

---

# ARCHITECTURE BIG DATA:

## Projet Avocado

---



### Description:

Le **projet Avocado** consiste à intégrer des données d'avocats depuis un **fichier CSV** dans une base de données **MySQL**, puis à les traiter et les analyser en utilisant **Apache NiFi**, **HDFS**, **Spark** et **Hive**. Les données sont transformées, agrégées, et stockées à différentes étapes du **pipeline** de traitement. Enfin, des tâches automatisées avec cron sont mises en place pour **exécuter périodiquement** le processus complet.

### Réalisé par:

Afdel Desmond KOMBOU

Papa Yeriba NIANG

### Sous la supervision:

Mr Patrick NGOUNE

---

# I - HDFS (Hadoop Distributed File System)

**Bon à savoir:** **HDFS** est un système de fichiers distribué conçu pour stocker et gérer de très gros volumes de données sur un grand nombre de machines équipées de disques durs banalisés. Il est l'un des composants clés du framework Hadoop Apache, un logiciel open-source pour le traitement de données massives.

## 1- Démarrage des service hadoop

```
hadoop@afdelk:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [afdelk]
afdelk: ssh: connect to host afdelk port 22: Connection timed out
Starting resourcemanager
Starting nodemanagers
hadoop@afdelk:~$ JPS
JPS : commande introuvable
hadoop@afdelk:~$ jps
24869 NameNode
26053 Jps
25478 ResourceManager
25063 DataNode
25615 NodeManager
```

## 2- Création des des répertoires **HDFS**

- '/raw\_avocado'
- '/staging\_avocado',
- '/refine\_avocado'

```
hadoop@afdelk:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [afdelk]
afdelk: ssh: connect to host afdelk port 22: Connection timed out
Starting resourcemanager
Starting nodemanagers
hadoop@afdelk:~$ JPS
JPS : commande introuvable
hadoop@afdelk:~$ jps
24869 NameNode
26053 Jps
25478 ResourceManager
25063 DataNode
25615 NodeManager
hadoop@afdelk:~$ hdfs dfs -mkdir /raw_avocado
hadoop@afdelk:~$ hdfs dfs -mkdir /staging_avocado
hadoop@afdelk:~$ hdfs dfs -mkdir /refine_avocado
hadoop@afdelk:~$
```

# II - MYSQL (MY Structured Query Language)

**Bon à savoir:** **MYSQL** est un système de gestion de bases de données relationnelles (SGBDR). Il fait partie des logiciels de gestion de base de données les plus utilisés au monde

## Connexion au serveur **MySQL**

```

5 juin 20:07
afdel@afdelk: ~
afdel@afdelk:~$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.36-2ubuntu3 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

```

### 1- Créer une **bd MySQL** nommé 'TP\_MASTER'

```

mysql> CREATE DATABASE TP_MASTER;
Query OK, 1 row affected (0,03 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| TP_MASTER |
| afdelk |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
6 rows in set (0,00 sec)

mysql>

```

### 2- Créer une table **MySQL** nommée **TP\_MASTER.'avocado'** respectant la structure du fichier 'avocado.csv'

```

mysql>
mysql> USE TP_MASTER;
Database changed
mysql>
mysql> CREATE TABLE avocado (
->     Date DATE,
->     AveragePrice FLOAT,
->     Volume FLOAT,
->     type VARCHAR(20),
->     year INT,
->     region VARCHAR(50)
-> );
Query OK, 0 rows affected (0,07 sec)

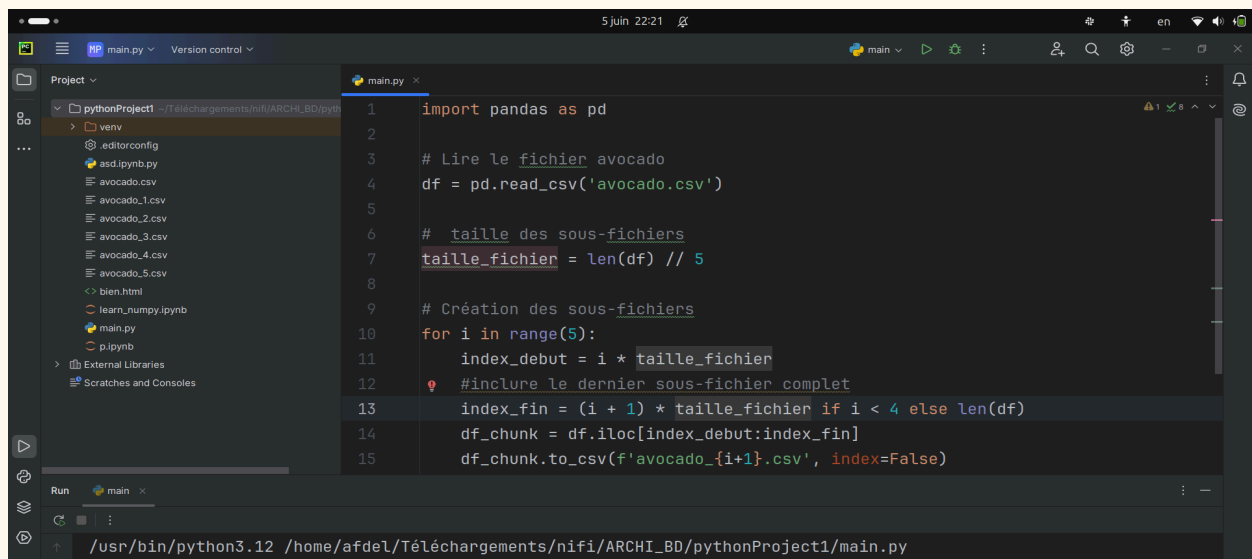
mysql>

```

### 3- Scinder le fichier avocado.csv en 5 fichiers (avocado\_1.csv, avocado\_2.csv, avocado\_3.csv, avocado\_4.csv, avocado\_5.csv)

#### Méthode utilisée : Script python (bibliothèque pandas)

Nous avons utilisé la bibliothèque Pandas pour diviser le fichier CSV principal ('avocado.csv') en cinq sous-fichiers de taille approximativement égale. le script itère à travers ces sous-fichiers en découpant le **DataFrame** en tranches de données, qu'il écrit ensuite individuellement en CSV sous le nom 'avocado\_1.csv', 'avocado\_2.csv', etc.



```

1 import pandas as pd
2
3 # Lire le fichier avocado
4 df = pd.read_csv('avocado.csv')
5
6 # taille des sous-fichiers
7 taille_fichier = len(df) // 5
8
9 # Création des sous-fichiers
10 for i in range(5):
11     index_debut = i * taille_fichier
12     #inclure le dernier sous-fichier complet
13     index_fin = (i + 1) * taille_fichier if i < 4 else len(df)
14     df_chunk = df.iloc[index_debut:index_fin]
15     df_chunk.to_csv(f'avocado_{i+1}.csv', index=False)
  
```

### 4- Chargez le premier fichier 'avocado\_1.csv' sur la table **MYSQL TP\_MASTER.'avocado'**

méthode utilisée : LOAD DATA INFILE '/chemin/vers/fichier/' INTO TABLE 'nom de la table';

Nous avons chargé le premier fichier 'avocado\_1.csv' dans la table **MySQL TP\_MASTER.'avocado'** en utilisant la méthode **LOAD DATA INFILE**. Nous avons ajouté des options comme **FIELDS TERMINATED BY ','** pour spécifier le délimiteur de champ et **IGNORE 1 LINES** pour sauter la première ligne si nécessaire.

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/avocado_1.csv' INTO TABLE avocado FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES (Date, AveragePrice, Volume, type, year, region);
Query OK, 67 rows affected (0.02 sec)
Records: 67 Deleted: 0 Skipped: 0 Warnings: 0

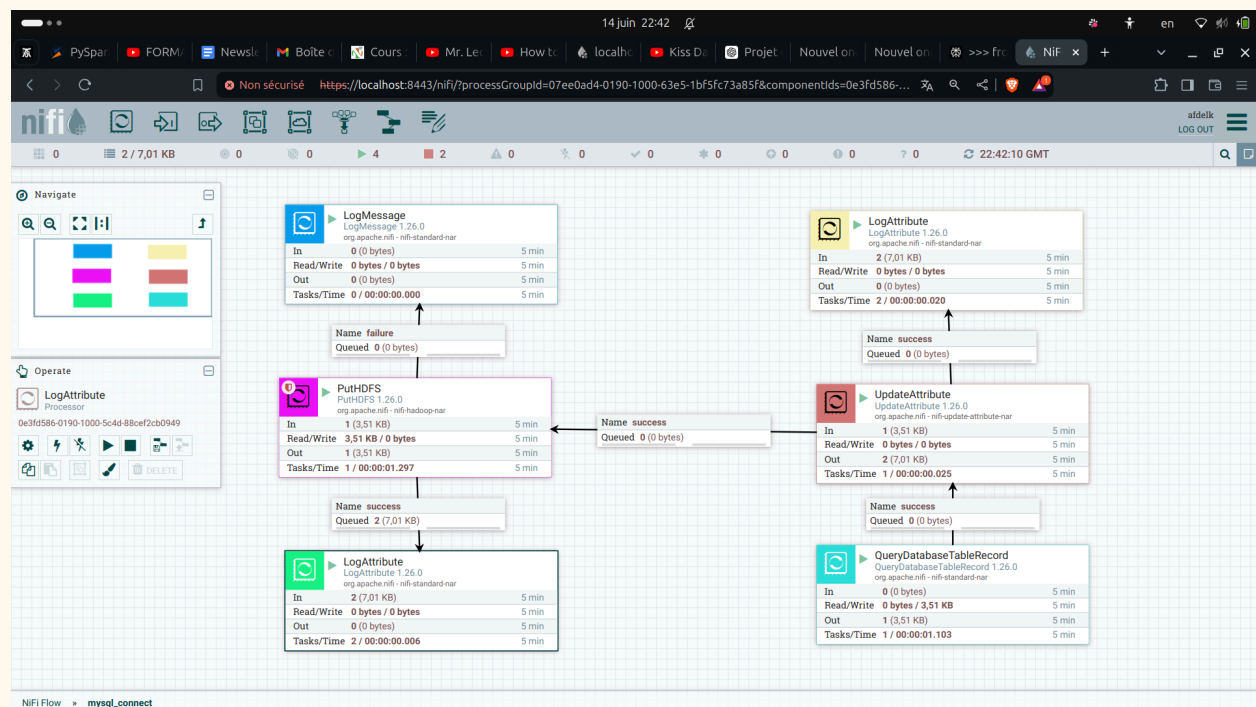
mysql> show tables;
+-----+
| Tables_in_TP_MASTER |
+-----+
| avocado              |
+-----+
1 row in set (0.01 sec)

mysql> select * from avocado;
+-----+-----+-----+-----+-----+-----+
| Date       | AveragePrice | Volume | type       | year | region |
+-----+-----+-----+-----+-----+-----+
| 2015-01-04 | 1            | 435022 | conventional | 2015 | Atlanta |
| 2015-01-04 | 1.22         | 40873.3 | conventional | 2015 | Albany  |
| 2015-01-11 | 1.24         | 41195.1 | conventional | 2015 | Albany  |
| 2015-01-11 | 1.11         | 397543 | conventional | 2015 | Atlanta |
| 2015-01-18 | 1.17         | 44511.3 | conventional | 2015 | Albany  |
| 2015-01-18 | 1.11         | 431491 | conventional | 2015 | Atlanta |
| 2015-01-25 | 1.06         | 45147.5 | conventional | 2015 | Albany  |
+-----+-----+-----+-----+-----+-----+
```

### III - NIFI (Niagara Files)

**Bon à savoir:** NIFI permet de gérer et d'automatiser des flux de données entre plusieurs systèmes informatiques, à partir d'une interface web et dans un environnement distribué.

#### 1- créer un process permettant d'ingérer les données de cette table **MYSQL**



Nous avons structuré notre workflow NiFi comme suit pour automatiser le traitement et l'ingestion de données :

- **QueryDatabaseTableRecord** : Nous interrogeons la base de données pour extraire les enregistrements nécessaires, garantissant que nous obtenons les données les plus récentes.
- **UpdateAttribute** : Nous modifions ou ajoutons des attributs aux données extraites pour préparer les métadonnées nécessaires avant de les stocker.
- **LogAttribute** : Nous enregistrons les attributs des données pour des fins de diagnostic et de vérification, assurant ainsi une traçabilité.
- **PutHDFS** : Nous chargeons les données dans HDFS pour les rendre disponibles pour des traitements ultérieurs avec des outils big data comme Spark.
- **LogAttribute** (après PutHDFS) : Nous vérifions et enregistrons les attributs des données après leur écriture dans HDFS pour confirmer l'opération.
- **LogMessage** : Nous consignons des messages de journal spécifiques pour une surveillance et un suivi détaillés de l'exécution du workflow.

Cette configuration a été choisie pour garantir un flux de données transparent et contrôlé, permettant de vérifier à chaque étape et de garantir que les données sont correctement extraites, transformées, et stockées

**2- stocker le résultat dans le répertoire HDFS '/raw\_avocado' dans un fichier csv dont le nom sera au format'avocado\_YYYYMMJJHHmm.csv'**

```
hadoop@afdelk:~/nifi-1.26.0/bin$ ./nifi.sh start

Java home: /usr/lib/jvm/java-8-openjdk-amd64
NiFi home: /home/hadoop/nifi-1.26.0

Bootstrap Config File: /home/hadoop/nifi-1.26.0/conf/bootstrap.conf

hadoop@afdelk:~/nifi-1.26.0/bin$ cd
hadoop@afdelk:~$ hadoop fs -ls hdfs:///raw_avocado
Found 4 items
-rwxr-xr-x 1 hadoop supergroup 3591 2024-06-12 21:01 hdfs:///raw_avocado/avocado_20240612210119.csv
```

## IV - HIVE partie 1

**Bon à savoir:** **HIVE** permet aux utilisateurs de lire, d'écrire et de gérer des pétaoctets de données à l'aide de SQL

- Créer une table interne **HIVE** nommé **TP\_MASTER**. 'avocado\_volume\_tracking' avec 2 champs 'filename' (STR) et 'somme\_volume' (FLOAT)

```

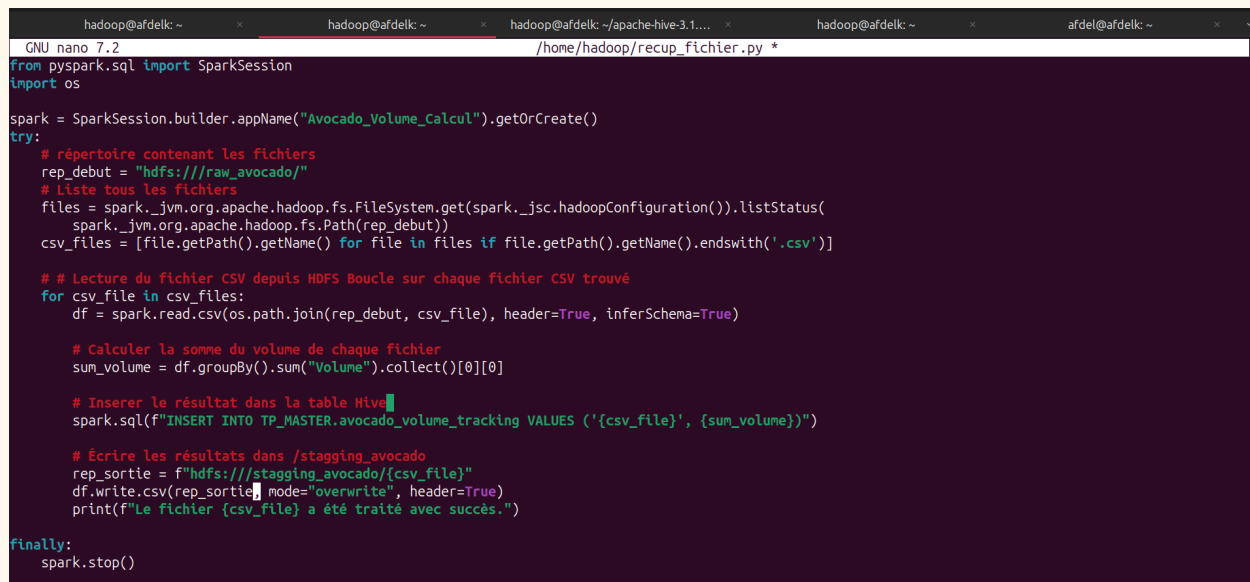
Time taken: 0.037 seconds, Fetched: 2 row(s)
hive> CREATE DATABASE IF NOT EXISTS TP_MASTER;
OK
Time taken: 0.213 seconds
hive> USE TP_MASTER;
OK
Time taken: 0.046 seconds
hive> CREATE TABLE IF NOT EXISTS avocado_volume_tracking (
>     filename STRING,
>     somme_volume FLOAT
> );
OK
Time taken: 0.533 seconds
hive> show tables;
OK
avocado_volume_tracking
Time taken: 0.037 seconds, Fetched: 1 row(s)
hive>

```

## V - SPARK partie 1

Écrire un script permettant de récupérer le fichier déposé dans '/raw\_avocado' et faire les opérations nécessaires pour alimenter la table HIVE 'avocado\_volume\_tracking'. Ensuite, déplacer le fichier csv de '/raw\_avocado' vers '/staging\_avocado'

### 1- Editer script **pyspark**



```

GNU nano 7.2 /home/hadoop/recup_fichier.py
from pyspark.sql import SparkSession
import os

spark = SparkSession.builder.appName("Avocado_Volume_Calcul").getOrCreate()
try:
    # répertoire contenant les fichiers
    rep_debut = "hdfs:///raw_avocado/"
    # Liste tous les fichiers
    files = spark._jvm.org.apache.hadoop.fs.FileSystem.get(spark._jsc.hadoopConfiguration()).listStatus(
        spark._jvm.org.apache.hadoop.fs.Path(rep_debut))
    csv_files = [file.getPath().getName() for file in files if file.getPath().getName().endswith('.csv')]

    # Lecture du fichier CSV depuis HDFS Boucle sur chaque fichier CSV trouvé
    for csv_file in csv_files:
        df = spark.read.csv(os.path.join(rep_debut, csv_file), header=True, inferSchema=True)

        # Calculer la somme du volume de chaque fichier
        sum_volume = df.groupBy().sum("Volume").collect()[0][0]

        # Insérer le résultat dans la table Hive
        spark.sql(f"INSERT INTO TP_MASTER.avocado_volume_tracking VALUES ('{csv_file}', {sum_volume})")

        # Écrire les résultats dans /staging_avocado
        rep_sortie = f"hdfs:///staging_avocado/{csv_file}"
        df.write.csv(rep_sortie, mode="overwrite", header=True)
        print(f"Le fichier {csv_file} a été traité avec succès.")
finally:
    spark.stop()

```

Ce script PySpark permet de lire les fichiers (avocado) CSV à partir de HDFS, calculer le volume total pour chaque fichier, et stocker les résultats dans une table Hive ainsi que dans un répertoire HDFS.

→ **SparkSession** : Nous utilisons Spark pour traiter efficacement les grands volumes de données CSV.

- **Liste des fichiers** : Nous listons tous les fichiers CSV dans le répertoire HDFS spécifié.
- **Lecture et calcul** : Pour chaque fichier CSV, nous lisons les données, calculons la somme du volume d'avocats, et collectons les résultats.
- **Insertion dans Hive** : Les résultats sont insérés dans une table Hive pour une analyse structurée et persistante.
- **Écriture dans HDFS** : Les fichiers traités sont ensuite écrits dans un nouveau répertoire HDFS pour un stockage centralisé et pour faciliter des traitements ultérieurs.

## 2- Exécution du fichier dans **pyspark**

```
>>> exec(open("/home/hadoop/recup_fichier.py").read())
2024-06-15 18:38:39,080 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
2024-06-15 18:38:40,166 WARN conf.HiveConf: HiveConf of name hive.metastore.wm.default.pool.size does not exist
2024-06-15 18:38:40,166 WARN conf.HiveConf: HiveConf of name hive.llap.task.scheduler.preempt.independent does not exist
2024-06-15 18:38:40,167 WARN conf.HiveConf: HiveConf of name hive.llap.output.format.arrow does not exist
2024-06-15 18:38:40,167 WARN conf.HiveConf: HiveConf of name hive.tez.llap.min.reducer.per.executor does not exist
2024-06-15 18:38:40,167 WARN conf.HiveConf: HiveConf of name hive.tez.llap.min.reducer.per.executor does not exist
2024-06-15 18:38:40,182 WARN conf.HiveConf: HiveConf of name hive.lock.query.string.max.length does not exist
2024-06-15 18:38:40,182 WARN conf.HiveConf: HiveConf of name hive.llap.io.track.cache.usage does not exist
2024-06-15 18:38:40,182 WARN conf.HiveConf: HiveConf of name hive.use.orc.codec.pool does not exist
2024-06-15 18:38:40,182 WARN conf.HiveConf: HiveConf of name hive.query.results.cache.max.size does not exist
2024-06-15 18:38:40,182 WARN conf.HiveConf: HiveConf of name hive.repl.bootstrap.dump.open.txn.timeout does not exist
Le fichier avocado_20240612210119.csv a été traité avec succès.
```

Le script `recup_fichier` s'est exécuté sans erreur

## 3- Vérification

```
hive> use tp_master;
OK
Time taken: 0.035 seconds
hive> show tables;
OK
avocado_volume_tracking
Time taken: 0.06 seconds, Fetched: 1 row(s)
hive> select * from avocado_volume_tracking;
OK
avocado_20240612210119.csv      1.8028781400000002E7
Time taken: 2.126 seconds, Fetched: 1 row(s)
hive> █
```

La table `volume_tracking` contient belle et bien les information souhaité à savoir le nom du fichier et son volume

## 4- Déplacer les fichier de '/raw\_avocado' vers '/staging\_avocado'

```
hadoop@afdelk:~$ hadoop fs -ls hdfs:///staging_avocado
Found 1 items
drwxr-xr-x - hadoop supergroup          0 2024-06-15 18:38 hdfs:///staging_avocado/avocado_20240612210119.csv
hadoop@afdelk:~$ █
```

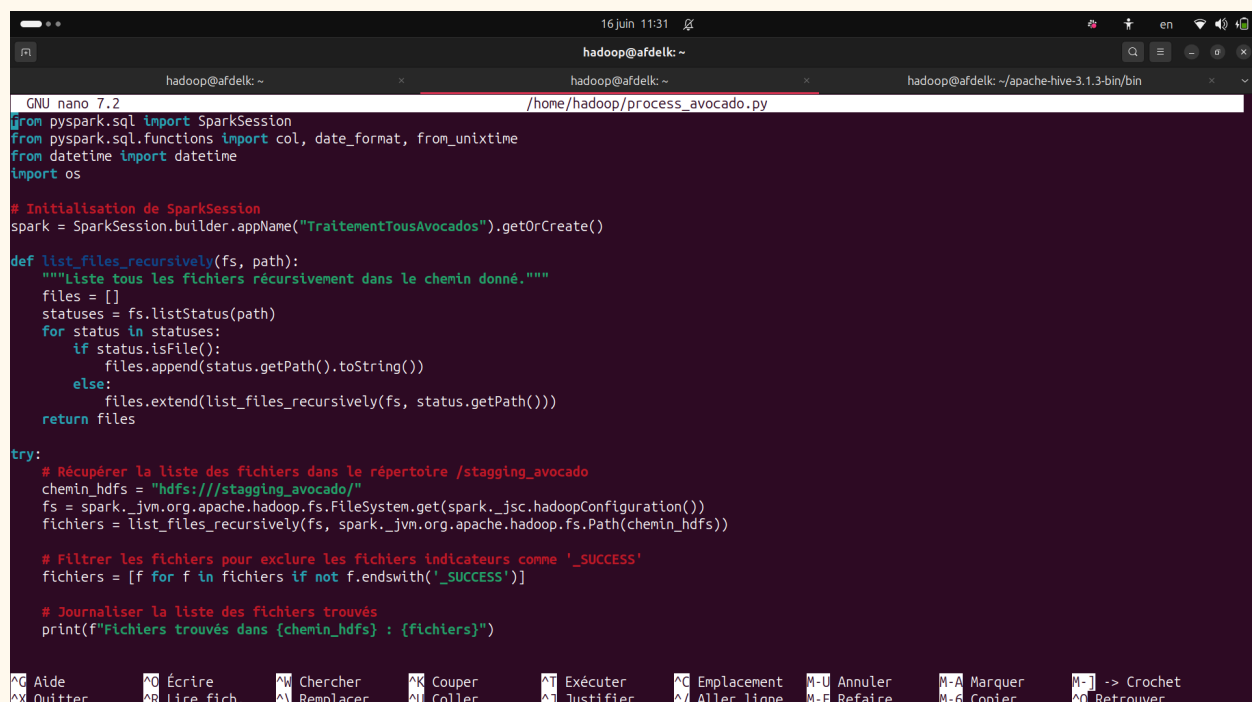


Nous remarquons que les fichier se trouvant préalablement dans le repertoire raw\_avocado son présentement dans le repertoire staging\_avocado

## VI - SPARK partie 2

Écrire un script PYSPARK qui récupère le fichier csv dans '/staging\_avocado' et pour chaque ligne du fichier, ajoute les colonnes 'jour' et 'mois' qui seront des extractions du champ 'date'. Sauvegardez le nouveau résultat dans un fichier csv (avocado\_cleaned\_YYYYMMJJHHmm.csv, ...) et le stocker dans '/refine\_avocado', ensuite supprimez le fichier ('avocado\_YYYYMMJJHHmm.csv', ...) du répertoire '/staging\_avocado'

### 1- script pyspark



```
GNU nano 7.2 /home/hadoop/process_avocado.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, date_format, from_unixtime
from datetime import datetime
import os

# Initialisation de SparkSession
spark = SparkSession.builder.appName("TraitementTousAvocados").getOrCreate()

def list_files_recursively(fs, path):
    """Liste tous les fichiers récursivement dans le chemin donné."""
    files = []
    statuses = fs.listStatus(path)
    for status in statuses:
        if status.isFile():
            files.append(status.getPath().toString())
        else:
            files.extend(list_files_recursively(fs, status.getPath()))
    return files

try:
    # Récupérer la liste des fichiers dans le répertoire /staging_avocado
    chemin_hdfs = "hdfs:///staging_avocado/"
    fs = spark._jvm.org.apache.hadoop.fs.FileSystem.get(spark._jsc.hadoopConfiguration())
    fichiers = list_files_recursively(fs, spark._jvm.org.apache.hadoop.fs.Path(chemin_hdfs))

    # Filtrer les fichiers pour exclure les fichiers indicateurs comme '_SUCCESS'
    fichiers = [f for f in fichiers if not f.endswith('_SUCCESS')]

    # Journaliser la liste des fichiers trouvés
    print(f"Fichiers trouvés dans {chemin_hdfs} : {fichiers}")
except Exception as e:
    print(f"Erreur : {e}")
```

Ce script PySpark traite les fichiers CSV d'avocats dans un répertoire HDFS en les nettoyant et en les formatant, puis en les déplaçant vers un répertoire HDFS de sortie.

- **Initialisation de SparkSession** : Nous démarrons une session Spark pour traiter les données.
- **Liste des fichiers** : Nous listons tous les fichiers dans le répertoire HDFS spécifié (/staging\_avocado).

- **Lecture et traitement des fichiers** : Pour chaque fichier CSV, nous lisons les données, convertissons la colonne 'date' (si nécessaire), et extrayons les colonnes 'jour' et 'mois'.
- **Sauvegarde des fichiers** : Les données nettoyées sont sauvegardées dans un nouveau fichier CSV avec un nom basé sur l'horodatage actuel, dans un répertoire de sortie (/refine\_avocado).
- **Suppression des fichiers originaux** : Les fichiers CSV originaux sont supprimés du répertoire de staging après traitement.

**Spark** permet un traitement distribué efficace des fichiers CSV, tandis que l'utilisation de HDFS garantit une gestion scalable et résiliente des fichiers. Les opérations de transformation et de nettoyage des données sont gérées directement au sein de Spark, optimisant ainsi le workflow de traitement des données.

## 2- Exécution du fichier dans **pyspark**

```
>>> exec(open("/home/hadoop/process_avocado.py").read())
2024-06-15 23:40:34,688 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Fichiers trouvés dans hdfs:///staging_avocado/ : ['hdfs:///127.0.0.1:9000/staging_avocado/avocado_20240612210119.csv/part-00000-05417bc6-c5e7-4a0a-92b9-a12b5270d753-c000.csv']
Lecture du fichier hdfs:///127.0.0.1:9000/staging_avocado/avocado_20240612210119.csv/part-00000-05417bc6-c5e7-4a0a-92b9-a12b5270d753-c000.csv
Sauvegarde du fichier nettoyé dans hdfs:///refine_avocado/avocado_nettoyé_202406152340.csv
Fichier hdfs:///127.0.0.1:9000/staging_avocado/avocado_20240612210119.csv/part-00000-05417bc6-c5e7-4a0a-92b9-a12b5270d753-c000.csv supprimé de hdfs:///staging_avocado/
>>>
```

## 3- Vérification

```
hadoop@afdelk: ~
hadoop@afdelk: ~
hadoop@afdelk: ~/apache-hive-3.1.3-bin/bin

hadoop@afdelk:~$ hadoop fs -ls hdfs:///staging_avocado/
Found 1 items
drwxr-xr-x - hadoop supergroup 0 2024-06-16 11:08 hdfs:///staging_avocado/avocado_20240612210119.csv
hadoop@afdelk:~$ hadoop fs -ls hdfs:///refine_avocado/avocado_cleaned_202406161108.csv
Found 2 items
-rw-r--r-- 1 hadoop supergroup 0 2024-06-16 11:08 hdfs:///refine_avocado/avocado_cleaned_202406161108.csv/_SUCCESS
-rw-r--r-- 1 hadoop supergroup 4883 2024-06-16 11:08 hdfs:///refine_avocado/avocado_cleaned_202406161108.csv/part-00000-98fa63f4-263e-42de-baf1-3e0a3cbe5d06-c000.csv
hadoop@afdelk:~$ hadoop fs -cat hdfs:///refine_avocado/avocado_cleaned_202406161108.csv/part-00000-98fa63f4-263e-42de-baf1-3e0a3cbe5d06-c000.csv | head -n 10
Date,AveragePrice,Volume,type,year,region,date_str,jour,mois
1420329600000,1.0,435022.0,conventional,2015,Atlanta,+46978-06-15,15,06
1420329600000,1.22,40873.3,conventional,2015,Atlanta,+46978-06-15,15,06
1420934400000,1.24,41195.1,conventional,2015,Albany,+46997-08-14,14,08
1420934400000,1.11,397543.0,conventional,2015,Atlanta,+46997-08-14,14,08
1421539200000,1.17,44511.3,conventional,2015,Albany,+47016-10-14,14,10
1421539200000,1.11,431491.0,conventional,2015,Atlanta,+47016-10-14,14,10
1422144000000,1.06,45147.5,conventional,2015,Albany,+47035-12-14,14,12
1422144000000,1.1,449333.0,conventional,2015,Atlanta,+47035-12-14,14,12
1422748800000,0.99,70873.6,conventional,2015,Albany,+47055-02-12,12,02
hadoop@afdelk:~$
```

# VII - HIVE partie 2

## 1- Créer une bd HIVE TP\_MASTER

```
hive> CREATE DATABASE IF NOT EXISTS TP_MASTER;
OK
Time taken: 0.058 seconds
hive> USE TP_MASTER;
OK
Time taken: 0.046 seconds
```

## 2- Créer une table externe **HIVE** qui pointe sur le répertoire HDFS /refine\_avocado.

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS refine_avocado (
>   'Date' BIGINT,
>   AveragePrice FLOAT,
>   Volume FLOAT,
>   type STRING,
>   year INT,
>   region STRING,
>   date_str STRING,
>   jour INT,
>   mois INT
> )
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> LOCATION '/refine_avocado';
OK
Time taken: 0.444 seconds
```

## 3- Créer une vue **HIVE** qui retourne la somme du champ 'Volume' et par mois, ce peu importe l'année

```
hive> CREATE VIEW IF NOT EXISTS volume_per_month AS
> SELECT mois, SUM(Volume) AS total_volume
> FROM refine_avocado
> GROUP BY mois;
OK
Time taken: 2.745 seconds
```

## 4- vérification et Explications

```

hive> SHOW TABLES LIKE 'volume_per_month';
OK
volume_per_month
Time taken: 0.184 seconds, Fetched: 1 row(s)
hive> SELECT * FROM volume_per_month;
Query ID = hadoop_20240617012925_efa62207-2672-468b-aa33-82bb379cffe1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1718587014226_0001, Tracking URL = http://afdelk:8088/proxy/application_1718587014226_0001/
Kill Command = /home/hadoop/hadoop/bin/mapred job -kill job_1718587014226_0001
Hadoop job information for Stage-1: number of mappers: 0; number of reducers: 1
2024-06-17 01:29:40,248 Stage-1 map = 0%, reduce = 0%
2024-06-17 01:29:47,669 Stage-1 map = 0%, reduce = 100%, Cumulative CPU 3.23 sec
MapReduce Total cumulative CPU time: 3 seconds 230 msec
Ended Job = job_1718587014226_0001
MapReduce Jobs Launched:
Stage-Stage-1: Reduce: 1   Cumulative CPU: 3.23 sec   HDFS Read: 7319 HDFS Write: 87 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 230 msec
OK
Time taken: 24.769 seconds

```

- **Query ID** : Identifiant unique de la requête lancée.
- **Total jobs** : Nombre total de jobs MapReduce lancés pour exécuter votre requête.
- **Launching Job** : Indique le démarrage du job MapReduce pour traiter la requête.
- **Hadoop job information** : Détails sur le job MapReduce, y compris le nombre de mappers et de reducers, ainsi que le suivi de l'URL où l'on peut surveiller son exécution.
- **MapReduce Jobs Launched** : Résume les informations sur les jobs MapReduce effectivement lancés pour traiter la requête.
- **Total MapReduce CPU Time Spent** : Temps total CPU cumulatif dépensé par les jobs MapReduce pour exécuter la requête.
- En fin de compte, le message **"OK"** indique que la requête a été exécutée avec succès et que l'on peut maintenant obtenir les résultats en fonction de la vue **volume\_per\_month**.

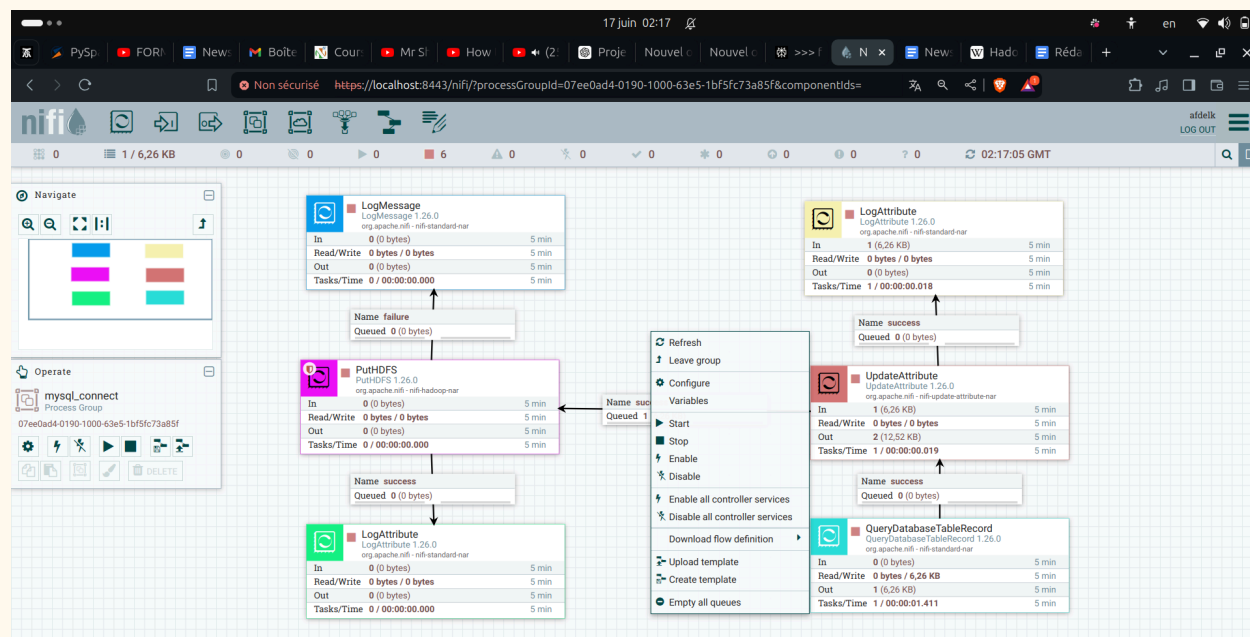
## VIII - Refaire :

Rajoutez les données du fichier 'avocado\_2.csv' dans la table MYSQL  
TP\_MASTER.'avocado'

Exécuter le process NIFI et les script PYSPARK, ensuite observez les résultats

Faire le même processus pour les fichiers 'avocado\_3.csv, avocado\_4.csv, avocado\_5.csv'

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/avocado_2.csv'
-> INTO TABLE avocado
-> FIELDS TERMINATED BY ','
-> LINES TERMINATED BY '\n'
-> IGNORE 1 LINES;
Query OK, 67 rows affected (0,04 sec)
Records: 67 Deleted: 0 Skipped: 0 Warnings: 0
```



```
hadoop@afdelk:~/nifi-1.26.0/bin$ cd
hadoop@afdelk:~$ hdfs dfs -ls /raw_avocado
Found 5 items
-rwxr-xr-x 1 hadoop supergroup 3591 2024-06-12 21:01 /raw_avocado/avocado_20240612210119.csv
-rw-r--r-- 1 hadoop supergroup 17982 2024-06-16 15:41 /raw_avocado/avocado_20240616154128.csv
-rw-r--r-- 1 hadoop supergroup 17982 2024-06-16 15:42 /raw_avocado/avocado_20240616154228.csv
-rw-r--r-- 1 hadoop supergroup 17982 2024-06-16 16:29 /raw_avocado/avocado_20240616162737.csv
-rw-r--r-- 1 hadoop supergroup 6408 2024-06-17 02:17 /raw_avocado/avocado_20240617021644.csv
hadoop@afdelk:~$
```

```
2024-06-17 02:20:07,502 WARN conf.HiveConf: HiveConf of name hive.llap.io.track.cache.usage does not exist
2024-06-17 02:20:07,502 WARN conf.HiveConf: HiveConf of name hive.use.orc.codec.pool does not exist
2024-06-17 02:20:07,502 WARN conf.HiveConf: HiveConf of name hive.query.results.cache.max.size does not exist
2024-06-17 02:20:07,505 WARN conf.HiveConf: HiveConf of name hive.repl.bootstrap.dump.open.txn.timeout does not exist
Le fichier avocado_20240612210119.csv a été traité avec succès.
Le fichier avocado_20240616154128.csv a été traité avec succès.
Le fichier avocado_20240616154228.csv a été traité avec succès.
Le fichier avocado_20240616162737.csv a été traité avec succès.
Le fichier avocado_20240617021644.csv a été traité avec succès.
>>>
```

```

hadoop@afdelk: ~
hadoop@afdelk: ~/apache-hive-3.1.3/bin/bin
2024-06-17 02:20:07,502 WARN conf.HiveConf: HiveConf of name hive.query.results.cache.max.size does not exist
2024-06-17 02:20:07,505 WARN conf.HiveConf: HiveConf of name hive.repl.bootstrap.dump.open.txn.timeout does not exist
Le fichier avocado_20240612210119.csv a été traité avec succès.
Le fichier avocado_20240616154128.csv a été traité avec succès.
Le fichier avocado_20240616154228.csv a été traité avec succès.
Le fichier avocado_20240616162737.csv a été traité avec succès.
Le fichier avocado_20240617021644.csv a été traité avec succès.
>>> exec(open("/home/hadoop/process_avocado.py").read())
2024-06-17 02:20:33,633 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Fichiers trouvés dans hdfs:///staggig_avocado/ : ['hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240612210119.csv/part-00000-d1ba91d1-af8e-4d29-a65e-942b180bb71-c000.csv', 'hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616154128.csv/part-00000-b6224035-9748-48da-a6ce-02fd523d591d-c000.csv', 'hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616154228.csv/part-00000-57d6f992-6e69-4623-abde-bcc0467d20e4-c000.csv', 'hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616162737.csv/part-00000-b8b10fc6-e466-41bb-8de3-4711e556b87a-c000.csv', 'hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240617021644.csv/part-00000-3bda7aec-8904-4258-83ba-97179b6b42e1-c000.csv']
Lecture du fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240612210119.csv/part-00000-d1ba91d1-af8e-4d29-a65e-942b180bb71-c000.csv
Sauvegarde du fichier nettoyé dans hdfs:///refine_avocado/avocado_cleaned_202406170220.csv
Fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240612210119.csv/part-00000-d1ba91d1-af8e-4d29-a65e-942b180bb71-c000.csv supprimé de hdfs:///staggig_avocado/
Lecture du fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616154128.csv/part-00000-b6224035-9748-48da-a6ce-02fd523d591d-c000.csv
Sauvegarde du fichier nettoyé dans hdfs:///refine_avocado/avocado_cleaned_202406170220.csv
Fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616154128.csv/part-00000-b6224035-9748-48da-a6ce-02fd523d591d-c000.csv supprimé de hdfs:///staggig_avocado/
Lecture du fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616154228.csv/part-00000-57d6f992-6e69-4623-abde-bcc0467d20e4-c000.csv
Sauvegarde du fichier nettoyé dans hdfs:///refine_avocado/avocado_cleaned_202406170220.csv
Fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240616162737.csv/part-00000-b8b10fc6-e466-41bb-8de3-4711e556b87a-c000.csv supprimé de hdfs:///staggig_avocado/
Lecture du fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240617021644.csv/part-00000-3bda7aec-8904-4258-83ba-97179b6b42e1-c000.csv
Sauvegarde du fichier nettoyé dans hdfs:///refine_avocado/avocado_cleaned_202406170220.csv
Fichier hdfs:///127.0.0.1:9000/staggig_avocado/avocado_20240617021644.csv/part-00000-3bda7aec-8904-4258-83ba-97179b6b42e1-c000.csv supprimé de hdfs:///staggig_avocado/
>>>

```

```

hadoop@afdelk: ~
hadoop@afdelk: ~
hadoop@afdelk:~$ hdfs dfs -ls /refine_avocado
Found 3 items
drwxr-xr-x - hadoop supergroup 0 2024-06-16 11:08 /refine_avocado/avocado_cleaned_202406161108.csv
drwxr-xr-x - hadoop supergroup 0 2024-06-17 01:38 /refine_avocado/avocado_cleaned_202406170138.csv
drwxr-xr-x - hadoop supergroup 0 2024-06-17 02:20 /refine_avocado/avocado_cleaned_202406170220.csv
hadoop@afdelk:~$

```

```

hive> select * from avocado_volume_tracking;
OK
avocado_20240616154128.csv      1.0236200049999996E8
avocado_20240616162737.csv      1.0236200049999996E8
avocado_20240616154128.csv      1.0236200049999996E8
avocado_20240612210119.csv      1.8028781400000002E7
avocado_20240612210119.csv      1.8028781400000002E7
avocado_20240612210119.csv      1.8028781400000002E7
avocado_20240616154228.csv      1.0236200049999996E8
avocado_20240617021644.csv      3.181443810000001E7
avocado_20240616154228.csv      1.0236200049999996E8
avocado_20240612210119.csv      1.8028781400000002E7
avocado_20240612210119.csv      1.8028781400000002E7
avocado_20240616162737.csv      1.0236200049999996E8
Time taken: 0.246 seconds, Fetched: 12 row(s)
hive>

```

```

hadoop@afdelk:~$ hadoop fs -cat /refine_avocado/avocado_cleaned_202406161108.csv/part-00000-98fa63f4-263e-42de-baf1-3e0a3cbe5d06-c000.csv | head -n 50
Date,AveragePrice,Volume,type,year,region,date_str,jour,mois
1420329600000,1.0,435022.0,conventional,2015,Atlanta,+46978-06-15,15,06
1420329600000,1.22,40873.3,conventional,2015,Albany,+46978-06-15,15,06
1420934400000,1.24,41195.1,conventional,2015,Albany,+46997-08-14,14,08
1420934400000,1.11,397543.0,conventional,2015,Atlanta,+46997-08-14,14,08
1421539200000,1.17,44511.3,conventional,2015,Albany,+47016-10-14,14,10
1421539200000,1.11,431491.0,conventional,2015,Atlanta,+47016-10-14,14,10
1422144000000,1.06,45147.5,conventional,2015,Albany,+47035-12-14,14,12
1422144000000,1.1,449333.0,conventional,2015,Atlanta,+47035-12-14,14,12
1422748800000,0.99,70873.6,conventional,2015,Albany,+47055-02-12,12,02
1422748800000,0.96,636771.0,conventional,2015,Atlanta,+47055-02-12,12,02
1423353600000,0.99,51254.0,conventional,2015,Albany,+47074-04-13,13,04
1423353600000,1.03,433884.0,conventional,2015,Atlanta,+47074-04-13,13,04
1423958400000,1.06,427391.0,conventional,2015,Atlanta,+47093-06-12,12,06
1423958400000,1.06,41567.6,conventional,2015,Albany,+47093-06-12,12,06
1424563200000,1.1,431309.0,conventional,2015,Atlanta,+47112-08-12,12,08
1424563200000,1.07,45675.1,conventional,2015,Albany,+47112-08-12,12,08
1425168000000,0.99,512532.0,conventional,2015,Atlanta,+47131-10-12,12,10
1425168000000,0.99,55595.7,conventional,2015,Albany,+47131-10-12,12,10
1425772800000,1.07,40507.4,conventional,2015,Albany,+47150-12-11,11,12
1425772800000,1.1,420826.0,conventional,2015,Atlanta,+47150-12-11,11,12
1426377600000,1.11,43045.8,conventional,2015,Albany,+47170-02-09,09,02
1426377600000,1.12,399566.0,conventional,2015,Atlanta,+47170-02-09,09,02
1426982400000,1.0,479591.0,conventional,2015,Atlanta,+47189-04-10,10,04
1426982400000,1.12,46346.9,conventional,2015,Albany,+47189-04-10,10,04
1427587200000,1.11,365722.0,conventional,2015,Atlanta,+47208-06-09,09,06
1427587200000,1.02,67799.1,conventional,2015,Albany,+47208-06-09,09,06
1428192000000,1.16,47362.1,conventional,2015,Albany,+47227-08-09,09,08
1428192000000,1.04,383140.0,conventional,2015,Atlanta,+47227-08-09,09,08
1428796800000,0.99,451102.0,conventional,2015,Atlanta,+47246-10-08,08,10
1428796800000,1.13,48364.3,conventional,2015,Albany,+47246-10-08,08,10

```

```

hive> SELECT * FROM volume_per_month;
Query ID = hadoop_20240617022208_15e57db5-dac1-4ade-a59e-d7271b15f62f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1718587014226_0008, Tracking URL = http://afdelk:8088/proxy/application_1718587014226_0008/
Kill Command = /home/hadoop/hadoop/bin/mapred job -kill job_1718587014226_0008
Hadoop job information for Stage-1: number of mappers: 0; number of reducers: 1
2024-06-17 02:22:20,886 Stage-1 map = 0%, reduce = 0%
2024-06-17 02:22:30,379 Stage-1 map = 0%, reduce = 100%, Cumulative CPU 4.07 sec
MapReduce Total cumulative CPU time: 4 seconds 70 msec
Ended Job = job_1718587014226_0008
MapReduce Jobs Launched:
Stage-Stage-1: Reduce: 1 Cumulative CPU: 4.07 sec HDFS Read: 7319 HDFS Write: 87 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 70 msec
OK
Time taken: 23.778 seconds

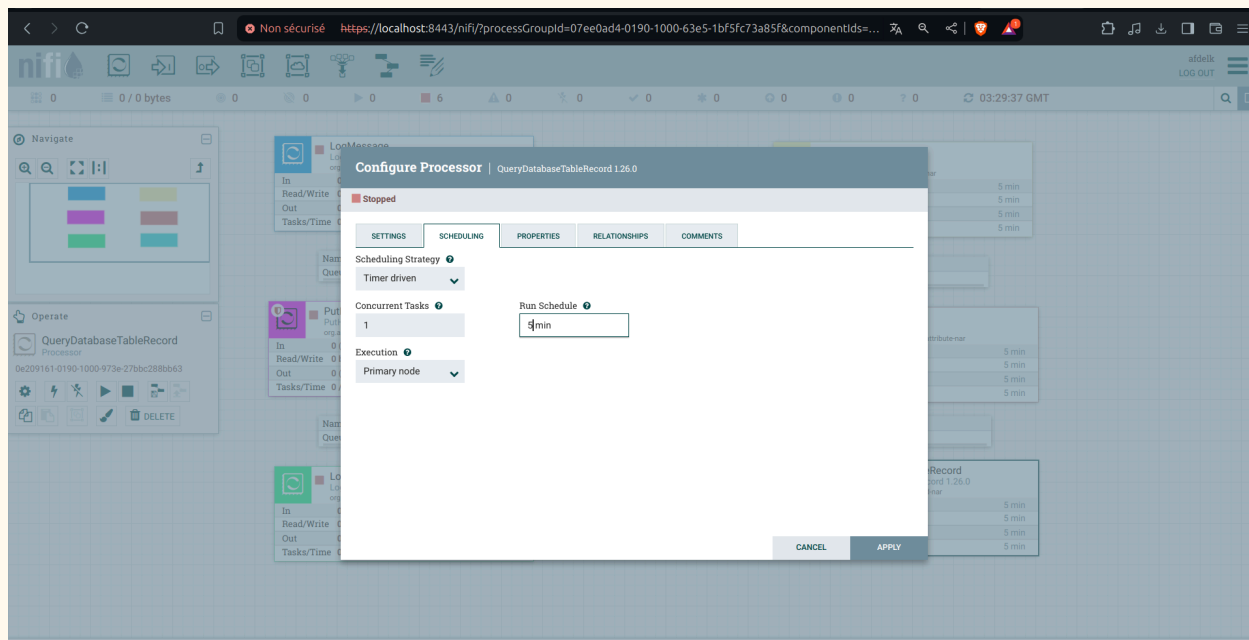
```

## IX- Cron :

### 1- Scheduler le process **NIFI** pour qu'il tourne toute les 5 mins

Nous avons configuré notre processus NiFi pour qu'il s'exécute automatiquement toutes les 5 minutes, assurant un flux continu de traitement des données en temps réel.





## 2- Cronner les **jobs PYSPARK** pour qu'il s'exécute tous les 10 et 7 mins

```
hadoop@afdelk:~$ crontab -e
no crontab for hadoop - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <----- easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny
 4. /bin/ed

Choose 1-4 [1]: 1
crontab: installing new crontab
```

Pour automatiser l'exécution de nos scripts PySpark à des intervalles réguliers afin de garantir un flux continu de traitement des données sans intervention manuelle, nous avons utilisé **crontab**. Plus précisément, nous avons configuré crontab avec l'éditeur **nano** pour exécuter les **jobs PySpark** toutes les **10 et 7 minutes**, ce qui permet une ingestion, un traitement et un transfert des données rapides et réguliers.



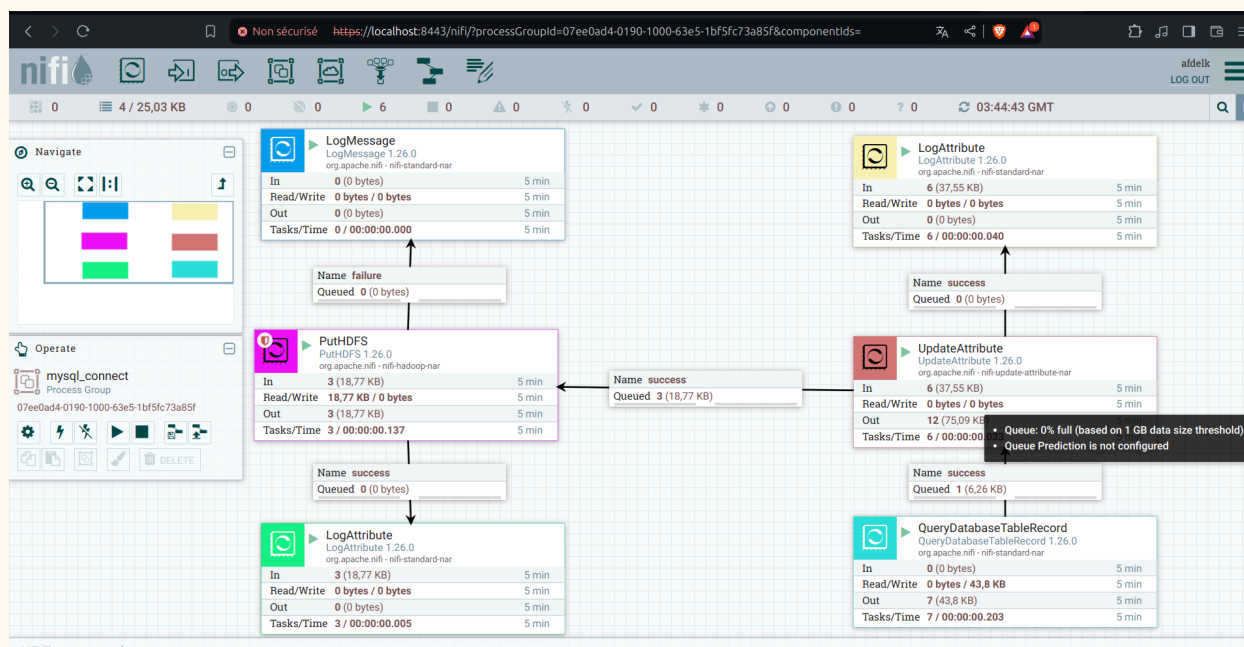
```

GNU nano 7.2 /tmp/crontab.QwL6Rm/crontab *
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
# cronne du script pysspark1
*/7 * * * * /home/hadoop/recup_fichier data.py
# cronne du script pyspark2
*/10 * * * * /home/hadoop/process_avocado.py

```

^C Aide    ^O Écrire    ^M Chercher    ^K Couper    ^T Exécuter    ^C Enplacement    M-U Annuler    M-A Marquer    M-T -> Crochet  
 ^X Quitter    ^R Lire fich.    ^W Remplacer    ^U Coller    ^D Justifier    ^Y Aller ligne    M-E Refaire    M-C Copier    ^Q Retrouver

### 3- Faire tourner tout le **WORKFLOW** (NIFI et SPARKs)



En parallèle, nous avons configuré l'envoi de mails pour recevoir des notifications sur l'état de nos scripts PySpark. Si un script échoue ou rencontre une erreur, nous recevons un email avec les détails de l'erreur. Cela nous permet de réagir rapidement pour résoudre les problèmes et minimiser les interruptions. L'utilisation combinée de crontab pour l'automatisation et des mails pour la surveillance et les alertes assure que notre pipeline de traitement de données reste robuste, fiable, et maintenable.

```
hadoop@afdelk:~$ mail -H
>N 1 Cron Daemon      lun. juin 17 04: 24/849  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 2 Cron Daemon      lun. juin 17 04: 19/678  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 3 Cron Daemon      lun. juin 17 04: 19/678  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 4 Cron Daemon      lun. juin 17 04: 23/805  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 5 Cron Daemon      lun. juin 17 04: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 6 Cron Daemon      lun. juin 17 04: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 7 Cron Daemon      lun. juin 17 04: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 8 Cron Daemon      lun. juin 17 04: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 9 Cron Daemon      lun. juin 17 04: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 10 Cron Daemon     lun. juin 17 04: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 11 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 12 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 13 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 14 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 15 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 16 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 17 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 18 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 19 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 20 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 21 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
N 22 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 23 Cron Daemon     lun. juin 17 05: 18/669  Cron <hadoop@afdelk> /home/hadoop/recup_fichier_data.py
N 24 Cron Daemon     lun. juin 17 05: 22/796  Cron <hadoop@afdelk> /home/hadoop/process_avocado.py
Vous avez du courrier dans /var/mail/hadoop
hadoop@afdelk:~$
```

Execution effective des cron sur les fichiers recup\_fichier.py = pyspark 1 et  
process\_avocado.py = pyspark2

NOS REMERCIEMENTS :-)