TÀI LIỆU HƯỚNG DẪN THỰC HÀNH

Tên bài thực hành:

Truyền dữ liệu trực tuyến sử dụng công nghệ dữ liệu Airflow, Kafka, Spark, Cassandra

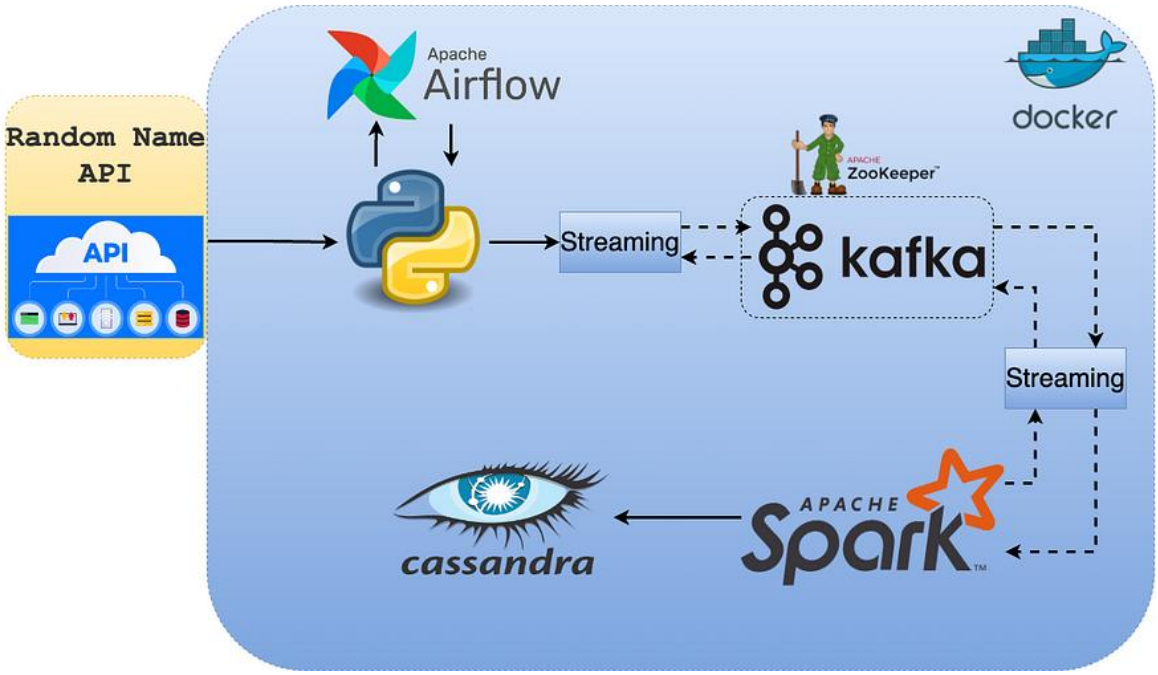
10/2023

MỤC LỤC

1. CƠ SỞ LÝ THUYẾT

1.1. Giới thiệu chung

Làm quen với kiến trúc



Hình 1 Kiến trúc của hệ thống truyền dữ liệu

+ Khái niệm**:**

- Apache Spark: Apache Spark là một framework xử lý dữ liệu phân tán mã nguồn mở được phát triển bởi Apache Software Foundation. Nó cung cấp khả năng xử lý và phân tích dữ liệu lớn một cách hiệu quả, hỗ trợ xử lý batch, xử lý dự đoán thời gian thực, và xử lý dữ liệu trực tiếp từ nhiều nguồn khác nhau. Spark sử dụng cơ sở dữ liệu phân tán in-memory để tăng tốc quá trình xử lý dữ liệu.
- Apache Kafka: Apache Kafka là một hệ thống truyền phát dữ liệu phân tán mã nguồn mở. Nó được thiết kế để xử lý việc truyền tải dữ liệu và sự kiện từ nhiều nguồn đến nhiều đích một cách hiệu quả. Kafka thường được sử dụng để xử lý luồng dữ liệu thời gian thực, như nhật ký (log) hệ thống, dữ liệu sự kiện, và dữ liệu IoT.
- Apache Airflow: Apache Airflow là một hệ thống quản lý luồng công việc và lên lịch công việc mã nguồn mở. Nó cho phép bạn định nghĩa, lên lịch, và

theo dõi các luồng công việc phức tạp, đặc biệt là trong lĩnh vực xử lý dữ liệu và khoa học dữ liệu. Airflow cung cấp giao diện đồ họa để thiết kế và theo dõi các luồng công việc.

- Docker: Docker là một nền tảng ảo hóa mã nguồn mở cho việc đóng gói, triển khai, và quản lý ứng dụng và dịch vụ. Docker sử dụng các container để đóng gói ứng dụng và các phụ thuộc của nó, giúp đảm bảo tính đồng nhất và di động khi triển khai ứng dụng trên các môi trường khác nhau.
- Apache Cassandra: Apache Cassandra là một hệ thống quản lý cơ sở dữ liệu phân tán mã nguồn mở được thiết kế để lưu trữ dữ liệu phân tán trên nhiều máy tính và các khu vực địa lý. Cassandra hỗ trợ việc lưu trữ và truy xuất dữ liệu một cách hiệu quả và có khả năng mở rộng ngang tốt.
- Python - Truyền phát có cấu trúc Spark: Đây là một cách sử dụng Python để xây dựng ứng dụng dựa trên Apache Spark để xử lý và phân tích dữ liệu có cấu trúc. Python là một ngôn ngữ lập trình phổ biến trong cộng đồng khoa học dữ liệu và Spark cung cấp API cho Python để xử lý dữ liệu một cách dễ dàng.

+ Đặc điểm

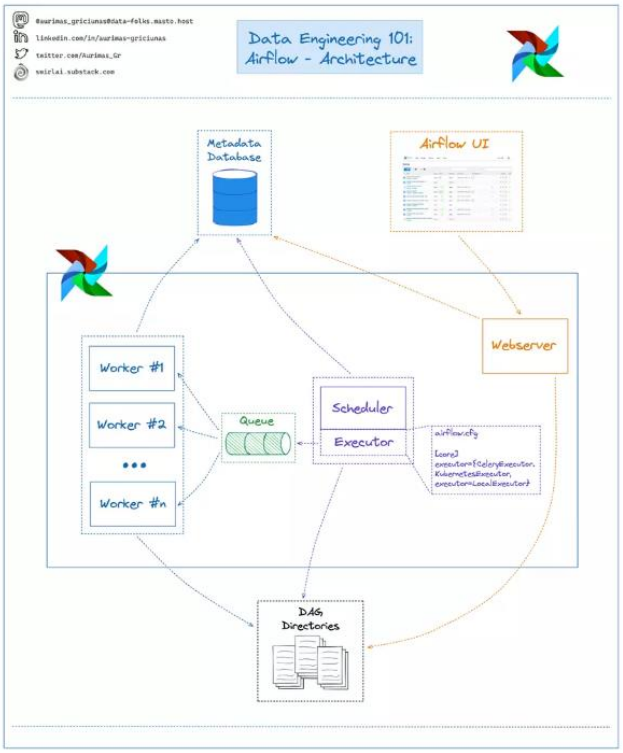
- Python:
  - Python là một ngôn ngữ lập trình mã nguồn mở, dễ đọc và dễ hiểu.
  - Đa năng: Python được sử dụng cho nhiều mục đích, bao gồm phát triển ứng dụng web, khoa học dữ liệu, máy học, và nhiều lĩnh vực khác.
  - Cộng đồng lớn: Python có cộng đồng phát triển và hỗ trợ lớn, có nhiều thư viện và framework sẵn sàng.
- API (Application Programming Interface):
  - Giao diện lập trình ứng dụng: API là tập hợp các quy tắc và hướng dẫn cho phép các ứng dụng giao tiếp với nhau.
  - Hỗ trợ tích hợp: API cho phép các ứng dụng và dịch vụ khác nhau tương tác và chia sẻ dữ liệu một cách dễ dàng.
- Apache Airflow:
  - Quản lý luồng công việc: Apache Airflow là một công cụ quản lý và lên lịch công việc phức tạp.

- Điều khiển công việc: Airflow cho phép bạn xác định các quy trình, lên lịch và theo dõi chúng.

- Apache Kafka:
  - Hệ thống truyền phát dữ liệu: Kafka là một hệ thống truyền phát dữ liệu phân tán, cho phép xử lý và truyền tải dữ liệu và sự kiện từ nhiều nguồn đến nhiều đích.
  - Bền bỉ và mở rộng: Kafka là một hệ thống bền bỉ và có khả năng mở rộng ngang tốt.
- Apache Spark:
  - Xử lý dữ liệu phân tán: Spark là một framework xử lý dữ liệu phân tán, có khả năng xử lý dữ liệu nhanh chóng và hiệu quả.
  - Hỗ trợ đa ngôn ngữ: Spark hỗ trợ nhiều ngôn ngữ lập trình và cung cấp thư viện mạnh mẽ cho xử lý dữ liệu.
- Apache Cassandra:
  - Cơ sở dữ liệu phân tán: Cassandra là hệ thống quản lý cơ sở dữ liệu phân tán với khả năng mở rộng và độ bền.
  - Hỗ trợ độ bền: Cassandra hỗ trợ độ bền và khả năng phục hồi dữ liệu, phù hợp cho các ứng dụng có yêu cầu cao về độ tin cậy.
- Docker:
  - Đóng gói ứng dụng: Docker cho phép đóng gói ứng dụng và các phụ thuộc của nó vào các container độc lập, giúp đảm bảo tính đồng nhất và di động khi triển khai.
  - Tích hợp dễ dàng: Docker có khả năng tích hợp dễ dàng với các công cụ và hệ thống khác, tạo môi trường ứng dụng cô lập.

+ Cấu trúc

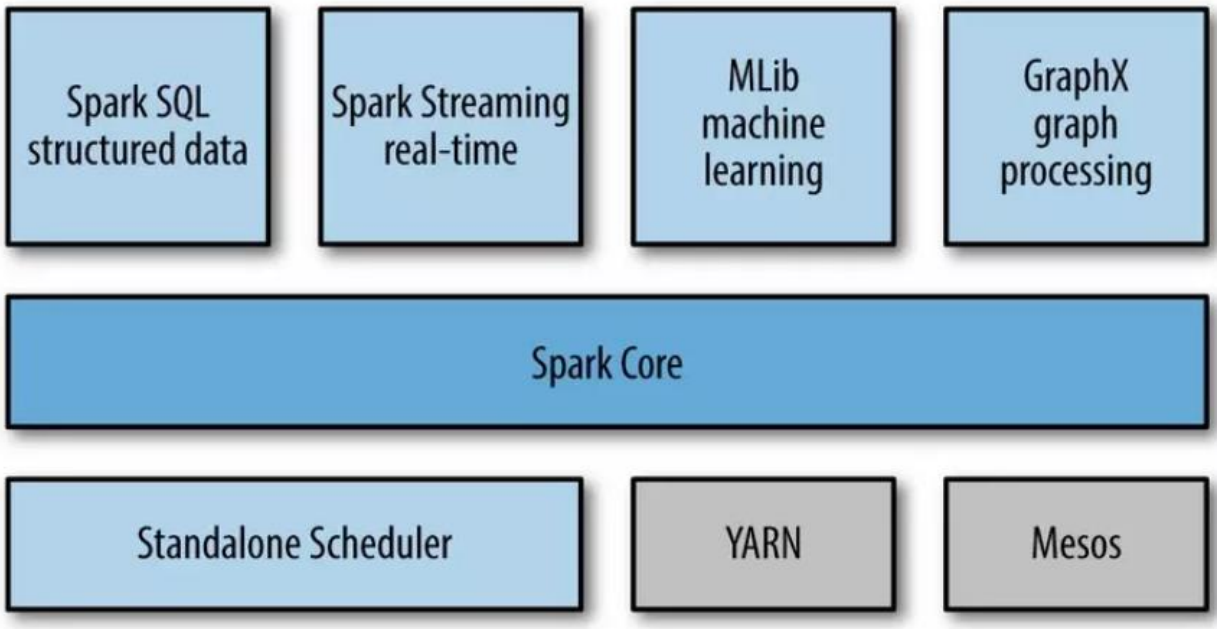
- Apache Airflow



Hình 2 Cấu trúc Apache Airflow

- Scheduler: Một trong những thành phần quan trọng nhất, xử lý cả việc kích hoạt workflow đã lên lịch và gửi Task cho executor để chạy.
- Executor các thành phần được sử dụng để chạy các Task, chúng có thể chạy trực tiếp trên Scheduler hoặc thực thi trên các Worker, tùy vào cách cấu hình
- WebServer: Chứa UI dạng Web được sử dụng để quản lý các Task, DagRun hoặc các giá trị của hệ thống

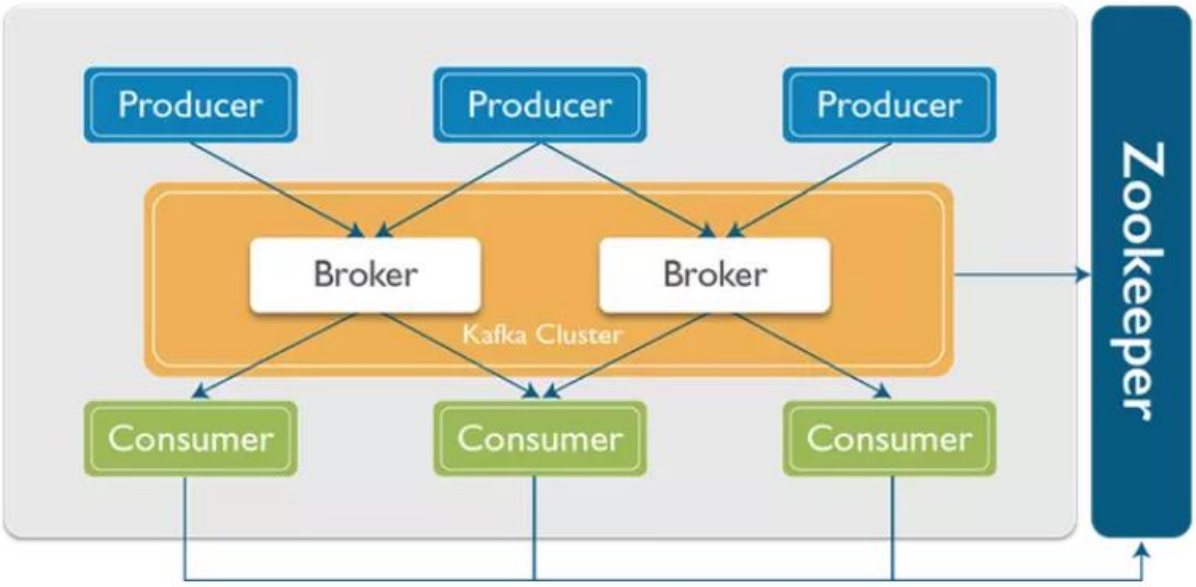
- Apache Spark



Hình 3 kiến trúc của Apache Spark

- Thành phần trung của Spark là Spark Core: cung cấp những chức năng cơ bản nhất của Spark như lập lịch cho các tác vụ, quản lý bộ nhớ, fault recovery, tương tác với các hệ thống lưu trữ…Đặc biệt, Spark Core cung cấp API để định nghĩa RDD (Resilient Distributed DataSet) là tập hợp của các item được phân tán trên các node của cluster và có thể được xử lý song song.
- Spark có thể chạy trên nhiều loại Cluster Managers như Hadoop YARN, Apache Mesos hoặc trên chính cluster manager được cung cấp bởi Spark được gọi là Standalone Scheduler.
- Spark SQL cho phép truy vấn dữ liệu cấu trúc qua các câu lệnh SQL. Spark SQL có thể thao tác với nhiều nguồn dữ liệu như Hive tables, Parquet, và JSON.
- Spark Streaming cung cấp API để dễ dàng xử lý dữ liệu stream,
- MLlib Cung cấp rất nhiều thuật toán của học máy như: classification, regression, clustering, collaborative filtering…
- GraphX là thư viện để xử lý đồ thị.

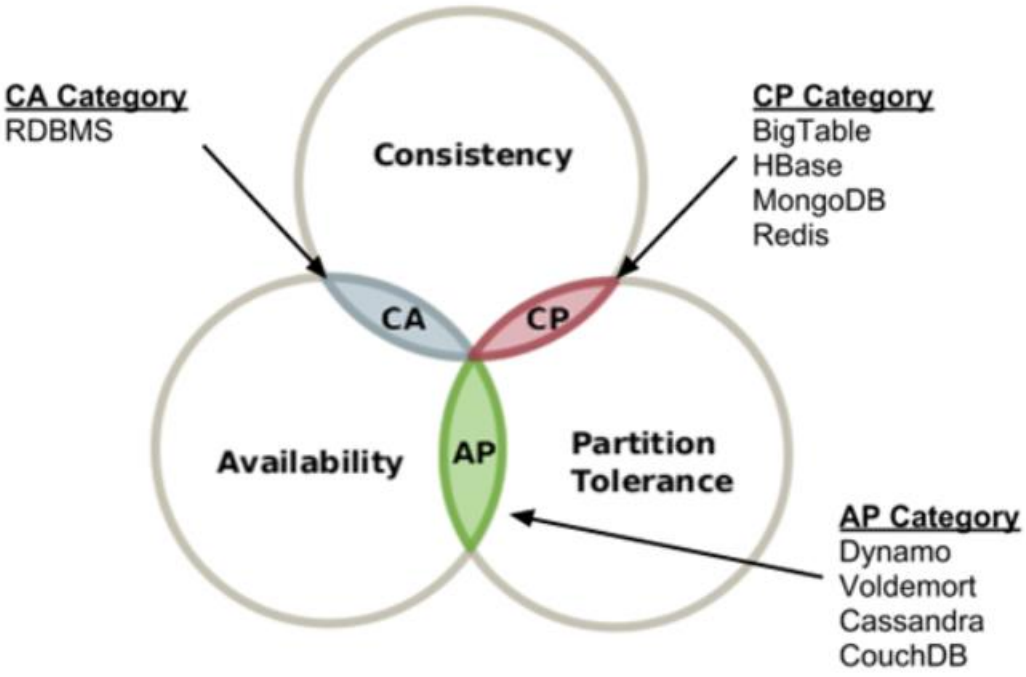
• Apache Kafka



Hình 4 Cấu trúc của Apache Kafka

- Producer: Một producer có thể là bất kì ứng dụng nào có chức năng publish message vào một topic.
- Messages: Messages đơn thuần là byte array và developer có thể sử dụng chúng để lưu bất kì object với bất kì format nào - thông thường là String, JSON và Avro
- Topic: Một topic là một category hoặc feed name nơi mà record được publish.
- Partitions: Các topic được chia nhỏ vào các đoạn khác nhau, các đoạn này được gọi là partition
- Consumer: Một consumer có thể là bất kì ứng dụng nào có chức năng subscribe vào một topic và tiêu thụ các tin nhắn.
- Broker: Kafka cluster là một set các server, mỗi một set này được gọi là 1 broker
- Zookeeper: được dùng để quản lý và bố trí các broker.

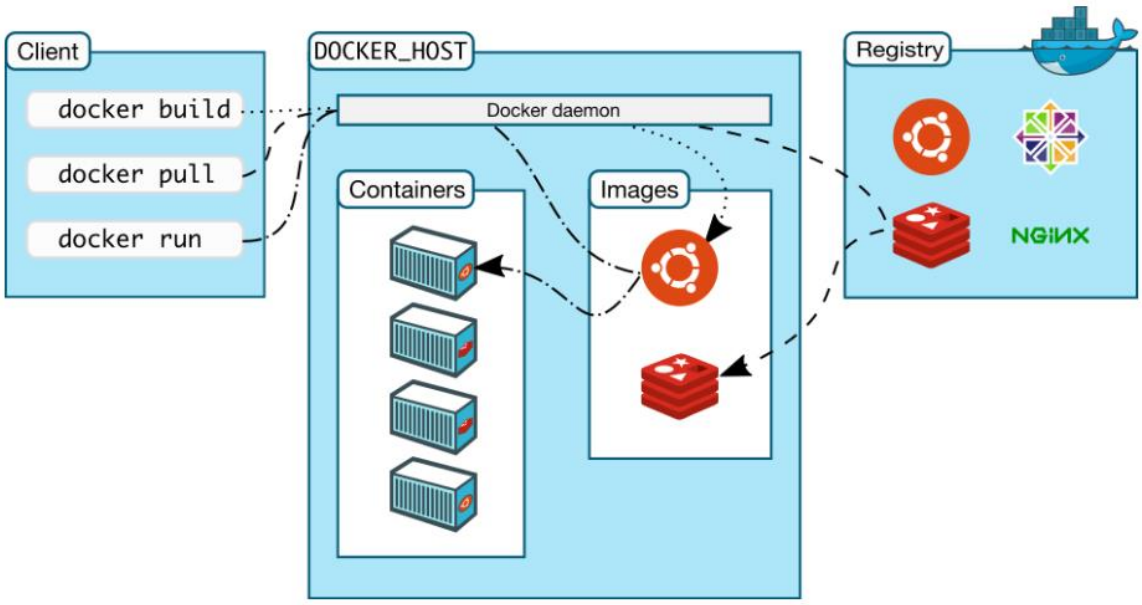
• Apache Cassandra



Hình 5 Cấu trúc của Apache Kafka

- Node : Một máy tính hoặc một máy ảo khởi chạy một Cassandra instance. Các node trong cluster có vai trò như nhau, không có node nào làm node chính
- Rack : Nhóm một hoặc nhiều nodes
- Data Center : Nhóm một hoặc nhiều Rack
- Cluster : Bao gồm một hoặc nhiều Data Center trong đó gồm một hay nhiều nodes kết nối với nhau, Dữ liệu của toàn hệ thống sẽ được tự động phân chia đến các nodes trong cluster
- Gossip Protocol : Giao thức truyền thông giữa các nodes
- Memtable và SSTable : Không gian bộ nhớ và không gian đĩa.

- Docker



Hình 6 Kiến trúc docker

- Docker Daemon: Docker daemon chạy trên các máy host. Người dùng sẽ không tương tác trực tiếp với các daemon, mà thông qua Docker Client.
- Docker Client: Là giao diện người dùng của Docker, nó cung cấp cho người dùng giao diện dòng lệnh và thực hiện phản hồi với các Docker daemon.
- Docker images: Là một template chỉ cho phép đọc, ví dụ một image có thể chứa hệ điều hành Ubuntu và web app. Images được dùng để tạo Docker container. Docker cho phép chúng ta build và cập nhật các image có sẵn một cách cơ bản nhất, hoặc bạn có thể download Docker images của người khác.
- Docker Container: Docker container có nét giống với các directory. Một Docker container giữ mọi thứ chúng ta cần để chạy một app. Mỗi container được tạo từ Docker image. Docker container có thể có các trạng thái run, started, stopped, moved và deleted.

1.2. Nguyên lý hoạt động

Nguyên lý hoạt động của quy trình này có thể được mô tả như sau:

- Nhận dữ liệu từ API: Quá trình bắt đầu bằng việc nhận dữ liệu từ một API bên ngoài, có thể là dữ liệu từ các nguồn như các ứng dụng web hoặc dịch vụ trực tuyến khác. Dữ liệu này có thể là thông tin từ cảm biến, dữ liệu thời tiết, dữ liệu giao dịch, và nhiều loại dữ liệu khác.
- Chạy tập lệnh được lên lịch với Airflow: Dữ liệu sau khi được nhận sẽ được xử lý bằng các tập lệnh đã được lên lịch trước đó sử dụng Apache Airflow.

Airflow cho phép định nghĩa và quản lý các công việc cụ thể, xác định luồng công việc và xác định lịch trình thực hiện chúng.

- Gửi dữ liệu tới Kafka: Khi dữ liệu đã được xử lý, nó sẽ được gửi tới Apache Kafka, một hệ thống truyền phát dữ liệu phân tán. Kafka sẽ đảm bảo dữ liệu được truyền đi một cách bền bỉ và có khả năng lưu trữ tạm thời.
- Sử dụng bằng Spark: Dữ liệu từ Kafka sau đó sẽ được sử dụng và xử lý bằng Apache Spark, một framework xử lý dữ liệu phân tán. Spark cung cấp các công cụ và thư viện mạnh mẽ để xử lý và phân tích dữ liệu, bao gồm xử lý batch và xử lý thời gian thực (streaming).
- Ghi vào Cassandra: Cuối cùng, sau khi dữ liệu đã được xử lý và phân tích bằng Spark, kết quả có thể được ghi vào Apache Cassandra, một hệ thống quản lý cơ sở dữ liệu phân tán. Cassandra đảm bảo tính bền bỉ và khả năng mở rộng cho dữ liệu đã được lưu trữ.

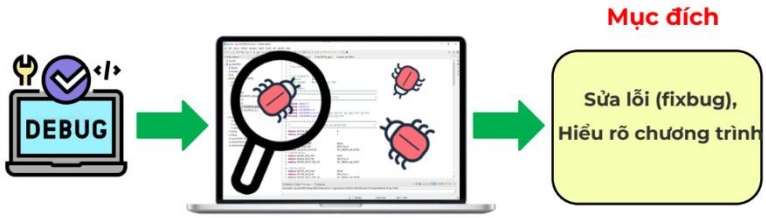
=>Toàn bộ quy trình này cho phép thu thập, xử lý, lưu trữ và truy cập dữ liệu một cách hiệu quả và có khả năng mở rộng, phù hợp cho các ứng dụng đòi hỏi xử lý lượng dữ liệu lớn và đáng tin cậy.

1.3. Nguyên lý dịch & gỡ rối

+ Bug được hiểu là thuật ngữ dùng để chỉ các lỗi kỹ thuật có thể xảy ra trong quá trình thiết kế và vận hành các chương trình lập trình.

+ Bug khiến cho phần mềm, ứng dụng không thực thi được hoặc thực thi sai.

+ Debug là quá trình tìm kiếm và phát hiện nguyên nhân gây ra lỗi



+ Các tùy chọn với chế độ debug

- *Debugging API Data Ingestion (Gỡ lỗi Nhận dữ liệu từ API):*
  - In-Memory Logging: Dữ liệu nhận được từ API có thể được ghi lại trong bộ nhớ để dễ dàng kiểm tra và gỡ lỗi.
  - Data Validation: Kiểm tra tính chính xác của dữ liệu đầu vào và gửi thông báo lỗi nếu có sự cố.
- Debugging Airflow Workflow (Gỡ lỗi Luồng công việc với Airflow):
  - Logging and Monitoring: Sử dụng hệ thống log và giám sát để theo dõi tiến trình của các công việc được lên lịch, xác định các công việc gặp vấn đề và theo dõi lịch sử thực thi.
  - Manual Intervention Points: Tạo điểm can thiệp thủ công trong luồng công việc để cho phép kiểm tra và gỡ lỗi từng bước.
- Debugging Kafka Data Transmission (Gỡ lỗi Truyền dữ liệu tới Kafka):
  - Data Inspection: Kiểm tra dữ liệu trước khi nó được gửi tới Kafka để đảm bảo tính chính xác và định dạng.
  - Kafka Monitoring: Sử dụng công cụ giám sát Kafka để theo dõi luồng dữ liệu và xác định sự cố nếu có.
- Debugging Spark Data Processing (Gỡ lỗi Xử lý dữ liệu bằng Spark):

Logging và Tracing: Sử dụng các hệ thống log và truy vết để theo dõi dữ liệu và tiến trình xử lý trong Spark.

  - Unit Testing: Phát triển các bài kiểm tra đơn vị để xác định xem các phần xử lý dữ liệu hoạt động đúng cách.
  - Spark UI: Sử dụng giao diện người dùng Spark để xem xét thông tin về quá trình xử lý và đánh giá hiệu suất.
- Debugging Cassandra Data Storage (Gỡ lỗi Lưu trữ dữ liệu trong Cassandra):
  - Data Validation: Kiểm tra dữ liệu trước khi ghi vào Cassandra để đảm bảo định dạng và tính chính xác.

- Cassandra Monitoring: Sử dụng giám sát Cassandra để theo dõi việc lưu trữ dữ liệu và xác định sự cố nếu có.

2. BÀI THÍ NGHIỆM THỰC HÀNH

2.1. Mục đích - Yêu cầu

2.1.1. Mục đích

+ Giúp sinh viên làm quen với các công nghệ hiện đại, nắm rõ cách thức tạo một chương trình thể hiện một luồng truyền dữ liệu trực tuyến.

+ Nhận dữ liệu từ API, chạy tập lệnh được lên lịch với Airflow, gửi dữ liệu tới Kafka và sử dụng bằng Spark, sau đó ghi vào Cassandra

+ Sử dụng và cài đặt các công nghệ Python, API, Apache Airflow, Apache Kafka, Apache Spark, Apache Cassandra, Docker

+ Khả năng sử dụng vào cài đặt phần mềm trên môi trường Linux, cách thực hiện sửa lỗi hệ thống

+ Làm quen với ảo hóa ở mức container sử dụng Docker

+ Biết cách đọc thư viện được dựng sẵn

+ Biết các tạo một đường truyền dữ liệu có cấu trúc Spark

2.1.2. Yêu cầu:

- Nắm vững kiến thức được giới thiệu, tự tìm hiểu thêm nhiều chức năng khác
- Hoàn thành bài code truyền dữ liệu có cấu trúc Spark theo yêu cầu của giảng viên

2.1.3. Thời gian thực hiện

+ Thời gian mỗi buổi thực hành là từ 3-4 giờ, chia làm 10-12 nhóm nhỏ, mỗi nhóm 2 sinh viên/1 máy tính. Sinh viên tìm hiểu cơ sở lý thuyết ngắn gọn và các bước thực hiện có thể thao tác dễ dàng.

2.2. Chuẩn bị

2.2.1. Danh mục thiết bị thực hành

+ Phần cứng: 01 máy tính. Tất cả được được đồng bộ theo số thứ tự từ 1-20

+ Phần mềm: Apache Airflow, Apache Kafka, Apache Spark, Apache Cassandra, Docker.

Danh mục thiết bị thực hành phòng lab

Stt	Tên thiết bị	Số lượng	Thông số kỹ thuật	Vai trò
1	Máy tính để bàn	01	- Intel Core i3-2100 3.1 Ghz/3M Cache - 4GB DDR3 (nâng cấp lên ram 8GB thêm 300K) -SSD 120G Gb chuẩn SATA 3 - 6Gb/s Ổ SSD	-Tính toán, viết chương trình, chạy chương trình

2.3. Nội dung

2.3.1. Các bước thực hiện

Nội dung 1: tạo luồng truyền dữ liệu trực tuyến và bao gồm nhiều công nghệ hiện đại trong xử lý dữ liệu (Nhận dữ liệu từ API, chạy tập lệnh được lên lịch với Airflow, gửi dữ liệu tới Kafka và sử dụng bằng Spark, sau đó ghi vào Cassandra)

- Chúng ta sẽ sử dụng dữ liệu từ trang web ***randomuser.me*** để thực hiện project này
- Bên cạnh đó với project này có thể sử dụng một trong 2 hệ điều hành Windows hoặc Linux (Ubuntu) để cài đặt

Bước 1: Tạo một project mới "PROJECT1"

Bước 2: Tạo và gõ lệnh trong file stream_to_kafka.py

```
"""
```

```
Gets the random user API data and writes the data to a Kafka topic every 10
seconds
"""
import requests
import json
import time
from kafka import KafkaProducer
def create_response_dict(url: str="https://randomuser.me/api/?results=1") ->
dict:
    """
    Creates the results JSON from the random user API call
    """
    response = requests.get(url)
    data = response.json()
    results = data["results"][0]
    return results
def create_final_json(results: dict) -> dict:
    """
    Creates the final JSON to be sent to Kafka topic only with necessary keys
    """
    kafka_data = {}
    kafka_data["full_name"] = f"{results['name']['title']}. {results['name']['first']}
{results['name']['last']}"
    kafka_data["gender"] = results["gender"]
    kafka_data["location"] = f"{results['location']['street']['number']},
{results['location']['street']['name']}"
    kafka_data["city"] = results['location']['city']
    kafka_data["country"] = results['location']['country']
    kafka_data["postcode"] = int(results['location']['postcode'])
    kafka_data["latitude"] = float(results['location']['coordinates']['latitude'])
    kafka_data["longitude"] = float(results['location']['coordinates']['longitude'])
    kafka_data["email"] = results["email"]
```

```
    return kafka_data
def create_kafka_producer():
    """
    Creates the Kafka producer object
    """
    return KafkaProducer(bootstrap_servers=['kafka1:19092', 'kafka2:19093',
'kafka3:19094'])
def start_streaming():
    """
    Writes the API data every 10 seconds to Kafka topic random_names
    """
    producer = create_kafka_producer()
    results = create_response_dict()
    kafka_data = create_final_json(results)
    end_time = time.time() + 120 # the script will run for 2 minutes
    while True:
        if time.time() > end_time:
            break
        producer.send("random_names", json.dumps(kafka_data).encode('utf-8'))
        time.sleep(10)
if __name__ == "__main__":
    start_streaming()
```

Bước 3: tạo và gõ lệnh trong file stream_to_kafka_dag.py

```
from datetime import timedelta
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
from stream_to_kafka import start_streaming
start_date = datetime(2018, 12, 21, 12, 12)
default_args = {
```

```
    'owner': 'airflow',
    'start_date': start_date,
    'retries': 1,
    'retry_delay': timedelta(seconds=5)
}
with DAG('random_people_names', default_args=default_args,
schedule_interval='0 1 * * *', catchup=False) as dag:
    data_stream_task = PythonOperator(
    task_id='kafka_data_stream',
    python_callable=start_streaming,
    dag=dag,
    )
    data_stream_task
```

Bước 4: tạo và gõ lệnh trong file spark_streaming.py

```
import logging
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,FloatType,IntegerType,StringType
from pyspark.sql.functions import from_json,col
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s:%(funcName)s:%(levelname)s:%(message)s')
```

```
logger = logging.getLogger("spark_structured_streaming")
def create_spark_session():
    """
    Creates the Spark Session with suitable configs.
    """
    try:
        # Spark session is established with cassandra and kafka jars. Suitable
versions can be found in Maven repository.
        spark = SparkSession \
                .builder \
                .appName("SparkStructuredStreaming") \
                .config("spark.jars.packages", "com.datastax.spark:spark-cassandra-
connector_2.12:3.0.0,org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0") \
                .config("spark.cassandra.connection.host", "cassandra") \
                .config("spark.cassandra.connection.port","9042")\
                .config("spark.cassandra.auth.username", "cassandra") \
                .config("spark.cassandra.auth.password", "cassandra") \
                .getOrCreate()
        spark.sparkContext.setLogLevel("ERROR")
        logging.info('Spark session created successfully')
    except Exception:
        logging.error("Couldn't create the spark session")
    return spark

def create_initial_dataframe(spark_session):
    """
    Reads the streaming data and creates the initial dataframe accordingly.
    """
    try:
        # Gets the streaming data from topic random_names
        df = spark_session \
            .readStream \
```

```
            .format("kafka") \
            .option("kafka.bootstrap.servers",
"kafka1:19092,kafka2:19093,kafka3:19094") \
            .option("subscribe", "random_names") \
            .option("delimeter",",") \
            .option("startingOffsets", "earliest") \
            .load()
        logging.info("Initial dataframe created successfully")
    except Exception as e:
        logging.warning(f"Initial dataframe couldn't be created due to exception:
{e}")
    return df
def create_final_dataframe(df, spark_session):
    """
    Modifies the initial dataframe, and creates the final dataframe.
    """
    schema = StructType([
                StructField("full_name",StringType(),False),
                StructField("gender",StringType(),False),
                StructField("location",StringType(),False),
                StructField("city",StringType(),False),
                StructField("country",StringType(),False),
                StructField("postcode",IntegerType(),False),
                StructField("latitude",FloatType(),False),
                StructField("longitude",FloatType(),False),
                StructField("email",StringType(),False)
            ])
    df = df.selectExpr("CAST(value AS
STRING)").select(from_json(col("value"),schema).alias("data")).select("data.*")
    print(df)
    return df
```

```
def start_streaming(df):
    """
    Starts the streaming to table spark_streaming.random_names in cassandra
    """
    logging.info("Streaming is being started...")
    my_query = (df.writeStream
                .format("org.apache.spark.sql.cassandra")
                .outputMode("append")
                .options(table="random_names", keyspace="spark_streaming")\
                .start())

    return my_query.awaitTermination()
def write_streaming_data():
    spark = create_spark_session()
    df = create_initial_dataframe(spark)
    df_final = create_final_dataframe(df, spark)
    start_streaming(df_final)
if __name__ == '__main__':
    write_streaming_data()
```

Bước 5: Tạo docker file

- Trước hết chúng ta có thể cài đặt Docker dựa theo link: https://docs.docker.com/get-docker/
- Sau khi cài đặt. Trong folder project, chúng ta sẽ tạo hai file docker: (1) cho cài đặt Airlfow và (2) cho việc cài đặt các module cần thiết cho chương trình
- File thứ 1 chúng ta đặt tên là: ***docker-compose-LocalExecutor.yml*** với nội dung như dưới đây

```
version: '3.7'
services:
    postgres:
        image: postgres:9.6
        environment:
```

```
      - POSTGRES_USER=airflow
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    logging:
      options:
        max-size: 10m
        max-file: "3"
    networks:
      - airflow-kafka

  webserver:
    image: puckel/docker-airflow:1.10.9
    restart: always
    depends_on:
      - postgres
    environment:
      - LOAD_EX=n
      - EXECUTOR=Local
    logging:
      options:
        max-size: 10m
        max-file: "3"
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./requirements.txt:/usr/local/airflow/requirements.txt
      # - ./plugins:/usr/local/airflow/plugins
    ports:
      - "8080:8080"
    command: webserver
    healthcheck:
      test: ["CMD-SHELL", "[ -f /usr/local/airflow/airflow-webserver.pid ]"]
      interval: 30s
      timeout: 30s
      retries: 3
    networks:
      - airflow-kafka

networks:
  airflow-kafka:
    external: true
```

- File thứ 2 chúng ta thực hiện viết docker-compose file với tên file là: **docker-compose.yml** với nội dung như sau:

```
version: '3'

services:
  zoo1:
    image: confluentinc/cp-zookeeper:7.3.2
```

```
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_SERVERS: zoo1:2888:3888
    networks:
      - kafka-network
      - airflow-kafka

  kafka1:
    image: confluentinc/cp-kafka:7.3.2
    ports:
      - "9092:9092"
      - "29092:29092"
    environment:
      KAFKA_ADVERTISED_LISTENERS:
INTERNAL://kafka1:19092,EXTERNAL://${DOCKER_HOST_IP:-
127.0.0.1}:9092,DOCKER://host.docker.internal:29092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,DOCKER:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 1
      KAFKA_LOG4J_LOGGERS:
"kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logge
r=INFO"
      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
    networks:
      - kafka-network
      - airflow-kafka

  kafka2:
    image: confluentinc/cp-kafka:7.3.2
    ports:
      - "9093:9093"
      - "29093:29093"
    environment:
      KAFKA_ADVERTISED_LISTENERS:
INTERNAL://kafka2:19093,EXTERNAL://${DOCKER_HOST_IP:-
127.0.0.1}:9093,DOCKER://host.docker.internal:29093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,DOCKER:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 2
```

```
      KAFKA_LOG4J_LOGGERS:
"kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logge
r=INFO"
      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
    networks:
      - kafka-network
      - airflow-kafka

  kafka3:
    image: confluentinc/cp-kafka:7.3.2
    ports:
      - "9094:9094"
      - "29094:29094"
    environment:
      KAFKA_ADVERTISED_LISTENERS:
INTERNAL://kafka3:19094,EXTERNAL://${DOCKER_HOST_IP:-
127.0.0.1}:9094,DOCKER://host.docker.internal:29094
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,DOCKER:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 3
      KAFKA_LOG4J_LOGGERS:
"kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logge
r=INFO"
      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
    networks:
      - kafka-network
      - airflow-kafka

  kafka-connect:
    image: confluentinc/cp-kafka-connect:7.3.2
    ports:
      - "8083:8083"
    environment:
      CONNECT_BOOTSTRAP_SERVERS: kafka1:19092,kafka2:19093,kafka3:19094
      CONNECT_REST_PORT: 8083
      CONNECT_GROUP_ID: compose-connect-group
      CONNECT_CONFIG_STORAGE_TOPIC: compose-connect-configs
      CONNECT_OFFSET_STORAGE_TOPIC: compose-connect-offsets
      CONNECT_STATUS_STORAGE_TOPIC: compose-connect-status
      CONNECT_KEY_CONVERTER: org.apache.kafka.connect.storage.StringConverter
      CONNECT_VALUE_CONVERTER: org.apache.kafka.connect.storage.StringConverter
      CONNECT_INTERNAL_KEY_CONVERTER:
"org.apache.kafka.connect.json.JsonConverter"
      CONNECT_INTERNAL_VALUE_CONVERTER:
"org.apache.kafka.connect.json.JsonConverter"
```

```
      CONNECT_REST_ADVERTISED_HOST_NAME: 'kafka-connect'
      CONNECT_LOG4J_ROOT_LOGLEVEL: 'INFO'
      CONNECT_LOG4J_LOGGERS:
'org.apache.kafka.connect.runtime.rest=WARN,org.reflections=ERROR'
      CONNECT_PLUGIN_PATH: '/usr/share/java,/usr/share/confluent-hub-components'
    networks:
      - kafka-network
      - airflow-kafka

  schema-registry:
    image: confluentinc/cp-schema-registry:7.3.2
    ports:
      - "8081:8081"
    environment:
      SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS:
kafka1:19092,kafka2:19093,kafka3:19094
      SCHEMA_REGISTRY_HOST_NAME: schema-registry
      SCHEMA_REGISTRY_LISTENERS: http://0.0.0.0:8081
    networks:
      - kafka-network
      - airflow-kafka

  kafka-ui:
    container_name: kafka-ui-1
    image: provectuslabs/kafka-ui:latest
    ports:
      - 8888:8080 # Changed to avoid port clash with akhq
    depends_on:
      - kafka1
      - kafka2
      - kafka3
      - schema-registry
      - kafka-connect
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS:
PLAINTEXT://kafka1:19092,PLAINTEXT_HOST://kafka1:19092
      KAFKA_CLUSTERS_0_SCHEMAREGISTRY: http://schema-registry:8081
      KAFKA_CLUSTERS_0_KAFKACONNECT_0_NAME: connect
      KAFKA_CLUSTERS_0_KAFKACONNECT_0_ADDRESS: http://kafka-connect:8083
      DYNAMIC_CONFIG_ENABLED: 'true'
    networks:
      - kafka-network
      - airflow-kafka

  spark:
    image: bitnami/spark:3
    container_name: spark_master
    ports:
```

```
      - 8085:8080
    environment:
      - SPARK_UI_PORT=8080
      - SPARK_MODE=master
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
    volumes:
      - ./:/home
      - spark-data:/opt/bitnami/spark/data
    networks:
      - airflow-kafka
      - kafka-network

  cassandra:
    image: cassandra:latest
    container_name: cassandra
    hostname: cassandra
    ports:
      - 9042:9042
    environment:
      - MAX_HEAP_SIZE=512M
      - HEAP_NEWSIZE=100M
      - CASSANDRA_USERNAME=cassandra
      - CASSANDRA_PASSWORD=cassandra
    volumes:
      - ./:/home
      - cassandra-data:/var/lib/cassandra
    networks:
      - airflow-kafka
      - kafka-network

volumes:
  cassandra-data:
  spark-data:

networks:
  kafka-network:
    driver: bridge
  airflow-kafka:
    external: true
```

Bước 6: Cài đặt và cấu hình Apache Airflow

// Chạy lệnh sau để sao chép repo cần thiết trên máy cục bộ của bạn

git clone https://github.com/dogukannulu/docker-airflow.git

// Sau khi nhân bản repo, chỉ chạy lệnh sau một lần

docker build --rm --build-arg AIRFLOW_DEPS="datadog,dask" --build-arg PYTHON_DEPS="flask_oauthlib>=0.9" -t puckel/docker-airflow .

/ Sau đó thay đổi tệp docker-compose-LocalExecutor.yml trong folder docker-airflow bằng tệp vừa tạo và thêm tệp requirements.txt vào thư mục. Điều này sẽ liên kết bộ chứa Airflow với bộ chứa Kafka và Spark và các mô-đun cần thiết sẽ tự động được cài đặt:*/*

docker-compose -f docker-compose-LocalExecutor.yml up –d

/* Bây giờ bạn có một Airflow container đang chạy và bạn có thể truy cập giao diện người dùng tại https://localhost:8080"*/

https://localhost:8080

Nội dung **requirement.txt** file:

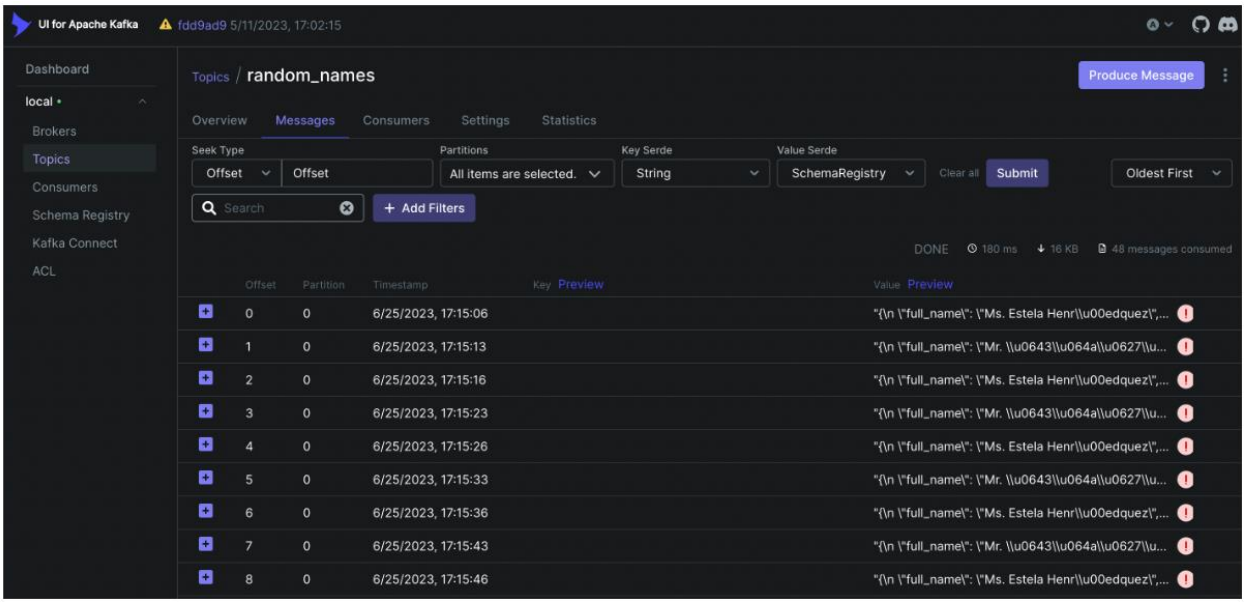
```
kafka==1.3.5
kafka-python==2.0.2
requests==2.27.1
json5==0.9.6
pyspark==3.4.1
```

Bước 7: Cài đặt và cấu hình Apache Airflow

/ docker-compose.yml sẽ tạo ra một cụm Kafka đa nút. Ta có thể định nghĩa hệ số sao chép là 3 vì có 3 nút (kafka1, kafka2, kafka3), có thể thấy giao diện người dùng Kafka trên localhost:8888. Gõ lênh chạy:*/*

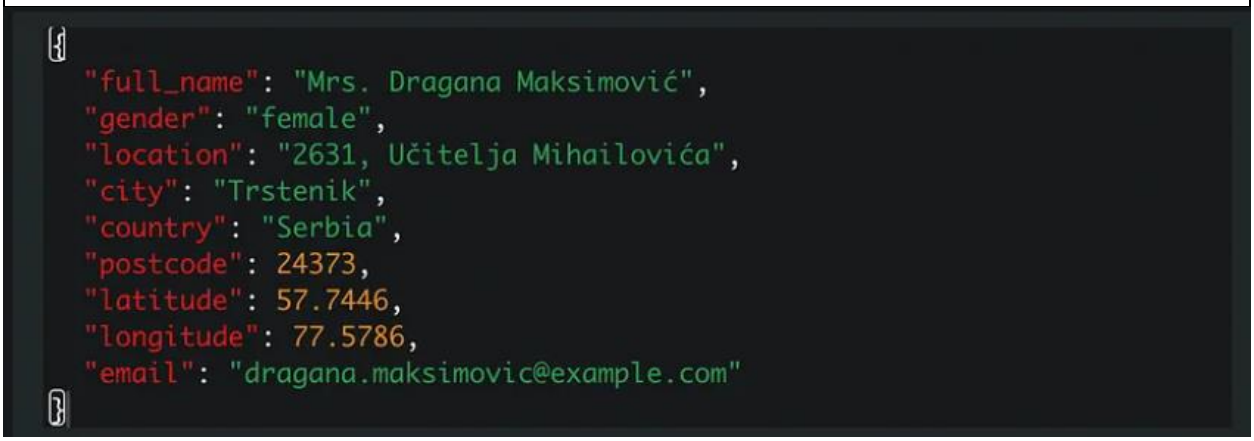
docker-compose up –d

Sau khi chạy lệnh và chờ một chút, chúng ta có thể truy cập Kafka-UI trên https://localhost:8888. Sau khi truy cập chúng ta có thể thấy Topic ở menu bên trái. Chúng ta có thể tạo một chủ đề mới với tên Random_names. Dựa vào số cụm Kafka đang hoạt động chúng ta thực hiện cài đặt cho Topic mới "replication factor" cùng số cluster. Sau khi thực thi chương trình ***stream_to_kafka.py*** *đầu tiên, chúng ta sẽ có thể xem dữ liệu đến như bên dưới.*



Chạy file stream_to_kafka.py để thực hiện gọi API lấy dữ liệu

/ Sau khi truy cập vào Kafka UI, Ta có thể tạo topic random_names. Sau đó, chúng ta có thể thấy các thông báo đến topic Kafka. Như ví dụ dưới đây*/*



Bước 8: Cấu hình và cài đặt Cassandra

/ docker-compose.yml cũng sẽ tạo một máy chủ Cassandra. Mọi biến env đều nằm trong docker-compose.yml và đã xác định chúng trong tập lệnh.*
Bằng cách chạy lệnh sau, chúng ta có thể truy cập vào máy chủ Cassandra:/*

docker exec -it cassandra /bin/bash

/ Sau khi truy cập bash, chúng ta có thể chạy lệnh sau để truy cập vào cqlsh cli.*/*

cqlsh -u cassandra -p cassandra

/ Sau đó, chúng ta có thể chạy các lệnh sau để tạo không gian khóa spark_streaming và bảng random_names:*/*

CREATE KEYSPACE spark_streaming WITH replication = {'class':'SimpleStrategy','replication_factor':1};

CREATE TABLE spark_streaming.random_names(full_name text primary key, gender text, location text, city text, country text, postcode int, latitude float, longitude float, email text);

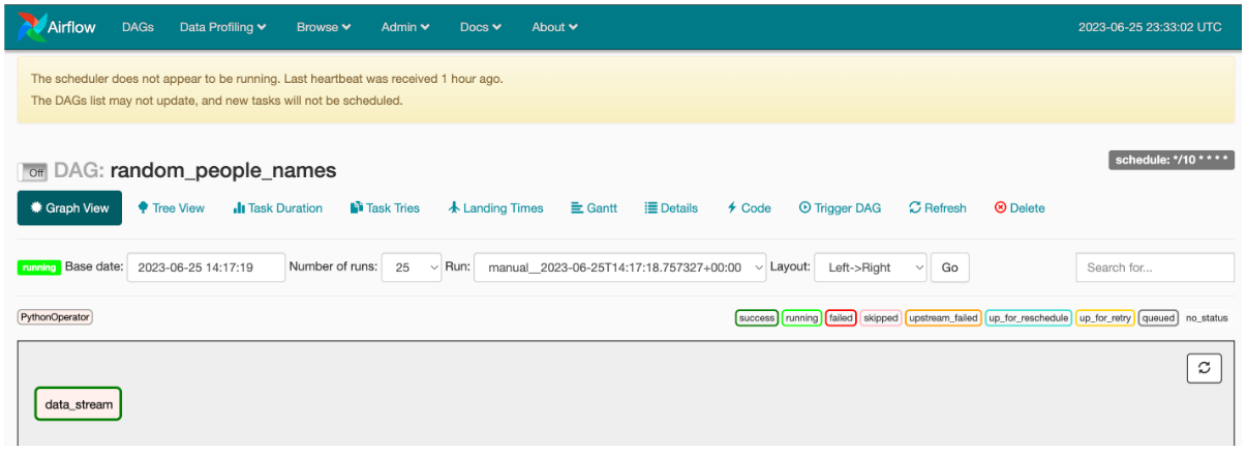
DESCRIBE spark_streaming.random_names;

```
CREATE TABLE spark_streaming.random_names (
    full_name text PRIMARY KEY,
    city text,
    country text,
    email text,
    gender text,
    latitude float,
    location text,
    longitude float,
    postcode int
) WITH additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
cassandra@cqlsh> select * from spark_streaming.random_names;

 full_name | city | country | email | gender | latitude | location | longitude | postcode
-----------+------+---------+-------+--------+----------+----------+-----------+----------
```

Bước 9: chạy DAGs

*/*Chúng ta nên di chuyển **stream_to_kafka.py** và **stream_to_kafka_dag.py** vào thư mục dags trong docker-airflow repo. Sau đó chúng ta có thể thấy nó random_people_names xuất hiện trong thư mục **dags**.*
Khi chúng ta chuyển nút OFF sang ON, chúng ta có thể thấy dữ liệu sẽ được gửi đến các chủ đề Kafka cứ sau 10 giây. Chúng ta có thể xem từ từ UI của Kafka./*



Bước 10: Cấu hình và cài đặt Spark

/* Trước hết chúng ta nên sao chép tập lệnh PySpark cục bộ vào container*/

docker cp spark_streaming.py spark_master:/opt/bitnami/spark/

/* *Sau đó, chúng ta nên truy cập vào Spark container và cài đặt các tệp JAR cần thiết trong thư mục jars.*/*

docker exec -it spark_master /bin/bash

/ Chúng ta nên chạy các lệnh sau để cài đặt các tệp JAR cần thiết cho Spark phiên bản 3.3.0:*/*

cd jars

curl -O https://repo1.maven.org/maven2/com/datastax/spark/spark-cassandra-connector_2.12/3.3.0/spark-cassandra-connector_2.12-3.3.0.jar

curl -O https://repo1.maven.org/maven2/org/apache/spark/spark-sql-kafka-0-10_2.13/3.3.0/spark-sql-kafka-0-10_2.13-3.3.0.jar

/ Mặc dù dữ liệu API được gửi topic Kafka random_names thường xuyên, chúng ta có thể gửi ứng dụng PySpark và ghi dữ liệu topic vào bảng Cassandra:*/*

cd ..

spark-submit --master local[2] --jars /opt/bitnami/spark/jars/spark-sql-kafka-0-10_2.13-3.3.0.jar,/opt/bitnami/spark/jars/spark-cassandra-connector_2.12-3.3.0.jar spark_streaming.py

/ Sau khi chạy lệnh, chúng ta có thể thấy dữ liệu được điền vào bảng Cassandra*/*

2.3.2. Kết quả thực hiện

```
2023-06-25 22:10:40,706:create_spark_session:INFO:Spark session created successfully
2023-06-25 22:11:04,932:create_initial_dataframe:INFO:Initial dataframe created successfully
DataFrame[full_name: string, gender: string, location: string, city: string, country: string, postcode: int, latitude: float, longitude: float, email: string]
2023-06-25 22:11:08,820:start_streaming:INFO:Streaming is being started...
```

The logs of the script

TÀI LIỆU HƯỚNG DẪN THỰC HÀNH

[1] Tài liệu: https://github.com/dogukannulu/kafka_spark_structured_streaming

PHỤ LỤC

Phiếu báo cáo kết quả thực hành (Sinh viên)

Tên bài: ……...

Họ và tên sinh viên……Mã sinh viên……...

……..……...

Nhóm……..Lớp…… ..Ngày…..tháng…..năm……..

Giáo viên hướng dẫn……Ca thực tập…….

STT	Nội dung thực hành	Mức độ hoàn thành (%)	Thời gian hoàn thành	Đánh giá kết quả (100)
1				
2				
3				
Thảo luận sinh viên:				

Phiếu đánh giá kết quả thực hành (Giảng viên)

Tên bài: ……...

Họ và tên sinh viên……Mã sinh viên……...

Nhóm……..Lớp……Ngày…..tháng…..năm……..

Giáo viên hướng dẫn……Ca thực tập…….

Thứ tự	Nội dung đánh giá	Điểm chuẩn	Yêu cầu	Ghi chú
Tổng điểm:				
Nhận xét giảng viên:				