

VIETTEL GROUP



Assignment Report

**BUILDING SIMPLE
DATA PLATFORM**

Dinh Viet Thanh

dvthanh19@gmail.com

Viettel Digital Talent 2024 Program

Major: Data Engineering

HO CHI MINH CITY, JUNE 2024

Contents

1	General	3
1.1	Requirements	3
2	Preparation	5
2.1	Prerequisites	5
2.2	Docker compose file	5
3	Deployment	6
3.1	Run the docker compose up	6
3.2	Pipeline	6
4	Source Code	12

1 General

1.1 Requirements

For this assignment, the task is to construct a basic data platform utilizing the following technologies: Apache Kafka, Apache NiFi, Hadoop HDFS, and Apache Spark.

Architecture

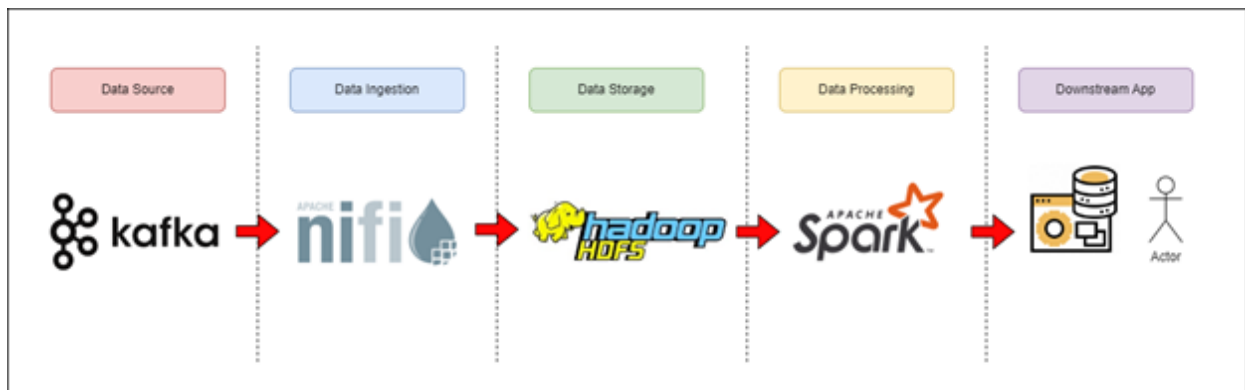


Figure 1: *Data platform architecture*

Input and Output

1. Apache Kafka:

- Task: Develop a program to read from *log_action.csv* and push data to a Kafka Topic `vdt2024`, including the following field: `student_code` (number), `activity` (string), `numberOfFile` (number), `timestamp` (string)

2. Apache NiFi:

- Task: Deploy Apache NiFi to pull data from the Kafka Topic `vdt2024`, process it, and store it in HDFS at the path `/raw_zone/fact/activity`
- Data Format: Store the data in Parquet format

3. Hadoop HDFS:

- Task: Store the file *danh_sach_sv_de.csv* into HDFS.

4. Apache Spark

- Task: Write a program to process data stored in HDFS using Apache Spark.
- Data Processing Requirement: Calculate the total number of files interacted with daily for each type of activity performed by each student. Save the result to an output file.

- Output File Details:
 - File Name: *student_name.csv*
 - Format: CSV
 - Schema: date, student_code, student_name, activity, totalFile

2 Preparation

To effectively deploy the data pipeline using Docker, we need to prepare the following components and configurations. This section outlines the necessary preparations.

2.1 Prerequisites

Before proceeding with the deployment, ensure that the following prerequisites are met:

- **Docker:** ensure Docker is installed and running on your system. Follow the official Docker installation guide for your operating system.
- **Docker Compose:** install Docker Compose, which is used to define and run multi-container Docker applications. Follow the official Docker Compose installation guide.
- **Python:** install Python, which will be used for scripting and running the Kafka producer program. Ensure you have Python 3.x installed. Follow the official Python installation guide for your operating system.
- **Docker images:** prepare the Docker images for the various technologies involved in the data pipeline.

2.2 Docker compose file

Create a *docker-compose.yml* file to define the services and their configurations. This file will specify how to run each component of the data pipeline as a container. There are several points that we need to notices:

- Image's name and version, dependent images.
- Ports and port mapping.
- Volume, mounted volume and networks.

You can see detail in my repository.

3 Deployment

3.1 Run the docker compose up

Start the Docker daemon and run the docker compose up:

```
1 docker-compose -f docker-compose.yml -p demo up -d
```

Then you will wait for it to install the image (if it is not available yet) and build.

```
• dvt@dvtanh:~/data_pipeline_2/src$ docker-compose -f docker-compose.yml -p demo up -d
WARN[0000] /home/dvt/data_pipeline_2/src/docker-compose.yml: 'version' is obsolete
[+] Running 29/29
 ✓ Network demo_default          Created                                0.0s
 ✓ Volume "demo_hadoop_datanode1" Created                                0.0s
 ✓ Volume "demo_hadoop_datanode2" Created                                0.0s
 ✓ Volume "demo_hadoop_datanode3" Created                                0.0s
 ✓ Volume "demo_hadoop_historyserver" Created                          0.0s
 ✓ Volume "demo_hadoop_namenode" Created                              0.0s
 ✓ Container datanode3           Started                                2.4s
 ✓ Container namenode            Started                                2.9s
 ✓ Container resourcemanager     Started                                2.4s
 ✓ Container datanode1           Started                                2.0s
 ✓ Container nodemanager         Started                                2.4s
 ✓ Container datanode2           Started                                1.7s
 ✓ Container sparkmaster         Started                                2.9s
 ✓ Container zookeeper           Healthy                         18.4s
 ✓ Container historyserver       Started                                2.4s
 ✓ Container nifi                Started                                2.9s
 ✓ Container sparkworker         Started                                3.1s
 ✓ Container broker02            Healthy                         44.9s
 ✓ Container broker01            Healthy                         44.9s
 ✓ Container kafka-ui            Started                                45.5s
```

Figure 2: Docker compose up

It is seen that all containers run successfully. You can also check in Docker application

3.2 Pipeline

Kafka and NiFi Firstly, go to Kafka UI at `localhost:8080` to check Kafka's status:

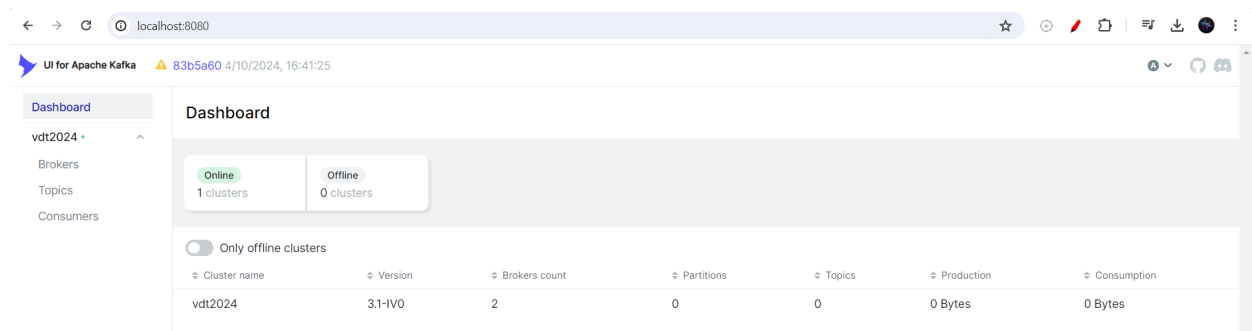
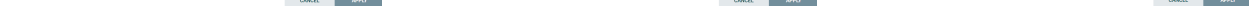


Figure 3: Apache Kafka interface

Second, go to NiFi UI at `localhost:8082/nifi` to check NiFi's status:



0 0 0 1 0

Configure Processor | PutHDFS 1.26.0

Invalid

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Required field

Property	Value
Hadoop Configuration Resources	/opt/nifi-current/hdfs_config/core-site.xml/opt/nifi_...
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	/raw_zone/fact/activity
Conflict Resolution Strategy	fail
Writing Strategy	Write and rename
Block Size	No value set

CANCEL APPLY

Configure Processor | PutHDFS 1.26.0

Invalid

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Automatically Terminate / Retry Relationships

failure

☒ terminate ☐ retry

Files that could not be written to HDFS for some reason are transferred to this relationship

success

☒ terminate ☐ retry

Files that have been successfully written to HDFS are transferred to this relationship

CANCEL APPLY

Figure 6: Configuration for processor PutHDFS

Then I connect these 2 processor and configure for this connection

Create Connection

DETAILS SETTINGS

From Processor
ConsumeKafkaRecord_2_0
ConsumeKafkaRecord_2_0

To Processor
PutHDFS
PutHDFS

Within Group
NiFi Flow

Within Group
NiFi Flow

For Relationships
☐ parse.failure
☒ success

CANCEL ADD

Figure 7: Configuration for connections among 2 processors

Enable all controller services in NiFi Flow and run it.

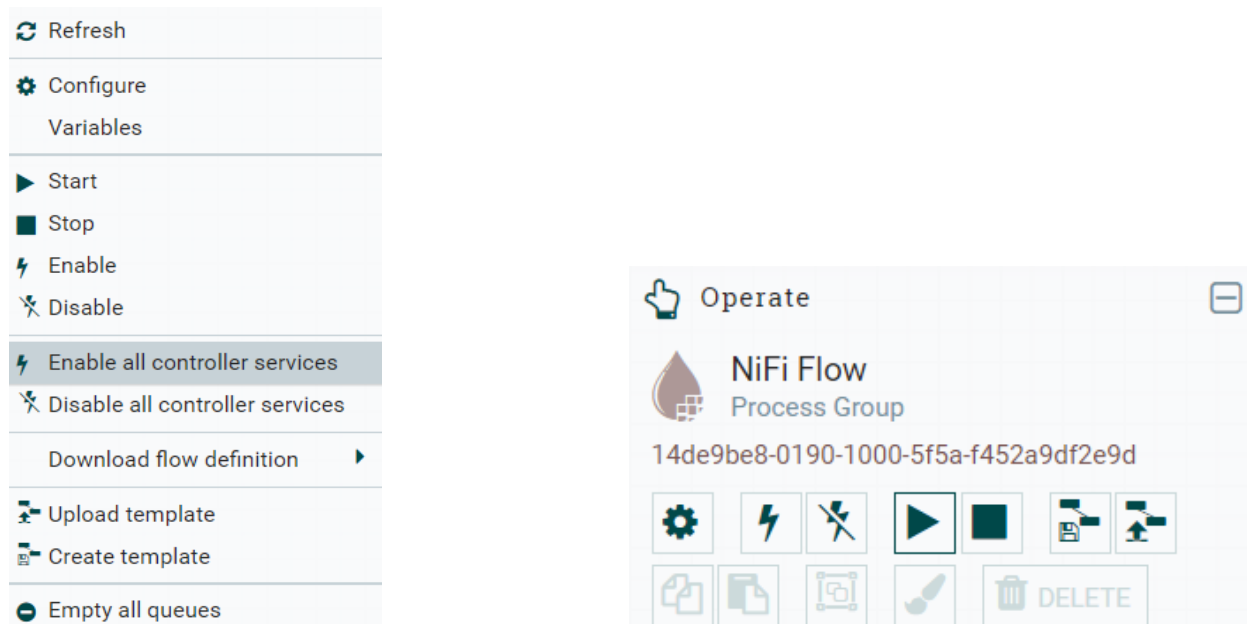


Figure 8: Enable controller services and Run the NiFi flow

You can see the NiFi flow has run already.

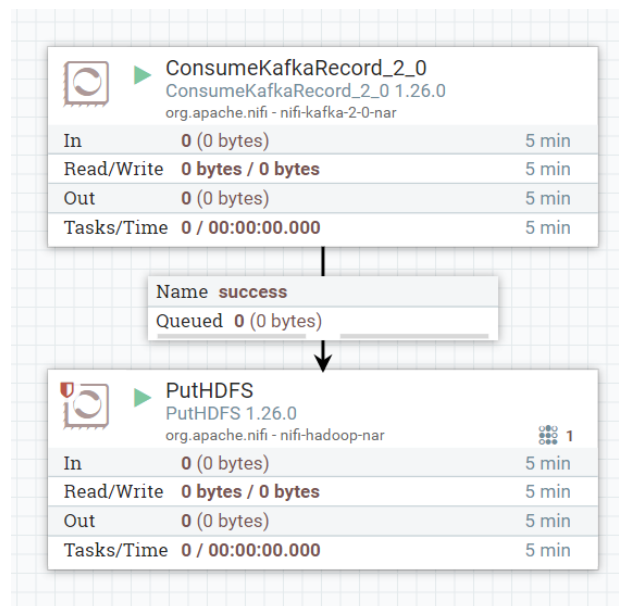


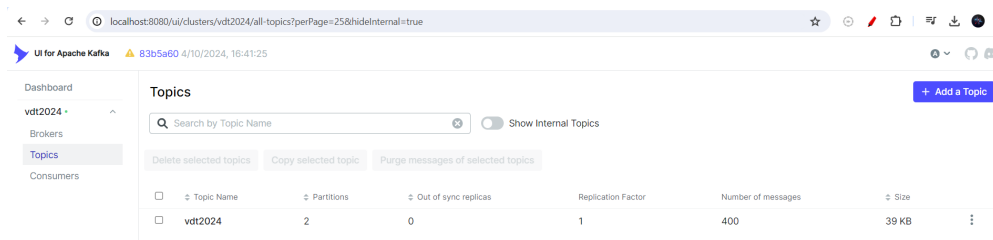
Figure 9: NiFi processors run

Next step is running the local consumer `kafka_consumer.py` to check if the producer send successfully. Moreover, you also can check by Kafka UI and NiFi UI. Now, start to stream the data to Kafka by Kafka producer `kafka_producer.py`

```
dvthanh@dvthanh:~/data_pipeline_2/src$ python3 ./main/kafka_producer.py
400 records sent to topic 'vdt2024' of Kafka successfully
dvthanh@dvthanh:~/data_pipeline_2/src$

dvthanh@dvthanh:~/data_pipeline_2/src$ python3 ./main/kafka_consumer.py
0: {'topic': 'vdt2024', 'partition': 0, 'offset': 0, 'key': 'even', 'value': '{"student_code": 4, "activity": "write", "numberOfFile": 7, "timestamp": "6/10/2024"}'}
1: {'topic': 'vdt2024', 'partition': 0, 'offset': 1, 'key': 'odd', 'value': '{"student_code": 33, "activity": "read", "numberOfFile": 5, "timestamp": "6/12/2024"}'}
2: {'topic': 'vdt2024', 'partition': 0, 'offset': 2, 'key': 'odd', 'value': '{"student_code": 33, "activity": "execute", "numberOfFile": 1, "timestamp": "6/13/2024"}'}
3: {'topic': 'vdt2024', 'partition': 0, 'offset': 3, 'key': 'even', 'value': '{"student_code": 6, "activity": "write", "numberOfFile": 6, "timestamp": "6/15/2024"}'}
4: {'topic': 'vdt2024', 'partition': 0, 'offset': 4, 'key': 'even', 'value': '{"student_code": 24, "activity": "execute", "numberOfFile": 8, "timestamp": "6/12/2024"}'}
5: {'topic': 'vdt2024', 'partition': 0, 'offset': 5, 'key': 'even', 'value': '{"student_code": 22, "activity": "write", "numberOfFile": 2, "timestamp": "6/12/2024"}'}
```

Figure 10: Local consumer consume message



The screenshot shows the Kafka UI for the 'vdt2024' topic. The table lists the topic with 2 partitions, 0 out of sync replicas, a replication factor of 1, 400 messages, and a size of 39 KB.

Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
vdt2024	2	0	1	400	39 KB

Figure 11: Check by Kafka UI

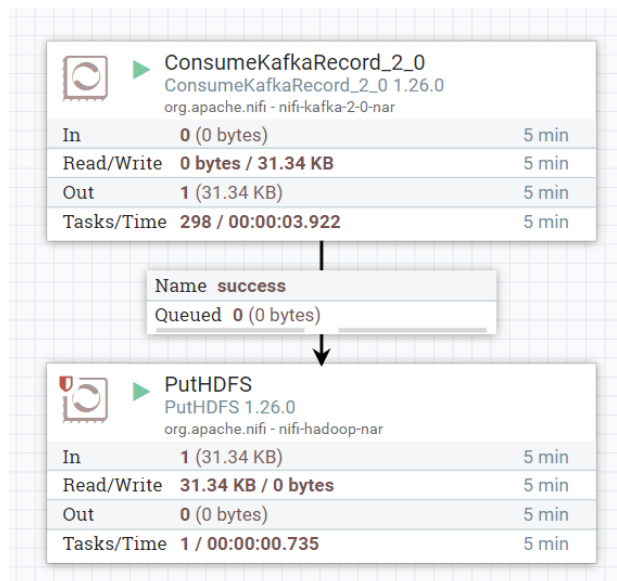


Figure 12: NiFi ingest the data from Kafka

HDFS and Spark First, it is needed to check whether the parquet was written by NiFi to HDFS by execute some command inside `nodemanager` container



```
nodemanager
root@9dc22389faf:/# hdfs dfs -ls /newzone/fact/activity
Found 1 items
-rw-r--r-- 3 nifi, supergroup 32089 2024-06-04 11:41 /newzone/fact/activity/fo44cd5b-948c-462c-9009-e0b6d80a772
```

Figure 13: Check parquet file put in HDFS from NiFi

Now we are going to move the file `danh_sach_sv_de.csv` into the HDFS by a command `hadoop fs -mv` inside `namenode`



Figure 14: Put file `danh_sach_sv_de.csv` into HDFS

Then we go inside `sparkworker` container to execute command in order to process the data stored in HDFS.

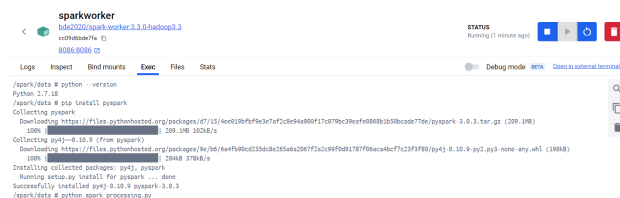


Figure 15: Run `spark_processing.py` in Spark container

And this is the result. Furthermore, the result is also written down in the output directory

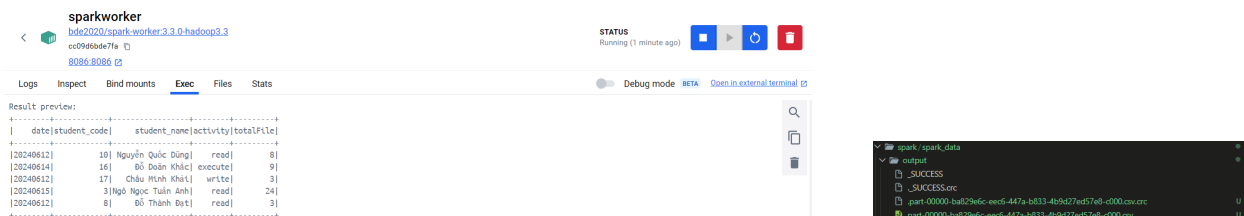


Figure 16: Spark output

4 Source Code

Follow the deployment on this repository. All the code is in directory *src*.

This is the link of my repository: github.com/dvthanh19/vdt_data_pipeline