

# **Cost Optimization and Cluster Management**

**Rohan Chaudhari**

([chaudharirohan24@gmail.com](mailto:chaudharirohan24@gmail.com))

**Vishal Jha**

([vishaljhavj786@gmail.com](mailto:vishaljhavj786@gmail.com))

## Table of Contents

S.no	Topic	Page No.
1	Project Statement	3
2	Project Overview	3
3	Project Requirements	3
3.1	Azure Subscription	3
3.2	Azure Databricks	3
3.3	Cluster	3
3.4	Autoscale	3
3.5	Databricks Notebook	4
3.6	SparkSQL	4
4	Architecture Diagram	4
5	Execution Overview	4
6	Project Implementation	5
6.1	Creating Cluster	5
6.2	Working with Spark SQL	6
6.3	Catalyst Optimizer	9
6.4	Monitoring in Spark UI	11
7	Conclusion	13

## 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.

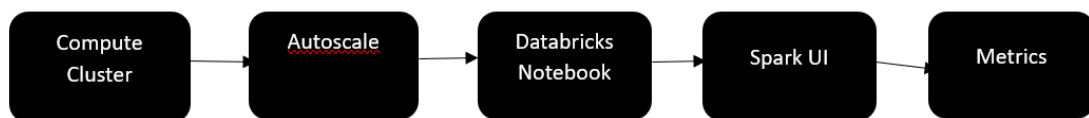
e. Databricks Notebook

- ❖ 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:

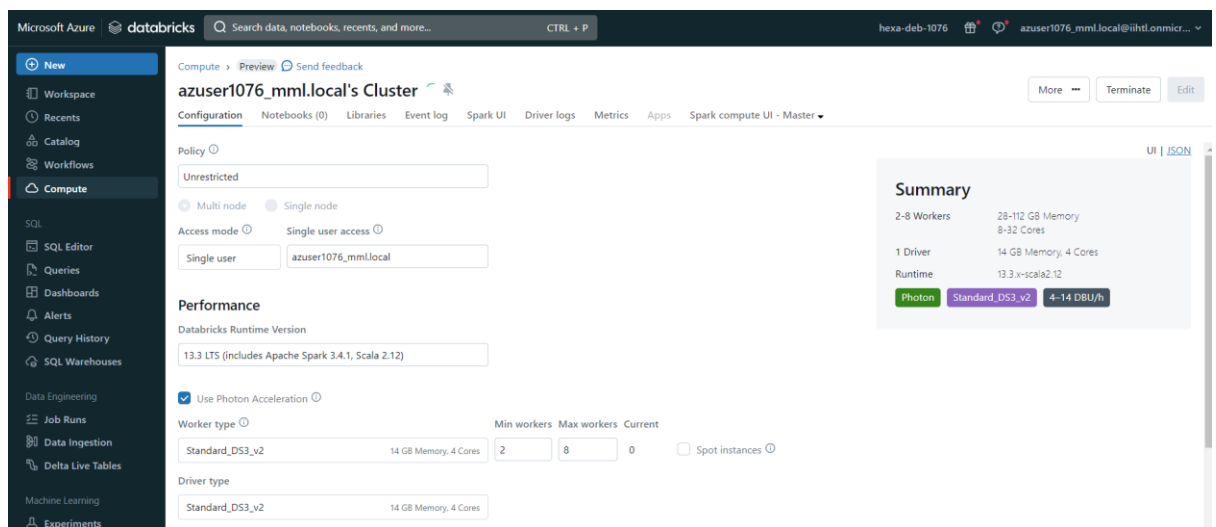


#### 5) Execution Overview:

- 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

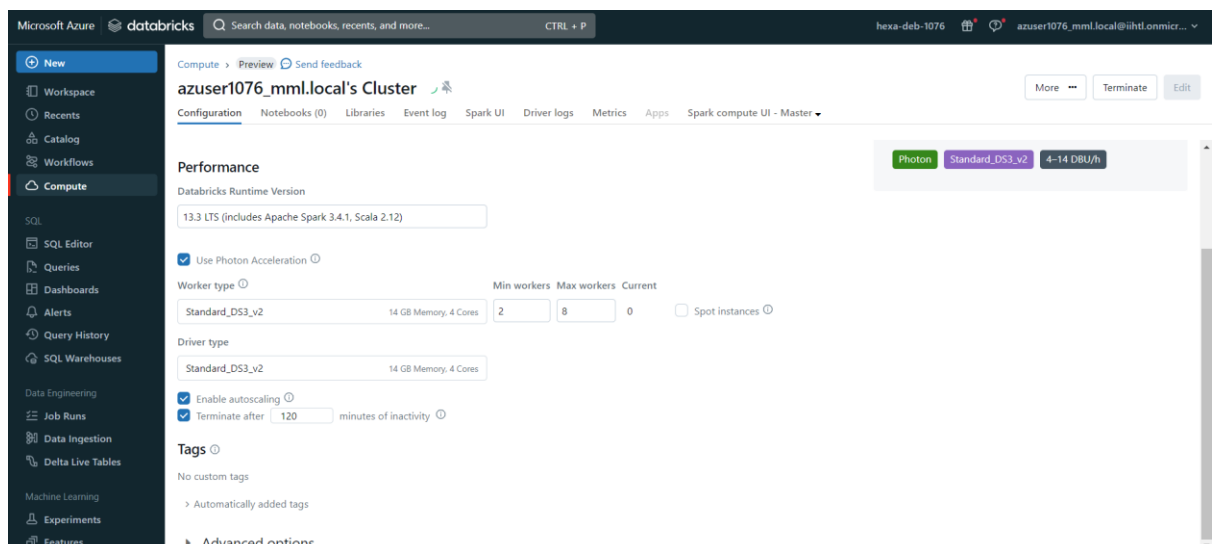
### 6.1) Creating Cluster:



#### 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:

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation options like Workspace, Recents, Catalog, Workflows, Compute, SQL, SQL Editor, Queries, Dashboards, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, Delta Live Tables, Machine Learning, Experiments, and Features. The main area displays a Spark SQL cell with the following code:

```
df=spark.read.csv('/FileStore/tables/output_file.csv',header=True,inferSchema=True)
```

The output shows the DataFrame schema: `DataFrame = [PassengerId: integer, Survived: integer ... 10 more fields]`. Below the code, the results of `df.show()` are displayed as a table with 16 rows and 10 columns.

PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
3	1	3 Heikkinen, Miss. ...	female	26.0	0	0	STON/O2. 3101282	7.925	NULL	S
4	1	1 Futrelle, Mrs. Ja...	female	35.0	1	0	113803	53.1	C123	S
5	0	3 Allen, Mr. Willia...	male	35.0	0	0	373450	8.05	NULL	S
6	0	3 Moran, Mr. James	male	NULL	0	0	330877	8.453	NULL	Q
7	0	1 McCarthy, Mr. Tim...	male	54.0	0	0	17463	51.8625	E46	S
8	0	3 Palsson, Master. ...	male	2.0	3	1	349909	21.075	NULL	S
9	1	3 Johnson, Mrs. Osc...	female	27.0	0	2	347742	11.1333	NULL	S
10	1	2 Nasser, Mrs. Nich...	female	14.0	1	0	237736	30.0708	NULL	C
11	1	3 Sandstrom, Miss. ...	female	4.0	1	1	PP 9549	16.7	G6	S
12	1	1 Bonnell, Miss. El...	female	58.0	0	0	113783	26.55	C103	S
13	0	3 Saunderscock, Mr. ...	male	20.0	0	0	A/5. 2151	8.05	NULL	S
14	0	3 Andersson, Mr. An...	male	39.0	1	5	347082	31.275	NULL	S
15	0	3 Vestrom, Miss. Hu...	female	14.0	0	0	350406	7.8542	NULL	S
16	1	2 Hewlett, Mrs. (Ma...	female	55.0	0	0	248706	16.0	NULL	S

## Creating View:

The screenshot shows the Databricks workspace interface. The left sidebar is the same as the previous screenshot. The main area displays a Spark SQL cell with the following code:

```
df.createOrReplaceTempView('Titanic')
```

Below the code, the results of `spark.sql('select Survived,Name,Sex from Titanic where Survived=1').show()` are displayed as a table with 16 rows and 3 columns.

Survived	Name	Sex
1	1 Futrelle, Mrs. Ja...	female
1	3 Johnson, Mrs. Osc...	female
1	2 Nasser, Mrs. Nich...	female
1	3 Sandstrom, Miss. ...	female
1	1 Bonnell, Miss. El...	female
1	2 Hewlett, Mrs. (Ma...	female
1	1 Williams, Mr. Cha...	male
1	1 Hassellman, Mrs. ...	female
1	1 Beesley, Mr. Lawr...	male
1	1 McGowan, Miss. A...	female
1	1 Sloper, Mr. Willi...	male
1	1 Asplund, Mrs. Car...	female
1	1 O'Dwyer, Miss. E...	female
1	1 Spencer, Mrs. Wil...	female
1	1 Glyn, Miss. Mary...	female
1	1 Hamee, Mr. Hanna	male
1	1 Nicola-Yarred, Mi...	female
1	1 Laroche, Miss. Si...	female

only showing top 20 rows

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

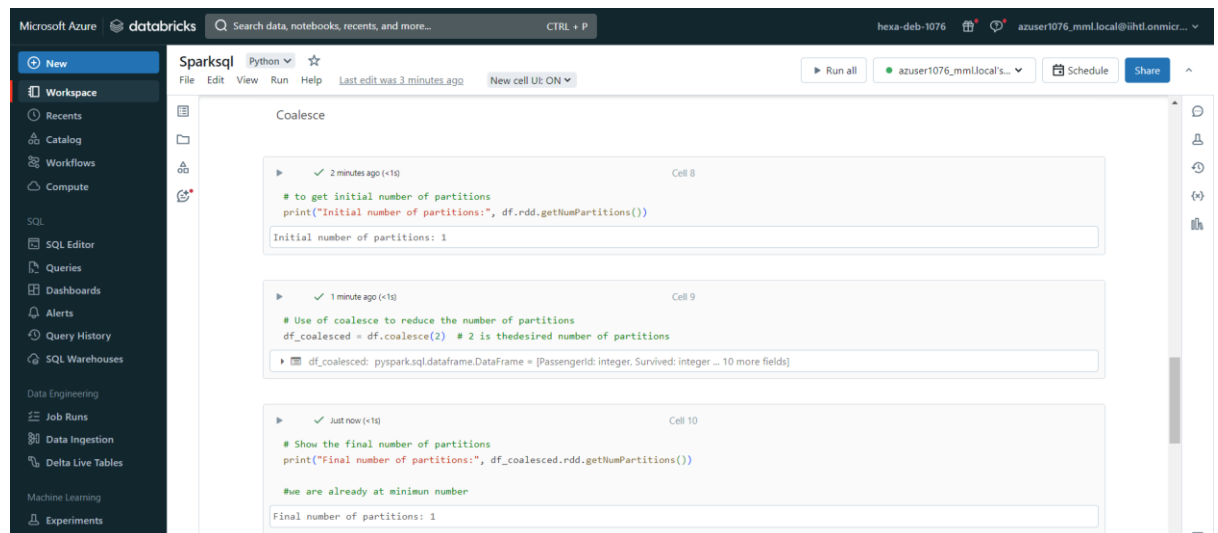
The screenshot shows the Databricks workspace interface. The left sidebar is the same as the previous screenshots. The main area displays a Spark SQL cell with the following code:

```
# Cache the table
df.cache()
```

The output shows the DataFrame schema: `DataFrame[PassengerId: int, Survived: int, Pclass: int, Name: string, Sex: string, Age: double, SibSp: int, Parch: int, Ticket: string, Fare: double, Cabin: string, Embarked: string]`.

**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.



```
Microsoft Azure databricks Search data, notebooks, recent, and more... CTRL + P hexa-deb-1076 azuser1076_mml.local@iitl.onmicr...

New Workspace Recents Catalog Workflows Compute SQL SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Data Engineering Job Runs Data Ingestion Delta Live Tables Machine Learning Experiments

Sparksq Python File Edit View Run Help Last edit was 3 minutes ago New cell UI: ON Run all azuser1076_mml.local's... Schedule Share

Coalesce

2 minutes ago (<1s) Cell 8
# to get initial number of partitions
print("Initial number of partitions:", df.rdd.getNumPartitions())
Initial number of partitions: 1

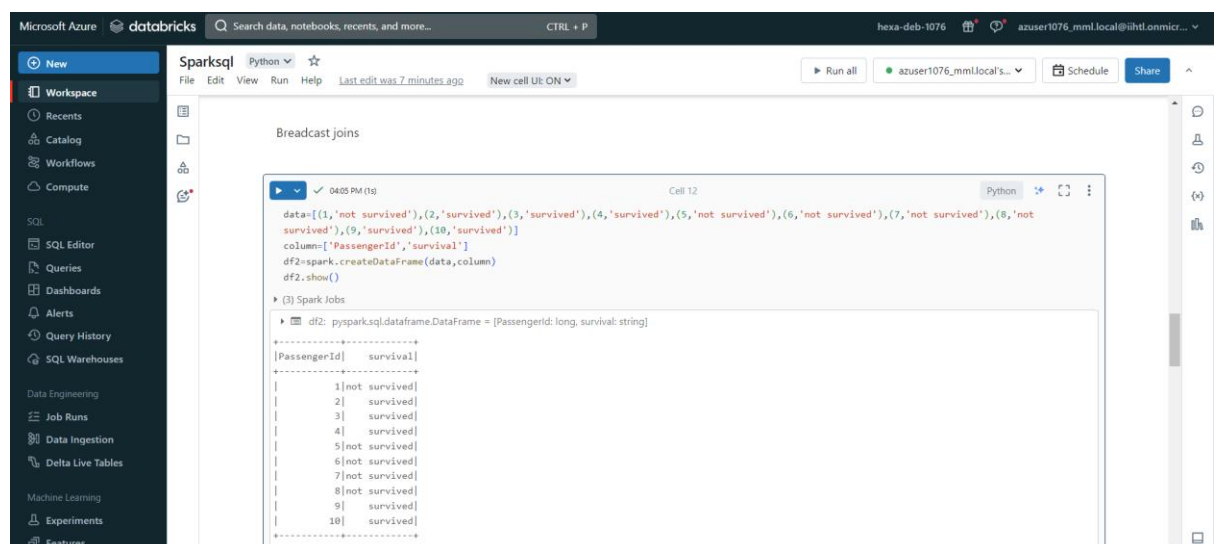
1 minute ago (<1s) Cell 9
# Use of coalesce to reduce the number of partitions
df_coalesced = df.coalesce(2) # 2 is the desired number of partitions
df_coalesced: pyspark.sql.dataframe.DataFrame = [PassengerId: integer, Survived: integer ... 10 more fields]

Just now (<1s) Cell 10
# Show the final number of partitions
print("Final number of partitions:", df_coalesced.rdd.getNumPartitions())
# we are already at minimum number
Final number of partitions: 1
```

**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:



```
Microsoft Azure databricks Search data, notebooks, recent, and more... CTRL + P hexa-deb-1076 azuser1076_mml.local@iitl.onmicr...

New Workspace Recents Catalog Workflows Compute SQL SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Data Engineering Job Runs Data Ingestion Delta Live Tables Machine Learning Experiments Features

Sparksq Python File Edit View Run Help Last edit was 7 minutes ago New cell UI: ON Run all azuser1076_mml.local's... Schedule Share

Broadcast joins

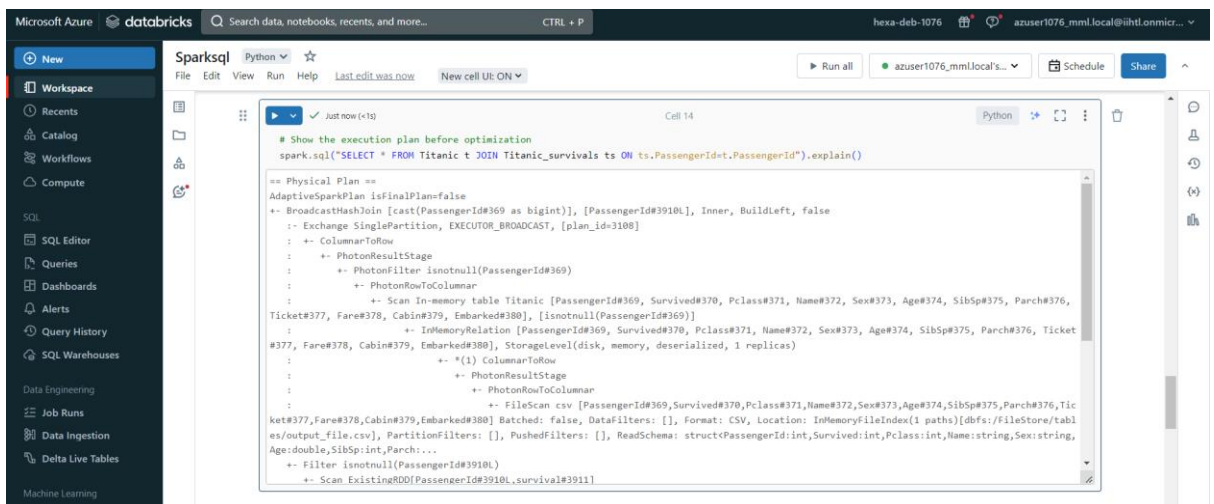
04:05 PM (1s) Cell 12 Python
data=[(1,'not survived'),(2,'survived'),(3,'survived'),(4,'survived'),(5,'not survived'),(6,'not survived'),(7,'not survived'),(8,'not survived'),(9,'survived'),(10,'survived')]
column=['PassengerId','survival']
df2=spark.createDataFrame(data,column)
df2.show()

(3) Spark Jobs
df2: pyspark.sql.dataframe.DataFrame = [PassengerId: long, survival: string]
+-----+-----+
| PassengerId | survival |
+-----+-----+
| 1 | not survived |
| 2 | survived |
| 3 | survived |
| 4 | survived |
| 5 | not survived |
| 6 | not survived |
| 7 | not survived |
| 8 | not survived |
| 9 | survived |
| 10 | survived |
+-----+-----+
```

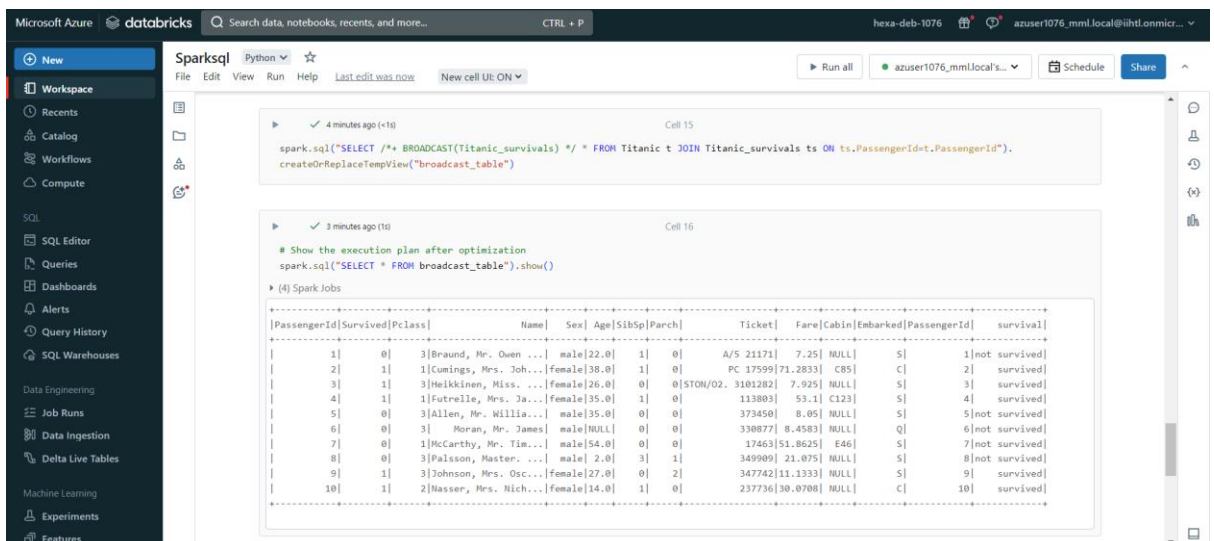
## Creating View:



## Execution of simple join:

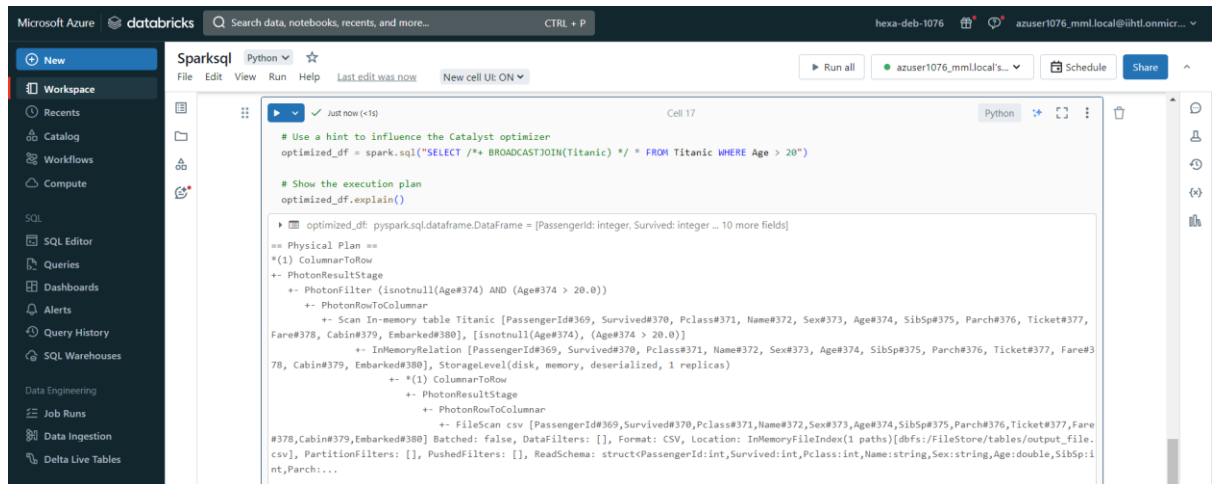


## Creating a broadcast join:





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.



```
Microsoft Azure databricks Search data, notebooks, recents, and more... CTRL + P hexa-deb-1076 azuser1076_mml.local@iitd.onmicr...

New Workspace Recents Catalog Workflows Compute SQL SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Data Engineering Job Runs Data Ingestion Delta Live Tables

Sparksql Python File Edit View Run Help Last edit was now New cell UI: ON

▶ Run all azuser1076_mml.local's... Schedule Share

# Use a hint to influence the Catalyst optimizer
optimized_df = spark.sql("SELECT /*+ BROADCASTJOIN(Titanic) */ * FROM Titanic WHERE Age > 20")

# Show the execution plan
optimized_df.explain()

optimized_df: pyspark.sql.dataframe.DataFrame = [PassengerId: integer, Survived: integer ... 10 more fields]

== Physical Plan ==
*(1) ColumnarToRow
+- PhotonResultStage
   +- PhotonFilter (isNotNull(Age#374) AND (Age#374 > 20.0))
      +- PhotonRowToColumnar
         +- Scan In-memory table Titanic [PassengerId#369, Survived#370, Pclass#371, Name#372, Sex#373, Age#374, SibSp#375, Parch#376, Ticket#377, Fare#378, Cabin#379, Embarked#380], [isNotNull(Age#374), (Age#374 > 20.0)]
            +- InMemoryRelation [PassengerId#369, Survived#370, Pclass#371, Name#372, Sex#373, Age#374, SibSp#375, Parch#376, Ticket#377, Fare#378, Cabin#379, Embarked#380], StorageLevel(disk, memory, deserialized, 1 replicas)
               +- *(1) ColumnarToRow
                  +- PhotonRowToColumnar
                     +- FileScan csv [PassengerId#369, Survived#370, Pclass#371, Name#372, Sex#373, Age#374, SibSp#375, Parch#376, Ticket#377, Fare#378, Cabin#379, Embarked#380] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs://FileStore/tables/output_file.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<PassengerId:int, Survived:int, Pclass:int, Name:string, Sex:string, Age:double, SibSp:integer, Parch:integer>
```

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.



```
Microsoft Azure databricks Search data, notebooks, recents, and more... CTRL + P hexa-deb-1076 azuser1076_mml.local@iitd.onmicr...

New Workspace Recents Catalog Workflows Compute SQL SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Data Engineering Job Runs Data Ingestion Delta Live Tables

Sparksql Python File Edit View Run Help Last edit was now New cell UI: ON

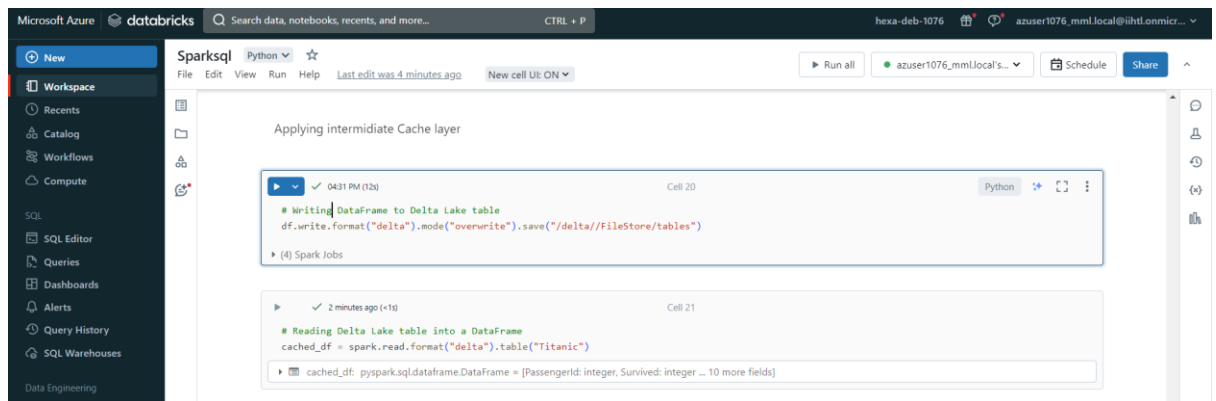
▶ Run all azuser1076_mml.local's... Schedule Share

# Set the autoBroadcastJoinThreshold property
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "10m")
```

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.

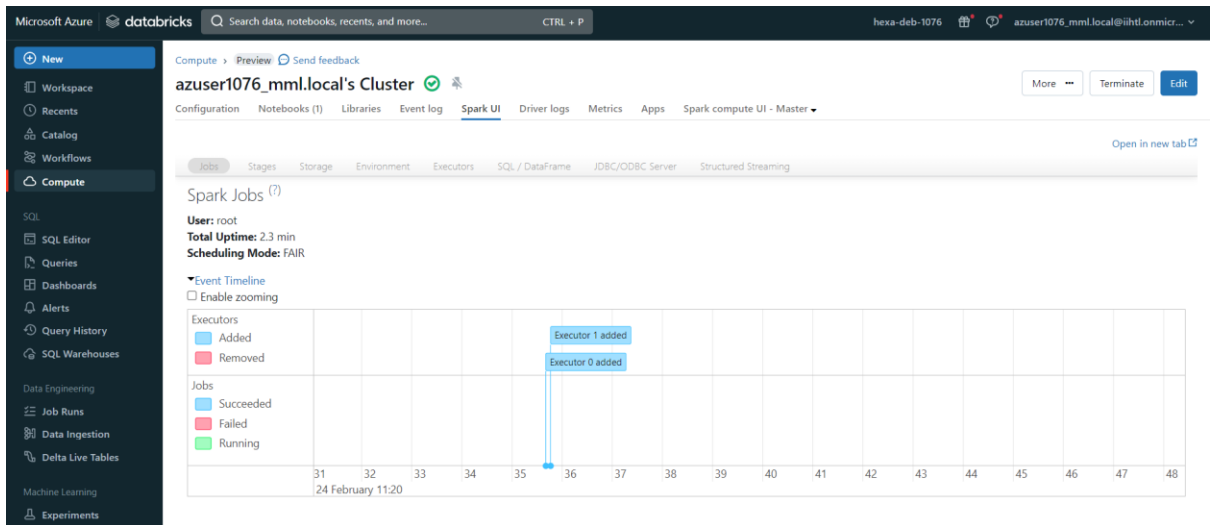


Performing Spark SQL queries & storing the intermediate results in the Delta Lake table

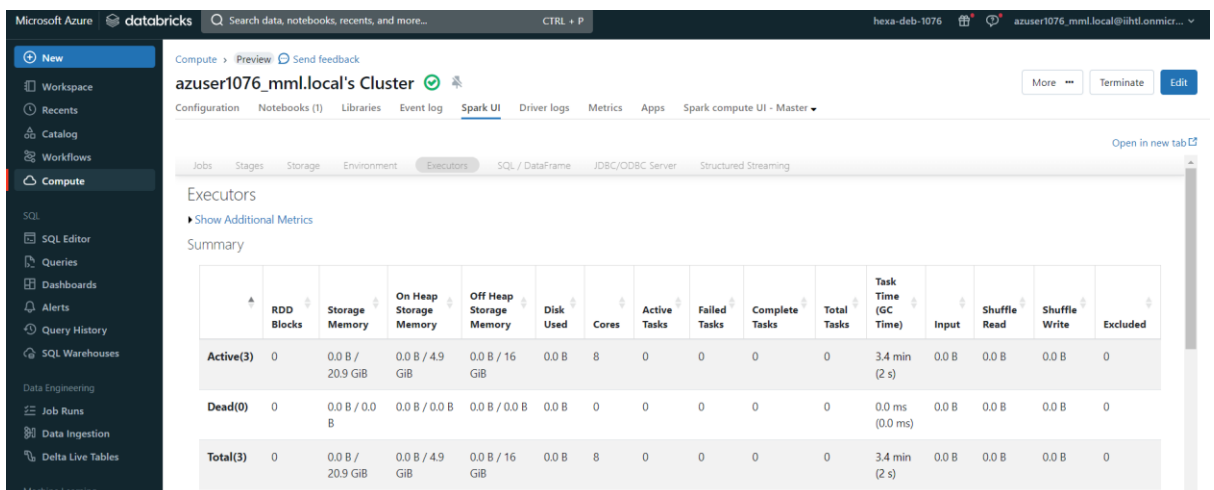


In This way, we're using Delta Lake as an intermediate data cache layer.

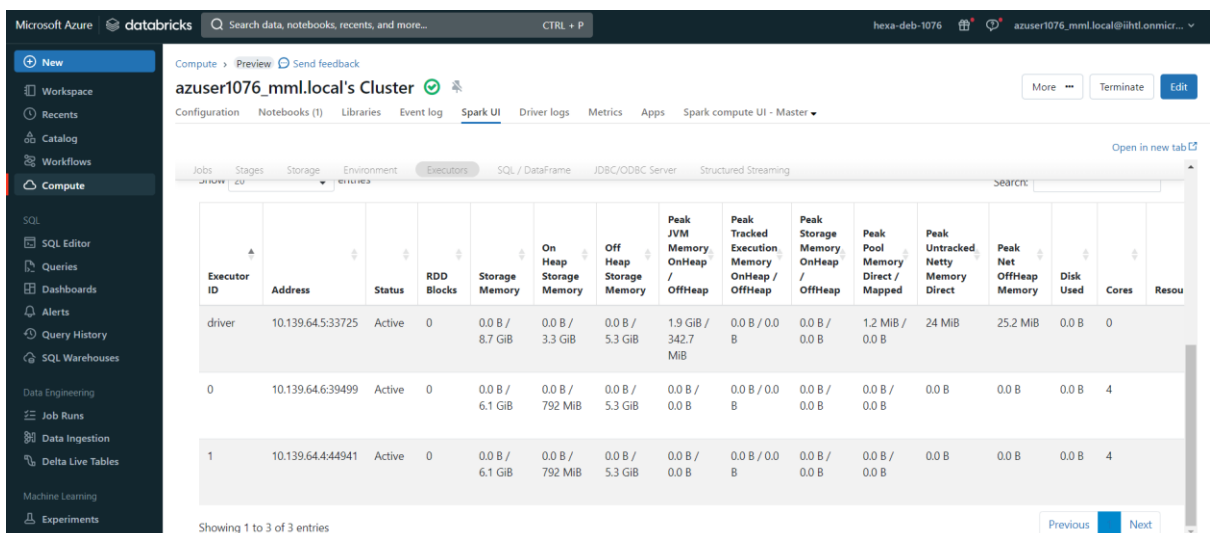
## 6.4) Monitoring in Spark UI:



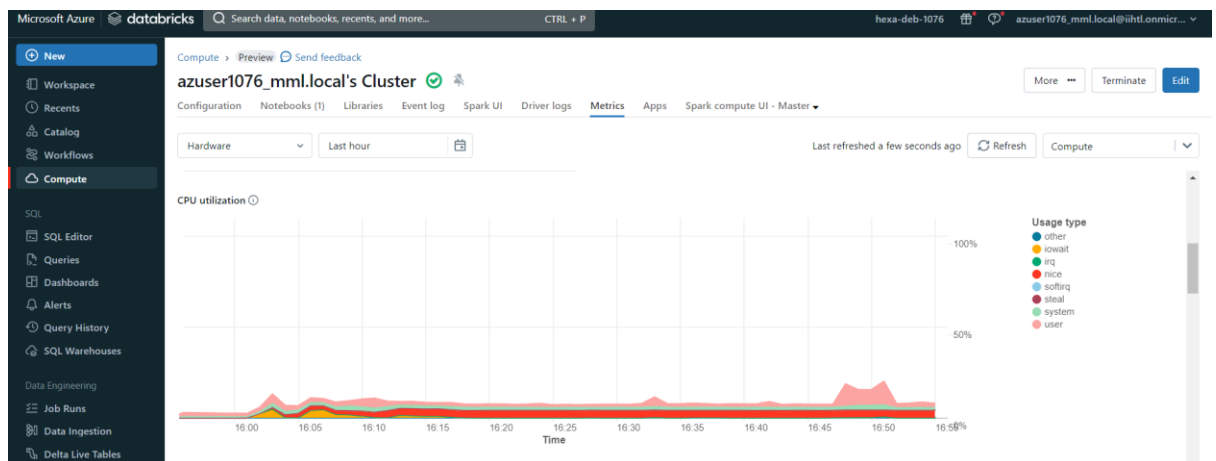
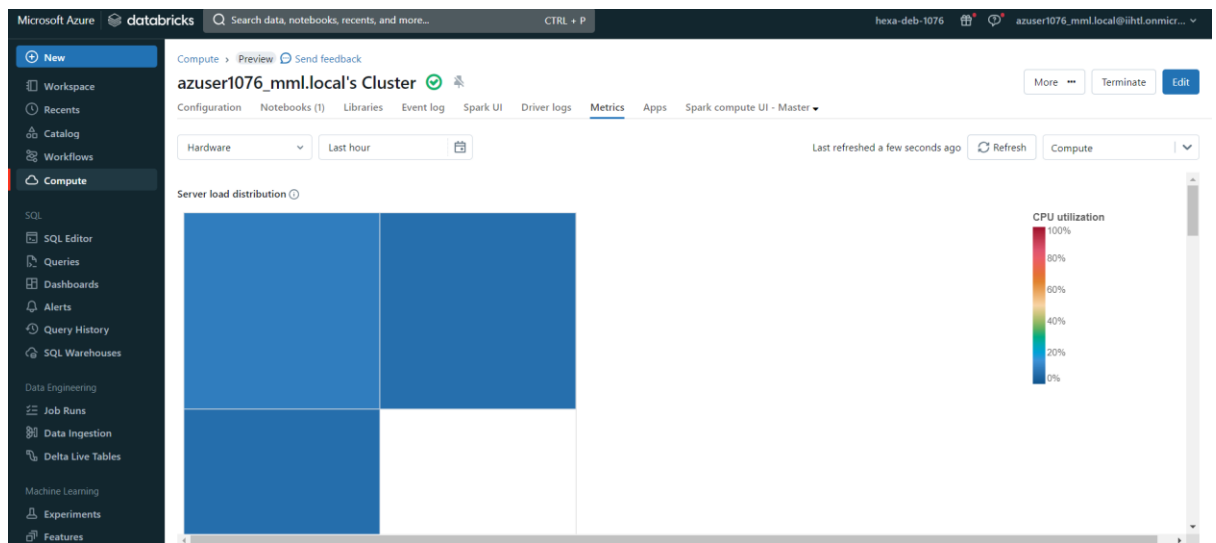
### Executors:



### Details of executors:



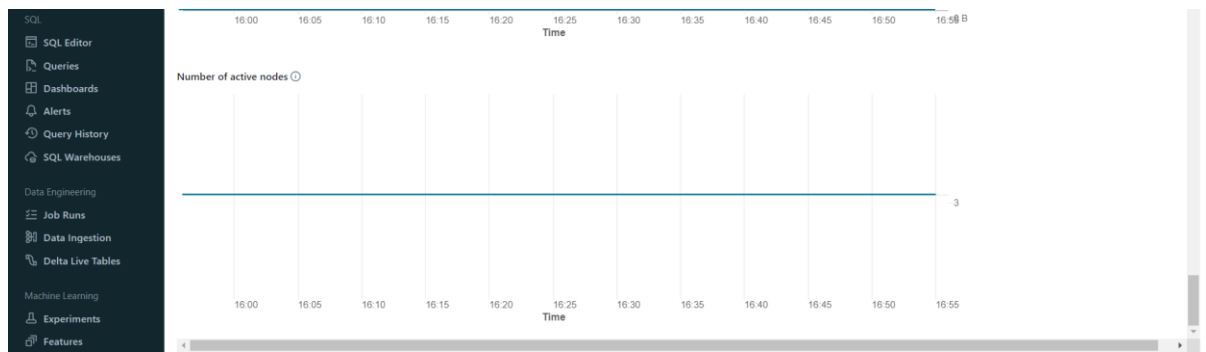
## CPU utilization:



## 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.