Activity 3

configuration of spark terminal code

change to the spark call

```
spark-submit \
  --master spark://spark-master:7077 \
  --packages org.apache.spark:spark-sql-kafka-0-10_2.13:4.0.0 \
  --num-executors 2 \
  --executor-cores 2 \
  --executor-memory 2G \
  /opt/spark-apps/spark_crash_event_monitor.py
```

Output

```
-------------------------------------------
Batch: 0
-------------------------------------------
+-----------------------------------------+---------+-----------+
|Interval                                 |user_id  |crash_count|
+-----------------------------------------+---------+-----------+
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1211|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1365|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1370|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1603|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1872|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1761|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1765|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1221|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1301|3          |
|{2026-01-20 19:59:40, 2026-01-20 19:59:50}|user_1384|3          |
+-----------------------------------------+---------+-----------+
```

Technical Discussion

Architecture

The solution impliments a Spark Structured Streaming close to what we learned and what I saw in the spark_structured_streaming_logs_processing.py It processes event logs from Kafka in near real-time. THe architecture follows a standard streaming pipeling:

Kafka Source → Parse JSON → Filter Crashes → Window Aggregation → Alert Output

Requirement satisfaction

Code explaination

```
analysis_df = (
    raw_df.select(from_json(col("value").cast("string"), schema).alias("data"))
    .select("data.*")
    .withColumn("event_time", to_timestamp(from_unixtime(col("timestamp") /
1000)))
    .filter(
        (lower(col("content")).contains("crash")) &
        (lower(col("severity")).isin("High", "Critical"))
    )
    .withWatermark("event_time", "2 minutes")
    .groupBy(
        window(col("event_time"), "10 seconds"),
        col("user_id")
    )
    .agg(count("*").alias("crash_count"))
    .filter(col("crash_count") > 2)
    .select(
        col("window").alias("Interval"),
        col("user_id"),
        col("crash_count")
    )
)
```

firstly I wanna discuss the .filter part

I decided to use the lower function to gurantee myself the matching here (for the content and the severity) just to be fully sure. Here I am filtering by the columns where content contains crash and where we have in the severity eather "High" or "Critical" standing, like we are supposed to do. I used the isin operator as this does string matching and I was able to match to this or that easily.

next up I also have a .groupBy part Where I Group events by the event_time (the 10 seconds) and by the column for user.

with my groupBy I already seperated groups for each unique user all 10 minutes, now i aggregate by count("*") to count all the number of crash events per user and per window and give it a name "crash_count" to work with it.

the withColumn part ensures that I use the time based on the event_time not the system time as far as I understand it. We have the raw timestamp (if that divided by 1000 we have it in seconds), then I can use two functions to convert it into a proper Tinestamp.

I also ensured to only let it output when the crash_count is over 2 with my second filtering.

I also implimented a Late event arrival (its the WithWatermark) and decided 2 Minutes waiting is a good time, everything that comes after the 2 minutes will get dropped. 2 Minutes is like 20% of the window, so I thought this might be a good compromise, its not the longest wait but it should get a lot of late Data in.

```
query = (
    analysis_df.writeStream
    .outputMode("update")
    .format("console")
    .option("truncate", "false")
    .start()
)
```

Now this is my query that shows our results from the windows. outputMode("update") outputs only our new or updated windows (new can happen if we don't get any updates in time). The output gets written out in the console, and we get the full content, without cutting with truncate false.

Spark reasoning why it is slower

My spark configuration is that i only have one kafka partition but I have more running for the Spark client, as this was my understanding of how it worked, which I now learned is less performative, but due to lack of time I will not change this now.

Non-Functional Requirements

Scalability - Horizontal Scaling Support.

As I did this already with my updated docker/spark code where I already have it running on multiple spark workers, it does scale horizontally and works. I am gonna be honest, I will not try to run it on any higher settings, as I do not want my laptop to burn.

Due to what we learned in class, we know that the structure of Kafka and Spark provides a Horizontal scaling for us, therefore it will work. Basically because of the frameworks we are using, we do not have to be worried about it, as they provide it already.

Apache Spark and Apache Kafka are both designed as distributed systems, which natively support horizontal scaling. Spark distributes computation across multiple worker nodes using executors, allowing workloads to be parallelized automatically as more workers are added to the cluster. Kafka supports horizontal scaling through partitioned topics and distributed brokers, enabling producers and consumers to scale out by increasing the number of partitions and brokers without changing application logic.

Fault Tolerance

I do not know how to fully check this by myself, but I do know that this again is provided by the framework we use and therefore its already provided automatically and I do not have to worry about it.

Fault tolerance in Spark is provided through RDD lineage, task re-execution, and data replication, allowing lost computations and data partitions to be automatically recomputed if a worker node fails. Kafka provides fault tolerance through replication of partitions across brokers and leader election, ensuring that if a broker fails, another replica can take over with no data loss and minimal service interruption.

Late Arrivals

As I already specified in my Code explaination, I handle late arrivings that I will accept any arrivals up to 2 minutes late and anything that arrives after that will be dropped. I think this is a fair timeframe as I give it a 20% grace period of the window time and 2 Minutes is not that long to wait, if it is more case sensitive and we need to get the data earlier I think we could cut this period down a bit.

Performance and scalability

<no name>	RUNNING	1b23ef0c-1517-4b33-bc3d-db994921d63e	986fbe95-12d6-42ec-a914-7422804a18c4	2026/01/21 11:51:30	3 minutes 45 seconds	37539.69	42642.37	16