Documentación Proyecto Hadoop + Hive + Spark

# 📂 Estructura del Proyecto

Proyecto-hive/  
│  
├── docker-compose.yml # Orquestación de servicios Hadoop + Hive + Spark  
├── Dockerfile # Imagen extendida de Hive con driver PostgreSQL  
├── hadoop.env # Variables de entorno de Hadoop  
├── hive.env # Variables de entorno de Hive  
├── hive-site.xml # Configuración de conexión de Hive al Metastore (PostgreSQL)  
├── postgresql-42.7.5.jar # Driver JDBC de PostgreSQL  
├── docker # (archivo vacío que no usamos, se puede borrar)  
└── app/ # Carpeta para notebooks o scripts Spark (Python/Scala)

# 🔹 Función de cada archivo

- docker-compose.yml: Define servicios Hadoop core (namenode, datanode, etc.), Hive (metastore + server), PostgreSQL y Spark. Configura puertos, volúmenes y dependencias.  
  
- Dockerfile: Extiende la imagen de Hive (bde2020/hive) para incluir el driver PostgreSQL en /opt/hive/lib/.  
  
- hadoop.env: Variables de entorno de Hadoop: CLUSTER\_NAME, configuración HDFS, YARN, etc.  
  
- hive.env: Variables de Hive, conecta Hive con PostgreSQL y HDFS.  
  
- hive-site.xml: Configuración crítica para el metastore. Define driver, URL de conexión, usuario y contraseña.  
  
- postgresql-42.7.5.jar: Driver JDBC para conexión de Hive con PostgreSQL.  
  
- app/: Scripts Spark o notebooks para procesar datos y construir el pipeline Bronze → Silver → Gold.

# 🔹 Flujo de trabajo

1. Levantar servicios en orden:  
 - Namenode y Datanode  
 - ResourceManager y NodeManager  
 - HistoryServer  
 - PostgreSQL (para metastore)  
 - Hive Metastore (con schema inicializado en PostgreSQL)  
 - Hive Server2  
 - Spark Master y Workers  
  
2. HDFS:  
 - Crear carpetas /bronze, /silver, /gold  
 - Cargar datasets: docker cp + hdfs dfs -put  
  
3. Hive:  
 - Crear tablas externas apuntando a carpetas en HDFS  
 - Ejemplo:  
 CREATE EXTERNAL TABLE movies (...)  
 ROW FORMAT DELIMITED FIELDS TERMINATED BY ','  
 STORED AS TEXTFILE  
 LOCATION '/data/input';  
  
4. Spark (carpeta app/):  
 - Scripts que limpian datos en Bronze y escriben en Silver  
 - Transformaciones adicionales hacia Gold  
 - Consultas desde Hive para validación

# 🔹 Comandos útiles

- Ver contenedores:  
 docker ps  
  
- Subir archivo local al contenedor namenode:  
 docker cp "C:/ruta/movie.csv" namenode:/tmp/movie.csv  
  
- Mover archivo a HDFS:  
 hdfs dfs -mkdir -p /data/input  
 hdfs dfs -put /tmp/movie.csv /data/input  
  
- Ver contenido en HDFS:  
 hdfs dfs -ls /data/input  
  
- Conexión Hive:  
 beeline -u jdbc:hive2://localhost:10000  
  
- Consulta rápida:  
 show databases;  
 show tables;  
 select count(\*) from movies;  
**Documento Técnico del Pipeline**

**1. Arquitectura**

* **Fuente de datos (Source):** SQL Server on-premise/local
* **Ingestión:** Spark (conector JDBC + driver mssql-jdbc)
* **Almacenamiento inicial (Bronze):** HDFS → datos crudos en formato **Parquet (Snappy)**
* **Procesamiento (Silver):** Spark limpia, estandariza y guarda datos listos para consumo analítico
* **Almacenamiento intermedio (Silver):** HDFS → datos refinados en Parquet
* **Capa de consulta:** Hive External Tables para análisis SQL

**2. Contenedores (Docker Compose)**

* spark-master → ejecuta jobs PySpark
* namenode, datanode → HDFS para almacenamiento
* hive-metastore + hive-server2 → consultas SQL externas
* resourcemanager y nodemanager → YARN para scheduling

**Scripts creados**

**🔹 test\_connection.py**

* Objetivo: validar conexión entre Spark ↔ SQL Server
* Acciones:
  + Se conecta con jdbc:sqlserver://host.docker.internal:1433
  + Lista las tablas de la BD olva

**🔹 extract\_clientes.py**

* Objetivo: extraer tabla Clientes desde SQL Server → HDFS Bronze
* Acciones:
  + Lee datos con .read.jdbc()
  + Guarda en /bronze/clientes en formato **Parquet Snappy**
  + Imprime registros de prueba

**🔹 process\_clientes.py**

* Objetivo: transformar datos de Bronze → Silver
* Acciones:
  + Lee datos de /bronze/clientes
  + Normaliza nombres (quita tildes, espacios, etc.)
  + Elimina duplicados
  + Guarda en /silver/clientes

**🔹 pipeline\_clientes.py**

* Objetivo: orquestar los pasos anteriores
* Acciones:
  1. Ejecuta extract\_clientes.py
  2. Espera 10s (para asegurar escritura)
  3. Ejecuta process\_clientes.py
  4. Confirma datos guardados en Silver

**5. Problemas encontrados y soluciones**

**Montaje de contenedores (Docker)**

* **Error al levantar servicios con docker compose up (contenedores que se caían o quedaban en unhealthy)**  
   Solución: revisar logs de cada contenedor (docker logs <container\_id>), corregir rutas de volúmenes, esperar a que el metastore de Hive inicialice antes de levantar hive-server2.
* **Problemas de red entre contenedores (Spark ↔ SQL Server local)**  
   Solución: usar host.docker.internal como hostname en lugar de la IP local o nombre de máquina (DESKTOP-XXXX), habilitar el puerto 1433 en el firewall con New-NetFirewallRule.
* **No se encontraba el driver JDBC en Spark**  
   Solución: montar el .jar (mssql-jdbc-13.2.0.jre8.jar) en la carpeta /opt/spark/jars del contenedor y añadirlo en la configuración de Spark (--jars y spark.jars).

**Ejecución de scripts PySpark**

* **Error con f-strings (SyntaxError: invalid syntax)**  
   Solución: contenedor trae Python 2.7 por defecto → reemplazar f-strings (f"...") por .format() para compatibilidad.
* **Error inicial de conexión JDBC**  
   Solución: confirmar que SQL Server estaba escuchando en 0.0.0.0:1433 y habilitar protocolo TCP/IP en SQL Server Configuration Manager.

**HDFS y Spark**

* **No se visualizaban los datos desde el contenedor**  
   Solución: usar hdfs dfs -ls /bronze/clientes desde namenode para listar y validar \_SUCCESS y archivos .parquet.
* **Procesamiento Silver no se ejecutaba después de extracción**  
   Solución: añadir un sleep(10) en el pipeline para dar tiempo a Spark/HDFS de finalizar la escritura antes de leer en el siguiente paso.

Captura de las Soluciones

Carpetas creadas en HDFS para el almacenamiento y procesamiento d ellos datos.

Texto

El contenido generado por IA puede ser incorrecto.

Creación de la base de datos en HIVE para el proyecto y creación de la tabla para probar funcionamiento de HIVE

Texto

El contenido generado por IA puede ser incorrecto.

Ejecución del Pipeline de datos Texto

El contenido generado por IA puede ser incorrecto.

Consulta a directorio de HDFS Bronce sobre los datos extraídos de SQL server de mi maquina local

Texto

El contenido generado por IA puede ser incorrecto.

Consulta al directorio Silver de HDFS a los datos Procesados desde Bronce

Texto

El contenido generado por IA puede ser incorrecto.

Resultados en Hive y su tabla externa apuntado al directorio silver/clientes

Texto

El contenido generado por IA puede ser incorrecto.

SQL Server Local con sus datos de pruebas

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Servicios levantados en Docker-compose

Texto

El contenido generado por IA puede ser incorrecto.

**Conclusiones**

Ha sido una experiencia enriquecedora que me permitió reafirmar la importancia de un Ingeniero de Datos o Desarrollador Big Data no radica únicamente en conocer las herramientas, sino en comprender cómo orquestar ecosistemas complejos para crear soluciones robustas y escalables.

Quiero expresar mi agradecimiento por esta oportunidad, ya que me permitió poner en práctica mis conocimientos y al mismo tiempo enfrentar nuevos retos dentro del ecosistema Big Data.

Más allá del resultado, valoro enormemente lo aprendido, pues cada error superado me permitió fortalecer mis capacidades como Ingeniero/Desarrollador Big Data. Así, incluso si no llego a quedar seleccionado, esta experiencia ya representa un paso importante en mi crecimiento profesional y una motivación extra para seguir avanzando en este camino.