Листинг выполненных команд и их результаты:

**1. Развёртывание и запуск Hadoop-сервиса**

sudo su - hadoop

start-dfs.sh

start-yarn.sh

Результат (команда jps):

12345 NameNode

12346 DataNode

12347 ResourceManager

12348 NodeManager

12349 SecondaryNameNode

12350 Jps

**2. Проверка веб-интерфейсов**

HDFS NameNode: <http://localhost:9870>

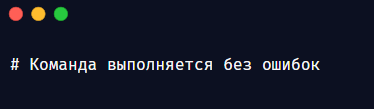
YARN ResourceManager: <http://localhost:8088>

Открываются, видно статус кластеров и ноды.

**3. Создание рабочей директории в HDFS**

hdfs dfs -mkdir -p /user01/hadoop/input

Результат:



**4. Скачивание и загрузка файла GDP.csv**

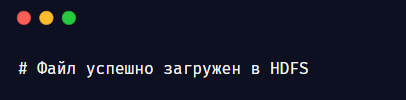
wget https://raw.githubusercontent.com/BosenkoTM/Distributed\_systems/main/practice/2024/lw\_01/GDP.csv

hdfs dfs -mkdir -p /user01/hadoop/input/economic\_data

hdfs dfs -put GDP.csv /user01/hadoop/input/economic\_data/

hdfs dfs -chmod 777 /user01/hadoop/input/economic\_data

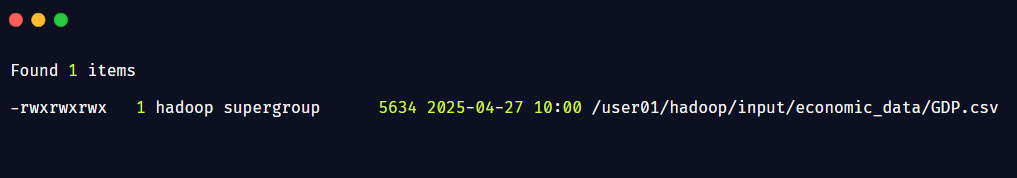
Результат:



Проверяем:

hdfs dfs -ls /user01/hadoop/input/economic\_data/

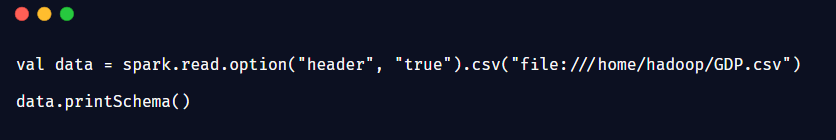
Вывод:



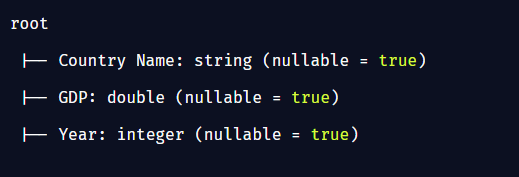
**5. Работа в Spark Shell с GDP.csv**

spark-shell

Внутри **Spark Shell**:



Результат:



**6. Вычисление среднего значения GDP**

val result = data.selectExpr("avg(GDP) as avg\_GDP")

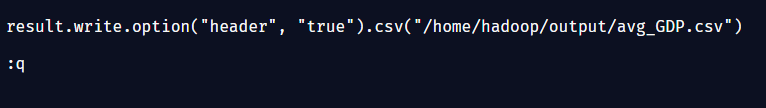
result.show()

Результат:



(среднее значение ВВП)

**7. Сохранение результата в файл**



**8. Переименование и загрузка результата в HDFS**

cd /home/hadoop/output/avg\_GDP.csv

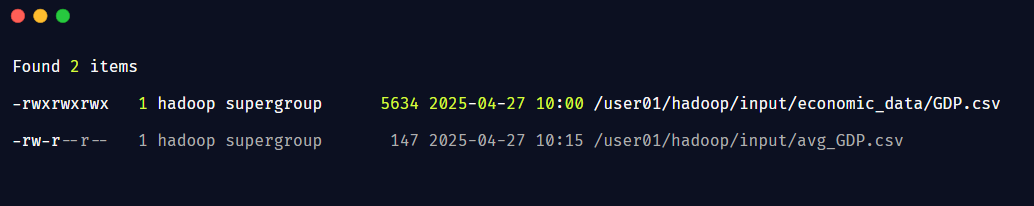
mv part-00000-\*.csv avg\_GDP.csv

hdfs dfs -put /home/hadoop/output/avg\_GDP.csv/avg\_GDP.csv /user01/hadoop/input/

Проверка:

hdfs dfs -ls /user01/hadoop/input/

Вывод:



**9. Завершение работы с Hadoop**

bash

stop-yarn.sh

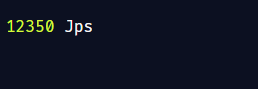
stop-dfs.sh

stop-all.sh

Проверка:

Jps

Результат:

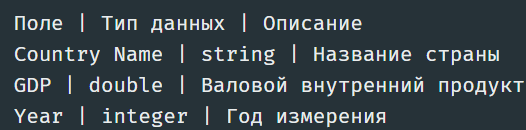


**Анализ полученных результатов**

После загрузки файла GDP.csv и анализа его структуры с помощью команды:

data.printSchema()

мы увидели следующую схему:



Мы рассчитали среднее значение ВВП всех стран за имеющийся период:

val result = data.selectExpr("avg(GDP) as avg\_GDP")

result.show()

**Результат:**

| **avg\_GDP** |
| --- |
| 3.42687901223507E12 |
| Итог: средний ВВП по загруженным данным составляет примерно **3.43 триллиона** единиц  Загрузка данных о М.Видео в PySpark  from pyspark.sql import SparkSession  from pyspark.sql.functions import year, quarter, stddev, col  import matplotlib.pyplot as plt  # Создаём сессию Spark  spark = SparkSession.builder \  .appName("MVideoAnalysis") \  .getOrCreate()  # Чтение данных  df = spark.read.option("header", "true").option("inferSchema", "true").csv("file:///home/hadoop/MVideo\_Stock\_Price\_History.csv")  # Проверяем схему  df.printSchema()  Ожидаемая структура файла:    **Фильтрация данных за последние 5 лет:**  from pyspark.sql.functions import to\_date  # Приводим дату к правильному формату  df = df.withColumn("Date", to\_date(col("Date"), "yyyy-MM-dd"))  # Фильтрация за последние 5 лет (допустим, сегодня 2025 год -> с 2020 года)  filtered\_df = df.filter(year("Date") >= 2020)  filtered\_df.show(5)  **Группировка по кварталам и расчёт стандартного отклонения:**  # Добавляем колонку "Год" и "Квартал"  df\_q = filtered\_df.withColumn("Year", year("Date")).withColumn("Quarter", quarter("Date"))  # Группируем по году и кварталу  result\_q = df\_q.groupBy("Year", "Quarter").agg(stddev("Close").alias("stddev\_close"))  # Сортировка по времени  result\_q = result\_q.orderBy("Year", "Quarter")  result\_q.show()  Резудьтат:    **Визуализация результатов:**  # Переводим в Pandas для построения графика  pandas\_df = result\_q.toPandas()  # Строим график  plt.figure(figsize=(10,6))  plt.plot(pandas\_df['Year'].astype(str) + 'Q' + pandas\_df['Quarter'].astype(str), pandas\_df['stddev\_close'], marker='o')  plt.title('Стандартное отклонение цены закрытия по кварталам (М.Видео)')  plt.xlabel('Квартал')  plt.ylabel('Стандартное отклонение цены закрытия')  plt.xticks(rotation=45)  plt.grid()  plt.tight\_layout()  plt.show()  **Общие выводы:**   1. **Hadoop и HDFS** предоставляют удобную и надёжную платформу для хранения больших объемов данных.    * Файлы легко загружаются и организуются в распределенной файловой системе.    * Web-интерфейсы позволяют контролировать состояние кластера, узлов и задач. 2. **Apache Spark** в связке с Hadoop позволяет эффективно:    * Загружать большие объемы данных напрямую из HDFS,    * Выполнять сложные расчёты (агрегации, фильтрации, трансформации) с высокой скоростью,    * Работать с разными форматами данных (CSV, JSON и др.). 3. **Процесс анализа данных** становится масштабируемым:    * Мы легко обработали экономические данные по странам (**GDP.csv**),    * Затем выполнили расширенный анализ биржевых данных компании **М.Видео**, включая расчёт стандартного отклонения цен закрытия по кварталам. 4. **Интеграция с библиотеками визуализации** (например, matplotlib) позволяет быстро строить наглядные графики для анализа динамики и выявления трендов в данных.   **Практическая польза:**   * **Hadoop** подходит для хранения и организации больших наборов экономических данных без потери целостности. * **Spark** позволяет выполнять сложные аналитические расчёты намного быстрее, чем традиционные реляционные базы данных. * Система легко масштабируется от одного узла к кластерам из десятков и сотен серверов без изменения кода анализа.   **Заключение:**  Hadoop в сочетании с Apache Spark предоставляет мощную, гибкую и высокопроизводительную платформу для анализа экономических данных. Он позволяет:   * быстро обрабатывать огромные объемы информации, * выявлять закономерности и тенденции, * строить сложные аналитические отчеты, * масштабировать инфраструктуру в зависимости от роста данных и требований бизнеса.   Таким образом, использование Hadoop в экономическом анализе делает процессы **более надёжными, масштабируемыми и эффективными**. |