeros y tickets, todos en formato Avro. En este caso, se configuró el uso de Schema Registry debido a que los datos presentaban una estructura definida y era importante garantizar su consistencia.

```
CREATE SOURCE CONNECTOR `source-traffic` WITH(
    "connector.class" = 'io.confluent.connect.s3.source.S3SourceConnector',
    "store.url" = 'https://izar.ls.fi.upm.es:30009',
    "s3.bucket.name" = '4stations',
    "s3.region" = 'eu-west-2',
    "aws.access.key.id" = 'YA9JokyWUb2hFUbKYEEN',
    "aws.secret.access.key" = '0k2ornkQpVTqUrBbOEsEXOnBEEWgJf4AFQOU4O7Y',
    "format.class" = 'io.confluent.connect.s3.format.avro.AvroFormat',
    "value.converter" = 'io.confluent.connect.avro.AvroConverter',
```

```
    "value.converter.schema.registry.url" = 'http://schema-registry:8081',
    "confluent.topic.bootstrap.servers" = 'kafka:9092',
    "confluent.topic.replication.factor" = '1',
    "topics.dir" = 'traffic/',
    "s3.partitioner.class" =  'io.confluent.connect.storage.partitioner.FieldPartitioner',
    "partition.field.name" = 'station',
    "topic.regex.list" =
's3arrivals:s3arrivals.*\\.avro,s3departures:s3departures.*\\.avro,s3passengers:s3passenger
s.*\\.avro,s3tickets:s3tickets.*\\.avro',
    "tasks.max" = '1');
```

Con esta configuración, se logró un flujo optimizado que permite manejar tanto datos semiestructurados como datos con esquema definido, adaptándose a las características propias de cada caso.

2. PROCESADO DE DATOS

2.1 SAFETY

STREAMS:

En la fase de procesado de datos mediante KSQL, se ha seguido la estructura propuesta en el enunciado para definir los streams correspondientes a los datos de safety.

En primer lugar, se han creado los streams raw_failures y raw_incidents sobre los tópicos failures e incidents, respectivamente. Estos streams permiten capturar directamente los datos crudos provenientes del conector, respetando la estructura original, que incluye campos como id, equipment, station, location, fts y duration para los fallos, y id, type, station, location, its, severity y duration para los incidentes.

Ejemplo de definición para raw_failures:

```
CREATE STREAM raw_failures (
```

```
 id VARCHAR,
 equipment VARCHAR,
 station VARCHAR,
 location VARCHAR,
 fts VARCHAR,
 duration DOUBLE)
WITH (kafka_topic='failures', value_format='JSON');
```

Posteriormente, se han definido los streams stg_failures y stg_incidents, que actúan como una capa de staging para preparar los datos antes de ser agregados en tablas. En stg_failures, se agrupan las claves equipment y station en una estructura STRUCT denominada FK, mientras que en stg_incidents se hace lo propio con los campos type y station. Además, se realiza un cast de los campos fts e its a formato TIMESTAMP para permitir un tratamiento temporal más preciso.

Ejemplo de transformación en stg_failures:

```
CREATE STREAM stg_failures
  WITH (
    KAFKA_TOPIC   = 'stg_failures',
    KEY_FORMAT    = 'AVRO',
    VALUE_FORMAT  = 'AVRO'
  ) AS
SELECT
  STRUCT(
    equipment := equipment,
    station   := station
  ) AS FK,
  location,
  CAST(fts AS TIMESTAMP)   AS fts,
  duration
FROM raw_failures
EMIT CHANGES;
```

Los datos resultantes se escriben en los tópicos intermedios stg_failures y stg_incidents en formato AVRO, lo que facilita su posterior procesamiento y asegura la compatibilidad con los esquemas de Kafka. Este diseño permite organizar los datos de forma consistente y alineada con lo especificado en el enunciado, garantizando que los pasos siguientes de agregación y análisis puedan ejecutarse de manera eficiente.

TABLAS:

Tras la definición de los streams de staging, se ha procedido a construir las tablas agregadas failures e incidents, que permiten resumir y analizar los datos de manera estructurada. Estas tablas cumplen con los requisitos descritos en el enunciado, donde se solicitaba calcular el número total de fallos e incidentes agrupados por estación, tipo y hora, además de obtener la media de duración de cada evento.

La tabla failures agrega los datos del stream stg_failures utilizando como claves de agrupación la estación (station), el tipo de equipo (equipment) y la hora (HH), que se obtiene transformando el campo de timestamp (fts) al formato de hora. En cada grupo se calcula el número de fallos (nf) y la duración media (df). El código implementado es el siguiente:

```
DROP TABLE IF EXISTS failures;

CREATE TABLE failures
  WITH (
    KAFKA_TOPIC    = 'stfail',
    KEY_FORMAT     = 'AVRO',
    VALUE_FORMAT   = 'AVRO'
  ) AS
SELECT
  FK->station       AS station,
  FK->equipment     AS equipment,
  FORMAT_TIMESTAMP(fts, 'HH')    AS hh,
  COUNT(*)          AS nf,
  AVG(duration)     AS df
FROM stg_failures
GROUP BY
  FK->station,
  FK->equipment,
  FORMAT_TIMESTAMP(fts, 'HH')
EMIT CHANGES;
```

Por su parte, la tabla incidents utiliza como claves la estación (station), el tipo de incidente (type) y la hora (HH), calculando el número total de incidentes (ni) y la duración media (di). El diseño sigue el mismo esquema lógico que failures, asegurando consistencia entre ambos procesos. El código correspondiente es:

```
DROP TABLE IF EXISTS incidents;

CREATE TABLE incidents
  WITH (
    KAFKA_TOPIC    = 'stincidents',
```

```
    KEY_FORMAT     = 'AVRO',
    VALUE_FORMAT   = 'AVRO'
  ) AS
SELECT
  FK->station       AS station,
  FK->type     AS type,
  FORMAT_TIMESTAMP(its, 'HH')    AS hh,
  COUNT(*)          AS ni,
  AVG(duration)     AS di
FROM stg_incidents
GROUP BY
  FK->station,
  FK->type,
  FORMAT_TIMESTAMP(its, 'HH')
EMIT CHANGES;
```

2.2 TRAFFIC

En la fase de procesado de datos del bloque de traffic, se han seguido las indicaciones del enunciado para definir los streams sobre los distintos tópicos provenientes del conector source-traffic.

STREAMS:

En primer lugar, se han creado los streams arrivals, departures, passengers y tickets sobre los tópicos s3arrivals, s3departures, s3passengers y s3tickets, respectivamente. Estos streams permiten capturar los datos originales provenientes de S3 en formato AVRO, respetando la estructura esperada, que incluye campos como aid, station, ats, passengers y dock_number en arrivals, o tid, train_id, station y validations en tickets, entre otros.

Ejemplo de definición para arrivals:

```
create stream `arrivals` (
 aid VARCHAR,
 station VARCHAR,
 ats TIMESTAMP,
 passengers INTEGER,
 dock_number INTEGER,
 wagon_count INTEGER,
 direction VARCHAR
```

```
 )
WITH (
 KAFKA_TOPIC='s3arrivals',
 VALUE_FORMAT='AVRO'
);
```

Posteriormente, se han definido los staging streams stg_arrivals, stg_departures, stg_passengers y stg_tickets, que permiten preparar los datos para un procesamiento posterior más controlado. A diferencia de los datos de safety, aquí no ha sido necesario realizar transformaciones adicionales ni cálculos intermedios: los staging streams se han diseñado principalmente para remitir los datos a tópicos intermedios, manteniendo la estructura original y homogeneizando el formato en AVRO.

Ejemplo de definición para stg_arrivals:

```
create stream `stg_arrivals` (
 aid VARCHAR,
 station VARCHAR,
 ats TIMESTAMP,
 passengers INTEGER,
 dock_number INTEGER,
 wagon_count INTEGER,
 direction VARCHAR
 )
WITH (
 KAFKA_TOPIC='stg_arrivals',
 PARTITIONS=1,
 VALUE_FORMAT='AVRO'
);
```

Una característica destacable de este bloque es que, a diferencia del bloque de safety, los datos de traffic ya vienen estructurados en formato AVRO desde el origen en S3. Esto ha simplificado el diseño de los staging streams, ya que no ha sido necesario realizar conversiones de formato ni agregar claves adicionales. Además, la separación explícita en tópicos individuales permite un procesamiento modular y escalable, facilitando posibles transformaciones específicas por tipo de evento sin interferencias entre flujos.

Para que el flujo de datos continúe a través de estos streams es necesario realizar las respectivas sentencias de INSERT, ya que no están conectados

directamente con los anteriores. Con el fin de poder garantizar que no existan valores nulos en los mensajes que se transmiten, se debería utilizar una sentencia como la que se muestra a continuación.

Ejemplo de INSERT para stg_arrivals:

```
INSERT INTO `stg_arrivals`
       SELECT *
       FROM `arrivals`
       WHERE aid IS NOT NULL
         AND station IS NOT NULL
         AND ats IS NOT NULL
         AND passengers IS NOT NULL
         AND dock_number IS NOT NULL
         AND wagon_count IS NOT NULL
         AND direction IS NOT NULL
       EMIT CHANGES;
```

3. VERIFICACIÓN (PRUEBAS UNITARIAS Y DE INTEGRACIÓN)

La verificación que se ha realizado ha consistido, principalmente, en la revisión y comprobación de los metadatos de todos los artefactos que se han creado, junto con el correcto funcionamiento de los conectores. A su vez, se ha comprobado que los mensajes devuelven los tipos de datos adecuados y su formato es el pedido.

Para comenzar se ha visto que todos los topics que eran necesarios para el trabajo han sido creados. Esto se puede comprobar en la siguiente captura:

Kafka Topic	Partitions	Partition Replicas
C_STORE	1	1
albsong1	1	1
albsong2	1	1
albsong3	1	1
connect-file-pulse-csv	1	1
connect-file-pulse-status	1	1
default_ksql_processing_log	1	1
failures	1	1
incidents	1	1
ksql-connect-configs	1	1
ksql-connect-offsets	25	1
ksql-connect-statuses	5	1
raw.customer	1	1
s3arrivals	1	1
s3departures	1	1
s3passengers	1	1
s3tickets	1	1
sak.customer	1	1
sch.customer	1	1
songs	1	1
stfail	1	1
stg_arrivals	1	1
stg_departures	1	1
stg_failures	1	1
stg_incidents	1	1
stg_passengers	1	1
stg_tickets	1	1
stincidents	1	1

De igual manera, se ha comprobado que las tareas de los conectores funcionan adecuadamente:

Connector Name	Type	Class	Status
sak_customer_sch	SOURCE	io.confluent.connect.jdbc.JdbcSourceConnector	RUNNING (1/1 tasks RUNNING)
source-safety	SOURCE	io.confluent.connect.s3.source.S3SourceConnector	RUNNING (1/1 tasks RUNNING)
raw_customer	SOURCE	io.confluent.connect.jdbc.JdbcSourceConnector	RUNNING (1/1 tasks RUNNING)
source-traffic	SOURCE	io.confluent.connect.s3.source.S3SourceConnector	RUNNING (1/1 tasks RUNNING)
sak_customer	SOURCE	io.confluent.connect.jdbc.JdbcSourceConnector	RUNNING (1/1 tasks RUNNING)
sink-jdbc-songs	SINK	io.confluent.connect.jdbc.JdbcSinkConnector	RUNNING (1/1 tasks RUNNING)
CONNECT_FILE_PULSE_CSV	SOURCE	io.streamthoughts.kafka.connect.filepulse.source.FilePulseSourceConnector	RUNNING (1/1 tasks RUNNING)

Una vez hechas estas comprobaciones iniciales, se ha probado si los mensajes atravesaban completamente el flujo:

```
ksql> select * from `arrivals`;
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
|AID              |STATION          |ATS              |PASSENGERS       |DOCK_NUMBER      |WAGON_COUNT      |DIRECTION        |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
|arr1742986146    |Station D        |2025-03-26T10:49:|14               |4                |9                |East             |
|                 |                 |25.000           |                 |                 |                 |                 |
|arr1742986147    |Station D        |2025-03-26T10:49:|17               |4                |9                |West             |
|                 |                 |47.000           |                 |                 |                 |                 |
|arr1742378128    |Station A        |2025-03-19T11:01:|18               |2                |6                |East             |
|                 |                 |16.000           |                 |                 |                 |                 |
|arr1745921103    |Station B        |2025-04-29T10:05:|17               |5                |4                |South            |
|                 |                 |02.000           |                 |                 |                 |                 |
|arr1742378130    |Station C        |2025-03-19T11:14:|16               |3                |9                |North            |
|                 |                 |35.000           |                 |                 |                 |                 |
|arr1745921105    |Station D        |2025-04-29T10:40:|20               |5                |9                |East             |
|                 |                 |32.000           |                 |                 |                 |                 |
|arr1742378129    |Station A        |2025-03-19T11:12:|15               |5                |7                |East             |
|                 |                 |03.000           |                 |                 |                 |                 |
|arr1742378131    |Station B        |2025-03-19T11:19:|25               |4                |6                |East             |
|                 |                 |16.000           |                 |                 |                 |                 |
|arr1745921108    |Station C        |2025-04-29T11:18:|46               |4                |7                |North            |
|                 |                 |22.000           |                 |                 |                 |                 |
|arr1742378132    |Station D        |2025-03-19T11:19:|16               |5                |6                |East             |
|                 |                 |16.000           |                 |                 |                 |                 |
|arr1742986148    |Station A        |2025-03-26T10:50:|20               |4                |4                |East             |
```

Esta última prueba ha servido a su vez para comprobar que los mensajes siguen la estructura deseada, aunque para confirmarlo es necesario comprobar que los metadatos de los streams y tablas son iguales que aquellos que se especifican en el enunciado.
Como muestra de ello se adjuntan imágenes de los metadatos de los siguientes streams y tablas:

Stg_arrivals

```
ksql> describe `stg_arrivals` extended;

Name                 : stg_arrivals
Type                 : STREAM
Timestamp field      : Not set - using <ROWTIME>
Key format           : KAFKA
Value format         : AVRO
Kafka topic          : stg_arrivals (partitions: 1, replication: 1)
Statement            : CREATE STREAM `stg_arrivals` (AID STRING, STATION STRING, ATS TIMESTAMP, PASSENGERS INTEGER, DOCK_NUMBER INTEGER, W
AGON_COUNT INTEGER, DIRECTION STRING) WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='stg_arrivals', KEY_FORMAT='KAFKA', PARTITIONS=1, VALUE_F
ORMAT='AVRO');

 Field       | Type
-------------------------------
 AID         | VARCHAR(STRING)
 STATION     | VARCHAR(STRING)
 ATS         | TIMESTAMP
 PASSENGERS  | INTEGER
 DOCK_NUMBER | INTEGER
 WAGON_COUNT | INTEGER
 DIRECTION   | VARCHAR(STRING)
-------------------------------
```

Stg_failures

```
ksql> describe stg_failures extended;

Name                 : STG_FAILURES
Type                 : STREAM
Timestamp field      : Not set - using <ROWTIME>
Key format           : AVRO
Value format         : AVRO
Kafka topic          : stg_failures (partitions: 1, replication: 1)
Statement            : CREATE STREAM STG_FAILURES WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='stg_failures', KEY_FORMAT='AVRO', PARTITIONS
=1, REPLICAS=1, RETENTION_MS=604800000, VALUE_FORMAT='AVRO') AS SELECT
  STRUCT(EQUIPMENT:=RAW_FAILURES.EQUIPMENT, STATION:=RAW_FAILURES.STATION) FK,
  RAW_FAILURES.LOCATION LOCATION,
  CAST(RAW_FAILURES.FTS AS TIMESTAMP) FTS,
  RAW_FAILURES.DURATION DURATION
FROM RAW_FAILURES RAW_FAILURES
EMIT CHANGES;

 Field    | Type
------------------------------------------------------------------------
 FK       | STRUCT<EQUIPMENT VARCHAR(STRING), STATION VARCHAR(STRING)>
 LOCATION | VARCHAR(STRING)
 FTS      | TIMESTAMP
 DURATION | DOUBLE
------------------------------------------------------------------------
```

Failures (tabla)

```
ksql> describe failures extended;

Name                 : FAILURES
Type                 : TABLE
Timestamp field      : Not set - using <ROWTIME>
Key format           : AVRO
Value format         : AVRO
Kafka topic          : stfail (partitions: 1, replication: 1)
Statement            : CREATE TABLE FAILURES WITH (CLEANUP_POLICY='compact', KAFKA_TOPIC='stfail', KEY_FORMAT='AVRO', PARTITIONS=1, REPLICAS=1, RETENTION_MS
=604800000, VALUE_FORMAT='AVRO') AS SELECT
  STG_FAILURES.FK->STATION STATION,
  STG_FAILURES.FK->EQUIPMENT EQUIPMENT,
  FORMAT_TIMESTAMP(STG_FAILURES.FTS, 'HH') HH,
  COUNT(*) NF,
  AVG(STG_FAILURES.DURATION) DF
FROM STG_FAILURES STG_FAILURES
GROUP BY STG_FAILURES.FK->STATION, STG_FAILURES.FK->EQUIPMENT, FORMAT_TIMESTAMP(STG_FAILURES.FTS, 'HH')
EMIT CHANGES;

 Field     | Type
-------------------------------------------
 STATION   | VARCHAR(STRING)  (primary key)
 EQUIPMENT | VARCHAR(STRING)  (primary key)
 HH        | VARCHAR(STRING)  (primary key)
 NF        | BIGINT
 DF        | DOUBLE
-------------------------------------------
```

Como se puede comprobar, los metadatos son iguales que aquellos que se proporcionaron en el enunciado, y siguen las indicaciones sobre el tipado y estructurado de datos que se solicitó.

Además, se ha probado a realizar algunas queries a las tablas finales de la sección de safety para asegurar que el funcionamiento era el correcto.

Ejemplo de query de `failures`:

```
ksql> SELECT * FROM `failures` WHERE station = 'Station A' AND hh = '11';
+-------------+-------------+-------------+-------------+-------------+
|STATION      |EQUIPMENT    |HH           |NF           |DF           |
+-------------+-------------+-------------+-------------+-------------+
|Station A    |beacon       |11           |2            |29.4758622642|
|             |             |             |             |6005         |
|Station A    |signal box   |11           |1            |82.3581155777|
|             |             |             |             |5575         |
|Station A    |power failure|11           |1            |13.9134659577|
|             |             |             |             |99353        |
Query terminated
```