

# مشروع تحويل التفاعل

## اللحظي للمستخدمين

Completed on

Jan 10, 2026

Prepared by

Khaled Alruwita

### 1) الفكرة

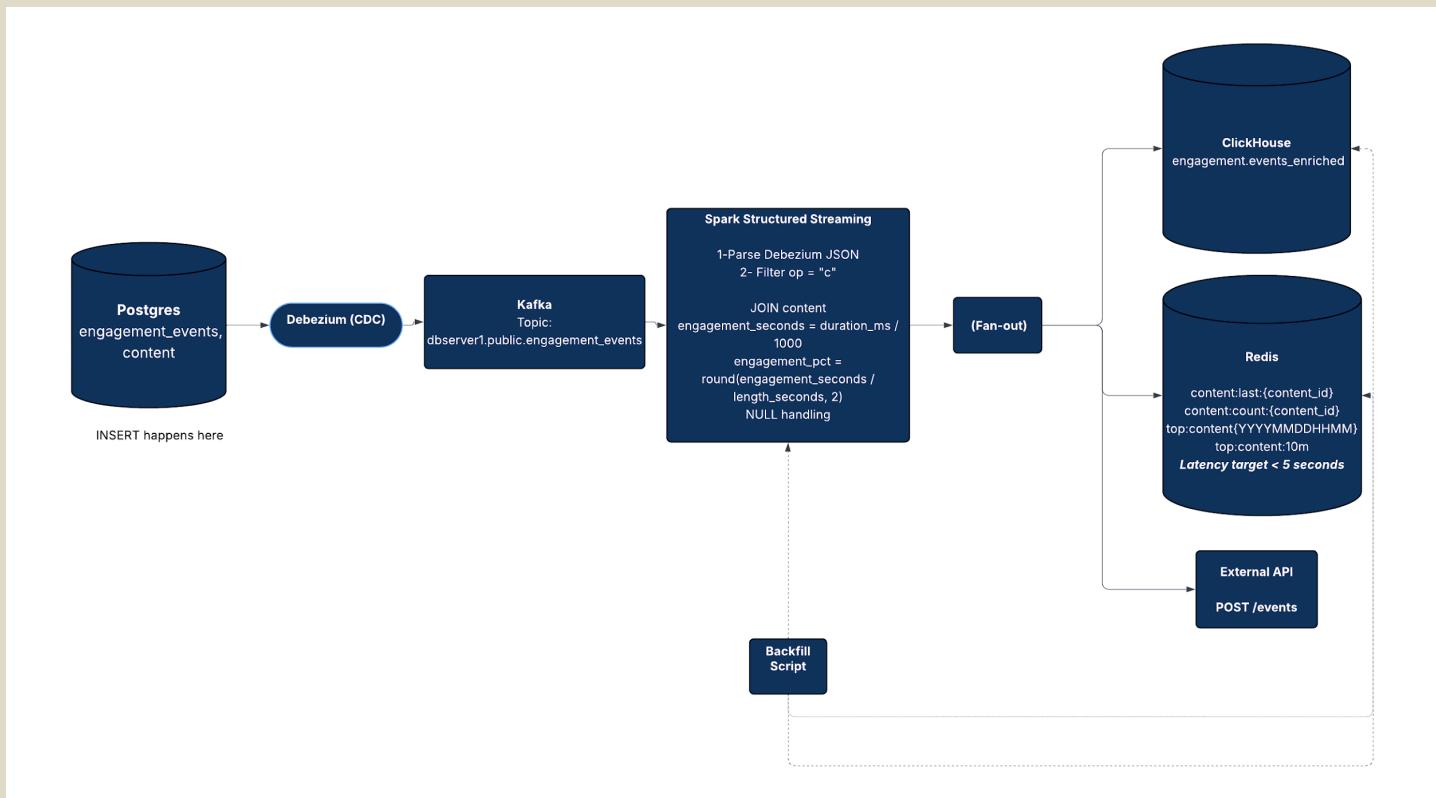
بناء نظام بث بيانات (Streaming) في Postgres يلقط الصفوف الجديدة من جدول engagement\_events في جدول content، ثم يرسل الحدث المُثري مباشرة إلى ثلاثة وجهات: ويحول كل حدث عبر ربطه بجدول Redis، ثم يرسل الحدث المُثري مباشرة إلى ثلاثة وجهات:

1. قاعدة بيانات عمودية للتحليلات (BigQuery) بديل ClickHouse
2. Redis ك “الأكثر تفاعلاً آخر 10 دقائق”
3. نظام خارجي (External API) لا نتحكم به

### 2) اختيار ال Stack

- polling من Debezium: CDC
- ناقل رسائل يفصل المصدر عن المعالجة: Kafka
- إطار تحقيق.streaming مع checkpointing (PySpark): دقة عالية + fan-out
- قاعدة عمودية محلية للتحليلات: ClickHouse

## 3) مسار البيانات



### Stage 0 – Source Tables (PostgreSQL)

Tool: PostgreSQL

Tables:

- `engagement_events` (fact table)
- `content` (dimension table)

(INSERT):

```

INSERT INTO engagement_events
(content_id, user_id, event_type, duration_ms, device)
VALUES (1, 777, 'play', 7000, 'web');
  
```

This represents a user interaction event.

---

## Stage 2 – Change Data Capture (CDC)

Tool: Debezium PostgreSQL Connector

- Debezium reads PostgreSQL
- Every INSERT into `engagement_events` is captured as a change event.

Output: JSON message sent to Kafka.

---

## Stage 2— Message Transport (Kafka)

Tool: Apache Kafka

Topic:

`dbserver1.public.engagement_events`

- Buffers events
- Guarantees ordering within partitions
- Allows replay (offset-based recovery)

Example Debezium message

```
{  
  "before": null,  
  "after": {  
    "id": 16,  
    "content_id": 1,  
    "user_id": 999,  
    "event_type": "play",  
    "event_ts": "2026-01-08T14:21:52Z",  
    "duration_ms": 7000,  
    "device": "web"  
  },
```

```
    "op": "c"
}
```

Spark only consumes **after** records where **op** = "c" (create).

---

## Stage 3 – Stream Ingestion

Tool: Spark Structured Streaming (PySpark)

- Spark reads Kafka as a streaming source.
  - Parses Debezium JSON.
  - Filters only **INSERT** operations.
  - Converts the stream into a structured DataFrame.
- 

## Stage 4 – Enrichment & Transformation

Tool: Spark (joins + derived columns)

Join:

```
engagement_events.content_id = content.content_id
```

Derived Fields:

- **content\_type** ← from **content**
- **length\_seconds** ← from **content**
- **engagement\_seconds** = **duration\_ms** / 1000
- **engagement\_pct** = round(**engagement\_seconds** / **length\_seconds**, 2)

Null Handling Rule:

- If `duration_ms` or `length_seconds` is NULL → engagement\_seconds and engagement\_pct are set to NULL.

Example enriched record:

```
{
  "event_id": 16,
  "content_id": 1,
  "user_id": 999,
  "event_type": "play",
  "event_ts": "2026-01-08 14:21:52",
  "duration_ms": 7000,
  "device": "web",
  "content_type": "video",
  "length_seconds": 300,
  "engagement_seconds": 7.0,
  "engagement_pct": 0.02
}
```

## 4. Multi-Sink Fan-out

Spark uses `foreachBatch` to fan out the same enriched data to three destinations in parallel.

---

Sink A — ClickHouse (Analytics Store [Column Based])

Table:

`engagement.events_enriched`

Write mode: Append

Example query:

```
SELECT content_id, avg(engagement_pct)
FROM engagement.events_enriched
GROUP BY content_id;
```

---

Sink B — Redis (Real-Time Layer)

Role: Low-latency serving & aggregation

1. Latest event per content

```
Key: content:last:{content_id}
Type: String (JSON)
TTL: ~1 hour
```

## 2. Total event counter per content

```
Key: content:count:{content_id}
Type: Integer
```

## 3. Per-minute aggregation buckets

```
Key: top:content:bucket:{YYYYMMDDHHMM}
Type: ZSET
Member: content_id
Score: number of events in that minute
TTL: 15 minutes
```

## 4. Last 10 minutes aggregation

```
Key: top:content:10m
Type: ZSET
Built by merging last 10 minute buckets
TTL: 120 seconds
```

### Latency Guarantee:

Redis updates arrive within < 5 seconds of event creation.

---

## Sink C — External System (HTTP API)

Tool: FastAPI (mock external service)

Endpoint:

```
POST /events
```

## 5. Failure Handling

### Spark Checkpointing

- Offsets and state stored in checkpoint directories
- Job resumes safely after failure

### Idempotency (Non-Transactional Sinks)

- Redis keys include `event_id`
- Duplicate events are ignored
- Provides exactly-once effect

## External API Failures

Two strategies:

- Fail-fast: Spark retries the batch
- Best-effort: Log failures and continue

(Current implementation demonstrates best-effort behavior.)

---

## 6. Backfill (Historical Reprocessing)

Tool: `backfill.py`

What it does:

- Reads historical data from Postgres by time range
- Applies the same enrichment logic
- Writes to ClickHouse + Redis
- Uses idempotency keys to avoid duplicates

Usage:

```
python backfill.py --start 2026-01-08T09:00:00 --end 2026-01-08T10:00:00
```

# ٤) تجربة البايبلайн

## ١. الكونكتر

```

khaled@LudenMateBook:~/engagement-pipeline$ docker exec -it spark-client bash -lc "
/opt/spark/bin/spark-submit \
--master spark://spark-master:7077 \
--conf spark.sql.shuffle.partitions=4 \
--conf spark.jars.ivy=/opt/checkpoints/ivy \
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1,org.postgresql:postgresql:42.7.4,com.clickhouse:clickhouse-jdbc:0.6.5 \
/opt/spark-apps/02_fanout_clickhouse_redis_external.py
"
:: loading settings :: url = jar:file:/opt/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /opt/checkpoints/ivy/cache
The jars for the packages stored in: /opt/checkpoints/ivy/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
org.postgresql#postgresql added as a dependency
com.clickhouse#clickhouse-jdbc added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-85247529-4e33-4d92-8151-5eb0ee3de9b2;1.0
  confs: [default]
    found org.apache.spark#spark-sql-kafka-0-10_2.12;3.5.1 in central
    found org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.5.1 in central
    found org.apache.kafka#kafka-clients;3.4.1 in central
    found org.lz4#lz4-java;1.8.0 in central
    found org.xerial.snappy#snappy-java;1.1.10.3 in central
    found org.slf4j#${slf4j-api};2.0.7 in central
    found org.apache.hadoop#hadoop-client-runtime;3.3.4 in central
    found org.apache.hadoop#hadoop-client-api;3.3.4 in central
    found commons-logging#commons-logging;1.1.3 in central
    found com.google.code.findbugs#jsr305;3.0.0 in central
    found org.apache.commons#commons-pool2;2.11.1 in central
    found org.postgresql#postgresql;42.7.4 in central
    found org.checkerframework#checker-qual;3.42.0 in central

```

بعد تنفيذ هذا الأمر، يبدأ Spark في:

- تحميل الاعتمادات
- تهيئة بيئة التشغيل
- تشغيل بريديوسر مستمر (Streaming)

بمجرد ما يصير اي Hit لافتت جديد راح يتحول للتحويلات من ثم فان اوت لـ 3 كنس يوم رز

## External API جل.2

```

khaled@LudenMateBook:~/engagement-pipeline$ ls
clickhouse  docker-compose.yml  external-api  initdb      tools
connectors  engagement-pipeline  external-mock  spark-apps
khaled@LudenMateBook:~/engagement-pipeline$ docker logs -f external-api
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     172.20.0.1:42920 - "POST /events HTTP/1.1" 200 OK
INFO:     172.20.0.1:49126 - "POST /events HTTP/1.1" 200 OK

```

في هذه الخطوة:

- شغلنا النظام الخارجي باستخدام حاوية external-api
- تابعنا السجلات عبر docker logs -f للتأكد أن الخدمة تعمل بشكل صحيح.

النتيجة:

- لا API اشتغل على المنفذ 8000.
- Spark بدأ يرسل له الأحداث المُتزايدة.
- ظهور رسائل POST /events 200 OK يؤكد أن البيانات وصلت بنجاح للنظام الخارجي.

## 3.3.1. INSERT

```
khaled@LudenMateBook:~/engagement-pipeline$ docker exec -it pg psql -U app -d engagement -c "INSERT INTO engagement_events (content_id, user_id, event_type, duration_ms, device) VALUES (1, 999, 'play', 7000, 'web');"
INSERT 0 1
khaled@LudenMateBook:~/engagement-pipeline$ |
```

- قمنا بإدخال حدث جديد يدوياً في جدول engagement\_events داخل PostgreSQL.
- هذا الإدخال يمثل مصدر البيانات (ال input ) للنظام كامل.

النتيجة:

- السطر تم إدخاله بنجاح ( INSERT 0 1 ).
- بمجرد الإدخال، يبدأ البابيللين تلقائياً بالتقاط الحدث وبّه لبقية المراحل ( Kafka → Spark ) → باقي الأنظمة).

## 4. التأكيد من وصول البيانات لأول كونسيومر (ClickHouse) قاعدة البيانات العمودية

```
khaled@LudenMateBook:~/engagement-pipeline$ docker exec -it clickhouse clickhouse-client -q "
SELECT event_id, content_id, user_id, event_type, engagement_seconds, engagement_pct
FROM engagement.events_enriched
ORDER BY event_id DESC
LIMIT 5;"
```

event_id	content_id	user_id	event_type	engagement_seconds	engagement_pct
19	1	777	play	7	0.02
18	1	999	play	7	0.02
17	1	12345	play	7	0.02
16	1	999	play	7	0.02
15	1	999	play	7	0.02

في هذه الخطوة:

- استعلممنا من جدول events\_enriched داخل ClickHouse.
- تأكدنا أن الحدث اللي دخلناه تم إثراؤه بنجاح (تحويل المدة للثواني + حساب نسبة التفاعل).

النتيجة:

- ظهور السجلات الجديدة مباشرة.
- القيم مثل  $engagement\_pct = 0.02$  و  $engagement\_seconds = 7$  محسوبة صح.
- هذا يثبت أن مسار Kafka → Spark → ClickHouse شغال بشكل صحيح.

## 5. التأكد من وصول بيانات التحديث الفوري Redis

```
khaled@LudenMateBook:~/engagement-pipeline$ docker exec -it redis redis-cli GET content:last
docker exec -it redis redis-cli GET content:count:1
"{"event_id": 19, "content_id": 1, "user_id": 777, "event_type": "play", "event_ts": "2026-01-09 14:04:31.163702", "duration_ms": 7000, "device": "web", "content_type": "video", "length_seconds": 300, "engagement_seconds": 7.0, "engagement_pct": 0.02}"
"13"
khaled@LudenMateBook:~/engagement-pipeline$ |
```

في هذه الخطوة:

- استعلمنا من جدول events\_enriched داخلي ClickHouse تأكيناً أن الحدث اللي دخلناه تم إثراوه بنجاح (تحويل المدة للثواني + حساب نسبة التفاعل).

:النتيجة

- ظهور السجلات الجديدة مباشرة.
- القيم مثل  $engagement\_pct = 0.02$  و  $engagement\_seconds = 7$  محسوبة صح.
- هذا يثبت أن مسار Kafka → Spark → ClickHouse شغال بشكل صحيح.

## 6. عرض المحتويات الأكثر تفاعلاً آخر 10 دقائق

```
khaled@LudenMateBook:~/engagement-pipeline$ docker exec -it redis redis-cli ZREVRANGE top:content:10m 0 10 WITHSCORES
1) "1"
2) "4"
khaled@LudenMateBook:~/engagement-pipeline$ |
```

في هذه الخطوة:

- استخدمنا Redis في ZREVRANGE على المفتاح top:content:10m
- هذا المفتاح يمثل تجميع آخر 10 دقائق مبني على Buckets زمنية.

:النتيجة

- الرقم "1" هو `.content_id`
- الرقم "4" هو مجموع التفاعلات (score) خلال آخر 10 دقائق.
- الترتيب تناظري، يعني هذا المحتوى هو الأعلى تفاعلاً حالياً.

## 7. تنفيذ الـ Backfill

```

INFO:     172.20.0.9:33898 - POST /events HTTP/1.1 200 OK
^Ckhaled@LudenMateBook:~/engagement-pipeline$ docker run --rm -it \
--network engagement-pipeline_default \
-v "$PWD/tools:/app" \
-w /app \
python:3.12-slim \
bash -lc "pip install -q psycopg2-binary redis clickhouse-connect && python backfill.py --start 2026-01-08T09:00:00 --end 2026-01-08T10:00:00"
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.

[notice] A new release of pip is available: 25.0.1 > 25.3
[notice] To update, run: pip install --upgrade pip
Backfill done. inserted=8
khaled@LudenMateBook:~/engagement-pipeline$ 

```

في هذه الخطوة:

- شُغّلنا حاوية Python مؤقتة (`docker run --rm`) داخل نفس شبكة المشروع.
- ثبّتّنا المكتبات المطلوبة (`(psycopg2, redis, clickhouse-connect)`).
- شُغّلنا سكريبت `backfill.py` مع تحديد فترة زمنية معينة.

النتيجة:

- تمّت إعادة معالجة البيانات التاريخية بين الوقتين المحددين.
- السطر `Backfill done. inserted=8` يعني أنه تم إدخال 8 أحداث قديمة بنجاح إلى النظام.

## 5) الخلاصة

العمل هذا المشروع تجربة عملية ممتعة بالذات إنها تقنيات لأول مره اتعامل معها مثل CDC باستخدام Debezium .Spark Structured Streaming وفائدته الفعلية، والتعامل مع Multi-sink fan-out في Redis والتجميع الزمني في التكليف كان تحدياً مفيداً وساعدني على تعميق فهمي لكيفية تصميم أنظمة Streaming بشكل عام

## 6) تحسينات ممكن اضافتها

- إضافة مراقبة (Monitoring Dashboard) (Monitoring Dashboard).
- تحسين إدارة الأخطاء وإعادة المحاولة (Retry & DLQ) خاصة مع النظام الخارجي.
- إضافة اختبارات تحميل (Load Testing) لمحاكاة ضغط إنتاجي أعلى.
- تصميم Front end للنظام

## 7) ملف الكود بـ Git

<https://github.com/KhaledAlruwita/engagement-pipeline.git>