Group 5: Nilo, Daniella Kim N.
Peñaflor, Rowencell A.

Course/Section: CSS182-2/CS-O

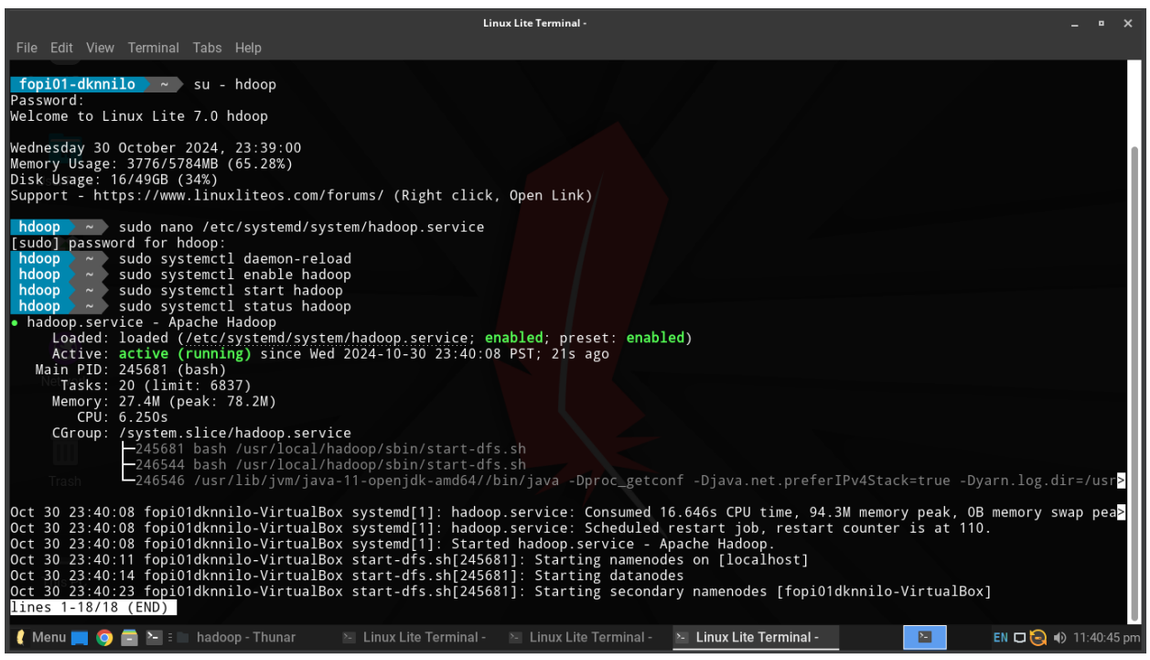
M2-FA1: Cassandra to Hive Migration Process

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 “hdoop” in the Cassandra database so we can have administrative privileges to manage the necessary schemas and data migrations effectively. It includes granting the “hdoop” user all the necessary privileges to read and write to the keyspaces that would be accessed during the migration process.

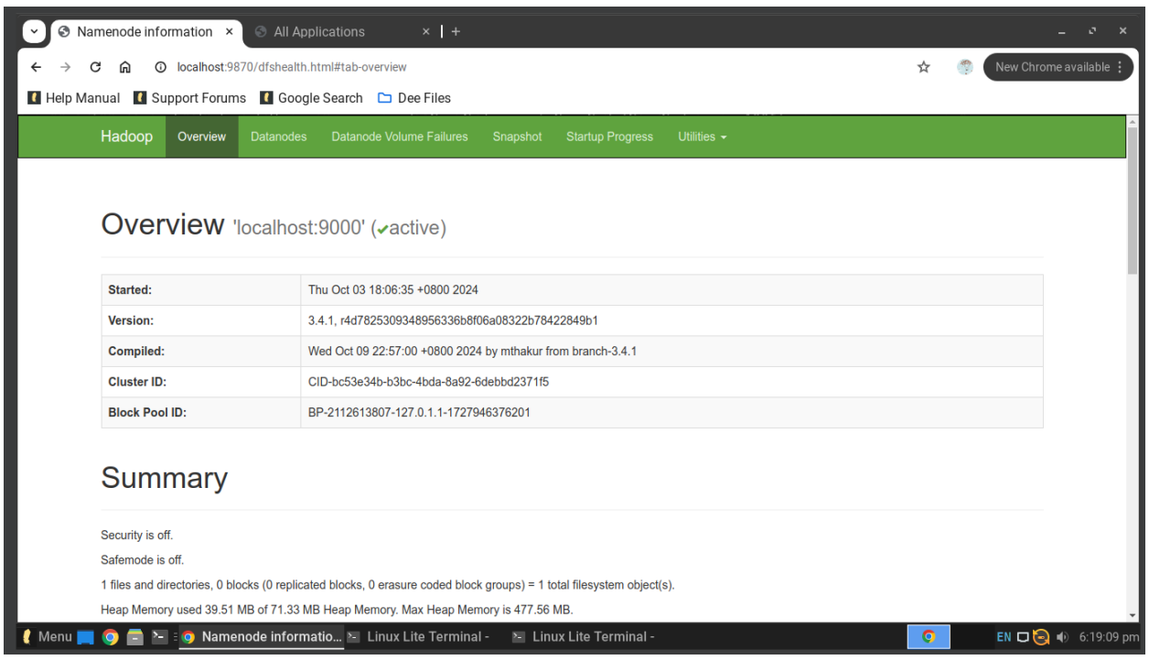
And then, after all the installation procedures, we can finally check the status of the Hadoop service by switching to the “hdoop” user using the `su - hdoop` command along with typing in our password. We created the Hadoop.service and set some configurations to keep it running in the background before 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.

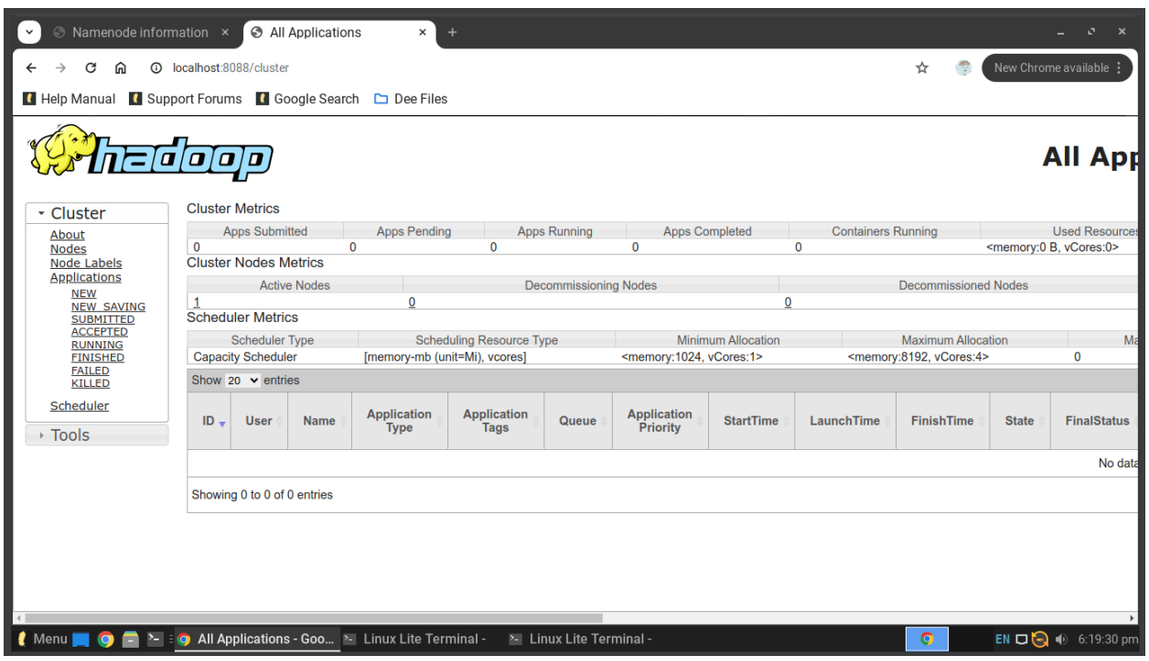
HADOOP NAMENODE WEB UI

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



HADOOP RESOURCE MANAGER UI

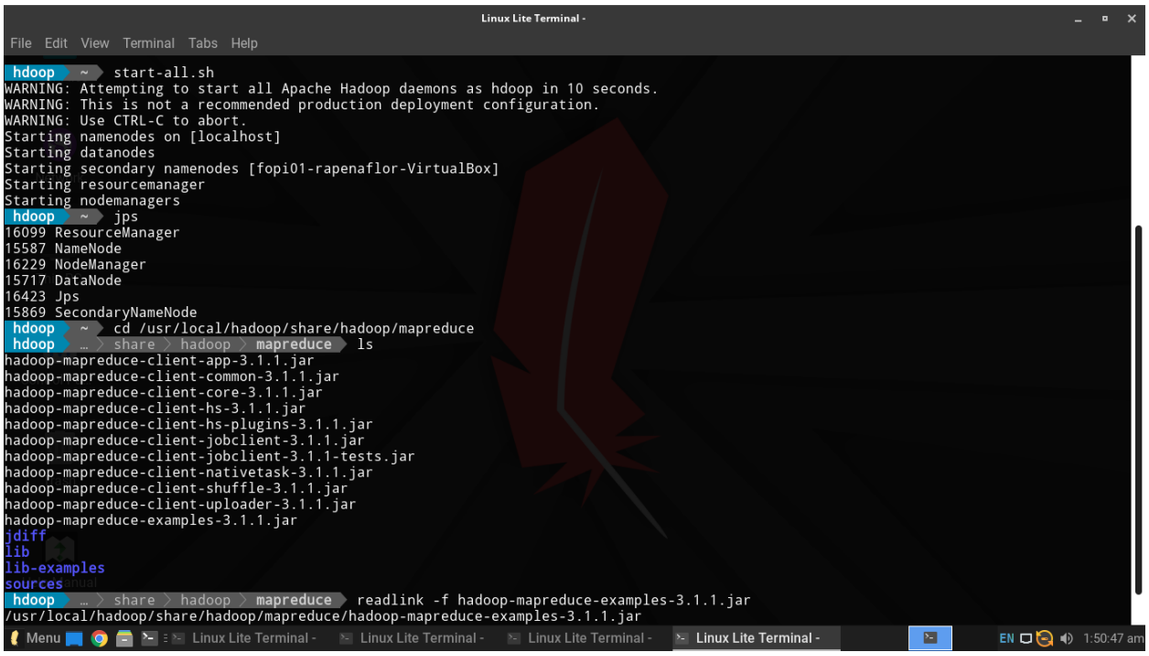
The screenshot below shows the NameNode web UI at http://localhost:8088.



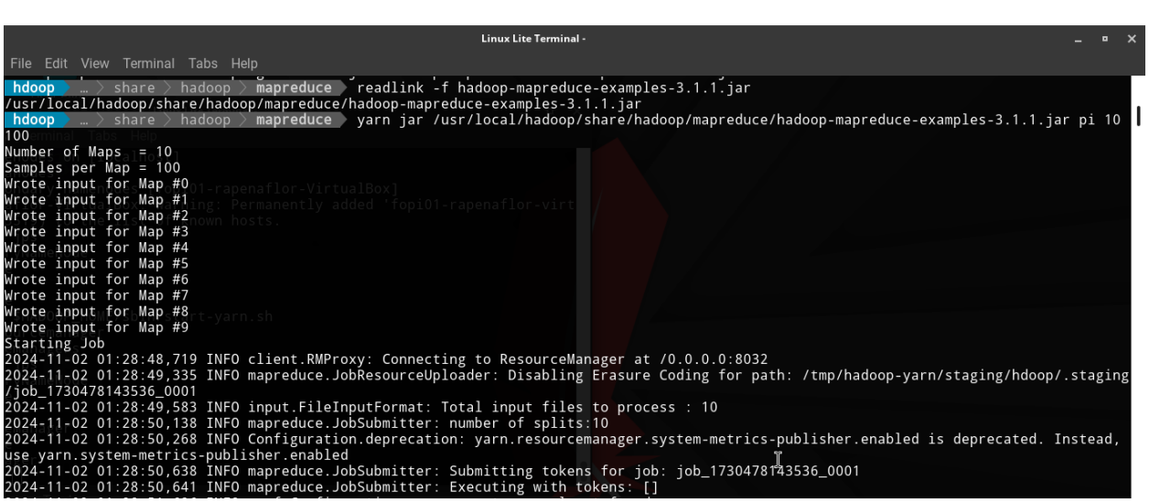
HIVE INSTALLATION

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 "hdoop" user and check for the Java version because Hive requires Java 8 to function correctly.

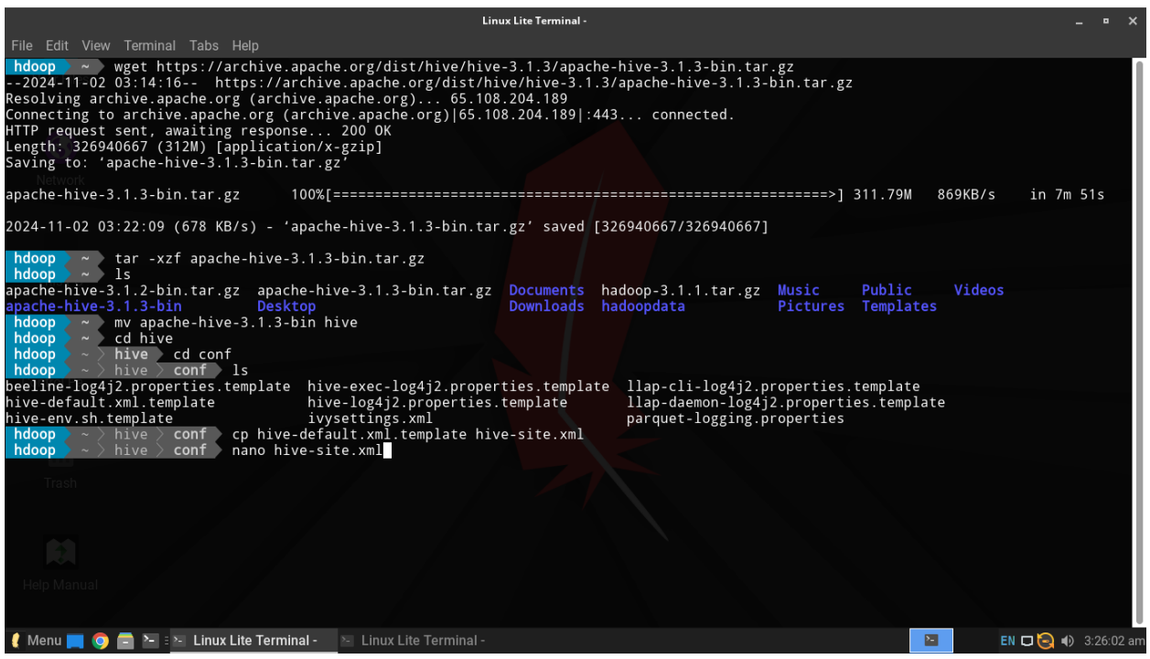
After checking the java version, we used the `start-all.sh` command to start all the Hadoop daemons and checked the status of the Hadoop nodes using the `jps` command to list all Java processes running on the machine. Based on the output below, the NameNode, DataNode, ResourceManager, and NodeManager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



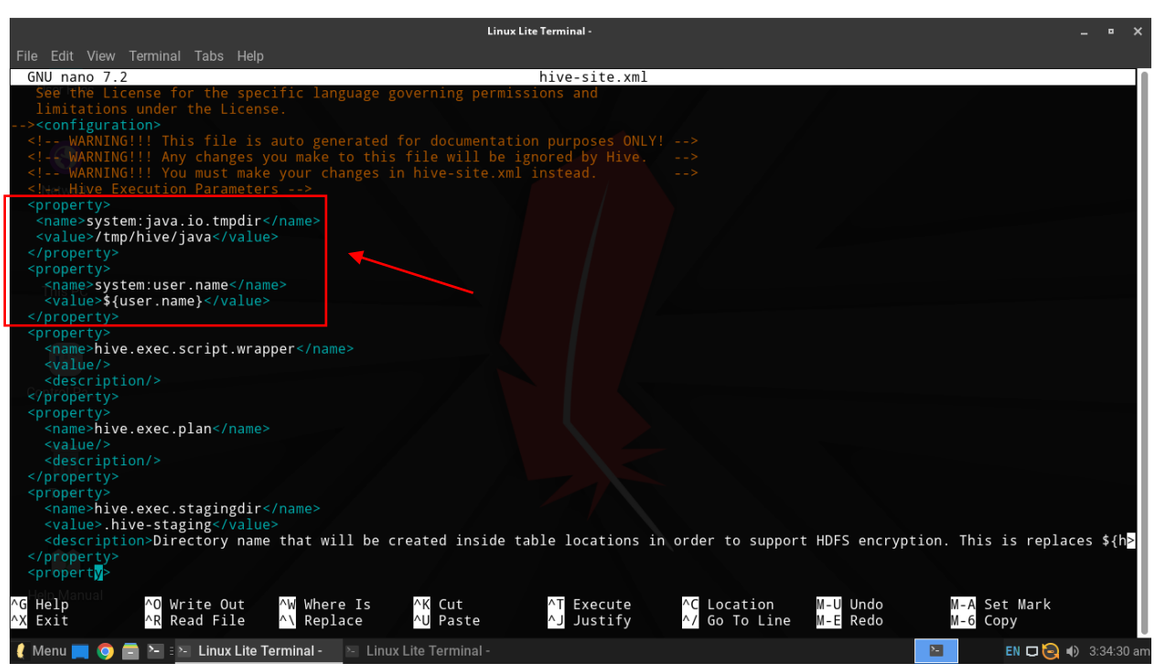
We also needed to verify the status of the MapReduce services by typing the `yarn jar /usr/local/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar pi 10 100` command. It will run a sample process of computing pi from Map #0 to Map #09 to start sample job along with the logs output. It is important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



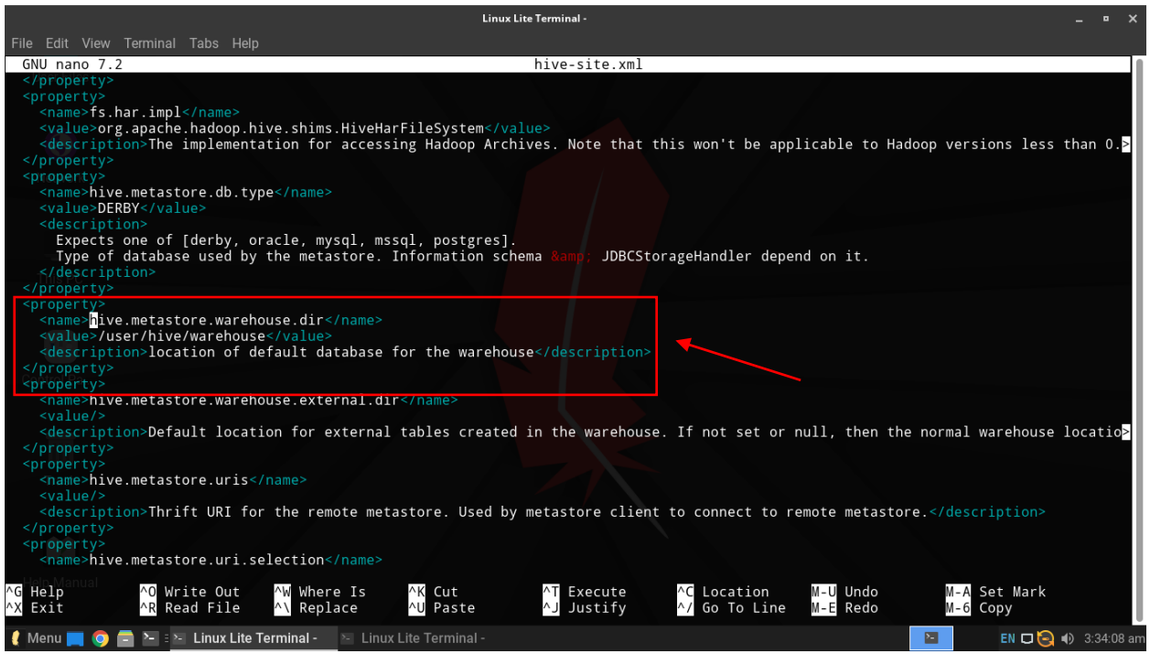
After checking the status of the MapReduce services, we can finally download the installation tarball file from the official Apache Software Foundation website using the `wget https://archive.apache.prg/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz` command and extracting it in our current directory 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.



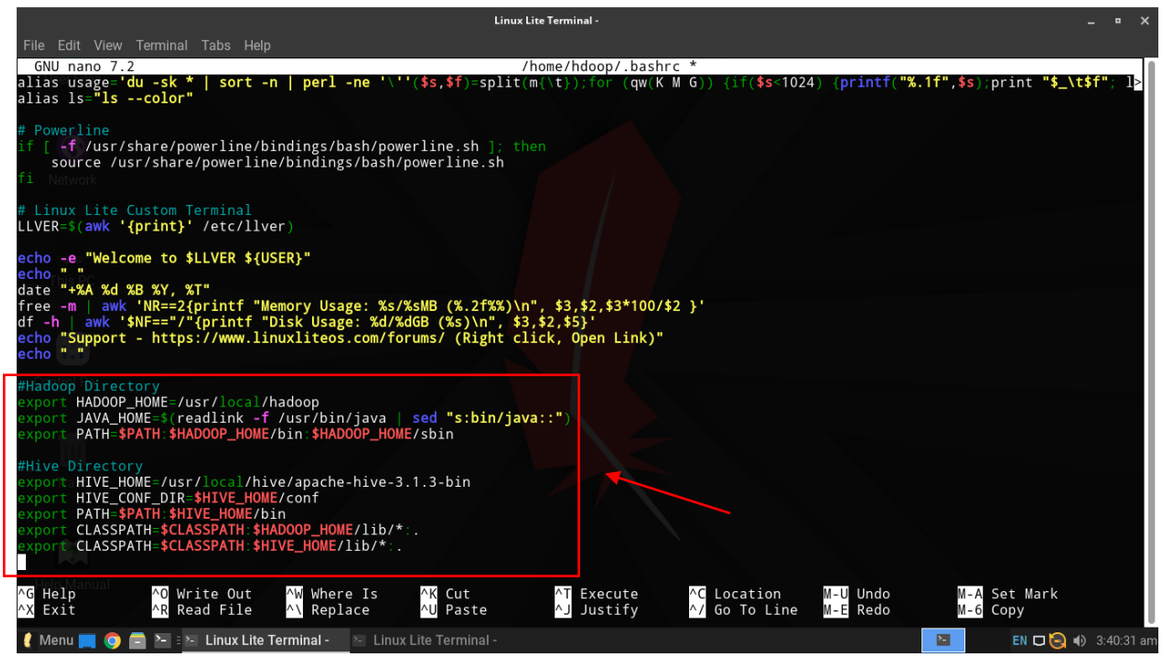
Inside the hive-site.xml file, we added two important configurations that are important for the proper functioning of Apache Hive. We defined a temporary directory where Hive can store intermediate data during query execution by adding the property `hive.exec.scratchdir` which we set to a directory as `/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.



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.

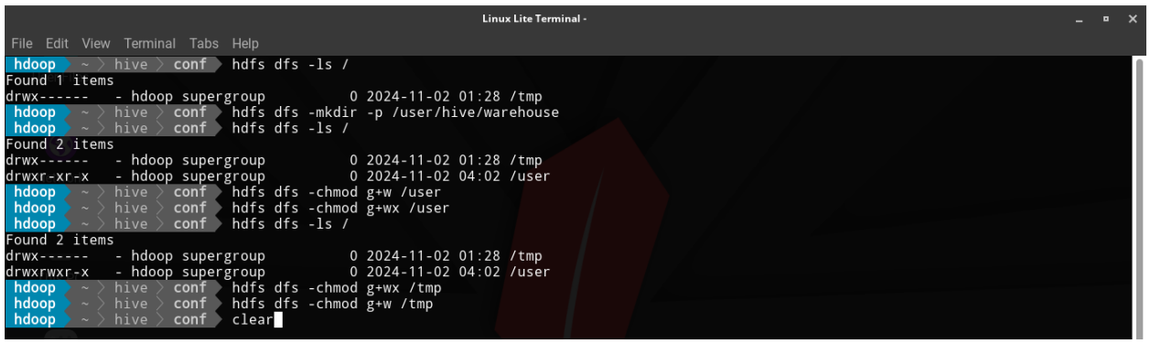


Using the `nano ~/.bashrc` command, we also need to set the proper Hive directory environment variables. It is one of the most important steps during the configuration process because it confirms that the system recognizes the Hive and Hadoop installations which will allow us to run Hive commands directly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

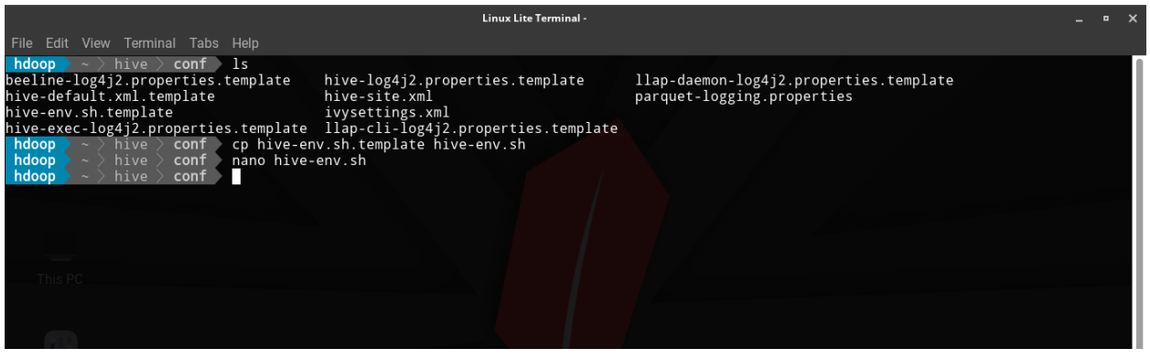


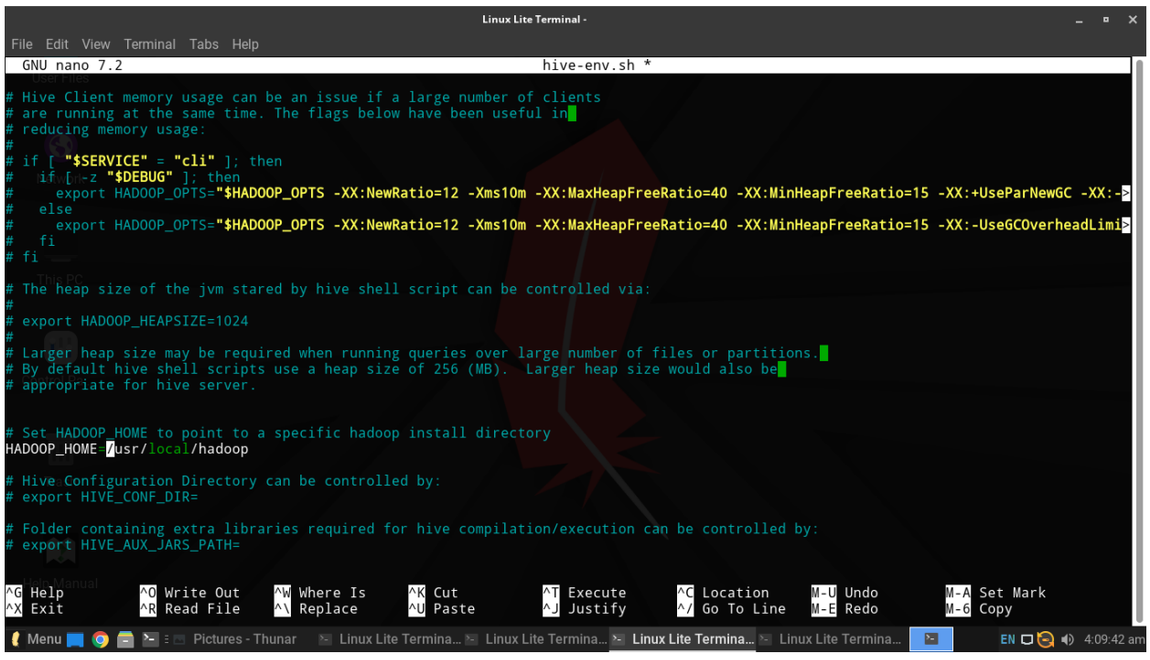
Next, we opened another terminal session as the “hdoop” user to create the necessary directories for Hive on HDFS. We created the Hive warehouse directory on HDFS using the `hdfs dfs -mkdir -p /user/hive/warehouse` command and verified the creation of the directory using the `hdfs dfs -ls /` command.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

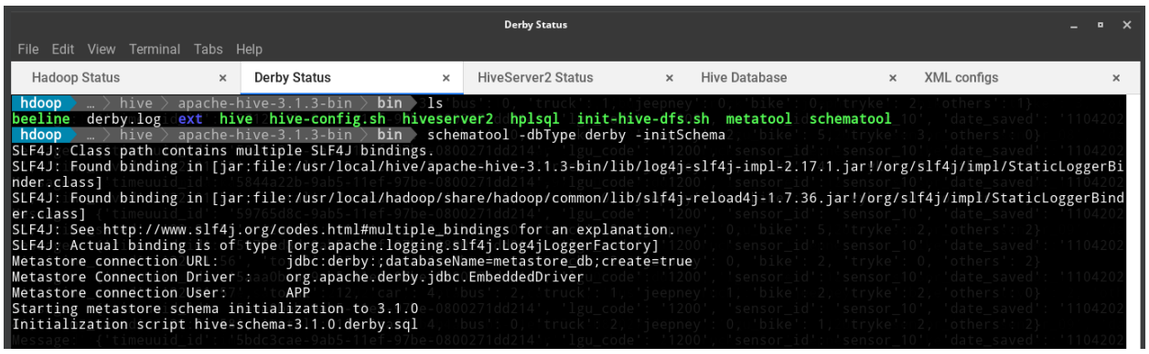


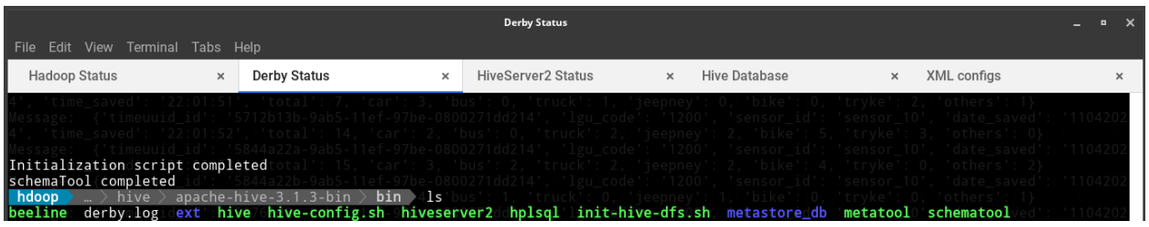
For the last configuration, we also copied the hive-env.sh template to create our own executable copy of the `hive-env.sh` file before editing its contents using the `nano hive-env.sh` command. Inside the `hive-env.sh` file, we commented out the `HADOOP_HOME` directory and set it as `/usr/local/hadoop`. 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.



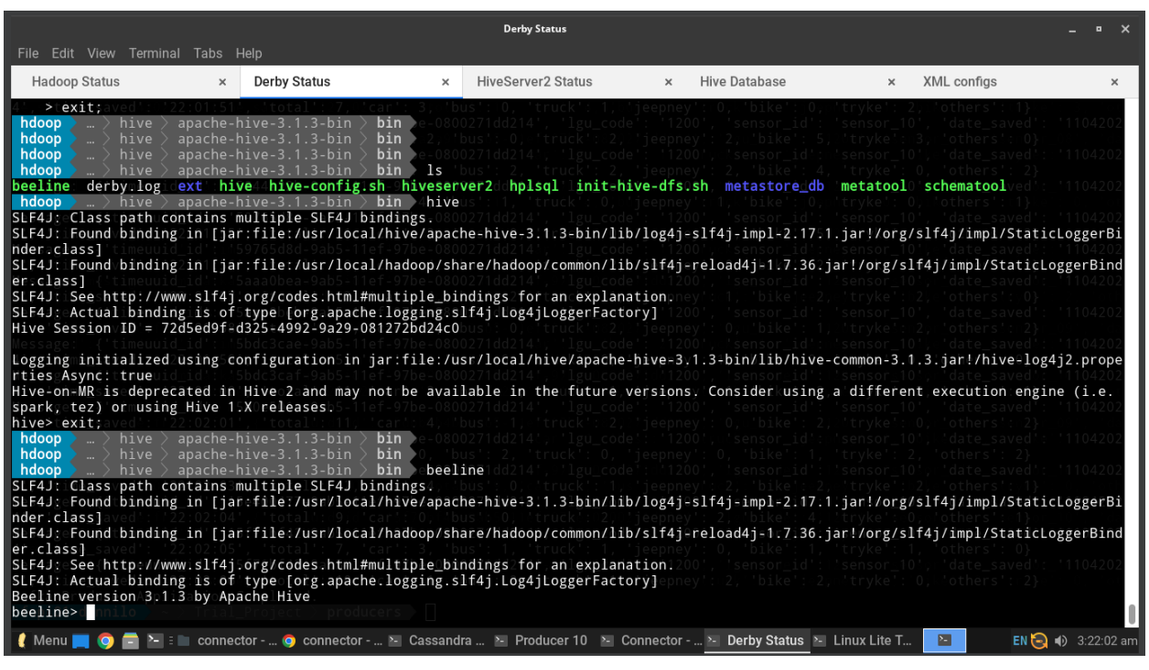


We can already proceed with initializing the Derby schema which is necessary for Hive to function correctly with its “metastore” by running a script provided in the Hive bin directory. This script sets up the Apache Derby database that Hive uses as its default metastore when running in standalone mode. 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.





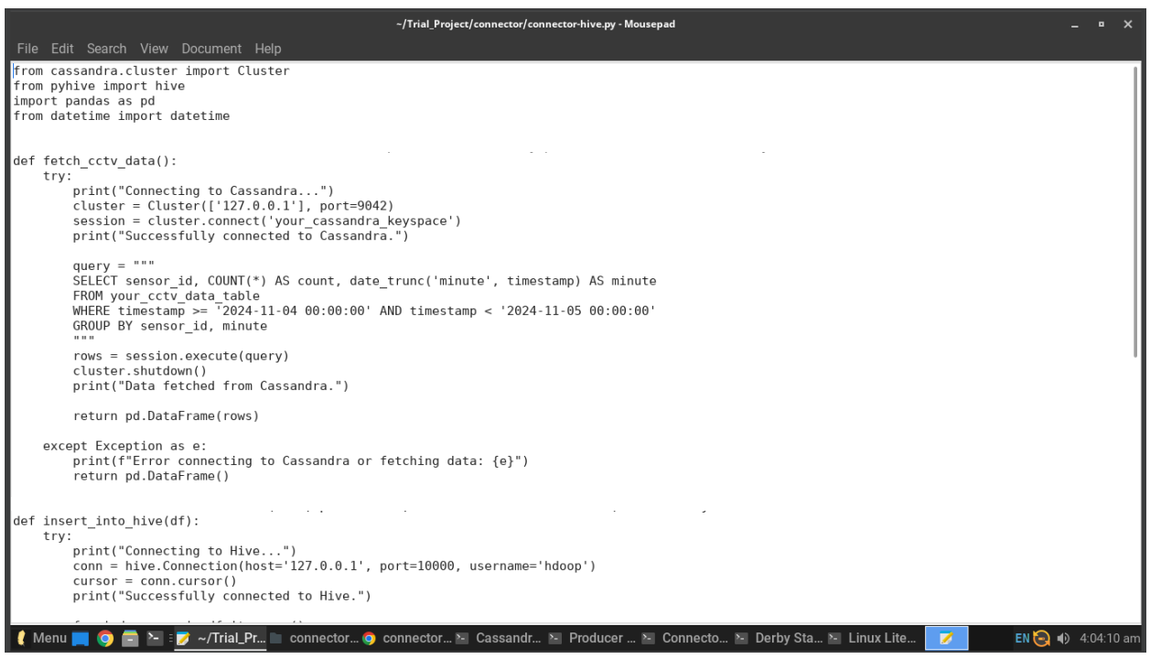
Since the initialization process is complete and we already have the `metastore_db` file, we can already access the Hive database through the command-line interface (CLI) by simply typing the “hive” command. It would launch the Hive shell where we can run HiveQL commands to create databases, tables, 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.



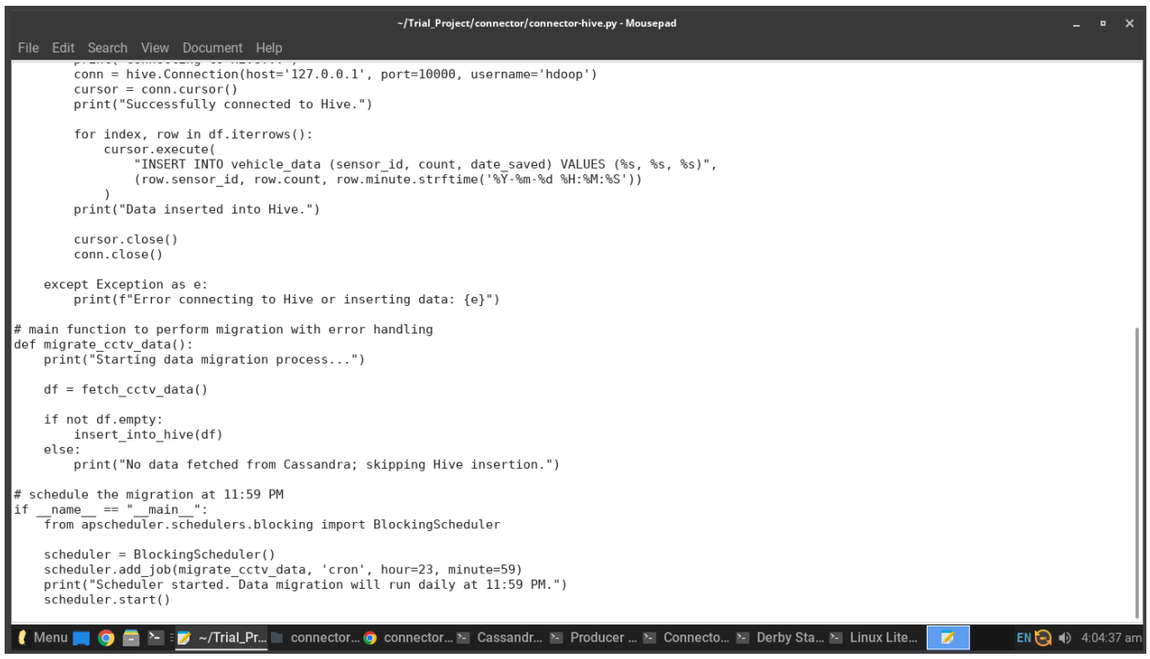
PYTHON SCRIPT (FOR THE MIGRATION PROCESS)

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.

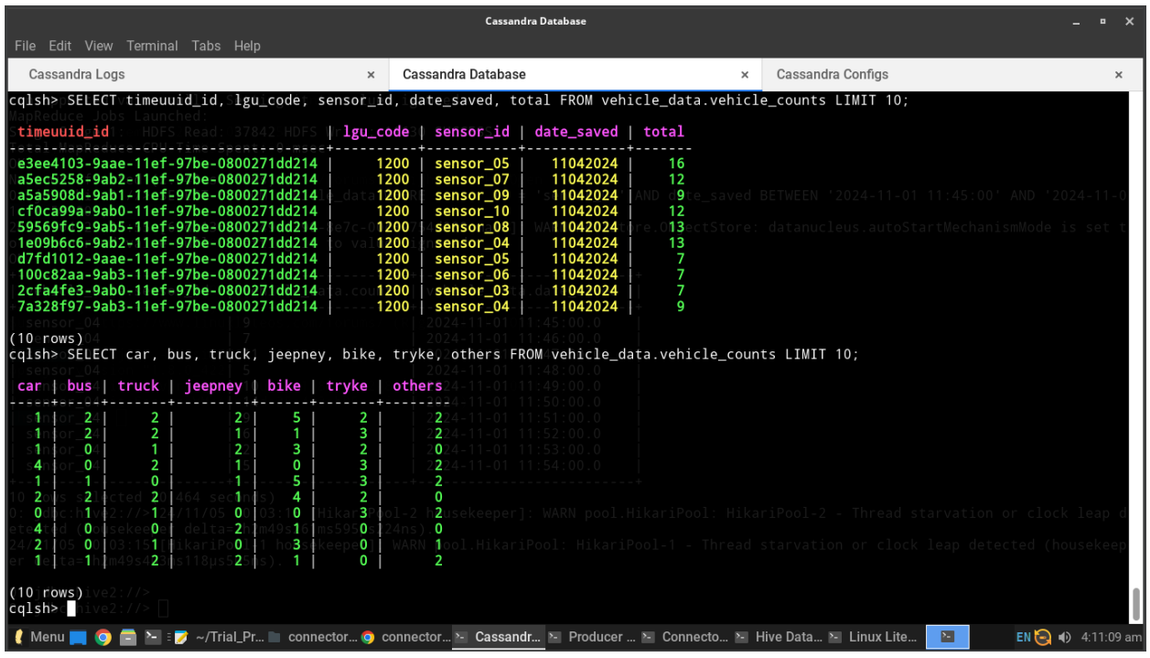


On the other hand, the migrate_cctv_data() function continues the process by first fetching data from Cassandra and then inserting it into Hive if the DataFrame is not empty. It makes sure that the script only attempts to insert data into Hive if there is valid data to process to prevent unnecessary 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.



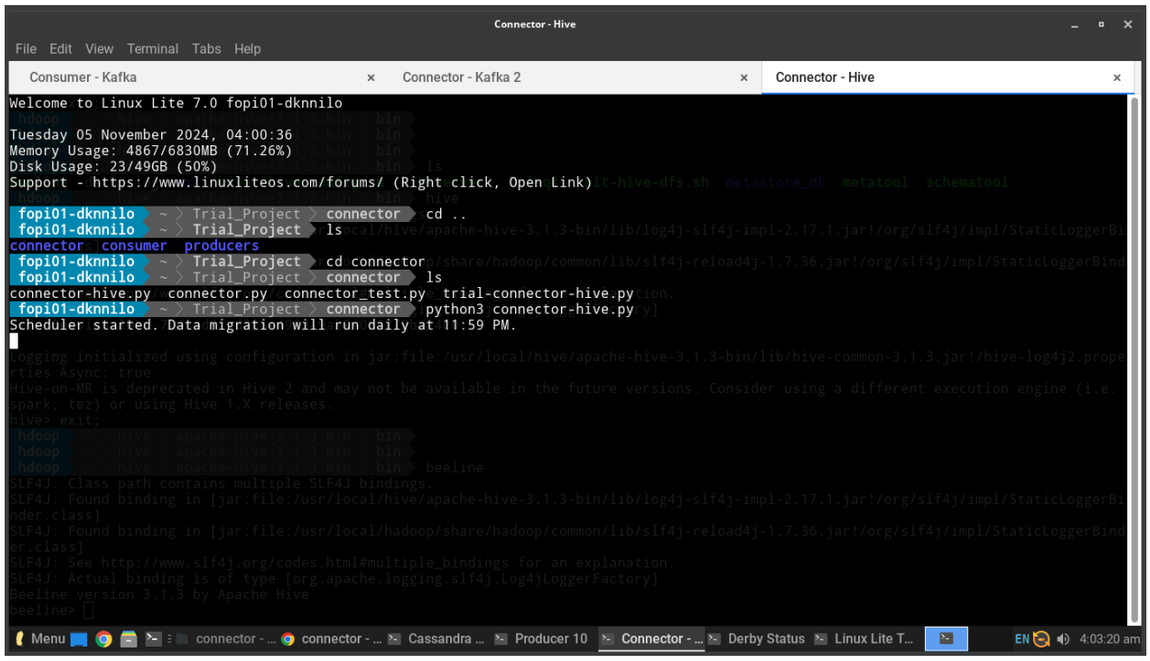
TEST RUN (FOR THE MIGRATION PROCESS)

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.

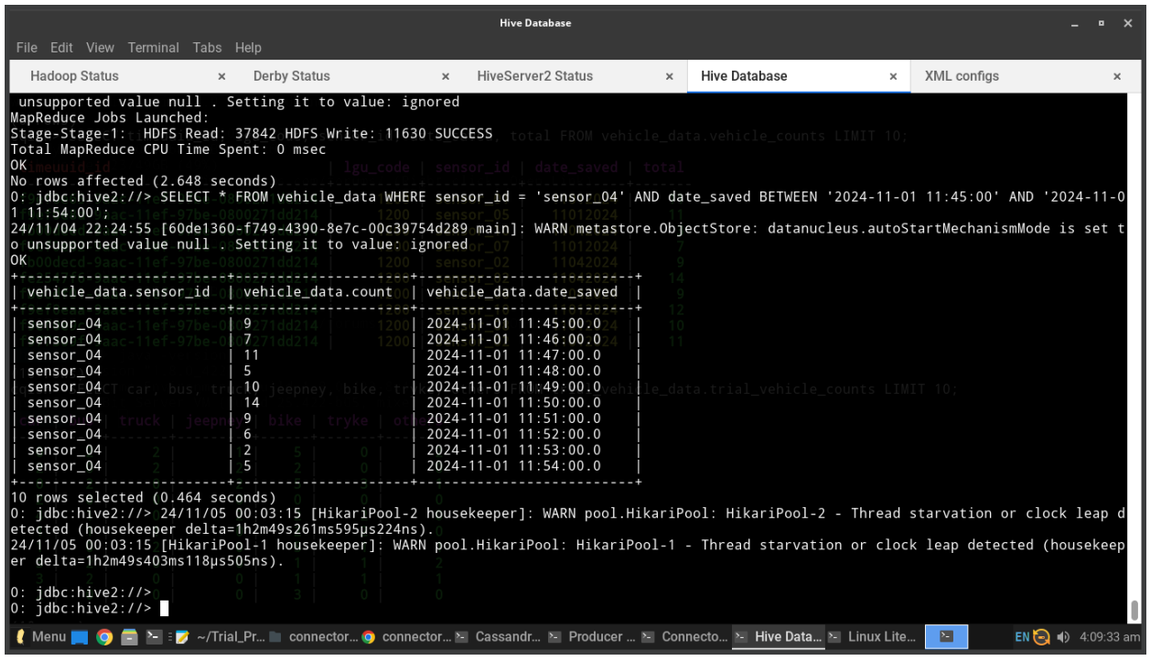


After checking the Cassandra database for content, we can already migrate the data from Cassandra to Hive. We opened another terminal and went to the connector directory of our trial project before looking at our list of connectors using the `ls` command. We started the script

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.

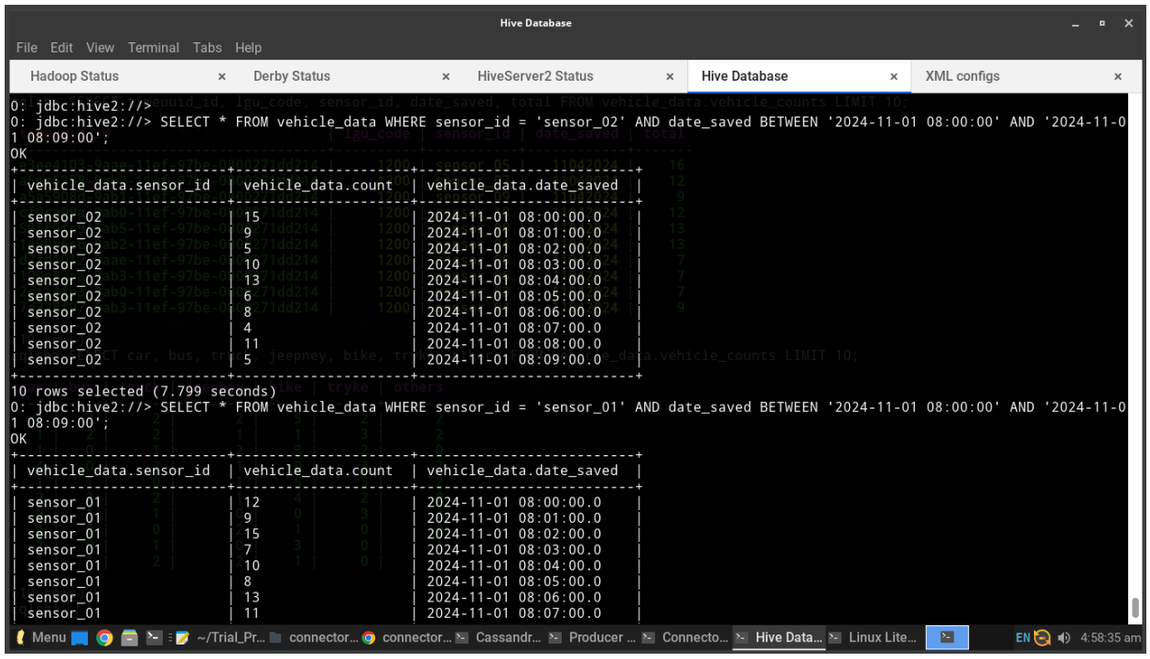


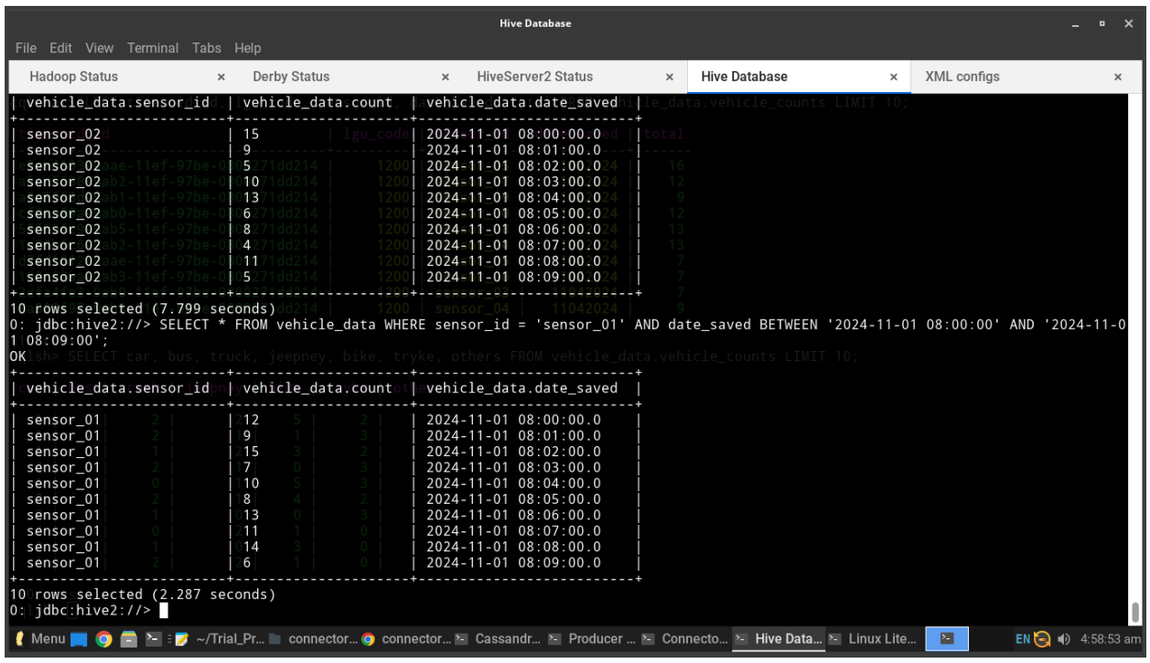
After waiting for at least six minutes, we prompted the Hive Database using the `SELECT * FROM vehicle_data WHERE sensor_id = 'sensor_04' AND date_saved BETWEEN '2024-11-01 11:45:00' AND '2024-11-01 11:54:00';` command through the "beeline" CLI to print out some data from sensor 4 before the end of the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