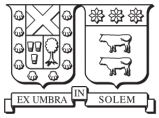

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Trabajo Práctico 1

INF356 - 2025-1 - 200

Computación Distribuida para Big Data

4 de mayo de 2025 - v1.0

Miguel Soto | 20430363-0
miguel.sotod@sansano.usm.cl

Índice

Índice de figuras

Índice de fragmentos de codigo

1. Despligue del cluster

1.1. Implementación del tutorial

Para la creacion del Cluster en Amazon Web Services, fue necesario seguir los pasos indicados en el tutorial entregado en el Slide T01 - Hadoop Cluster - v1.0 entregado en Aula. El instructivo constaba de los siguientes pasos:

- Crear la cuenta de AWS
- Preparar en entorno del Cluster
- Configurar la maquina Master
- Configurar todos los Workers
- Inicializar el Cluster
- Ejecutar el script de Map Reduce con Hadoop

Crear la cuenta de AWS

Para la creacion de la cuenta de AWS simplemente segui las indicaciones en el correo de invitacion indicado por el profesor. Posterior a eso, me fui directamente a la seccion de Modules, en donde estaba el Modulo de Launch AWS Academy Learner Lab, que sirve para entrar a la Console Home de AWS.

Preparar en entorno del Cluster

Una vez en Console Home, creamos una instancia de tipo EC2, para la cual creamos los Key Pairs, el Security Group y la instancia Master.

Para los Key Pairs, creamos una llave de tipo RSA con formato de archivo PEM.

Para el Security Group, creamos tres reglas, una Inbound Rule de tipo SSH con tipo de fuente Anywhere-IPv4 para entrar por SSH, una Inbound Rule de tipo Custom TCP con rango de puerto 8088 y tipo de fuente Anywhere-IPv4 para la consola web de Hadoop, una Inbound Rule de tipo Custom TCP con rango de puerto 9870 y tipo de fuente Anywhere-IPv4 para la consola web de HDFS y finalmente, una Inbound Rule de tipo Custom TCP con rango de puerto de 0 a 65535 y tipo de fuente Anywhere-IPv4.

Para la instancia Master creamos un nodo con Ubuntu 24.04 de tipo t2.micro y con 16gb de almacenamiento. Esta instancia usara la Key y Security Group recien creados (al igual que las instancias Worker mas adelante). Una vez configurados los parametros, lanzamos la instancia y nos conectamos por SSH:

```
ssh ubuntu@34.207.118.42 -i .\Documents\llave.pem
```

Código 1: Conectarse por primera vez al Master usando la llave PEM con la IP publica

Configurar la maquina Master

El primer paso es logicamente actualizar los paquetes del sistema, para esto es necesario ejecutar el siguiente comando:

```
sudo apt update && sudo apt upgrade -y
```

Código 2: Actualizar repositorios locales y descargar actualizaciones

Una vez hecho esto, instalamos el paquete correspondiente a Java / OpenJDK 11:

```
sudo apt-get install openjdk-11-jdk -y
```

Código 3: Descargar Java / OpenJDK 11 usando APT

Una vez que hayamos resuelto todos los paquetes necesarios, tenemos que crear una llave publica con el siguiente comando:

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Código 4: Crear una llave publica en formato RSA y autorizarla

Esta llave publica la guardaremos en la carpeta de Authorized Keys para SSH, lo cual permitira al Master conectarse a si mismo por LocalHost.

Descargamos Hadoop 3.3.6 desde la pagina oficial usando el comando WGET, el cual hace una request a la pagina para descargar el archivo comprimido en formato TAR GZ, el cual tendremos que descomprimir y modificar para indicarle parametros especiales. Ejecutamos entonces los siguientes comandos:

```
wget https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.6/hadoop-3.3.6-src.tar.
    ↪ gz
tar -xvz hadoop-3.3.6.tar.gz
```

Código 5: Descargar y descomprimir Hadoop en el Master

Esto creara una carpeta llamada hadoop-3.3.6 en el directorio Home del Master, esta carpeta albergara los binarios yu scripts ejecutables para desplegar nuestro Distributed File System (DFS). Para poder ejecutar los comandos en dicha carpeta de manera directa, modificaremos las Variables de Entorno de la maquina Master, de esta manera podremos ejecutar los comandos en cualquier directorio. Dichas Variables tambien contienen rutas para la ejecucion de OpenJDK y la ubicacion de nuestra carpeta Hadoop, las cuales debemos obedecer para que funcione correctamente nuestro cluster. Las variables de entorno en Linux se guardan en el directorio /etc/environment y solo puede modificarse por el Super Usuario (Root). Personalmente, a mi me gusta usar Vim, pero para aquellas personas con gustos inferiores existen alternativas:

```
sudo vim /etc/environment
```

Código 6: Editar las Variables de Entorno usando el usuario Root y el editor de texto Vim

Las Variables debiesen quedar asi:

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:..."
JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
HADOOP_HOME="/home/ubuntu/hadoop-3.3.6"
HADOOP_COMMON_HOME="/home/ubuntu/hadoop-3.3.6"
```

Código 7: Variables de Entorno que usaremos para el cluster

Lo ultimo es modificar los archivos de configuracion de Hadoop en base a lo indicado por el profesor. Estos archivos luego seran enviados a los Workers cuando comprimamos la carpeta hadoop-3.3.6.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://172.31.30.146</value>
    </property>
</configuration>
```

Código 8: hadoop-3.3.6/etc/hadoop/core-site.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>dfs.block.size</name>
        <value>33554432</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
        <value>file:///home/ubuntu/dfs/name</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>file:///home/ubuntu/dfs/data</value>
    </property>
    <property>
        <name>dfs.namenode.heartbeat.recheck-interval</name>
        <value>10000</value>
    </property>
</configuration>
```

Código 9: hadoop-3.3.6/etc/hadoop/hdfs-site.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
        <name>mapreduce.map.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx1024m</value>
    </property>
    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>1024</value>
    </property>
    <property>
        <name>mapreduce.reduce.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx1024m</value>
    </property>
    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>1024</value>
    </property>
    <property>
        <name>mapreduce.job.maps</name>
        <value>4</value>
    </property>
    <property>
        <name>mapreduce.job.running.map.limit</name>
        <value>4</value>
    </property>
    <property>
        <name>mapreduce.job.reduces</name>
        <value>4</value>
    </property>
    <property>
        <name>mapreduce.job.running.reduce.limit</name>
        <value>4</value>
    </property>
</configuration>
```

Código 10: hadoop-3.3.6/etc/hadoop/mapred-site.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>172.31.30.146</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>1024</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>1024</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.command-opts</name>
        <value>-Xmx1024m</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.resource.cpu-vcores</name>
        <value>1</value>
    </property>
</configuration>
```

Código 11: hadoop-3.3.6/etc/hadoop/yarn-site.xml

Configurar todos los Workers

Los Workers tambien seran maquinas con Ubuntu 24.04 y de tipo t2.micro, usaran las mismas Key Pairs y el mismo Security Group. Lo unico que cambiara entre ellos y el Master sera el almacenamiento, las cuales constaran solo de 8gb en total. Una vez creados los Workers desde el interfaz de EC2 de AWS, tendremos que obtener sus IP's publicas y privadas. La primera para conectarnos por primera vez, la segunda para establecer una conexion permanente. Anotamos las IP's privadas de la siguiente manera:

```
172.31.30.0
172.31.17.157
172.31.24.51
172.31.27.79
```

Código 12: IP's privadas de los Workers en la carpeta Home del Master

Luego de tener esto, borramos el archivo comprimido de Hadoop que descargamos inicialmente y comprimimos el directorio que modificamos en el paso anterior para luego enviarselo a los Workers. Estos luego descomprimiran el archivo de manera local para asi no abusar de nuestro ancho de banda limitado:

```
tar czf hadoop-3.3.6.tar.gz hadoop-3.3.6
```

Código 13: Borrar el archivo hadoop-3.3.6.tar.gz y comprimimos el directorio hadoop-3.3.6

Y como la tarea de configurar cada Worker manualmente es tedioso, usamos un script de Bash que realiza el trabajo por nosotros. El cual itera los siguientes pasos:

Para cada una de las instancias Worker proceder de la siguiente forma:

1. Instalar la llave pública de la máquina Master en el Worker
2. Crear una llave para el Worker
3. Instalar la llave del Worker en el Master
4. Instalar la llave del Worker en el Worker
5. Actualizar el sistema operativo en el Worker
6. Instalar Java en el Worker (openjdk-11-jdk)
7. Editar el archivo /etc/environment de la misma forma que para Master
8. Reiniciar el worker
9. Transferir el archivo de despliegue de Hadoop de la máquina Master al Worker
10. Expandir el archivo de despliegue en la máquina Worker

El script utilizado quedo de la siguiente formula:

```
#!/bin/bash

# === CONFIGURACIÓN ===

# IPs publicas de los Workers y el Master
WORKERS=("172.31.30.0" "172.31.17.157" "172.31.24.51" "172.31.27.79")
MASTER="172.31.30.146"

# User de el Master y los Workers
USER="ubuntu"
HOME="/home/$USER"

# Ubicacion de la llave privada
KEY="$HOME/.ssh/llave.pem"

# Variables de Entorno
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games..."
JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
HADOOP_HOME="/home/ubuntu/hadoop-3.3.6"
HADOOP_COMMON_HOME="/home/ubuntu/hadoop-3.3.6"

# Path del archivo comprimido de Hadoop
HADOOP_ARCHIVE="/home/ubuntu/hadoop-3.3.6.tar.gz"

# Ubicacion de las Variables de Entorno
ENV_VARS="/etc/environment"

for WORKER in "${WORKERS[@]}"; do
    echo ">>> Configurando al Worker de IP $WORKER <<<"

    echo ""
    echo "1) Instalando llave publica del Master en el Worker"
        MASTER_KEY="$(cat ~/.ssh/id_rsa.pub)"
    echo " > ssh $USER@$WORKER -i $KEY echo $MASTER_KEY >> ~/.ssh/authorized_keys"
    ssh $USER@$WORKER -i $KEY "echo $MASTER_KEY >> ~/.ssh/authorized_keys"

    echo ""
    echo "2) Creando llave publica para el Worker"
    echo " > ssh $USER@$WORKER -i $KEY ssh-keygen -t rsa"
    ssh $USER@$WORKER -i $KEY "ssh-keygen -t rsa"
    echo ""

    echo ""
    echo "3) Instalando llave del Worker en el Master"
    echo " > ssh $USER@$WORKER -i $KEY cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"
    ssh $USER@$WORKER -i $KEY "cat ~/.ssh/id_rsa.pub" >> ~/.ssh/authorized_keys
    echo ""

    echo ""
    echo "4) Instalando llave del Worker en el mismo Worker"
    echo " > ssh $USER@$WORKER cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"
    ssh $USER@$WORKER "cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"
    echo ""

    echo ""
    echo "5) Actualizando sistema operativo en el Worker"
    echo " > ssh $USER@$WORKER sudo apt update -qq && sudo apt upgrade -y -qq"
    ssh $USER@$WORKER "sudo apt update -qq && sudo apt upgrade -y -qq"
    echo ""
...
```

Código 14: Script para configurar todos los Workers desde el Master

```
...
    echo ""
    echo "6) Instalando Java en el Worker"
    echo "> ssh $USER@$WORKER sudo apt-get install -y -qq openjdk-11-jdk"
    ssh $USER@$WORKER "sudo apt-get install -y -qq openjdk-11-jdk"
    echo ""

    echo ""
    # Esta seccion me dio problemas asi que este paso lo hice manual al final
    echo "7) Copiando las Variables de Entorno en el Worker"
        echo "> cat $HOME/environment"
        cat $HOME/environment
    echo "> scp $HOME/environment $USER@$WORKER:$HOME/"
    scp $HOME/environment $USER@$WORKER:$HOME/
    echo "ssh $USER@$WORKER cat $HOME/enviroment"
    ssh $USER@$WORKER "cat $HOME/enviroment"
    echo "> ssh $USER@$WORKER sudo cp $HOME/enviroment $ENV_VARS"
    ssh $USER@$WORKER "sudo cp $HOME/enviroment $ENV_VARS"
    echo "> ssh $USER@$WORKER cat $ENV_VARS"
    ssh $USER@$WORKER "cat $ENV_VARS"

    echo ""

    echo ""
    echo "8) Copiando archivo de despliegue Hadoop al Worker..."
    echo "ssh $USER@$WORKER rm $HADOOP_ARCHIVE && ls -la"
    ssh $USER@$WORKER "rm $HADOOP_ARCHIVE && ls -la"
    echo "> scp $HADOOP_ARCHIVE $USER@$WORKER:$HOME/"
    scp $HADOOP_ARCHIVE $USER@$WORKER:$HOME/
    echo "> ssh $USER@$WORKER ls -la"
    ssh $USER@$WORKER "ls -la"
    echo ""

    echo ""
    echo "9) Descomprimiendo el archivo de despliegue en el Worker..."
    echo "> ssh $USER@$WORKER tar -xzf $HADOOP_HOME.tar.gz -C $HOME/"
    ssh $USER@$WORKER "tar -xzf $HADOOP_HOME.tar.gz -C $HOME/"
    echo ""

    echo ""
    echo "10) Reiniciando Worker..."
    echo "> ssh $USER@$WORKER sudo reboot now"
    ssh $USER@$WORKER "sudo reboot now"
    echo ""

    echo "El Worker de IP $WORKER ha sido configurado correctamente! :)"
    echo ""
done

echo "Todos los Workers han sido configurados! :D"
```

Código 15: Script para configurar todos los Workers desde el Master

Una vez configurado todo, reiniciamos el nodo Master y reingresamos cuando la maquina vuelva a estar activa.

Inicializar el Cluster

Si las maquinas estan correctamente configuradas, podremos levantar el File System distribuido sin problemas. Para esto es necesario formatear el disco e iniciar los servicios para la distribucion de este y para la gestion de recursos entre los Workers. Cabe notar que por disco me refiero a unidad de almacenamiento, como vimos en clases, un DFS no almacena datos de forma continua, si no que la reparte entre varios Nodos.

Dicho esto, ejecutamos la siguiente serie de comandos:

```
hdfs namenode -format
start-dfs.sh
start-yarn.sh
```

Código 16: Formateamos el File System, lo iniciamos y llamamos al negociador de recursos YARN

Luego, para corroborar de que todo esta en orden, ejecutamos el siguiente comando:

```
hdfs fsck / -files -blocks
```

Código 17: Corroborar estado de nuestro HDFS

La salida debiese ser algo del siguiente estilo:

```
...
Status: HEALTHY
 Number of data-nodes:  4
 Number of racks:               1
 Total dirs:                    11
 Total symlinks:                0

Replicated Blocks:
 Total size:    428156 B
 Total files:   9
 Total blocks (validated):      8 (avg. block size 53519 B)
 Minimally replicated blocks:   8 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Blocks queued for replication: 0
...
The filesystem under path '/' is HEALTHY
```

Código 18: Estado del HDFS luego de haber ejecutado un Map Reduce

Notaremos que existe una seccion que muestra la cantidad de Data Nodes, si muestra 4 entonces configuramos correctamente nuestro cluster.

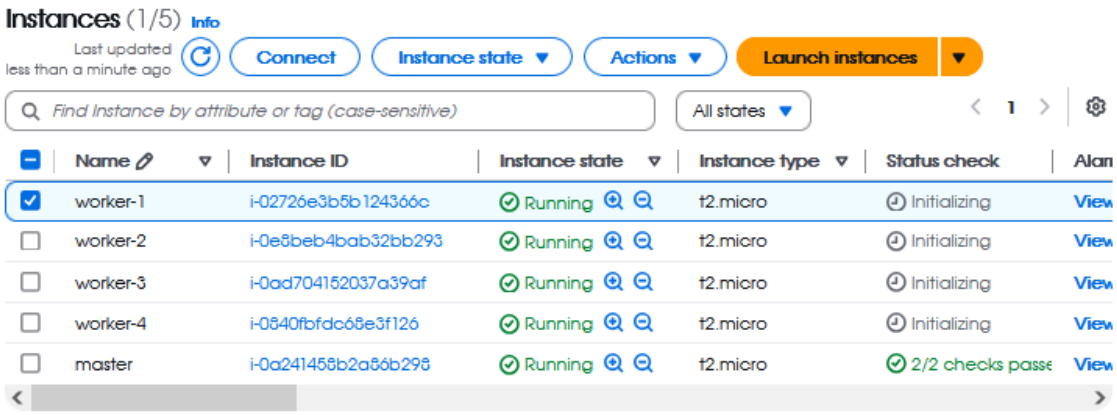

Figura 1: Captura de pantalla de la “AWS Management Console - EC2” que muestra las máquinas del cluster creado con el tutorial

Ejecutar el script de Map Reduce con Hadoop

Finalmente, ejecutamos nuestro algoritmo de Map Reduce. Para esto basta con crear un directorio /data, almacenar nuestro archivo a leer y finalmente ejecutar el ejecutable JAR contenido en nuestra carpeta de Hadoop. Una vez hecho esto, podemos analizar la salida del Map Reduce haciendole un Cat y pasandolo por Pipe a More para desplezgar la informacion por pantalla:

```
hdfs dfs -mkdir /data
hdfs dfs -put sw-script-e04.txt /data/
hadoop jar hadoop-3.3.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar \
wordcount /data/sw-script-e04.txt /data/sw-script-e04.wordcount
hadoop fs -cat /data/sw-script-e04.wordcount/part-r-00000 | more
```

Código 19: Ejecutar el Map Reduce y leer su salida

1.2. Expansión del cluster

Para expandir el cluster a 8 Workers, lo primero seria replicar las instancias desde AWS, estas debiesen tener los mismos atributos que los workers originales. Lo segundo seria actualizar la configuracion desde el Master para incluir las nuevas IP's y habilitar el uso de 4 Nodos desde los XML de Hadoop. Lo tercero seria configurar las conexiones de SSH hacia los workers nuevos, para esto podemos reutilizar el script de Bash pero solo con las IP's privadas nuevas.

```
# hadoop-3.3.6/etc/hadoop/mapred-site.xml
<property>
    <name>mapreduce.job.running.map.limit</name>
    <value>8</value> # Modificar de 4 a 8
</property>
# ...
<property>
<name>mapreduce.job.reduces</name>
    <value>8</value> # Modificar de 4 a 8
</property>
# ...
<property>
    <name>mapreduce.job.running.reduce.limit</name>
    <value>8</value> # Modificar de 4 a 8
</property>

# hadoop-3.3.6/etc/hadoop/workers
172.31.30.0
172.31.17.157
172.31.24.51
172.31.27.79
172.31.84.79
172.31.93.162
172.31.92.28
172.31.81.101

# /home/ubuntu/install.sh
# WORKERS=("172.31.30.0" "172.31.17.157" "172.31.24.51" "172.31.27.79")
WORKERS=("172.31.30.0" "172.31.17.157" "172.31.24.51" "172.31.27.79"  \
"172.31.84.79" "172.31.93.162" "172.31.92.28" "172.31.81.101")
MASTER="172.31.30.146"
```

Código 20: Los archivos que hay que modificar antes de ejecutar el script de instalacion nuevamente

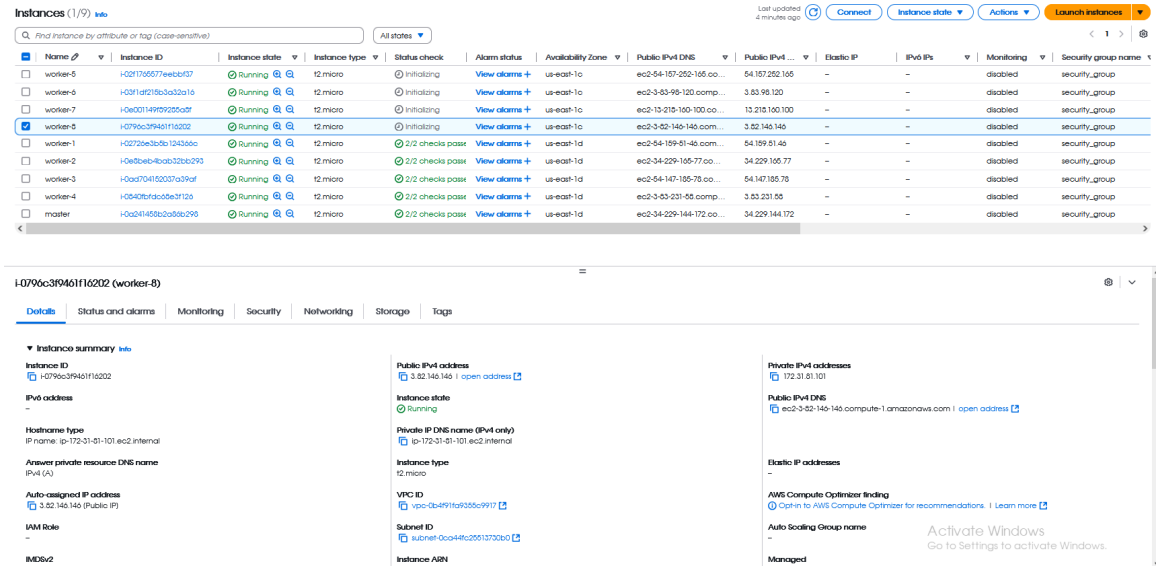

Figura 2: Captura de pantalla de la “AWS Management Console - EC2” que muestra las máquinas del cluster expandido

1.3. Cliente web

Aqui podemos apreciar todas las instancias desde la interfaz de EC2 de AWS, las instancias expandidas tienen el nombre de Worker y un numoro que va desde 5 hasta 8. Sus direcciones IP privadas aparecen en el apartado de expansion y tienen las mismas caracteristicas que los otros Nodos.

La figura 3 muestra la consola web de Hadoop y la figura 4 la consola web del HDFS.

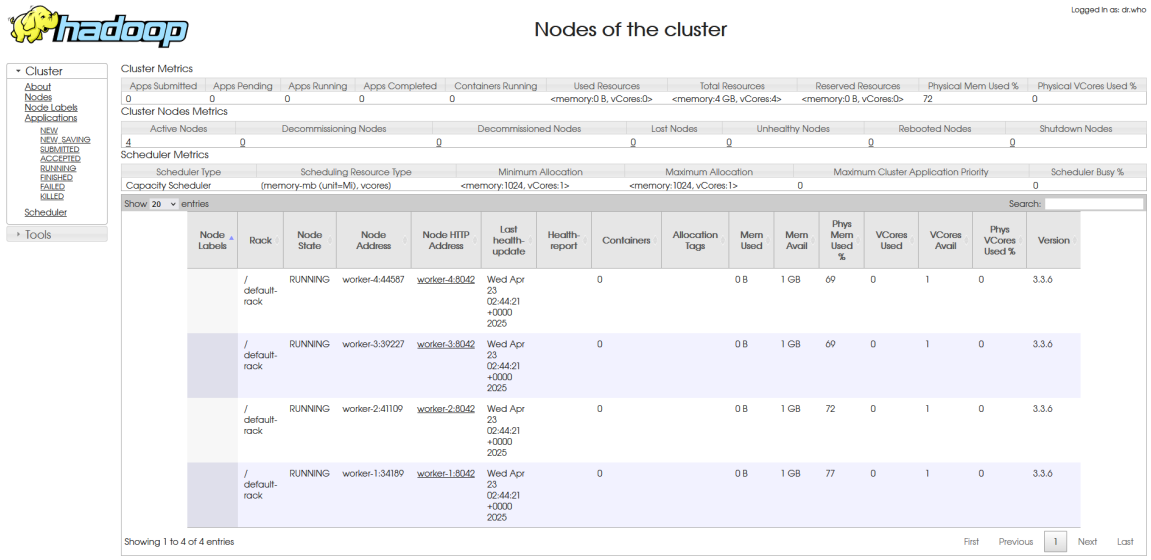

Figura 3: Captura de pantalla del cliente web de Hadoop

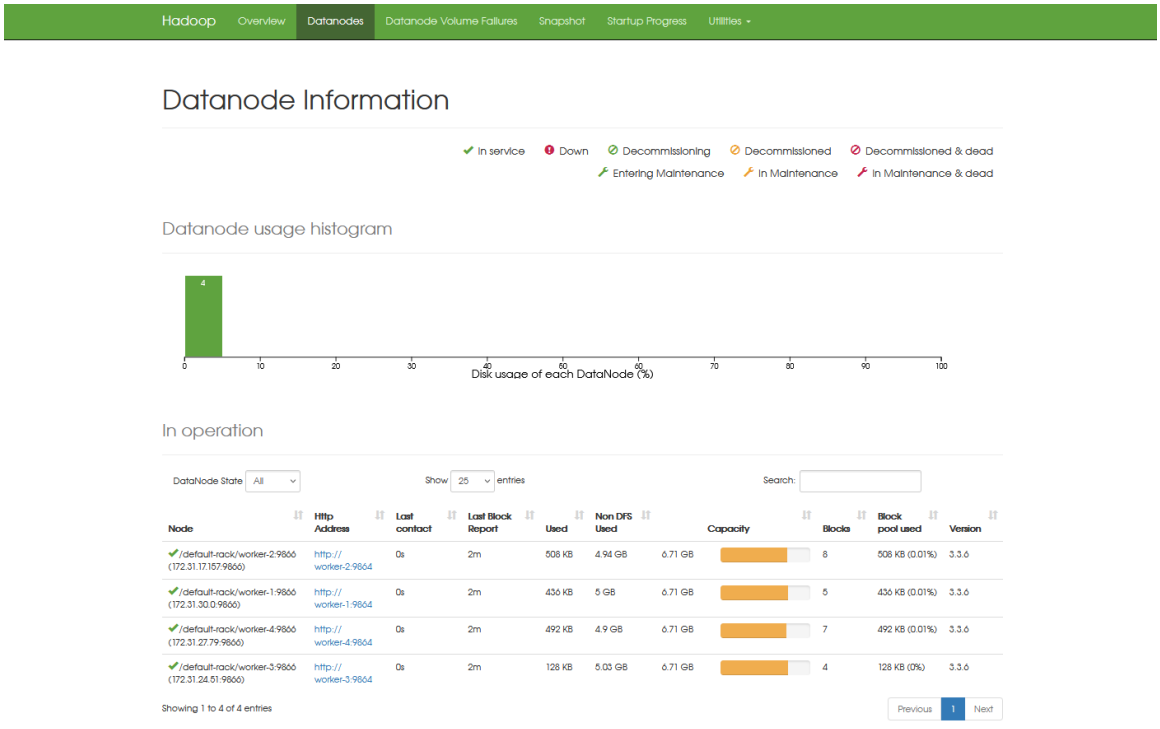

Figura 4: Captura de pantalla del cliente web de HDFS

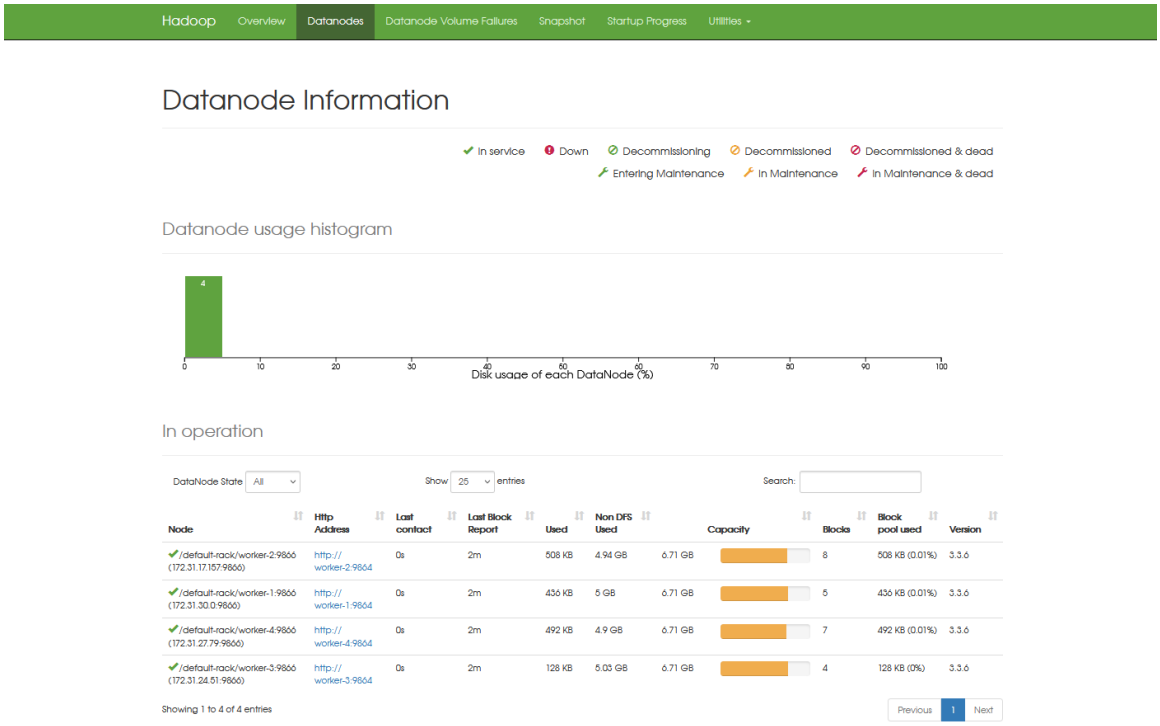

Figura 5: Instancias mejoradas para no sufrir

2. Instalación de Apache Hive

2.1. Procedimiento

Lo primero que debemos hacer para evitar cualquier posterior de cabeza grande, es ampliar las capacidades de las instancias que tenemos construidas en AWS. Las mauinas Worker pasaran a ser de tipo t3.small mientras que el Master a t3.medium. Esta mejora la realizamos desde la interfaz de EC2 de AWS.

Lo segundo que hacemos es ir a la pagina oficial de Apache Hive para descargar la version 4.0.1. Desde el nodo Master tendremos que usar el comando WGET para descargar el comprimido correspondiente:

```
wget https://dlcdn.apache.org/hive/hive-4.0.1/apache-hive-4.0.1-bin.tar.gz
```

Código 21: Usar WGET para descargar Apache Hive 4.0.1

Con el archivo ya descargado, lo descomprimimos de la misma manera que con Hadoop:

```
tar -xvzf apache-hive-4.0.1-bin.tar.gz -C .
```

Código 22: Descomprimimos la carpeta de Apache Hive 4.0.1

Luego, para poder utilizar el Apache Hive, debemos crear el archivo de configuracion necesario para que pueda funcionar por sobre nuestra instalcion de Hadoop, para esto creamos el siguiente archivo XML y lo guardamos en /home/ubuntu/apache-hive-4.0.1-bin/conf/

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:derby:metastore_db;create=true</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>org.apache.derby.jdbc.EmbeddedDriver</value>
    </property>
    <property>
        <name>datanucleus.schema.autoCreateAll</name>
        <value>true</value>
    </property>
    <property>
        <name>hive.metastore.schema.verification</name>
        <value>false</value>
    </property>
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
    </property>
    <property>
        <name>hive.server2.thrift.port</name>
        <value>10000</value>
    </property>
    <property>
        <name>hive.server2.thrift.bind.host</name>
        <value>0.0.0.0</value>
    </property>
    <property>
        <name>hive.server2.authentication</name>
        <value>NOSASL</value>
 </property>
    <property>
        <name>hive.server2.enable.doAs</name>
        <value>false</value>
    </property>
</configuration>
```

Código 23: Archivo de configuracion de Apache Hive

Creado dicho archivo, reiniciamos las maquinas.

Con la maquinas ya reiniciadas, levantamos el servicio metastore y de hiveserver2 de la siguiente manera:

```bash
# Levanta ambos servicios en segundo plano y esconde su salida
nohup hive --service hiveserver2 > /dev/null 2>&1
nohup hive --service metastore > /dev/null 2>&1
```

Código 24: Levantar MetaStore y HiveServer2

Antes de seguir, es importante recordar que Apache Hive funciona por sobre Hadoop, por ende el DFS debe estar funcionando al igual que el negociador de recursos, asi que los levantamos manualmente a traves de

los scripts que vienen con Hadoop de forma normal, luego, cuando ambos serivicios esten funcionando, corremos la shell de Apache Hive usando Beeline:

```
beeline -u "jdbc:hive2://localhost:10000/default;auth=noSasl" --verbose=true
```

Código 25: Usar Beeline como interfaz a Hive

2.2. Prueba

Se nos entrega la Query a continuacion para probar el funcionamiento correcto de Hive:

```
CREATE DATABASE sw_dialogs;

USE sw_dialogs;

CREATE TABLE sw04_dialogs (
  line      INT,
  character STRING,
  dialog    STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = " ",
  "quoteChar" = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA INPATH
'/data/sw-script-e04.txt'
INTO TABLE
sw04_dialogs;

SELECT character, COUNT(*) AS lines
FROM sw04_dialogs
GROUP BY character
ORDER BY lines DESC
LIMIT 12;
```

Código 26: Ejemplo de uso de Apache Hive

Para corroborar que Hive funciona correctamente, descargue la salida completa tras ejecutar la Query anteriormente mencionada. En las paginas siguientes esta la salida completa tras la ejecucion de dicho comando desde la Shell de Beeline.

```
ubuntu@ip-172-31-30-146:~$ beeline -u "jdbc:hive2://localhost:10000/default;auth=noSasl" --
    ↪ verbose=true
!connect jdbc:hive2://localhost:10000/default;auth=noSasl '' [passwd stripped]
Connecting to jdbc:hive2://localhost:10000/default;auth=noSasl
Connected to: Apache Hive (version 4.0.1)
Driver: Hive JDBC (version 4.0.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 4.0.1 by Apache Hive
0: jdbc:hive2://localhost:10000/default> show databases;
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : Compiling command(queryId=ubuntu_20250501025436_a463ca5e-c723-4037-a672-4465
    ↪ e790fae9): show databases
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:database_name, type:
    ↪ string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025436_a463ca5e-c723-4037-a672
    ↪ -4465e790fae9); Time taken: 2.378 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025436_a463ca5e-c723-4037-a672-4465
    ↪ e790fae9): show databases
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=ubuntu_20250501025436_a463ca5e-c723-4037-a672
    ↪ -4465e790fae9); Time taken: 0.217 seconds
printed operations logs
Getting log thread is interrupted, since query is done!
+----------------+
| database_name  |
+----------------+
| default        |
+----------------+
1 row selected (3.366 seconds)
0: jdbc:hive2://localhost:10000/default> !run hive-script.hql
>>>  CREATE DATABASE sw_dialogs;
going to print operations logs
printed operations logs
Getting log thread is interrupted, since query is done!
INFO  : Compiling command(queryId=ubuntu_20250501025448_df7a8fbc-1987-45fb-ad48-
    ↪ d61a6bb732a5): CREATE DATABASE sw_dialogs
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025448_df7a8fbc-1987-45fb-ad48-
    ↪ d61a6bb732a5); Time taken: 0.009 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025448_df7a8fbc-1987-45fb-ad48-
    ↪ d61a6bb732a5): CREATE DATABASE sw_dialogs
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=ubuntu_20250501025448_df7a8fbc-1987-45fb-ad48-
    ↪ d61a6bb732a5); Time taken: 0.224 seconds
No rows affected (0.253 seconds)
>>>
>>>  USE sw_dialogs;
going to print operations logs
printed operations logs
```

Código 27: Salida de Apache Hive (Parte 1)

```
Getting log thread is interrupted, since query is done!
INFO  : Compiling command(queryId=ubuntu_20250501025448_a7a58e45-3065-4b8d-987f-382
    ↪ f6be6ac0d): USE sw_dialogs
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025448_a7a58e45-3065-4b8d-987f
    ↪ -382f6be6ac0d); Time taken: 0.026 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025448_a7a58e45-3065-4b8d-987f-382
    ↪ f6be6ac0d): USE sw_dialogs
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=ubuntu_20250501025448_a7a58e45-3065-4b8d-987f
    ↪ -382f6be6ac0d); Time taken: 0.041 seconds
No rows affected (0.108 seconds)
>>>
>>>  CREATE TABLE sw04_dialogs (
line      INT,
character STRING,
dialog    STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
"separatorChar" = " ",
"quoteChar" = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
going to print operations logs
printed operations logs
Getting log thread is interrupted, since query is done!
INFO  : Compiling command(queryId=ubuntu_20250501025448_7e5cb1f6-8c26-4e7c-aa1e-7
    ↪ a905630ea48): CREATE TABLE sw04_dialogs (
line      INT,
character STRING,
dialog    STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
"separatorChar" = " ",
"quoteChar" = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1")
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025448_7e5cb1f6-8c26-4e7c-aa1e-7
    ↪ a905630ea48); Time taken: 0.102 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025448_7e5cb1f6-8c26-4e7c-aa1e-7
    ↪ a905630ea48): CREATE TABLE sw04_dialogs (
line      INT,
character STRING,
dialog    STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
"separatorChar" = " ",
"quoteChar" = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1")
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=ubuntu_20250501025448_7e5cb1f6-8c26-4e7c-aa1e-7
    ↪ a905630ea48); Time taken: 0.724 seconds
```

Código 28: Salida de Apache Hive (Parte 2)

```
No rows affected (0.866 seconds)
>>>
>>>  LOAD DATA INPATH
'/data/sw-script-e04.txt'
INTO TABLE
sw04_dialogs;
going to print operations logs
printed operations logs
going to print operations logs
INFO  : Compiling command(queryId=ubuntu_20250501025449_c327dfb8-5749-408f-b787-70
    ↪ bc8925a884): LOAD DATA INPATH
'/data/sw-script-e04.txt'
INTO TABLE
sw04_dialogs
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025449_c327dfb8-5749-408f-b787
    ↪ -70bc8925a884); Time taken: 0.406 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025449_c327dfb8-5749-408f-b787-70
    ↪ bc8925a884): LOAD DATA INPATH
'/data/sw-script-e04.txt'
INTO TABLE
sw04_dialogs
INFO  : Starting task [Stage-0:MOVE] in serial mode
INFO  : Loading data to table sw_dialogs.sw04_dialogs from hdfs://172.31.30.146/data/sw-
    ↪ script-e04.txt
INFO  : Starting task [Stage-1:STATS] in serial mode
INFO  : Executing stats task
INFO  : Table sw_dialogs.sw04_dialogs stats: [numFiles=2, totalSize=156556,
    ↪ numFilesErasureCoded=0]
INFO  : Completed executing command(queryId=ubuntu_20250501025449_c327dfb8-5749-408f-b787
    ↪ -70bc8925a884); Time taken: 0.866 seconds
printed operations logs
Getting log thread is interrupted, since query is done!
No rows affected (1.308 seconds)
>>>
>>>  SELECT character, COUNT(*) AS lines
FROM sw04_dialogs
GROUP BY character
ORDER BY lines DESC
LIMIT 12;
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : Compiling command(queryId=ubuntu_20250501025450_36332ece-1f44-45b7-998f-69
    ↪ c953bdb7af): SELECT character, COUNT(*) AS lines
FROM sw04_dialogs
GROUP BY character
ORDER BY lines DESC
LIMIT 12
```

Código 29: Salida de Apache Hive (Parte 3)

```
INFO  : No Stats for sw_dialogs@sw04_dialogs, Columns: character
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:character, type:string,
    ↪ comment:null), FieldSchema(name:lines, type:bigint, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=ubuntu_20250501025450_36332ece-1f44-45b7-998f
    ↪ -69c953bdb7af); Time taken: 3.888 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=ubuntu_20250501025450_36332ece-1f44-45b7-998f-69
    ↪ c953bdb7af): SELECT character, COUNT(*) AS lines
FROM sw04_dialogs
GROUP BY character
ORDER BY lines DESC
LIMIT 12
WARN  : Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions.
    ↪  Consider using a different execution engine (i.e. tez) or using Hive 1.X releases.
INFO  : Query ID = ubuntu_20250501025450_36332ece-1f44-45b7-998f-69c953bdb7af
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : number of splits:2
INFO  : Submitting tokens for job: job_1746065552179_0001
INFO  : Executing with tokens: []
printed operations logs
going to print operations logs
INFO  : The url to track the job: http://master:8088/proxy/application_1746065552179_0001/
INFO  : Starting Job = job_1746065552179_0001, Tracking URL = http://master:8088/proxy/
    ↪ application_1746065552179_0001/
INFO  : Kill Command = /home/ubuntu/hadoop-3.3.6/bin/mapred job  -kill
    ↪ job_1746065552179_0001
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
```

Código 30: Salida de Apache Hive (Parte 4)

```
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
INFO  : 2025-05-01 02:55:12,158 Stage-1 map = 0%,  reduce = 0%
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : 2025-05-01 02:55:21,611 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 2.27 sec
INFO  : 2025-05-01 02:55:22,648 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.43 sec
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : 2025-05-01 02:55:33,002 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.68 sec
printed operations logs
going to print operations logs
INFO  : MapReduce Total cumulative CPU time: 6 seconds 680 msec
INFO  : Ended Job = job_1746065552179_0001
INFO  : Launching Job 2 out of 2
INFO  : Starting task [Stage-2:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
```

Código 31: Salida de Apache Hive (Parte 5)

```
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1746065552179_0002
INFO  : Executing with tokens: []
INFO  : The url to track the job: http://master:8088/proxy/application_1746065552179_0002/
INFO  : Starting Job = job_1746065552179_0002, Tracking URL = http://master:8088/proxy/
    ↪ application_1746065552179_0002/
INFO  : Kill Command = /home/ubuntu/hadoop-3.3.6/bin/mapred job  -kill
    ↪ job_1746065552179_0002
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2025-05-01 02:55:50,592 Stage-2 map = 0%,  reduce = 0%
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : 2025-05-01 02:56:00,914 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.66 sec
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
printed operations logs
going to print operations logs
INFO  : 2025-05-01 02:56:12,332 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 5.7 sec
INFO  : MapReduce Total cumulative CPU time: 5 seconds 700 msec
INFO  : Ended Job = job_1746065552179_0002
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 6.68 sec   HDFS Read: 179440
    ↪ HDFS Write: 1807 HDFS EC Read: 0 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.7 sec   HDFS Read: 10647 HDFS
    ↪ Write: 348 HDFS EC Read: 0 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 12 seconds 380 msec
INFO  : Completed executing command(queryId=ubuntu_20250501025450_36332ece-1f44-45b7-998f
    ↪ -69c953bdb7af); Time taken: 78.686 seconds
printed operations logs
Getting log thread is interrupted, since query is done!
```

Código 32: Salida de Apache Hive (Parte 6)

```
+-------------+--------+
|  character  | lines  |
+-------------+--------+
| LUKE        | 508    |
| HAN         | 306    |
| THREEPIO    | 238    |
| BEN         | 164    |
| LEIA        | 114    |
| VADER       | 82     |
| RED LEADER  | 74     |
| BIGGS       | 68     |
| TARKIN      | 56     |
| OWEN        | 50     |
| TROOPER     | 38     |
| WEDGE       | 28     |
+-------------+--------+
12 rows selected (82.764 seconds)
```

Código 33: Salida de Apache Hive (Parte 7)

Mientras que la salida de referencia era:

```
+-------------+--------+
|  character  | lines  |
+-------------+--------+
| LUKE        | 524    |
| HAN         | 135    |
| THREEPIO    | 119    |
| BEN         | 82     |
| LEIA        | 57     |
| VADER       | 41     |
| RED LEADER  | 73     |
| BIGGS       | 34     |
| TARKIN      | 28     |
| OWEN        | 25     |
| TROOPER     | 19     |
| WEDGE       | 14     |
+-------------+--------+
12 rows selected (67.678 seconds)
```

Código 34: Resultado esperado para ejemplo de uso de Apache Hive

3. Exploración del HDFS

Para desarrollar el Script que muestre algo similar a lo que sale a continuacion, me aproveche de la salida que tuve. En mi caso, se dio perfectamente que Hadoop guardo el archivo en un unico bloque, por ende hice un Script en Bash que funcionara para mi situacion especifica. De no ser asi el caso, hubiese guardado cada IP por bloque en archivos separados y luego guardarlos en un arreglo usano un pipe desde LS.

```
File:             /mydata/myfile.txt
Blocks:           2
Avg. Replication: 3.0

0 - blk_1073742511_1694 - 172.31.40.100 172.31.32.111 172.31.35.247
1 - blk_1073742511_1695 - 172.31.32.111 172.31.210.111 172.31.35.247
```

Código 35: Ejemplo de uso de Apache Hive

El Script que hice fue el siguiente:

```
#!/bin/bash
HDFS_FILE="/user/hive/warehouse/sw_dialogs.db/sw04_dialogs/sw-script-e04.txt"

hdfs fsck $HDFS_FILE -files -blocks -locations > fsck_output

grep -o '[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*' fsck_output.txt > IP_with_Star_Wars.txt
grep -o 'blk_[0-9]*_[0-9]*' fsck_output.txt > Blocks_with_Star_Wars.txt
grep "Average␣block␣replication:" fsck_output.txt | grep -o "[0-9\.]*" > Rep_Fact_Star_Wars
    ↪ .txt

BLOCKS_QTY="$(wc␣-l␣Blocks_with_Star_Wars.txt␣|␣grep␣-o␣'[0-9]*')"
AVG_REP="$(cat␣Rep_Fact_Star_Wars.txt)"

COUNTER=0

echo "File:␣$HDFS_FILE"
echo "Blocks:␣$BLOCKS_QTY"
echo "Avg.␣Replication:␣$AVG_REP"

cat Blocks_with_Star_Wars.txt | while read line
do
    echo "$COUNTER␣-␣$line␣-␣$(cat␣Blocks_with_Star_Wars.txt)"
    ((counter++))
done
```

Código 36: Muestra el archivo, los bloques que ocupa, el factor de replicacion y la ubicacion de cada bloque

Para descargar el archivo solicitado, segui las intrucciones para instalar AWS CLI indicadas en la documentacion de AWS (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) y luego descargue el archivo usando el ultimo comando:

```
# Instalacion de AWS CLI

# Descargar archivo comprimido
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
# Descomprimir
unzip awscliv2.zip
# Instalar
sudo ./aws/install

# Descargar el CSV indicado
aws s3 cp s3://utfsm-inf356-dataset/vlt_observations_000.csv vlt_observations_000.csv --no-
    ↪ sign-request
```

Código 37: Instalar AWS CLI

4. Uso del cluster

4.1. Importación

Primero, para determinar el orden correcto de los datos, los insertare de forma directa a la Base de Datos desde Hive, para esto hice una Query generada por un Script en donde cada parametro recibe el nombre de parametro_n, donde n es el numero de la columna, el cual se obtiene contando la cantidad de comas en la primera fila del archivo CSV. De forma preliminar todas las columnas seran de tipo String, ya que no tenemos forma de saber el tipo de dato de antemano.

```
# Campos que tendra el archivo. Como desconocemos el contenido, le asignamos parametros genericos

# Exportamos cada linea del Script de manera dinamica
echo "CREATE DATABASE dirty_database;" > dirty_data_script.hql
echo "USE dirty_database;" >> dirty_data_script.hql
echo "CREATE TABLE dirty_table (" >> dirty_data_script.hql

# Obtenemos la cantidad de parametros extrayendo una fila aleatoria del archivo
# al que posteriormente se le remplazaron las comas por saltos de linea
grep -Eo "," vlt_observations_000_single_row.csv > comas
N_OF_PARAMS=$(wc -l < comas)

# Contador para nombrar cada parametro
COUNTER=0

# Guarda cada parametro generico como un String, posteriormente los
# segregaremos en base a las salidas que nos dan las Querys de esta
# Base de Datos Generica
while [ "$COUNTER" -le "$N_OF_PARAMS" ]; do
    echo "      parametro_$COUNTER string," >> dirty_data_script.hql
        COUNTER=$((COUNTER+1))
done

# Quitar la ultima Coma que queda molestando
truncate -s-2 dirty_data_script.hql

echo " )" >> dirty_data_script.hql
echo "COMMENT 'Datos Genericos'" >> dirty_data_script.hql
echo "ROW FORMAT DELIMITED" >> dirty_data_script.hql
echo "FIELDS TERMINATED BY ',';" >> dirty_data_script.hql

# Finalmente cargamos el CSV en nuestro HDFS para rellenar los datos
echo "LOAD DATA INPATH '/data/vlt_observations_000.csv' INTO TABLE vlt_obs_generic;" >> dirty_data_script.hql

# Subir el archivo a nuestro DFS
hdfs dfs -put vlt_observations_000.csv /data/
```

Código 38: Importacion de los datos sucios

```
CREATE DATABASE dirty_database;
USE dirty_database;
CREATE TABLE dirty_table (
        parametro_0 string,
        parametro_1 string,
        parametro_2 string,
        parametro_3 string,
        parametro_4 string,
        parametro_5 string,
        parametro_6 string,
        parametro_7 string,
        parametro_8 string,
        parametro_9 string,
        parametro_10 string,
        parametro_11 string,
        parametro_12 string,
        parametro_13 string,
        parametro_14 string,
        parametro_15 string,
        parametro_16 string,
        parametro_17 string,
        parametro_18 string,
        parametro_19 string,
        parametro_20 string,
        parametro_21 string,
        parametro_22 string,
        parametro_23 string,
        parametro_24 string,
        parametro_25 string,
        parametro_26 string,
        parametro_27 string,
        parametro_28 string,
        parametro_29 string,
        parametro_30 string,
        parametro_31 string,
        parametro_32 string
)
COMMENT 'Datos Genericos'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
LOAD DATA INPATH '/data/vlt_observations_000.csv' INTO TABLE dirty_table;
```

Código 39: Importacion de los datos sucios

Una vez con la Base de Datos generica creada, realizamos una Query para corroborar que los datos se insertaron correctamente.

4.2. Parsing

Al revizar los datos, notamos inmediatamente que el Script toma las comas que estan dentro de comillas dobles. Para solucionar esto cree un Script en Python que se encarga de borrar todas las comas que se encuentran dentro de Strings con comillas dobles.

```
import csv
import re

# Lee el archivo original y guarda la correccion en otro archivo
with open("vlt_observations_000.csv") as fin, open("vlt_observations_000_clean.csv", "w")
    ↪ as fout:
    for line in fin:
        # Reemplaza comas dentro de paréntesis por otro carácter temporal
        line_fixed = re.sub(r'\(([^()]*)\)', lambda m: m.group(0).replace(",", "␣"), line)
        # Muestra por pantalla la linea a corregir
        print(f"Linea␣a␣corregir:␣{line_fixed}")
        # Escribe en el archivo la linea corregida
        fout.write(line_fixed)
```

Código 40: Limpieza de los datos sucios

Con los datos arreglados, realizamos otra tabla ya con los datos arreglados. Con esta tabla nueva creada, realizamos una Query para cada parametro en donde se devuelven los 100 valores unicos por cada columna, de esta manera podemos saber que significa cada columna del CSV original para poder crear una ultima tabla con cada variable con un nombre y tipo mas apropiado.

```
CREATE DATABASE dirty_database;
USE dirty_database;
CREATE TABLE clean_table (
        parametro_0 string,
        parametro_1 string,
        parametro_2 string,
        ...
        parametro_30 string,
        parametro_31 string,
        parametro_32 string
)
COMMENT 'Datos␣Genericos'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
LOAD DATA INPATH '/data/vlt_observations_000_clean.csv' INTO TABLE clean_table;
```

Código 41: Importacion de los datos sucios

```
USE dirty_database;
SELECT DISTINCT parametro_0 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_1 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_2 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_3 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_4 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_5 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_6 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_7 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_8 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_9 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_10 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_11 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_12 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_13 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_14 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_15 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_16 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_17 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_18 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_19 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_20 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_21 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_22 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_23 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_24 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_25 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_26 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_27 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_28 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_29 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_30 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_31 FROM clean_table LIMIT 100;
SELECT DISTINCT parametro_32 FROM clean_table LIMIT 100;
```

Código 42: Analisis de los datos limpios para entender que significa cada columna

Como ya hemos automatizado todo a este punto, le pasamos las tablas a nuestra LLM de preferencia y le preguntamos a que corresponden los datos devueltos por la Query.

4.3. Análisis

En base a el analisis entregado por la Inteligencia Artificial, ademas de indagacion y criterio propio, la Query que agrupa los valores de cada columna con su nombre y tipo adecuado queda de la siguiente forma:

```
 -- Creacion de una Base de Daots con los registros limpios
CREATE DATABASE base_de_datos_observaciones;
USE base_de_datos_observaciones;

 -- Datos con su significado y tipo correspondiente
CREATE TABLE observaciones(
        tipo_de_calibracion string,
        tiempo_en_ascencion_recta time,
        tiempo_en_declinacion time,
        fecha_y_hora_de_observacion datetime,
        codigo_asignado_eso string,
        investigadores string,
        tipo_de_observador string,
        titulo_de_proyecto string,
        tipo_de_programa string,
        tipo_de_instrumento string,
        tipo_de_archivo string,
        tipo_de_calibracion string,
        modo_de_observacion string,
        nombre_unico string,
        nombre_fits string,
        fecha_de_observacion date,
        objeto_observado string,
        codigo_observatorio string,
        pipeline_de_reduccion string,
        tiempo_de_finalizacion time
        redshift float
        longitud_onda_central float,
        longitud_onda_final float,
        modo_de_adquisicion string,
        dispersor_usado string,
        configuracion_espectral string,
        geometria_de_rendija string,
        fecha_julian_date double,
        airmass float,
        seeing float,
        null_0 string,
        null_1 string,
        null_2 string
)
 -- Darles el formato de lectura separado por comas
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
 -- Leer registros desde el archivo subido al DFS
LOAD DATA INPATH '/data/vlt_observations_000_clean.csv' INTO TABLE observaciones;
```

Código 43: Observacion con nombres y tipos correctos

```
...

 -- Agrupar en base a los modos de observacion mas comunes
SELECT modo_de_observacion, COUNT(*) AS cantidad
FROM observaciones
GROUP BY modo_de_observacion;

 -- Tiempo promedio en declinacion por cada tipo de instrumento
SELECT tipo_de_instrumento, AVG(tiempo_en_declinacion) AS promedio_declinacion
FROM observaciones
GROUP BY tipo_de_instrumento;

 -- Seeing promedio por hora de observacion
SELECT HOUR(fecha_y_hora_de_observacion) AS hora, AVG(seeing) AS promedio_seeing
FROM observaciones
GROUP BY HOUR(fecha_y_hora_de_observacion)
ORDER BY hora;
```

Código 44: Observacion con nombres y tipos correctos