

TẬP ĐOÀN CÔNG NGHIỆP - VIỄN THÔNG QUÂN ĐỘI

o0o



BÁO CÁO ASSIGNMENT GIAI ĐOẠN 1

CHỦ ĐỀ: XÂY DỰNG NỀN TẢNG DỮ LIỆU

VŨ HỮU SỸ

vuhuusyft1@gmail.com

Chương trình Viettel Digital Talent 2024


Lĩnh vực: Data Engineering

Mentor: NGUYỄN VIỆT TIẾN

Đơn vị: Ban CNTT

HÀ NỘI, 06/2024

Mục lục

Chương 1	Mô tả nền tảng dữ liệu	2
1.1	Giới thiệu chung về nền tảng	2
1.2	Các công nghệ sử dụng trong nền tảng dữ liệu	3
Chương 2	Triển khai nền tảng	4
2.1	Giới thiệu	4
2.2	Triển khai các thành phần	4
2.2.1	Apache Zookeeper	4
2.2.2	Apache Kafka	5
2.2.3	Apache NiFi	6
2.2.4	Apache Hadoop	7
2.2.5	Apache Spark	7
2.3	Link Source Code 	8
2.4	Xây dựng luồng dữ liệu	8
2.4.1	Luồng đẩy dữ liệu vào Kafka Topic	8
2.4.2	Luồng dữ liệu trong NiFi	9
2.5	Xử lý dữ liệu bằng Apache Spark	15

Danh sách hình vẽ

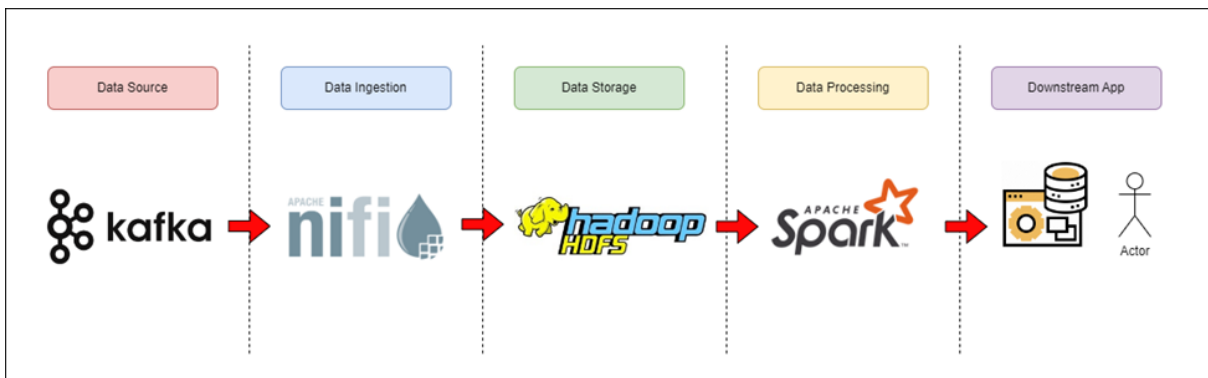
1.1	Các thành phần trong nền tảng dữ liệu	2
2.1	Kết quả trên Kafka UI	8
2.2	Luồng 1	9
2.3	Cấu hình của ListFile	10
2.4	Cấu hình của FetchFile	10
2.5	Cấu hình của PutHDFS	11
2.6	Kết quả	11
2.7	Luồng 2	12
2.8	Cấu hình của ConsumeKafka_2_6	12
2.9	Cấu hình của CSVReader	13
2.10	Cấu hình của ParquetReader	13
2.11	Cấu hình của MergeRecord	14
2.12	Cấu hình của ParquetRecordSetWriter	14
2.13	Kết quả	15
2.14	Khởi tạo SparkSession	15
2.15	Đọc file	16
2.16	Đổi tên cột	16
2.17	Xử lý cột timestamp	17
2.18	Join	17
2.19	Kết quả	18
2.20	Lưu kết quả vào HDFS	18
2.21	File kết quả trong HDFS	18

Chương 1

Mô tả nền tảng dữ liệu

1.1 Giới thiệu chung về nền tảng

Nền tảng dữ liệu được triển khai bằng cách sử dụng các công nghệ đã được đào tạo trong chương trình Viettel Digital Talent 2024. Hình 1.1 mô tả các thành phần có trong nền tảng.



Hình 1.1: Các thành phần trong nền tảng dữ liệu

Nền tảng bao gồm các lớp:

1. Data Source.
2. Data Ingestion.
3. Data Storage.
4. Data Processing.
5. Downstream App.

1.2 Các công nghệ sử dụng trong nền tảng dữ liệu

Các công nghệ sử dụng bao gồm: Apache Kafka, Apache NiFi, Apache Hadoop, Apache Spark và Docker.

Vai trò của các công nghệ trong nền tảng:

- Apache Kafka: Sử dụng Kafka Producer đọc dữ liệu từng dòng trong file "log_action.csv" sau đó đẩy vào Kafka Topic "vdt2024".
- Apache NiFi: Sử dụng các Processors để kéo dữ liệu từ Kafka Topic "vdt2024" và lưu dữ liệu vào HDFS với đường dẫn: "/raw_zone/fact/activity", lưu trữ dữ liệu dưới dạng parquet. Sử dụng các Processors để lưu trữ file "danh_sach_sv_de.csv" xuống HDFS.
- Apache Hadoop: Lưu trữ dữ liệu.
- Apache Spark: Đọc dữ liệu từ HDFS và xử lý dữ liệu theo yêu cầu là "Đưa ra tổng số file được tương tác hàng ngày theo mỗi loại activity mà sinh viên đó thực hiện và lưu ra 1 file output".
- Docker: Sử dụng docker-compose để chạy và liên kết các công nghệ trong nền tảng.

Chương 2

Triển khai nền tảng

Chương 2 sẽ mô tả chi tiết cách triển khai các thành phần trong nền tảng bao gồm cách thức triển khai cụm Kafka, cụm NiFi, cụm Hadoop, cụm Spark, ... và quy trình thực hiện các bước.

2.1 Giới thiệu

Nền tảng được chạy trên Docker, sử dụng các images của Confluent Inc., Provecus, The Apache Software Foundation và Big Data Europe. Tất cả các services cần thiết đều được định nghĩa và cấu hình trong file "docker-compose.yml".

2.2 Triển khai các thành phần

2.2.1 Apache Zookeeper

Đầu tiên chúng ta chạy service Zookeeper sử dụng image confluentinc/cp-zookeeper từ Confluent Platform với vai trò quản lý metadata, điều phối, leader election, ... cho cụm Kafka và cụm NiFi.

Cấu hình của service Zookeeper:

- **ports: "2181:2181"**: Để Zookeeper lắng nghe trên cổng 2181.
- **ZOOKEEPER_CLIENT_PORT: 2181**: Đặt cổng client cho Zookeeper là 2181.
- **ZOOKEEPER_TICK_TIME: 2000** Thiết lập tick time cho Zookeeper là 2000.

Zookeeper là một thành phần quan trọng để đảm bảo rằng cụm Kafka, NiFi hoạt động một cách ổn định và có thể mở rộng. Cấu hình chi tiết của Zookeeper xem từ dòng 5 \rightarrow 24 trong: `docker-compose.yml`.

2.2.2 Apache Kafka

Sau khi cấu hình xong service Zookeeper, chúng ta sẽ cấu hình các service của Kafka để chạy một cụm Kafka gồm hai Kafka Brokers (`kafka0` và `kafka1`) và một giao diện quản lý Kafka (`kafka-ui`).

Chi tiết cách cấu hình từng service:

1. `kafka0` và `kafka1`:

- **image**: Cả hai service đều sử dụng image `confluentinc/cp-kafka` từ Confluent Platform.
- **ports**: Mở các cổng 9092 (cho `kafka0`) và 9091 (cho `kafka1`) để truy cập từ bên ngoài.
- **Environment Variables**:
 - **KAFKA_ADVERTISED_LISTENERS**: Quảng bá các listener nội bộ (`kafka0:29092` và `kafka1:29091`), bên ngoài (`localhost:9092` và `localhost:9091`).
 - **KAFKA_LISTENER_SECURITY_PROTOCOL_MAP**: Chỉ định giao thức bảo mật cho các listener là PLAINTEXT.
 - **KAFKA_JMX_PORT**: Cấu hình cổng JMX là 9090.
 - **KAFKA_ZOOKEEPER_CONNECT**: Kết nối đến Zookeeper thông qua cổng 2181.
 - **KAFKA_BROKER_ID**: ID của Kafka Broker (1 cho `kafka0` và 2 cho `kafka1`).
 - **KAFKA_NUM_PARTITIONS**: Số lượng partition mỗi khi tạo một Topic mới mặc định là 2.
 - **KAFKA_AUTO_CREATE_TOPICS_ENABLE**: Đặt TRUE để tự động tạo Topic mới khi cần.

- **depends_on**: Đặt điều kiện phụ thuộc, đảm bảo container kafka0 và kafka1 chỉ khởi động khi service zookeeper ở trạng thái khỏe mạnh.

2. kafka-ui:

- **image**: Sử dụng image provectuslabs/kafka-ui:latest - giao diện web để quản lý và giám sát cụm Kafka.
- **ports**: Ánh xạ cổng 8080 trong container với cổng 8080 ở máy host để có thể truy cập giao diện web từ máy host.
- **Environment Variables**:
 - **KAFKA_CLUSTERS_0_NAME**: Đặt tên của cụm Kafka là vdt-kafka-cluster.
 - **KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS**: Danh sách các bootstrap servers của cụm Kafka, bao gồm kafka0 và kafka1 với các cổng nội bộ là 29092 và 29091.
 - **KAFKA_CLUSTERS_0_METRICS_PORT**: Cổng metrics của cụm Kafka là 9090.
- **depends_on**: Đặt điều kiện phụ thuộc, đảm bảo container kafka-ui chỉ khởi động khi cả service kafka0 và service kafka1 ở trạng thái khỏe mạnh.

Cấu hình chi tiết của cụm Kafka và Kafka UI xem từ dòng 26 → 120 trong: docker-compose.yml.

2.2.3 Apache NiFi

Triển khai cụm NiFi gồm hai nodes là nifi0, nifi1 được zookeeper quản lý.

Chi tiết cách cấu hình từng service:

- **image**: Sử dụng image apache/nifi của The Apache Software Foundation.
- **ports**: Cổng 8080 của container nifi0 và nifi1 lần lượt được ánh xạ với cổng 6980 và 6979 của máy host.
- **Environment Variables**:

- **NIFI_CLUSTER_IS_NODE**: Cấu hình node để tham gia vào cụm, giá trị là true.
 - **NIFI_CLUSTER_NODE_PROTOCOL_PORT**: Cổng giao thức của node, đặt là 8082.
 - **NIFI_ZK_CONNECT_STRING**: Kết nối đến zookeeper:2181.
 - **NIFI_ELECTION_MAX_WAIT**: Thời gian chờ tối đa cho bầu cử là 1 min.
- **depends_on**: Đảm bảo các container nifi0 và nifi1 chỉ khởi động khi service Zookeeper đang ở trạng thái khỏe mạnh.

Cấu hình chi tiết của cụm NiFi xem từ dòng 122 → 177 trong: docker-compose.yml.

2.2.4 Apache Hadoop

Chạy một cụm Hadoop gồm một namenode và hai datanode, có các service sau:

- namenode (image: bde2020/hadoop-namenode)
- datanode1, datanode2 (image: bde2020/hadoop-datanode)
- resourcemanager (image: bde2020/hadoop-resourcemanager)
- nodemanager1 (image: bde2020/hadoop-nodemanager)
- historyserver (image: bde2020/hadoop-historyserver)

Cấu hình chi tiết của cụm Hadoop xem từ dòng 179 → 267 trong: docker-compose.yml.


2.2.5 Apache Spark

Chạy một cụm Spark gồm một master và một workers có các service sau:

- spark-master (image: bde2020/spark-master)
- spark-worker-1 (image: bde2020/spark-worker)
- jupyter-notebook (image: huusy/jupyter-notebook): để có thể coding với Spark trên Jupyter Notebook.

Cấu hình chi tiết của cụm Spark xem từ dòng 269 → 318 trong: `docker-compose.yml`.

2.3 Link Source Code

Link Github : <https://github.com/vuhuusy/data-flow>

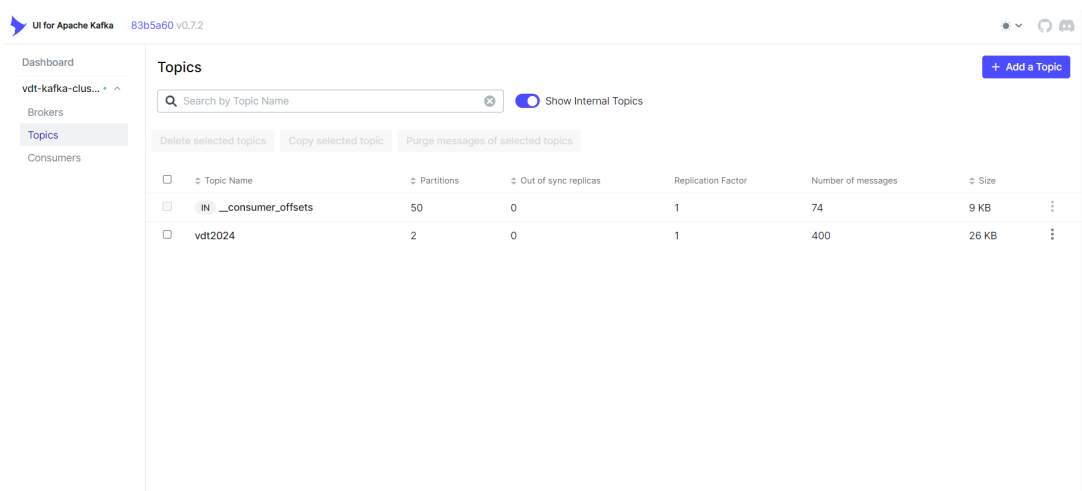
2.4 Xây dựng luồng dữ liệu

2.4.1 Luồng đẩy dữ liệu vào Kafka Topic

Đầu tiên chạy file `producer.py` chứa đoạn mã Python thực hiện những công việc sau:

- Cấu hình Kafka Producer.
- Đọc dữ liệu từng dòng trong file `log_action.csv`.
- Gửi từng message tới Kafka Topic tên là "vdt2024" với khoảng nghỉ 1 giây giữa các lần gửi.
- Sử dụng hàm callback để xử lý phản hồi từ Kafka Broker về việc gửi message thành công hay thất bại.

Sau khi chạy thành công và tất cả các messages được gửi thành công và lưu trữ trong các partitions của topic "vdt2024", chúng ta có thể xem kết quả trong giao diện Kafka UI. Hình 2.1 cho thấy tất cả 400 messages đã được gửi thành công.

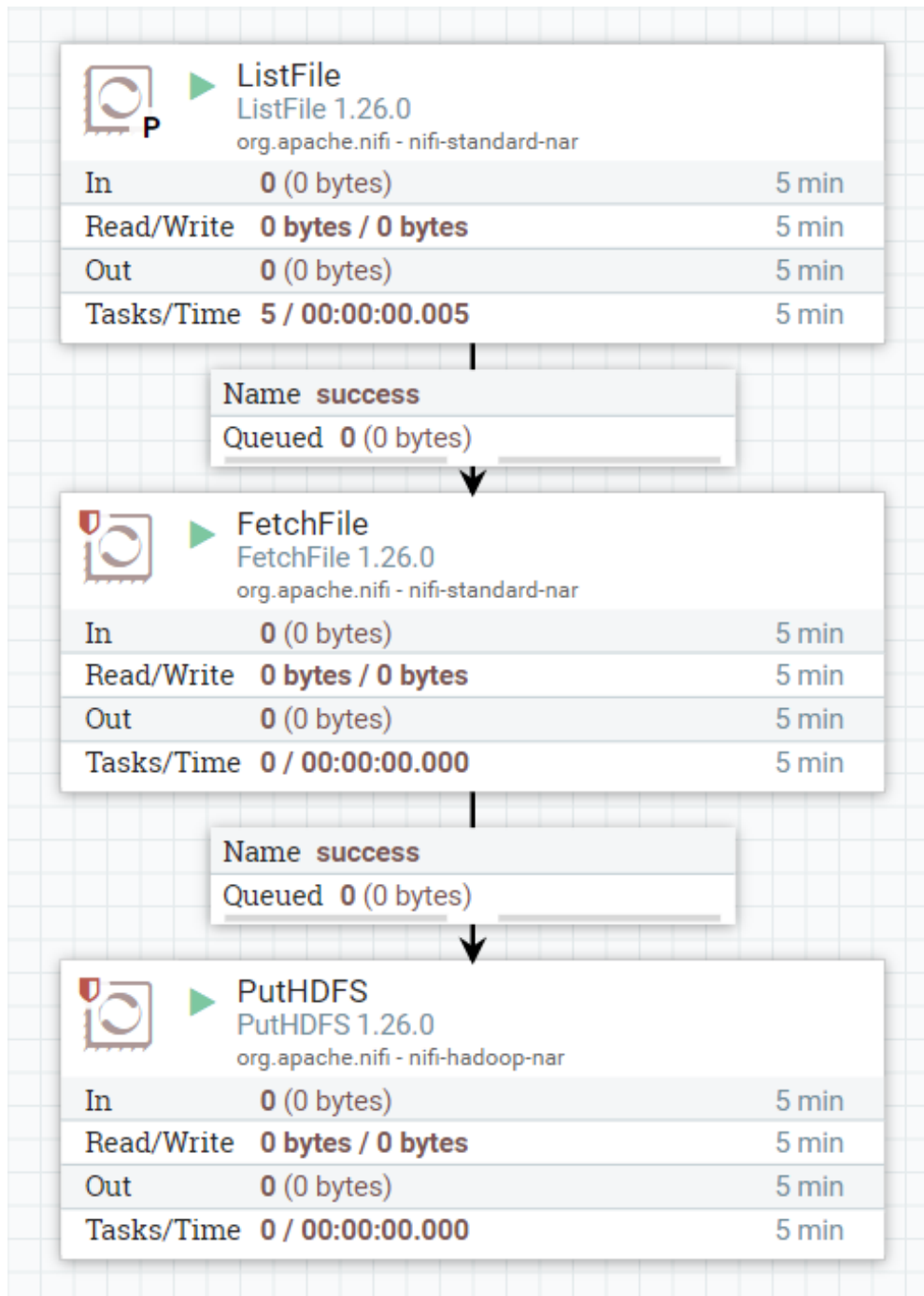


<input type="checkbox"/>	Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size	
<input type="checkbox"/>	<code>_consumer_offsets</code>	50	0	1	74	9 KB	⋮
<input type="checkbox"/>	<code>vdt2024</code>	2	0	1	400	26 KB	⋮

Hình 2.1: Kết quả trên Kafka UI

2.4.2 Luồng dữ liệu trong NiFi

- **Luồng 1:** Lưu trữ file "danh_sach_sv_de.csv" vào HDFS.



Hình 2.2: Luồng 1

Hình 2.2 mô tả luồng dữ liệu sử dụng các Processors sau:

ListFile: Chạy trên PrimaryNode, lấy ra danh sách các files có trong folder với đường dẫn "/opt/nifi/nifi-current/data". Với mỗi file có trong folder đó, tạo ra một FlowFile đại diện cho file đó để có thể được fetch với FetchFile.

Processor Details

ListFile 1.26.0

Running

STOP & CONFIGURE

SETTINGS

SCHEDULING

PROPERTIES

RELATIONSHIPS

COMMENTS

Required field

Property	Value
Input Directory	/opt/nifi/nifi-current/data
Listing Strategy	Tracking Timestamps
Recurse Subdirectories	true
Record Writer	No value set
Input Directory Location	Local
File Filter	[^\.]*
Path Filter	No value set
Include File Attributes	true
Minimum File Age	0 sec
Maximum File Age	No value set
Minimum File Size	0 B
Maximum File Size	No value set

OK

Hình 2.3: Cấu hình của ListFile

FetchFile: Đọc contents của file "danh_sach_sv_de.csv" từ ổ đĩa với đường dẫn được chỉ định trong cấu hình, sau đó truyền contents đó vào FlowFile được tạo ra bởi ListFile.

Processor Details

FetchFile 1.26.0

Running

STOP & CONFIGURE

SETTINGS

SCHEDULING

PROPERTIES

RELATIONSHIPS

COMMENTS

Required field

Property	Value
File to Fetch	/opt/nifi/nifi-current/data/danh_sach_sv_de.csv
Completion Strategy	None
Move Destination Directory	No value set
Move Conflict Strategy	Rename
Log level when file not found	ERROR
Log level when permission denied	ERROR

OK

Hình 2.4: Cấu hình của FetchFile

PutHDFS: Cấu hình đường dẫn tới các files cấu hình của HDFS (core-site.xml, hdfs-site.xml) và đường dẫn trong HDFS nơi file "danh_sach_sv_de.csv" sẽ được lưu trữ.

Processor Details

PutHDFS 1.26.0

Running

STOP & CONFIGURE

SETTINGS

SCHEDULING

PROPERTIES

RELATIONSHIPS

COMMENTS

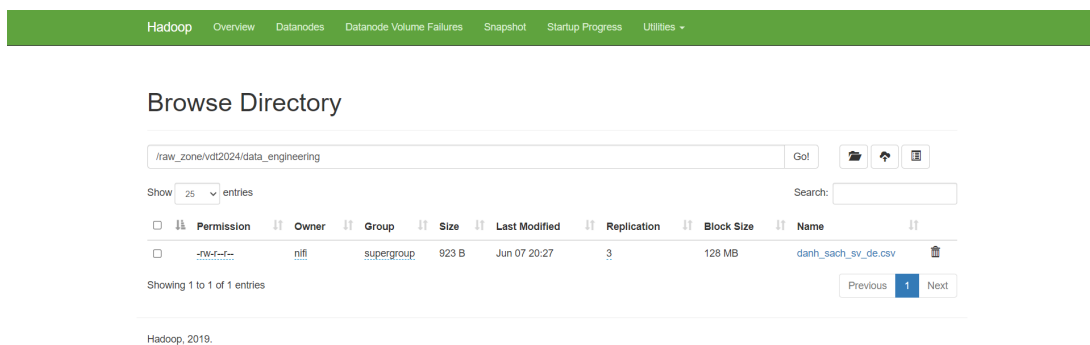
Required field

Property	Value
Hadoop Configuration Resources	/opt/nifi/nifi-current/core-site.xml, /opt/nifi/nifi-current/...
Kerberos Credentials Service	No value set
Kerberos User Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
Kerberos Password	No value set
Kerberos Relogin Period	4 hours
Additional Classpath Resources	No value set
Directory	hdfs://namenode:9000/raw_zone/vdt2024/data_engine...
Conflict Resolution Strategy	replace
Writing Strategy	Write and rename
Block Size	No value set

OK

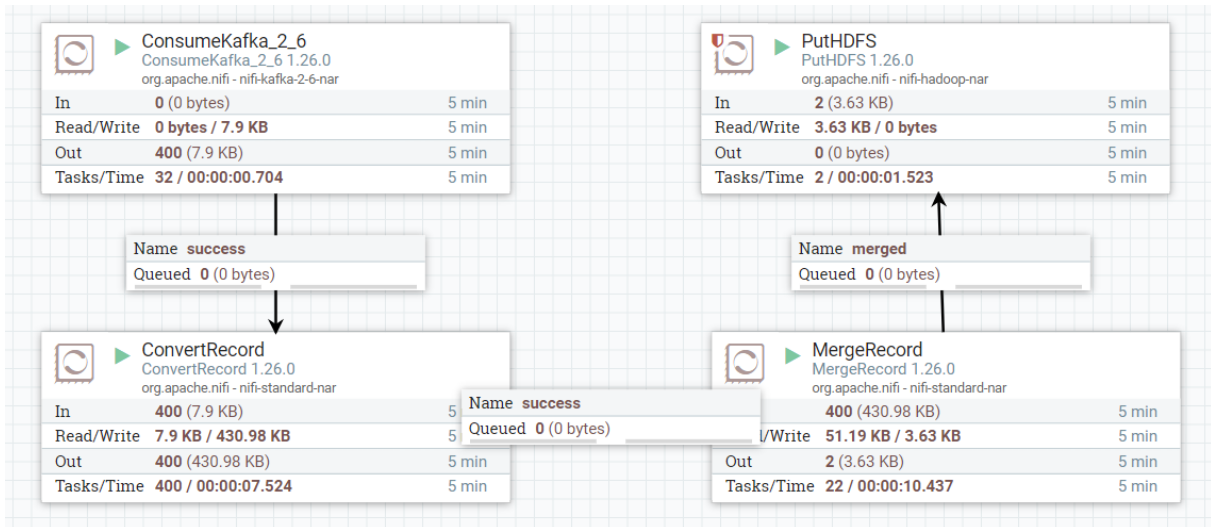
Hình 2.5: Cấu hình của PutHDFS

Sau khi chạy luồng 1 thì file "danh_sach_sv_de.csv" sẽ được lưu trữ trong HDFS như trong hình 2.6.



Hình 2.6: Kết quả

- **Luồng 2:** Consume dữ liệu từ Kafka Topic tên là "vdt2024" và lưu trữ xuống HDFS dưới dạng parquet.



Hình 2.7: Luồng 2

Hình 2.7 mô tả luồng dữ liệu sử dụng các Processors sau:

ConsumeKafka_2_6: Consume các messages từ Kafka Topic tên là "vdt2024" nằm trong cụm Kafka gồm hai Kafka Broker. Mỗi node trong cụm NiFi được coi là một consumer trong group **consume** để cùng consume topic "vdt2024". Các consumer sẽ consume từ offset đầu tiên của các partitions.

Processor Details | ConsumeKafka_2_6 1.26.0

Running STOP & CONFIGURE

SETTINGS SCHEDULING **PROPERTIES** RELATIONSHIPS COMMENTS

Required field

Property	Value
Kafka Brokers	kafka0:29092,kafka1:29091
Topic Name(s)	vdt2024
Topic Name Format	names
Group ID	consume
Commit Offsets	true
Max Uncommitted Time	1 secs
Honor Transactions	true
Message Demarcator	No value set
Separate By Key	false
Security Protocol	PLAINTEXT
SASL Mechanism	GSSAPI
Kerberos Credentials Service	No value set

OK

Hình 2.8: Cấu hình của ConsumeKafka_2_6

ConvertRecord: Processor này dùng để chuyển đổi định dạng file thành parquet. Cấu hình Record Reader sử dụng CSVReader (hình 2.9) và Record Writer sử dụng ParquetRecordSetWriter (hình 2.10).

Controller Service Details | CSVReader 1.26.0

ENABLED DISABLE & CONFIGURE

SETTINGS **PROPERTIES** COMMENTS

Required field

Property	Value
Schema Access Strategy	Use 'Schema Text' Property
Schema Text	{...}
CSV Parser	Apache Commons CSV
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set
CSV Format	Custom Format
Value Separator	,
Record Separator	\n
Treat First Line as Header	false
Ignore CSV Header Column Names	false
Quote Character	"

```

1 {
2   "type": "record",
3   "name": "KafkaRecord",
4   "fields": [
5     {"name": "student_code", "type": "int"},
6     {"name": "activity", "type": "string"},
7     {"name": "numberOfFile", "type": "int"},
8     {"name": "timestamp", "type": "string"}
9   ]
10 }
11

```

OK

OK

Hình 2.9: Cấu hình của CSVReader

Controller Service Details | ParquetReader 1.26.0

ENABLED DISABLE & CONFIGURE

SETTINGS **PROPERTIES** COMMENTS

Required field

Property	Value
Avro Read Compatibility	true

OK

Hình 2.10: Cấu hình của ParquetReader

MergeRecord: Processor này dùng để gộp các message thành một file đủ lớn (100 messages trong ví dụ này) sau đó ghi dưới dạng parquet với thuật toán nén SNAPPY. Cấu hình Record Reader sử dụng ParquetReader (hình 2.10) và Record Writer sử dụng ParquetRecordSetWriter (hình 2.12).

Processor Details | MergeRecord 1.26.0

Running
STOP & CONFIGURE

SETTINGS
SCHEDULING
PROPERTIES
RELATIONSHIPS
COMMENTS

Required field

Property	Value
Record Reader	ParquetReader
Record Writer	ParquetRecordSetWriter
Merge Strategy	Bin-Packing Algorithm
Correlation Attribute Name	No value set
Attribute Strategy	Keep Only Common Attributes
Minimum Number of Records	100
Maximum Number of Records	1000
Minimum Bin Size	0 B
Maximum Bin Size	No value set
Max Bin Age	No value set
Maximum Number of Bins	10

OK

Hình 2.11: Cấu hình của MergeRecord

Controller Service Details | ParquetRecordSetWriter 1.26.0

ENABLED
DISABLE & CONFIGURE

SETTINGS
PROPERTIES
COMMENTS

Required field

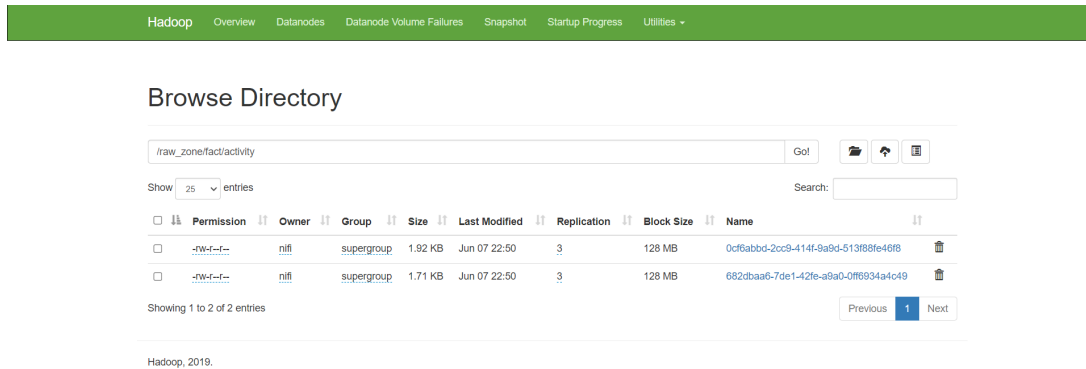
Property	Value
Schema Write Strategy	Do Not Write Schema
Schema Cache	No value set
Schema Access Strategy	Inherit Record Schema
Cache Size	1000
Compression Type	SNAPPY
Row Group Size	No value set
Page Size	No value set
Dictionary Page Size	No value set
Max Padding Size	No value set
Enable Dictionary Encoding	No value set
Enable Validation	No value set
Writer Version	No value set

OK

Hình 2.12: Cấu hình của ParquetRecordSetWriter

PutHDFS: Tương tự với processor PutHDFS ở luồng 1 nhưng khác với đường dẫn lưu trữ là: `"/raw_zone/fact/activity"`.

Sau khi chạy luồng 2 thì 400 messages trong topic đã được nén và lưu trữ dưới định dạng parquet, tạo thành hai file parquet lưu trữ trong HDFS như trong hình 2.13.



Hình 2.13: Kết quả

2.5 Xử lý dữ liệu bằng Apache Spark

Trong phần này, chúng ta sẽ sử dụng Jupyter Notebook để coding ứng dụng Spark giải yêu cầu của bài toán.

1. Import thư viện và khởi tạo SparkSession

Import SparkSession và tất cả các hàm để tương tác với DataFrame. Sau đó khởi tạo SparkSession đặt là **spark** làm điểm vào cho các hoạt động tương tác với Spark.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

spark = SparkSession.\
    builder.\
    appName("pyspark-notebook").\
    master("spark://spark-master:7077").\
    config("spark.executor.memory", "512m").\
    getOrCreate()
```

Hình 2.14: Khởi tạo SparkSession

2. Đọc files được lưu trữ trong HDFS

Dữ liệu đã được đẩy vào HDFS bằng các luồng trong NiFi, tiến hành đọc và tạo ra DataFrame để bắt đầu xử lý dữ liệu.

- **logDF**: DataFrame của file "log_activity.csv" được lưu trữ dưới dạng parquet trong HDFS.
- **listDF**: DataFrame của file "danh_sach_sv_de.csv".

```
# Đường dẫn tới log_activity parquet
log_path = "hdfs://namenode:9000/raw_zone/fact/activity"
# Đường dẫn tới file danh_sach_sv_de.csv
list_path = "hdfs://namenode:9000/raw_zone/vdt2024/data_engineering/danh_sach_sv_de.csv"

try:
    logDF = spark.read \
        .format("parquet") \
        .load(log_path)
except Exception as e:
    print(f"Error reading parquet files: {e}")

try:
    listDF = spark.read \
        .format("csv") \
        .option("header", "false") \
        .option("inferSchema", "true") \
        .load(list_path)
except Exception as e:
    print(f"Error reading csv file: {e}")
```

Hình 2.15: Đọc file

3. Đổi lại tên cột

Do file "danh_sach_sv_de.csv" không có header nên Spark đã đặt tên mặc định của các cột của listDF là **__c0** và **__c1**. Đổi lại tên cột như sau:

- **__c0** → **student_code**
- **__c1** → **student_name**

```
listDF = listDF.withColumnRenamed("__c0", "student_code") \
    .withColumnRenamed("__c1", "student_name")
listDF.printSchema()

root
 |-- student_code: integer (nullable = true)
 |-- student_name: string (nullable = true)
```

Hình 2.16: Đổi tên cột

4. Xử lý cột timestamp theo yêu cầu

Đầu tiên chuyển định dạng của cột từ string về dạng date (M/d/yyyy) sau đó chuyển về định dạng yyyyMMdd.

Chuyển cột timestamp thành dạng 'yyyyMMdd' và chuyển tên cột thành date

```
logDF = logDF.withColumn("timestamp", date_format(to_date(col("timestamp"), "M/d/yyyy"), "yyyyMMdd")) \
    .withColumnRenamed("timestamp", "date")
```

Hình 2.17: Xử lý cột timestamp

5. Join hai DataFrame

Join hai DataFrame theo cột **student_code** với kiểu join là INNER JOIN.

```
joinedDF = logDF.join(listDF, "student_code", "inner")
joinedDF.show(5)
```

```
+-----+-----+-----+-----+-----+
|student_code|activity|numberOfFile|    date|    student_name|
+-----+-----+-----+-----+-----+
|          33|execute|          1|20240613|    Nguyễn Bá Thiêm|
|          24|execute|          8|20240612|    Nguyễn Hoài Nam|
|          31|  write|          9|20240613|      Tạ Đức Tiến|
|          21|  read|          5|20240612|Trần Phúc Mạnh Linh|
|          24|  read|         10|20240612|    Nguyễn Hoài Nam|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Hình 2.18: Join

6. Tính toán đưa ra kết quả

Sử dụng **groupBy()** để nhóm theo các cột "date", "student_code", "student_name", "activity" sau đó tính tổng của cột "numberOfFile" theo activity. Sử dụng thêm **orderBy()** để kết quả trông gọn gàng hơn.

```
result_df = joinedDF.groupBy("date", "student_code", "student_name", "activity") \
    .agg(sum("numberOfFile").alias("totalFile")) \
    .orderBy(col("student_code"), col("date").asc(), col("activity"))

result_df.show(5)
```

date	student_code	student_name	activity	totalFile
20240610	1	Mai Đức An	read	25
20240610	1	Mai Đức An	write	6
20240611	1	Mai Đức An	execute	3
20240611	1	Mai Đức An	read	11
20240612	1	Mai Đức An	execute	10

only showing top 5 rows

Hình 2.19: Kết quả

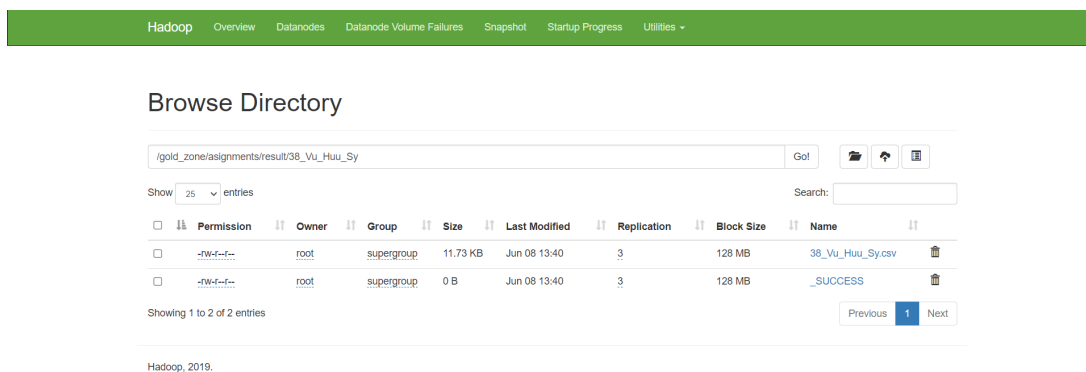
7. Lưu trữ kết quả vào HDFS

Repartition về thành một partition duy nhất sau đó lưu trữ kết quả vào trong HDFS dưới dạng file csv.

```
result_df.repartition(1) \
    .write \
    .csv("hdfs://namenode:9000/gold_zone/assignments/result/38_Vu_Huu_Sy",
        header=True,
        mode="overwrite")
```

Hình 2.20: Lưu kết quả vào HDFS

Hình 2.21 cho thấy file kết quả đã được lưu trữ thành công xuống HDFS.



Hình 2.21: File kết quả trong HDFS