



Recruitment Pipeline

Prepared by: Vu Thien Son
Date: November 2025

Mục lục



01. TỔNG QUAN DỰ ÁN	1 - 5
02. CHUẨN BỊ DOCKER	6 - 11
03. ETL PIPELINE	12 - 18
04. TRỰC QUAN HÓA	19 - 20
05. CHUẨN BỊ SERVER	21 - 24
06. TRIỂN KHAI & CI/CD	25 - 26

01. Tổng quan



MỤC TIÊU DỰ ÁN



Xây dựng pipeline Micro-Batch ETL gần real-time từ CSV tĩnh và API giả lập CDC. Dữ liệu thô được lưu vào Cassandra (Data Lake), xử lý transform bằng Spark + Python, sau đó load vào MySQL (Data Warehouse) và trực quan hóa trên Grafana. Toàn bộ hệ thống được container hóa bằng Docker, triển khai trên máy ảo VirtualBox, kèm CI/CD trên GitHub.

INPUT

- File CSV tracking.
- API Python sinh sự kiện (giả lập CDC). Sanh ra bản ghi tương ứng metadata của tracking.

COLUMNS	MEANINGS
id	Thời điểm event được tạo (UUID v1)
job_id	ID công việc liên quan đến event
custom_track	Loại event: click, conversion, qualified, unqualified
BID	Giá thầu (bid) cho event
CAMPAIGN_ID	ID chiến dịch quảng cáo
GROUP_ID	ID nhóm
PUBLISHER_ID	ID nhà xuất bản

OUTPUT

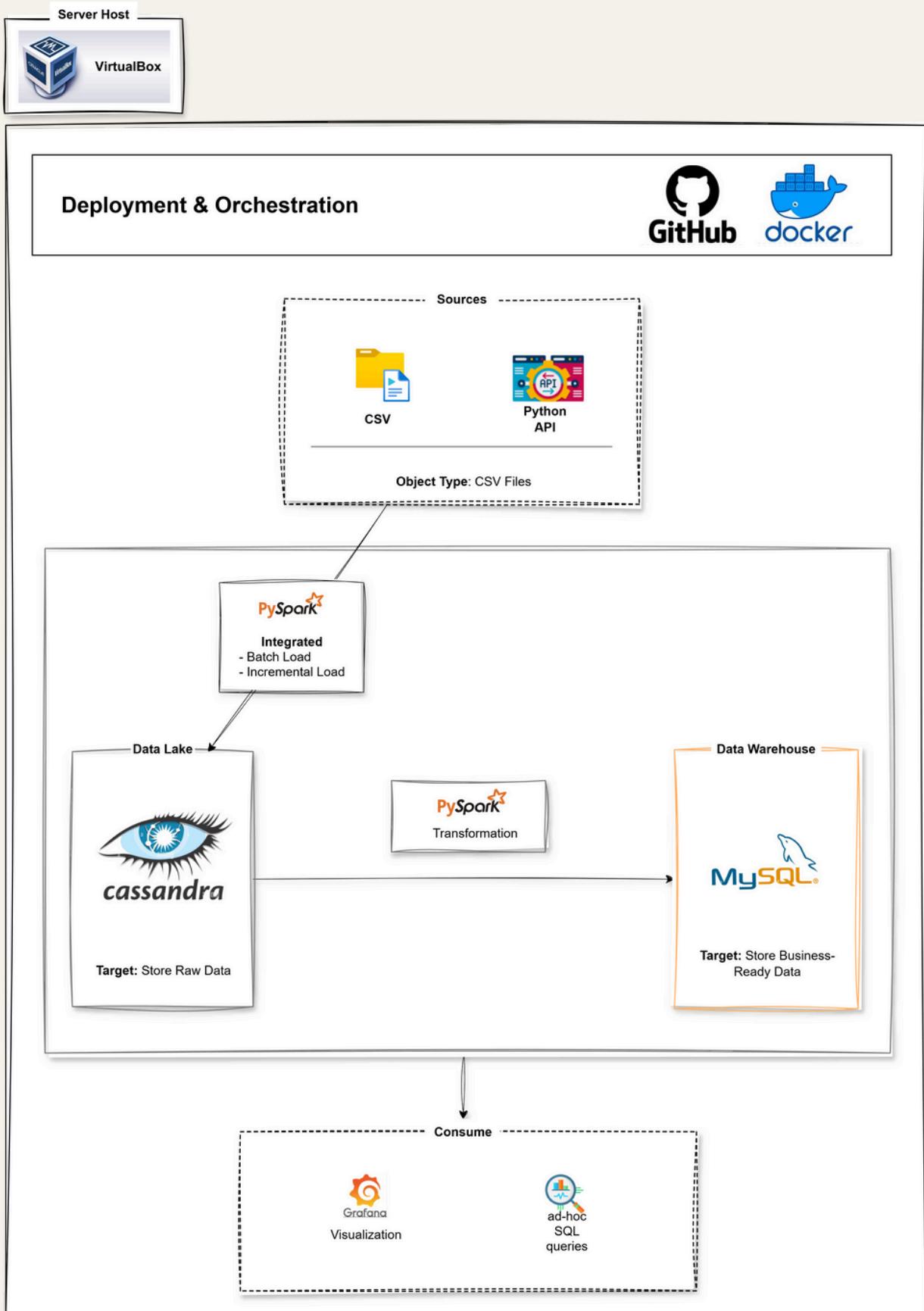


- Dashboard phân tích trên Grafana
- Pipeline ETL micro-batch tự động
- Tất cả chạy trên single VM VirtualBox
- Flat Table output:

COLUMNS	MEANINGS
id	ID bản ghi tự sinh (unique)
job_id	ID công việc liên quan đến event
dates	Ngày xảy ra event (YYYY-MM-DD)
HOURS	Giờ trong ngày khi event xảy ra (0-23)
COMPANY_ID	ID công ty sở hữu job/campaign
GROUP_ID	ID nhóm
CAMPAIGN_ID	ID chiến dịch quảng cáo
publisher_id	ID nhà xuất bản
CLICK	Số lần click
CONVERSION	Số lần chuyển đổi
QUALIFIED	Số lượt qualified
UNQUALIFIED	Số lượt unqualified
BID_SET	Giá thầu trung bình
SPEND_HOUR	Tổng chi tiêu theo giờ
SOURCES	Nguồn dữ liệu (ví dụ: Cassandra)
UPDATED_AT	Thời điểm bản ghi được cập nhật

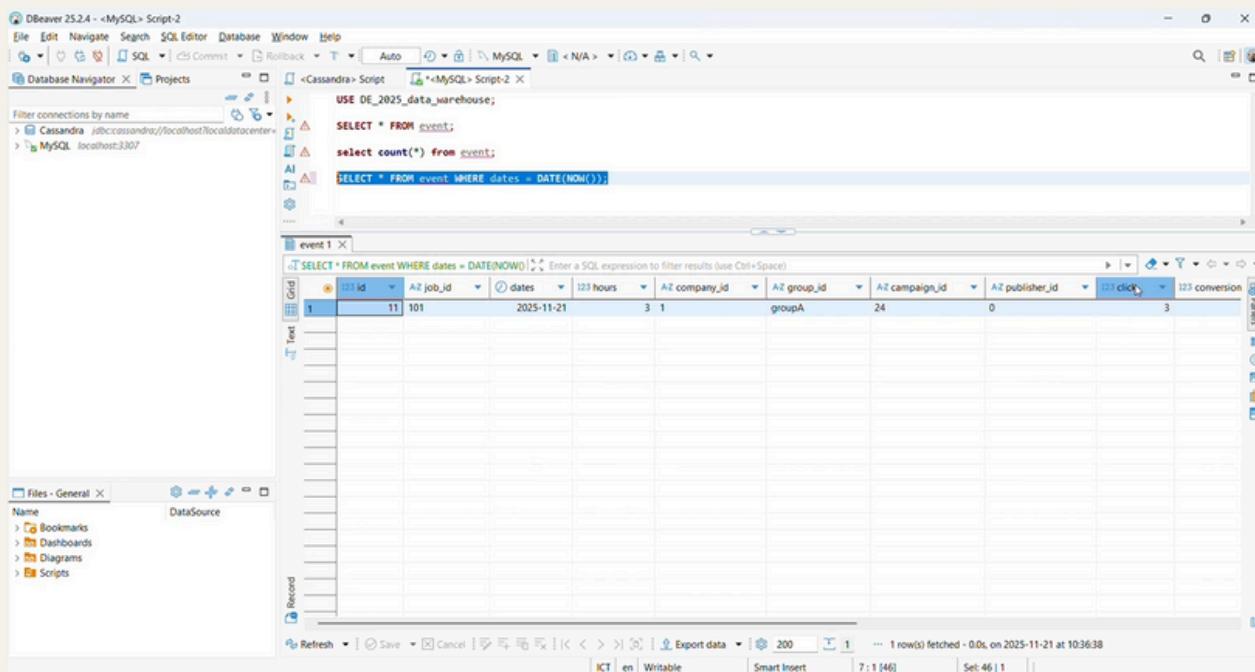
HIGH LEVEL ARCHITECTURE

Dữ liệu đi từ Sources → Ingest → Data Lake (Cassandra) → ETL (Spark) → Data Warehouse (MySQL) → Consumers (Grafana / SQL). Tất cả components đều được đóng gói bằng docker và chạy trong VM.



DEMO KẾT QUẢ

ETL chạy tự động liên tục



DBeaver 25.2.4 - <MySQL> Script-2

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator Projects

Filter connections by name

> Cassandra jdbc:cassandra://localhost/localspacecenter

> MySQL localhost:3307

USE DE_2025_data_warehouse;

SELECT * FROM event;

select count(*) from event;

AI SELECT * FROM event WHERE dates = DATE(NOW());

event 1

1 row(s) fetched - 0.0s, on 2025-11-21 at 10:36:38

Grid Test Record

File - General

Name Bookmarks Dashboards Diagrams Scripts

DataSource

Refresh Save Cancel Export data 200 1 1 row(s) fetched - 0.0s, on 2025-11-21 at 10:36:38

ICT en Writable Smart Insert 7:1 [46] Set: 46 | 1

Grafana Dashboard



SERVER HOẠT ĐỘNG TỰ ĐỘNG

Cassandra (Data Lake)

Spark xử lý micro-batch liên tục

MySQL (Data Warehouse)

02. Chuẩn bị Docker



MỤC TIÊU



Build image Spark riêng, Cassandra và MySQL dùng image từ Docker Hub.

```
docker/
├── config
│   └── spark-defaults.conf
├── spark
│   └── Dockerfile
├── .env
└── docker-compose.yml
```

spark-defaults.conf

- File cấu hình mặc định của Spark

spark/Dockerfile

- Định nghĩa image Spark tùy chỉnh, cài connector, thư viện cần thiết...

docker-compose.yml

- File chính để chạy toàn bộ hệ thống bằng Docker Compose.
- Quản lý nhiều container như Spark, Cassandra, MySQL, Grafana...

entrypoint.sh

- Script chạy khi container Spark khởi động.

requirements.txt

- Danh sách các thư viện Python cần cài cho project

.env

- Chứa các biến môi trường cho Docker Compose

SPARK-DEFAULTS.CONF



```
docker > config > spark-defaults.conf
1 # Master & UI
2 spark.master          spark://spark-master:7077
3 spark.ui.enabled      true
4
5 # Event log (lưu lại history của job)
6 spark.eventLog.enabled   true
7 spark.eventLog.dir       file:/opt/spark/spark-events
8 spark.history.fs.logDirectory  file:/opt/spark/spark-events
9
10 # Driver
11 spark.driver.memory     5g
12 spark.driver.cores      2 # dùng 2 core cho driver
13
14 # Executors (2 worker)
15 spark.executor.memory    4g # mỗi worker 4 GB
16 spark.executor.cores      3 # mỗi worker dùng 4 core
```

- Hệ thống dùng Spark theo mô hình cluster, có một Spark Master và nhiều Worker.
- Spark UI được bật để theo dõi quá trình chạy job.
- Lịch sử chạy job được lưu tại /opt/spark/spark-events để có thể xem lại khi cần.
- Driver dùng 5 GB RAM / 2 CPU, mỗi executor dùng 4 GB RAM / 3 CPU, đủ cho việc xử lý dữ liệu theo dạng Micro-Batch.

ENTRYPOINT.SH



```
docker > entrypoint.sh
1 #!/bin/bash
2
3 SPARK_WORKLOAD=$1
4
5 echo "SPARK_WORKLOAD: $SPARK_WORKLOAD"
6
7 if [ "$SPARK_WORKLOAD" == "master" ];
8 then
9   start-master.sh -p 7077
10 elif [ "$SPARK_WORKLOAD" == "worker" ];
11 then
12   start-worker.sh spark://spark-master:7077
13 fi
```

Script xác định vai trò của container:

- “master” → chạy Spark Master.
- “worker” → chạy Spark Worker và nối vào Master.

SPARK/DOCKERFILE



```
FROM python:3.10-bookworm as spark-base

# Cài tool cần thiết
RUN apt-get update && \
    apt-get install -y sudo curl vim unzip rsync openjdk-17-jdk build-essential software-properties-common ssh && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

# Thiết lập môi trường
ENV SPARK_HOME=/opt/spark
ENV HADOOP_HOME=/opt/hadoop
RUN mkdir -p ${SPARK_HOME} ${HADOOP_HOME}
WORKDIR ${SPARK_HOME}

# Tải Spark
RUN curl https://archive.apache.org/dist/spark/spark-3.5.6/spark-3.5.6-bin-hadoop3.tgz -o spark.tgz \
&& tar xvzf spark.tgz --strip-components 1 \
&& rm spark.tgz

# Cài Python dependencies
COPY requirements.txt .
RUN pip3 install -r requirements.txt

# Config Spark
ENV PATH="$SPARK_HOME/sbin:$SPARK_HOME/bin:$PATH"
ENV SPARK_MASTER="spark://spark-master:7077"
ENV PYSPARK_PYTHON=python3
COPY config/spark-defaults.conf $SPARK_HOME/config
RUN chmod +x $SPARK_HOME/sbin/* $SPARK_HOME/bin/*

ENV PYTHONPATH=$SPARK_HOME/python/:$PYTHONPATH

# Entrypoint
COPY entrypoint.sh .
ENTRYPOINT ["/entrypoint.sh"]
```

- Dockerfile tạo image Spark tùy chỉnh dựa trên Python 3.10.
- Nó cài các công cụ cần thiết, Java 17, tải Spark 3.5.6, cài Python packages, thêm cấu hình Spark và thiết lập biến môi trường.
- Cuối cùng, gán entrypoint để container có thể chạy dưới vai trò master hoặc worker.

DOCKER-COMPOSE.YML



```
# =====
# Cassandra (Data Lake Storage)
# =====
▷ Run Service
cassandra:
  image: cassandra:4.1
  container_name: cassandra_dl
  ports:
    - "9042:9042"          # Cổng truy cập Cassandra
  volumes:
    - cassandra-data:/var/lib/cassandra  # Lưu data bền vững
  environment:
    CASSANDRA_CLUSTER_NAME: "first_cluster"
  healthcheck:           # Kiểm tra Cassandra đã sẵn sàng chưa
    test: ["CMD-SHELL", "cqlsh -e 'describe keyspaces'"]
    interval: 10s
    retries: 5
  networks:
    - de_project
```

Cassandra container dùng image 4.1, mở cổng 9042, lưu dữ liệu qua volume, thiết lập tên cluster và có healthcheck để đảm bảo dịch vụ sẵn sàng.

```
# =====
# MySQL (Data Warehouse)
# =====
▷ Run Service
mysql:
  image: mysql:8.0.44-debian
  container_name: mysql_dwh
  ports:
    - "3307:3306"          # Map cổng máy host -> container
  environment:
    MYSQL_ROOT_PASSWORD: 123
  volumes:
    - mysql-data:/var/lib/mysql      # Lưu database bền vững
  healthcheck:           # Ping kiểm tra MySQL hoạt động
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    interval: 10s
    retries: 5
  networks:
    - de_project
```

MySQL container dùng image 8.0.44, mở cổng 3307, lưu dữ liệu qua volume, thiết lập mật khẩu root và có healthcheck để đảm bảo dịch vụ sẵn sàng.

```

# =====
# Spark Master (Engine xử lý dữ liệu)
# =====
▷Run Service
spark-master:
  container_name: spark-engine
  build:
    context: .
    dockerfile: spark/Dockerfile      # Build image Spark tùy chỉnh
    image: da-spark-image
    entrypoint: ['./entrypoint.sh', 'master']  # Khởi chạy Master
    depends_on:
      mysql:                                # Chờ MySQL & Cassandra sẵn sàng
      mysql:
        condition: service_healthy
      cassandra:
        condition: service_healthy
  healthcheck:
    test: [CMD, "curl", "-f", "http://localhost:8080"]  # Kiểm tra UI
    interval: 5s
    timeout: 3s
    retries: 3
  volumes:
    - ./data:/opt/spark/data          # Input data
    - ./etl:/opt/spark/etl            # Mã ETL Python
    - spark-logs:/opt/spark/spark-events # Lưu event logs
    - ./driver:/opt/spark/driver      # MySQL driver
  env_file:
    - .env                           # Biến môi trường
  ports:
    - '9090:8080'      # Spark UI
    - '7077:7077'      # Spark Master
    - '4041:4040'      # Spark job UI
  networks:
    - de_project

```

Spark Master container dùng image tùy chỉnh, khởi chạy qua entrypoint, phụ thuộc MySQL & Cassandra, map các cổng UI và Master, mount volume dữ liệu/ETL, và có healthcheck để đảm bảo sẵn sàng.

```

# =====
# Grafana (Visualization)
# =====
▷Run Service
grafana:
  image: grafana/grafana:latest
  container_name: grafana
  environment:
    - GF_SECURITY_ADMIN_USER=admin      # Tài khoản đăng nhập
    - GF_SECURITY_ADMIN_PASSWORD=admin
  ports:
    - "3000:3000"                      # UI Grafana
  networks:
    - de_project

```

Grafana container dùng image mới nhất, thiết lập tài khoản admin, mở cổng 3000 để truy cập UI và kết nối vào mạng chung de_project.

```

# =====
# Persistent Volumes
# =====
volumes:
  cassandra-data:
  mysql-data:
  spark-logs:

# =====
# Network chung cho toàn hệ thống
# =====
networks:
  de_project:
    name: de_project
    driver: bridge

```

Grafana container dùng image mới nhất, thiết lập tài khoản admin, mở cổng 3000 để truy cập UI và kết nối vào mạng chung de_project.

.ENV

.ENV

```
1 COMPOSE_PROJECT_NAME=de_project # đặt tên chung cho container
2 SPARK_NO_DAEMONIZE=true #Spark sẽ chạy trực tiếp trên terminal, không chạy nền (daemon)
```

- **COMPOSE_PROJECT_NAME=de_project** → đặt tên chung cho project Docker, các container sẽ tự động lấy tiền tố de_project để tránh trùng tên.
- **SPARK_NO_DAEMONIZE=true** → Spark sẽ chạy trực tiếp trên terminal, không chạy nền (daemon), giúp dễ theo dõi log và debug.

REQUIREMENTS.TXT



```
pandas
numpy
pyspark
cassandra-driver
mysql-connector-python
```

- **pandas** → xử lý dữ liệu dạng bảng, thao tác CSV/Excel.
- **numpy** → tính toán số học, xử lý mảng và ma trận.
- **pyspark** → xử lý dữ liệu lớn, chạy ETL bằng Spark.
- **cassandra-driver** → kết nối và thao tác dữ liệu với Cassandra.
- **mysql-connector-python** → kết nối và thao tác dữ liệu với MySQL.

03. ETL Pipeline



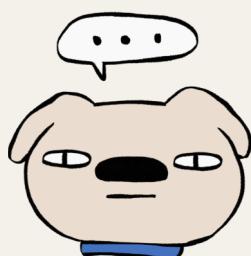
MỤC TIÊU



Cassandra.py và **MySql.py** đảm nhiệm việc kết nối và thao tác dữ liệu với Cassandra và MySQL; **Main.py** là file chính để chạy toàn bộ pipeline ETL; còn **generate_data Automatically.py** dùng để tạo dữ liệu giả lập, mô phỏng CDC tự động.



- └─  **Cassandra.py**
- └─  **Main.py**
- └─  **MySql.py**
- └─  **generate_data Automatically.py**



CÁC FILE ĐƯỢC VIẾT NHƯ THẾ NÀO?

Cassandra.py

```
# -----
# Lớp Cassandra
# -----
class Cassandra:
    # -----
    # Constructor
    # -----
    def __init__(self, spark):
        self.spark = spark

    # -----
    # Đọc dữ liệu từ bảng
    # -----
    def read(self, table):
        df = (
            self.spark.read.format("org.apache.spark.sql.cassandra")
            .options(keyspace="de_2025_datalake", table=table)
            .load()
        )
        return df

    # -----
    # Ghi DataFrame vào bảng (overwrite), dùng cho lần ghi dữ liệu vào đầu tiên
    # -----
    def insert(self, table, df):
        try:
            df.write.format("org.apache.spark.sql.cassandra").mode("overwrite").options(
                **{
                    "keyspace": "de_2025_datalake",
                    "table": table,
                    "confirm.truncate": "true", # Truncate & Load
                }
            ).save()
            return True
        except Exception as e:
            return False

    # -----
    # Ghi DataFrame vào bảng (append), dùng cho CDC
    # -----
    def insert_random(self, table, df):
        try:
            df.write.format("org.apache.spark.sql.cassandra").mode("append").options(
                {"keyspace": "de_2025_datalake", "table": table}
            ).save()
            return True
        except Exception as e:
            print("Insert random error:", e)
            return False
```

File này định nghĩa lớp **Cassandra** để tương tác với cơ sở dữ liệu Cassandra từ PySpark.

Mục đích chính:

- **read** → đọc dữ liệu từ bảng **Cassandra** thành DataFrame.
- **insert** → ghi DataFrame vào bảng với chế độ overwrite, dùng cho lần ghi ban đầu.
- **insert_random** → ghi DataFrame với chế độ append, dùng cho dữ liệu mới hoặc CDC (Change Data Capture).

MySql.py

```
# =====
# Lớp hỗ trợ MySQL
# =====
class MySql:
    # -----
    # Khởi tạo kết nối
    # -----
    def __init__(self, spark):
        self.spark = spark
        # self.url = "jdbc:mysql://localhost:3307/DE_2025_data_warehouse"
        self.url = "jdbc:mysql://mysql_dwh:3306/DE_2025_data_warehouse" # URL kết nối tới MySQL container
        self.user = "root"
        self.password = "123"
        self.driver = "com.mysql.cj.jdbc.Driver"

    # -----
    # Đọc dữ liệu từ bảng MySQL
    # -----
    def read(self, table):
        df = (
            self.spark.read.format("jdbc")
            .option("url", self.url)
            .option("dbtable", table)
            .option("user", self.user)
            .option("password", self.password)
            .option("driver", self.driver)
            .load()
        )
        return df

    # -----
    # Ghi DataFrame vào MySQL (overwrite)
    # -----
    def insert(self, table, df):
        try:
            df.write.format("jdbc").option("url", self.url).option(
                "dbtable", table
            ).option("user", self.user).option("password", self.password).option(
                "driver", self.driver
            ).mode(
                "overwrite"
            ).save()

            return True
        except Exception as e:
            print("Insert random error:", e)
            return False
```

File này định nghĩa lớp **MySQL** để tương tác với cơ sở dữ liệu **MySQL** từ PySpark.

Mục đích chính:

- **read** → đọc dữ liệu từ bảng **MySQL** thành DataFrame.
- **insert** → ghi DataFrame vào bảng với chế độ overwrite.

Main.py

```
import os
from uuid import UUID
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StringType
from cassandra.util import datetime_from_uuid1

from Cassandra import Cassandra
from MySql import MySql

# =====
# SPARK CONFIG
# =====

MySQL_JAR = os.path.abspath("../driver/mysql-connector-j-8.0.33.jar")

spark = (
    SparkSession.builder.config(
        "spark.jars.packages", "com.datastax.spark:spark-cassandra-connector_2.12:3.1.0"
    )
    .config("spark.cassandra.connection.host", "cassandra_dl")
    .config("spark.cassandra.connection.port", "9042")
    .config("spark.driver.extraClassPath", MySQL_JAR)
    .config("spark.executor.extraClassPath", MySQL_JAR)
    .getOrCreate()
)

# DB Instances
cass = Cassandra(spark)
mysql = MySql(spark)
```

Khởi tạo môi trường Spark và kết nối tới cơ sở dữ liệu:

- Cấu hình **Spark** với **Spark-Cassandra Connector** và **MySQL driver** (host Cassandra, port, đường dẫn driver MySQL).
- Khởi tạo các **instance**: cass để thao tác Cassandra, mysql để thao tác MySQL.

Link tải connector

[MySQL::Download MySQL Connector/J \(Archived Versions\)](#)

```
# =====
#     UDF
# =====

@udf(returnType=StringType())
def extract_timestamp_from_uuid(uuid_str):
    try:
        u = UUID(uuid_str)
        dt = datetime_from_uuid1(u)
        return dt.strftime("%Y-%m-%d %H:%M:%S")
    except:
        return None

# =====
#     UTILITY READER
# =====

def spark_read_file(path):
    base_dir = os.path.dirname(os.path.abspath(__file__))
    return spark.read.csv(os.path.join(base_dir, path), header=True)
```

UDF extract_timestamp_from_uuid:

- Chuyển **UUID** (kiểu UUID v1) thành **timestamp**.
- Trả về chuỗi thời gian dạng "YYYY-MM-DD HH:MM:SS".
- Dùng trong Spark để trích xuất thời gian từ các ID tạo sẵn.

Hàm spark_read_file:

- Đọc file CSV từ đường dẫn tương đối.
- Trả về DataFrame.

```

class DataTransformer:
    valid_events = ["click", "conversion", "qualified", "unqualified"]

    @staticmethod
    def preprocess(df):
        # Get timestamp from uuid
        df = df.withColumn("system_ts", extract_timestamp_from_uuid(col("create_time")))
        df = df.withColumn(
            "system_ts", to_timestamp("system_ts", "yyyy-MM-dd HH:mm:ss")
        )

        # Select useful rows and cols
        df = df.select(
            "create_time",
            "system_ts",
            "job_id",
            "custom_track",
            "bid",
            "campaign_id",
            "group_id",
            "publisher_id",
        ).filter("job_id IS NOT NULL AND custom_track IS NOT NULL")

        return df

    @staticmethod
    def aggregate(df):
        df = df.filter(col("custom_track").isin(DataTransformer.valid_events))

        # Split dates and hours from timestamp
        df = df.withColumn("dates", to_date("system_ts"))
        df = df.withColumn("hours", hour("system_ts"))

        pivot_df = (
            df.groupBy(
                "job_id", "dates", "hours", "publisher_id", "campaign_id", "group_id"
            )
            .pivot("custom_track", DataTransformer.valid_events)
            .agg(count("*").alias("count"))
        )

        # Rename {event}_count → event
        for e in DataTransformer.valid_events:
            pivot_df = pivot_df.withColumnRenamed(f"{e}_count", e)

        # Spend & bid
        metric_df = df.groupBy("job_id", "publisher_id", "campaign_id", "group_id").agg(
            round(sum("bid"), 2).alias("spend_hour"),
            round(avg("bid"), 2).alias("bid_set"),
        )

        return pivot_df.join(
            metric_df, ["job_id", "publisher_id", "campaign_id", "group_id"], "left"
        )

    @staticmethod
    def fill_null(df):
        fill_values = {
            "click": 0,
            "conversion": 0,
            "qualified": 0,
            "unqualified": 0,
            "spend_hour": 0,
            "bid_set": 0,
        }
        return df.fillna(fill_values)

    @staticmethod
    def post_process(df):
        # Add Primary Key
        df = df.withColumn("id", monotonically_increasing_id())

        # Mark update dates
        df = df.withColumn("updated_at", current_timestamp())
        return df.select(
            "id",
            "job_id",
            "dates",
            "hours",
            "company_id",
            "group_id",
            "campaign_id",
            "publisher_id",
            "click",
            "conversion",
            "qualified",
            "unqualified",
            "bid_set",
            "spend_hour",
            "sources",
            "updated_at",
        )

    @staticmethod
    def transform_full(df):
        df = DataTransformer.preprocess(df)
        df = DataTransformer.aggregate(df)
        df = df.withColumn("sources", lit("Cassandra"))
        df = DataTransformer.fill_null(df)

        job_df = mysql.read("job").select(col("id").alias("job_id"), "company_id")
        df = df.join(job_df, "job_id", "left")

        return DataTransformer.post_process(df)

```

valid_events

- Các sự kiện phù hợp: click, conversion, qualified, unqualified.

preprocess(df)

- Trích xuất timestamp từ create_time (UUID).
- Chuyển sang định dạng timestamp và giữ các cột cần thiết.
- Lọc các bản ghi có job_id và custom_track hợp lệ.

aggregate(df)

- Lọc chỉ các sự kiện hợp lệ.
- Tách ngày (dates) và giờ (hours) từ timestamp.
- Pivot các loại sự kiện để đếm số lượng theo mỗi job/publisher/campaign/group.
- Tính tổng chi tiêu (spend_hour) và giá thầu trung bình (bid_set).
- Kết hợp dữ liệu pivot với dữ liệu chi tiêu.

fill_null(df)

- Thay giá trị NULL bằng 0.

post_process(df)

- Thêm primary key id.
- Thêm cột updated_at đánh dấu thời gian xử lý.

transform_full(df)

- Thực hiện toàn bộ pipeline: preprocess → aggregate → fill null → thêm nguồn (sources) → join với bảng job để lấy company_id → post_process.



```
# =====
#           SYNC CHECK BETWEEN MYSQL & CASS
# =====

class DataSync:

    @staticmethod
    def last_mysql_date():
        df = mysql.read("event")
        return df.select(max("updated_at")).first()[0]

    @staticmethod
    def last_cassandra_date():
        df = cass.read("tracking")
        df = df.withColumn("create_time", extract_timestamp_from_uuid("create_time"))
        df = df.withColumn("create_time", to_timestamp("create_time"))
        df = df.withColumn(
            "create_time", from_utc_timestamp("create_time", "Asia/Ho_Chi_Minh")
        )
        return df.select(max("create_time")).first()[0]
```

```
# =====
#           MAIN ETL
# =====

def run_etl():
    df = cass.read("tracking")
    df = DataTransformer.transform_full(df)
    mysql.insert("event", df)

# =====
#           ENTRY POINT
# =====

if __name__ == "__main__":
    # Initial load
    print("Insert Cassandra")
    cass.insert("tracking", spark_read_file("../data/cassandra/tracking.csv"))

    print("Insert MySQL")
    mysql.insert("job", spark_read_file("../data/mysql/job.csv"))

    # First ETL
    run_etl()

    # Continuous sync
    while True:
        if DataSync.last_mysql_date() < DataSync.last_cassandra_date():
            run_etl()
```

last_mysql_date()

- Lấy thời gian cập nhật mới nhất trong bảng event của MySQL.

last_cassandra_date()

- Lấy thời gian tạo mới nhất trong bảng tracking của Cassandra (chuyển UUID → timestamp trước).

run_etl()

- Đọc dữ liệu từ Cassandra (tracking)
- Transform theo pipeline
- Ghi kết quả vào MySQL (event)

__main__

- Tải dữ liệu mẫu lần đầu
 - Ghi file CSV vào Cassandra (tracking)
 - Ghi file CSV vào MySQL (job)
- Chạy ETL lần đầu
- Vòng lặp đồng bộ liên tục
 - Nếu Cassandra có timestamp mới hơn MySQL → chạy ETL cập nhật.

generate_data_automatically

```
import uuid
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
from Cassandra import Cassandra

spark = (
    SparkSession.builder.config(
        "spark.jars.packages", "com.datastax.spark:spark-cassandra-connector_2.12:3.1.0"
    )
    .getOrCreate()
)

# Dữ liệu cơ bản
data = [
    { ... }
]

# Tạo DataFrame từ dữ liệu cơ bản
df = spark.createDataFrame(data)

# UDF để tạo UUID mới cho create_time
uuid_udf = udf(lambda: str(uuid.uuid1()), StringType())

# Thêm cột create_time mới mỗi lần insert
df = df.withColumn("create_time", uuid_udf())
```

```
import time

cass = Cassandra(spark)
while True:
    cass.insert_random("tracking", df)
    time.sleep(5)
    print("Thêm thành công")
```

```
Thêm thành công
```

Mục đích chính:

- Giả lập dữ liệu đưa vào **Cassandra** liên tục.

04. Trực quan hóa



Cấu hình Grafana

The screenshot shows the Grafana interface for configuring a MySQL data source. The left sidebar is collapsed, and the main area displays the 'mysql' configuration page. The 'Settings' tab is selected. The 'Name' field is set to 'mysql'. A note at the top states: 'Before you can use the MySQL data source, you must configure it below or in the config file. For detailed instructions, [view the documentation](#)'. Below this, under 'User Permission', there is a warning about database user permissions. The 'Connection' section includes fields for 'Host URL' (set to 'mysql_dwh:3306') and 'Database name' (set to 'DE_2025_data_warehouse'). The 'Authentication' section shows 'Username' as 'root' and 'Password' as 'configured'. At the bottom, there is a note about 'TLS Client Auth'.

Kết nối mysql (data warehouse)

Luồng: Grafana → Connections → Data Sources → mysql

Host URL: <container_name>:<port>

Tên DB: <tên_database_mysql>

User name và password: đã thiết lập ở docker-compose

Dashboard



Chỉ số:

- Phạm vi hẹp: Chỉ 1 công ty và 5 việc làm.
- Tỷ lệ chuyển đổi: Đạt 1.21%, ở mức trung bình-thấp.
- Chi tiêu: Tổng chi là 1475 đơn vị tiền tệ.

Biểu đồ Cột (Hoạt động theo Giờ):

- Hoạt động cao điểm: Tập trung mạnh vào lúc 2 giờ (231 lượt) và 17 giờ (166 lượt).
- Hoạt động thấp điểm: Rơi vào giờ lẻ (1, 12, 13, 15, 23) với số lượng chỉ từ 3 đến 17 lượt.

Biểu đồ Tròn (Tỷ trọng Công việc):

- Tập trung tuyệt đối: Thành phần màu Xanh lá cây (công việc có job_id = "2") chiếm tỷ trọng áp đảo (trên 80%).
- Phân bổ không đồng đều: Cho thấy sự tập trung gần như toàn bộ vào một công việc duy nhất.

05. Chuẩn bị Server



1. Cài đặt VM

Cài VirtualBox: <https://www.virtualbox.org/wiki/Downloads>



Tải Ubuntu Server: <https://ubuntu.com/download/server>

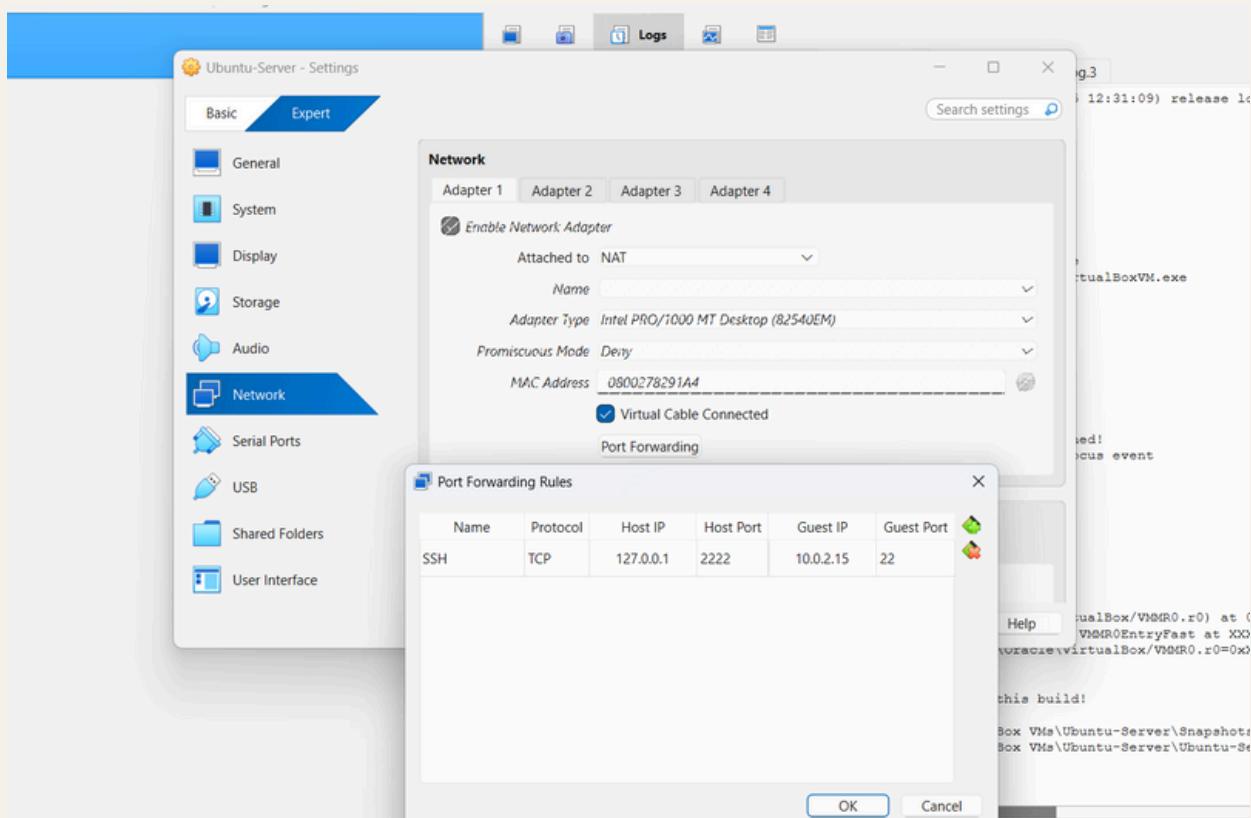
ubuntu®

2. Cấu hình

2.1 Tạo máy ảo

1. VirtualBox → New → Name: Ubuntu_VM
2. Type: Linux, Version: Ubuntu (64-bit)
3. RAM: 2–4 GB
4. Hard disk: 20GB+
5. Start VM → cài Ubuntu từ ISO

2.2 Cấu hình mạng



- **Luồng:** Settings → Network → Adapter 1 → Port Forwarding
- Mở port SSH host → VM (ví dụ host port **2222** → guest port **22**)

2.3 Cài OpenSSH server

```
sudo apt update
sudo apt install openssh-server -y
sudo systemctl enable ssh
sudo systemctl start ssh
```

Cài đặt và bật dịch vụ SSH trên Ubuntu để cho phép truy cập từ xa.

Test

```
C:\Users\sonvuu>ssh thomsay@127.0.0.1 -p 2222
thomsay@127.0.0.1's password:
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Fri Nov 21 02:39:41 AM UTC 2025

System load:          0.08
Usage of /:            20.7% of 15.31GB
Memory usage:          13%
Swap usage:            0%
Processes:             119
Users logged in:      1
IPv4 address for enp0s3: 10.0.2.15
IPv6 address for enp0s3: fd17:625c:f037:2:a00:27ff:fe82:91a4

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Fri Nov 21 02:35:57 2025 from 10.0.2.2
thomsay@Ubuntu-Server:~$ docker --version
Docker version 28.2.2, build 28.2.2-0ubuntu1~24.04.1
thomsay@Ubuntu-Server:~$ git --version
git version 2.43.0
```

Kiểm tra SSH từ host:

```
ssh <username>@<host_ip> -p <host_port>
```

2.4 Cài đặt thêm các gói hỗ trợ



Trong bước này, chúng ta sẽ cài đặt những công cụ cần thiết để chạy project: Git, Docker Engine, và Docker Compose v2.

1. Cài đặt công cụ cơ bản

```
sudo apt update  
sudo apt install -y git ca-certificates curl gnupg
```

Giải thích:

- git : dùng để clone mã nguồn từ GitHub
- curl : dùng để tải file từ internet
- gnupg : dùng để xác thực chữ ký GPG
- ca-certificates : đảm bảo kết nối HTTPS an toàn

2. Thêm Docker GPG Key

```
sudo install -m 0755 -d /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --batch --yes --dearmor -o /etc/apt/  
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

Ghi chú:

- Docker yêu cầu key GPG để đảm bảo package tải về là thật, không bị sửa đổi.
- File key được lưu trong /etc/apt/keyrings (chuẩn mới của Ubuntu).

3. Thêm Docker repository vào hệ thống

```
echo \  
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.dock  
$ (. /etc/os-release && echo $VERSION_CODENAME) stable" | \  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Cài Docker Engine + Docker Compose v2

```
sudo apt update  
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

06. Triển khai & CI/CD



MỤC TIÊU



Mục tiêu của phần **Deployment** là triển khai hệ thống **ETL** và toàn bộ các service (*Spark, Cassandra, MySQL, Grafana*) lên môi trường chạy thực tế một cách ổn định và nhất quán. Phát triển và kiểm thử toàn bộ hệ thống ở local, sau đó push code lên **Git**, và server sẽ pull bản mới nhất về rồi chạy **docker compose up -d** để cập nhật dịch vụ. Cách làm này giúp việc triển khai trở nên đơn giản, nhanh chóng và dễ tái lập.

LOCAL

```
● PS D:\2025\DE_2025_Recruitment_Pipeline> git add .
● PS D:\2025\DE_2025_Recruitment_Pipeline> git commit -m "Done Project"
[main 19f6e16] Done Project
 7 files changed, 83 insertions(+), 136 deletions(-)
  delete mode 100644 etl/test.ipynb
● PS D:\2025\DE_2025_Recruitment_Pipeline> git push origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 12 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 2.57 KiB | 2.57 MiB/s, done.
Total 10 (delta 6), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.
To https://github.com/sonvuuu28/DE_2025_Recruitment_Pipeline.git
 937bf9a..19f6e16  main -> main
```

Đẩy code lên github

SERVER

Mục đích: kéo code từ github về server và chạy ở server

```
git clone https://github.com/sonvuuu28/DE_2025_Recruitment_Pipeline.git
```

```
thomsay@Ubuntu-Server:~$ git clone https://github.com/sonvuuu28/DE_2025_Recruitment_Pipeline.git
Cloning into 'DE_2025_Recruitment_Pipeline'...
remote: Enumerating objects: 67, done.
remote: Total 67 (delta 0), reused 0 (delta 0), pack-reused 67 (from 1)
Receiving objects: 100% (67/67), 45.30 MiB | 2.93 MiB/s, done.
Resolving deltas: 100% (20/20), done.
thomsay@Ubuntu-Server:~$ ls
DE_2025_Recruitment_Pipeline
thomsay@Ubuntu-Server:~$ cd DE_2025_Recruitment_Pipeline/
thomsay@Ubuntu-Server:~/DE_2025_Recruitment_Pipeline$ ls
data docker driver etl image README.md
thomsay@Ubuntu-Server:~/DE_2025_Recruitment_Pipeline$ cd docker/
```

```
sudo docker-compose up -d
```

```
thomsay@Ubuntu-Server:~/DE_2025_Recruitment_Pipeline/docker$ sudo docker compose up -d
[+] Running 5/5
✓ Network de_project     Created
✓ Container cassandra_dl Started
✓ Container mysql_dwh     Started
✓ Container spark-engine   Started
✓ Container grafana        Started
```