

Índice

[Instalación y configuración del cluster 3](#_Toc190265668)

[Almacenamiento y tratamiento de datos 4](#_Toc190265669)

[HDFS 4](#_Toc190265670)

[MapReduce Job: 6](#_Toc190265671)

[Pig Script 7](#_Toc190265672)

[Sqoop 10](#_Toc190265673)

[Flume 15](#_Toc190265674)

[Procesamiento avanzado de datos 16](#_Toc190265675)

[Apache Spark 17](#_Toc190265676)

Cluster Hadoop

Tendréis que ir respondiendo a las preguntas utilizando comentarios en caso de que os pregunten algo y/o capturas de pantalla.

IMPORTANTE: Asegúrate de documentar todos los cambios realizados y utiliza capturas de pantalla para justificar cada paso completado. Incluye cualquier código necesario para las configuraciones y ejecuciones.

# Instalación y configuración del cluster

1. Describe brevemente los contenedores actuales de vuestro cluster.

**NameNode**: Administra el sistema de archivos distribuido de Hadoop (HDFS) y mantiene el namespace.

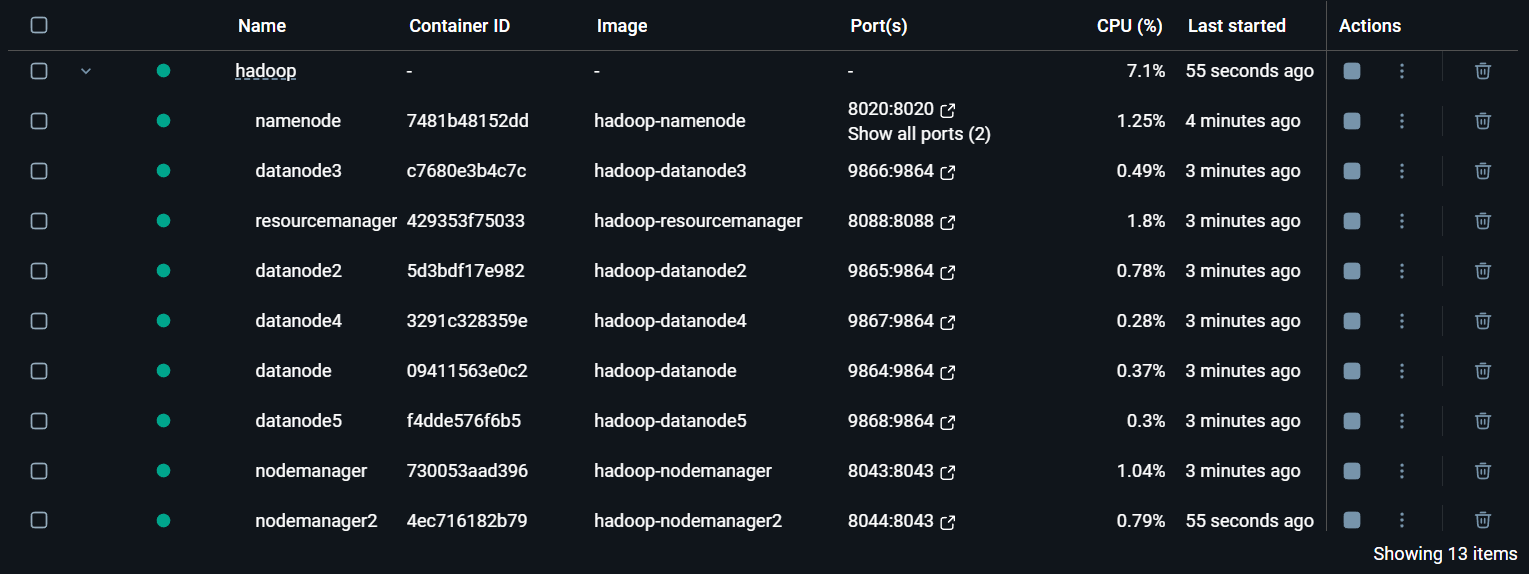
**NodeManager**: Ejecuta las tareas de las aplicaciones en cada nodo y monitorea el estado de los contenedores.

**ResourceManager**: Administra los recursos y coordina la ejecución de las aplicaciones en el clúster.

**DataNode**: Almacena los bloques de datos del HDFS.

1. Haz los cambios necesarios en el Dockerfile y Docker Compose para añadir 4 nuevos nodos Datanode y un nuevo nodo NodeManager.

¿Podría añadir otro nodo ResourceManager o Namenode? ¿Cómo mejorarías el Cluster?



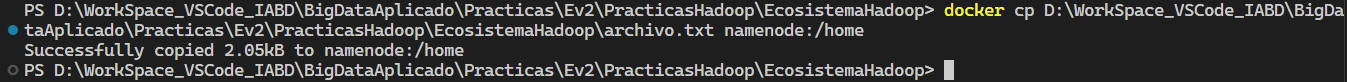
Por poder, pueden añadirse tanto ResourceManager como el NameNode.

Para mejorarlo se podria meter un segundary namenode o un history server.

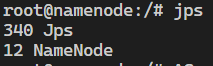
# Almacenamiento y tratamiento de datos

## HDFS

1. Crea un archivo txt cuyo contenido es tu nombre y la fecha actual y súbelo al cluster. Comprueba que está subido en HDFS a través de la página web ofrecida por el NameNode.



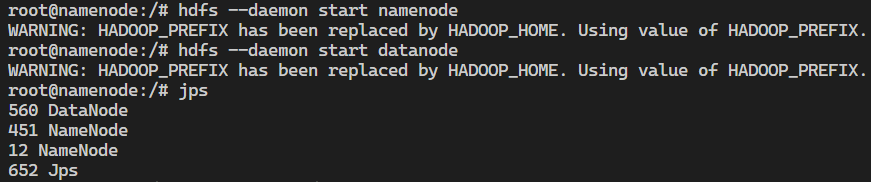
Subimos el archivo al namenode



Como no esta activo el datanode, tendremos que arrancarlo de forma manual

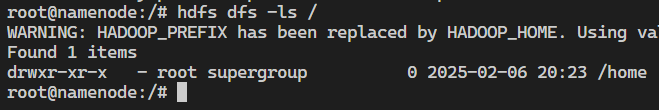


Este comando lo usaremos solo si es la primera vez, ya que esto borra datos en hdfs

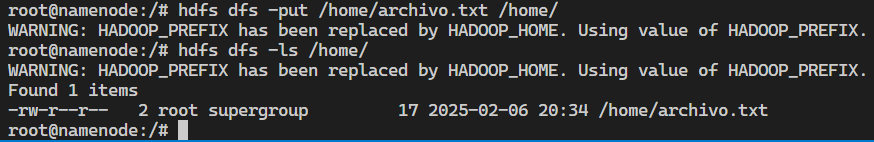
  
Despues iniciamos el namenode y el datanode en -d y con jps vemos que están arrancados.



Creamos un directorio /home

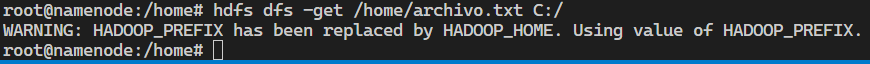


Vemos que se ha creado correctamente

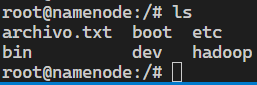


Subimos el archivo.txt a hdfs y comprobamos que se ha subido correctamente.

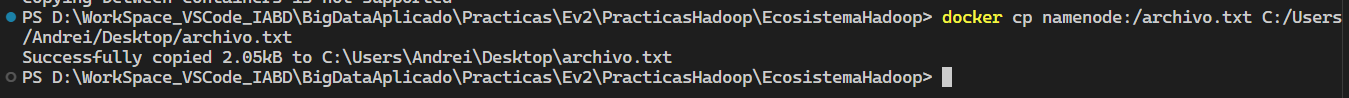
1. Haz todos los pasos necesarios para descargar tu archivo en el Desktop de tu HOST.



Lo descargamos desde HDFS

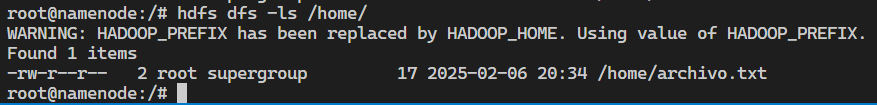


Se nos habrá puesto aquí



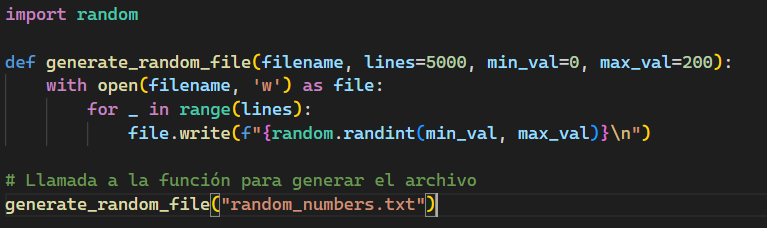
Y desde el namenode al Desktop

1. Muestra los permisos de los archivos subidos en HDFS.

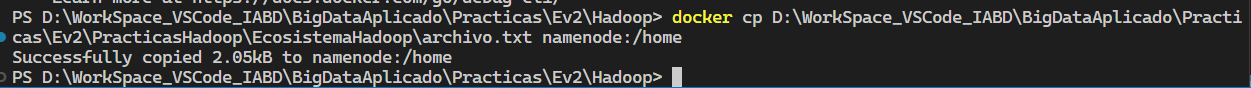


## MapReduce Job:

1. Genera un archivo con 5000 líneas donde cada línea será un número aleatorio entre 0 y 200 usando python.

  
Si ejecutamos el .py, generará un archivo random\_numbers.txt

1. Sube el archivo a HDFS en código.

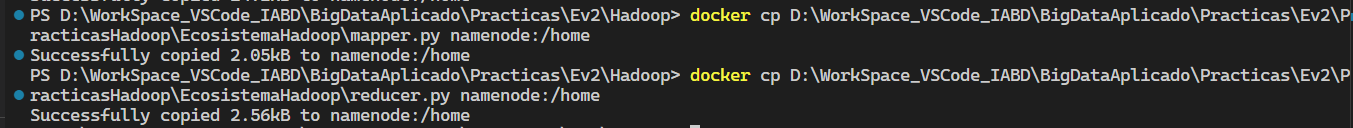


Lo subimos primero al namenode

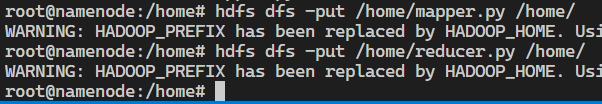


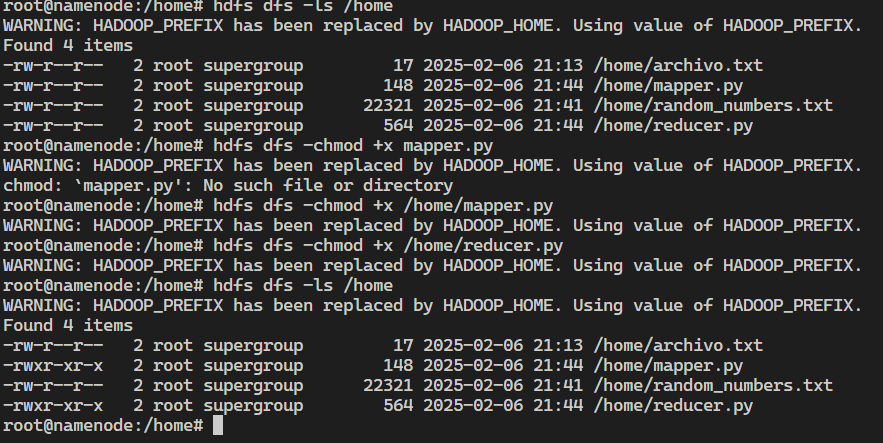
Lo subimos del namenode a HDFS

1. Ejecuta un trabajo MapReduce que calcule la mediana; el resultado debería ser un número.

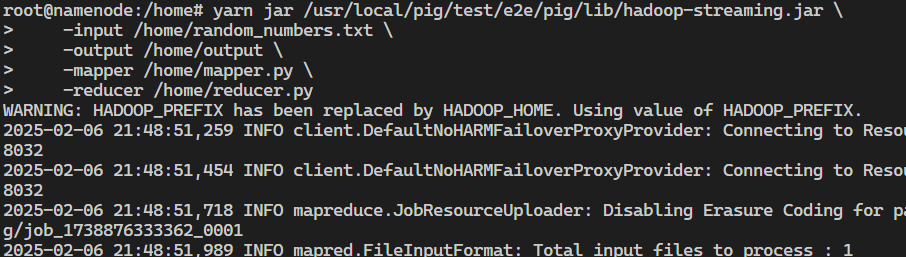


Subimos los archivos .py al namenode

  
Los subimos al HDFS



Una vez subidos, les ponemos permiso de ejecución

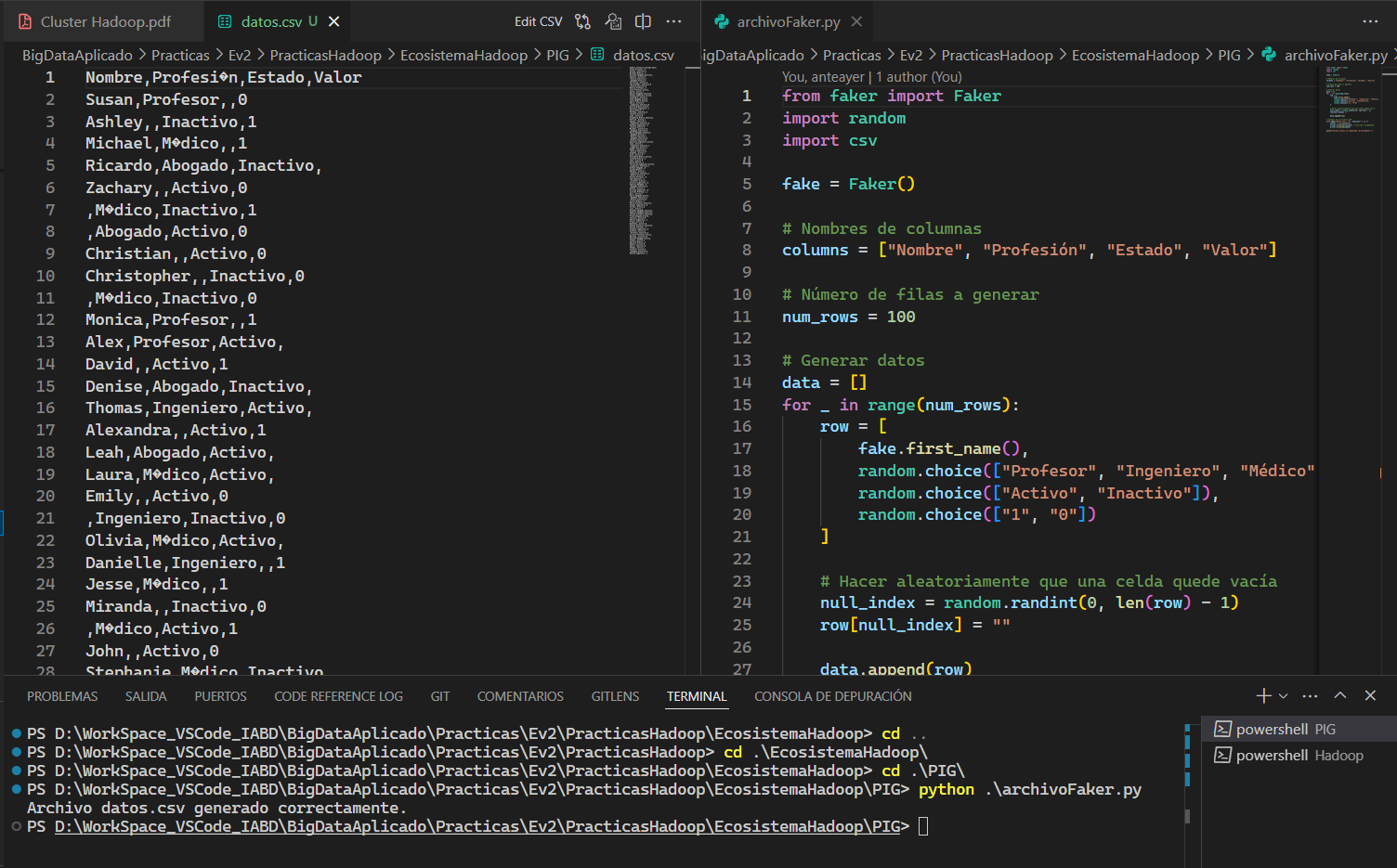


Ejecutamos el MapReduce

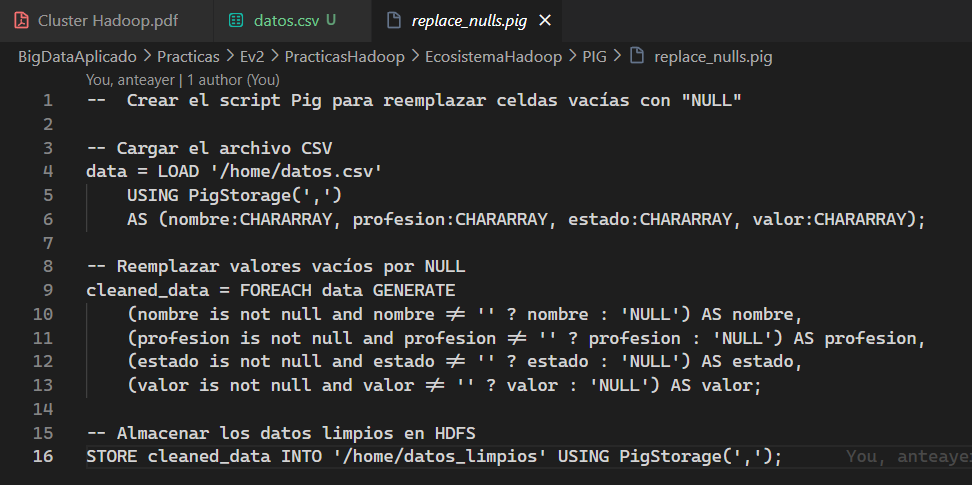
1. Utiliza HDFS para mostrar los datos. Pista: utiliza cat /\* de la carpeta generada en HDFS tras el trabajo.

# Pig Script

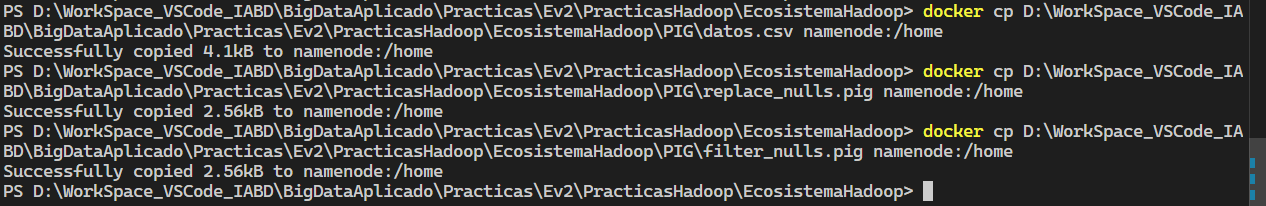
1. Genera un archivo utilizando Faker que devuelva un CSV con 4 columnas, de manera aleatoria, añade una celda sin ningún dato. Un ejemplo: [Jorge, Profesor, Activo, 1], [Rafael, Profesor,, 0].

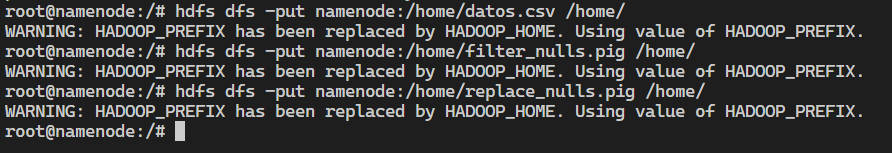


1. Crea un script Pig que ponga como valor a “NULL” esas celdas. La celda nula no tiene que pasar siempre en la misma columna, una fila puede tener más de 1 celda con “NULL”.

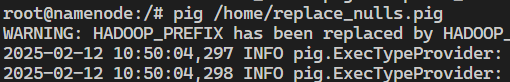


1. Ejecuta el script y muestra los datos en HDFS.

  
movemos los archivos a namenode



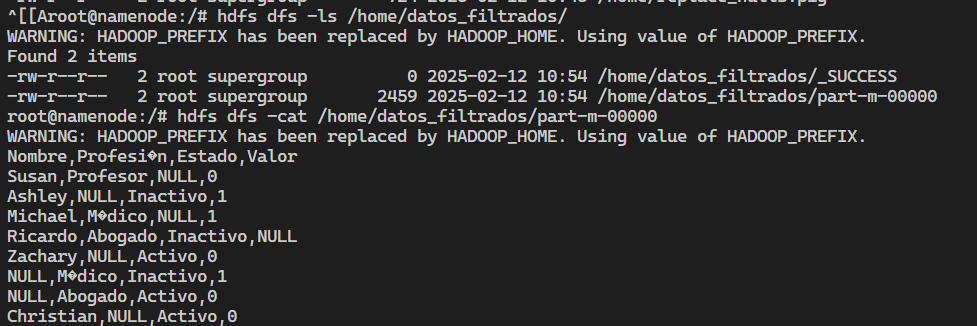
Subimos a hdfs los 3 ficheros



Ejecutamos el .pig

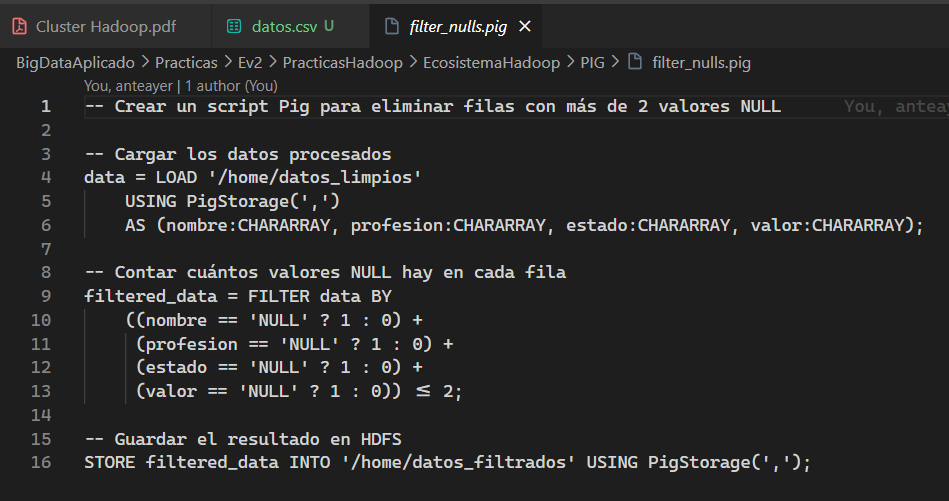


Captura de que se ha ejecutado correctamente

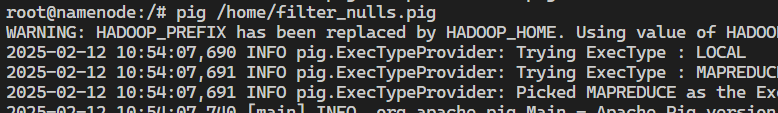


Captura de que los ha reemplazado

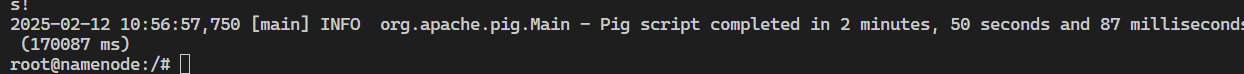
1. Crea un archivo Pig que borre las filas con más de 2 celdas con valores “NULL”.



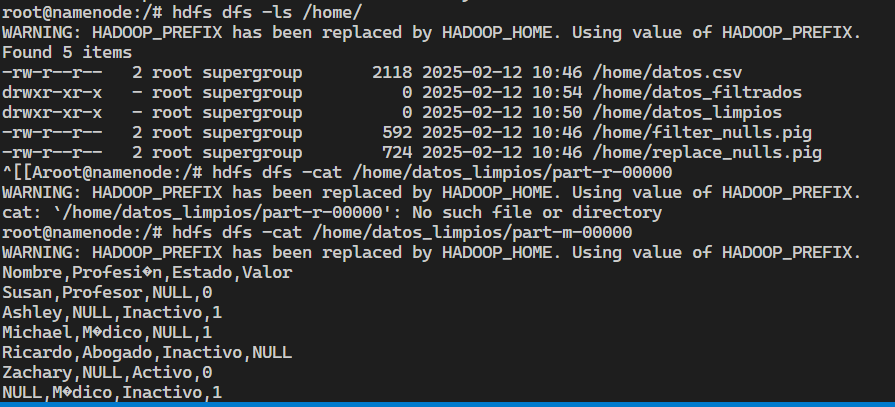
1. Ejecuta el script y muestra los datos en HDFS.



Ejecutamos el .pig



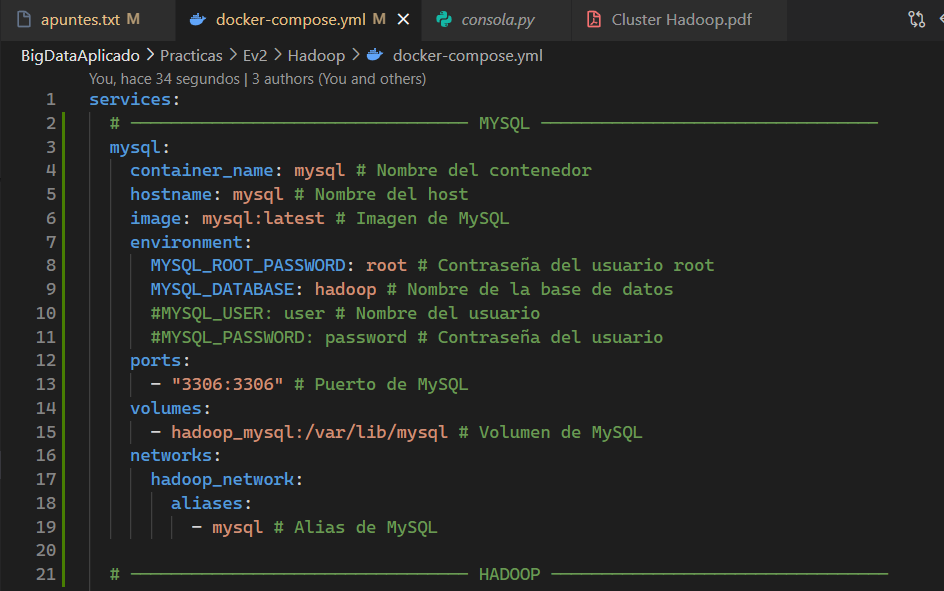
Captura de que se ha ejecutado correctamente

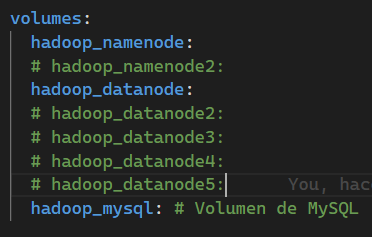


Vemos que funciona correctamente

# Sqoop

1. Añade un contenedor al cluster con una BBDD MySQL. Dentro de la base de datos, crea una tabla llamada empleados con las siguientes columnas: id INT PRIMARY KEY, nombre VARCHAR(50), departamento VARCHAR(50), salario DECIMAL(10,2), fecha\_contratacion DATE.

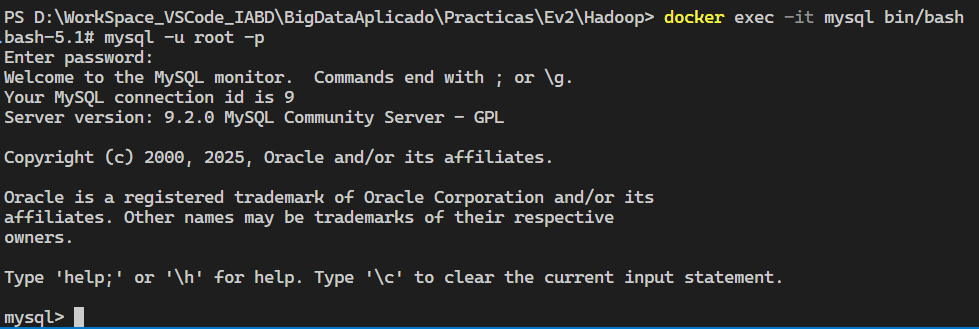




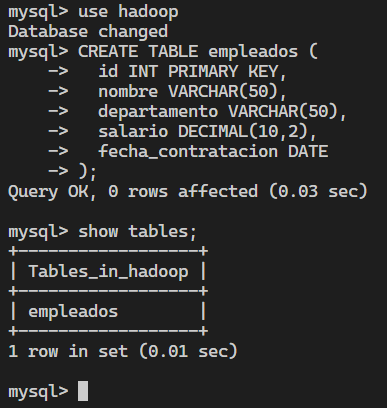
Modificamos el Docker-compose de manera que en *service* añadimos el mysql y su respectivo volumen en *volumes*



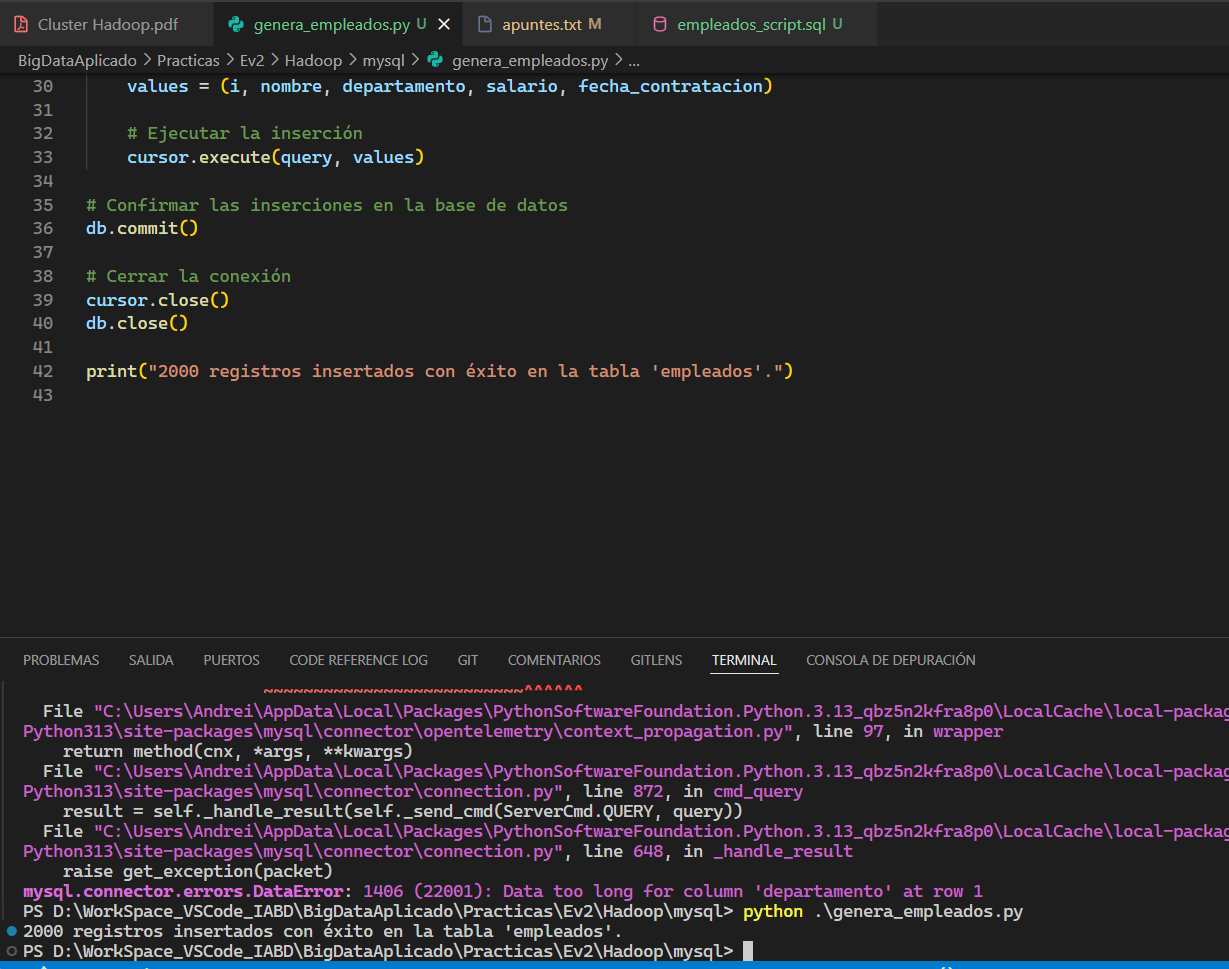
Aquí vemos que se ha creado correctamente.



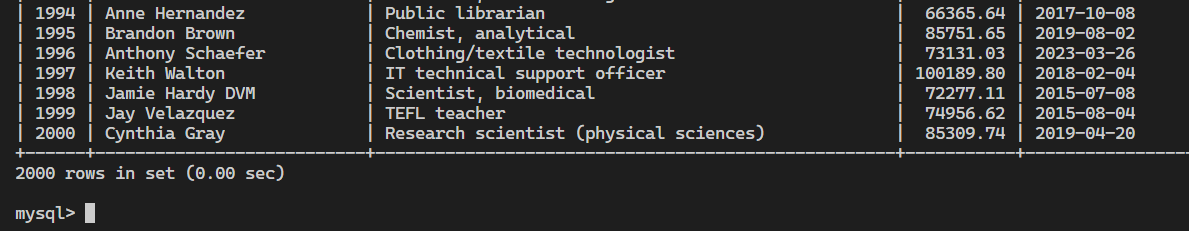
Entramos a la bd (usuario root y password root)



Creamos la tabla. Tambien podemos hacer un .sql para que la cree nada mas iniciarse.

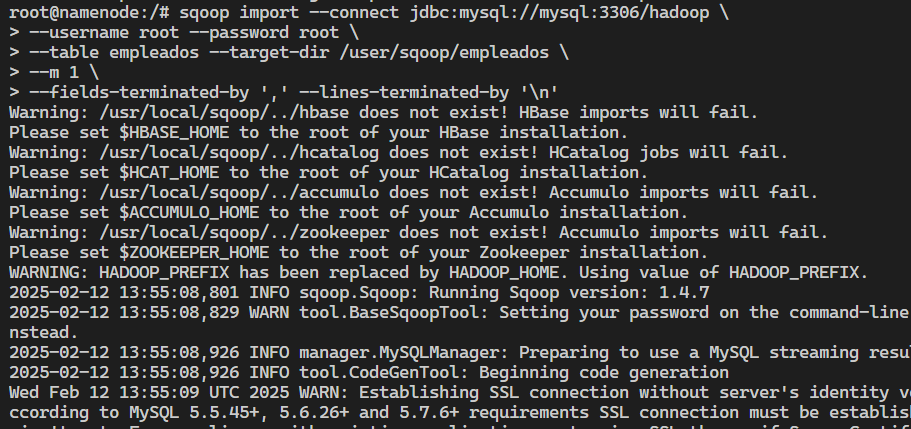
1. Inserta al menos 2000 registros en la tabla empleados.´

Ejecutamos el script para añadir los 2000 registros (El error era que el faker me daba 100 caracteres en vez de 50 para departamento).

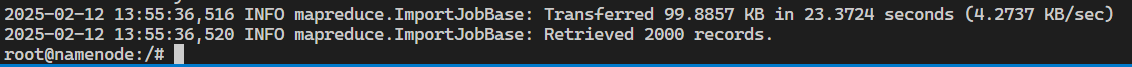


Con select \* from empleados vemos que tiene 2000 lineas.

1. Utiliza Sqoop para importar los datos de la tabla empleados desde la base de datos MySQL hacia HDFS y almacénalos en un directorio llamado /user/sqoop/empleados en HDFS.

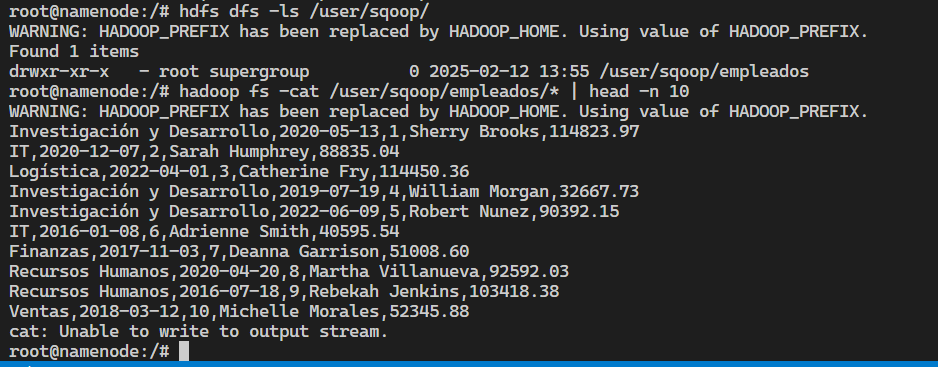


Ejecutamos el comando para importar



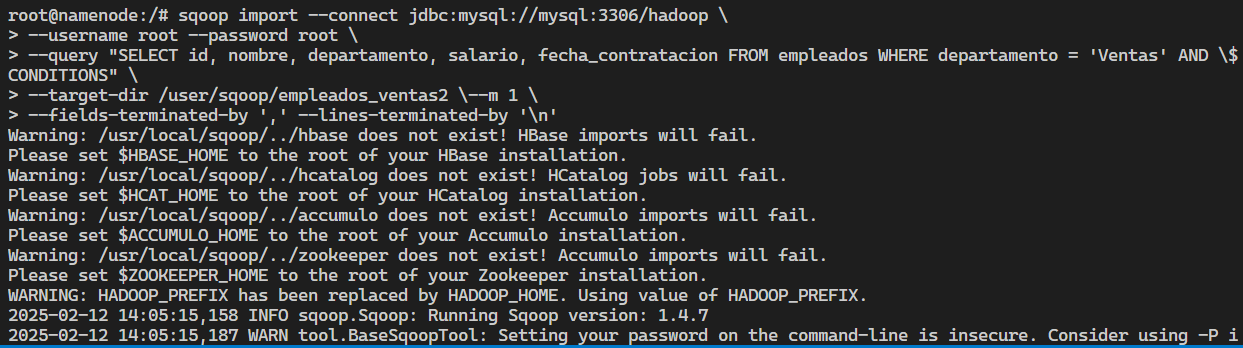
Captura de finalización del comando

1. Comprueba los datos importados en HDFS utilizando los comandos hdfs dfs -ls y hdfs dfs -cat.



Comprobación con -ls y -cat

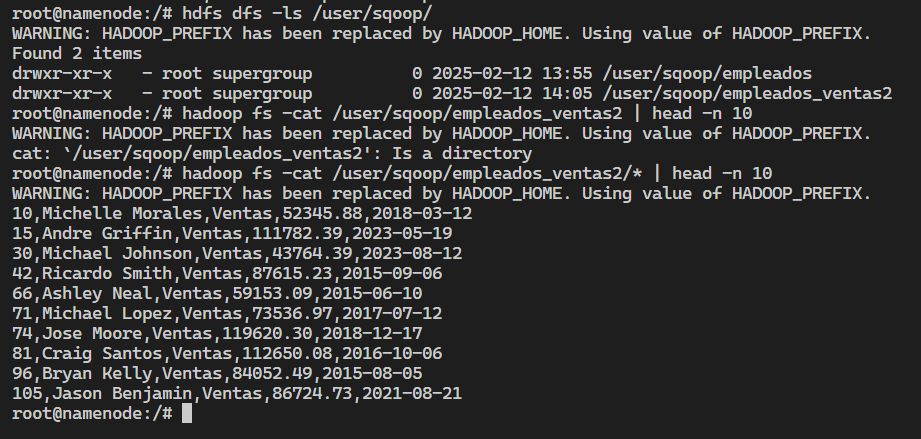
1. Realiza una segunda importación utilizando una consulta SQL que filtre los datos. Por ejemplo: importa solo los empleados del departamento "Ventas".



Ejecutamos el comando



Captura de la finalización



Comprobación con -ls y -cat

1. Automatiza la importación para que se ejecute diariamente mediante un cron job en el contenedor. Configura el cron para que ejecute el comando Sqoop a las 00:00 cada día.

# Flume

1. Crea un script bash que genere automáticamente logs de prueba. Este script debería escribir una línea en un archivo /var/logs/app.log cada 5 segundos. Un ejemplo de línea podría ser: INFO - 2025-01-01 12:00:00 - Usuario accedió al sistema. Pista: Usa un comando como while true; do echo "INFO - $(date '+%Y-%m-%d %H:%M:%S') - Usuario accedió al sistema." >> /var/logs/app.log; sleep 5; done.
2. Configura los componentes Source, Channel y Sink de Flume para recoger los datos del archivo /var/logs/app.log y almacenarlos en un directorio HDFS llamado /user/flume/logs.
3. Monitorea el agente Flume para asegurarte de que está recopilando y transfiriendo datos a HDFS. Usa comandos como hdfs dfs -ls

/user/flume/logs para comprobar los datos almacenados.

# Procesamiento avanzado de datos

1. Haz los cambios necesarios en el Dockerfile y Docker Compose para que funcione Apache Hive..
2. Crea una base de datos llamada bigdata\_practica.
3. Importa el CSV generado en el Task 2 con Pig y crea una tabla externa en Hive llamada usuarios con las siguientes columnas: nombre STRING, profesion STRING, estado STRING, activo INT.
4. Realiza consultas SQL que permitan:
   1. Obtener el número de filas donde alguna columna tenga el valor "NULL".
   2. Agrupar los datos por la columna profesion y mostrar la frecuencia de cada valor.
   3. Añade particiones a la tabla basándote en la columna estado para mejorar el rendimiento de las consultas.

# Apache Spark

1. Haz los cambios necesarios en el Dockerfile y Docker Compose para que se puedan lanzar trabajos Spark.
2. Configura un flujo de datos en Spark Streaming que lea datos en tiempo real desde el directorio /user/flume/logs en HDFS.
3. Modifica el script the bash del apartado de FLUME para que de manera aleatoria añada líneas con valor ERROR- 2025-01-01 12:00:00 o WARN- 2025-01-01 12:00:00.
4. El flujo debe filtrar solo las líneas que contengan el texto "ERROR" y almacenarlas en otro directorio de HDFS llamado /user/spark/errors.
5. Crea un job Spark que analice los datos de logs almacenados en HDFS para calcular:
6. El número total de líneas procesadas.
7. La frecuencia de cada tipo de mensaje (INFO, WARN, ERROR).