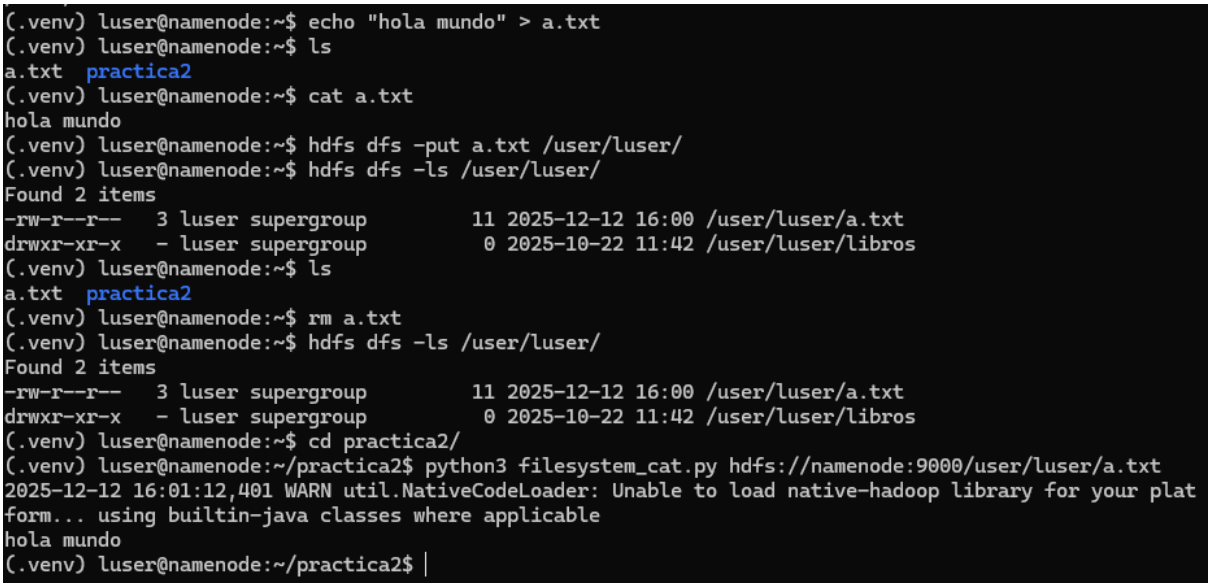
TCDM Práctica 2

José Antonio

- Primera parte: ejecución del código filesystem_cat

Creamos el entorno virtual, descargamos las librerías y ejecutamos filesystem_cat, un script que lee el contenido de un archivo que puede estar en cualquier sistema de ficheros, gracias a pyarrow, y lo muestra por la salida estándar.



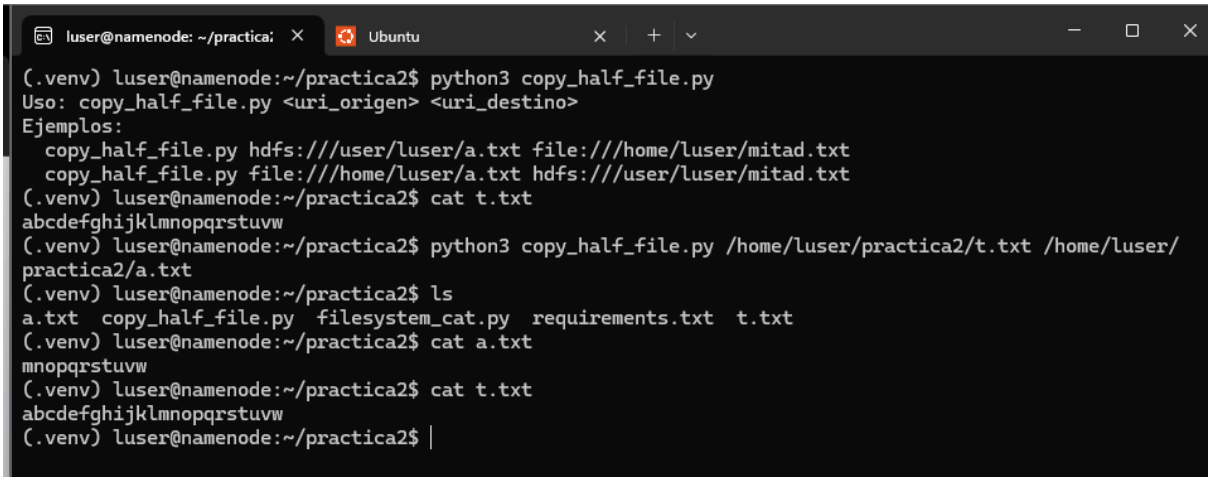
- Segunda parte: código copy_half_file

El script copy_half_file.py es un programa en Python pensado para copiar solo la segunda mitad de un archivo desde un origen hasta un destino, midiendo esa “mitad” por tamaño en bytes.

Para poder trabajar igual con archivos en HDFS o en el sistema local, el script utiliza PyArrow, concretamente el módulo pyarrow.fs (pyarrow filesystem), que como ya comentamos antes ofrece una API unificada de “sistema de archivos”. Así, el usuario puede pasar rutas en formato URI como hdfs://... o file:///..., y el programa se encarga de interpretar cada una, obteniendo el tipo de filesystem correcto y la ruta interna real. Antes de copiar nada, el script consulta los metadatos del archivo de origen mediante FileInfo para comprobar que existe, que realmente es un archivo (no un directorio) y para saber su tamaño exacto.

Con el tamaño ya conocido, calcula el punto medio usando división entera (si el tamaño es impar, redondea hacia abajo). Luego abre el archivo origen para lectura con posibilidad de hacer seek (mover el puntero) y abre el archivo destino como un stream de escritura. A continuación, desplaza el puntero del origen a la posición de la mitad y copia todo lo que queda hasta el final, usando shutil.copyfileobj para hacerlo por bloques, sin cargar el archivo completo en memoria.

Comprobamos que funciona el local:



Comprobamos que funciona en hdfs:

```
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py
Uso: copy_half_file.py <uri_origen> <uri_destino>
Ejemplos:
  copy_half_file.py hdfs:///user/luser/a.txt file:///home/luser/mitad.txt
  copy_half_file.py file:///home/luser/a.txt hdfs:///user/luser/mitad.txt
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvw
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py /home/luser/practica2/t.txt /home/luser/
practica2/a.txt
(.venv) luser@namenode:~/practica2$ ls
a.txt  copy_half_file.py  filesystem_cat.py  requirements.txt  t.txt
(.venv) luser@namenode:~/practica2$ cat a.txt
mnopqrstuvw
(.venv) luser@namenode:~/practica2$ cat t.txt
abcdefghijklmnopqrstuvw
(.venv) luser@namenode:~/practica2$ python3 copy_half_file.py hdfs://namenode:9000/user/luser/a.txt hd
fs://namenode:9000/user/luser/b.txt
2025-12-12 16:58:12,960 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
(.venv) luser@namenode:~/practica2$ hdfs dfs -ls /user/luser/
Found 3 items
-rw-r--r--   3 luser supergroup         11 2025-12-12 16:00 /user/luser/a.txt
-rw-r--r--   3 luser supergroup          6 2025-12-12 16:58 /user/luser/b.txt
drwxr-xr-x   - luser supergroup          0 2025-10-22 11:42 /user/luser/libros
(.venv) luser@namenode:~/practica2$ python3 filesystem_cat.py hdfs://namenode:9000/user/luser/b.txt
2025-12-12 16:58:57,282 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
mundo
(.venv) luser@namenode:~/practica2$
```

- Tercera parte: Probar el comando hdfs dfsadmin

En este apartado nos piden establecer una **cuota de 4** (cuota de nombres) y comprobar cuántos ficheros se pueden copiar en ese directorio. Para ello, primero hay que entender qué son las cuotas en HDFS.

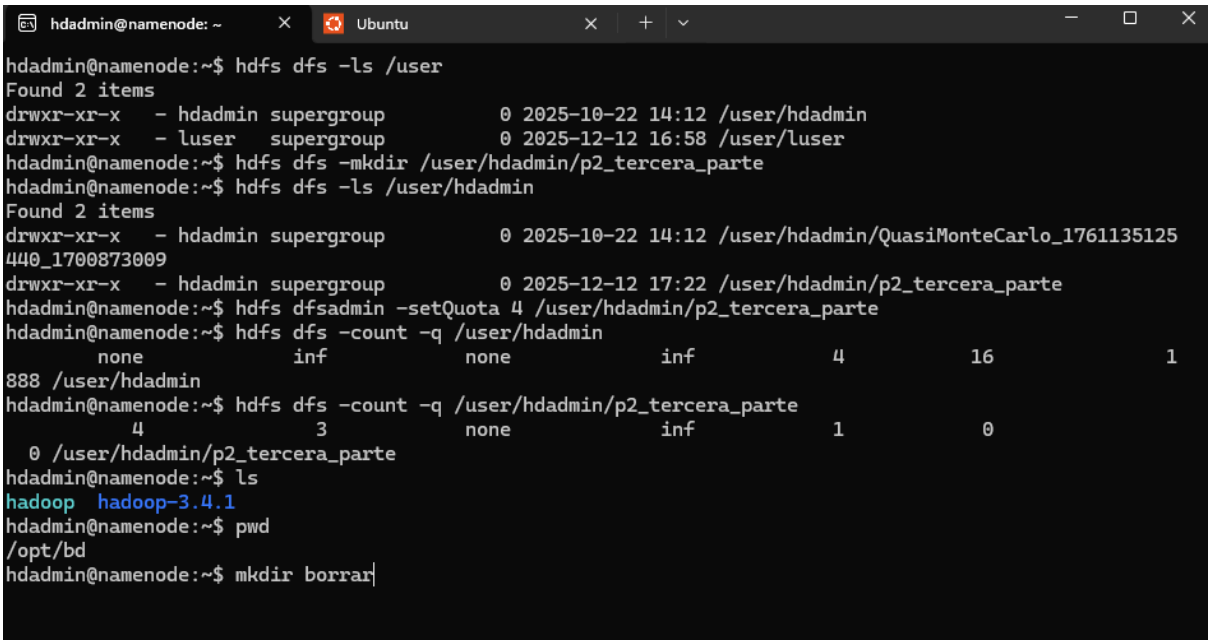
En HDFS, las cuotas son límites que el **NameNode** impone sobre un directorio HDFS (y todo su subárbol) para evitar que un usuario o aplicación utilice demasiados recursos.

Hay dos tipos principales de cuotas, cuota de nombres (namespace quota) y cuota de espacio (space quota).

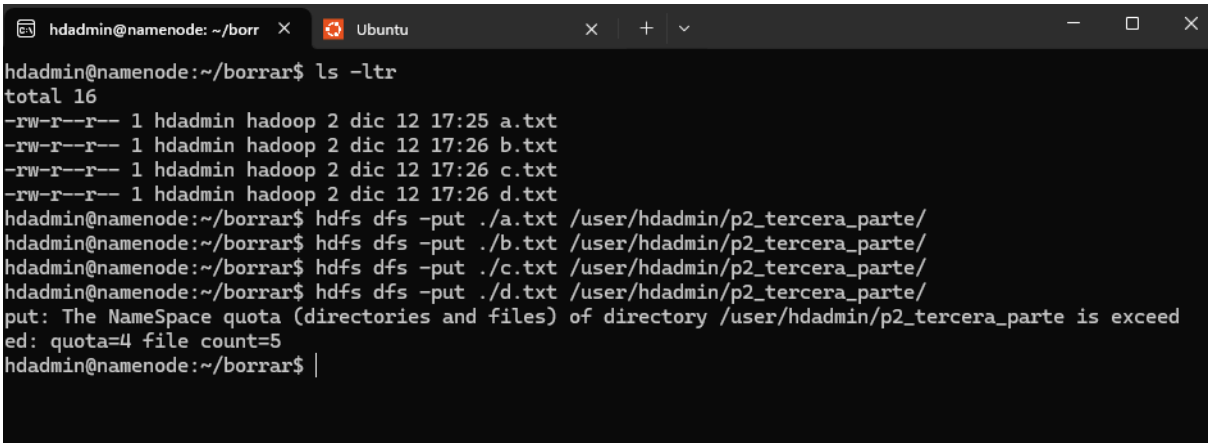
- Cuota de nombres: limita cuántos “objetos” (inodos) puedes tener dentro de ese directorio y sus subdirectorios. En la práctica cada archivo y directorio consume 1. El conteo se aplica al árbol bajo ese directorio y se incluye el propio directorio, por eso al poner cuota en /ruta, el propio directorio ya cuenta como 1.

- Cuota de espacio: limita cuántos bytes pueden ocupar los datos dentro de ese directorio (y subdirectorios), cuenta el espacio replicado. Si un archivo ocupa 100 MB y el factor de replicación es 3, suele contar como ~300 MB de cuota.

Primero creamos un directorio para el ejercicio y le ponemos la cuota de 4:



Intentamos copiar 4 ficheros (a.txt, b.txt, c.txt, d.txt) en el directorio con cuota y vemos que falla al intentar copiar el cuarto (d.txt). Es decir, nos deja copiar 3 ficheros con una cuota de 4. Este comportamiento es correcto: al establecer una cuota de nombres, el directorio también consume 1 unidad de cuota, por lo que quedan 3 unidades disponibles para ficheros (u otros inodos) dentro de ese subárbol.

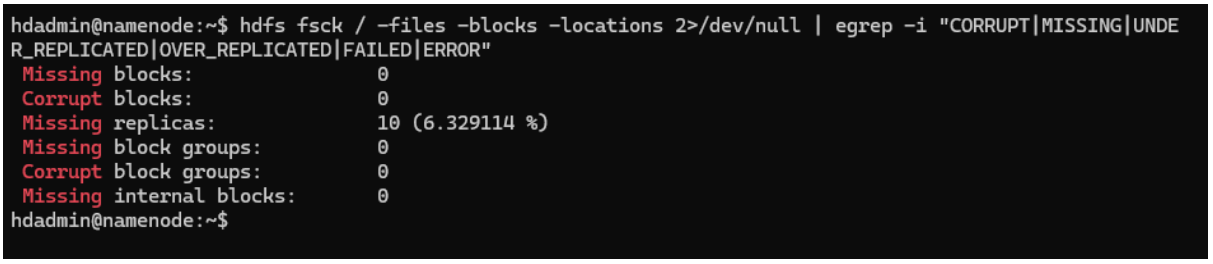


- Cuarta parte: Probar el comando hdfs dfsadmin

Paso 1)

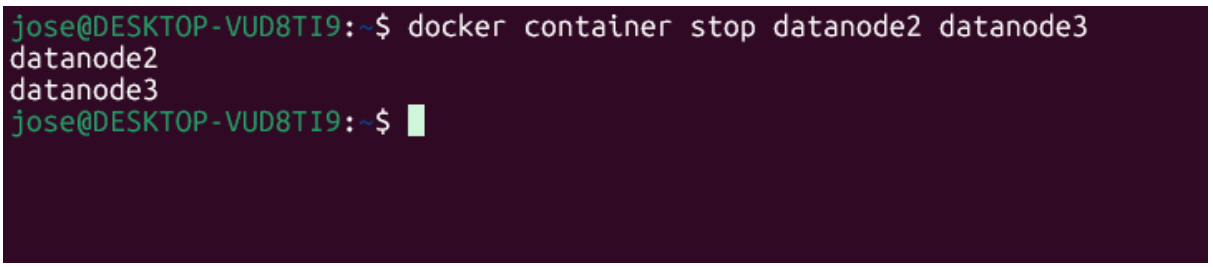
hdfs fsck / es un comando hdfs que nos permite hacer un “file system check”. Le pide al NameNode un reporte del estado de archivos/bloques bajo la ruta que le especifiquemos.

Ejecutamos *hdfs fsck* para comprobar si hay errores. Recorre el árbol desde / y reporta problemas (bloques perdidos, replicación incorrecta, etc.). Con egrep vemos solo los errores.

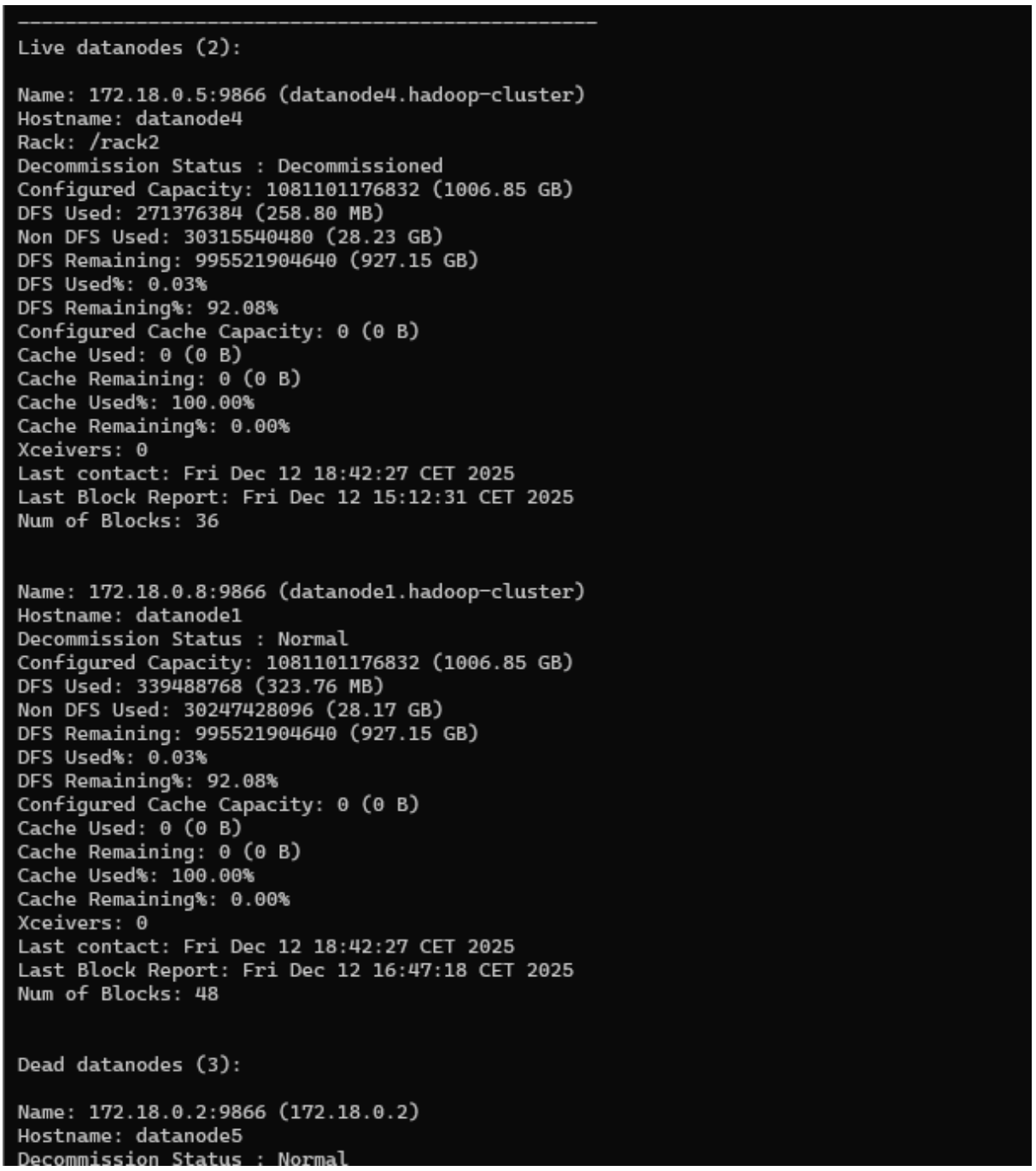


Paso 2)

Detenemos los datanodes



y esperamos. Comprobamos luego de 10 minutos que solo nos quedan 2 datanodes vivos:



Paso 3)

Volvemos a hacer un diagnóstico del sistema (esta vez sin el grep) y nos centramos en la parte de replicación:

```
Replicated Blocks:
 Total size:    336501550 B
 Total files:   44
 Total blocks (validated):      48 (avg. block size 7010448 B)
 Minimally replicated blocks:   48 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       48 (100.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     1.75
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              74 (46.83544 %)
 DecommissionedReplicas:        36
 Blocks queued for replication: 0
```

Vemos que ahora nos aparecen 48 under-replicated blocks, el 100% de los bloques. Solo nos quedan dos datanodes vivos y nuestro factor de replicación es de 3. Es por eso que nos sale este resultado.

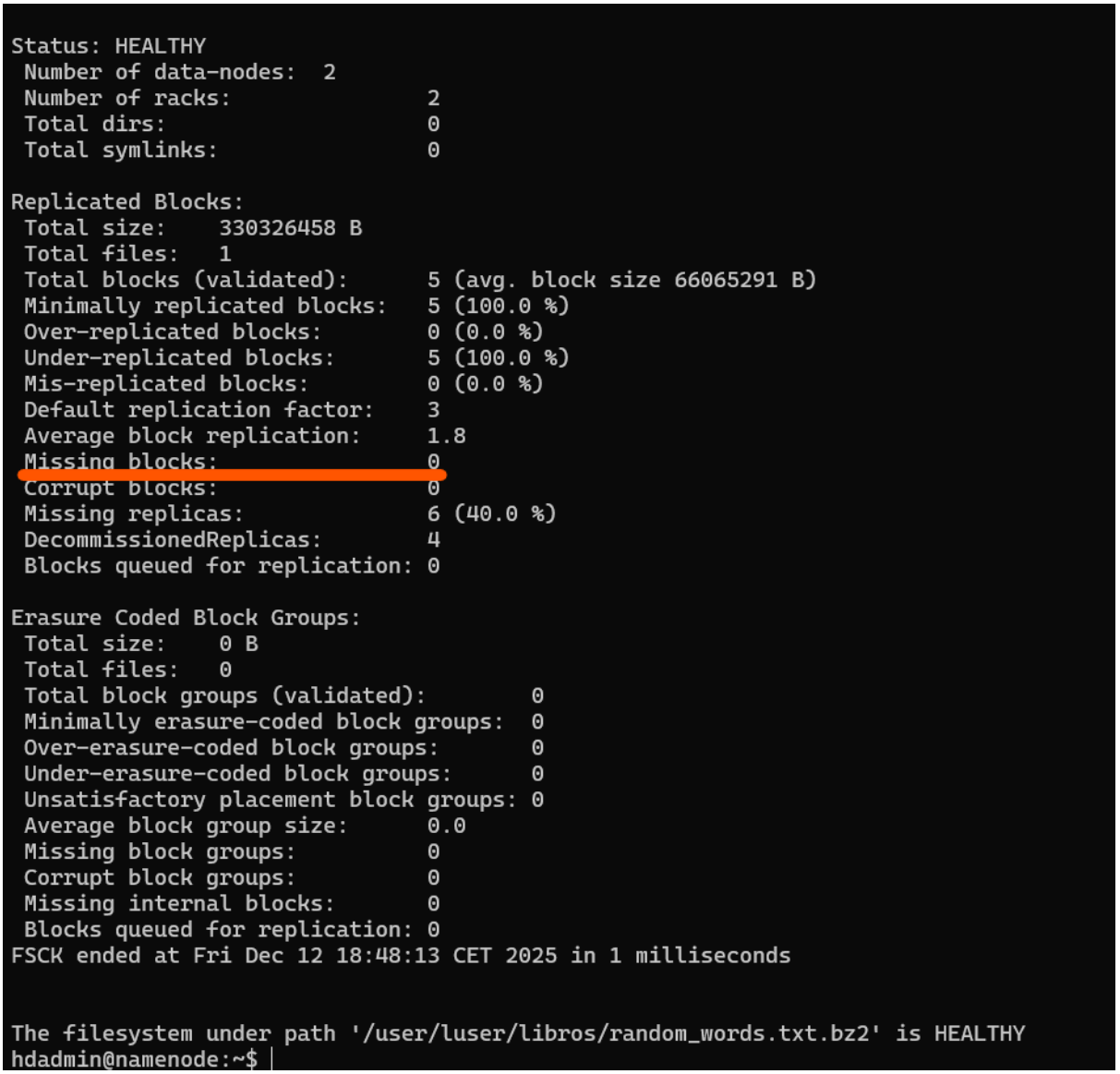
Paso 4)

Si hacemos el *get* vemos que sigue funcionando:

```
hdadmin@namenode:~$ hdfs dfs -get /user/luser/libros/random_words.txt.bz2
hdadmin@namenode:~$ ls
hadoop  hadoop-3.4.1  random_words.txt.bz2
```

Esto es normal, mientras cada bloque del fichero tenga al menos una réplica disponible en alguno de los dos datanodes que quedan vivos. El get fallaría en caso de Missing blocks (no queda ninguna réplica de ese bloque en ningún DN vivo).

Si comprobamos el fichero con *hdfs fsck* vemos que obviamente esta under-replicated, por lo que explicamos en el apartado anterior, pero Missing blocks es igual a 0, por lo que el get funciona perfectamente.



Paso 5)

Añadimos el datanode 6 y ejecutamos el diagnostico de nuevo:

```
Replicated Blocks:
 Total size:    336501550 B
 Total files:   44
 Total blocks (validated):      48 (avg. block size 7010448 B)
 Minimally replicated blocks:   48 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       23 (47.916668 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     2.4583333
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              40 (25.316456 %)
 DecommissionedReplicas:        36
 Blocks queued for replication: 0
```

Vemos que ha funcionado correctamente porque han bajado los under-replicated blocks.