Cost Optimization and Cluster Management

Rohan Chaudhari
(chaudharirohan24@gmail.com)

Vishal Jha
(vishaljhavj786@gmail.com)

Table of Contents

Cost Optimization and Cluster Management

1) **Project Statement**:
   Create a project that optimizes costs and manages Databricks clusters efficiently using PySparkSQL. Implement auto-scaling policies, optimize query performance, and monitor resource utilization.

2) **Project Overview**:
   The primary goal of this project is to achieve cost optimization and efficient cluster management in Databricks by implementing auto-scaling policies, optimizing query performance, and monitoring resource utilization. By following these steps, the project will help reduce costs, improve performance, and ensure the efficient use of resources in Databricks clusters.

3) **Project Requirements**:
   a. Azure Subscription
      - You’re required to have an Azure Subscription to perform this project
   b. Azure Databricks
      - Azure Databricks is like a super-smart workspace in the cloud where you can easily analyse and process large amounts of data using the power of Apache Spark.
   c. Cluster
      - A cluster refers to a group of virtual or physical machines (nodes) that work together to process and analyze large volumes of data. Clusters are essential for handling the computational and storage requirements of big data workloads.
   d. Auto Scale
      - Auto Scale is a feature that automatically adjusts the number of compute resources (such as virtual machines or instances) in a cluster based on the workload. It helps optimize resource usage and ensures that the cluster has enough capacity to handle varying levels of demand without manual intervention.

e. Databricks Notebook

- ❖ Azure Databricks Notebooks are interactive, collaborative environments for data scientists and engineers to explore, visualize, and analyze data using languages like Python, Scala, SQL, and R. They integrate seamlessly with Azure services and provide built-in support for Apache Spark, enabling scalable data processing and machine learning workflows.

f. SparkSQL

- ❖ SparkSQL is a module in Apache Spark that provides a programming interface for working with structured data. It allows users to run SQL queries and access data using the DataFrame API, enabling seamless integration of SQL queries with Spark's distributed processing capabilities.

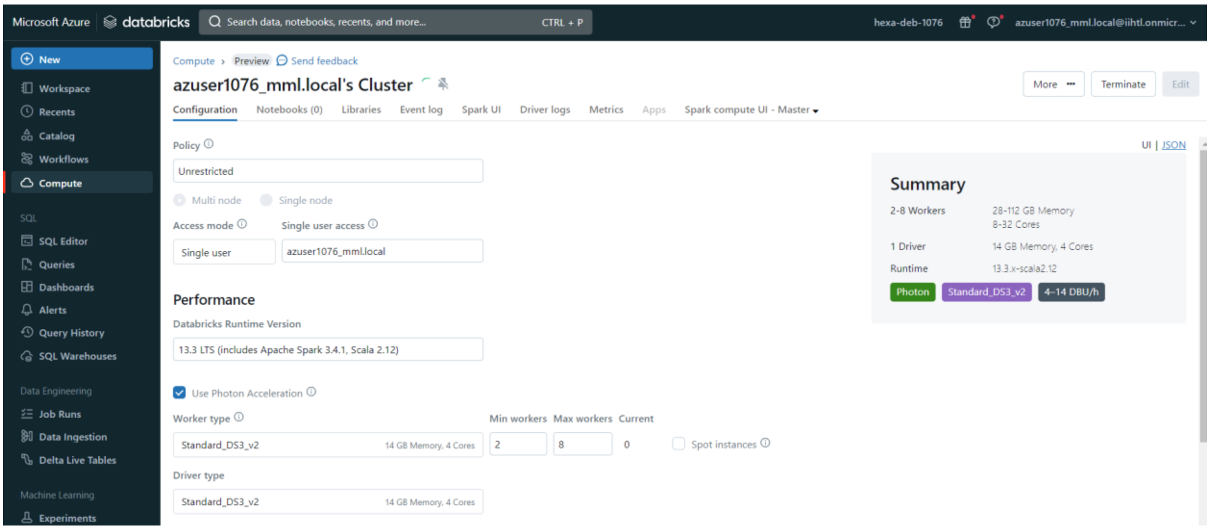
4) Architecture Diagram:

5) Execution Overview:

- ➢ Use Databricks' auto-scaling feature to automatically adjust the number of worker nodes in your cluster based on workload.
- ➢ Configure auto-scaling policies to define the conditions under which the cluster should scale up or down.
- ➢ Use best practices for optimizing PySparkSQL queries, such as using appropriate data formats and caching intermediate results.
- ➢ Monitor query performance using Databricks' query execution plans and optimize queries accordingly.
- ➢ Use of spark UI to monitor number of nodes , executor to check each and every execution done by nodes and use of metrics to graphical representation of CPU utilisation, how much Memory is occupied and how much Memory is allocated to cache, etc

6) Project Implementation

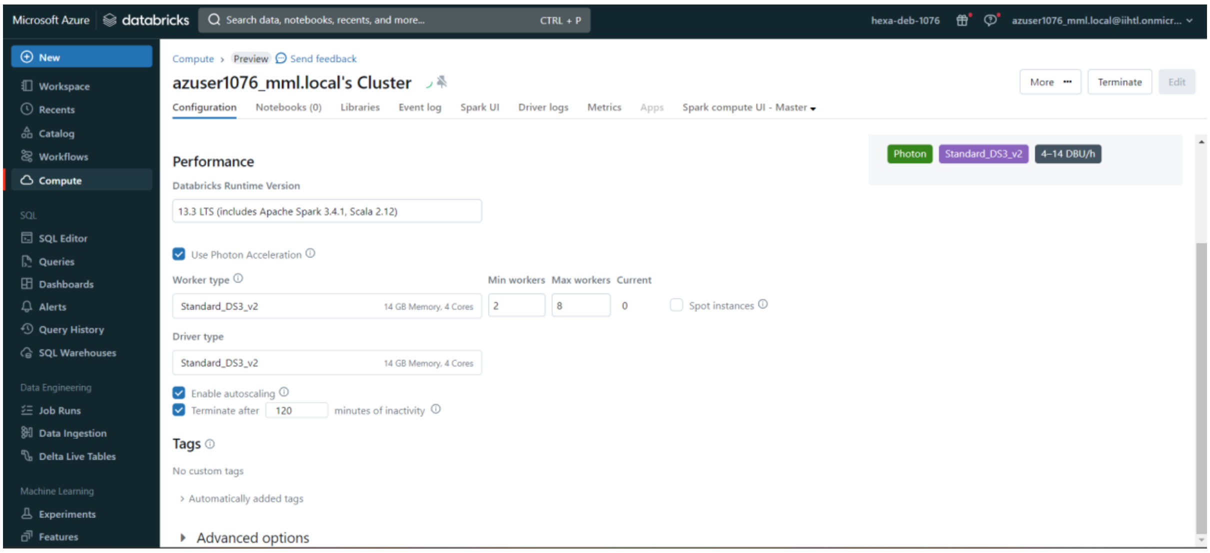
6.1) Creating Cluster:



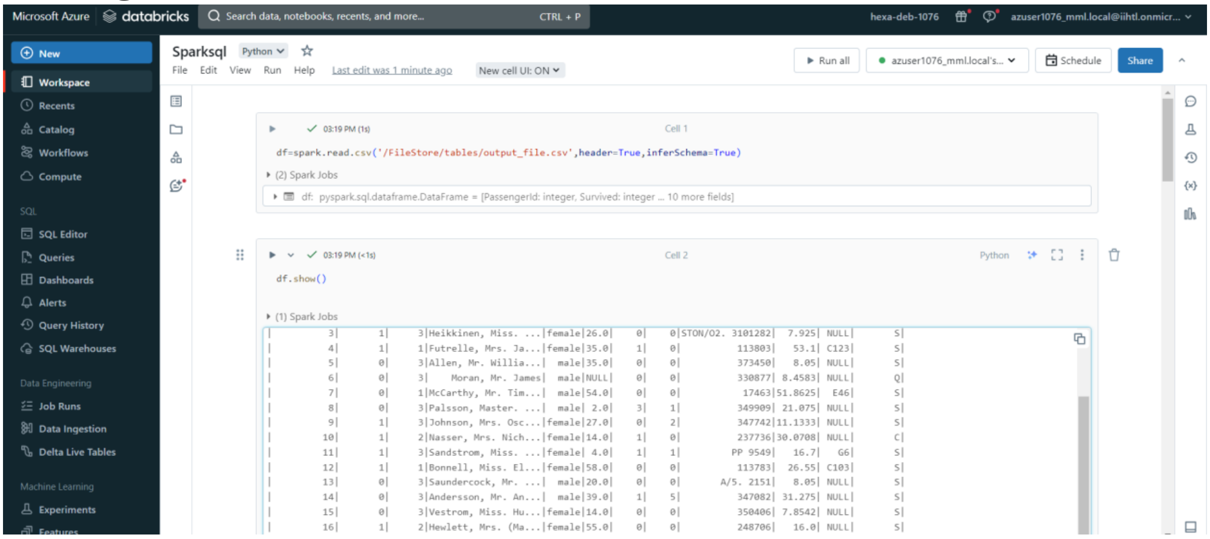
Enabling Autoscaling:

Auto Scale is a feature that automatically adjusts the number of compute resources (such as virtual machines or instances) in a cluster based on the workload. It helps optimize resource usage and ensures that the cluster has enough capacity to handle varying levels of demand without manual intervention.

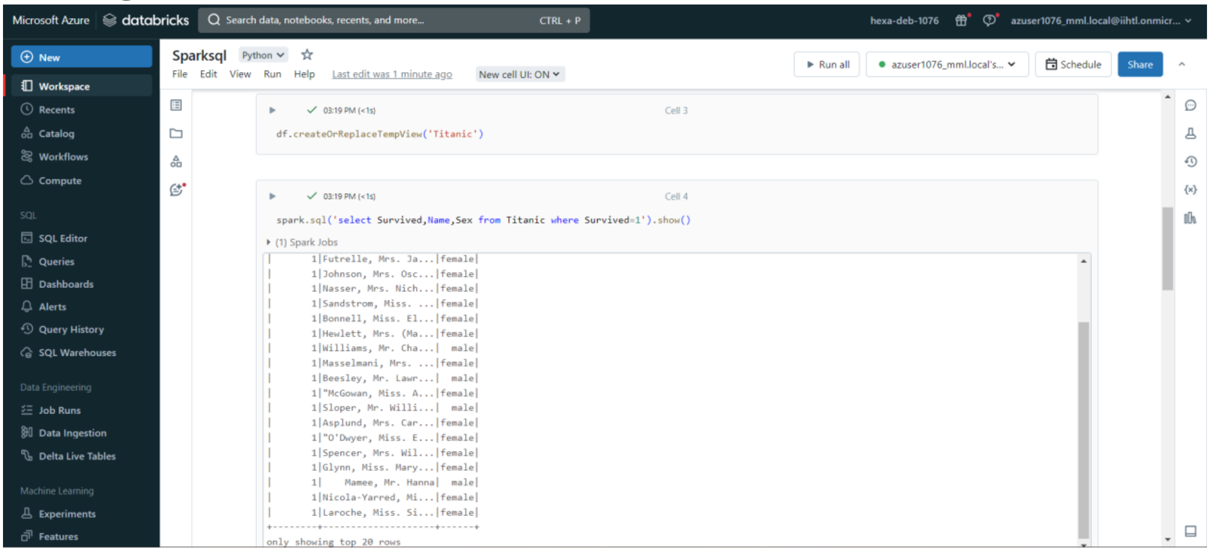
Here, we have assigned minimum nodes as 2 and maximum nodes as 8.



Reading Table:



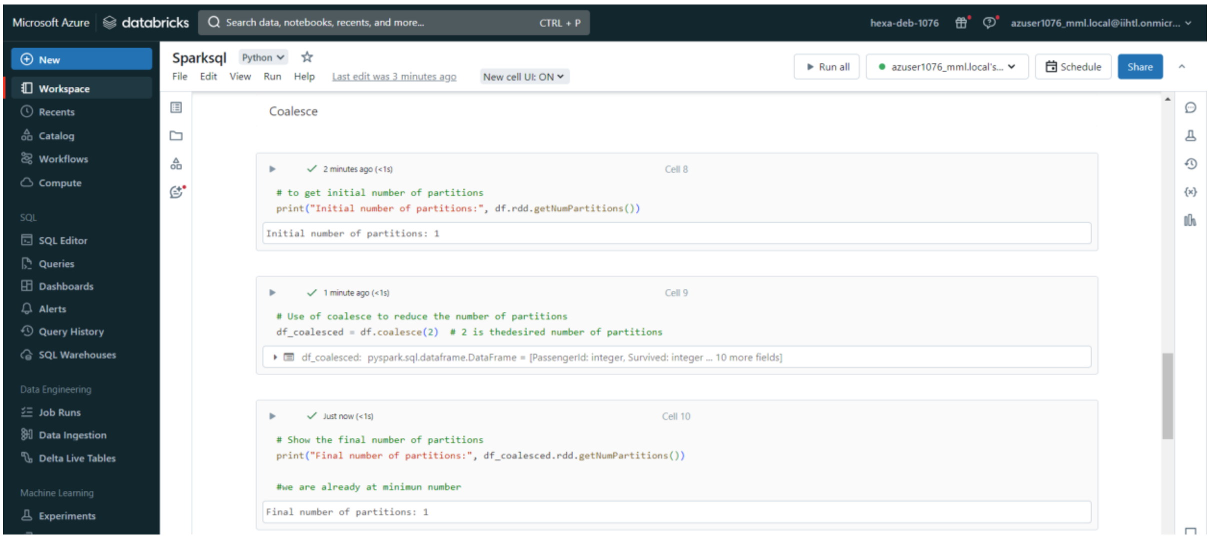
Creating View:



6.2) Spark SQL - Spark SQL will scan only required columns and will automatically tune compression to minimize memory usage.

Use of Coalesce : In Spark SQL, coalesce is a function that can be used to optimize queries by reducing the number of partitions in a DataFrame or RDD (Resilient Distributed Dataset).

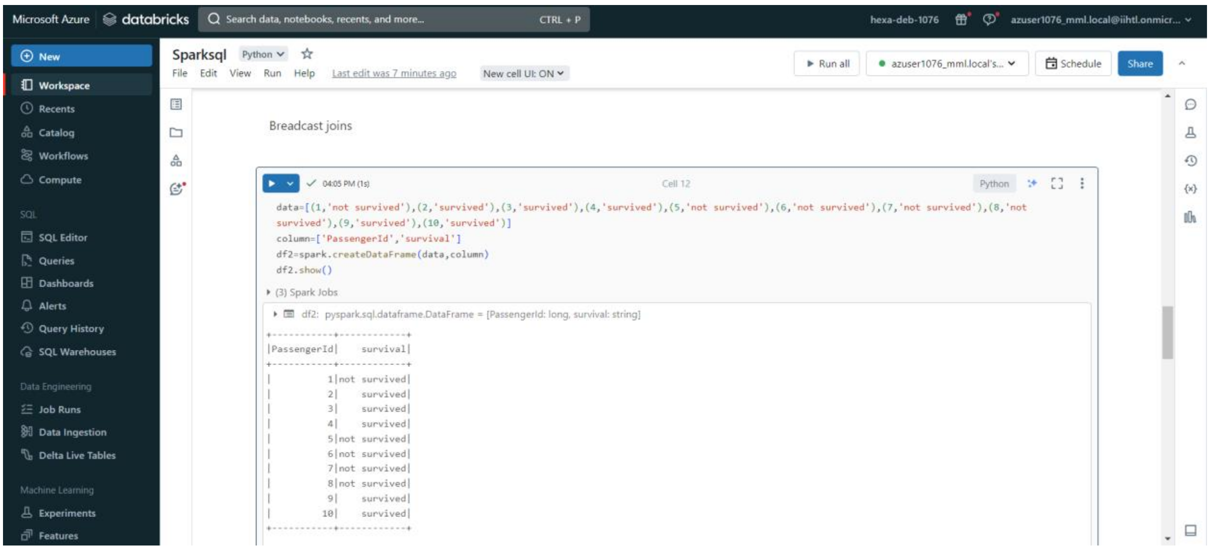
This can be particularly useful in scenarios where you want to minimize the number of partitions to improve query performance.



Broadcast join: In Spark SQL, a broadcast join can be used to optimize certain types of join operations by broadcasting the smaller DataFrame to all the worker nodes, avoiding shuffling data across the network.

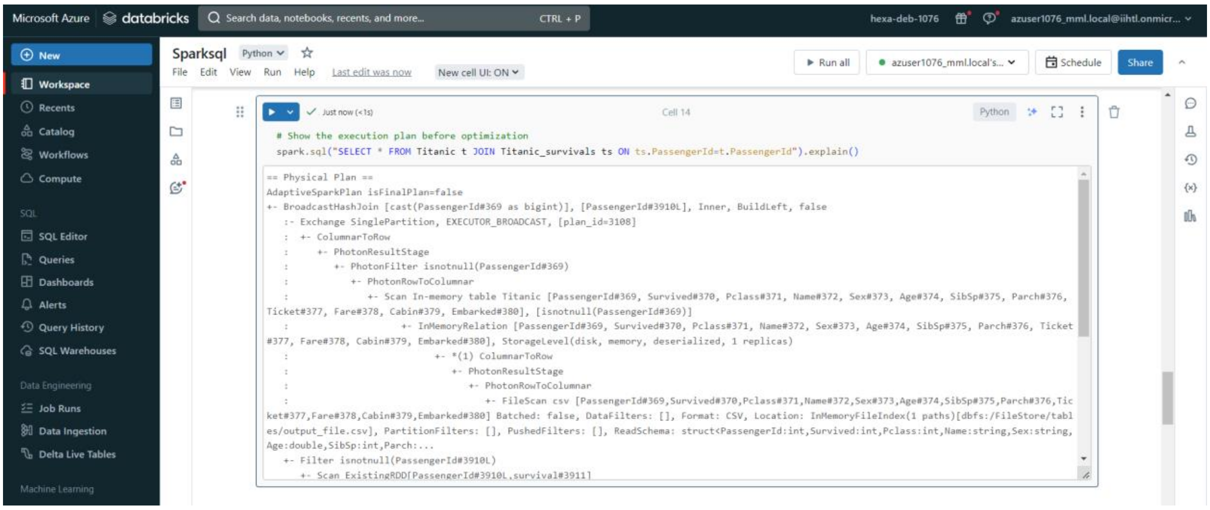
This is particularly useful when one of the DataFrames is small enough to fit in the memory of each executor.

Created new dataframe to perform joins:

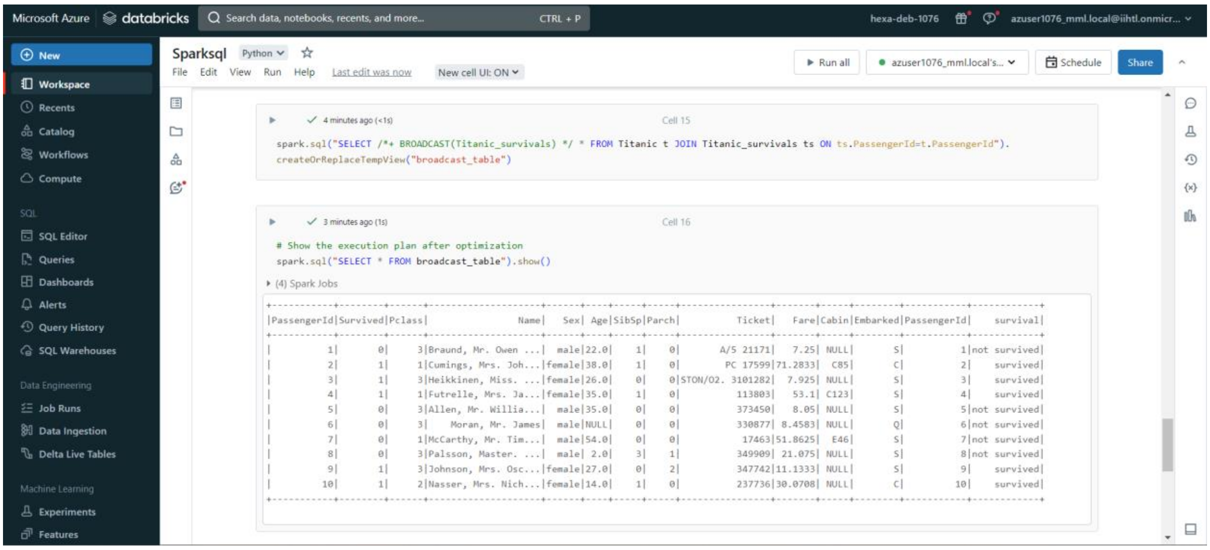


Creating View:

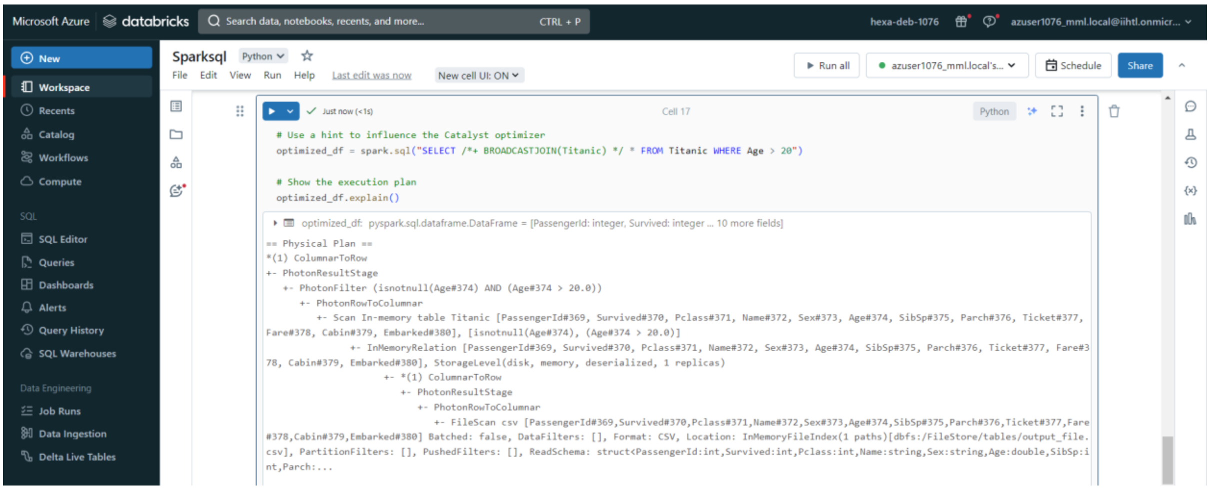
Execution of simple join:



Creating a broadcast join:



6.3) Catalyst Optimizer :In Spark SQL, the Catalyst optimizer is responsible for optimizing the logical and physical plans of queries. It's an integral part of Apache Spark, and we generally don't need to write code for it. However, you can influence the optimization process by using hints or by configuring Spark properties.



In this example, the /*+ BROADCASTJOIN(Titanic) */ hint is used within the SQL query to suggest the use of a broadcast join for better performance.

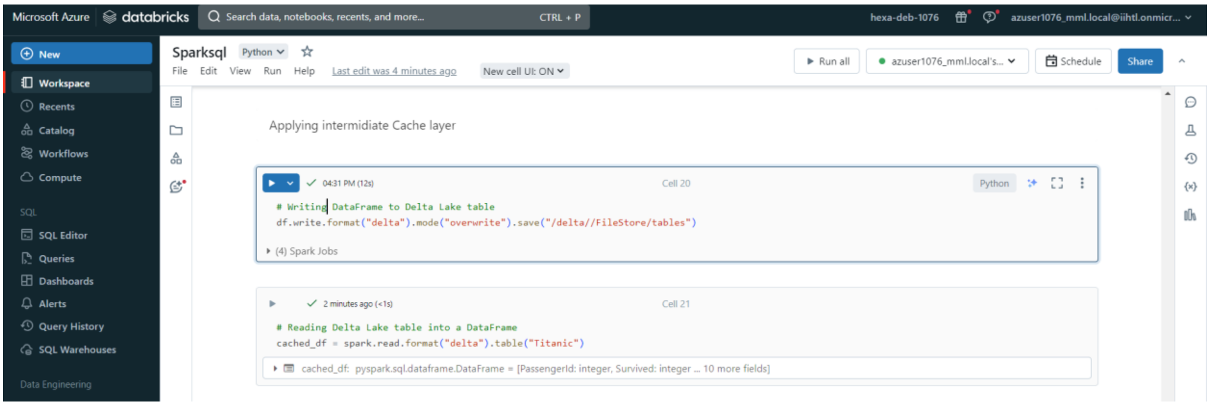
Configuring Spark properties to influence the optimizer's behavior:

We can adjust the spark.sql.autoBroadcastJoinThreshold property to control when Spark should automatically broadcast small tables.

Above code sets the threshold for automatically broadcasting tables smaller than 10 megabytes.

Intermediate Cache Layer: Creating an intermediate data cache layer in Spark SQL on Azure Databricks typically involves storing the intermediate results of Spark SQL queries in a persistent storage system.

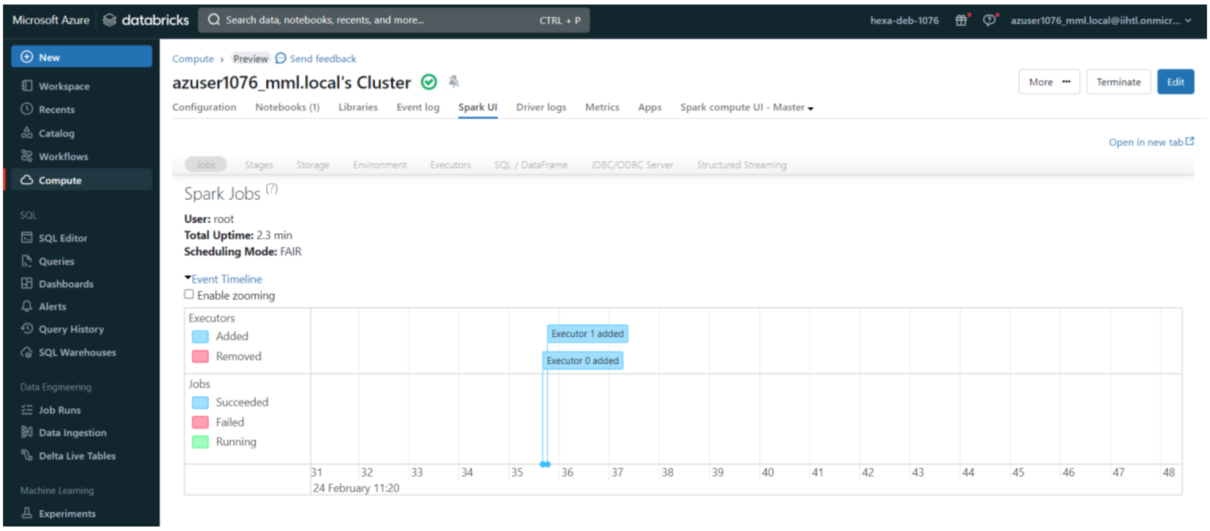
Creating a Delta Lake table to store the intermediate results.



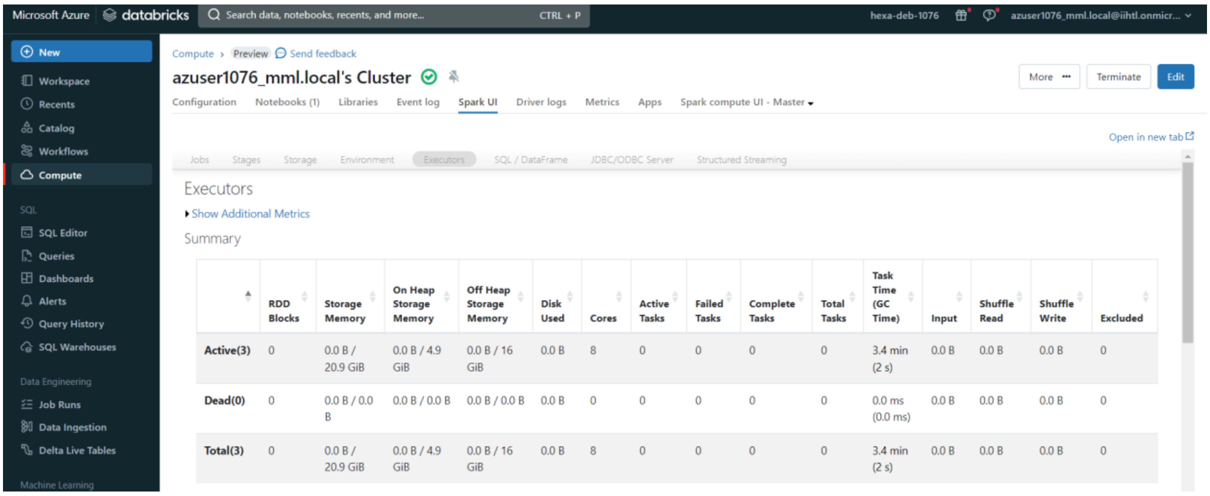
Performing Spark SQL queries & storing the intermediate results in the Delta Lake table

In This way, we're using Delta Lake as an intermediate data cache layer.

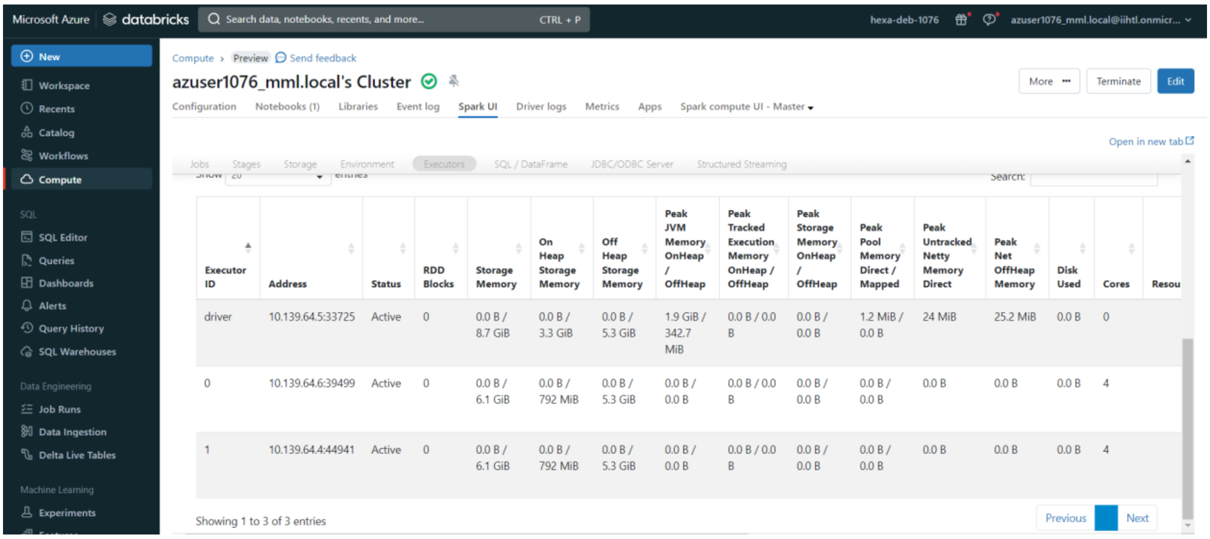
6.4) Monitoring in Spark UI:



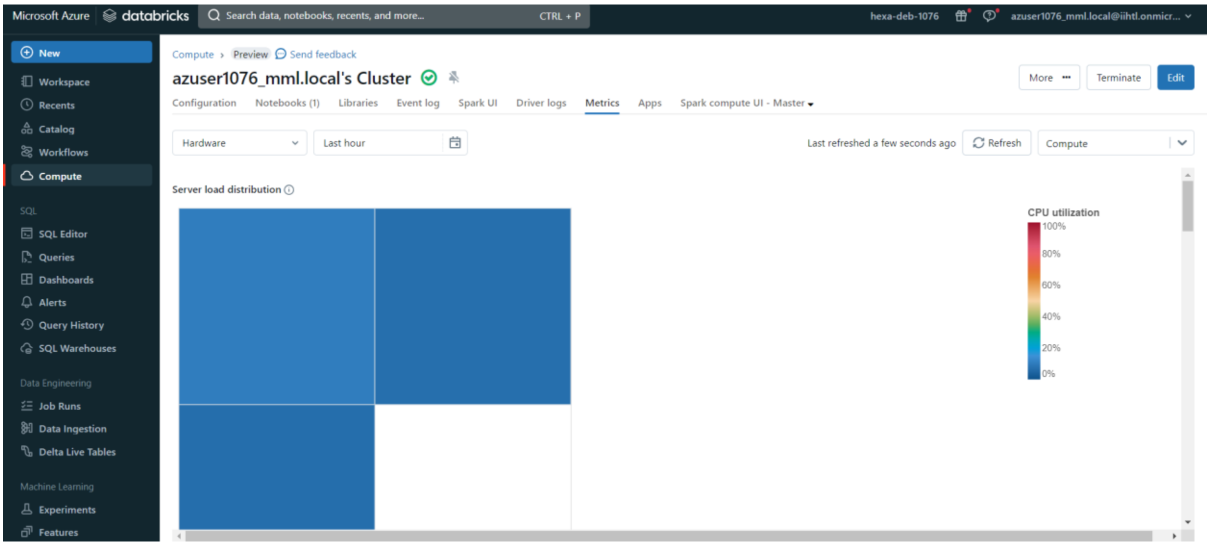
Executors:

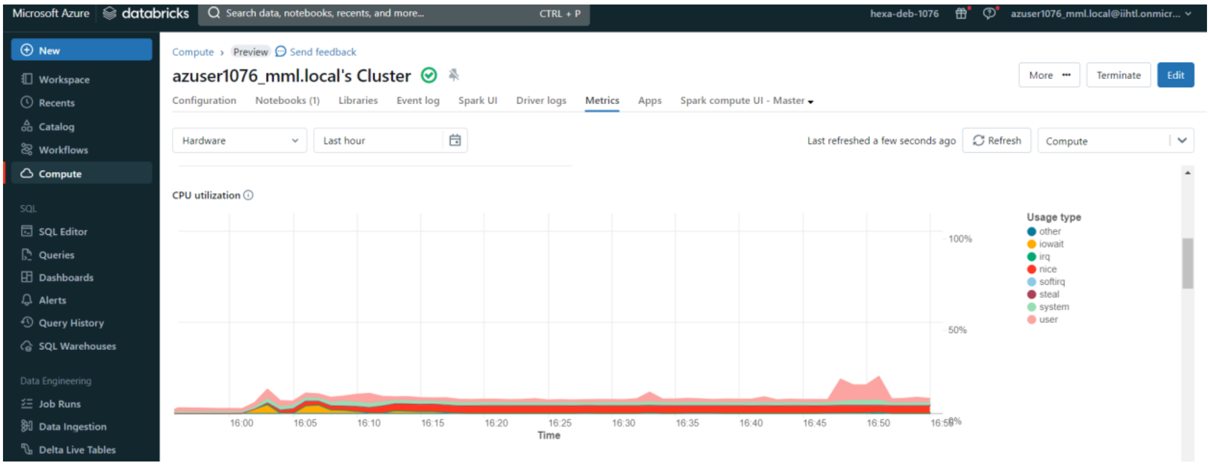


Details of executors:

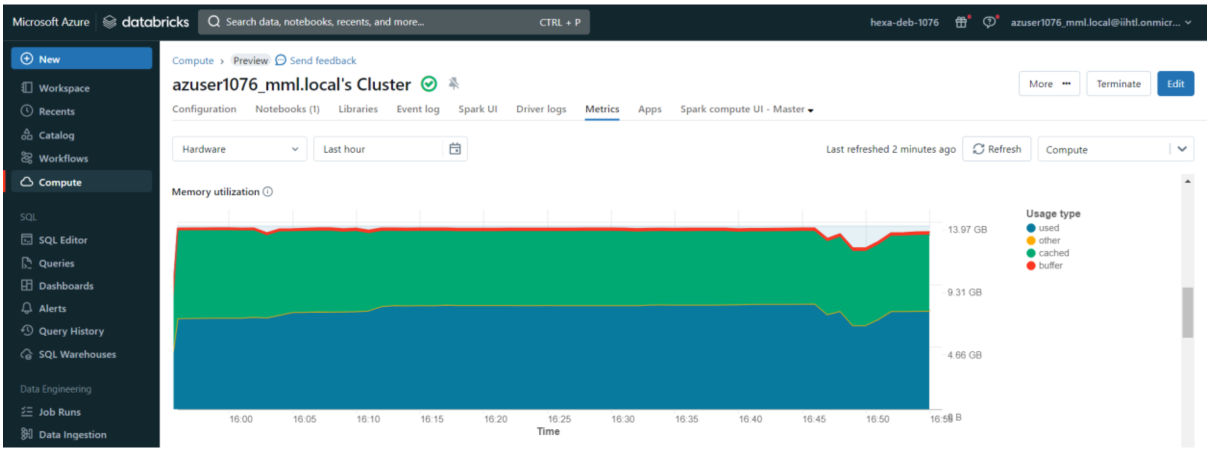


CPU utilization:

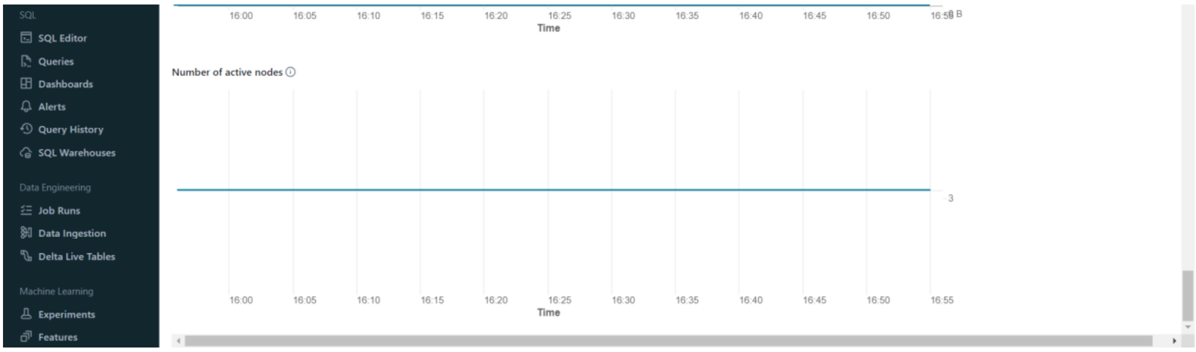




Memory Utilization:



Active nodes:



7) Conclusion:

This project successfully demonstrates the implementation of cost optimization and efficient cluster management in Databricks using PySparkSQL. By implementing auto-scaling policies, optimizing query performance, and monitoring resource utilization, the project showcases best practices for managing Databricks clusters effectively. The project's outcomes contribute to reducing operational costs, improving performance, and enhancing the overall management of Databricks clusters for data processing and analytics workflows.