Rapport de stage

Data engineering

Nom : BARRY, Mouhamadou Adji

Entreprise : Atos Sénégal

Université : Dakar Institute of Technology (DIT)

Durée : 2 mois (prolongation de 2 mois à venir)

Remerciements

Cheikh Ahmet Tidiane FALL
Ibrahima Oumar LY
Mouhamet Macquillou Alpha DIENE
Fatoumata DIALLO

Résumé

1. **Apache Spark**

   Un framework open source pour le traitement de données distribuées.

2. **Cluster Big Data**

   Un ensemble de machines interconnectées pour stocker et traiter de grandes quantités de données.

3. **Pipeline de Traitement Batch**

   Un processus automatisé pour traiter des données en lots.

4. **Data Engineering**

   La discipline qui consiste à concevoir, construire et maintenir des systèmes de données.

Table des Matières

Introduction

Leader Mondial

Atos est un leader mondial de la transformation digitale.

Présence Internationale

L'entreprise est présente dans 71 pays.

Solutions Intégrées

Atos propose des solutions intégrées pour tous les secteurs.

Implantation en Afrique

Atos compte plus de 1000 collaborateurs et collaboratrices en Afrique, avec un siège à Dakar, Sénégal.

Produits et Services

Atos OneCloud

Une plateforme cloud pour les entreprises.

Cybersécurité

Des solutions pour protéger les données et les systèmes.

Intelligence Artificielle

Des solutions pour automatiser les processus et améliorer l'efficacité.

Services de Décarbonation

Des solutions pour réduire l'empreinte carbone des entreprises.

Réalisations Notables

Atos a reçu plusieurs prix et distinctions, notamment le prix Elevate de Juniper Networks, le label « France Sécurité » pour Evidian IDaaS, et la sécurisation des Jeux Olympiques et Paralympiques de Paris 2024.

Prix	Description
Elevate de Juniper Networks	Reconnaissance pour l'innovation en matière de réseaux.
Label « France Sécurité »	Certification pour la sécurité des solutions d'identité et d'accès.
Sécurisation des Jeux Olympiques	Responsabilité pour la sécurité des Jeux Olympiques et Paralympiques de Paris 2024.

Objectifs du Stage

1 Objectifs Généraux

Acquérir une expérience pratique en Data Engineering et développer des compétences techniques dans le domaine du Big Data.

2 Objectifs Spécifiques

Apprendre à utiliser Apache Spark, installer et configurer un cluster Big Data, et développer une pipeline de traitement batch.

Description du Poste et des Missions

Intern en Data Engineering

Mon rôle était de contribuer à la mise en place d'une infrastructure Big Data et au développement d'une pipeline de traitement de données.

Présentation d'Apache Spark

J'ai appris à utiliser Apache Spark, un framework open source pour le traitement distribué de données à grande échelle.

Installation et Configuration

J'ai participé à l'installation et à la configuration d'un cluster Big Data, en utilisant des technologies telles que Hadoop et YARN.

Méthodologie

1

Méthodes de travail

Documentation et Recherche
Prototypage
Collaboration
Feedback Régulier

2

Outils et technologies

Apache (Spark, Hadoop, Hive e Airflow) Python, Postgresql, Git, et Teams.

3

Processus de travail

J'ai suivi un processus de travail rigoureux, en respectant les normes de qualité et les délais.

Difficultés Rencontrées

1. Erreur lors de la construction d'images Docker
2. Importation des JARs pour un cluster YARN multi-nodes
3. Limité par Zscaler
4. Espace de Disque Trop Petit
5. Intégration d'Airflow pour le Workflow

Compétences Acquises

Techniques

J'ai maîtrisé les outils et technologies Big Data tels que Hadoop, Spark, Hive, et Docker.

Analyse

J'ai acquis la capacité d'analyser des problèmes complexes, d'identifier les causes profondes et de proposer des solutions efficaces

Collaboration

J'ai développé des compétences en communication et en travail d'équipe, en travaillant efficacement avec des collègues de différents niveaux d'expérience

Résultats

Cluster Big Data Fonctionnel	J'ai réussi à mettre en place un cluster Big Data intégrant Hadoop, Spark, Hive et Jupyter sur Docker.
Pipeline de Traitement	J'ai développé et déployé une pipeline de traitement batch utilisant Apache Spark et Hive.
Projets de Test Réussis	J'ai réalisé plusieurs projets de test démontrant l'efficacité et la robustesse des configurations et pipelines développés.

Conclusion

1

Expérience enrichissante

Ce stage a été une expérience enrichissante, me permettant d'acquérir de nouvelles compétences et de mettre en pratique mes connaissances.

2

Compétences développées

J'ai développé mes compétences techniques et analytiques, me permettant de mieux comprendre les concepts et les outils du Big Data.

Perspectives et Recommandations

Perspectives professionnelles

Ce stage m'a permis de me familiariser avec les technologies et les pratiques du Big Data, me préparant ainsi à une carrière dans ce domaine.

Recommandations pour les futurs stagiaires

Je recommande aux futurs stagiaires de se familiariser avec les outils et les technologies du Big Data avant de commencer leur stage.

Annexes et Bibliographie

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Pré-requis

Serveurs : 1 master, 2 workers

Système d'exploitation : Ubuntu 20.04

Java : OpenJDK 8

Hadoop : 3.2.1

Spark : 3.2.0

Hive : 2.3.2

Docker et Docker Compose

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Installation et Configuration de Hadoop

```yaml
# Hadoop Services
namenode:
  image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
  container_name: cov19-namenode
  restart: always
  ports:
    - "9870:9870"
    - "9001:9000"
  volumes:
    - hadoop_namenode:/hadoop/dfs/name
    - ./scripts/hadoop:/opt/scripts
    - ./data:/opt/data
    - spark-jars:/opt/spark/jars:ro
    - hadoop-config:/opt/hadoop-3.2.1/etc/hadoop/
    - hadoop-jars:/opt/hadoop-3.2.1/share/hadoop/
    - hadoop-lib:/opt/hadoop-3.2.1/lib/
  environment:
    - CLUSTER_NAME=test
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  env_file:
    - ./docker-compose.env
  healthcheck:
    test: ["CMD", "curl", "-f", "http://namenode:9870"]
    interval: 30s
    timeout: 10s
    retries: 5
```

```yaml
datanode:
  image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
  container_name: cov19-datanode
  restart: always
  ports:
    - "9864:9864"
  volumes:
    - hadoop_datanode:/hadoop/dfs/data
    - spark-jars:/opt/spark/jars:ro
  environment:
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  depends_on:
    namenode:
      condition: service_healthy
  env_file:
    - ./docker-compose.env
```

```yaml
historyserver:
  image: bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8
  container_name: cov19-historyserver
  restart: always
  ports:
    - "8188:8188"
  volumes:
    - hadoop_historyserver:/hadoop/yarn/timeline
    - spark-jars:/opt/spark/jars:ro
  environment:
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  depends_on:
    - namenode
    - datanode
    - resourcemanager
  env_file:
    - ./docker-compose.env
```

```yaml
resourcemanager:
  image: bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8
  container_name: cov19-resourcemanager
  restart: always
  ports:
    - "8088:8088"
  environment:
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  depends_on:
    - namenode
    - datanode
  env_file:
    - ./docker-compose.env
  volumes:
    - spark-jars:/opt/spark/jars:ro

nodemanager:
  image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
  container_name: cov19-nodemanager
  restart: always
  ports:
    - "8042:8042"
  environment:
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  depends_on:
    - namenode
    - datanode
    - resourcemanager
  env_file:
    - ./docker-compose.env
  volumes:
    - spark-jars:/opt/spark/jars:ro
```

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Installation et Configuration de Hadoop

docker-compose.env

```
# Environment variables for Hadoop and Hive
CLUSTER_NAME=test
CORE_CONF_fs_defaultFS=hdfs://namenode:9000
HIVE_SITE_CONF_javax_jdo_option_ConnectionURL=jdbc:postgresql://hive-metastore-postgresql/metastore
HIVE_SITE_CONF_javax_jdo_option_ConnectionDriverName=org.postgresql.Driver
HIVE_SITE_CONF_javax_jdo_option_ConnectionUserName=hive
HIVE_SITE_CONF_javax_jdo_option_ConnectionPassword=hive
HIVE_SITE_CONF_datanucleus_autoCreateSchema=false
HIVE_SITE_CONF_hive_metastore_uris=thrift://hive-metastore:9083
HDFS_CONF_dfs_namenode_datanode_registration_ip___hostname___check=false

# Additional Hadoop configuration
CORE_CONF_hadoop_http_staticuser_user=root
CORE_CONF_hadoop_proxyuser_hue_hosts=*
CORE_CONF_hadoop_proxyuser_hue_groups=*
CORE_CONF_io_compression_codecs=org.apache.hadoop.io.compress.SnappyCodec

# HDFS configuration
HDFS_CONF_dfs_webhdfs_enabled=true
HDFS_CONF_dfs_permissions_enabled=false
HDFS_CONF_dfs_namenode_datanode_registration_ip___hostname___check=false

# YARN configuration
YARN_CONF_yarn_log___aggregation___enable=true
YARN_CONF_yarn_log_server_url=http://historyserver:8188/applicationhistory/logs/
YARN_CONF_yarn_resourcemanager_recovery_enabled=true
YARN_CONF_yarn_resourcemanager_store_class=org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore
YARN_CONF_yarn_resourcemanager_scheduler_class=org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler
YARN_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___mb=8192
YARN_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___vcores=4
YARN_CONF_yarn_resourcemanager_fs_state___store_uri=/rmstate
YARN_CONF_yarn_resourcemanager_system___metrics___publisher_enabled=true
YARN_CONF_yarn_resourcemanager_hostname=resourcemanager
YARN_CONF_yarn_resourcemanager_address=resourcemanager:8032
YARN_CONF_yarn_resourcemanager_scheduler_address=resourcemanager:8030
YARN_CONF_yarn_resourcemanager_resource__tracker_address=resourcemanager:8031
YARN_CONF_yarn_timeline___service_enabled=false
YARN_CONF_yarn_timeline___service_generic___application___history_enabled=true
YARN_CONF_yarn_timeline___service_hostname=historyserver
YARN_CONF_mapreduce_map_output_compress=true
YARN_CONF_mapred_map_output_compress_codec=org.apache.hadoop.io.compress.SnappyCodec
YARN_CONF_yarn_nodemanager_resource_memory___mb=16384
YARN_CONF_yarn_nodemanager_resource_cpu___vcores=8
YARN_CONF_yarn_nodemanager_disk___health___checker_max___disk___utilization___per___disk___percentage=98.5
YARN_CONF_yarn_nodemanager_remote___app___log___dir=/app-logs
YARN_CONF_yarn_nodemanager_aux___services=mapreduce_shuffle

# MapReduce configuration
MAPRED_CONF_mapreduce_framework_name=yarn
MAPRED_CONF_mapred_child_java_opts=-Xmx4096m
MAPRED_CONF_mapreduce_map_memory_mb=4096
MAPRED_CONF_mapreduce_reduce_memory_mb=8192
MAPRED_CONF_mapreduce_map_java_opts=-Xmx3072m
MAPRED_CONF_mapreduce_reduce_java_opts=-Xmx6144m
MAPRED_CONF_yarn_app_mapreduce_am_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1/
MAPRED_CONF_mapreduce_map_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1/
MAPRED_CONF_mapreduce_reduce_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1/
```

```
volumes:
  hadoop_namenode:
  hadoop_datanode:
  hadoop_historyserver:
  hadoop-config:
  hadoop-jars:
  hadoop-lib:
  spark-jars:
  postgres-db-volume:
```

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Installation et Configuration de Spark

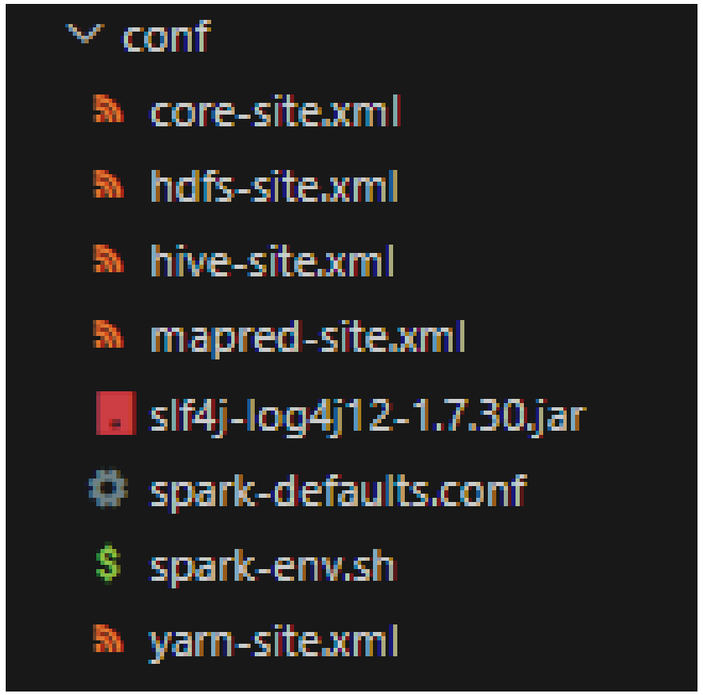
```yaml
spark-master:
  image: bde2020/spark-master:3.2.0-hadoop3.2
  container_name: cov19-spark-master
  ports:
    - "8080:8080"
    - "7077:7077"
  environment:
    - INIT_DAEMON_STEP=setup_spark
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  volumes:
    - ./scripts/spark:/opt/scripts
    - ./conf:/spark/conf
    - hadoop-config:/opt/hadoop-config:ro
    - hadoop-jars:/opt/hadoop-jars:ro
    - spark-jars:/spark/jars
    - hadoop-lib:/opt/hadoop-lib/:ro
  env_file:
    - ./docker-compose.env
  depends_on:
    - namenode

spark-worker-1:
  image: bde2020/spark-worker:3.2.0-hadoop3.2
  container_name: cov19-spark-worker-1
  depends_on:
    - spark-master
  ports:
    - "8081:8081"
  environment:
    # - SPARK_MASTER=spark://spark-master:7077
    - SPARK_MASTER=spark-master:7077
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  volumes:
    - ./conf:/spark/conf
    - hadoop-config:/opt/hadoop-config:ro
    - hadoop-jars:/opt/hadoop-jars:ro
    - hadoop-lib:/opt/hadoop-lib/:ro
  env_file:
    - ./docker-compose.env
```

```yaml
spark-worker-1:
  image: bde2020/spark-worker:3.2.0-hadoop3.2
  container_name: cov19-spark-worker-1
  depends_on:
    - spark-master
  ports:
    - "8081:8081"
  environment:
    # - SPARK_MASTER=spark://spark-master:7077
    - SPARK_MASTER=spark-master:7077
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  volumes:
    - ./conf:/spark/conf
    - hadoop-config:/opt/hadoop-config:ro
    - hadoop-jars:/opt/hadoop-jars:ro
    - hadoop-lib:/opt/hadoop-lib/:ro
  env_file:
    - ./docker-compose.env

spark-worker-2:
  image: bde2020/spark-worker:3.2.0-hadoop3.2
  container_name: cov19-spark-worker-2
  depends_on:
    - spark-master
  ports:
    - "8083:8081"
  environment:
    - SPARK_MASTER=spark://spark-master:7077
    - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
  volumes:
    - ./conf:/spark/conf
    #- ./conf/spark/hadoop-conf:/etc/hadoop
    - ./conf/spark/hadoop-lib:/usr/local/hadoop/lib
    - hadoop-config:/etc/hadoop
  env_file:
#     - ./docker-compose.env

# Hive Services
hive-server:
  image: bde2020/hive:2.3.2-postgresql-metastore
```

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Installation et Configuration de Spark



spark-defaults.conf

```
spark.master=yarn
spark.eventLog.enabled=true
spark.eventLog.dir=hdfs://namenode:9000/user/spark/spark-logs
spark.serializer=org.apache.spark.serializer.KryoSerializer
spark.driver.memory=8g
spark.executor.memory=5g
spark.executor.cores=2
spark.memory.fraction=0.6
spark.driver.extraJavaOptions=-Dderby.system.home=/tmp/derby/
spark.sql.repl.eagerEval.enabled=true
spark.sql.shuffle.partitions=200
spark.history.fs.logDirectory=hdfs://namenode:9000/user/spark/spark-logs
spark.history.ui.port=18080
spark.yarn.jars=hdfs://namenode:9000/user/spark/jars/*
spark.yarn.archive=hdfs://namenode:9000/user/spark/jars
spark.network.timeout=300s
spark.executor.heartbeatInterval=60s
spark.shuffle.compress=true
spark.shuffle.spill.compress=true
spark.executor.extraJavaOptions=-XX:+UseG1GC -XX:InitiatingHeapOccupancyPercent=35 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:/var/log/spark/gc-%t-%p.log
spark.yarn.am.memory=2g
spark.yarn.am.cores=1

# Dynamic allocation settings
spark.dynamicAllocation.enabled=true
spark.shuffle.service.enabled=true
spark.dynamicAllocation.minExecutors=2
spark.dynamicAllocation.maxExecutors=10

# Use the correct memory overhead key
spark.executor.memoryOverhead=1024

# Ensure spark.executor.instances is aligned with minExecutors
spark.executor.instances=2
```

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Installation et Configuration de Hive

```yaml
# Hive Services
hive-server:
  image: bde2020/hive:2.3.2-postgresql-metastore
  container_name: cov19-hive-server
  depends_on:
    - namenode
    - datanode
  env_file:
    - ./docker-compose.env
  environment:
    HIVE_CORE_CONF_javax_jdo_option_ConnectionURL: "jdbc:postgresql://hive-metastore/metastore"
    SERVICE_PRECONDITION: "hive-metastore:9083"
  ports:
    - "10000:10000"

hive-metastore:
  image: bde2020/hive:2.3.2-postgresql-metastore
  container_name: cov19-hive-metastore
  env_file:
    - ./docker-compose.env
  command: /opt/hive/bin/hive --service metastore
  environment:
    SERVICE_PRECONDITION: "namenode:9870 namenode:9000 datanode:9864 hive-metastore-postgresql:5432"
  ports:
    - "9083:9083"

hive-metastore-postgresql:
  image: bde2020/hive-metastore-postgresql:2.3.0
  container_name: cov19-hive-metastore-postgresql
```

Annexe 1 : Documentation technique sur la configuration du cluster Big Data

Configuration du Cluster Big Data

Docker

```
barryma@DESKTOP-LBFN3RA:~$ docker ps --format "table {{.Names}}\t{{.Status}}"
NAMES                                       STATUS
cov19-historyserver                         Up 4 minutes (healthy)
cov19-nodemanager                           Up 4 minutes (healthy)
project_covid19_final-airflow-triggerer-1   Up 4 minutes (healthy)
project_covid19_final-airflow-worker-1      Up 4 minutes (healthy)
project_covid19_final-airflow-scheduler-1   Up 4 minutes (unhealthy)
project_covid19_final-airflow-webserver-1   Up 4 minutes (healthy)
cov19-resourcemanager                       Up 4 minutes (healthy)
cov19-hive-server                           Up 4 minutes
cov19-spark-worker-1                        Up 5 minutes
cov19-datanode                              Up 4 minutes (healthy)
cov19-spark-master                          Up 5 minutes
project_covid19_final-dev-1                 Up 5 minutes
spark-notebook                              Up 5 minutes (healthy)
project_covid19_final-redis-1               Up 5 minutes (healthy)
project_covid19_final-postgres-1            Up 5 minutes (healthy)
cov19-hive-metastore-postgresql             Up 5 minutes
cov19-hive-metastore                        Up 5 minutes
cov19-namenode                              Up 5 minutes (healthy)
project_covid19_final-docker-proxy-1        Up 5 minutes
```

Annexe 2 : Exemples de codes et scripts utilisés pour les projets de test

Exemple de Script PySpark

dev > src > processors > data_cleaning.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date
from pyspark.sql.types import IntegerType
import logging
import re

# Setup logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class DataCleaner:
    def __init__(self, hdfs_url, raw_path, processed_path):
        self.hdfs_url = hdfs_url
        self.raw_path = raw_path
        self.processed_path = processed_path

    def is_date_column(self, column_name):
        pattern = re.compile(r'\d{1,2}/\d{1,2}/\d{2}')
        return bool(pattern.fullmatch(column_name))

    def transform_date_columns(self, df):
        for column in df.columns:
            if self.is_date_column(column):
                df = df.withColumnRenamed(column, f"date_{column.replace('/', '_')}")
        return df

    def clean_and_transform(self, df, date_column=None, int_columns=None):
        df_cleaned = df.dropna()
        if date_column:
            df_cleaned = df_cleaned.withColumn(date_column, to_date(col(date_column), "yyyy-MM-dd"))
        if int_columns:
            for col_name in int_columns:
                df_cleaned = df_cleaned.withColumn(col_name, col(col_name).cast(IntegerType()))
        return df_cleaned
```

dev > src > processors > data_cleaning.py > DataCleaner > process_data

```python
class DataCleaner:

    def reshape_dataframe(self, df, fixed_columns, date_columns):
        stack_expr = "stack({0}, {1}) as (Date, Value)".format(
            len(date_columns),
            ', '.join([f"'{col}' as `{col}`" for col in date_columns])
        )
        df_reshaped = df.selectExpr(*fixed_columns, stack_expr)
        df_reshaped = self.transform_date_columns(df_reshaped)  # Rename columns in the reshaped dataframe
        return df_reshaped

    def process_data(self, spark, dataset_name, date_column=None, reshape=False, int_columns=None):
        try:
            logger.info(f"Processing dataset: {dataset_name}")
            df = spark.read.csv(f"{self.hdfs_url}{self.raw_path}/{dataset_name}", header=True, inferSchema=True)
            df_cleaned = self.clean_and_transform(df, date_column, int_columns)

            if reshape:
                fixed_columns = [col for col in df_cleaned.columns if not self.is_date_column(col)]
                date_columns = [col for col in df_cleaned.columns if self.is_date_column(col)]
                df_cleaned = self.reshape_dataframe(df_cleaned, fixed_columns, date_columns)

            # Count and log the number of date columns
            date_columns_count = sum(1 for col in df_cleaned.columns if 'date_' in col)
            logger.info(f"{dataset_name} has {date_columns_count} date columns after processing.")

            output_path = f"{self.hdfs_url}{self.processed_path}/{dataset_name.replace('.csv', '_cleaned.parquet')}"
            df_cleaned.write.mode("overwrite").parquet(output_path)
            logger.info(f"{dataset_name} cleaned and saved to {output_path}")
        except Exception as e:
            logger.error(f"Error cleaning {dataset_name}: {e}")
```

```python
def run_all(self):
    spark = SparkSession.builder \
        .appName("CovidDataCleaning") \
        .config("spark.hadoop.fs.defaultFS", self.hdfs_url) \
        .getOrCreate()

    self.process_data(spark, 'owid/full_data.csv', date_column='date', int_columns=['new_cases', 'new_deaths', 'total_cases', 'total_deaths'])
    #self.process_data(spark, 'google_cloud/main.csv', date_column='date', int_columns=['new_cases', 'new_deaths', 'total_cases', 'total_deaths'])
    #self.process_data(spark, 'csse/time_series_covid19_confirmed_global.csv', reshape=True, int_columns=['cases'])
    #self.process_data(spark, 'csse/time_series_covid19_deaths_global.csv', reshape=True, int_columns=['deaths'])
    self.process_data(spark, 'who/WHO-COVID-19-global-data.csv', date_column='Date_reported', int_columns=['new_cases', 'new_deaths', 'cumulative_cases', 'cumulative_deaths'])

    spark.stop()
```

Annexe 2 : Exemples de codes et scripts utilisés pour les projets de test

Exemple de Script HiveQL

dev > src > analyzers > hive_queries.py > ...

```python
from pyhive import hive
import logging

# Setup logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class HiveQueries:
    def __init__(self, host, hdfs_path_proc):
        self.conn = hive.Connection(host=host)
        self.hdfs_path_proc = hdfs_path_proc

    def validate_schema(self, schema):
        """
        Validate schema format. Basic validation for simplicity.
        """
        required_types = ['STRING', 'INT', 'DOUBLE', 'DATE']
        for field in schema.split(','):
            parts = field.strip().split()
            if len(parts) != 2 or parts[1] not in required_types:
                raise ValueError(f"Invalid schema field: {field}")
        logger.info("Schema validated successfully.")

    def create_table(self, table_name, schema, hdfs_path):
        try:
            self.validate_schema(schema)
            query = f"""
            CREATE EXTERNAL TABLE IF NOT EXISTS {table_name} (
                {schema}
            )
            STORED AS PARQUET
            LOCATION '{hdfs_path}'
            """
            with self.conn.cursor() as cursor:
                cursor.execute(query)
                logger.info(f"Created table {table_name}.")
        except Exception as e:
            logger.error(f"Error creating table {table_name}: {e}")
```

src > analyzers > hive_queries.py > HiveQueries > create_table_with_dynamic_schema

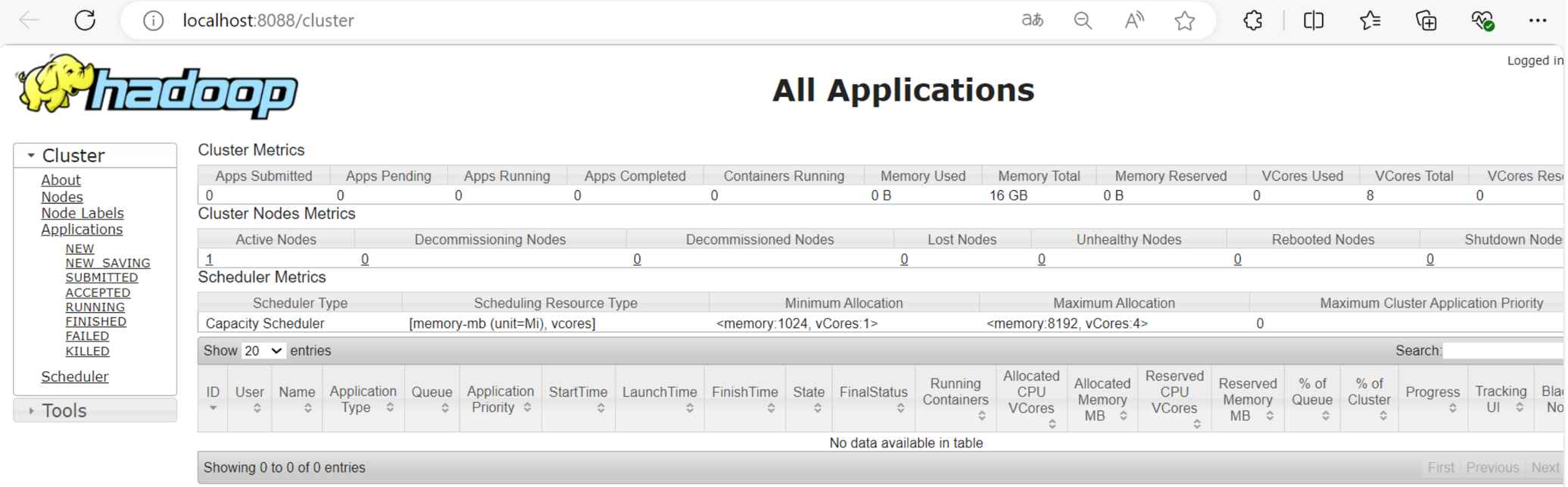
```python
class HiveQueries:

    def create_table_with_dynamic_schema(self, filename, schema_key):
        schemas = {
            'owid': """
            date_ DATE,
            location STRING,
            new_cases INT,
            new_deaths INT,
            total_cases INT,
            total_deaths INT
            """,
            'google_cloud': """
            date_ DATE,
            location STRING,
            new_cases INT,
            new_deaths INT,
            total_cases INT,
            total_deaths INT
            """,
            'csse_confirmed': """
            country STRING,
            state STRING,
            lat DOUBLE,
            long DOUBLE,
            date_ DATE,
            cases INT
            """,
            'csse_deaths': """
            country STRING,
            state STRING,
            lat DOUBLE,
            long DOUBLE,
            date_ DATE,
            deaths INT
            """,
            'who': """
            date_ DATE,
            country STRING,
            new_cases INT,
            new_deaths INT,
            cumulative_cases INT,
            cumulative_deaths INT
            """
```

src > analyzers > hive_queries.py > HiveQueries > run_all

```python
class HiveQueries:
    def create_table_with_dynamic_schema(self, filename, schema_key):
            ,
            'who': """
            date_ DATE,
            country STRING,
            new_cases INT,
            new_deaths INT,
            cumulative_cases INT,
            cumulative_deaths INT
            """
        }
        schema = schemas.get(schema_key, "")
        if schema:
            hdfs_path = f'{self.hdfs_path_proc}/{schema_key}/{filename}_cleaned.parquet'
            self.create_table(f'{schema_key}_data', schema, hdfs_path)
            self.load_data_into_table(f'{schema_key}_data', hdfs_path)
        else:
            logger.error(f"Schema key '{schema_key}' not found.")

    def load_data_into_table(self, table_name, hdfs_path):
        try:
            query = f"LOAD DATA INPATH '{hdfs_path}' INTO TABLE {table_name}"
            with self.conn.cursor() as cursor:
                cursor.execute(query)
                logger.info(f"Loaded data into table {table_name} from {hdfs_path}.")
        except Exception as e:
            logger.error(f"Error loading data into table {table_name}: {e}")

    def run_all(self):
        schema_to_filename = {
            'owid': 'full_data',
            'csse_confirmed': 'time_series_covid19_confirmed_global',
            'google_cloud': 'main',
            'csse_deaths': 'time_series_covid19_deaths_global',
            'who': 'WHO-COVID-19-global-data'
        }
        for schema_key in ['owid', 'google_cloud', 'csse_confirmed', 'csse_deaths', 'who']:
            self.create_table_with_dynamic_schema(schema_to_filename.get(schema_key, "default_filename"

if __name__ == "__main__":
    hive_queries = HiveQueries(host='hive-server', hdfs_path_proc='/user/admin/data/processed')
    hive_queries.run_all()
```

Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Configurations



Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Configurations

localhost:9870/dfshealth.html#tab-overview

Hadoop | Overview | Datanodes | Datanode Volume Failures | Snapshot | Startup Progress | Utilities

Overview 'namenode:9000' (active)

Started:	Wed Aug 07 11:02:15 +0000 2024
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 15:56:00 +0000 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-43086225-1393-4770-9ab8-fb0381838092
Block Pool ID:	BP-261503638-172.22.0.8-1722983589259

Summary

Security is off.

Safemode is off.

542 files and directories, 496 blocks (496 replicated blocks, 0 erasure coded block groups) = 1 038 total filesystem object(s).

Heap Memory used 114.18 MB of 254.5 MB Heap Memory. Max Heap Memory is 1.73 GB.

Non Heap Memory used 50.93 MB of 52.31 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	1006.85 GB
Configured Remote Capacity:	0 B
DFS Used:	604.67 MB (0.06%)
Non DFS Used:	32.56 GB

Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Configurations

localhost:8080

Spark 3.2.0

Spark Master at spark://63b42c644563:7077

URL: spark://63b42c644563:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 6.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240807110215-172.22.0.11-38211	172.22.0.11:38211	ALIVE	4 (0 Used)	6.8 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Configurations

localhost:8080

Spark 3.2.0

Spark Master at spark://63b42c644563:7077

URL: spark://63b42c644563:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 6.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240807110215-172.22.0.11-38211	172.22.0.11:38211	ALIVE	4 (0 Used)	6.8 GiB (0.0 B Used)	

Running Applications (0)

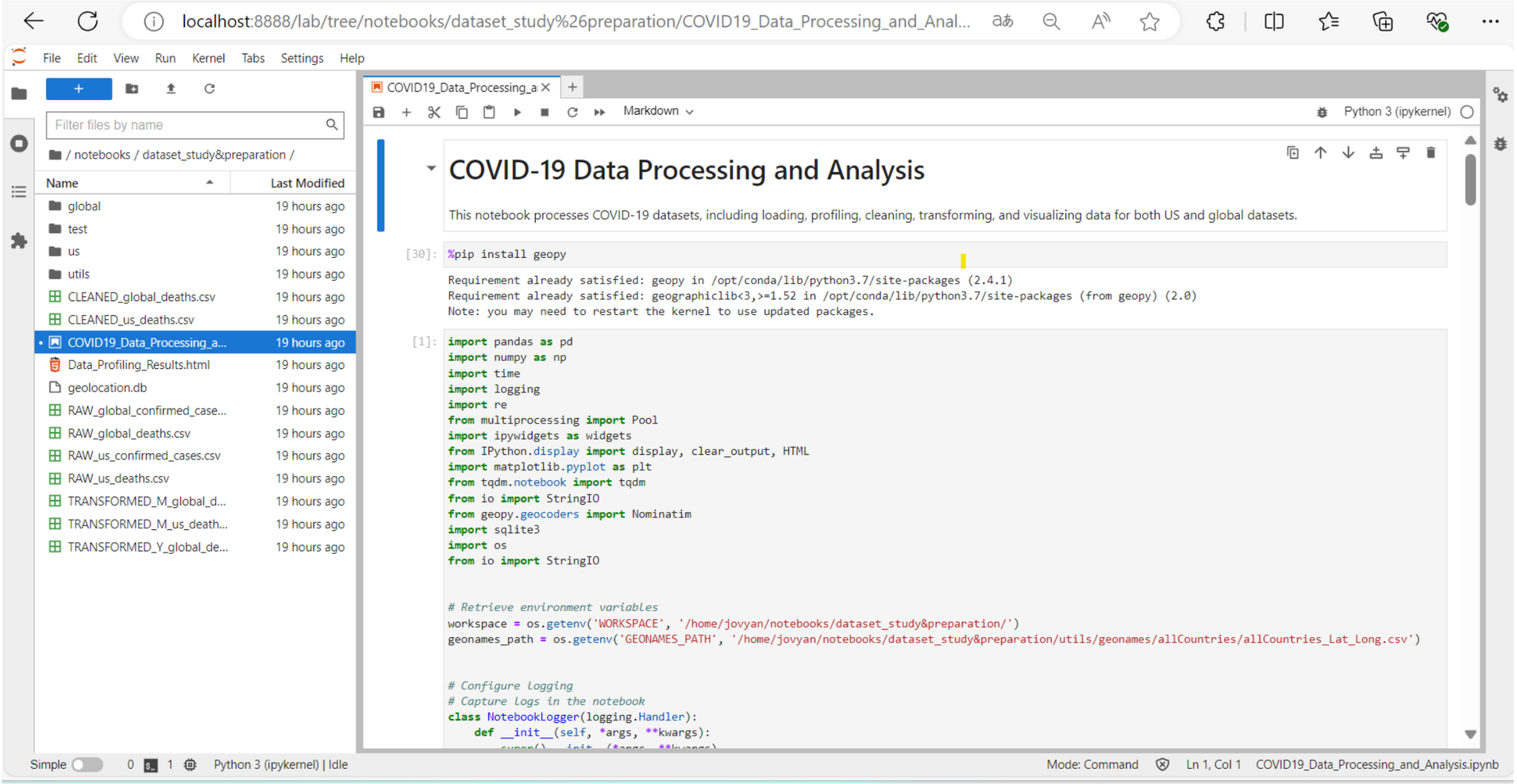
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

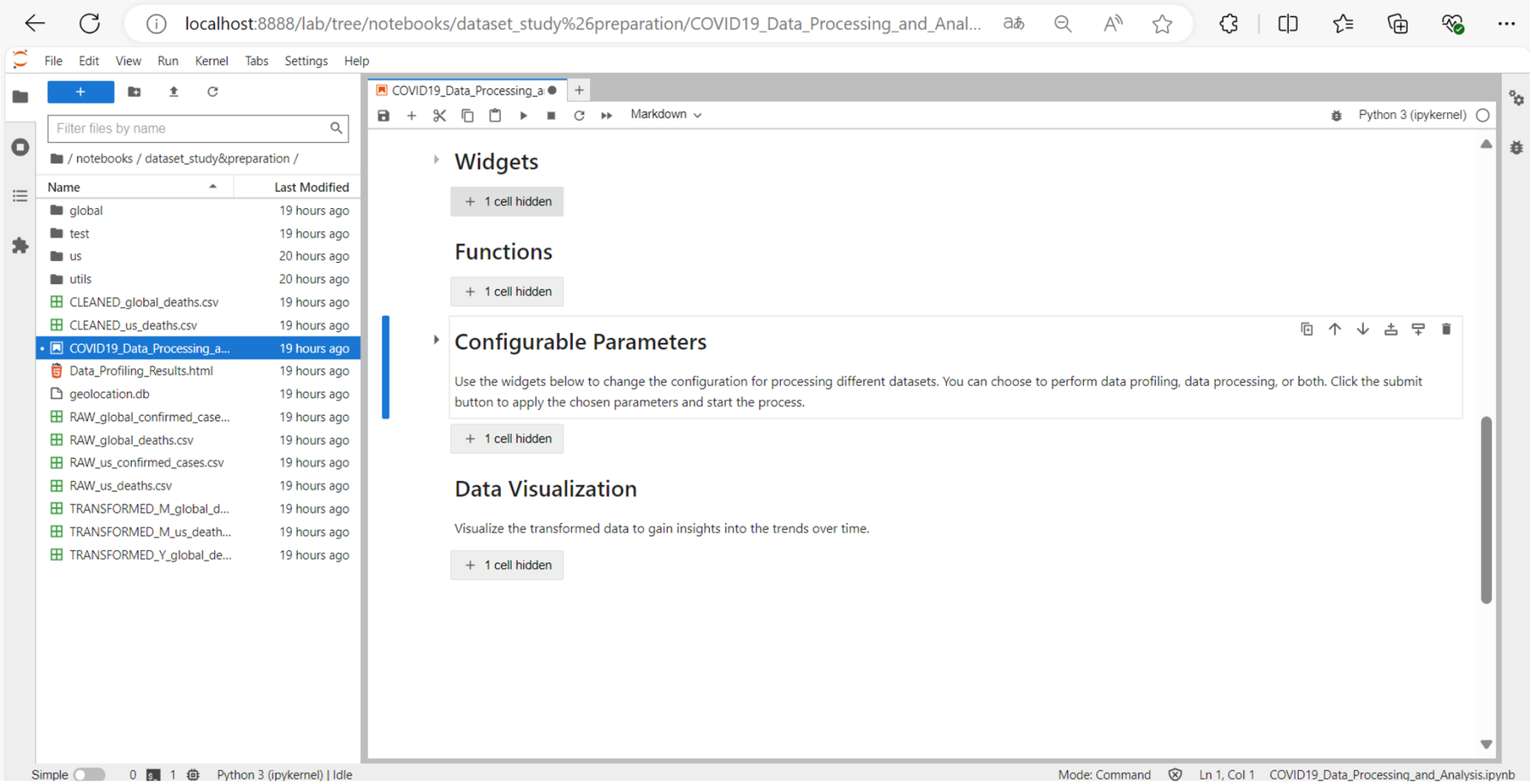
Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement



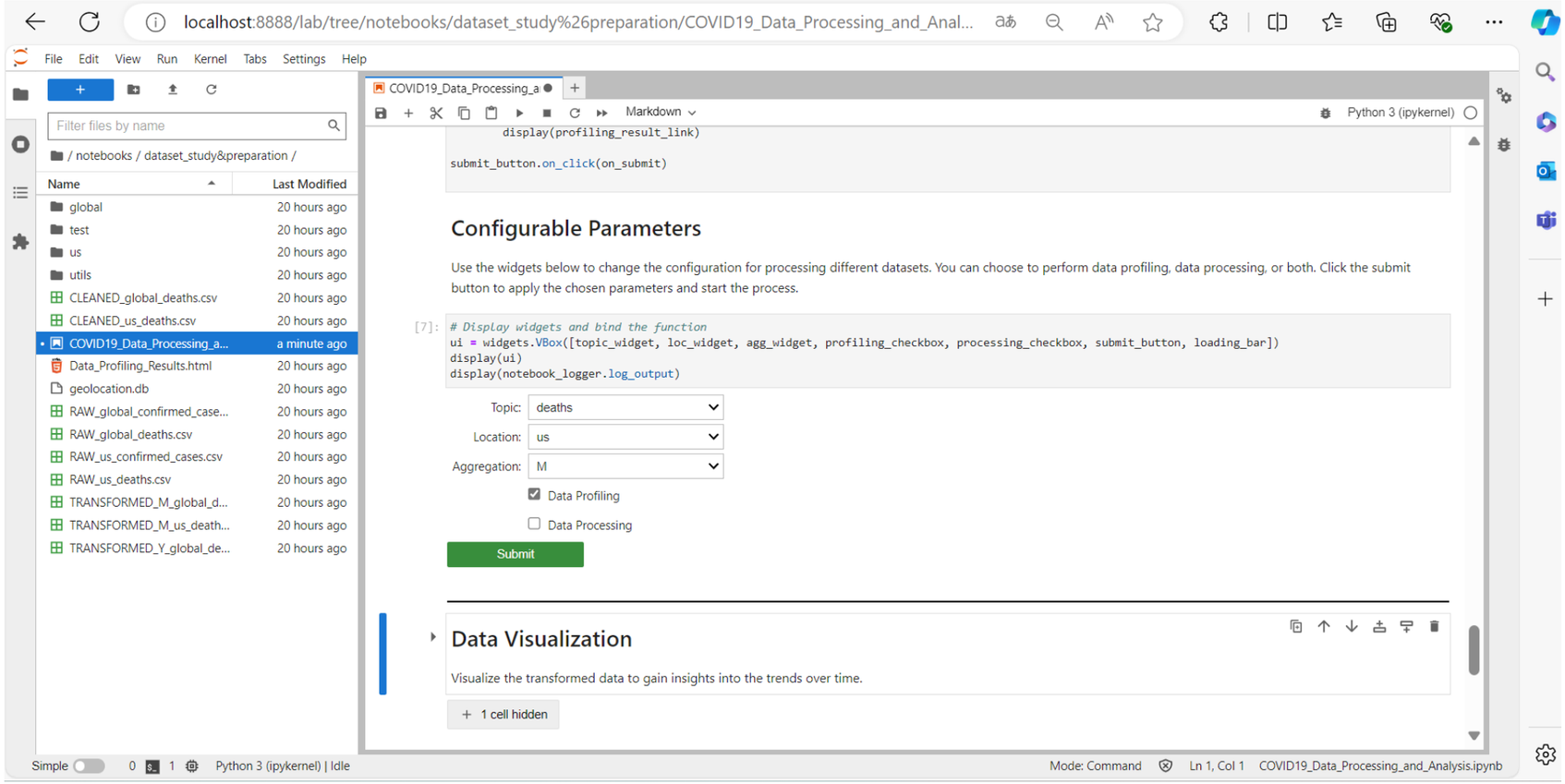
Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement



Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement



Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement

Configurable Parameters

Use the widgets below to change the configuration for processing different datasets. You can choose to perform data profiling, data processing, or both. Click the button to apply the chosen parameters and start the process.

[7]:
```
# Display widgets and bind the function
ui = widgets.VBox([topic_widget, loc_widget, agg_widget, profiling_checkbox, processing_checkbox, submit_button, loading_bar])
display(ui)
display(notebook_logger.log_output)
```

Topic: deaths

Location: us

Aggregation: M

☑ Data Profiling

☐ Data Processing

Submit

Loading:

```
Profiling file: /home/jovyan/notebooks/dataset_study&preparation/RAW_us_deaths.csv
Loading data from /home/jovyan/notebooks/dataset_study&preparation/RAW_us_deaths.csv
Profiling data
```

Data Visualization

/notebooks/dataset_study%26preparation/Data_Profiling_Results.html

COVID19_Data_Processing_a | Data_Profiling_Results.html

Trust HTML

Data Profiling Results for /home/jovyan/notebooks/dataset_study&preparation/RAW_us_de

Head of the DataFrame

	Province_State	Admin2	UID	iso2	iso3	code3	FIPS	Country_Region	Lat	Long_	Combined_Key	Popu
0	Alabama	Autauga	84001001	US	USA	840	1001.0	US	32.539527	-86.644082	Autauga, Alabama, US	5586
1	Alabama	Baldwin	84001003	US	USA	840	1003.0	US	30.727750	-87.722071	Baldwin, Alabama, US	2232
2	Alabama	Barbour	84001005	US	USA	840	1005.0	US	31.868263	-85.387129	Barbour, Alabama, US	2468
3	Alabama	Bibb	84001007	US	USA	840	1007.0	US	32.996421	-87.125115	Bibb, Alabama, US	2239
4	Alabama	Blount	84001009	US	USA	840	1009.0	US	33.982109	-86.567906	Blount, Alabama, US	5782

Shape of the DataFrame

Shape: (3342, 1155)

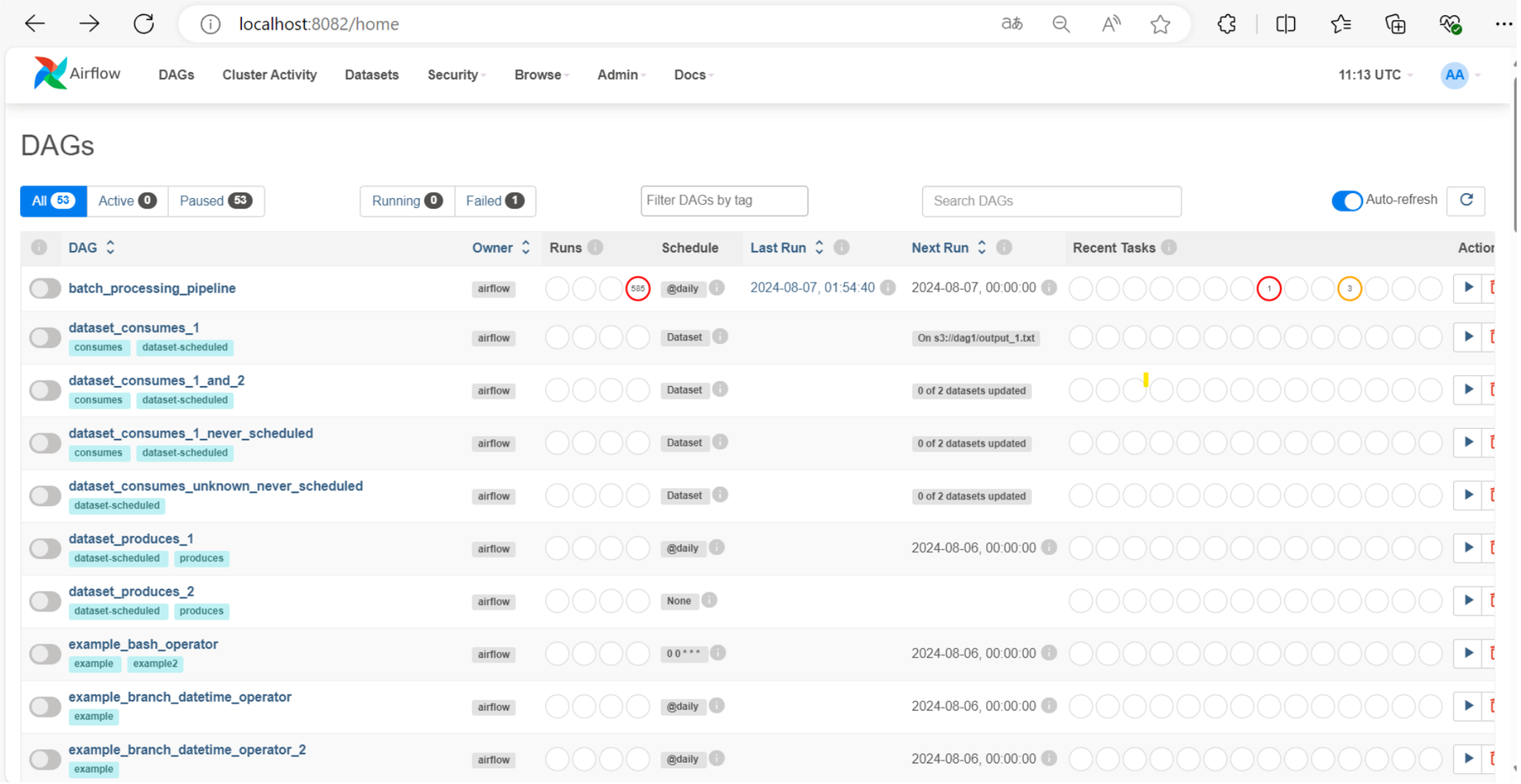
Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement

```
barryma@DESKTOP-LBFN3RA:~$ docker ps | grep dev
35d0a9684518   barryma22/cov19-dev                                  "tail -f /dev/null"      21 minutes ago   Up 21 minu
tes               0.0.0.0:5000->5000/tcp                                  project_covid19_final-dev-1
barryma@DESKTOP-LBFN3RA:~$ docker exec -it project_covid19_final-dev-1 bash
root@35d0a9684518:/app# python main.py

COVID-19 Data Pipeline Menu
1. Direct Download
2. Kaggle Download
3. Web Scraping
4. Data Processing
5. Data Analysis
6. Jupyter
7. Exit
Enter your choice:
```

Annexe 3 : Captures d'écran des configurations et des pipelines de traitement

Pipelines de traitement



Bibliographie

Documentation officielle d'Apache Spark:

Apache Spark Documentation

Documentation officielle de Hadoop:

Apache Hadoop Documentation

Documentation officielle de Hive:

Apache Hive Documentation

Ressources en ligne et forums techniques:

Stack Overflow: https://stackoverflow.com/
GitHub: https://github.com/
Medium: https://medium.com/

Merci !