Phan Anh Tuan

Nov 2025

Job Application Tracking

Data Engineer Project Report

tuandte27@gmail.com 0943377126 https://github.com/tuandte27

Introduction

Background

This project is a near real-time (5-10s interval) recruitment campaign performance tracking system. Its primary function is to measure the instantaneous efficiency of job campaigns across various publishers and job IDs.

By capturing and processing key funnel metrics—such as clicks, spends, applications, and conversions—with minimal latency, the system allows analysts to monitor and immediately optimize bidding strategies (bid_set) and campaign health in a truly dynamic environment.

Data Source

Dataset Information

This dataset is the main performance log for the near real-time recruitment campaign tracking system. Each record shows the performance of a specific campaign at a certain time (by date and hour). Data is often grouped by Job ID, Campaign ID (campaign_id), and Publisher ID.
The dataset includes:

- Identifiers: Key fields like job_id, company_id, campaign_id, and publisher_id help analyze performance across different jobs, campaigns, and publishers.
- Cost & Action Metrics: Metrics like clicks, spends, and bid_set (bidding price) are used to calculate efficiency measures such as Cost Per Click (CPC).
- Funnel Events: Tracks candidate progress through disqualified_application, qualified_application, and the final goal, conversion.

The main purpose of this data is to provide fast, up-to-date insights so that automated systems and analysts can quickly optimize campaigns and adjust budgets based on the latest performance.

Data Flow

Tools

This project leverages tools such as **Cassandra**, **Apache Spark**, **MySQL**, **Grafana**, and **Docker** to manage data ingestion, processing, storage, and near real-time visualization in a scalable and efficient manner.

- Cassandra serves as a data lake for storing raw, high-velocity data.
- Apache Spark performs ETL processing and transforms data for analysis.
- MySQL acts as a data warehouse for structured storage and querying.
- Grafana provides near real-time dashboards and visualizations.
- Docker containerizes the system for easy deployment and orchestration.

ETL Flow

The ETL pipeline in this project follows a near real-time **CDC (Change Data Capture)** approach to ensure fresh data is continuously processed and loaded into the warehouse. It consists of three main stages:

Extract

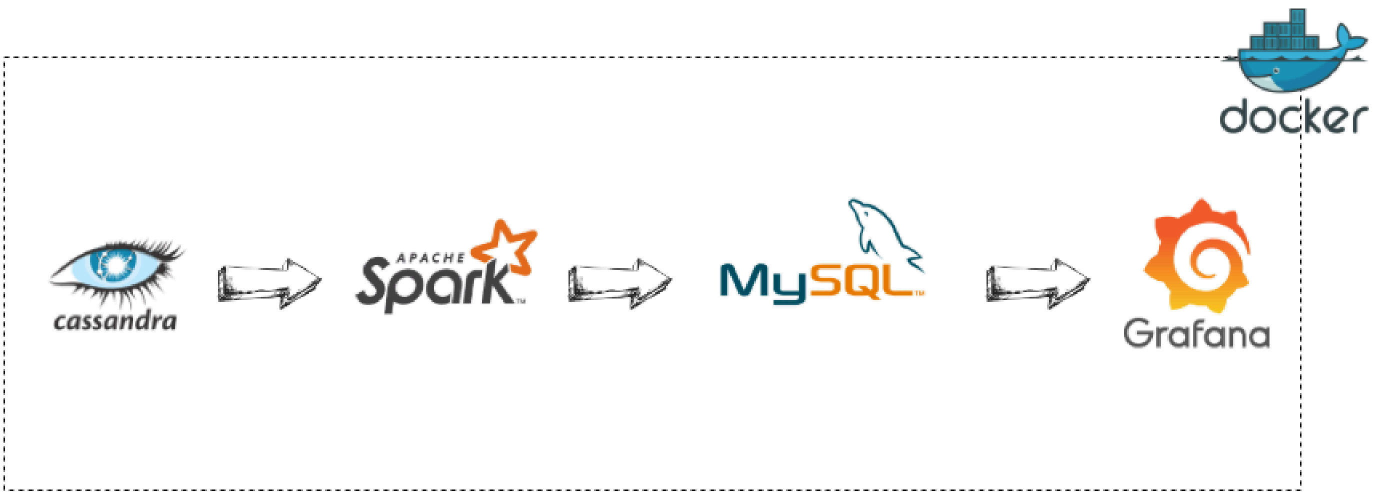
- Data is extracted from Cassandra, which acts as the data lake storing raw event logs.
- Using Spark, only new or updated records since the last successful load in MySQL are retrieved, based on the timestamp (ts).

Transform

- Spark processes the raw event logs by filtering, aggregating, and computing key metrics such as clicks, qualified/unqualified applications, and conversions.
- Data is cleaned (e.g., filling nulls), grouped by relevant dimensions (job_id, campaign_id, publisher_id, group_id, date, hour), and enriched by joining with company information from MySQL.
- The transformed output represents a structured, analytics-ready dataset.

Load

- The final dataset is loaded into MySQL, the data warehouse, in append mode.
- Each record includes aggregated metrics and timestamps, supporting near real-time reporting.
- This enables downstream dashboards (e.g., Grafana) to visualize up-to-date performance metrics without delay.



Output

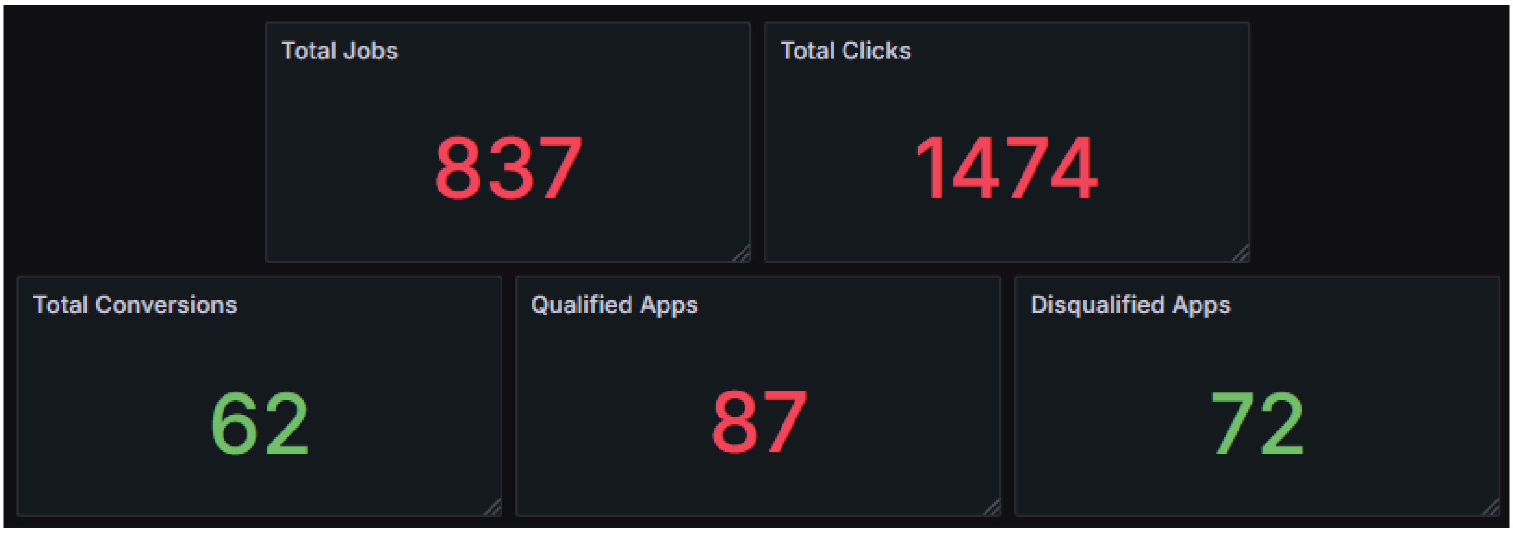
Schema

Column	Type	Description
id	INT	Surrogate Key. A unique, auto-generated ID for the record (internal tracking).
job_id	INT	The unique identifier for the job or recruitment posting.
dates	DATE	The date on which this performance data was recorded.
hours	INT	The hour block during which this performance data was recorded.
disqualified_application	INT	The number of applications that were disqualified/did not meet the minimum requirements.
qualified-application	INT	The number of applications that were deemed qualified after initial screening.
conversion	INT	The number of candidates that resulted in a final conversion (e.g., successful hire).
company_id	INT	The unique identifier of the company owning the job posting.
group_id	INT	The identifier for the group or department related to the job posting (often used for classification or aggregation).
campaign_id	INT	The unique identifier for the recruitment advertising/marketing campaign.
publisher_id	INT	The identifier for the advertising vendor or source (e.g., Google Ads, Job Board A).
bid_set	DOUBLE	The amount of money paid for each click generated by the campaign or job posting.

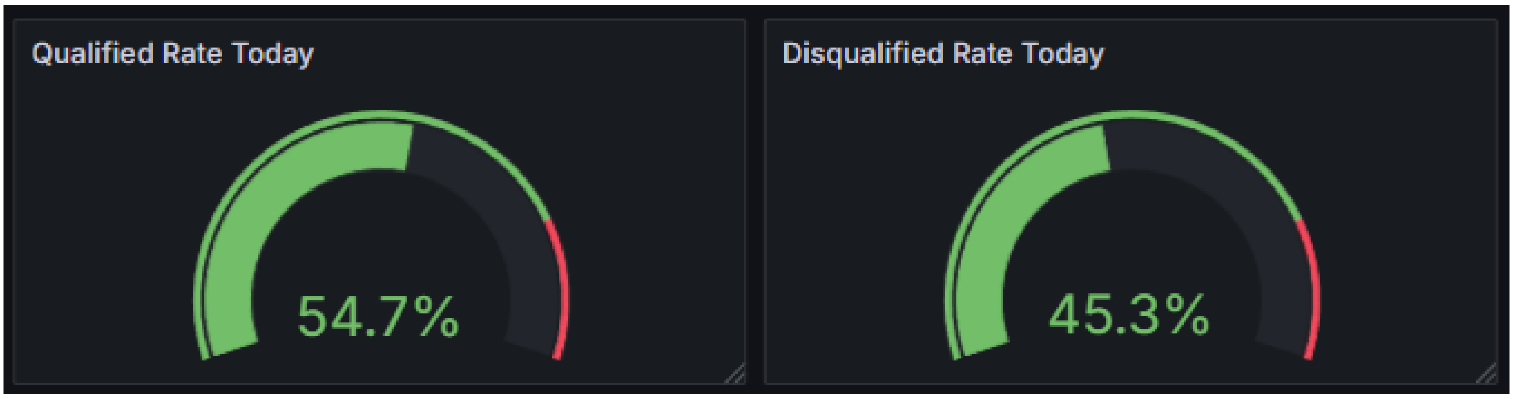
clicks	INT	The total number of times users clicked on the ad or job posting.
spend_hour	DOUBLE	The total cost spent on the campaign or job posting in 1 hour.
sources	VARCHAR	The technical origin of the data (e.g., Cassandra, Log File).
updated_at	TIMESTAMP	The timestamp indicating the last time this record was updated in the system.

Visualizations

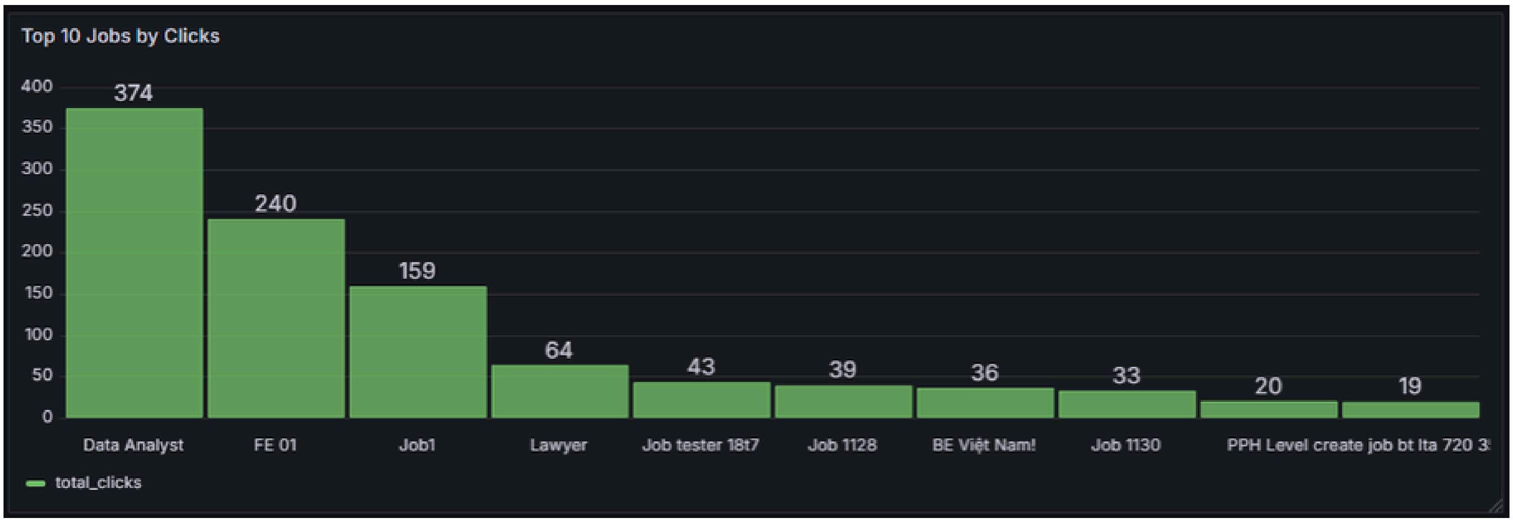
Charts



This section of the dashboard shows 5 key stats: total Jobs, total Clicks, total Conversions, total Qualified Applications, and total Disqualified Applications. These metrics provide a quick overview of recruitment scale, candidate engagement, campaign effectiveness, candidate quality, and help businesses optimize recruitment processes, allocate resources efficiently, and improve campaign performance.

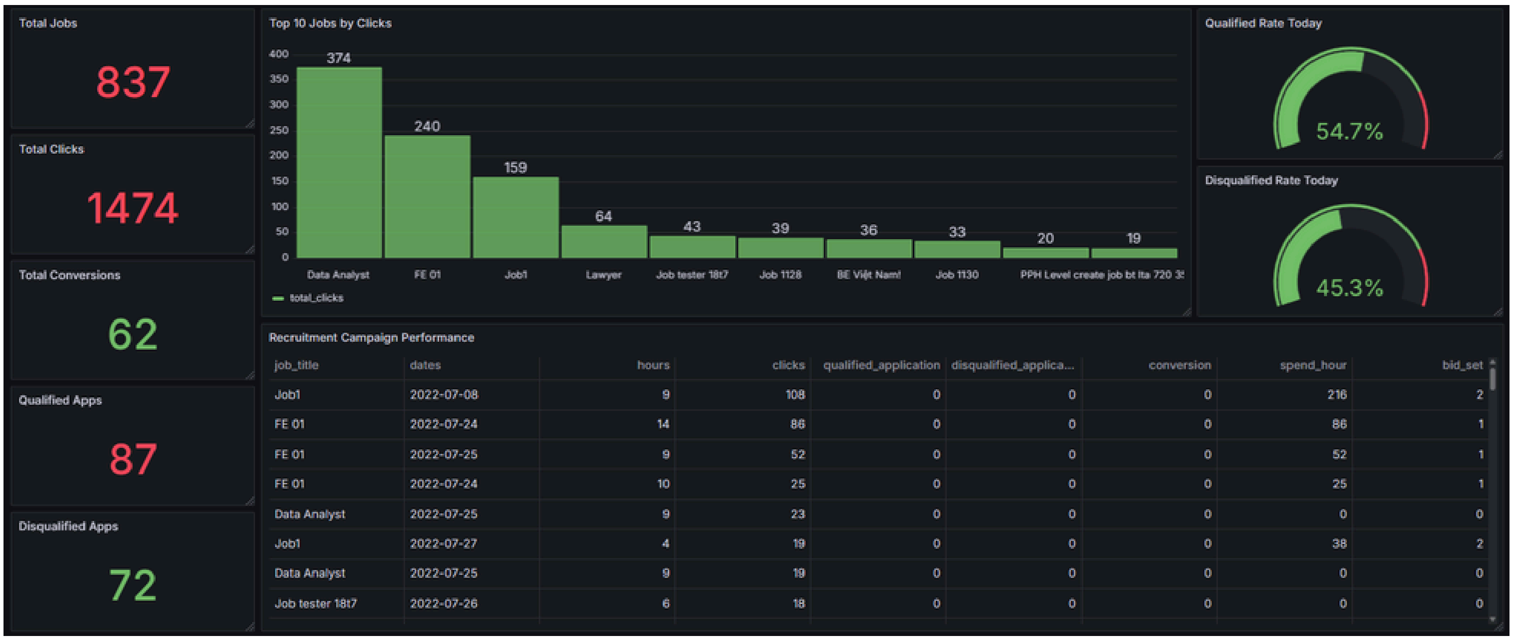


These two gauges display the daily percentage of Qualified and Disqualified applicants. These metrics allow businesses to quickly assess the quality of the candidate pool, monitor daily recruitment campaign effectiveness, optimize the screening process, and adjust recruitment channels or job postings to improve candidate quality, saving time and cost.



This chart displays the Top 10 jobs with the highest number of clicks from applicants. It highlights which job positions attract the most attention, helping businesses assess the appeal of each role, optimize job descriptions and recruitment campaigns, and allocate recruitment resources more effectively.

Dashboard



Total Jobs
837

Total Clicks
1474

Total Conversions
62

Qualified Apps
87

Disqualified Apps
72

Qualified Rate Today
54.7%

Disqualified Rate Today
45.3%

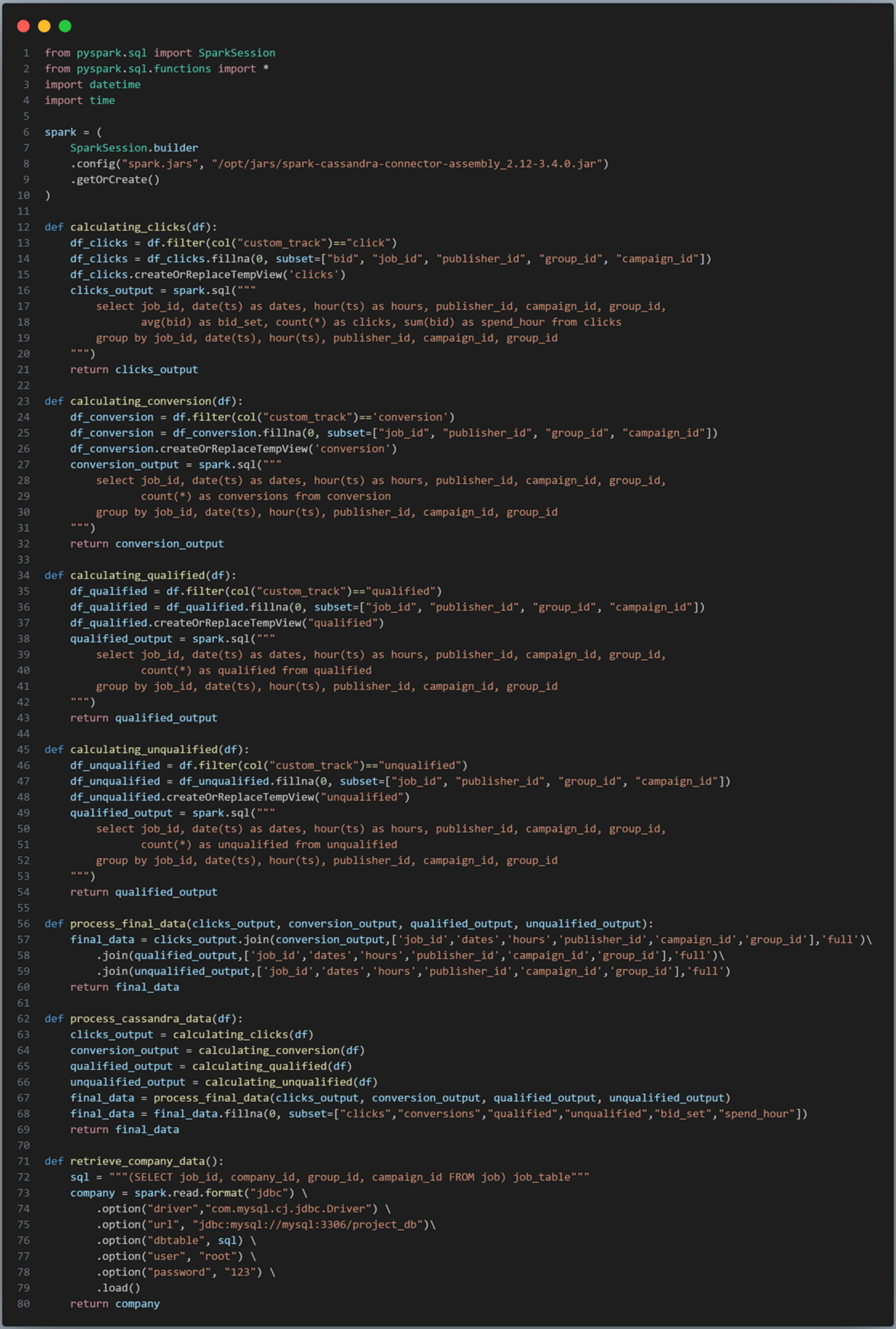
Recruitment Campaign Performance

job_title	dates	hours	clicks	qualified_application	disqualified_applica...	conversion	spend_hour	bid_set
Job1	2022-07-08	9	108	0	0	0	216	2
FE 01	2022-07-24	14	86	0	0	0	86	1
FE 01	2022-07-25	9	52	0	0	0	52	1
FE 01	2022-07-24	10	25	0	0	0	25	1
Data Analyst	2022-07-25	9	23	0	0	0	0	0
Job1	2022-07-27	4	19	0	0	0	38	2
Data Analyst	2022-07-25	9	19	0	0	0	0	0
Job tester 18t7	2022-07-26	6	18	0	0	0	0	0

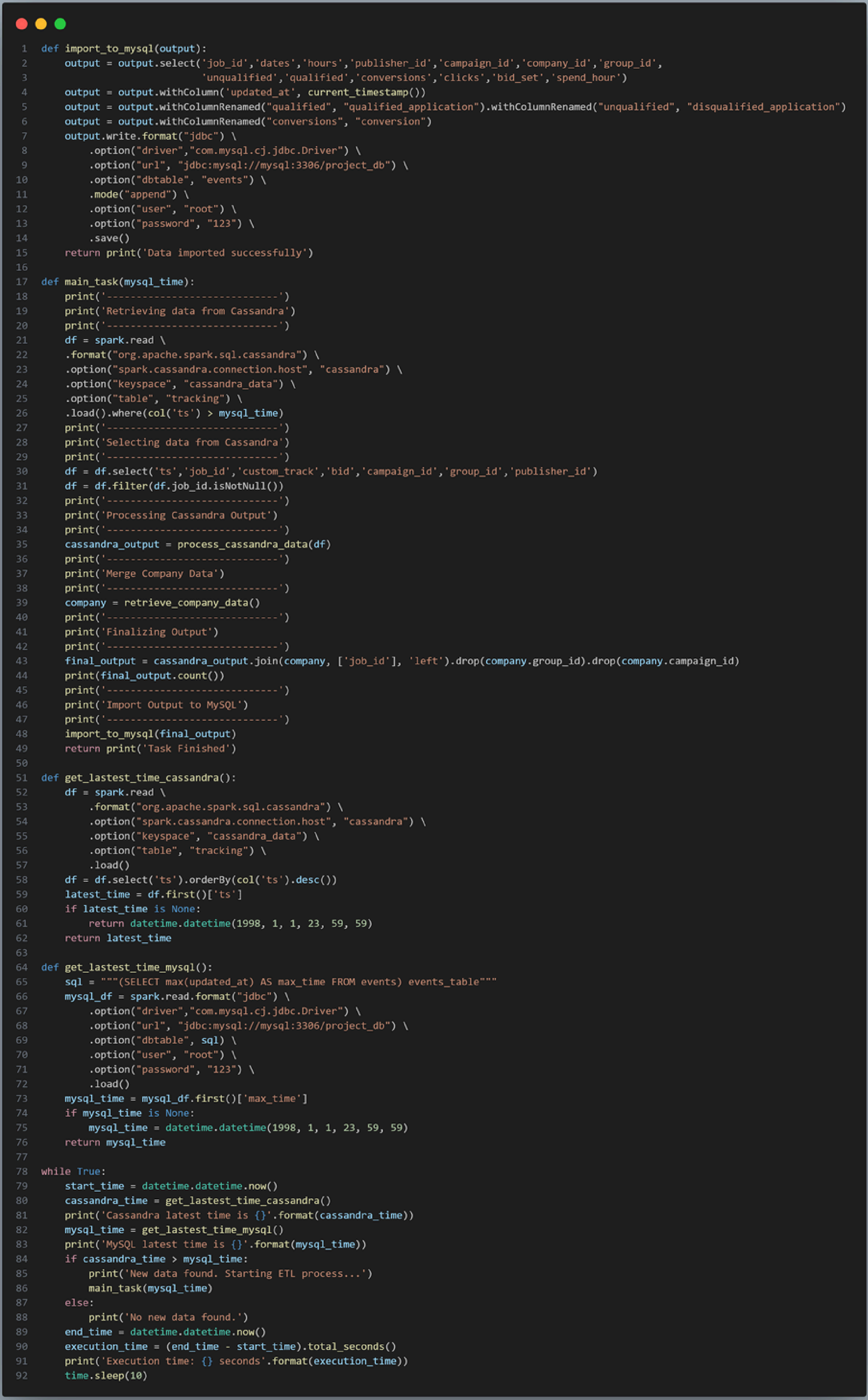
The entire dashboard is built on **Grafana**, providing a comprehensive and visual overview of recruitment activities. It aggregates key metrics such as total Jobs, Clicks, Conversions, and the daily Qualified and Disqualified rates, along with detailed charts like the top 10 jobs with the highest clicks.

A key feature of this dashboard is its **real-time data capability**, updating instantly as data changes, allowing businesses to monitor campaign effectiveness, assess candidate quality, and optimize recruitment processes, enabling timely and informed decision-making.

Python Code



```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import datetime
import time

spark = (
    SparkSession.builder
    .config("spark.jars", "/opt/jars/spark-cassandra-connector-assembly_2.12-3.4.0.jar")
    .getOrCreate()
)

def calculating_clicks(df):
    df_clicks = df.filter(col("custom_track")=="click")
    df_clicks = df_clicks.fillna(0, subset=["bid", "job_id", "publisher_id", "group_id", "campaign_id"])
    df_clicks.createOrReplaceTempView('clicks')
    clicks_output = spark.sql("""
        select job_id, date(ts) as dates, hour(ts) as hours, publisher_id, campaign_id, group_id,
               avg(bid) as bid_set, count(*) as clicks, sum(bid) as spend_hour from clicks
        group by job_id, date(ts), hour(ts), publisher_id, campaign_id, group_id
    """)
    return clicks_output

def calculating_conversion(df):
    df_conversion = df.filter(col("custom_track")=='conversion')
    df_conversion = df_conversion.fillna(0, subset=["job_id", "publisher_id", "group_id", "campaign_id"])
    df_conversion.createOrReplaceTempView('conversion')
    conversion_output = spark.sql("""
        select job_id, date(ts) as dates, hour(ts) as hours, publisher_id, campaign_id, group_id,
               count(*) as conversions from conversion
        group by job_id, date(ts), hour(ts), publisher_id, campaign_id, group_id
    """)
    return conversion_output

def calculating_qualified(df):
    df_qualified = df.filter(col("custom_track")=="qualified")
    df_qualified = df_qualified.fillna(0, subset=["job_id", "publisher_id", "group_id", "campaign_id"])
    df_qualified.createOrReplaceTempView("qualified")
    qualified_output = spark.sql("""
        select job_id, date(ts) as dates, hour(ts) as hours, publisher_id, campaign_id, group_id,
               count(*) as qualified from qualified
        group by job_id, date(ts), hour(ts), publisher_id, campaign_id, group_id
    """)
    return qualified_output

def calculating_unqualified(df):
    df_unqualified = df.filter(col("custom_track")=="unqualified")
    df_unqualified = df_unqualified.fillna(0, subset=["job_id", "publisher_id", "group_id", "campaign_id"])
    df_unqualified.createOrReplaceTempView("unqualified")
    qualified_output = spark.sql("""
        select job_id, date(ts) as dates, hour(ts) as hours, publisher_id, campaign_id, group_id,
               count(*) as unqualified from unqualified
        group by job_id, date(ts), hour(ts), publisher_id, campaign_id, group_id
    """)
    return qualified_output

def process_final_data(clicks_output, conversion_output, qualified_output, unqualified_output):
    final_data = clicks_output.join(conversion_output,['job_id','dates','hours','publisher_id','campaign_id','group_id'],'full')\
        .join(qualified_output,['job_id','dates','hours','publisher_id','campaign_id','group_id'],'full')\
        .join(unqualified_output,['job_id','dates','hours','publisher_id','campaign_id','group_id'],'full')
    return final_data

def process_cassandra_data(df):
    clicks_output = calculating_clicks(df)
    conversion_output = calculating_conversion(df)
    qualified_output = calculating_qualified(df)
    unqualified_output = calculating_unqualified(df)
    final_data = process_final_data(clicks_output, conversion_output, qualified_output, unqualified_output)
    final_data = final_data.fillna(0, subset=["clicks","conversions","qualified","unqualified","bid_set","spend_hour"])
    return final_data

def retrieve_company_data():
    sql = """(SELECT job_id, company_id, group_id, campaign_id FROM job) job_table"""
    company = spark.read.format("jdbc") \
        .option("driver","com.mysql.cj.jdbc.Driver") \
        .option("url", "jdbc:mysql://mysql:3306/project_db")\
        .option("dbtable", sql) \
        .option("user", "root") \
        .option("password", "123") \
        .load()
    return company
```



```python
def import_to_mysql(output):
    output = output.select('job_id','dates','hours','publisher_id','campaign_id','company_id','group_id',
                           'unqualified','qualified','conversions','clicks','bid_set','spend_hour')
    output = output.withColumn('updated_at', current_timestamp())
    output = output.withColumnRenamed("qualified", "qualified_application").withColumnRenamed("unqualified", "disqualified_application")
    output = output.withColumnRenamed("conversions", "conversion")
    output.write.format("jdbc") \
        .option("driver","com.mysql.cj.jdbc.Driver") \
        .option("url", "jdbc:mysql://mysql:3306/project_db") \
        .option("dbtable", "events") \
        .mode("append") \
        .option("user", "root") \
        .option("password", "123") \
        .save()
    return print('Data imported successfully')

def main_task(mysql_time):
    print('-----------------------------')
    print('Retrieving data from Cassandra')
    print('-----------------------------')
    df = spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .option("spark.cassandra.connection.host", "cassandra") \
    .option("keyspace", "cassandra_data") \
    .option("table", "tracking") \
    .load().where(col('ts') > mysql_time)
    print('-----------------------------')
    print('Selecting data from Cassandra')
    print('-----------------------------')
    df = df.select('ts','job_id','custom_track','bid','campaign_id','group_id','publisher_id')
    df = df.filter(df.job_id.isNotNull())
    print('-----------------------------')
    print('Processing Cassandra Output')
    print('-----------------------------')
    cassandra_output = process_cassandra_data(df)
    print('-----------------------------')
    print('Merge Company Data')
    print('-----------------------------')
    company = retrieve_company_data()
    print('-----------------------------')
    print('Finalizing Output')
    print('-----------------------------')
    final_output = cassandra_output.join(company, ['job_id'], 'left').drop(company.group_id).drop(company.campaign_id)
    print(final_output.count())
    print('-----------------------------')
    print('Import Output to MySQL')
    print('-----------------------------')
    import_to_mysql(final_output)
    return print('Task Finished')

def get_lastest_time_cassandra():
    df = spark.read \
        .format("org.apache.spark.sql.cassandra") \
        .option("spark.cassandra.connection.host", "cassandra") \
        .option("keyspace", "cassandra_data") \
        .option("table", "tracking") \
        .load()
    df = df.select('ts').orderBy(col('ts').desc())
    latest_time = df.first()['ts']
    if latest_time is None:
        return datetime.datetime(1998, 1, 1, 23, 59, 59)
    return latest_time

def get_lastest_time_mysql():
    sql = """(SELECT max(updated_at) AS max_time FROM events) events_table"""
    mysql_df = spark.read.format("jdbc") \
        .option("driver","com.mysql.cj.jdbc.Driver") \
        .option("url", "jdbc:mysql://mysql:3306/project_db") \
        .option("dbtable", sql) \
        .option("user", "root") \
        .option("password", "123") \
        .load()
    mysql_time = mysql_df.first()['max_time']
    if mysql_time is None:
        mysql_time = datetime.datetime(1998, 1, 1, 23, 59, 59)
    return mysql_time

while True:
    start_time = datetime.datetime.now()
    cassandra_time = get_lastest_time_cassandra()
    print('Cassandra latest time is {}'.format(cassandra_time))
    mysql_time = get_lastest_time_mysql()
    print('MySQL latest time is {}'.format(mysql_time))
    if cassandra_time > mysql_time:
        print('New data found. Starting ETL process...')
        main_task(mysql_time)
    else:
        print('No new data found.')
    end_time = datetime.datetime.now()
    execution_time = (end_time - start_time).total_seconds()
    print('Execution time: {} seconds'.format(execution_time))
    time.sleep(10)
```

Knowledge Gained

- Near Real-Time Data Pipeline: Built and deployed a data engineering system for recruitment campaign performance with minimal latency (5-10s interval).
- CDC (Change Data Capture) ETL: Implemented a continuous ETL flow by extracting new/updated records from Cassandra based on timestamps, transforming them using Spark, and loading the aggregated metrics into MySQL.
- Big Data Tech Stack Proficiency: Gained hands-on experience integrating and orchestrating Cassandra (Data Lake), Apache Spark (ETL processing), MySQL (Data Warehouse), and Grafana (Visualization).
- Containerization & Orchestration: Leveraged Docker to containerize the entire system for easy deployment and management.