FACULTÉ DES SCIENCES DE SFAX

Université de Sfax
TUNISIE

RAPPORT DE PROJET BIG DATA

Pipeline Big Data pour l'Ingestion, le Stockage, et la Visualisation Temps Réel des Données de Transactions Bancaires

ELABORÉ PAR :

Ahmed Rami Belghuith

Dhia Elhak Toukebri

Année universitaire : 2024/2025

Table des matières

Table des figures

1.1 Objectif

Ce Projet a pour but de fournir une description détaillée des étapes de configuration et de mise en œuvre d'un pipeline Big Data. Le pipeline vise à automatiser l'ingestion, le stockage, l'analyse, et la visualisation des données en temps réel en utilisant des outils tels qu'Apache Airflow, Kafka, HDFS, Hive, et Superset. Il sert de guide pratique pour déployer la solution et garantir une compréhension claire des processus sous-jacents.

1.2 Contexte

Dans un contexte où les entreprises manipulent d'énormes volumes de données, il est crucial de mettre en place des systèmes automatisés pour gérer ces données efficacement. Le projet répond au besoin d'automatiser l'ingestion de fichiers CSV, leur stockage dans un système distribué (HDFS), leur exploitation via une base de données (Hive), et leur visualisation en temps réel à travers un tableau de bord interactif (Superset). L'objectif est de permettre une mise à jour fluide et continue des tableaux de bord sans intervention manuelle, assurant ainsi une prise de décision basée sur des données toujours à jour.

1.3 Analyse des Besoins Fonctionnels

Le pipeline couvre les étapes suivantes :

0. **Ingestion de Données** : Surveillance d'un répertoire pour détecter de nouveaux fichiers CSV à traiter.
0. **Transport via Kafka** : Transmission des fichiers détectés vers un cluster HDFS en utilisant Apache Kafka.
0. **Stockage dans HDFS** : Enregistrement des fichiers CSV dans un système de fichiers distribué (HDFS).
0. **Analyse avec Hive** : Intégration des données stockées dans HDFS dans une base de données Hive pour permettre des requêtes SQL.
0. **Visualisation avec Superset** : Création d'un tableau de bord interactif et mise à jour automatique des données visualisées grâce à une connexion JDBC entre Hive et Superset.

1.4 Bénéfices attendus

— **Automatisation complète** : Réduction de l'intervention humaine dans le traitement des fichiers.

— **Efficacité** : Traitement rapide et fiable de grandes quantités de données.

— **Visualisation en temps réel** : Mise à jour immédiate des tableaux de bord dès l'ingestion d'un nouveau fichier.

— **Scalabilité** : Extensibilité pour inclure de nouvelles sources de données ou de nouveaux cas d'utilisation.

2.1 Introduction

Cette section décrit les configurations nécessaires pour déployer et exécuter le pipeline sur un cluster Hadoop comprenant un nœud maître (master) et deux nœuds esclaves (slave1 et slave2). Elle détaille les configurations des outils utilisés (Airflow, Kafka, HDFS,Hue, Hive, Superset) ainsi que la mise en place des connexions entre les composants.

2.2 Configuration du Hadoop

Nœud maître (master)

— Responsable : Gestion des ressources, coordination des tâches et supervision globale du cluster.

— **Services principaux** :

— *Namenode* : Gère les métadonnées du système de fichiers HDFS.

— *ResourceManager* : Alloue les ressources aux applications distribuées.

— *Zookeeper* : Service de coordination distribué pour la gestion des configurations et le suivi des états.

— Hue : Interface utilisateur pour interagir avec Hadoop et exécuter diverses tâches (soumission de jobs, gestion des fichiers, etc.).

Nœuds esclaves (slave1, slave2)

— Responsable : Exécution des tâches distribuées.

— **Services :**

— *Datanode* : Stocke les données de manière distribuée dans HDFS.

— *NodeManager* : Gère les conteneurs pour l'exécution des tâches MapReduce ou Spark.

— *Zookeeper client* : Assure la communication avec le service Zookeeper pour garantir une coordination fiable et une haute disponibilité.

2.2.1 Configuration de core-site.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <configuration>
4     <property>
5         <name>fs.defaultFS</name>
```

```
        <value>hdfs://master:9000</value>
    </property>
    <property>
        <name>dfs.webhdfs.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>dfs.webhdfs.address</name>
        <value>master:50070</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hue.groups</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hue.hosts</name>
        <value>*</value>
    </property>
</configuration>
```

Listing 2.1 – core-site.xml

2.2.2 Configuration de hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/home/master/hdata/hdfs/namenode</value>
</property>
<property>
```

```
<name>dfs.datanode.data.dir</name>
<value>file:/home/master/hdata/hdfs/datanode</value>
</property>
<property>
<name>dfs.namenode.checkpoint.dir</name>
<value>file:/home/master/hdata/hdfs/namesecondary</value>
</property>
<property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
</property>
<property>
    <name>dfs.namenode.http-address</name>
    <value>master:50070</value>
</property>
<property>
    <name>dfs.permissions.enabled</name>
    <value>false</value>
</property>
<property>
  <name>dfs.namenode.acls.enabled</name>
  <value>false</value>
</property>
<property>
    <name>dfs.permissions</name>
    <value>false</value>
  </property>

</configuration>
```

Listing 2.2 – hdfs-site.xml

2.2.3 Configuration de yarn-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>master</value>
</property>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>file:/home/master/hdata/yarn/local</value>
</property>
<property>
<name>yarn.nodemanager.log-dirs</name>
<value>file:/home/master/hdata/yarn/log</value>
</property>
<property>
<name>yarn.log.server.url</name>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle,spark_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler,
org.apache.spark.network.yarn.YarnShuffleService</value>
</property>
<property>
<name>yarn.nodemanager.disk-health-checker
.max-disk-utilization-per-disk-percentage</name>
<value>100</value>
</property>
<property>
  <name>mapred.jobtracker.taskScheduler</name>
  <value>org.apache.hadoop.mapred.CapacityTaskScheduler</value>
</property>
<property>
```

```
38   <name>yarn.resourcemanager.proxyuser.hue.groups</name>
39   <value>*</value>
40 </property>
41 <property>
42   <name>yarn.resourcemanager.proxyuser.hue.hosts</name>
43   <value>*</value>
44 </property>
45 </configuration>
```

Listing 2.3 – yarn-site.xml

2.2.4 Configuration de zoo.cfg (Zookeeper)

```
1 tickTime=2000
2 initLimit=10
3 syncLimit=5
4 dataDir=/var/lib/zookeeper
5 clientPort=2181
6 server.1=master:2888:3888
7 server.2=slave1:2888:3888
8 server.3=slave2:2888:3888
```

Listing 2.4 – zoo.cfg

2.3 Ajout de Hue

2.3.1 configuration de Hue.ini

```
1 [desktop]
2 default_user = master
3 default_superuser = true
4 log_dir=/var/log/hue
5 logging_level=DEBUG
6 secret_key=
7 http_host=0.0.0.0
8 http_port=8888
9 time_zone=America/Los_Angeles
```

```
10 django_debug_mode=false
11 http_500_debug_mode=false
12 [[custom]]
13 [[auth]]
14 idle_session_timeout=-1
15 host=localhost
16 port=25
17 user=
18 password=
19 tls=no
20 [hadoop]
21 proxy_user_enable=true
22 [[hdfs_clusters]]
23 [[[default]]]
24 hadoop_username=master
25 fs_defaultfs=hdfs://192.168.1.2:9000
26 hadoop_conf_dir=/home/master/hadoop-3.3.6/etc/hadoop
27 security_enabled=false
28 webhdfs_url=http://192.168.1.2:50070/webhdfs/v1/
29 submit_to=True
30 enable_doas=True
31 [[yarn_clusters]]
32 [[[default]]]
33 resourcemanager_host=192.168.1.2
34 resourcemanager_port=8088
35 submit_to=True
36 resourcemanager_api_url=http://192.168.1.2:8088
37 history_server_api_url=http://192.168.1.2:19888
38 [beeswax]
39 hive_server_host=localhost
40 hive_server_port=10000
41 hive_server_username=hive
42 hive_server_password=password
43 thrift_transport=TBinaryProtocol
44 hive_server2_authentication=NOSASL
```

```
45
46 [filebrowser]
47 show_upload_button=true
48 level = DEBUG
49 file = /var/log/hue/hue.log
50 rotation = true
51 max_size = 10485760
52 backup_count = 5
53 console = true
```

Listing 2.5 – Hue.ini

2.4 Configuration d'Apache Kafka

2.4.1 Configuration de server.properties

Sur chaque nœud :

```
1 broker.id=1 # =2 pour slave1 et =3 pour slave2
2 listeners=PLAINTEXT://:9092
3 advertised.listeners=PLAINTEXT://master:9092
4 num.network.threads=3
5 num.io.threads=8
6 socket.send.buffer.bytes=102400
7 socket.receive.buffer.bytes=102400
8 socket.request.max.bytes=104857600
9 log.dirs=/var/lib/kafka
10 num.partitions=1
11 offsets.topic.replication.factor=1
12 transaction.state.log.replication.factor=1
13 transaction.state.log.min.isr=1
14 log.retention.hours=168
15 log.retention.check.interval.ms=300000
16 zookeeper.connect=master:2181,slave1:2181,slave2:2181
17 zookeeper.connection.timeout.ms=18000
18 group.initial.rebalance.delay.ms=0
```

Listing 2.6 – server.properties

2.5 Configuration d'Apache Hive

2.5.1 Configuration de hive-site.xml

```
<property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost/metastore_dbcreateDatabaseIfNotExist=
      true&amp;allowPublicKeyRetrieval=true</value>
    </property>
    <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.cj.jdbc.Driver</value>
    </property>
    <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hiveuser</value>
    </property>
    <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hivepassword</value>
    </property>
    <property>
    <name>hive.server2.thrift.bind.host</name>
    <value>localhost</value>
    </property>
    <property>
    <name>hive.server2.thrift.port</name>
    <value>10000</value>
    </property>
     <property>
    <name>hive.server2.thrift.port</name>
    <value>10000</value>
    </property>
```

```
    <!---autre ----------->
```

Listing 2.7 – Hive-site.xml

2.5.2 L'ajout de connector dans hive-env.sh

```
# export HIVE_AUX_JARS_PATH=
export HIVE_CLASSPATH=/opt/hive/lib/mysql-connector-j-8.0.32.jar:$HIVE_CLASSPATH
```

Listing 2.8 – hive-env.sh

2.6 Configuration de Apache Airflow

2.6.1 Configuration de l'environnement Airflow

```
# Configuration de l'environnement
AIRFLOW_HOME = '/opt/airflow'
DAGS_FOLDER = '/opt/airflow/dags'

# Configuration des connexions
KAFKA_BROKERS = 'master:9092,slave1:9092,slave2:9092'
KAFKA_TOPIC = 'csv-topic'
HDFS_URL = 'http://master:50070'
HDFS_USER = 'hdfs'
HDFS_BASE_PATH = '/kafka_ingestion/csv_files'

MONITORED_FOLDER = os.path.join(os.path.expanduser('~'), 'kafka_ingestion')
PROCESSED_FOLDER = os.path.join(MONITORED_FOLDER, 'processed')

# Configuration du DAG
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email_on_failure': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}
```

Listing 2.9 – Configuration de l'environnement Airflow

2.6.2 Implémentation des Composants

2.6.2.1 Producteur Kafka

```
def create_kafka_producer():
    return Producer({
        'bootstrap.servers': KAFKA_BROKERS,
        'client.id': 'airflow_producer',
        'acks': 'all',
        'retries': 3,
        'linger.ms': 1,
        'batch.size': 16384,
        'buffer.memory': 33554432
    })

def produce_to_kafka(**kwargs):
    csv_files = kwargs['ti'].xcom_pull(key='csv_files', task_ids='scan_csv_files')
    producer = create_kafka_producer()
    try:
        for csv_file in csv_files:
            full_path = os.path.join(MONITORED_FOLDER, csv_file)
            producer.produce(KAFKA_TOPIC, full_path.encode('utf-8'))
            producer.flush()
    finally:
        producer.close()
```

Listing 2.10 – Implémentation du Producteur Kafka

2.6.2.2 Consommateur Kafka

```
def create_kafka_consumer():
    return Consumer({
        'bootstrap.servers': KAFKA_BROKERS,
        'group.id': 'csv-hdfs-ingestion-airflow',
```

```
        'auto.offset.reset': 'earliest',
        'enable.auto.commit': False,
        'max.poll.interval.ms': 300000
    })

def consume_and_ingest_to_hdfs(**kwargs):
    consumer = create_kafka_consumer()
    consumer.subscribe([KAFKA_TOPIC])
    try:
        while True:
            msg = consumer.poll(1.0)
            if msg is None:
                break
            if msg.error():
                continue
            csv_path = msg.value().decode('utf-8').strip()
            filename = os.path.basename(csv_path)
            hdfs_destination = os.path.join(HDFS_BASE_PATH, filename)

            with open(csv_path, 'rb') as local_file:
                with hdfs_client.write(hdfs_destination, overwrite=True) as hdfs_file
                    shutil.copyfileobj(local_file, hdfs_file)
    finally:
        consumer.close()
```

Listing 2.11 – Implémentation du Consommateur Kafka

2.6.3 Intégration avec HDFS

```
def write_to_hdfs(data, hdfs_path):
    client = InsecureClient(HDFS_URL, user=HDFS_USER)
    with client.write(hdfs_path, overwrite=True) as writer:
        writer.write(data)
```

Listing 2.12 – Intégration avec HDFS

2.6.4 Pipeline Airflow

Le pipeline Airflow défini ci-dessous orchestre les différentes étapes du processus, de la détection des fichiers CSV à leur ingestion dans HDFS.

```
with DAG(
    'kafka_hdfs_ingestion_pipeline',
    default_args=default_args,
    description='Pipeline to monitor folder, send to Kafka, and ingest to HDFS',
    schedule_interval='*/1 * * * *',
    start_date=datetime(2024, 1, 1),
    catchup=False,
) as dag:

    scan_task = PythonOperator(
        task_id='scan_csv_files',
        python_callable=scan_csv_files,
        provide_context=True,
    )

    produce_task = PythonOperator(
        task_id='produce_to_kafka',
        python_callable=produce_to_kafka,
        provide_context=True,
    )

    ingest_task = PythonOperator(
        task_id='consume_and_ingest_to_hdfs',
        python_callable=consume_and_ingest_to_hdfs,
        provide_context=True,
    )

    scan_task >> produce_task >> ingest_task
```

Listing 2.13 – Pipeline Airflow

2.7 Configuration de Apache superset

2.7.1 configuration de supersetconfig

```
# Flask application settings
FLASK_APP = 'superset.app:create_app()'
APP_DIR = '/home/master/spark_env/lib/python3.11/site-packages/superset'

# Superset specific config
ROW_LIMIT = 10000
SQLLAB_ASYNC_TIME_LIMIT_SEC = 3600

SQLLAB_TIMEOUT = 3600
SQLLAB_HARD_TIMEOUT = 3660
SUPERSET_WEBSERVER_TIMEOUT = 3600  # Timeout serveur web
QUERY_TIMEOUT = 3600
SQLALCHEMY_QUERY_TIMEOUT = 3600  # Timeout SQLAlchemy
CACHE_DEFAULT_TIMEOUT = 60 * 60 * 24  # 24 heures
SQLLAB_DOCKER_TIMEOUT = 3600
# Flask App Builder configuration
SECRET_KEY = 'TdBVVxe3H5dmoJHL2ftGFY1AhH6Nq/7F2oruFOWfFYQ9/UfSGKWXLtrT'
# Flask-WTF flag for CSRF
WTF_CSRF_ENABLED = True
WTF_CSRF_EXEMPT_LIST = []
WTF_CSRF_TIME_LIMIT = 60 * 60 * 24 * 365  # 1 an
# Client-side override
 OVERRIDE_TIMEOUT = True
# Debugging and extra options
FLASK_APP_EXTRA_OPTIONS = {'debug': True}
```

Listing 2.14 – superset_config.py

2.8 Configuration de Fichier .bashrc

2.8.1 Configuration ajouté à .bashrc

```
export JAVA_HOME=/home/master/jdk1.8.0_92
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_HOME=$HOME/hadoop-3.3.6
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HIVE_HOME=/opt/hive
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HADOOP_CLASSPATH=$HADOOP_HOME/lib/*:$HADOOP_CONF_DIR
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/opt/hive/lib/mysql-connector-j-8.0.32.jar
export SPARK_HOME=/opt/spark  # Adaptez le chemin selon votre install>
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/home/master/spark_env/bin/python3
export PYSPARK_DRIVER_PYTHON=/home/master/spark_env/bin/python3
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
export SUPERSET_CONFIG_PATH="/home/master/superset/superset_config.py"
export FLASK_APP=superset

export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"

export NVM_DIR="$HOME/.config/nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

Listing 2.15 – .bashrc

3.1 Architecture

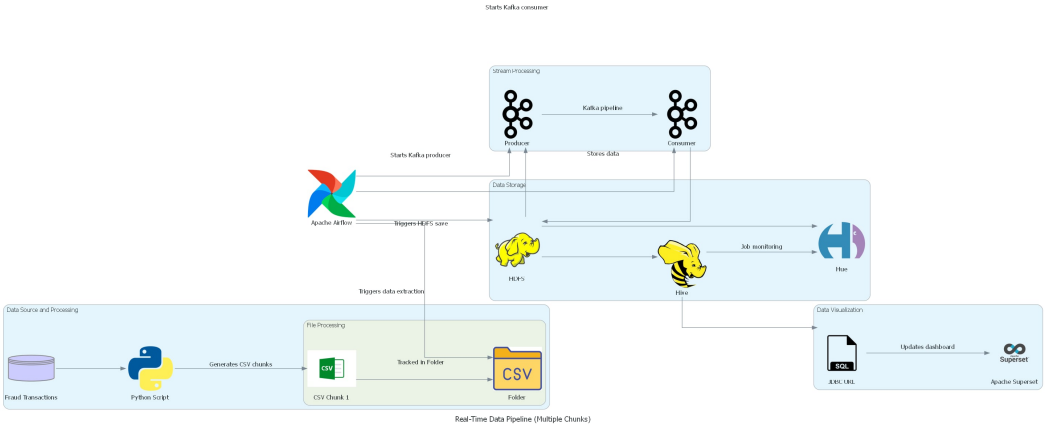

FIGURE 3.1 : Architecture BIGDATA

3.2 Activation des technologies

3.2.1 Activation de Hadoop et ces services

3.2.1.1 HDFS/YARN/Zookeeper

Hadoop Cluster Startup

Pour démarrer les clusters HDFS et YARN :

Démarrer les démons HDFS

— Démarrer le daemon **Namenode** sur la machine virtuelle du Namenode :

```
hdfs --daemon start namenode
```

— Démarrer les daemons **DataNode** sur les nœuds Datanode :

```
hdfs --daemon start datanode
```

Démarrer les démons YARN

— Démarrer le daemon **resourcemanager** sur le nœud Namenode :

```
yarn --daemon start resourcemanager
```

— Démarrer les daemons **nodemanager** sur les nœuds Datanode :

```
yarn --daemon start nodemanager
```

— Démarrer le serveur **JobHistory MapReduce** :

```
mapred --daemon start historyserver
```

Démarrer Zookeeper

Configuration

— Modifier les permissions des fichiers Zookeeper :

```
sudo chown -R master:master /opt/zookeeper/logs
sudo chown -R master:master /var/lib/zookeeper
```

Démarrer le serveur Zookeeper

— Démarrer le serveur :

```
/opt/zookeeper/bin/zkServer.sh start
```

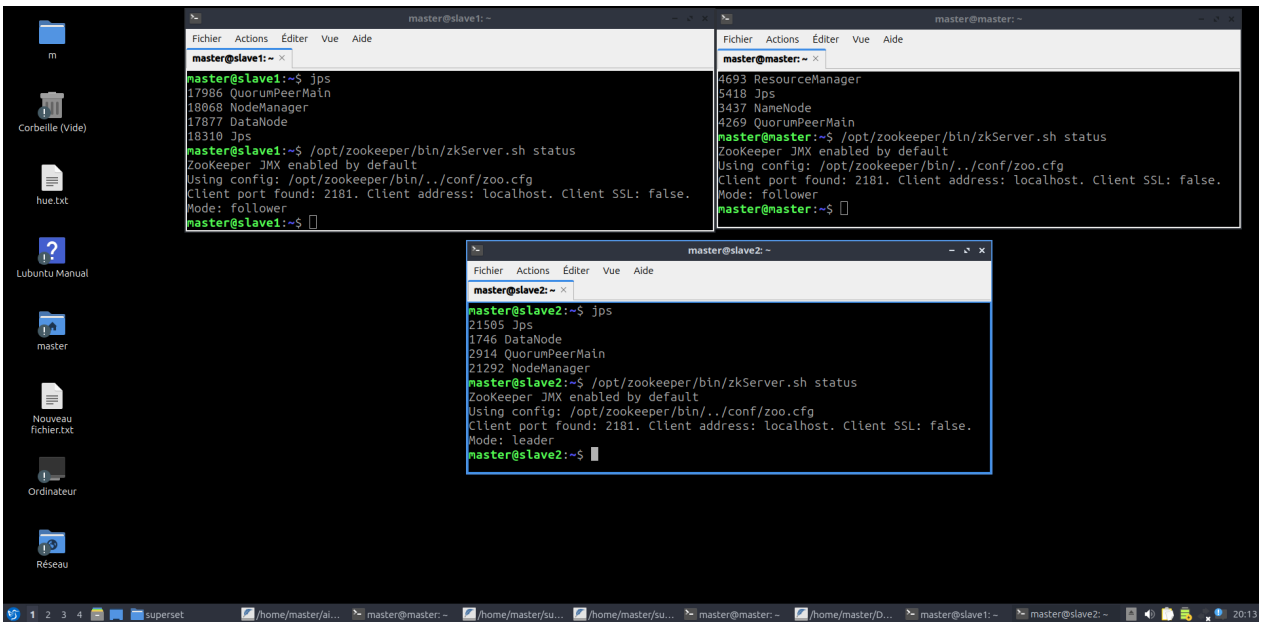

FIGURE 3.2 : activation de HDFS/Yarn/zoo

3.2.2 Activation de Hive et ces services

— Démarrer les services Hive :

```
hive --service metastore &
hive --service hiveserver2 &
```

— Connexion à Hive via Beeline :

```
beeline -u jdbc:hive2://localhost:10000/default -n hive -p password
```

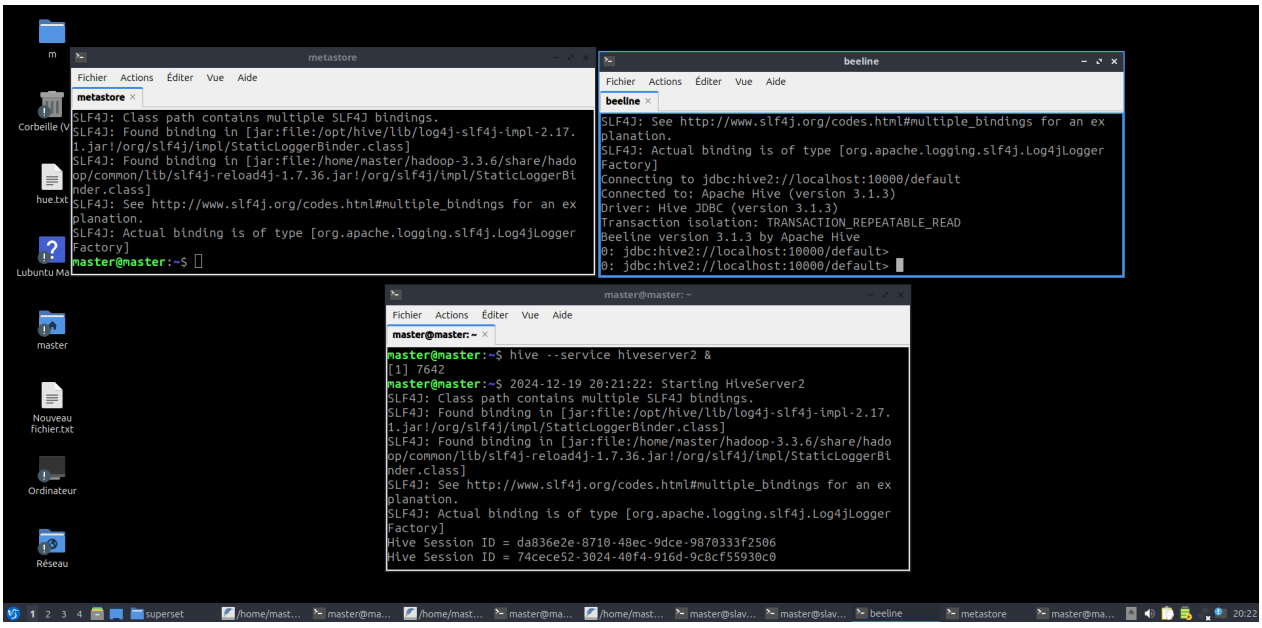

FIGURE 3.3 : Activation de hive

3.2.3 Activation de Superset

Configuration initiale

— Configurer Superset :

```
mkdir -p ~/superset
echo "SECRET_KEY = 'secret key'" > ~/superset/superset_config.py
superset db upgrade
superset init
```

Démarrer Superset

— Activer l'environnement Superset :

```
source superset_env/bin/activate
```

— Démarrer Superset :

```
superset run -p 8088 --with-threads --reload --debugger
```

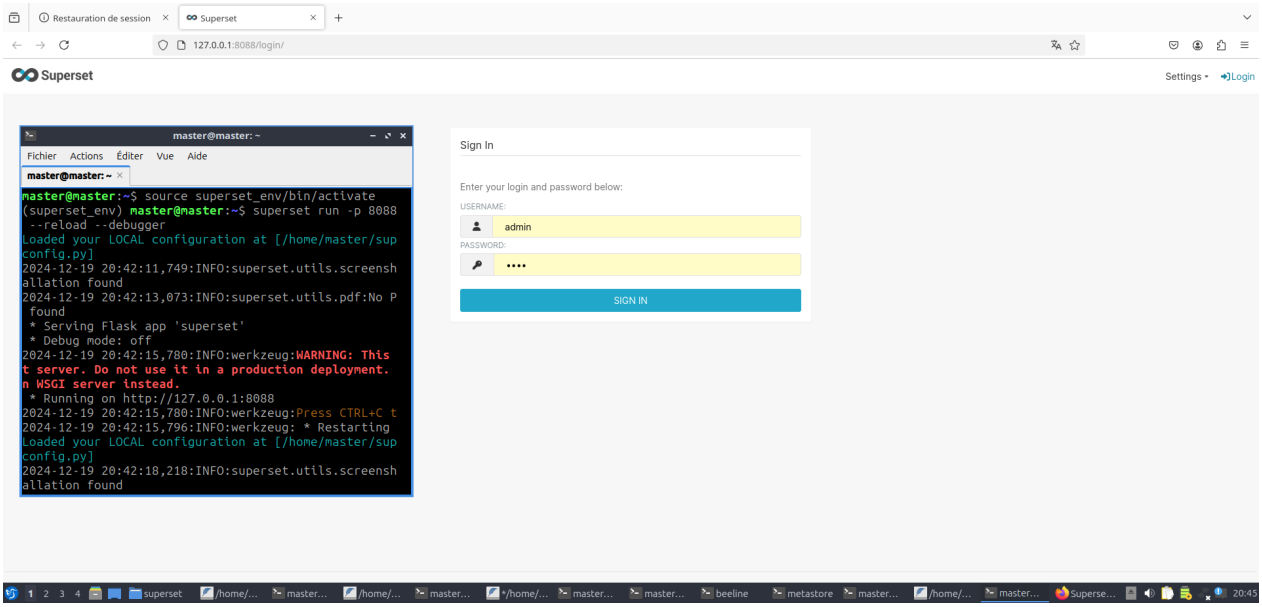

FIGURE 3.4 : Activation de superset

3.2.4 Activation de Hue

3.2.4.1 Démarrage de Hue

— Activer l'environnement Hue :

```
source /home/master/hue/build/env/bin/activate
```

— Démarrer le serveur Hue :

```
sudo -u master /home/master/hue
/build/env/bin/python /home/master/hue/build/env/bin/hue runserver 0.0.0.0:8000
```

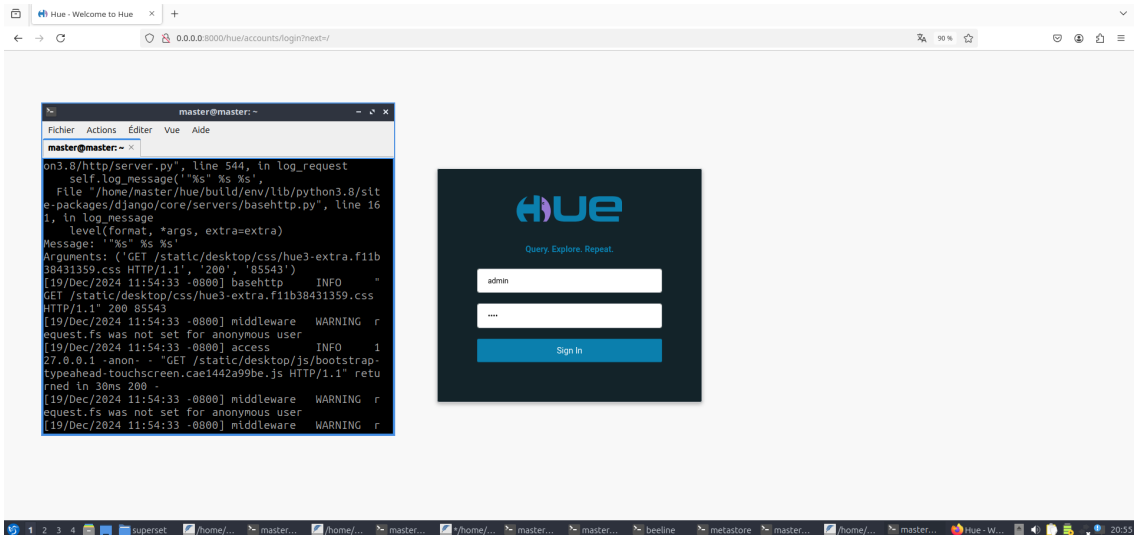

FIGURE 3.5 : Activation de Hue

3.2.5 Activation de Kafka

3.2.5.1 Démarrage des services Kafka

— Démarrer le serveur Kafka :

```
bin/kafka-server-start.sh config/server.properties
(executer dans tout les noeuds)
```

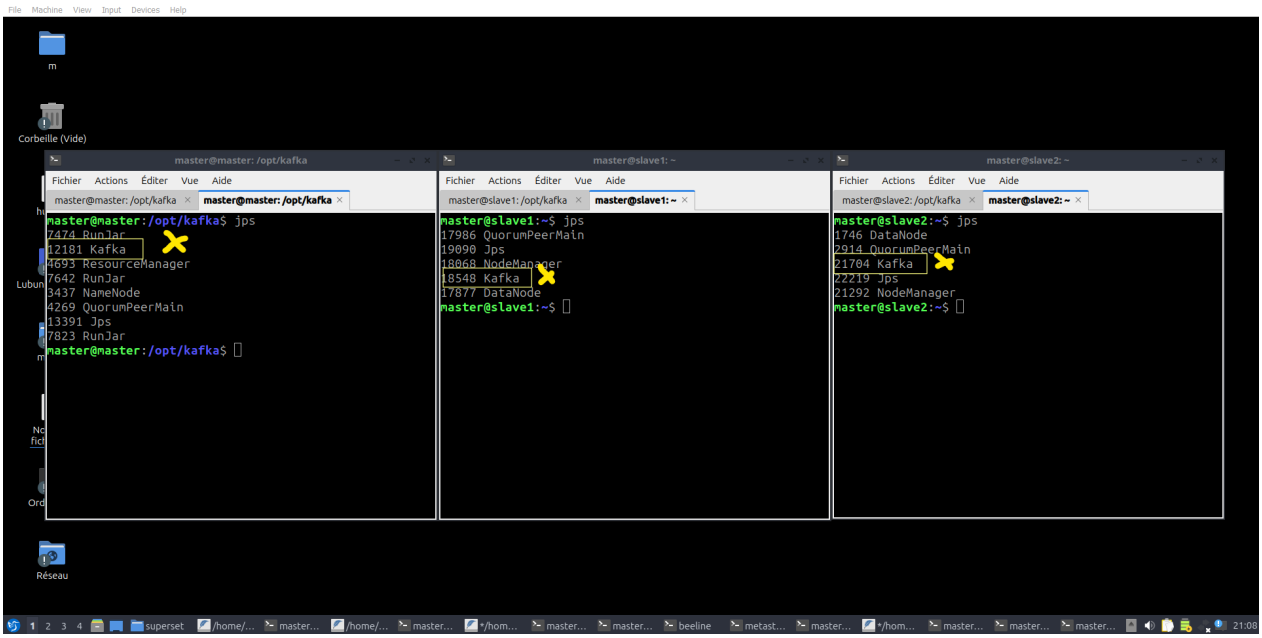

FIGURE 3.6 : Activation de Kafka

3.2.6 Activation de Airflow

— Afficher la liste des DAGs :

```
airflow dags list
```

— Démarrer le scheduler :

```
airflow scheduler
```

— Démarrer le serveur web Airflow :

```
airflow webserver --port 8081
```

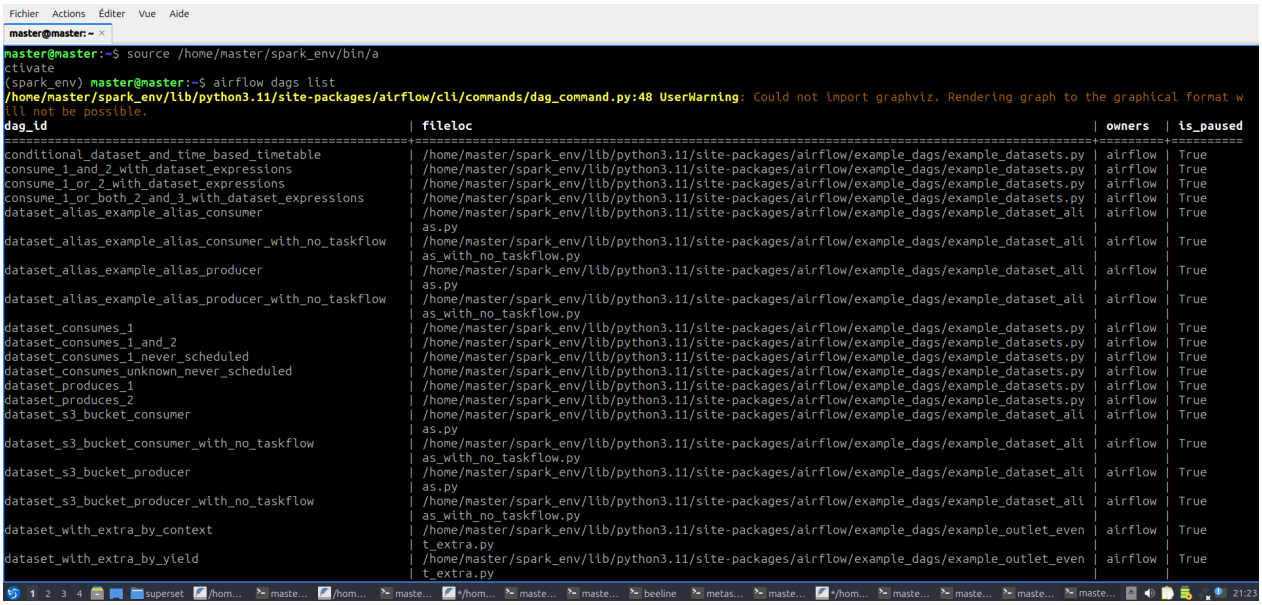

FIGURE 3.7 : listes des Dags (airflow)

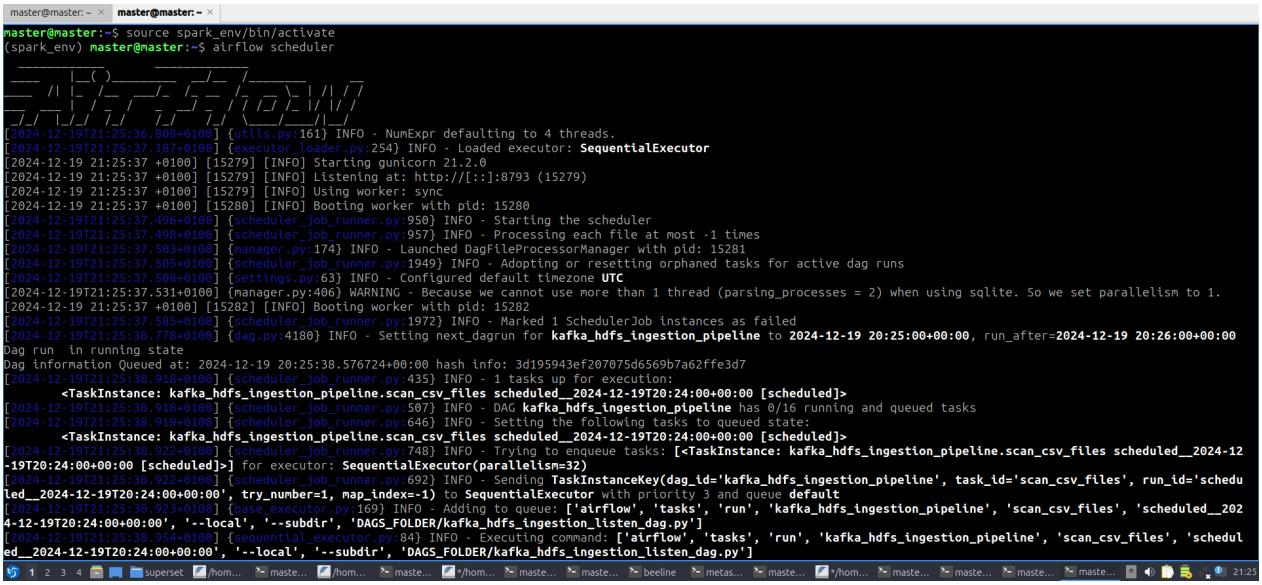

FIGURE 3.8 : Activer scheduler de Airflow

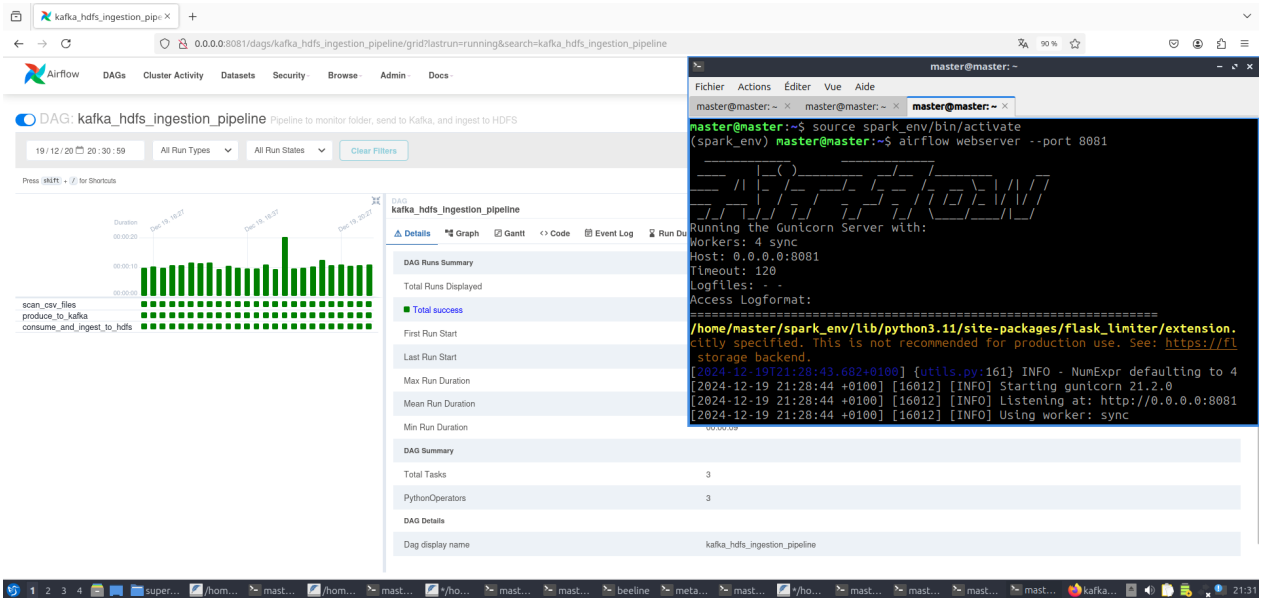

FIGURE 3.9 : Activation Airflow Webserver

3.3 Réalisation finale

3.3.1 Analyse sur dataset de transactions

La base de données `transactions.csv` contient un ensemble de **1 852 394** transactions, avec **23 colonnes** variées décrivant chaque transaction. Cette vaste base de données semble concerner des transactions financières, incluant des informations telles que les **dates et heures des transactions**, les **numéros de carte de crédit** (`cc_num`), les **commerçants** impliqués, les **catégories de dépenses**, ainsi que les **montants des transactions**

(`amt`). En outre, elle contient des données personnelles sur les utilisateurs, notamment leurs **prénoms**, **noms de famille**, **sexe**, **adresses**, **villes**, **états**, **codes postaux**, et **dates de naissance**.

Certaines informations **géographiques** telles que les **coordonnées GPS** (latitude et longitude) des commerçants et des utilisateurs sont également présentes, ce qui pourrait permettre de croiser des informations géographiques avec des habitudes de consommation. Les transactions sont accompagnées de données telles que le **numéro de transaction**, un identifiant **UNIX** pour la transaction (`unix_time`), et des informations sur l'occupation des utilisateurs, incluant leur **métier**. Enfin, la colonne `is_fraud` est utilisée pour indiquer si une transaction a été classifiée comme **frauduleuse**, avec une valeur de **1** pour frauduleuse et **0** pour non frauduleuse.

La base de données semble être principalement utilisée pour l'analyse de **fraudes** dans les transactions financières, avec un fort potentiel d'application dans les modèles d'**apprentissage automatique** et de détection des fraudes, notamment en croisant des informations **transactionnelles**, **géographiques** et **personnelles** pour prédire les comportements suspects.

3.3.2 Création de tableau "fraude" avec Hue

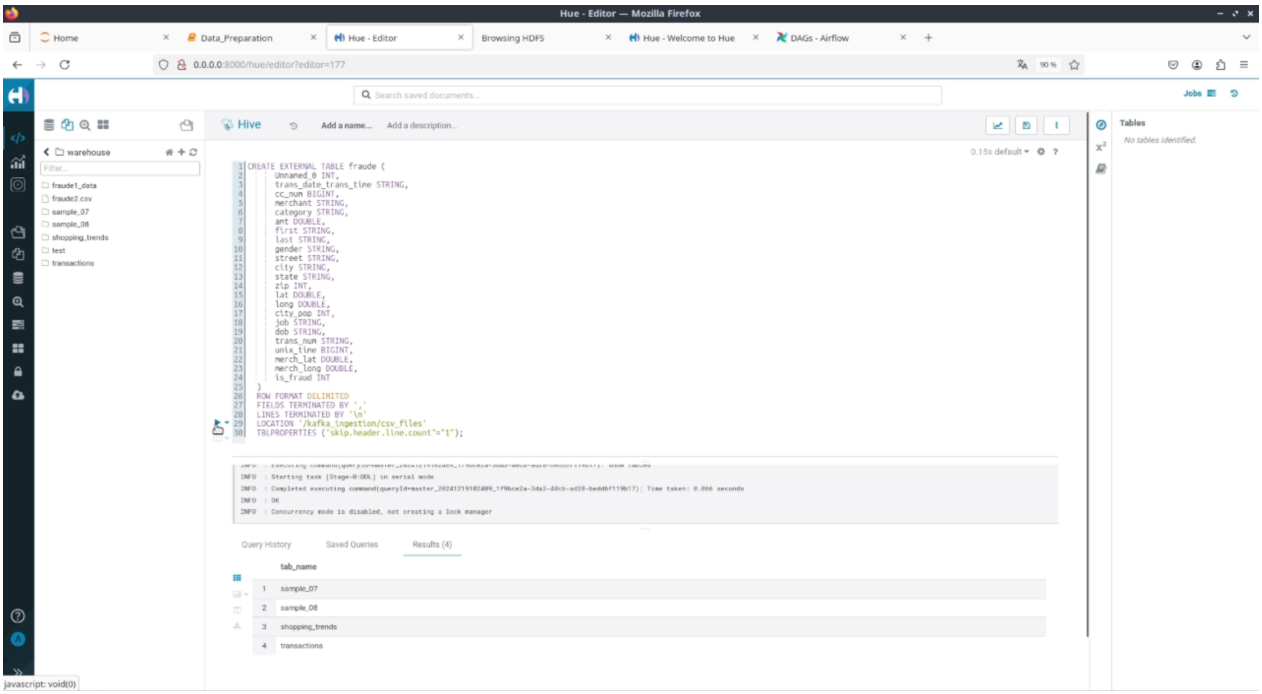

FIGURE 3.10 : création de tableau dans hive

Ce code SQL crée une **table externe** appelée `fraude` dans un système de gestion de bases de données(**Apache Hive**) s fichiers stockés à l'extérieur du système de base de données, comme dans **HDFS** (Hadoop Distributed File System) ou un autre emplacement de stockage.

La structure de la table `fraude` est définie avec **23 colonnes**, comprenant des informations telles que l'**ID de la transaction**, la **date et l'heure de la transaction**, le **numéro de carte de crédit**, le **commerçant**, la **catégorie**, le **montant de la transaction**, des informations personnelles sur les utilisateurs (**prénom**, **nom**, **sexe**, **adresse**, etc.), ainsi que des informations géographiques et de localisation (**latitude et longitude du commerçant et de l'utilisateur**). La colonne `is_fraud` est utilisée pour indiquer si la transaction est frauduleuse.

Le format des données est spécifié comme étant **délimité**, avec une virgule (,) comme séparateur de champs et un saut de ligne (`\n`) comme délimiteur de lignes. Le chemin d'accès aux fichiers CSV source est défini comme étant `/kafka_ingestion/csv_files`, ce qui indique où les fichiers de données sont stockés. Une propriété de table supplémentaire, `skip.header.line.count`, est définie à **1**, ce qui permet d'ignorer la première ligne du fichier (souvent utilisée pour les en-têtes de colonnes).

3.3.3 Division et l'envoie vers pipeline d'ingestion

Une fonction `split_csv_by_parts` qui permet de diviser le fichier CSV en plusieurs morceaux. La fonction prend en entrée le fichier transactions.csv (`input_file`), un répertoire de sortie (`output_dir`) et un nombre spécifié de morceaux (`num_parts`). Elle commence par créer le répertoire de sortie si nécessaire, puis elle compte le nombre total de lignes dans le fichier CSV (en excluant l'en-tête). Ensuite, elle calcule la taille de chaque morceau, lit le fichier par morceaux et écrit chaque morceau dans un nouveau fichier CSV. Un message est affiché pour chaque morceau sauvegardé. Dans l'exemple d'utilisation, le fichier `transactions_1.csv` est divisé en 10 morceaux et sauvegardé dans le répertoire `csv_batches`.

```
Chunk 1 sauvegardé : csv_batches/part_1.csv
Chunk 2 sauvegardé : csv_batches/part_2.csv
Chunk 3 sauvegardé : csv_batches/part_3.csv
Chunk 4 sauvegardé : csv_batches/part_4.csv
Chunk 5 sauvegardé : csv_batches/part_5.csv
Chunk 6 sauvegardé : csv_batches/part_6.csv
Chunk 7 sauvegardé : csv_batches/part_7.csv
Chunk 8 sauvegardé : csv_batches/part_8.csv
Chunk 9 sauvegardé : csv_batches/part_9.csv
Chunk 10 sauvegardé : csv_batches/part_10.csv
Chunk 11 sauvegardé : csv_batches/part_11.csv
```

FIGURE 3.11 : Division de fichier

Les fichiers CSV d'un répertoire source vers un répertoire de destination(repot où kakfa lire produce et consume et l envoie vers hdfs), un fichier à la fois. Il commence par lister tous les fichiers CSV dans le répertoire source `source_directory` et les trie par ordre alphabétique. Si le répertoire de destination `destination_directory` n'existe pas, il est créé. Ensuite, pour chaque fichier CSV, le programme attend que l'utilisateur appuie sur Entrée avant de déplacer le fichier du répertoire source vers le répertoire de destination. Un message est affiché après chaque transfert pour indiquer que le fichier a été envoyé avec succès. Enfin, une fois tous les fichiers envoyés, un message final confirme que tous les fichiers ont été transférés.

```
Appuyez sur Entrée pour envoyer chaque fichier un par un.
Prêt à envoyer part_1.csv? Appuyez sur Entrée pour continuer...
Fichier part_1.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_10.csv? Appuyez sur Entrée pour continuer...
Fichier part_10.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_11.csv? Appuyez sur Entrée pour continuer...
Fichier part_11.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_2.csv? Appuyez sur Entrée pour continuer...
Fichier part_2.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_3.csv? Appuyez sur Entrée pour continuer...
Fichier part_3.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_4.csv? Appuyez sur Entrée pour continuer...
Fichier part_4.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_5.csv? Appuyez sur Entrée pour continuer...
Fichier part_5.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_6.csv? Appuyez sur Entrée pour continuer...
Fichier part_6.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_7.csv? Appuyez sur Entrée pour continuer...
Fichier part_7.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_8.csv? Appuyez sur Entrée pour continuer...
Fichier part_8.csv envoyé vers /home/master/kafka_ingestion/.
Prêt à envoyer part_9.csv? Appuyez sur Entrée pour continuer...
Fichier part_9.csv envoyé vers /home/master/kafka_ingestion/.
Tous les fichiers ont été envoyés.
```

FIGURE 3.12 : Envoi les fichiers

le processus d'intégration et de traitement des fichiers `transactions` à l'aide d'Apache Kafka, ainsi que leur ingestion dans Hadoop HDFS, permet de gérer efficacement de grandes quantités de données distribuées. Le fichier source est découpé en segments plus petits et chaque segment est envoyé sous forme de message Kafka, garantissant une gestion distribuée et parallèle des données. Kafka agit comme un médiateur entre le producteur et le consommateur, permettant une ingestion continue et fiable des transactions.

Une fois les messages consommés, les données sont transformées et stockées dans HDFS, assurant une gestion et un traitement optimisés des données pour des analyses futures. L'automatisation de ce flux de travail avec des outils comme Apache Airflow renforce l'efficacité du système, garantissant une mise à jour en temps réel des données dans l'écosystème Hadoop.

Ce mécanisme permet non seulement de maintenir l'ordre et la cohérence des données, mais aussi de tirer parti des capacités de scalabilité et de fiabilité de Kafka et Hadoop pour des applications de grande envergure.

3.3.4 Execution de pipline d'ingestion avec airflow

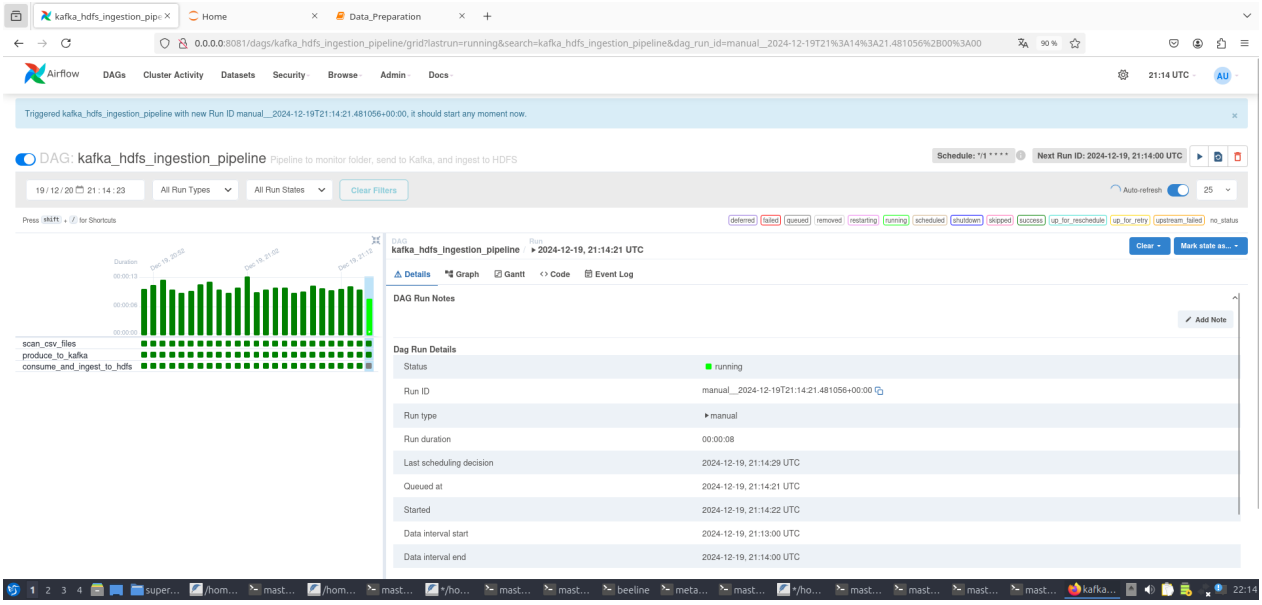

FIGURE 3.13 : Execution de Dag

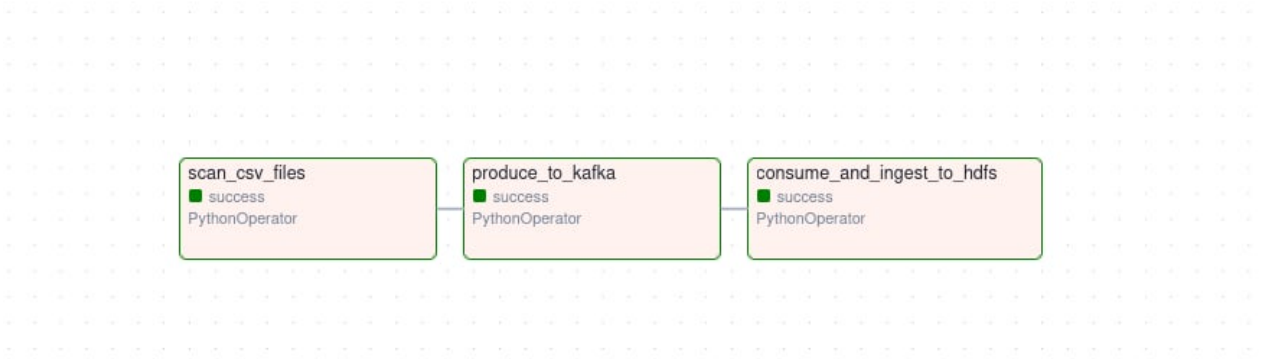

FIGURE 3.14 : Pipeline

L'intégration d'Airflow dans ce processus permet de coordonner toutes les étapes de l'ingestion, du traitement et de la visualisation des données de manière fluide et automatisée. Cela garantit une mise à jour continue des tableaux de bord, permettant aux utilisateurs finaux d'avoir accès aux données les plus récentes en temps réel. Grâce à Kafka, HDFS, Hive, et Apache Superset, ce processus devient hautement évolutif et fiable, capable de gérer de grandes quantités de données tout en minimisant les erreurs humaines.

3.3.5 Mise à jour de tableau et des visualisations en temps réelle

Avant L'ingestion total des transactions

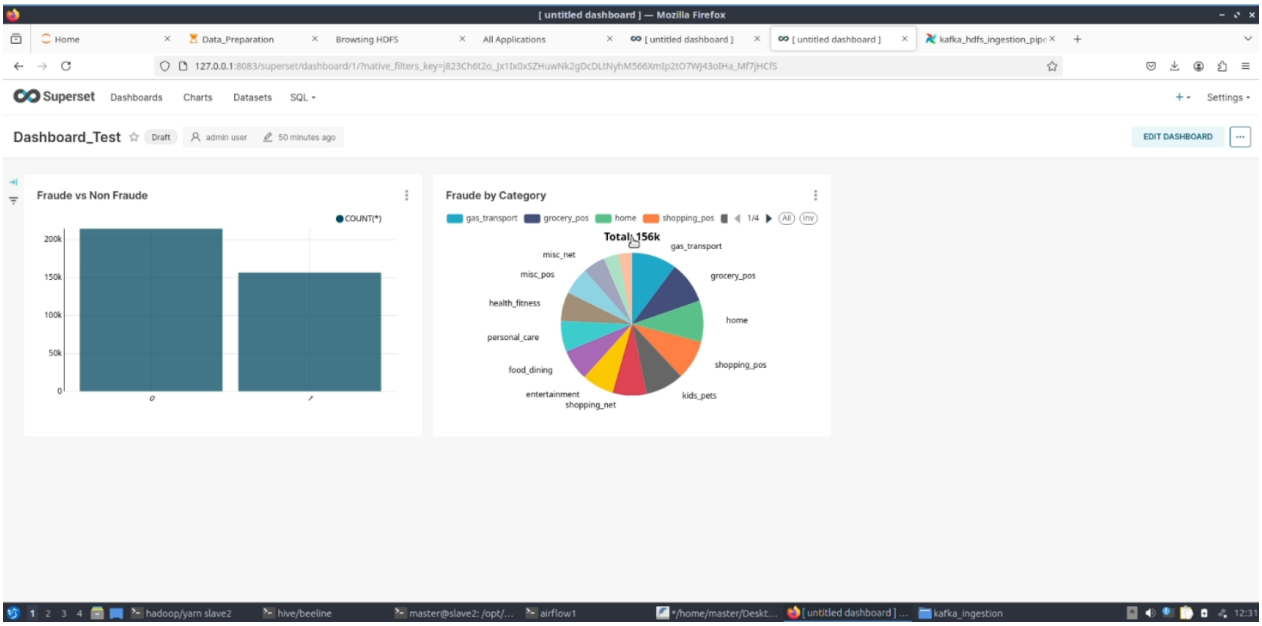

FIGURE 3.15 : Visualisation avant l'ingestion totale de données

Aprés L'ingestion total des transactions

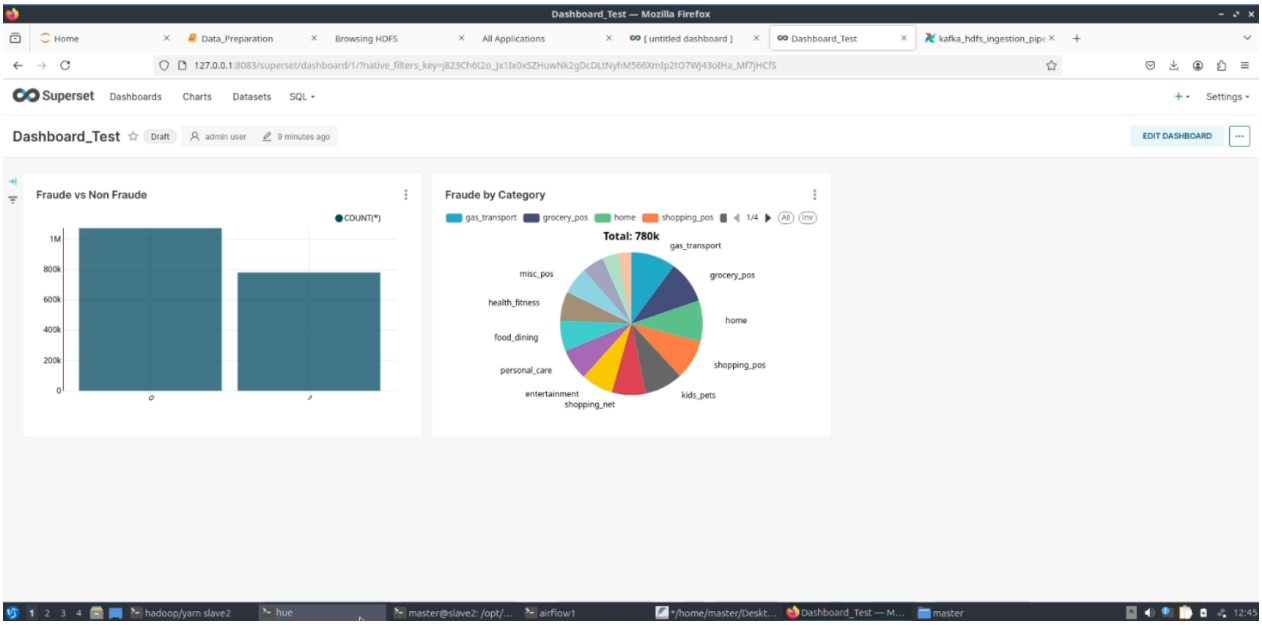

FIGURE 3.16 : Visualisation aprés l'ingestion totale de données

3.3.6 Dashboard Final avec superset

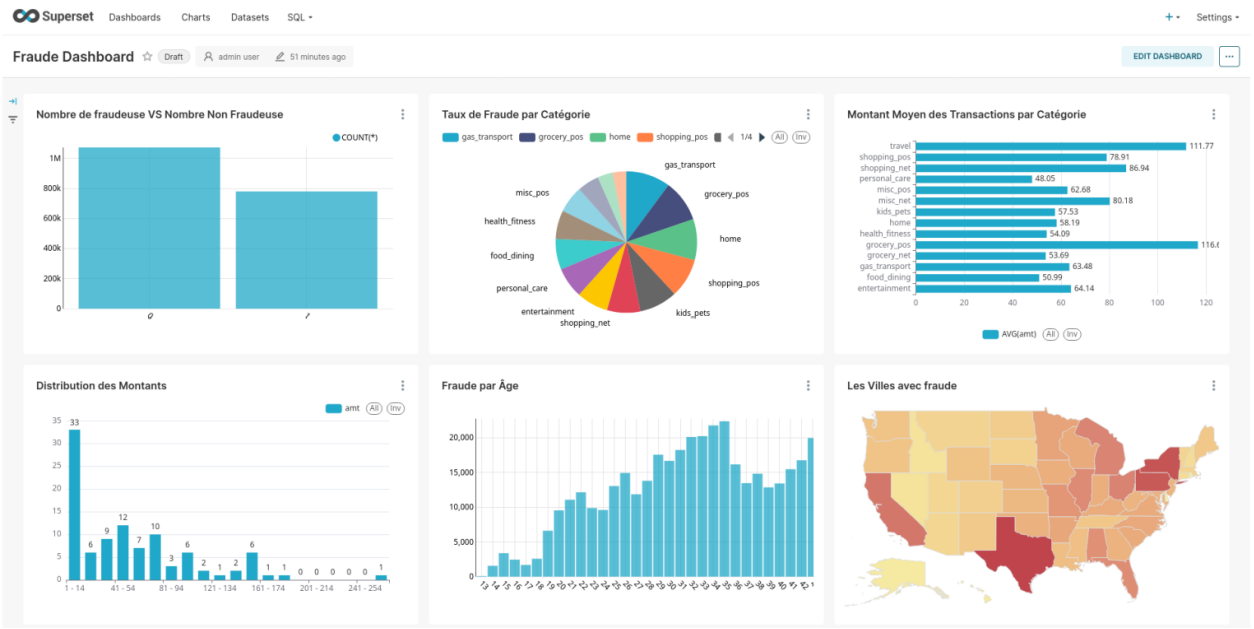

FIGURE 3.17 : Dashboard Final avec superset

Conclusion Globale

Ce projet a permis de mettre en œuvre une architecture **Big Data** robuste et évolutive pour le traitement, la gestion, et la visualisation des données en temps réel. En intégrant des technologies modernes telles que **Apache Kafka**, **Hadoop** (HDFS, Hive), **Apache Airflow**, et **Apache Superset**, nous avons conçu un pipeline complet répondant aux exigences de traitement des volumes de données massifs tout en assurant une automatisation et une optimisation des performances.

Bilan du Projet

- **Ingestion et stockage des données :** Grâce à Kafka, les fichiers de transactions sont efficacement collectés et traités, avant d'être stockés dans HDFS, garantissant ainsi une persistance fiable et scalable.
- **Traitement et mise à jour automatisée :** L'utilisation de Hive pour structurer les données a permis d'automatiser les mises à jour des tables et des partitions, facilitant ainsi leur exploitation pour l'analyse.
- **Visualisation en temps réel :** Avec Superset, les données sont rendues accessibles via des tableaux de bord interactifs et dynamiques, offrant une compréhension immédiate et approfondie des tendances et métriques.
- **Orchestration des processus :** Airflow a joué un rôle crucial en orchestrant l'ensemble du pipeline, garantissant la coordination des tâches, la surveillance des erreurs, et la planification des exécutions, réduisant ainsi la complexité opérationnelle.

Perspectives d'Amélioration

Bien que ce projet ait démontré son efficacité, des améliorations futures pourraient être envisagées :

- **Intégration d'un système de monitoring avancé** (comme Prometheus ou Grafana) pour surveiller la performance des services.
- **Optimisation des performances** des requêtes Hive en utilisant des formats de données comme Parquet ou ORC.
- **Mise en œuvre de mécanismes de sécurité avancés**, comme le chiffrement des données et l'authentification renforcée, pour protéger les données sensibles.

Conclusion Finale

En conclusion, ce projet illustre l'efficacité des technologies **Big Data** pour relever les défis liés au traitement de grandes quantités de données en temps réel. Il offre une solution flexible, scalable et automatisée, tout en posant les bases pour des extensions futures. Cette expérience a également renforcé notre compréhension des systèmes distribués et des outils de traitement des données modernes, offrant une valeur ajoutée significative pour des projets similaires à venir.