**Máster Universitario en Big Data y Ciencia de Datos**

**ASIGNATURA: 03MBID *Procesamiento de datos masivos***

*Instalación Spark en Docker (Linux)*

Alumno: **Bru Montes, Israel**

Edición **Octubre 2024 – Grupo A** a 24/12/2024

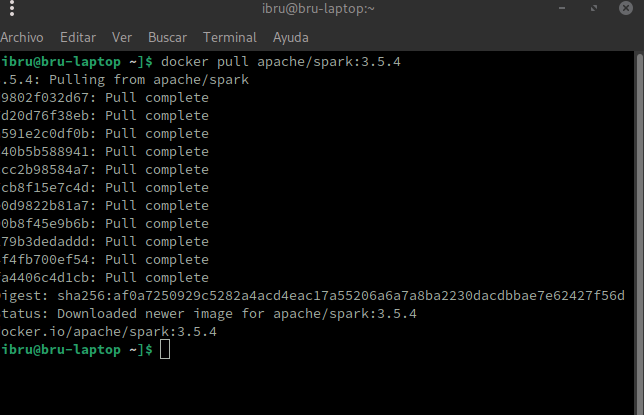
# Introducción.

En el siguiente documento vamos a documentar como hemos instalado spark y docker dentro de un Sistema Operativo Linux (Manjaro).

Vamos a partir de la instalación y configuración del servidor que ya tenemos creado de hadoop.

Nos bajamos la última versión que existe en <https://hub.docker.com/> de la imagen que ofrece apache:

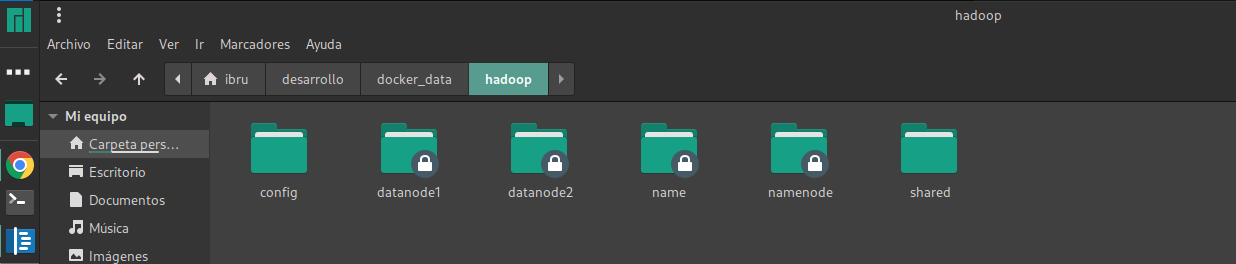
[ibru@bru-laptop ~]$ docker pull apache/spark:3.5.4



[ibru@bru-laptop desarrollo]$ mkdir docker\_data

[ibru@bru-laptop desarrollo]$ mkdir docker\_data\hadoop

[ibru@bru-laptop desarrollo]$ mkdir docker\_data\hadoop\shared



Definimos la configuración tanto del docker (docker-compose-spark.yml) como su configuración por defecto. Detallamos aquí el archivo, aunque lo adjunto en la documentación:

Config:

*HADOOP\_HOME=/opt/hadoop*

*CORE-SITE.XML\_fs.default.name=hdfs://namenode*

*CORE-SITE.XML\_fs.defaultFS=hdfs://namenode*

*HDFS-SITE.XML\_dfs.namenode.rpc-address=namenode:8020*

*HDFS-SITE.XML\_dfs.replication=3*

*HDFS-SITE.XML\_dfs.namenode.data.dir=file:///root/hdfs/datanode*

*HDFS-SITE.XML\_dfs.datanode1.data.dir=file:///root/hdfs/datanode*

*HDFS-SITE.XML\_dfs.datanode2.data.dir=file:///root/hdfs/datanode*

*MAPRED-SITE.XML\_mapreduce.framework.name=yarn*

*MAPRED-SITE.XML\_yarn.app.mapreduce.am.env=HADOOP\_MAPRED\_HOME=$HADOOP\_HOME*

*MAPRED-SITE.XML\_mapreduce.map.env=HADOOP\_MAPRED\_HOME=$HADOOP\_HOME*

*MAPRED-SITE.XML\_mapreduce.reduce.env=HADOOP\_MAPRED\_HOME=$HADOOP\_HOME*

*YARN-SITE.XML\_yarn.resourcemanager.hostname=resourcemanager*

*YARN-SITE.XML\_yarn.nodemanager.pmem-check-enabled=false*

*YARN-SITE.XML\_yarn.nodemanager.delete.debug-delay-sec=600*

*YARN-SITE.XML\_yarn.nodemanager.vmem-check-enabled=false*

*YARN-SITE.XML\_yarn.nodemanager.aux-services=mapreduce\_shuffle*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.maximum-applications=10000*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.maximum-am-resource-percent=0.1*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.resource-calculator=org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.queues=default*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.capacity=100*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.user-limit-factor=1*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.maximum-capacity=100*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.state=RUNNING*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.acl\_submit\_applications=\**

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.root.default.acl\_administer\_queue=\**

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.node-locality-delay=40*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.queue-mappings=*

*CAPACITY-SCHEDULER.XML\_yarn.scheduler.capacity.queue-mappings-override.enable=false*

*LOG4J.PROPERTIES\_log4j.rootLogger=INFO, stdout*

*LOG4J.PROPERTIES\_log4j.appender.stdout=org.apache.log4j.ConsoleAppender*

*LOG4J.PROPERTIES\_log4j.appender.stdout.layout=org.apache.log4j.PatternLayout*

*LOG4J.PROPERTIES\_log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n*

*docker-compose-spark.yml:*

services:

namenode:

image: apache/hadoop:3.4.1

hostname: namenode

user: root

command: ["hdfs", "namenode"]

ports:

- 9870:9870

environment:

ENSURE\_NAMENODE\_DIR: "/tmp/hadoop-root/dfs/name"

volumes:

- ./hadoop/shared:/opt/hadoop/input\_files

env\_file:

- ./config

networks:

- hdfs\_network

datanode\_1:

image: apache/hadoop:3.4.1

command: ["hdfs", "datanode"]

env\_file:

- ./config

networks:

- hdfs\_network

datanode\_2:

image: apache/hadoop:3.4.1

command: ["hdfs", "datanode"]

env\_file:

- ./config

networks:

- hdfs\_network

resourcemanager:

image: apache/hadoop:3.4.1

command: ["yarn", "resourcemanager"]

ports:

- 8088:8088

env\_file:

- ./config

networks:

- hdfs\_network

nodemanager:

image: apache/hadoop:3.4.1

command: ["yarn", "nodemanager"]

environment:

ENSURE\_NODEMANAGER\_DIR: "tmp/hadoop-hadoop/nm-local-dir/"

networks:

- hdfs\_network

env\_file:

- ./config

volumes:

datanode1:

datanode2:

namenode:

networks:

hdfs\_network:

# Specify driver options

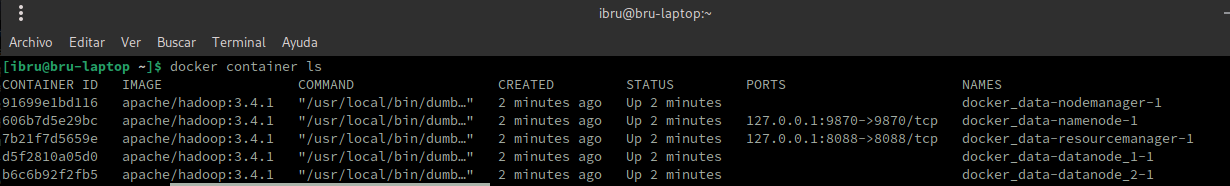
driver: bridge

driver\_opts:

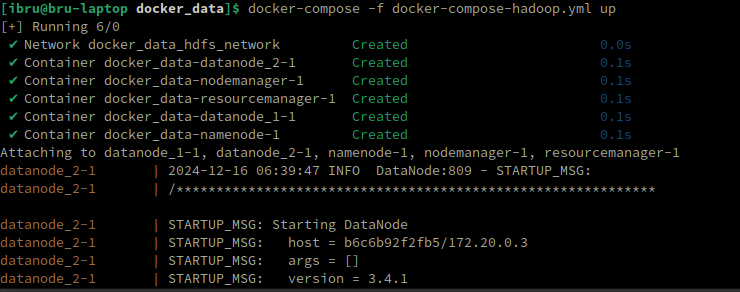
*com.docker.network.bridge.host\_binding\_ipv4: "127.0.0.1"*

Nos movemos hasta la carpeta que hemos creado para alojar tanto los archivos de configuración como la carpeta “shared” para hacer el intercambio de archivos entre el contenedor principal y nuestro sistema operativo. A continuación iniciamos docker con el siguiente comando:

[ibru@bru-laptop ~]$ docker-compose -f docker-compose-hadoop.yml up

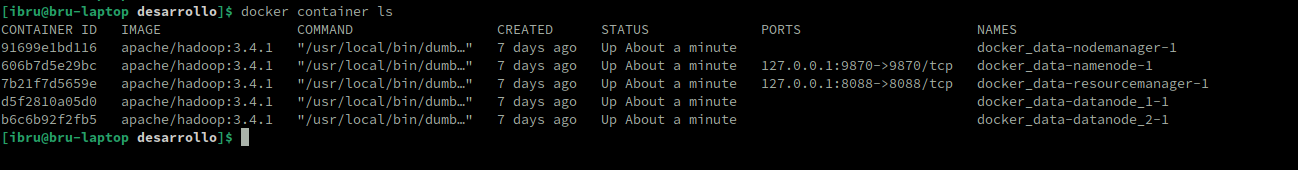


Al arrancar nos muestra la siguiente información:



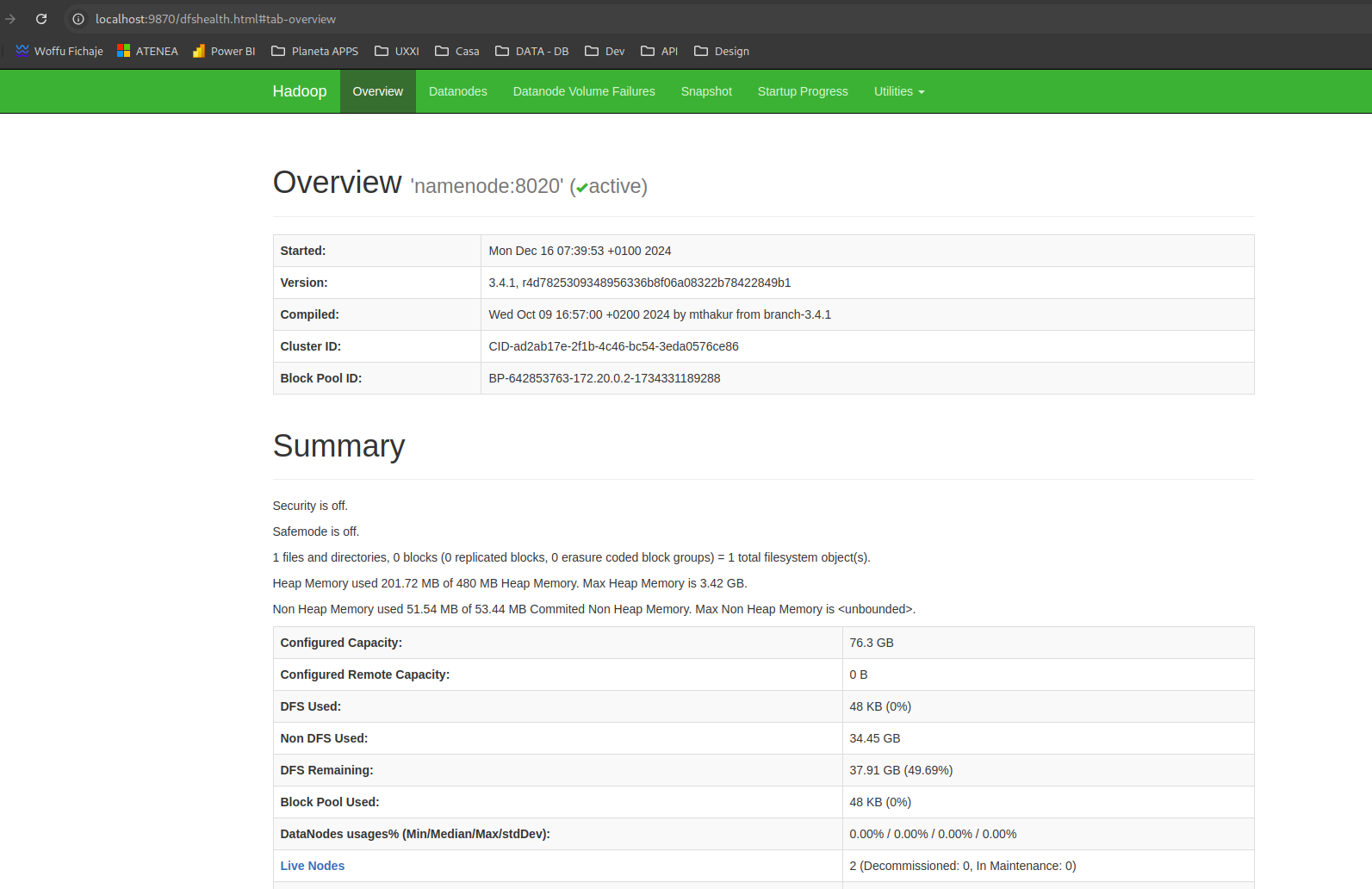
Abrimos un terminal nuevo, siempre y cuando tengamos bloqueada la terminal actual y lanzamos el siguiente comando para ver los contenedores y el id que tienen asignado.

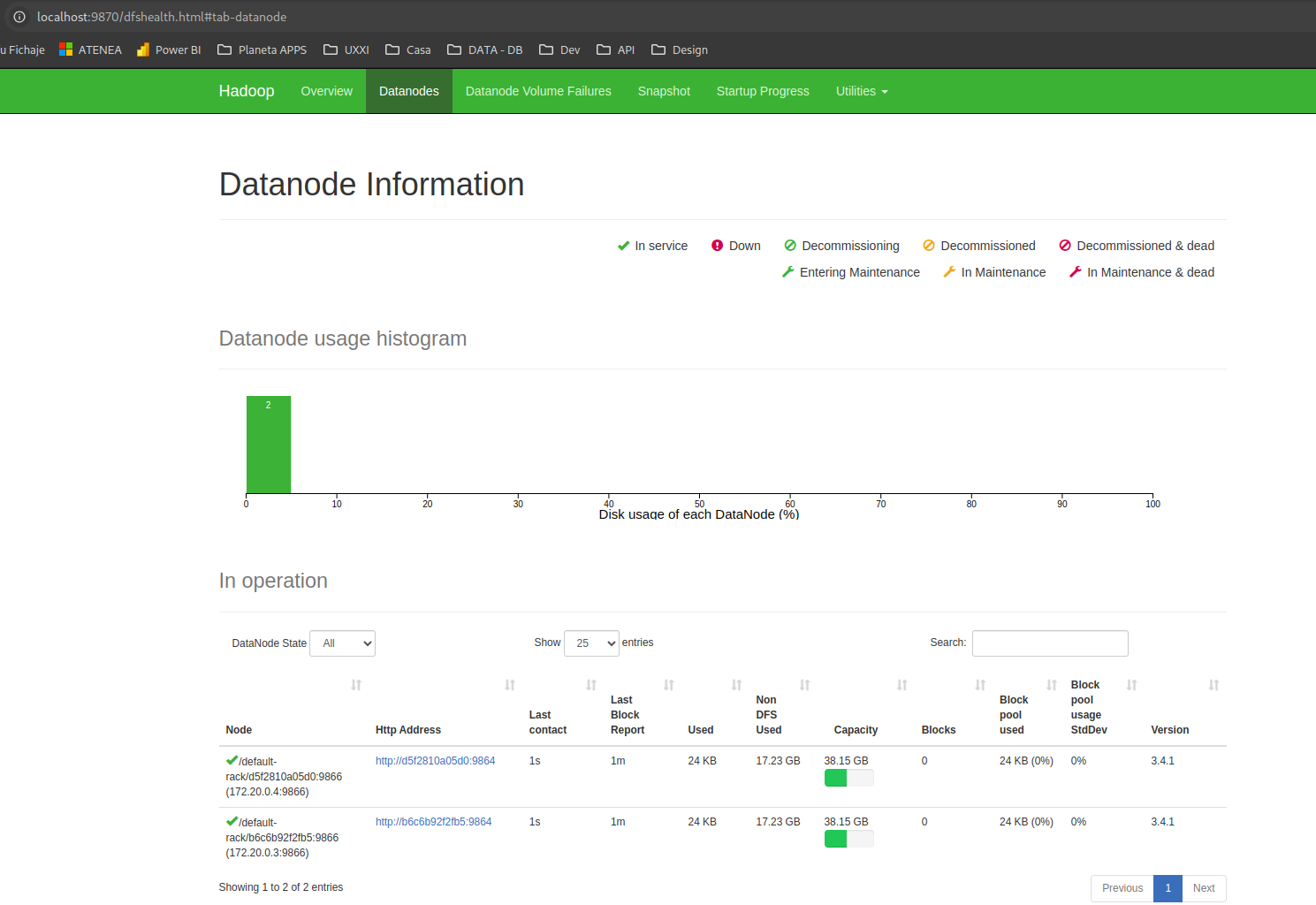
bru@bru-laptop ~]$ docker container ls

Y con el siguiente comando nos permite conectarnos dentro del contenedor que tiene

[ibru@bru-laptop desarrollo]$ docker exec -it 606b7d5e29bc bash

También podemos comprobar la información del sistema de hadoop accediendo a la siguiente url: http://localhost:9070

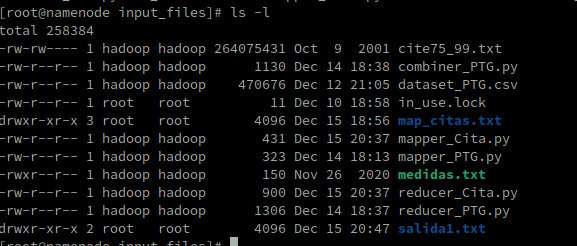




Creamos en el sistema hdfs, tanto la carpeta user como la carpeta root, y subimos un archivo para poder probar que funciona todo el sistema.

[root@namenode input\_files]# hadoop fs -mkdir /user

[root@namenode input\_files]# hadoop fs -mkdir /user/root



Nos movemos dentro de la maquina “namenode” a la carpeta que hemos creado del intercambio de ficheros con la máquina anfitriona:

[root@namenode hadoop]# cd input\_files/

Subimos el archivo a hdfs para poder ejecutar con posterioridad.

[root@namenode input\_files]# hdfs dfs -put medidas.txt

root@namenode input\_files]# hadoop jar ../share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -files mapper\_PTG.py,reducer\_PTG.py,combiner\_PTG.py -mapper mapper\_PTG.py -combiner combiner\_PTG.py -reducer reducer\_PTG.py -input dataset\_PTG.csv -output ptg\_out.txt