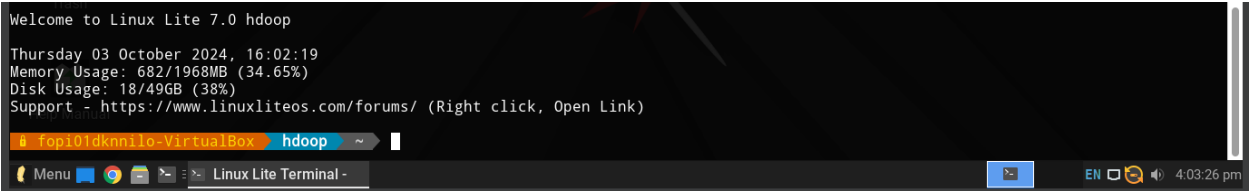


M2-FA1: Cassandra to Hive Migration Process

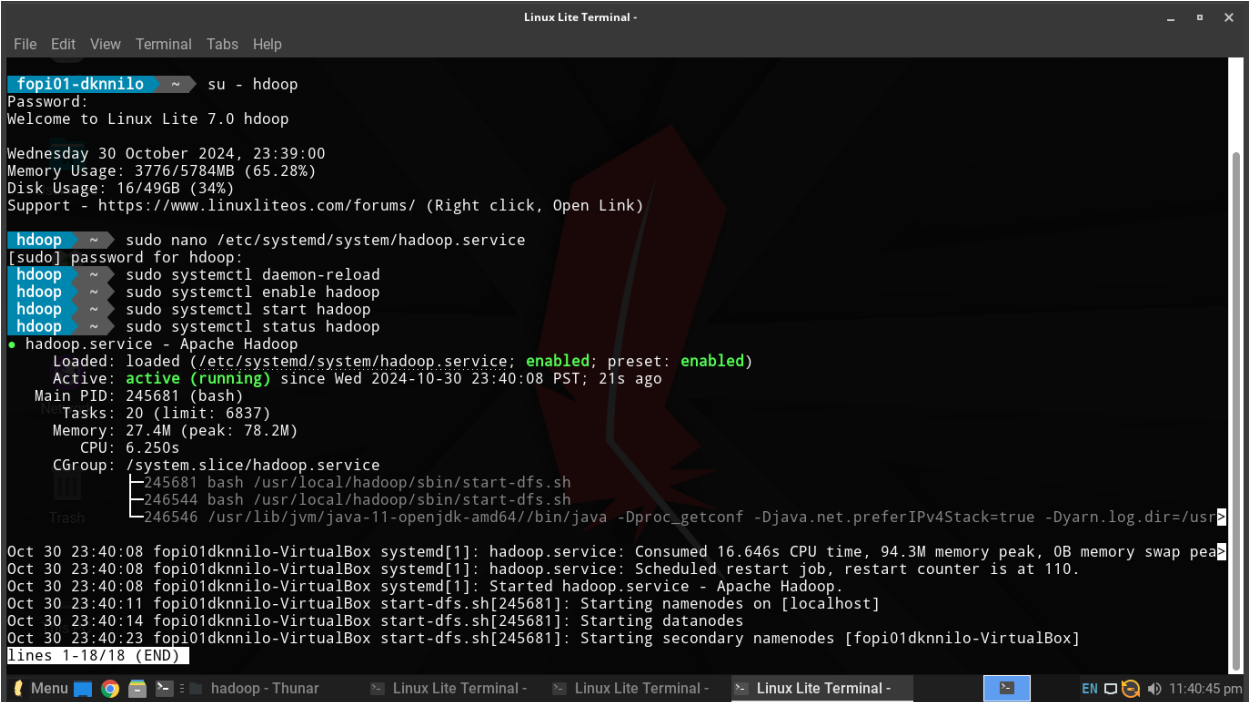
In order to establish a connection between Cassandra and Hive, we first installed Apache Hadoop from the official website (<https://www.apache.org/dyn/closer.cgi/hadoop/common/>) because Apache Hive requires it to function correctly. Once Hadoop was installed, we configured the necessary environment variables and edited the configuration files to set up the Hadoop ecosystem. We also installed Apache Hive by downloading it from the official Apache Hive releases page (<https://hive.apache.org/downloads.html>).

HADOOP STATUS CHECK

The first thing that we did was to create a superuser named “hadoop” in the Cassandra database so we can have administrative privileges to manage the necessary schemas and data migrations effectively. It includes granting the “hadoop” user all the necessary privileges to read and write to the keyspaces that would be accessed during the migration process.



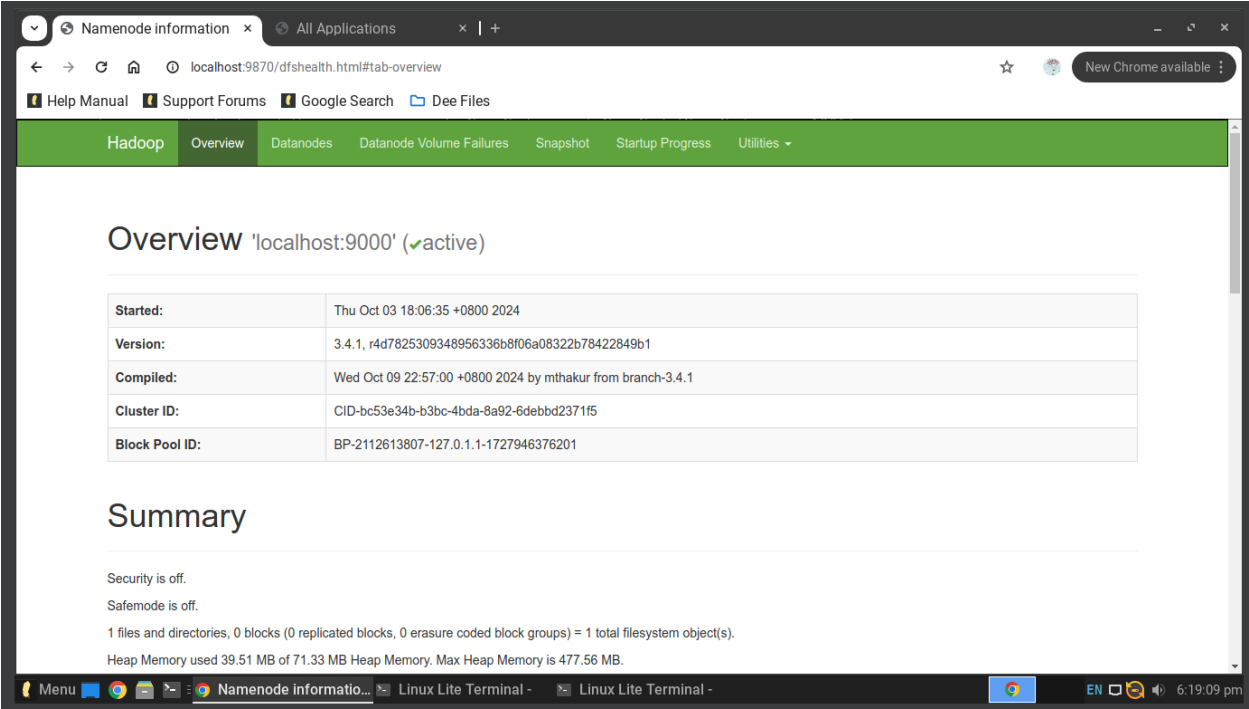
And then, after all the installation procedures, we can finally check the status of the Hadoop service by switching to the “hadoop” user using the `su - hadoop` command along with typing in our password. We created the `Hadoop.service` and set some configurations to keep it running in the background before we can check its status using the `sudo systemctl status hadoop` command. From the screenshot below, the Hadoop service is active (running).



As an alternative and more intuitive approach, we can also monitor the status of our Hadoop namenodes and datanodes through the web interfaces provided by Hadoop by accessing the <http://localhost:9870> so we can reach the NameNode web UI while we can also check the ResourceManager UI at <http://localhost:8088>.

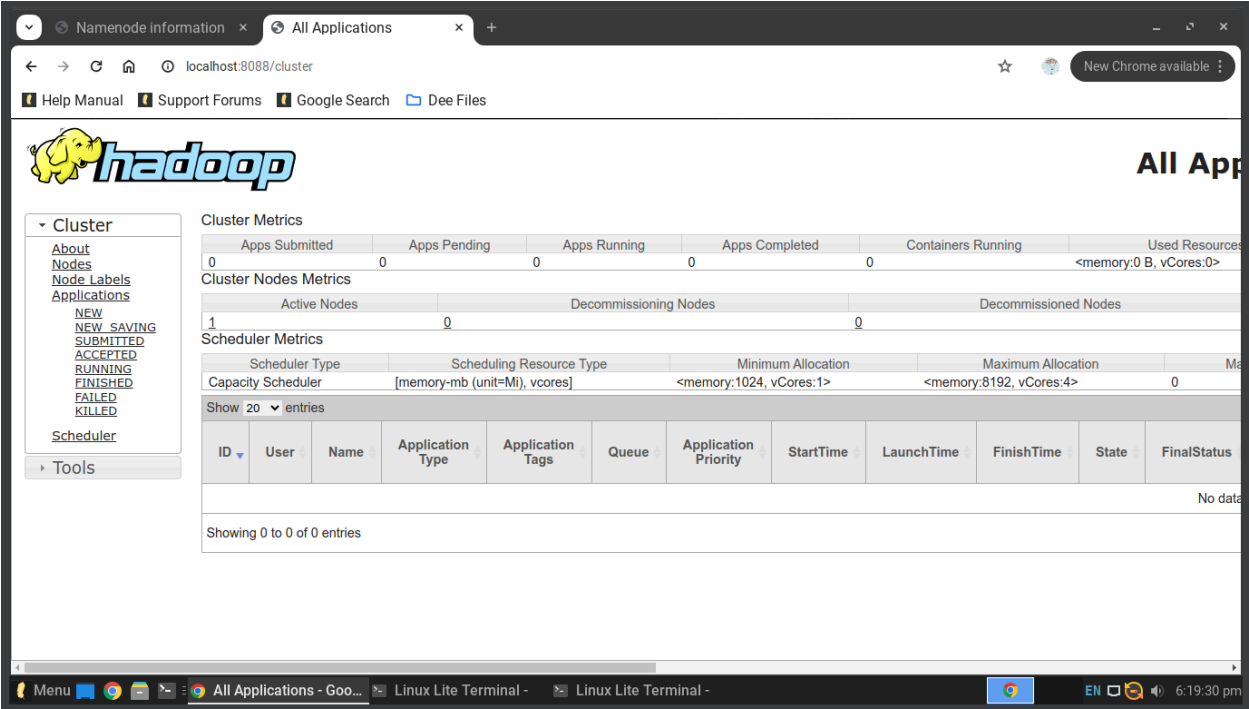
HADOOP NAMENODE WEB UI

The screenshot below shows the NameNode web UI at <http://localhost:9870>.



HADOOP RESOURCE MANAGER UI

The screenshot below shows the NameNode web UI at <http://localhost:8088>.



HIVE INSTALLATION

Since we already successfully installed Apache Hadoop and made sure that it's running properly, we can proceed with installing Apache Hive. The first step is to switch again to the "hadoop" user and check for the Java version because Hive requires Java 8 to function correctly.

```
Connected to Test Cluster at 127.0.0.1:9042
hadoop ~ java -version 1.7 | CQL spec 3.4.6 | Native protocol v5]
openjdk version "1.8.0_422"
OpenJDK Runtime Environment (build 1.8.0_422-8u422-b05-1~24.04-b05)
OpenJDK 64-Bit Server VM (build 25.422-b05, mixed mode)
hadoop ~ * FROM system_schema.keyspaces;
```

After checking the java version, we used the start-all.sh command to start all the Hadoop daemons and checked the status of the Hadoop nodes using the jps command to list all Java processes running on the machine. Based on the output below, the NameNode, DataNode, ResourceManager, and NodeManager were all operational.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
hadoop ~ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [fopi01-rapeaflor-VirtualBox]
Starting resourcemanager
Starting nodemanagers
hadoop ~ jps
16099 ResourceManager
15587 NameNode
16229 NodeManager
15717 DataNode
16423 Jps
15869 SecondaryNameNode
hadoop ~ cd /usr/local/hadoop/share/hadoop/mapreduce
hadoop ~ share > hadoop > mapreduce ls
hadoop-mapreduce-client-app-3.1.1.jar
hadoop-mapreduce-client-common-3.1.1.jar
hadoop-mapreduce-client-core-3.1.1.jar
hadoop-mapreduce-client-hs-3.1.1.jar
hadoop-mapreduce-client-hs-plugins-3.1.1.jar
hadoop-mapreduce-client-jobclient-3.1.1.jar
hadoop-mapreduce-client-jobclient-3.1.1-tests.jar
hadoop-mapreduce-client-nativetask-3.1.1.jar
hadoop-mapreduce-client-shuffle-3.1.1.jar
hadoop-mapreduce-client-uploader-3.1.1.jar
hadoop-mapreduce-examples-3.1.1.jar
jdifff
lib
lib-examples
sources
hadoop ~ share > hadoop > mapreduce readlink -f hadoop-mapreduce-examples-3.1.1.jar
/usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar
Menu Linux Lite Terminal - Linux Lite Terminal - Linux Lite Terminal - Linux Lite Terminal - EN 1:50:47 am
```

We also needed to verify the status of the MapReduce services by typing the yarn jar /usr/local/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar pi 10 100 command. It will run a sample process of computing pi from Map #0 to Map #09 to start sample job along with the logs output. It is important to run these samples to test if the service is working.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
hadoop ~ share > hadoop > mapreduce readlink -f hadoop-mapreduce-examples-3.1.1.jar
/usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar
hadoop ~ share > hadoop > mapreduce yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar pi 10
100
Number of Maps = 10
Samples per Map = 100
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Starting Job
2024-11-02 01:28:48,719 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
2024-11-02 01:28:49,335 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging
/job_1730478143536_0001
2024-11-02 01:28:49,583 INFO input.FileInputFormat: Total input files to process : 10
2024-11-02 01:28:50,138 INFO mapreduce.JobSubmitter: number of splits:10
2024-11-02 01:28:50,268 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead,
use yarn.system-metrics-publisher.enabled
2024-11-02 01:28:50,638 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1730478143536_0001
2024-11-02 01:28:50,641 INFO mapreduce.JobSubmitter: Executing with tokens: []
```

After checking the status of the MapReduce services, we can finally download the installation tarball file from the official Apache Software Foundation website using the `wget https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz` command and extracting it in our current directory using the `tar -xzf apache-hive-3.1.3-bin.tar.gz` command before navigating to the configuration directory (`conf`) and copying the `hive-default.xml` template into an executable `hive-site.xml`. We edited the mentioned xml file using the `nano hive-default.xml` command.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
hadoop ~$ wget https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz
--2024-11-02 03:14:16-- https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 326940667 (312M) [application/x-gzip]
Saving to: 'apache-hive-3.1.3-bin.tar.gz'

Network
apache-hive-3.1.3-bin.tar.gz 100%[=====] 311.79M 869KB/s in 7m 51s

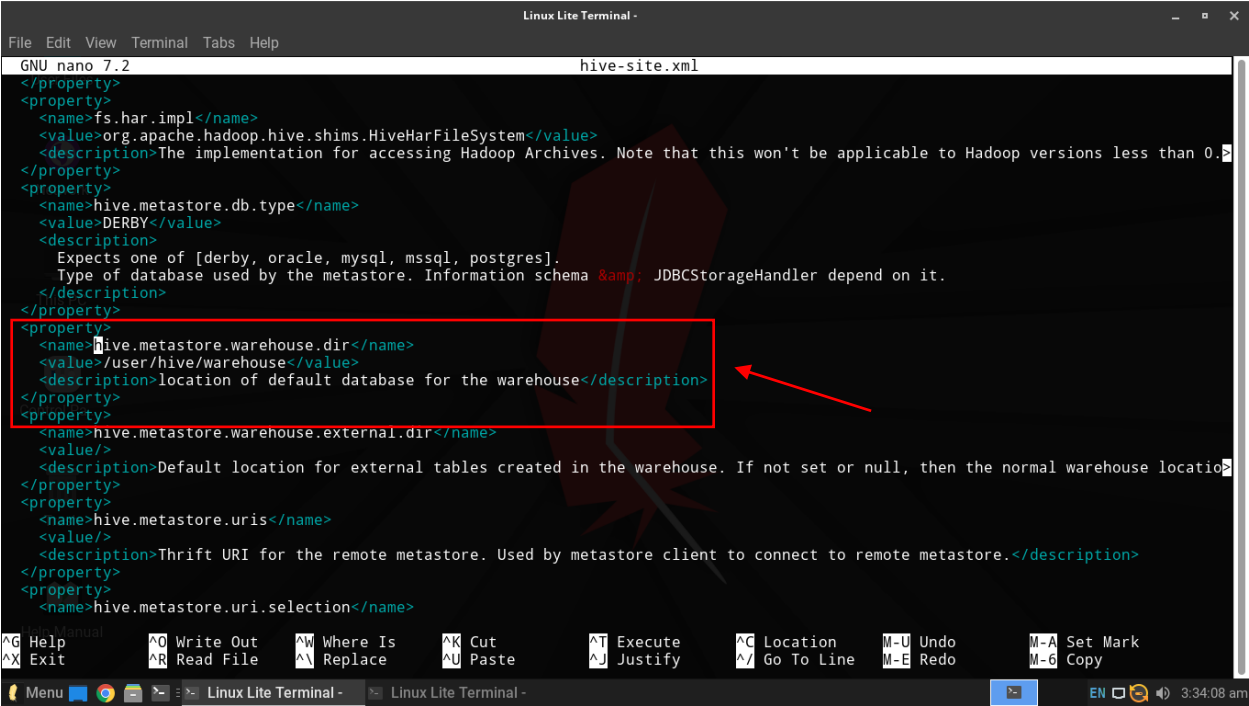
2024-11-02 03:22:09 (678 KB/s) - 'apache-hive-3.1.3-bin.tar.gz' saved [326940667/326940667]

hadoop ~$ tar -xzf apache-hive-3.1.3-bin.tar.gz
hadoop ~$ ls
apache-hive-3.1.2-bin.tar.gz  apache-hive-3.1.3-bin.tar.gz  Documents  hadoop-3.1.1.tar.gz  Music  Public  Videos
apache-hive-3.1.3-bin Desktop
hadoop ~$ mv apache-hive-3.1.3-bin hive
hadoop ~$ cd hive
hadoop ~$ cd conf
hadoop ~$ ls
beeline-log4j2.properties.template  hive-exec-log4j2.properties.template  llap-cli-log4j2.properties.template
hive-default.xml.template            hive-log4j2.properties.template       llap-daemon-log4j2.properties.template
hive-env.sh.template                ivysettings.xml                       parquet-logging.properties
hadoop ~$ cp hive-default.xml.template hive-site.xml
hadoop ~$ nano hive-site.xml
```

Inside the `hive-site.xml` file, we added two important configurations that are important for the proper functioning of Apache Hive. We defined a temporary directory where Hive can store intermediate data during query execution by adding the property `hive.exec.scratchdir` which we set to a directory as `/tmp/hive/java`. The second configuration involves setting the user under which Hive jobs will run. We added the property `hive.exec.user` where we can specify `$(user.name)` that dynamically fetches the name of the currently logged-in user.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 7.2 hive-site.xml
See the license for the specific language governing permissions and
limitations under the license.
--><configuration>
<!-- WARNING!!! This file is auto generated for documentation purposes ONLY! -->
<!-- WARNING!!! Any changes you make to this file will be ignored by Hive. -->
<!-- WARNING!!! You must make your changes in hive-site.xml instead. -->
<!-- Hive Execution Parameters -->
<property>
<name>system:java.io.tmpdir</name>
<value>/tmp/hive/java</value>
</property>
<property>
<name>system:user.name</name>
<value>${user.name}</value>
</property>
<property>
<name>hive.exec.script.wrapper</name>
<value/>
<description/>
</property>
<property>
<name>hive.exec.plan</name>
<value/>
<description/>
</property>
<property>
<name>hive.exec.stagingdir</name>
<value>hive-staging</value>
<description>Directory name that will be created inside table locations in order to support HDFS encryption. This is replaces ${h
</property>
</configuration>
^G Help
^X Exit
^O Write Out
^R Read File
^W Where Is
^M Replace
^K Cut
^U Paste
^T Execute
^J Justify
^C Location
^G Go To Line
^M-U Undo
^M-E Redo
^M-A Set Mark
^M-6 Copy
```

We also set the `metastore.warehouse.dir` property in the `hive-site.xml` file to `/usr/hive/warehouse`. This configuration specifies the default location for the Hive metastore to store the tables and partitions created in Apache Hive so that all Hive data files are organized in a structured manner to make it easier for us to manage and access the data.



```
GNU nano 7.2 hive-site.xml
</property>
<property>
  <name>fs.hdfs.impl</name>
  <value>org.apache.hadoop.hive.shims.HiveHdfsFileSystem</value>
  <description>The implementation for accessing Hadoop Archives. Note that this won't be applicable to Hadoop versions less than 0.
</property>
<property>
  <name>hive.metastore.db.type</name>
  <value>DERBY</value>
  <description>
    Expects one of [derby, oracle, mysql, mssql, postgres].
    Type of database used by the metastore. Information schema & JDBCStorageHandler depend on it.
  </description>
</property>
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/usr/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
<property>
  <name>hive.metastore.warehouse.external.dir</name>
  <value></value>
  <description>Default location for external tables created in the warehouse. If not set or null, then the normal warehouse location
</property>
<property>
  <name>hive.metastore.uris</name>
  <value></value>
  <description>Thrift URI for the remote metastore. Used by metastore client to connect to remote metastore.</description>
</property>
<property>
  <name>hive.metastore.uri.selection</name>
```

Using the `nano ~/.bashrc` command, we also need to set the proper Hive directory environment variables. It is one of the most important steps during the configuration process because it confirms that the system recognizes the Hive and Hadoop installations which will allow us to run Hive commands directly from the terminal. After making these changes, we saved the file and executed `source ~/.bashrc` to apply the updates immediately.



```
GNU nano 7.2 /home/hadoop/.bashrc *
alias usage='du -sk * | sort -n | perl -ne '\''($s,$f)=split(m/\t/);for (qw(K M G)) {if($s<1024) {printf("%.1f",$s);print "$_t$f\n";}}'\''
alias ls='ls --color'

# Powerline
if [ -f /usr/share/powerline/bindings/bash/powerline.sh ]; then
  source /usr/share/powerline/bindings/bash/powerline.sh
fi

# Linux Lite Custom Terminal
LLVER=$(awk '{print}' /etc/llver)

echo -e "Welcome to $LLVER ${USER}"
echo " "
date "+%A %d %B %Y, %T"
free -m | awk 'NR==2{printf "Memory Usage: %s/%sMB (%.2f%%)\n", $3,$2,$3*100/$2 }'
df -h | awk 'NF==5' {printf "Disk Usage: %d/%dGB (%s)\n", $3,$2,$5}'
echo "Support - https://www.linuxliteos.com/forums/ (Right click, Open Link)"
echo " "

#Hadoop Directory
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
export PATH=$PATH $HADOOP_HOME/bin $HADOOP_HOME/sbin

#Hive Directory
export HIVE_HOME=/usr/local/hive/apache-hive-3.1.3-bin
export HIVE_CONF_DIR=$HIVE_HOME/conf
export PATH=$PATH $HIVE_HOME/bin
export CLASSPATH=$CLASSPATH $HADOOP_HOME/lib/*:.
export CLASSPATH=$CLASSPATH $HIVE_HOME/lib/*:.
```

Next, we opened another terminal session as the “hadoop” user to create the necessary directories for Hive on HDFS. We created the Hive warehouse directory on HDFS using the `hdfs dfs -mkdir -p /user/hive/warehouse` command and verified the creation of the directory using the `hdfs dfs -ls /` command. After listing all the contents of the root directory in HDFS, we

also need to adjust the permissions of the Hive “hdoop” user. To make sure that the Hive user has the necessary permissions to access and modify the newly created directories, we executed the `hdfs dfs -chmod g+w /user`, `hdfs dfs -chmod g+wx /user`, `hdfs dfs -chmod g+wx /tmp`, and `hdfs dfs -chmod g+w /tmp` commands. After executing these commands, we cleared the terminal to maintain a clean workspace.

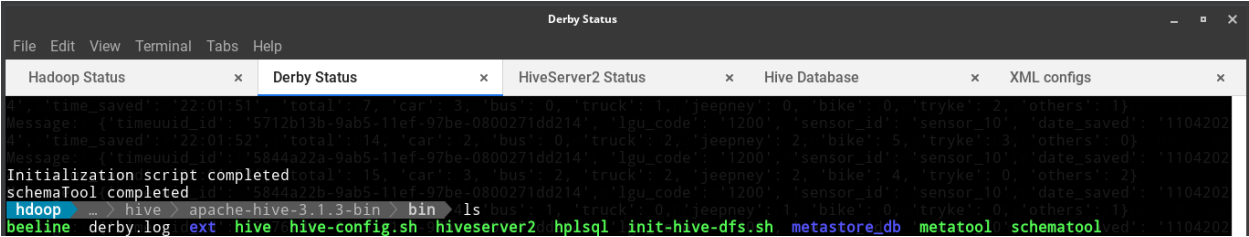
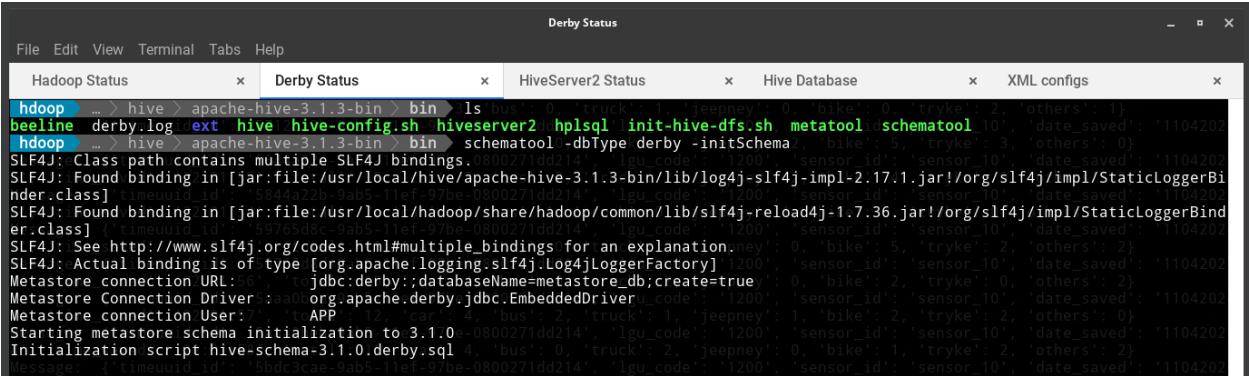
```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
hdoop ~ > hive > conf hdfs dfs -ls /
Found 1 items
drwx----- - hdoop supergroup          0 2024-11-02 01:28 /tmp
hdoop ~ > hive > conf hdfs dfs -mkdir -p /user/hive/warehouse
hdoop ~ > hive > conf hdfs dfs -ls /
Found 2 items
drwx----- - hdoop supergroup          0 2024-11-02 01:28 /tmp
drwxr-xr-x - hdoop supergroup          0 2024-11-02 04:02 /user
hdoop ~ > hive > conf hdfs dfs -chmod g+w /user
hdoop ~ > hive > conf hdfs dfs -chmod g+wx /user
hdoop ~ > hive > conf hdfs dfs -ls /
Found 2 items
drwx----- - hdoop supergroup          0 2024-11-02 01:28 /tmp
drwxrwxr-x - hdoop supergroup          0 2024-11-02 04:02 /user
hdoop ~ > hive > conf hdfs dfs -chmod g+wx /tmp
hdoop ~ > hive > conf hdfs dfs -chmod g+w /tmp
hdoop ~ > hive > conf clear
```

For the last configuration, we also copied the `hive-env.sh` template to create our own executable copy of the `hive-env.sh` file before editing its contents using the `nano hive-env.sh` command. Inside the `hive-env.sh` file, we commented out the `HADOOP_HOME` directory and set it as `/usr/local/hadoop`. After saving and closing the `hive-env.sh` file, these configurations guarantee that Hive could effectively communicate with Hadoop and Java as well as store logs in the specified directory. With these configurations complete, we were ready to run Hive with the correct environment variables. It completes the essential setup for Hive on top of Hadoop.

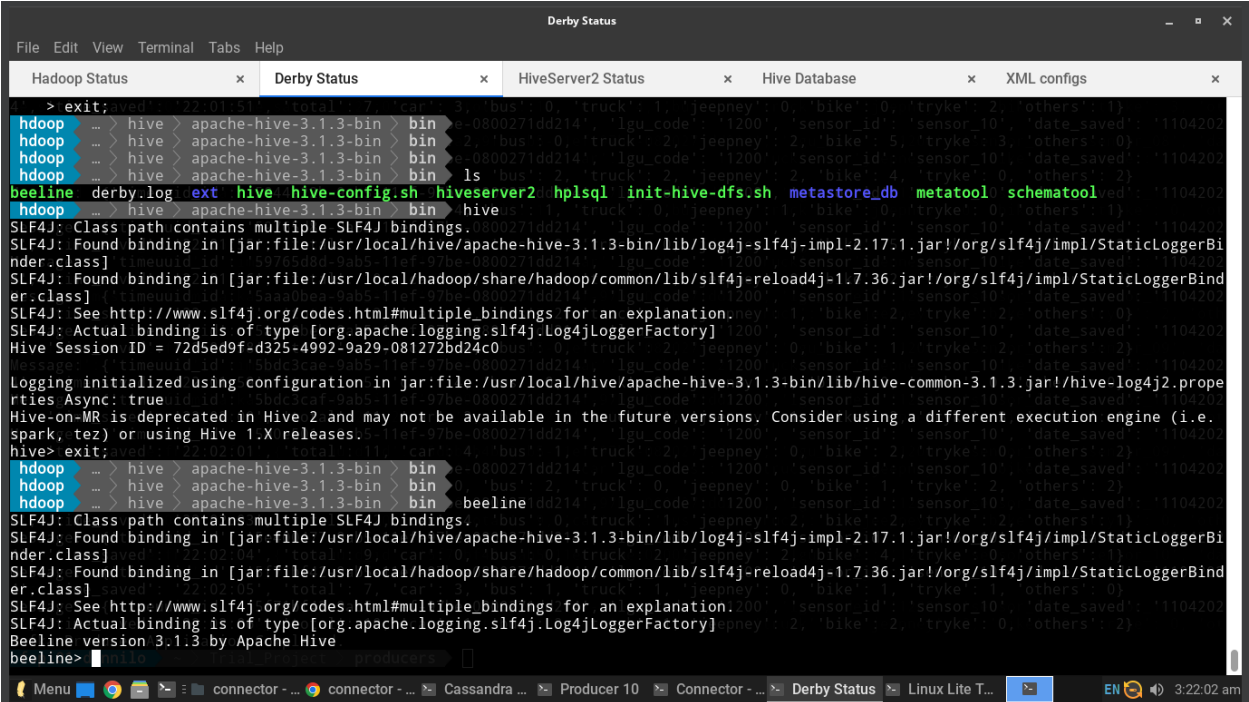
```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
hdoop ~ > hive > conf ls
beeline-log4j2.properties.template  hive-log4j2.properties.template  llap-daemon-log4j2.properties.template
hive-default.xml.template           hive-site.xml                    ivysettings.xml
hive-env.sh.template               llap-cli-log4j2.properties.template
hive-exec-log4j2.properties.template cp hive-env.sh.template hive-env.sh
hdoop ~ > hive > conf nano hive-env.sh
```

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 7.2                                hive-env.sh *
# Hive Client memory usage can be an issue if a large number of clients
# are running at the same time. The flags below have been useful in
# reducing memory usage:
#
# if [ "$SERVICE" = "cli" ]; then
#   if [ -z "$DEBUG" ]; then
#     export HADOOP_OPTS="$HADOOP_OPTS -XX:NewRatio=12 -Xms10m -XX:MaxHeapFreeRatio=40 -XX:MinHeapFreeRatio=15 -XX:+UseParNewGC -XX:-
#   else
#     export HADOOP_OPTS="$HADOOP_OPTS -XX:NewRatio=12 -Xms10m -XX:MaxHeapFreeRatio=40 -XX:MinHeapFreeRatio=15 -XX:-UseGCOverheadLim
#   fi
# fi
#
# The heap size of the jvm started by hive shell script can be controlled via:
#
# export HADOOP_HEAPSIZE=1024
#
# Larger heap size may be required when running queries over large number of files or partitions.
# By default hive shell scripts use a heap size of 256 (MB). Larger heap size would also be
# appropriate for hive server.
#
# Set HADOOP_HOME to point to a specific hadoop install directory
HADOOP_HOME=/usr/local/hadoop
#
# Hive Configuration Directory can be controlled by:
# export HIVE_CONF_DIR=
#
# Folder containing extra libraries required for hive compilation/execution can be controlled by:
# export HIVE_AUX_JARS_PATH=
#
# Help
# Exit
# Write Out
# Read File
# Where Is
# Replace
# Cut
# Paste
# Execute
# Justify
# Location
# Go To Line
# Undo
# Redo
# Set Mark
# Copy
Menu Pictures - Thunar Linux Lite Termina... Linux Lite Termina... Linux Lite Termina... Linux Lite Termina... 4:09:42 am
```

We can already proceed with initializing the Derby schema which is necessary for Hive to function correctly with its “metastore” by running a script provided in the Hive bin directory. This script sets up the Apache Derby database that Hive uses as its default metastore when running in standalone mode. We used the `schematool -dbType derby -initSchema` command to start the initialization process and waited for a few moments before the initialization script is successfully completed as shown in the following two screenshots below.



Since the initialization process is complete and we already have the `metastore_db` file, we can already access the Hive database through the command-line interface (CLI) by simply typing the “hive” command. It would launch the Hive shell where we can run HiveQL commands to create databases, tables, and execute queries against the data stored in HDFS. We can also access Hive through the Beeline command-line interface using the “beeline” command in the terminal. Beeline is a JDBC client that allows us to connect to Hive in a more secure and efficient way compared to the older Hive CLI. It's commonly used for production environments and supports remote connections according to the Apache Hive documentation.



PYTHON SCRIPT (FOR THE MIGRATION PROCESS)

The purpose of the python script we wrote below is to migrate the CCTV sensor data from a Cassandra database to a Hive database at 11:59 PM. The script connects to Cassandra to fetch aggregated vehicle count data from different sensors within a specified date range, transforms it into a suitable format using pandas, and inserts the data into a designated Hive table.

The script starts by defining the `fetch_cctv_data()` function which connects to Cassandra. After establishing the connection, it executes a query that counts the number of records from each sensor and are grouped by minute. It filters the records based on a specified date range to allow data collection within a daily window. The data retrieved from Cassandra is then stored in a Pandas DataFrame. Once the data is available in a DataFrame, the `insert_into_hive()` function is used to transfer it into Hive. It makes a connection to the Hive server, iterates over each row in the DataFrame, and inserts the data into the `vehicle_data` table. Each row represents a sensor reading at a specific minute, with columns for the sensor ID, count of vehicles, and timestamp. The function also includes error handling for any Hive issues.

A screenshot of a code editor window titled '~\Trial_Project\connector\connector-hive.py - Mousepad'. The editor contains Python code for connecting to Cassandra, fetching data, and inserting it into Hive. The code includes imports for Cassandra, PyHive, Pandas, and Datetime. It defines two functions: `fetch_cctv_data()` and `insert_into_hive(df)`. The `fetch_cctv_data()` function connects to a Cassandra cluster, executes a SQL query to fetch aggregated vehicle count data for a specific date range, and returns a Pandas DataFrame. The `insert_into_hive(df)` function connects to a Hive server and inserts the data from the DataFrame into a table. The code is written in a dark-themed editor with a menu bar (File, Edit, Search, View, Document, Help) and a taskbar at the bottom showing various application icons and the system clock (4:04:10 am).

On the other hand, the `migrate_cctv_data()` function continues the process by first fetching data from Cassandra and then inserting it into Hive if the DataFrame is not empty. It makes sure that the script only attempts to insert data into Hive if there is valid data to process to prevent unnecessary Hive operations. In order for us to automate the migration, we also included in the script the `BlockingScheduler` from the `apscheduler` library to schedule the `migrate_cctv_data()` function to run at 11:59 PM daily. The scheduler triggers the migration process daily at the specified time to help maintain up-to-date data in the Hive database. The script also initiates the scheduler upon execution and prints a confirmation message to indicate a successful startup.


```
~/Trial_Project/connector/connector-hive.py - Mousepad
File Edit Search View Document Help

conn = hive.Connection(host='127.0.0.1', port=10000, username='hadoop')
cursor = conn.cursor()
print("Successfully connected to Hive.")

for index, row in df.iterrows():
    cursor.execute(
        "INSERT INTO vehicle_data (sensor_id, count, date_saved) VALUES (%s, %s, %s)",
        (row.sensor_id, row.count, row.minute.strftime('%Y-%m-%d %H:%M:%S'))
    )
    print("Data inserted into Hive.")

cursor.close()
conn.close()

except Exception as e:
    print(f"Error connecting to Hive or inserting data: {e}")

# main function to perform migration with error handling
def migrate_cctv_data():
    print("Starting data migration process...")

    df = fetch_cctv_data()

    if not df.empty:
        insert_into_hive(df)
    else:
        print("No data fetched from Cassandra; skipping Hive insertion.")

# schedule the migration at 11:59 PM
if __name__ == "__main__":
    from apscheduler.schedulers.blocking import BlockingScheduler

    scheduler = BlockingScheduler()
    scheduler.add_job(migrate_cctv_data, 'cron', hour=23, minute=59)
    print("Scheduler started. Data migration will run daily at 11:59 PM.")
    scheduler.start()
```

TEST RUN (FOR THE MIGRATION PROCESS)

To test the migration process from Cassandra to Hive, we first need to examine and validate the data stored in the Cassandra database. Using the vehicle_data.vehicle_counts table, two SQL-like queries were executed to display the structure and content of the data. The first query, SELECT timeuuid_id, lgu_code, sensor_id, date_saved, total FROM vehicle_data.vehicle_counts LIMIT 10; retrieves columns including timeuuid_id, lgu_code, sensor_id, date_saved, and total. The second query, SELECT car, bus, truck, jeepney, bike, tryke, others FROM vehicle_data.vehicle_counts LIMIT 10; shows the counts for different vehicle types such as car, bus, truck, jeepney, bike, tryke, and others.

```
Cassandra Database
File Edit View Terminal Tabs Help

Cassandra Logs x Cassandra Database x Cassandra Configs x

cqlsh> SELECT timeuuid_id, lgu_code, sensor_id, date_saved, total FROM vehicle_data.vehicle_counts LIMIT 10;
timeuuid_id | HDFS Read: 37842 HDFS | lgu_code | sensor_id | date_saved | total
-----
e3ee4103-9aae-11ef-97be-0800271dd214 | 1200 | sensor_05 | 11042024 | 16
a5ec5258-9ab2-11ef-97be-0800271dd214 | 1200 | sensor_07 | 11042024 | 12
a5a5908d-9ab1-11ef-97be-0800271dd214 | 1200 | sensor_09 | 11042024 | 12
cf0ca99a-9ab0-11ef-97be-0800271dd214 | 1200 | sensor_10 | 11042024 | 12
59569fc9-9ab5-11ef-97be-0800271dd214 | 1200 | sensor_08 | 11042024 | 13
1e09b6c6-9ab2-11ef-97be-0800271dd214 | 1200 | sensor_04 | 11042024 | 7
d7fd1012-9aae-11ef-97be-0800271dd214 | 1200 | sensor_05 | 11042024 | 7
100c82aa-9ab3-11ef-97be-0800271dd214 | 1200 | sensor_06 | 11042024 | 7
2cfa4fe3-9ab0-11ef-97be-0800271dd214 | 1200 | sensor_03 | 11042024 | 7
7a328f97-9ab3-11ef-97be-0800271dd214 | 1200 | sensor_04 | 11042024 | 9

(10 rows)

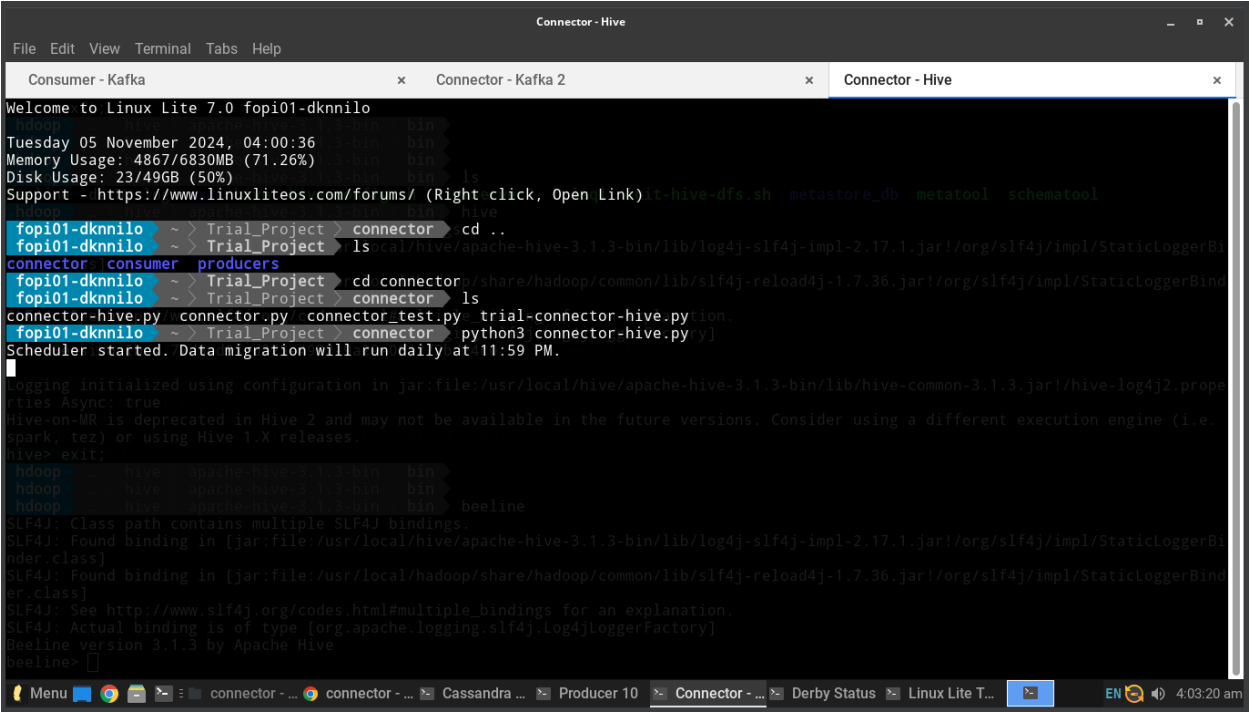
cqlsh> SELECT car, bus, truck, jeepney, bike, tryke, others FROM vehicle_data.vehicle_counts LIMIT 10;
car | bus | truck | jeepney | bike | tryke | others
-----
s1 | 0 | 2 | 2 | 5 | 2 | 224-11-01 11:48:00.0
s1 | 0 | 2 | 2 | 1 | 3 | 224-11-01 11:49:00.0
s1 | 0 | 1 | 2 | 3 | 2 | 2024-11-01 11:50:00.0
s4 | 0 | 0 | 2 | 0 | 3 | 224-11-01 11:51:00.0
s1 | 1 | 1 | 0 | 1 | 2 | 224-11-01 11:52:00.0
s1 | 0 | 2 | 1 | 4 | 3 | 224-11-01 11:53:00.0
s1 | 0 | 2 | 1 | 5 | 2 | 224-11-01 11:54:00.0

(10 rows)

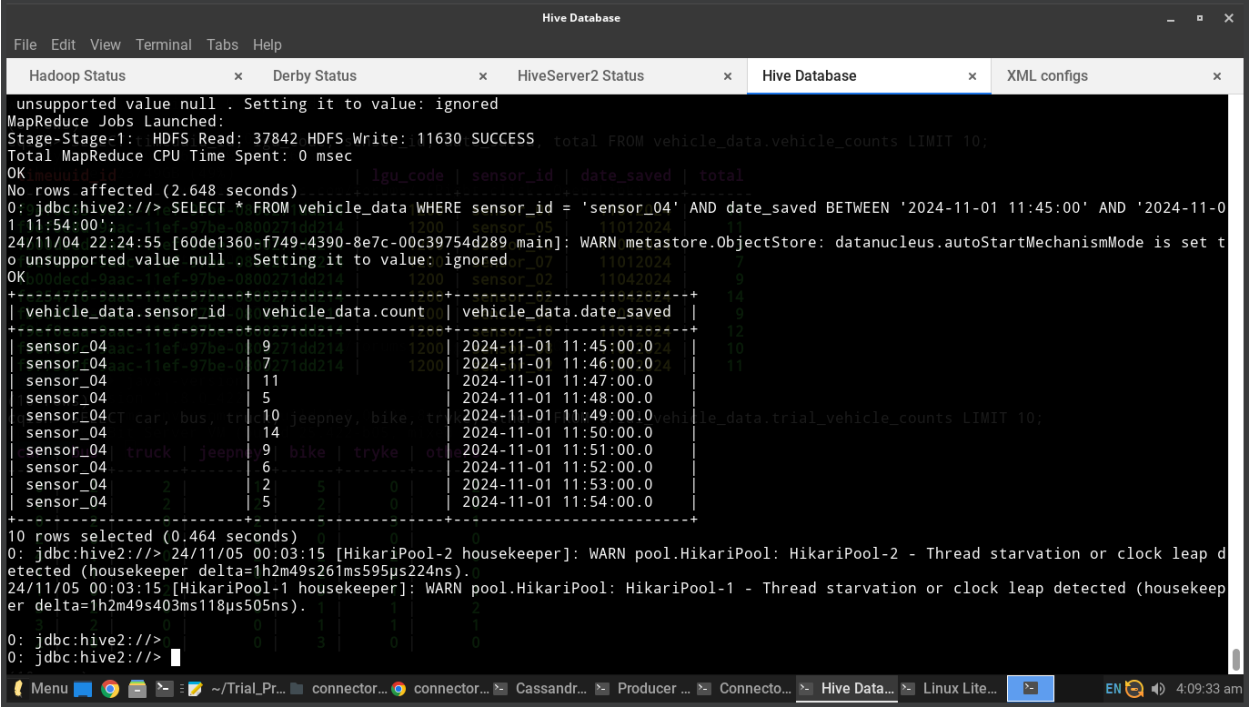
cqlsh> 
```

Afdr checking the Cassandra database for content, we can already migrate the data from Cassandra to Hive. We opened another terminal and went to the connector directory of our trial project before looking at our list of connectors using the ls command. We started the script

execution by running the python3 connector-hive.py command and waited for awhile for the time at 11:59 PM. For the sake of practicality, we already run an alternative python command the day before our testing in a separate terminal to migrate the data so that we can already output some results for the sake of testing the python script.

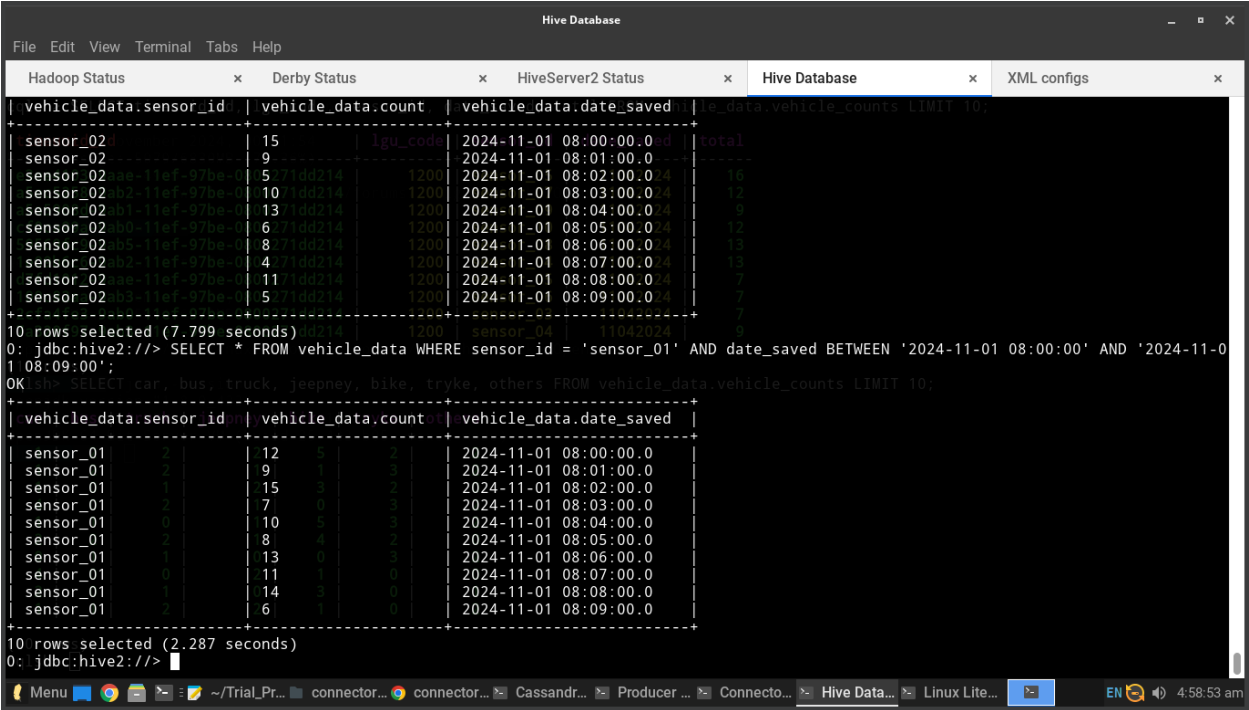
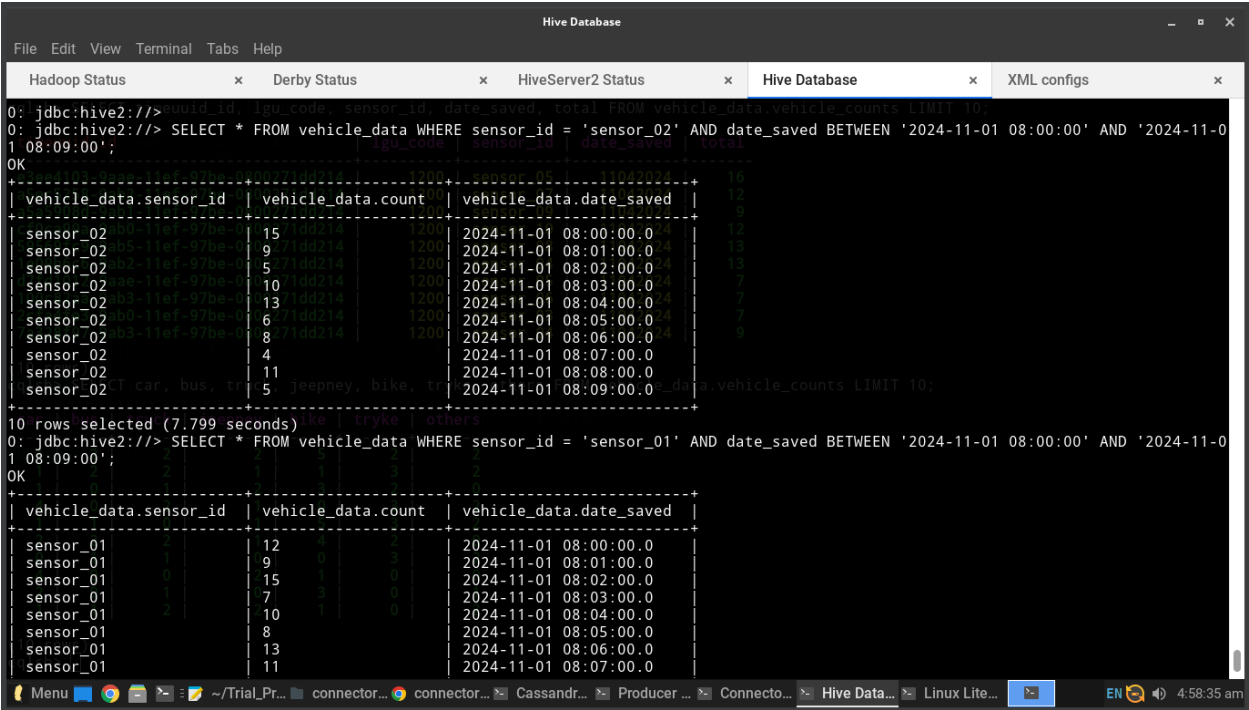


After waiting for at least six minutes, we prompted the Hive Database using the SELECT * FROM vehicle_data WHERE sensor_id = 'sensor_04' AND date_saved BETWEEN '2024-11-01 11:45:00' AND '2024-11-01 11:54:00; command through the “beeline” CLI to print out some data from sensor 4 before the end of the day and the output can be seen below:



We also tried to check for the migrated data for sensor 2 at the start of the day using the SELECT * FROM vehicle_data WHERE sensor_id = 'sensor_02' AND date_saved BETWEEN '2024-11-01 08:00:00' AND '2024-11-01 08:09:00; command. We also did the same thing for sensor 1 by typing in the SELECT * FROM vehicle_data WHERE sensor_id = 'sensor_01' AND date_saved BETWEEN '2024-11-01 08:00:00' AND '2024-11-01 08:09:00; command.

The outputs from the two screenshots below prove that these commands would show the records captured for both sensors during the specified time interval and would serve as proof that the migration is successful from Cassandra to Hive that can be used for further analysis.



REFERENCES:

Apache Software Foundation (n.d.). *Cassandra Documentation Version 4.1*.
<https://cassandra.apache.org/doc/4.1/index.html>

Coppage, Karen. (2020, January 14). *GettingStarted Installing Hive from a Stable Release*.
<https://cwiki.apache.org/confluence/display/Hive/GettingStarted#GettingStarted-InstallingHivefromaStableRelease>

IvyProSchool. (2023, December 30). *Configure Hive on Hadoop in Ubuntu | Data Engineering Tutorial for Beginners | Ivy Pro School* [Video]. YouTube.
<https://www.youtube.com/watch?v=LMrW2BFerh8&t=367s>