# Memoria Practica 1

## Architecture de procesamiento de datos masivos

**Ksenia Turava Turav**

**Enunciado**
Instrucciones: Deberás instalar y configurar HDFS y Apache Spark en un entorno Docker, siguiendo los pasos detallados en el documento.
3. Secciones del proyecto
El proyecto se divide en secciones que abarcan desde la instalación hasta la ejecución de HDFS y Apache Spark en Docker.
4. Introducción a HDFS y su Ejecución en Diferentes Entornos
Exploramos qué es HDFS y cómo puede desplegarse en entornos locales, clústeres físicos, la nube y mediante contenedores Docker.
5. HDFS y Apache Spark en Entornos Docker y Cloud Computing
Aprendemos a instalar y configurar HDFS y Spark en Docker para simular un clúster distribuido y prepararnos para entornos de cloud computing.
6. Instalación del Proyecto (Docker)
Proporcionamos una guía paso a paso para configurar WSL2, Docker y clonar el repositorio del proyecto para iniciar los servicios necesarios.
7. Exploración y Operaciones Básicas en HDFS
Realizamos operaciones fundamentales en HDFS usando comandos básicos y la interfaz web de Hue para gestionar archivos.
8. Apache Spark (PySpark en HDFS)
Ejecutamos scripts en PySpark para procesar datos almacenados en HDFS, aprovechando el poder de Spark para el procesamiento en paralelo.
9. Resumen de Comandos Importantes
Compilamos una lista de comandos esenciales

# Instalaciones entorno Docker (Apple M3)

## Repositorio Github

https://github.com/turava/BigDataOps
Repositorio publico ya que enviar todos los archivos puede ser complejo en el portal UAX.

**$Command line**
Docker version 24.0.6
git version 2.43.0
git clone https://github.com/CanaletaFast/BigDataOps.git
cd BigDataOps
make —version GNU Make 3.81

## Arrancar el clúster Big Data con los contenedores

/ **BigDataOps**:

**$Command line**
make start-all
make start-spark

start-spark
Building Spark Base...
Detected OS: Darwin
Target Spark Version: 3.5.7
Building Jupyter Base...
Building Hadoop Base...
Starting Spark and Jupyter services...
**make: *** [start-spark] Error 1**
Docker Compose version v2.40.3-desktop.1

## Resolución de problemas en la construcción de imágenes base (Hadoop, Spark y Jupyter)

Durante el despliegue del entorno Big Data aparecieron dos incidencias principales:

1. **La imagen local-base-hadoop:latest no existía**, lo que provocaba que Docker intentara buscarla en Docker Hub y fallara.
   **Solución:** construir manualmente la imagen base de Hadoop.

2. **Las imágenes de Spark y Jupyter fallaban al descargar Spark**, debido a una URL generada incorrectamente. curl obtenía HTML en lugar del .tgz, y tar fallaba.
   **Solución:** descargar Spark manualmente en el host y copiar el archivo dentro de las imágenes durante la build.

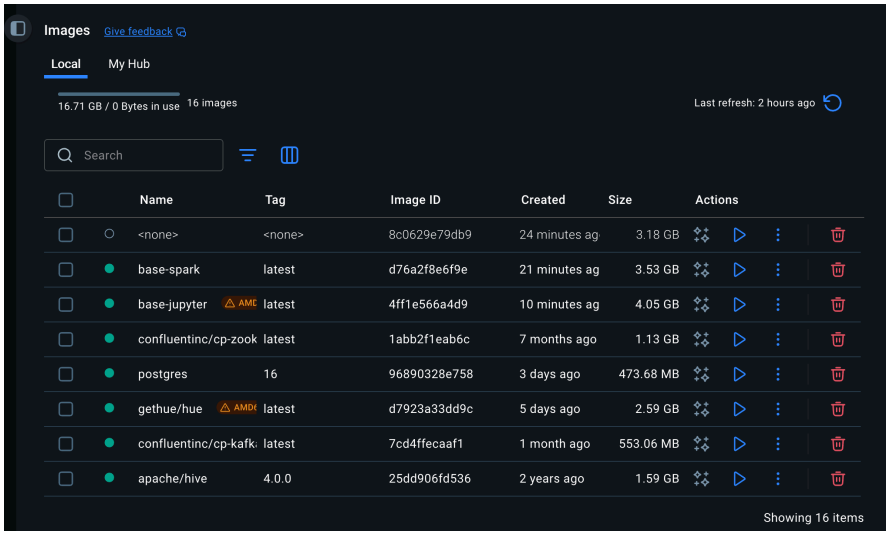
Con ambas correcciones, el clúster se construyó correctamente y se validó su funcionamiento accediendo a Hue y Jupyter.

**$Command line**

```
# Construcción Hadoop base
docker build -t local-base-hadoop:latest ./hadoop/base
# Descarga manual (host)
spark-3.4.2-bin-hadoop3.tgz
# Construcción Spark y Jupyter
docker build -t base-spark:latest ./spark
docker build -t base-jupyter:latest ./jupyter
# Reconstrucción servicios Hadoop
docker compose build namenode datanode1 datanode2 resourcemanager nodemanager historyserver
# Levantar clúster
docker compose up -d
docker ps
```

## Verificación

- Hue: http://localhost:9999
- Jupyter: http://localhost:8889/lab

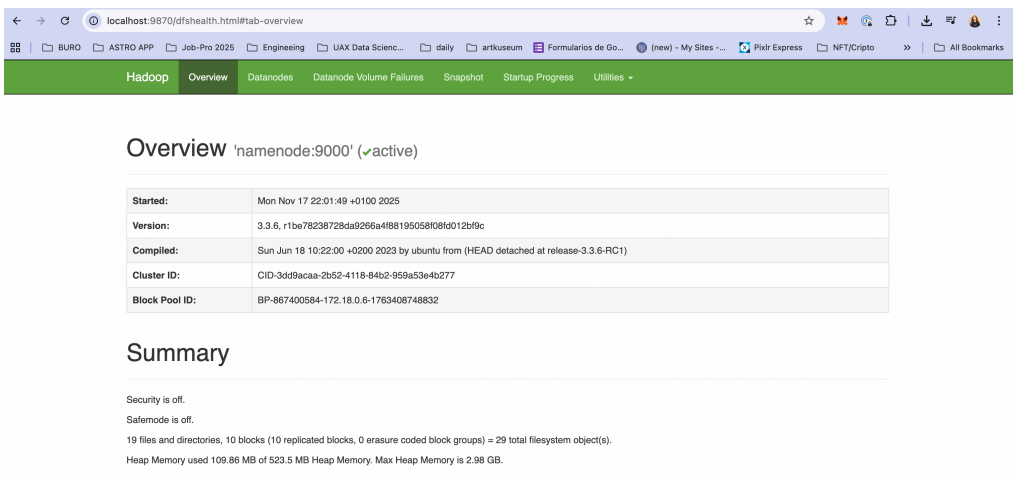



# Incidencia de Hue en Apple Silicon (resumen)

En sistemas Apple Silicon, la imagen oficial de Hue se ejecuta bajo emulación x86 y utiliza librerías como **Polars**, compiladas para procesadores con instrucciones **AVX/AVX2**, no disponibles en ARM. Esto provoca un fallo interno (Illegal instruction) durante la inicialización de Hue, impidiendo que la interfaz web pueda arrancar.
Debido a esta incompatibilidad, las operaciones sobre HDFS se realizaron desde la consola del contenedor namenode en lugar de utilizar Hue.

**$Command line**

```
# Copiar el CSV al volumen del namenode
cp /ruta/local/bus_trips.csv ./hadoop/applications/
# Entrar en el contenedor namenode
docker exec -it namenode bash
# Crear carpeta en HDFS
hdfs dfs -mkdir -p /datos-uax
# Subir el fichero CSV a HDFS
hdfs dfs -put /hadoop/applications/bus_trips.csv /datos-uax
# Verificar la carga del fichero
hdfs dfs -ls /datos-uax
```



# Conectar Spark desde el Notebook

http://localhost:8889/lab/tree/Untitled.ipynb

## En Jupyter

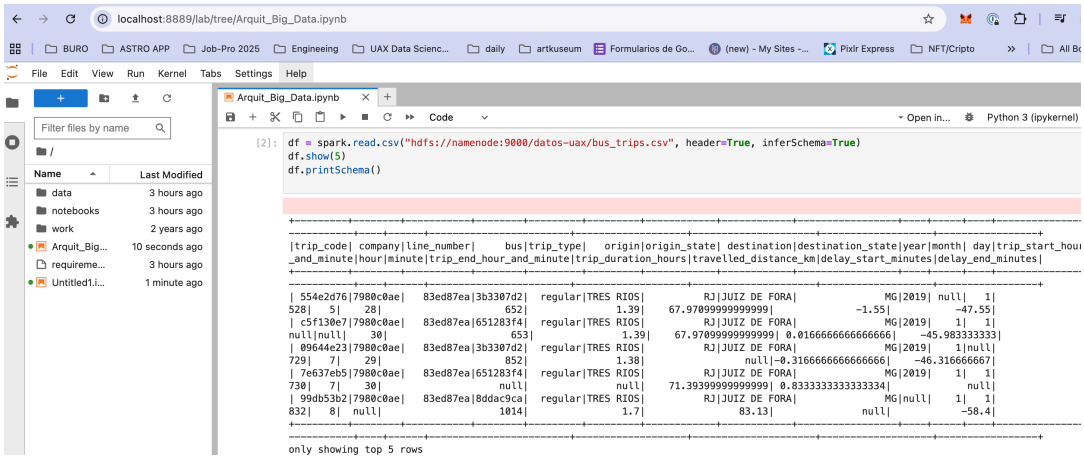
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("BusTripsAnalysis") \
    .master("spark://spark-master:7077") \
    .getOrCreate()

# Lectura del CSV desde HDFS
df = spark.read.csv("hdfs://namenode:9000/datos-uax/bus_trips.csv", header=True,
inferSchema=True)
df.show(5)
df.printSchema()

# Operaciones básicas
df.count()
df.select("route_id", "trip_id").show(5)
df.filter(df["route_id"] == "1").show(5)
df.groupBy("route_id").count().show()

spark.stop()
```




Se cargó el fichero bus_trips.csv desde HDFS mediante Spark, utilizando inferSchema para obtener automáticamente el tipo de cada columna. Tras validar el esquema, se realizaron operaciones básicas de análisis: conteo de registros, agregaciones por empresa, cálculo de medias (duración del viaje, retrasos y distancia recorrida) y distribución temporal por año y mes. Estas acciones confirman la correcta integración entre Spark y el sistema de almacenamiento distribuido.

## Carga de un archivo de texto sencillo en HDFS

Como parte de la práctica, se solicita generar un archivo de texto simple para comprobar el funcionamiento básico del sistema de ficheros distribuido HDFS. Este fichero se utiliza posteriormente como entrada para un procesamiento elemental en Spark (word count), demostrando que el clúster puede gestionar tanto datasets estructurados como archivos de texto plano.

El archivo simple_text.txt fue creado localmente y transferido al contenedor del Namenode mediante docker cp. Una vez dentro del contenedor, se almacenó en HDFS dentro del directorio de trabajo /datos-uax/. Finalmente, se verificó su correcta subida consultando el listado del directorio con hdfs dfs -ls.

**$Command line**

```
# Copiar el archivo simple_text.txt al contenedor Namenode
docker cp simple_text.txt namenode:/tmp/simple_text.txt
# Acceder al contenedor Namenode
docker exec -it namenode bash
# Subir el archivo a HDFS
hdfs dfs -put /tmp/simple_text.txt /datos-uax/
# Verificar que el archivo está disponible en HDFS
hdfs dfs -ls /datos-uax/
```

## Procesamiento básico en Spark: Word Count

Como parte del ejercicio, se requería ejecutar un proceso sencillo en Spark utilizando un archivo de texto cargado previamente en HDFS. El objetivo es validar que Spark puede:

- leer archivos desde HDFS,
- realizar transformaciones distribuidas,
- producir resultados derivados,
- y almacenarlos nuevamente en el sistema distribuido.

Para ello se creó el script process_bus_trips.py, que implementa un conteo de palabras (word count) a partir del archivo simple_text.txt. El script genera un DataFrame con la frecuencia de cada palabra y escribe el resultado en una nueva carpeta dentro de HDFS.

Este flujo confirma la integración completa entre Spark y HDFS dentro del entorno desplegado.

## Procesamiento de un archivo de texto con Spark

Para verificar la correcta integración entre Spark y HDFS, se desarrolló un script de procesamiento denominado process_bus_trips.py. Este script implementa un proceso sencillo de *word count* a partir del archivo simple_text.txt, previamente almacenado en

HDFS dentro del directorio /datos-uax/.
El objetivo es demostrar que Spark es capaz de:

- leer datos directamente desde HDFS,
- realizar transformaciones distribuidas,
- generar resultados derivados,
- y escribirlos nuevamente en HDFS.

De este modo, se valida el flujo completo de procesamiento de datos dentro del entorno distribuido desplegado con Docker.

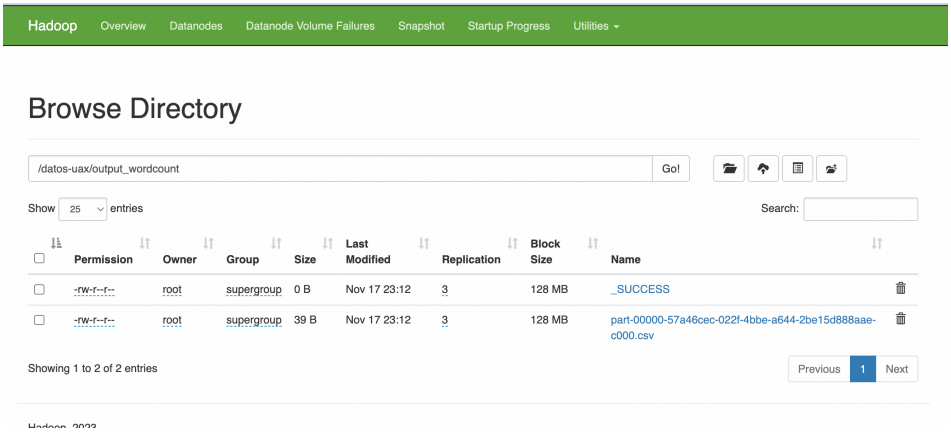
El script se ubicó dentro de la ruta del proyecto: airflow/dags/pyspark_apps/ process_bus_trips.py y el procesamiento se ejecutó mediante spark-submit dentro del contenedor spark-master.

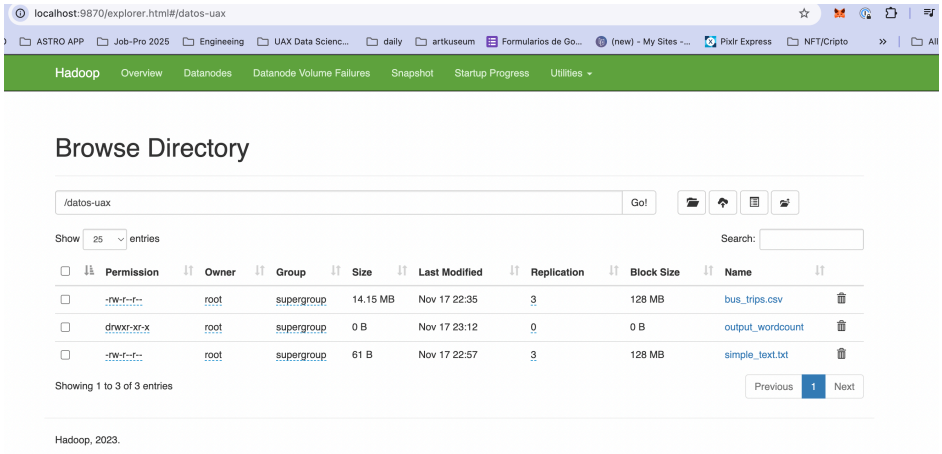
**Coding Script:**

```
from pyspark.sql import SparkSession
# Crear sesión de Spark
spark = SparkSession.builder.appName("WordCount").getOrCreate()
# Lectura del archivo en HDFS
df = spark.read.text("hdfs://namenode:9000/datos-uax/simple_text.txt")
# Transformación: generar palabras individuales
words = df.selectExpr("explode(split(value, ' ')) as word")
# Conteo de frecuencias
word_counts = words.groupBy("word").count()
# Mostrar resultado en consola
word_counts.show()
# Guardar el resultado en HDFS
word_counts.write.csv(
    "hdfs://namenode:9000/datos-uax/output_wordcount",
    mode="overwrite"
)
spark.stop()
```

## $ Commands

```
# Copiar el script al contenedor Spark
docker cp airflow/dags/pyspark_apps/process_bus_trips.py spark-master:/tmp/
# Ejecutar el proceso con Spark
docker exec -it spark-master bash
cd /tmp
spark-submit process_bus_trips.py
```




# Manual de Uso del Entorno Big Data (Guía Rápida)

## 1. Iniciar el entorno

Desde la carpeta del proyecto:
docker compose up -d
Comprobar servicios:
docker ps

## 2. Acceder a las interfaces principales

| Servicio | URL |
|---|---|
| HDFS NameNode | http://localhost:9870 |
| YARN ResourceManager | http://localhost:8088 |
| Spark Master | http://localhost:8080 |
| Spark History Server | http://localhost:18080 |
| JupyterLab | http://localhost:8889 |

## 3. Subir archivos a HDFS

1. Copiar archivo al contenedor:

docker cp fichero.csv namenode:/tmp/

2. Entrar al namenode:

docker exec -it namenode bash

3. Crear carpeta y subir:

hdfs dfs -mkdir -p /misdatos
hdfs dfs -put /tmp/fichero.csv /misdatos/

4. Verificar:

hdfs dfs -ls /misdatos

## 4. Ejecutar un script Spark

1. Copiar el script:

docker cp script.py spark-master:/tmp/

2. Ejecutarlo:

docker exec -it spark-master bash
cd /tmp
spark-submit script.py

## 5. Leer o escribir archivos desde Spark (ejemplo)

spark.read.csv("hdfs://namenode:9000/misdatos/fichero.csv", header=True)
Guardar resultados:
df.write.csv("hdfs://namenode:9000/misdatos/salida", mode="overwrite")

## 6. Consultar resultados en HDFS

docker exec -it namenode bash
hdfs dfs -ls /misdatos/salida
hdfs dfs -cat /misdatos/salida/part-00000*.csv

## 7. Parar el entorno

docker compose down

## Repositorio Github

https://github.com/turava/BigDataOps
Repositorio publico ya que enviar todos los archivos puede ser complejo en el portal UAX.