**About the dataset:**

This is the Amazon Fine Food Reviews dataset downloaded from kaggle link <https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews>

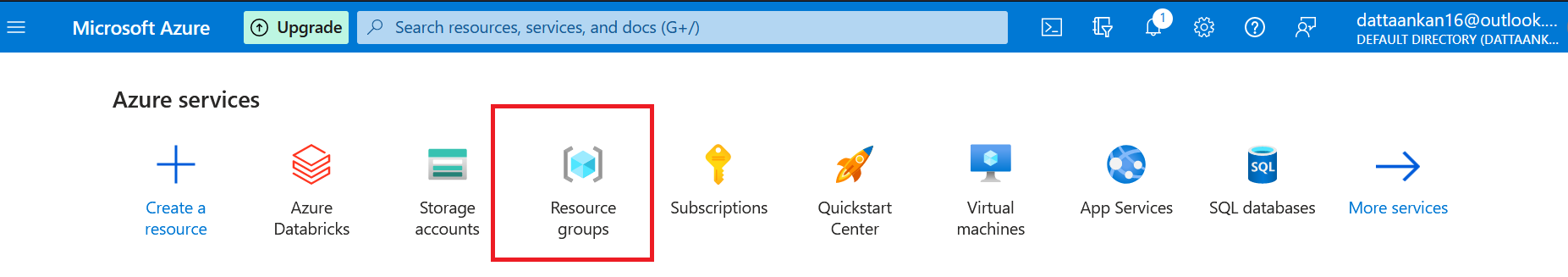
**1. Upload 2 datasets of your choice one big and one small (less than 10 mb)**

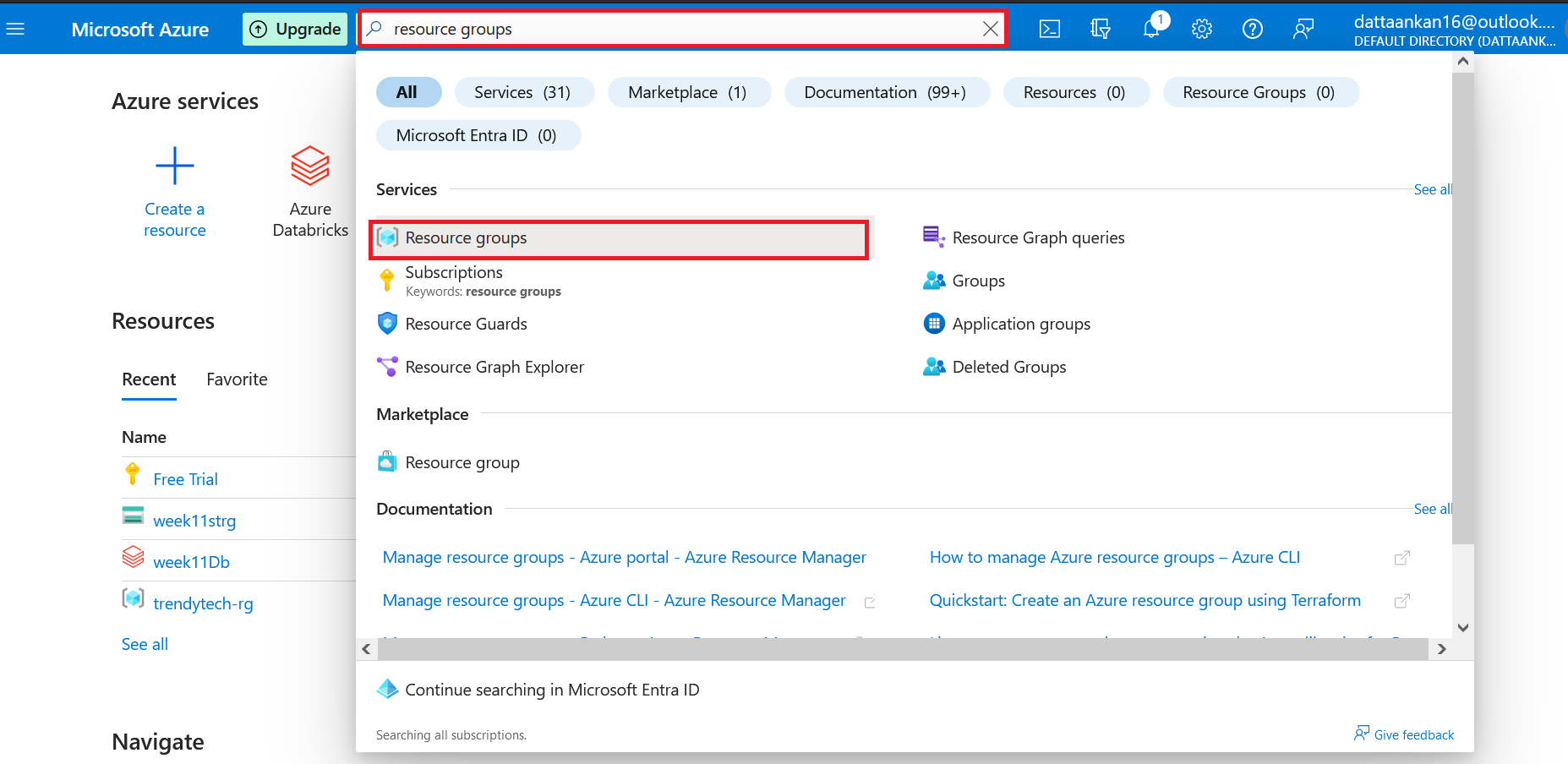
**to ADLS Gen2 so that we can perform join**

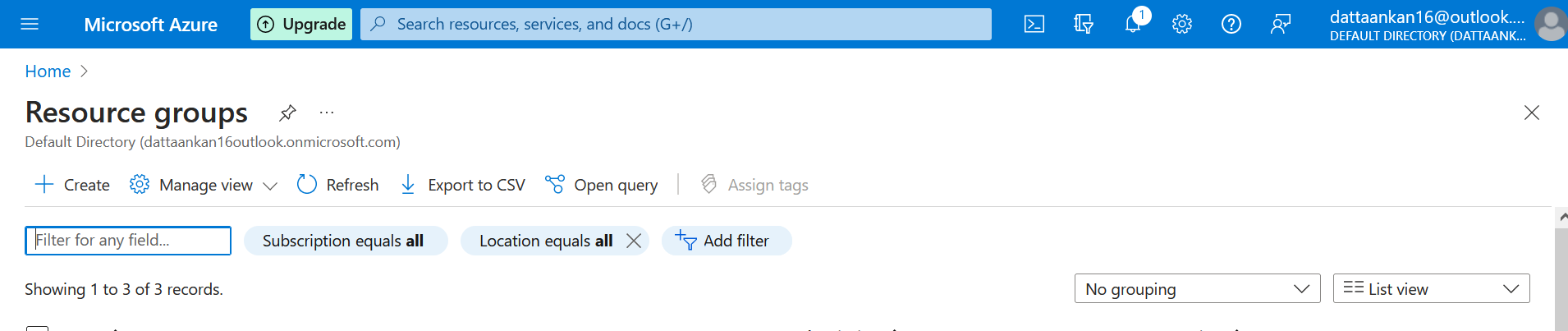
1. Go to <https://portal.azure.com>



1. Type resource groups in the search bar or click on resource groups in the options as shown, then create a resource group and name it

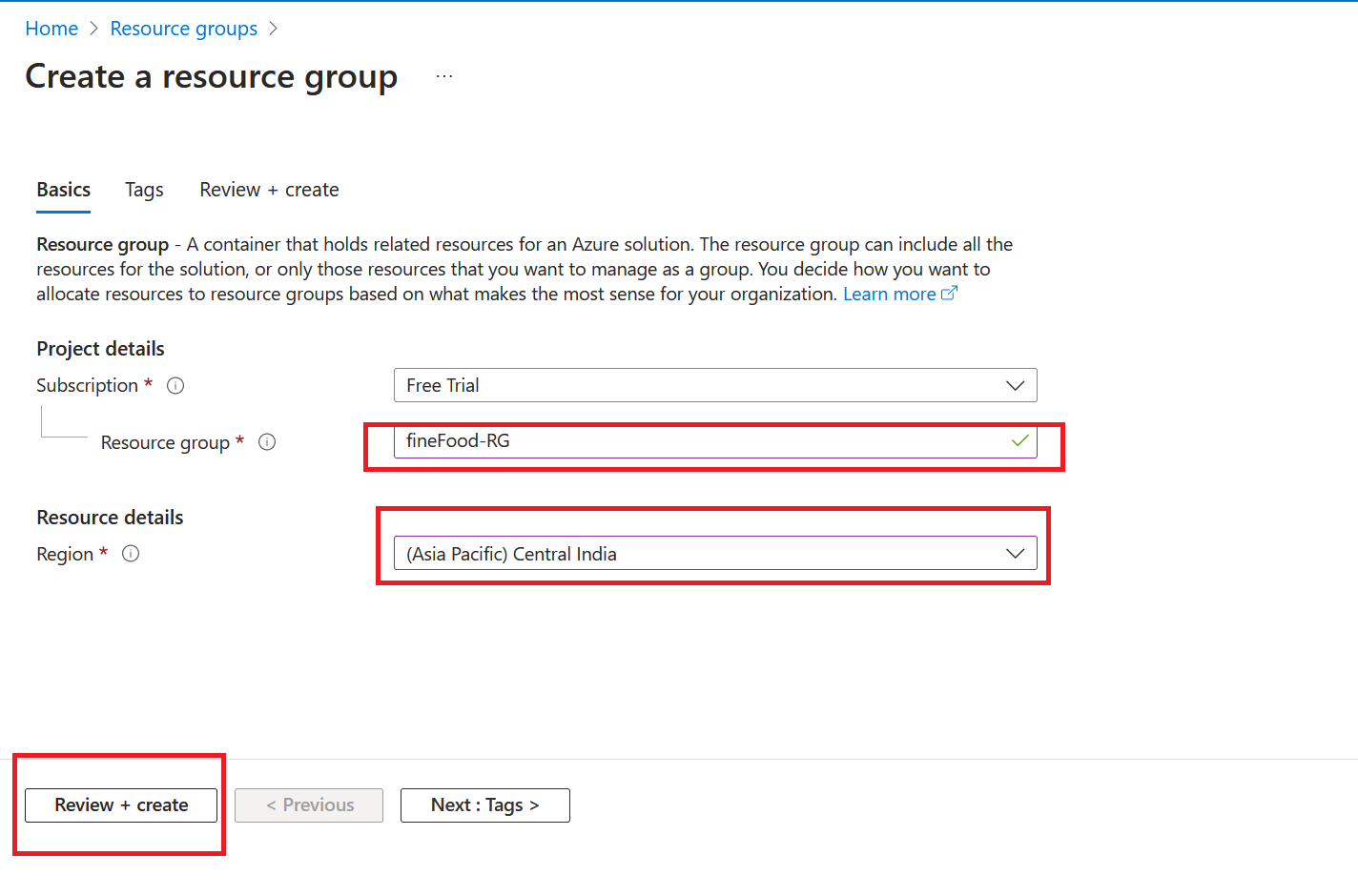




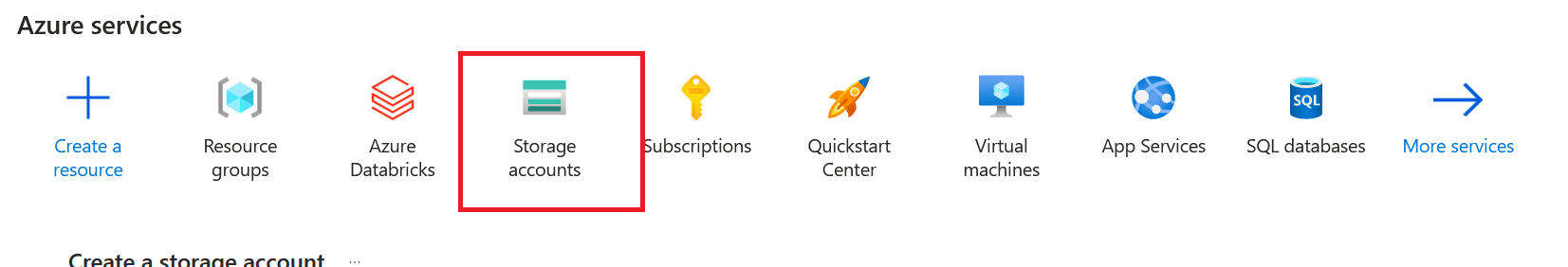


Click on create

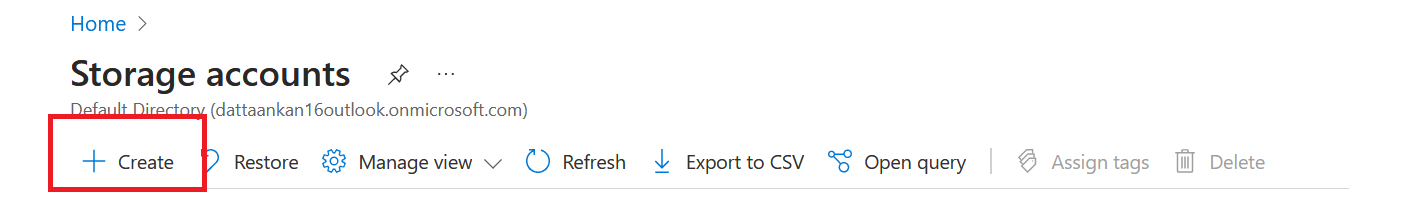
1. Name your resource group and click on Review+Create



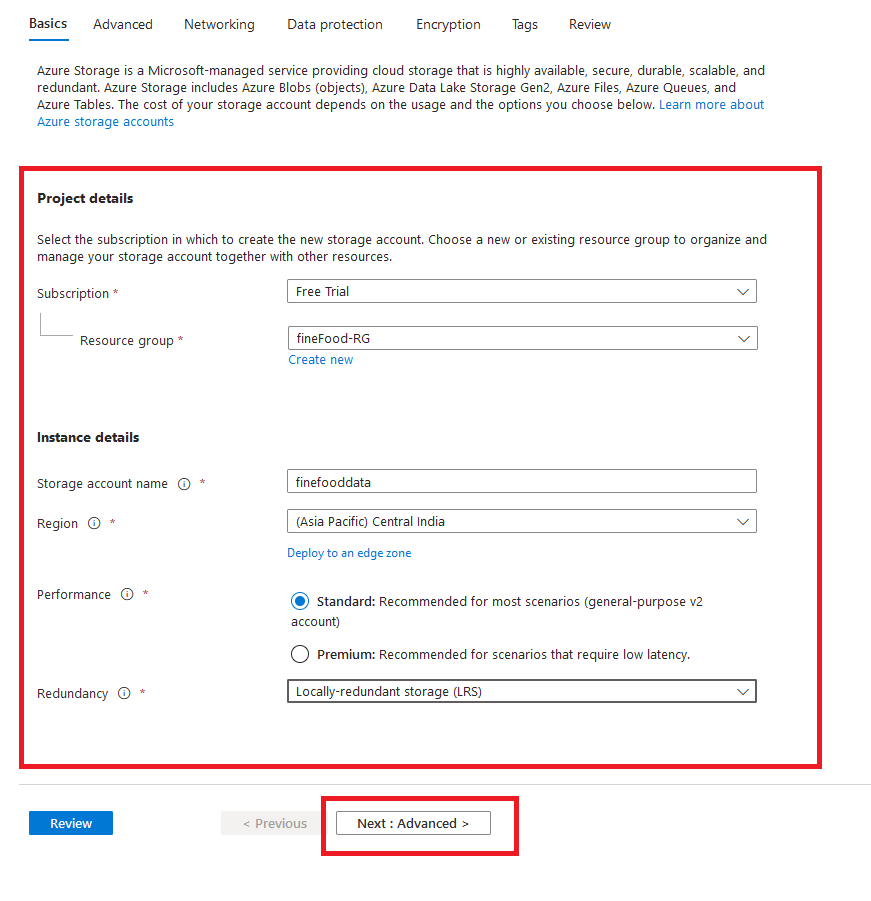
1. Then go to storage accounts, and create a storage account



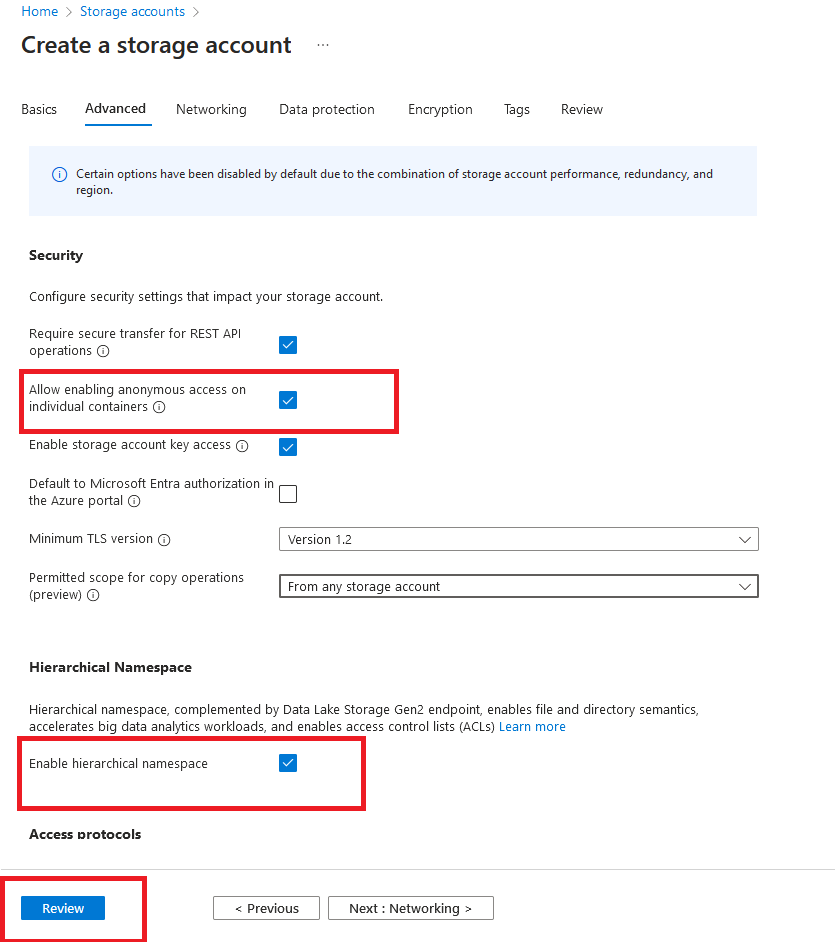
Click on Create



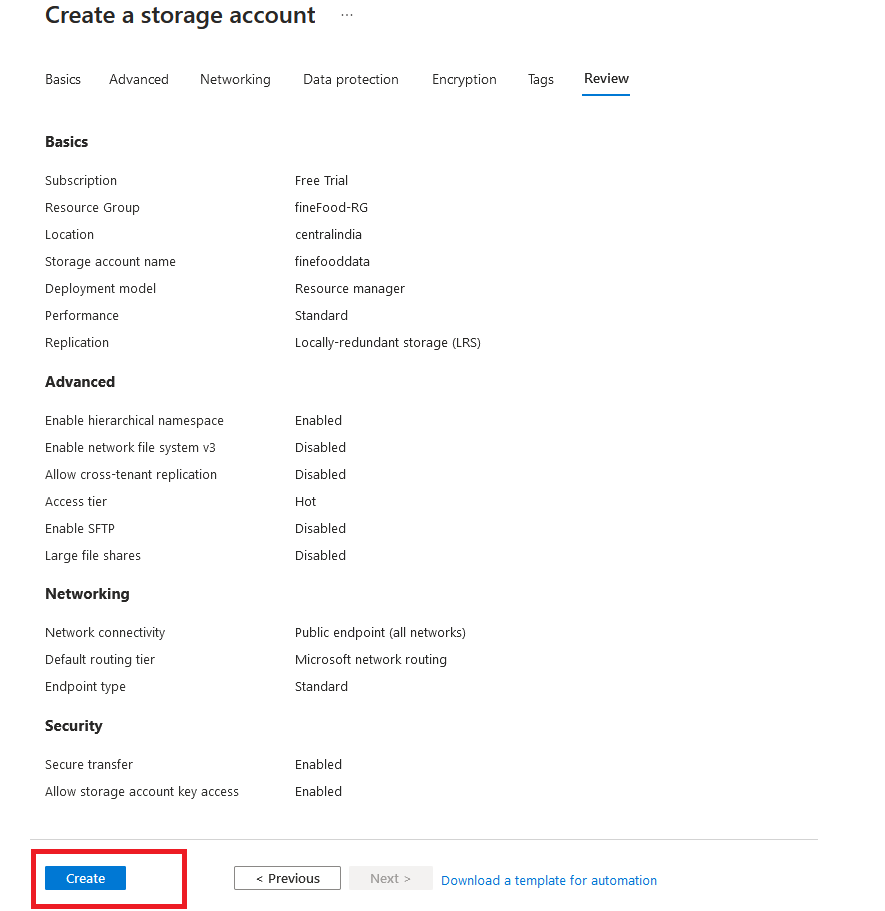
1. Select Advanced and keep the settings intact. Please make sure to click on advanced options before creating the storage account

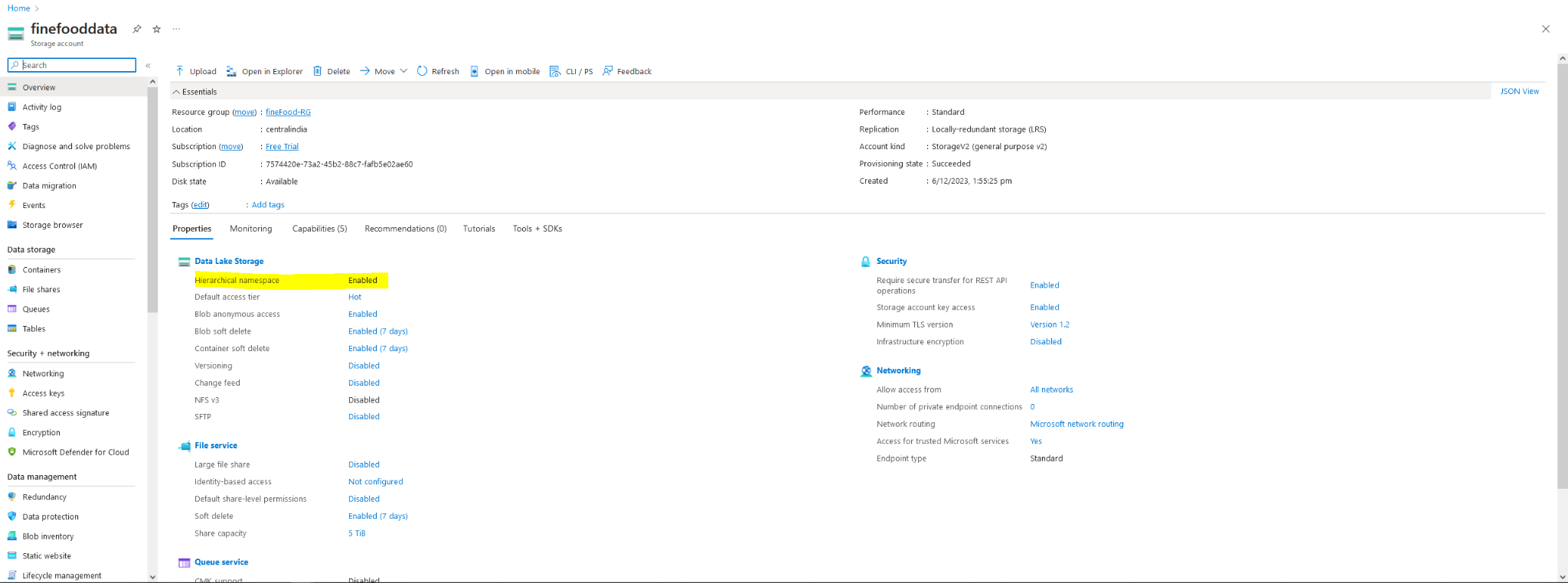


1. Check the Allow enabling anonymous access on individual containers and the enable hierarchical namespace boxes in order to make sure that our Data Lake gen 2 is active and our databricks workspace can access the files inside the storage accounts without fail



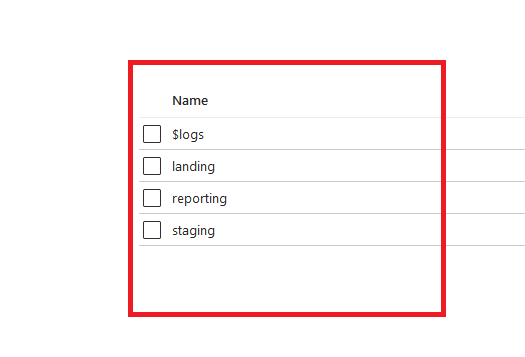
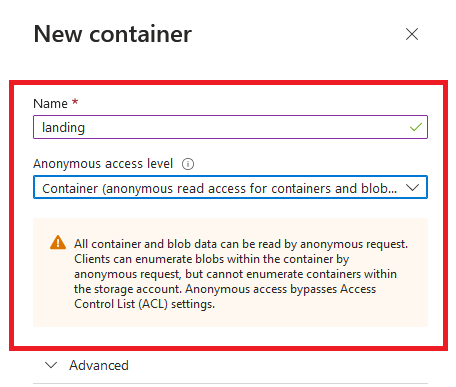
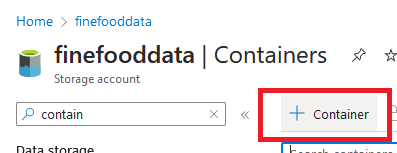
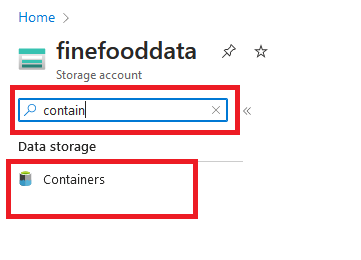
1. Click on create and your storage account will be created





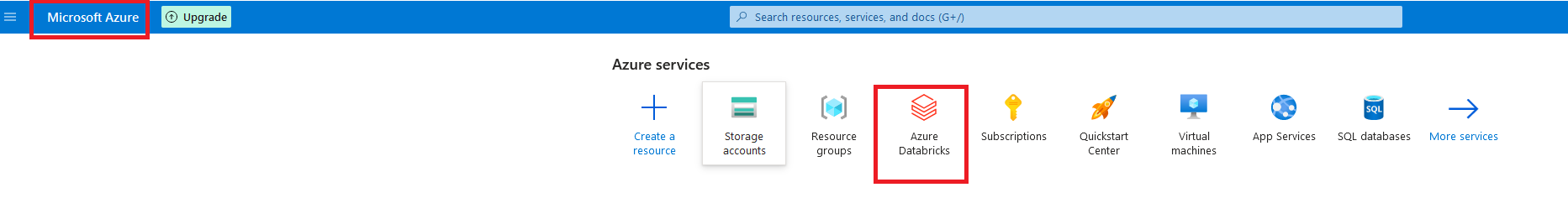
1. Search Containers and create containers. Here We have created 3 containers named :

* Landing : here we will keep the RAW data
* Staging : here we will keep the transformed data
* Reporting : here we will keep the data after all the transformations are done and ready for reporting

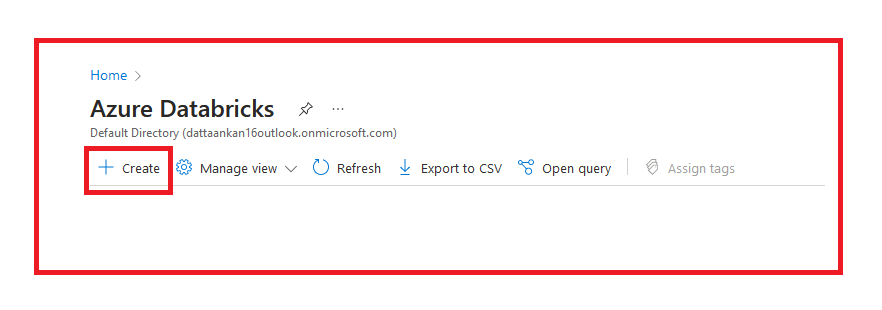


**2. Create a databricks single node cluster.**

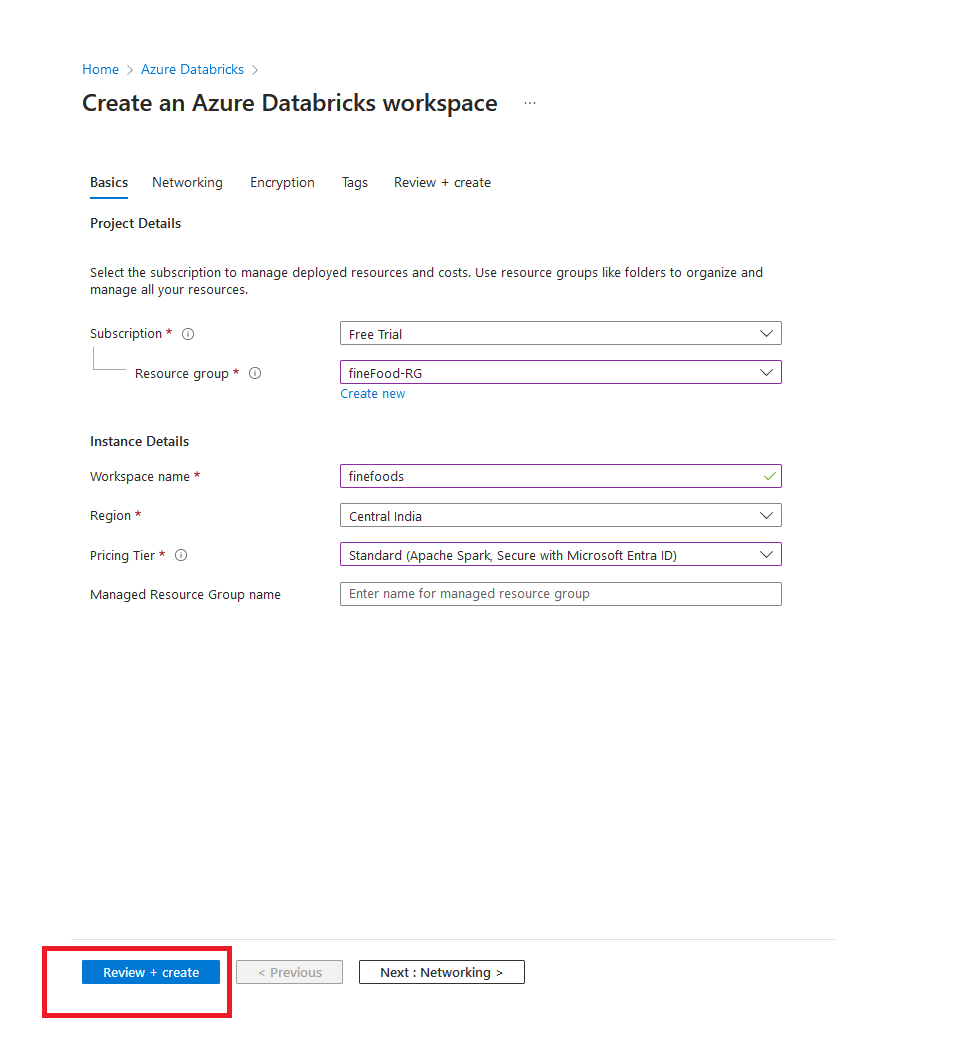
1. Click on the home page and select Azure data bricks Option

****

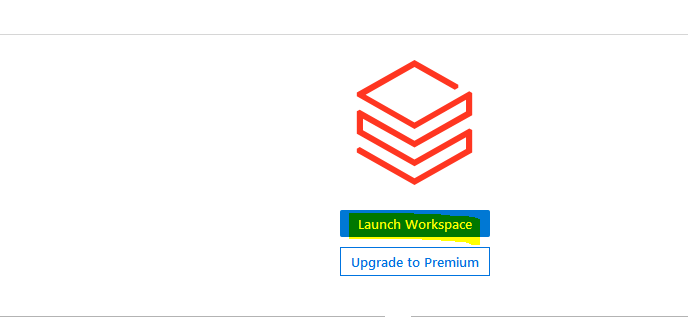
1. Click on create

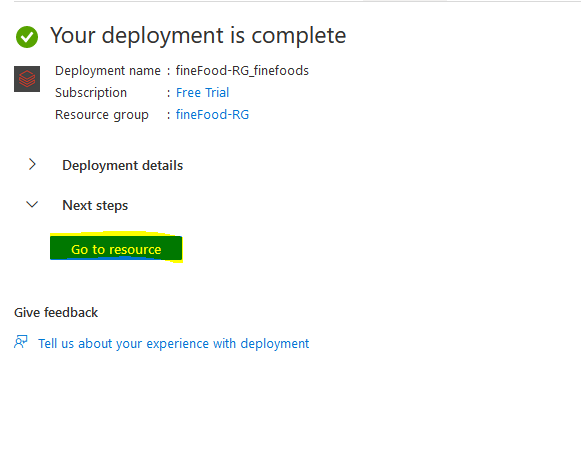


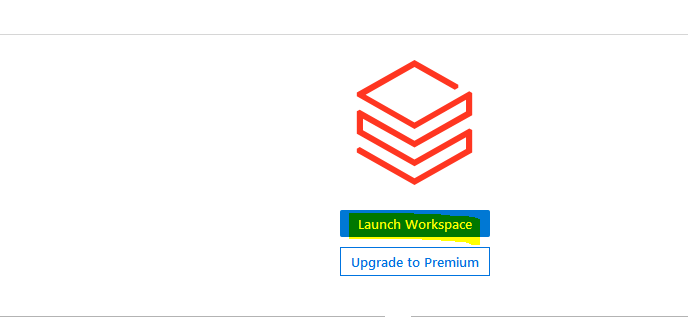
1. Click on review + create



1. Click on Create. It will take some time for the cluster to be created. Once created, go to resource and Launch the Workspace

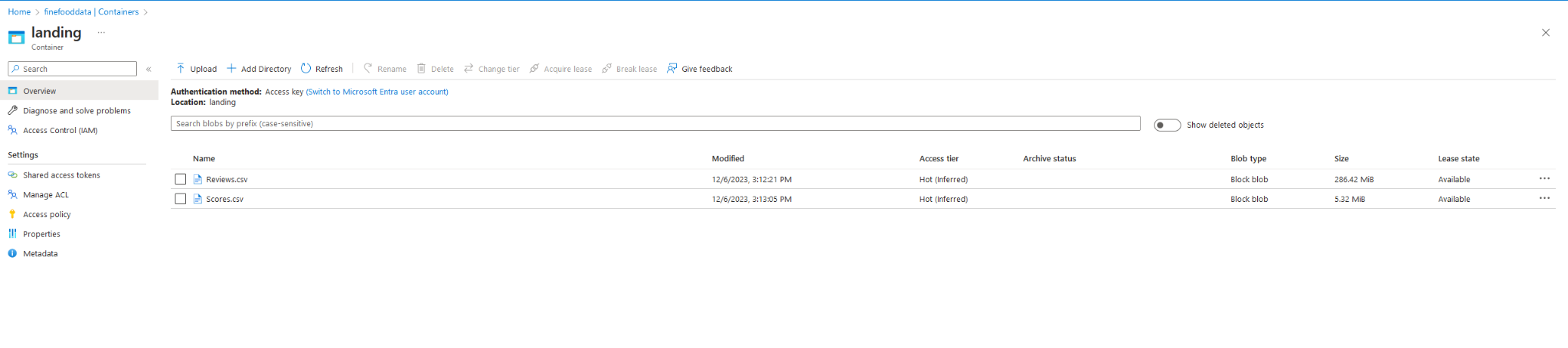




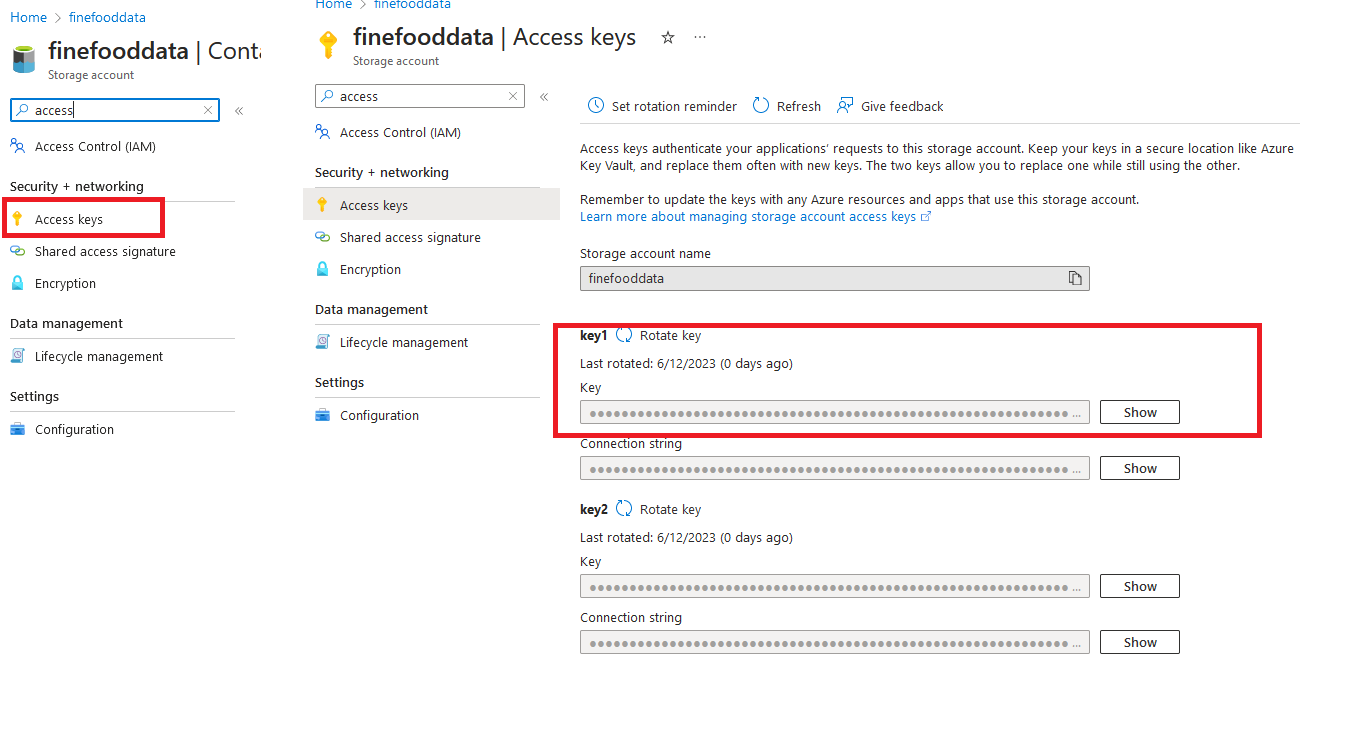


**3. Mount the storage so that you can access the files in ADLS gen2**

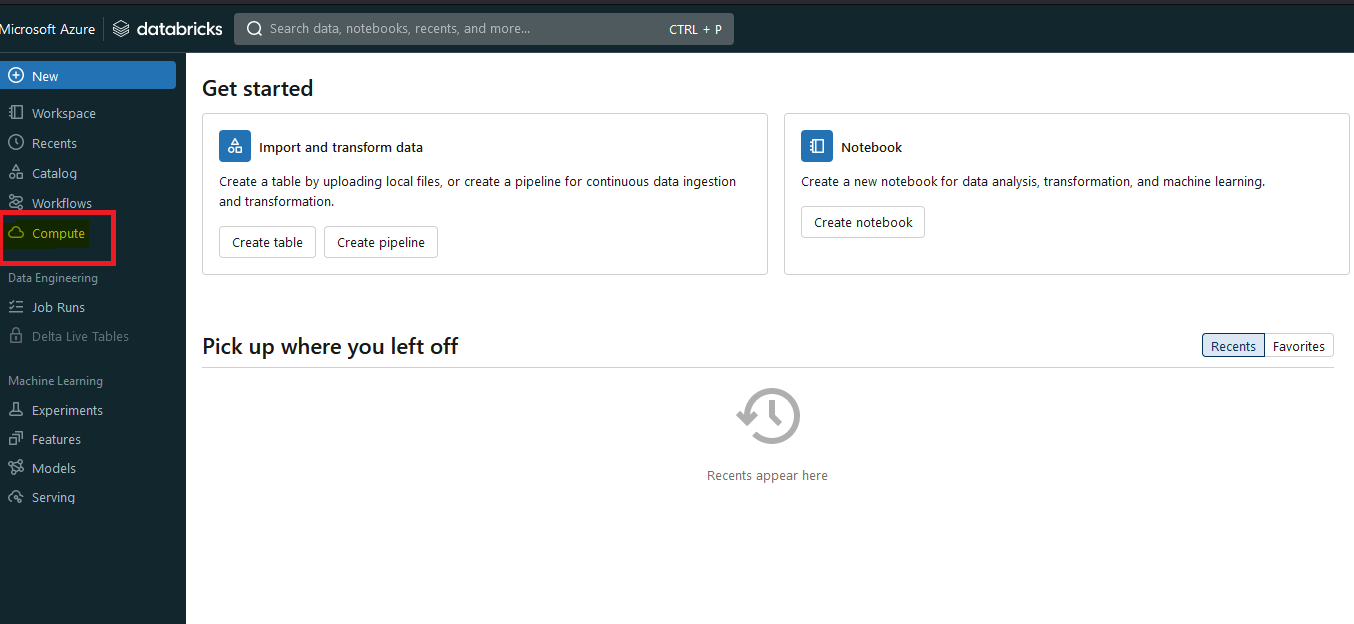
1. Upload the fine foods data in the landing container in the storage account



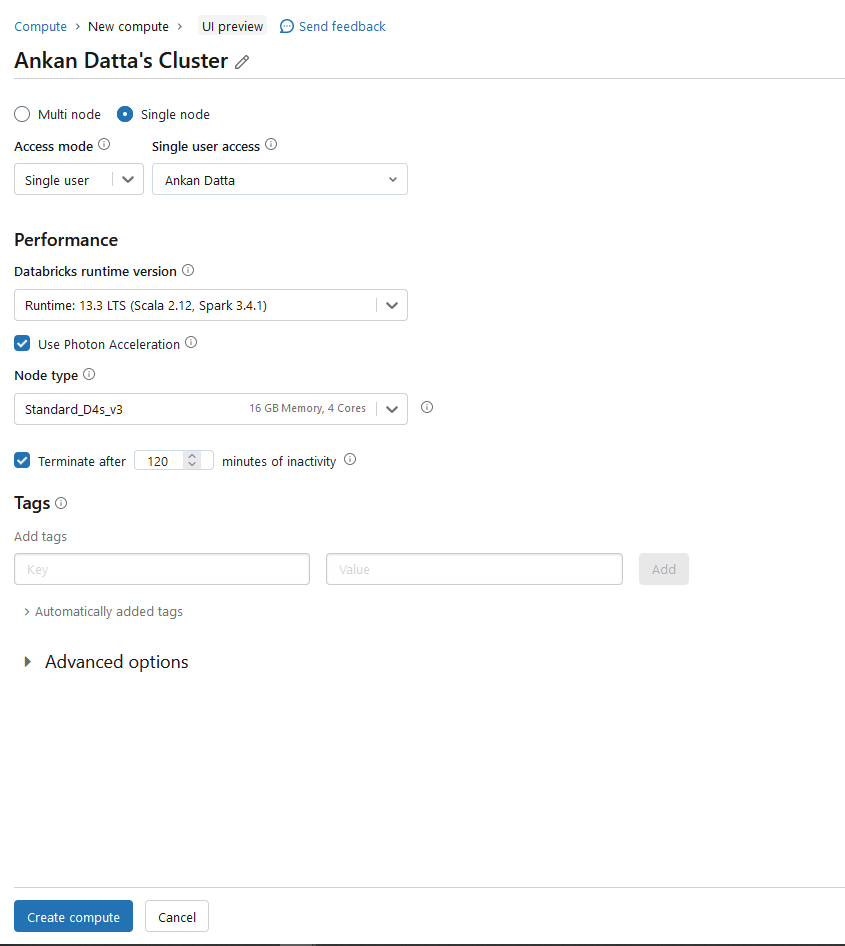
1. Got to access keys and copy the keys and store it somewhere

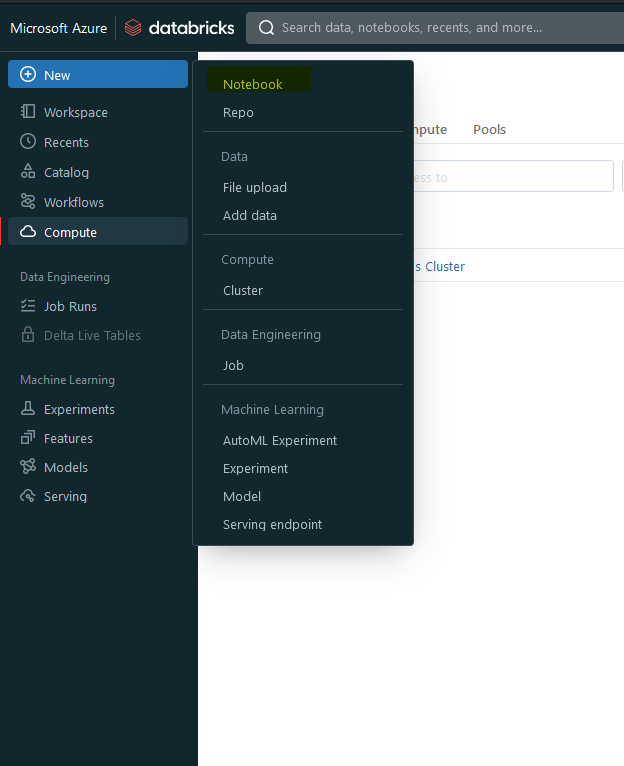


1. Go to the databricks workspace and click on compute



1. Select “single node cluster”, name it and then create compute. A compute will be created, it will take some time to finish



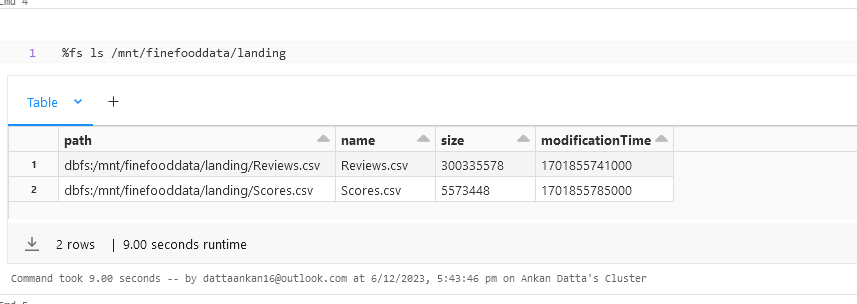
1. Once the cluster is created, click on New → Notebook to create a new notebook
2. Mount the storages :

* We can use this boilerplate code for mounting the storages

dbutils.fs.mount(source = "wasbs://containeer\_name@storage\_account\_name.blob.core.windows.net", mount\_point = "/mnt/folder/container\_name", extra\_configs = {"fs.azure.account.key.storage\_account\_name.blob.core.windows.net":"key"})

* Landing : paste the key that was copied earlier

dbutils.fs.mount(source = "wasbs://landing@finefooddata.blob.core.windows.net", mount\_point = "/mnt/finefooddata/landing", extra\_configs = {"fs.azure.account.key.finefooddata.blob.core.windows.net":"BFXqu52ydqWJ4WSTchnAGvGLdKY69OxYB20CpMxmSraxSZBbQlQBEggovAbc81OcrQtWX8ZBfoLF+AStQkhkwQ=="})



* Staging : dbutils.fs.mount(source = "wasbs://staging@finefooddata.blob.core.windows.net", mount\_point = "/mnt/finefooddata/staging", extra\_configs = {"fs.azure.account.key.finefooddata.blob.core.windows.net":"BFXqu52ydqWJ4WSTchnAGvGLdKY69OxYB20CpMxmSraxSZBbQlQBEggovAbc81OcrQtWX8ZBfoLF+AStQkhkwQ=="})
* 
* Reporting : dbutils.fs.mount(source = "wasbs://reporting@finefooddata.blob.core.windows.net", mount\_point = "/mnt/finefooddata/reporting", extra\_configs = {"fs.azure.account.key.finefooddata.blob.core.windows.net":"BFXqu52ydqWJ4WSTchnAGvGLdKY69OxYB20CpMxmSraxSZBbQlQBEggovAbc81OcrQtWX8ZBfoLF+AStQkhkwQ=="})

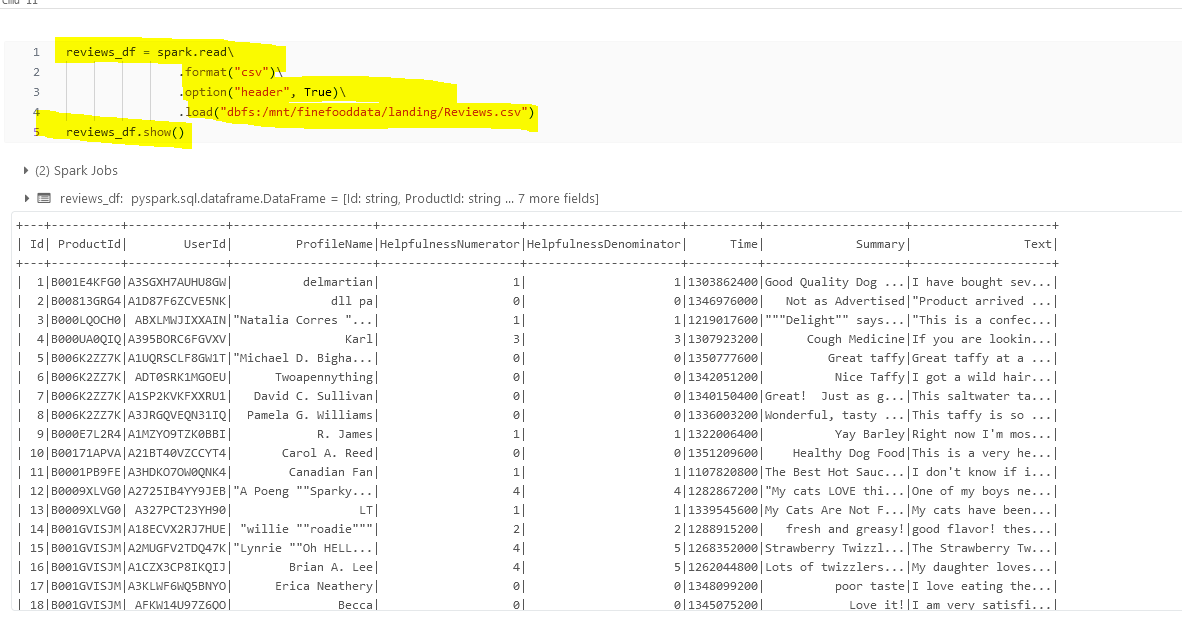


**4. Create 2 data frames, one on each file and perform a join using Dataframes**

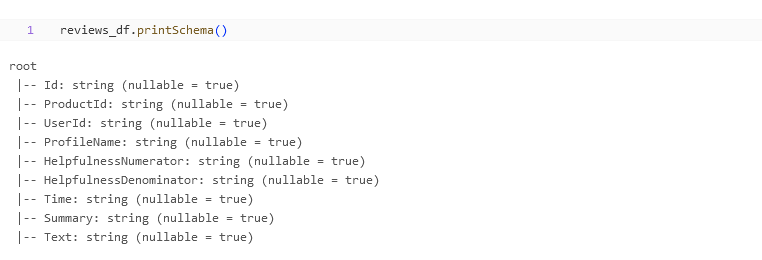
**approach as well as spark SQL style. Do check the spark UI to see the join**

**strategy used.**

Reading the reviews data frame



After reading the reviews dataframe, we see that all columns are interpreted as strings



So, we skip the first row and read it again  
  
from pyspark.sql.types import StructType, StructField, TimestampType, IntegerType, StringType

from pyspark.sql.functions import from\_unixtime, expr

reviews\_schema = StructType([

StructField("Id", IntegerType(), True),

StructField("ProductId", StringType(), True),

StructField("UserId", StringType(), True),

StructField("ProfileName", StringType(), True),

StructField("HelpfulnessNumerator", IntegerType(), True),

StructField("HelpfulnessDenominator", IntegerType(), True),

StructField("Time", IntegerType(), True),

StructField("Summary", StringType(), True),

StructField("Text", StringType(), True)

])  
  
reviews\_df = spark.read\

.format("csv")\

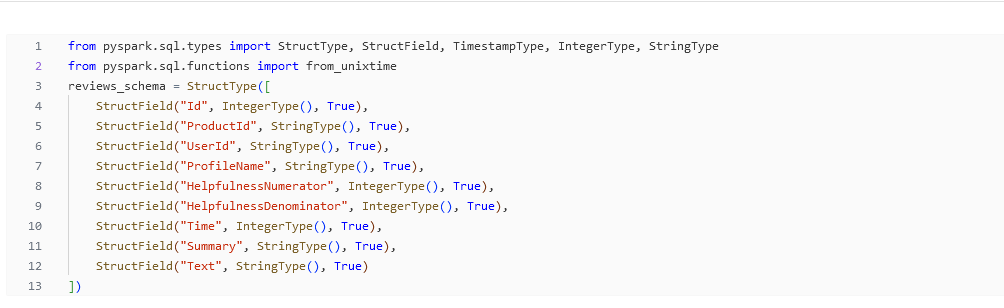
.option("header", False)\

.schema(reviews\_schema)\

.load("dbfs:/mnt/finefooddata/landing/Reviews.csv", skipRows=1)

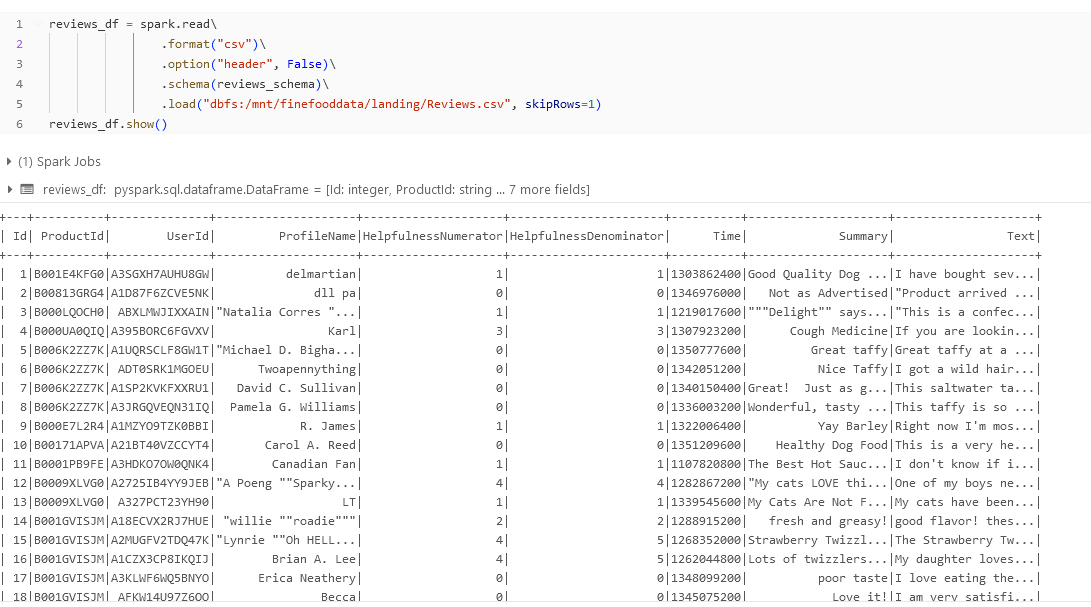
reviews\_df.show()

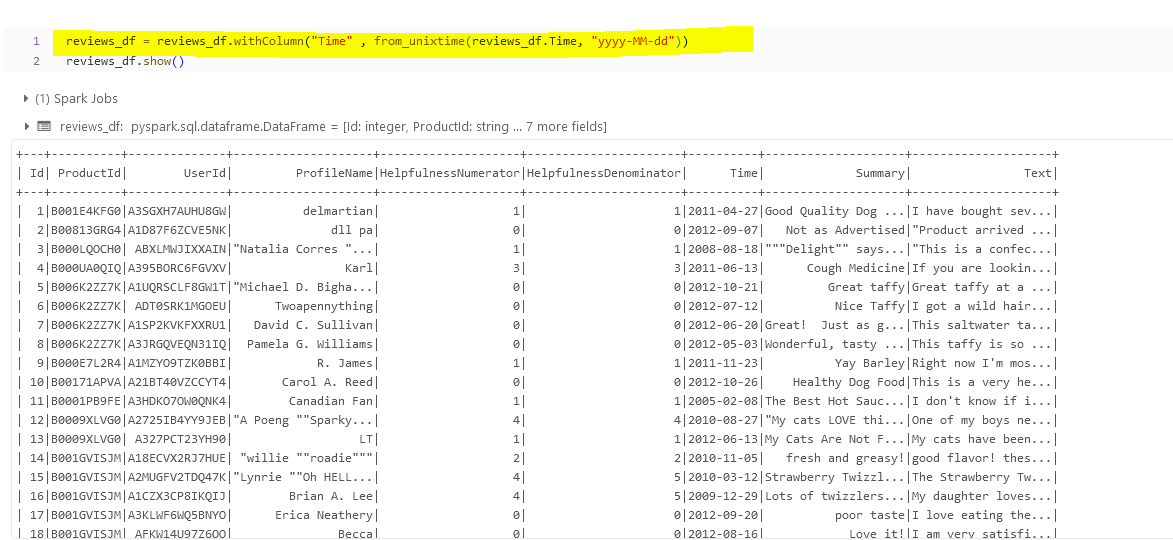


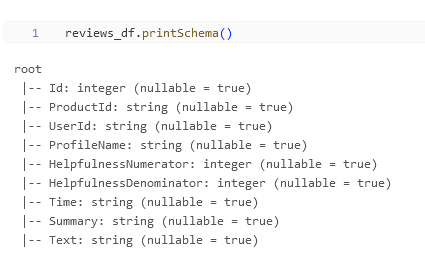
Giving it a schema   


Reading the file again  
  
reviews\_df = reviews\_df.withColumn("Time" , from\_unixtime(reviews\_df.Time, "yyyy-MM-dd"))

reviews\_df.show()



Converting the time column to a proper date formatted timestamp   


Printing the schema again  


Now that we have read the reviews data frame, we are now going to read the scores data set. We will be giving it a schema and the reading it  
  
scores\_schema = StructType([

StructField("Id", IntegerType(), True),

StructField("Score", IntegerType(), True)

])  
  
scores\_df = spark.read\

.format("csv")\

.option("header", False)\

.schema(scores\_schema)\

.load("dbfs:/mnt/finefooddata/landing/Scores.csv",skipRows=1)

scores\_df.show()



Filtering and caching the results  
  
reviews\_df.count()  
  
reviews\_df = reviews\_df.filter(reviews\_df.HelpfulnessDenominator>reviews\_df.HelpfulnessNumerator)

reviews\_df = reviews\_df.dropDuplicates()

reviews\_df = reviews\_df.cache()

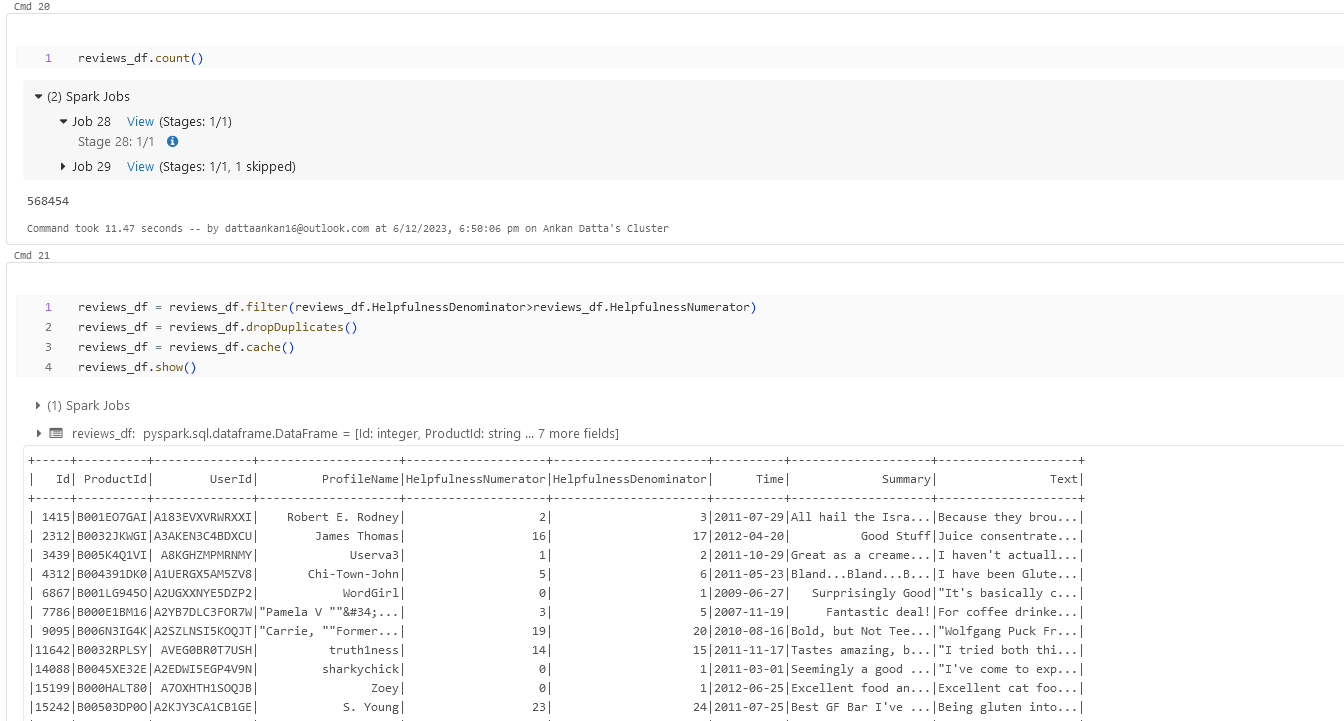
reviews\_df.show()

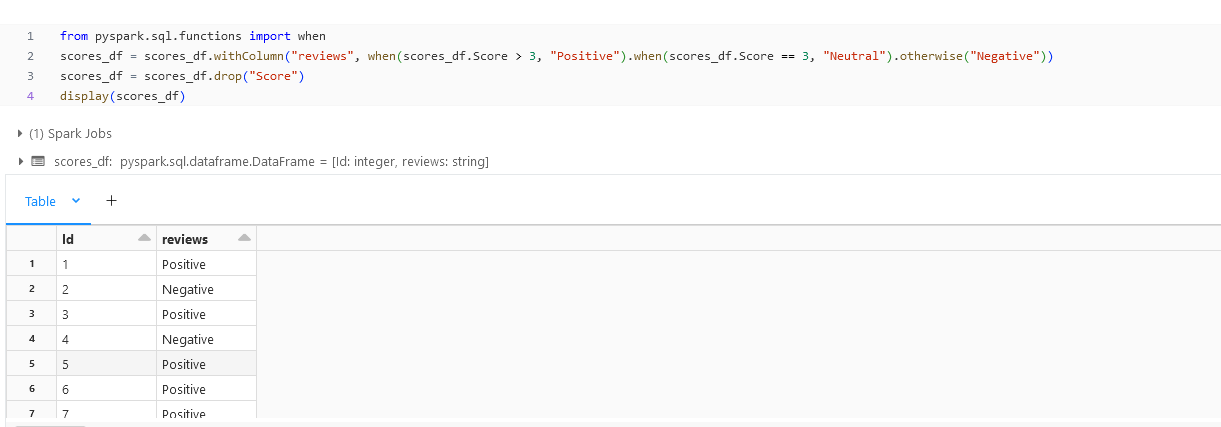
from pyspark.sql.functions import when

scores\_df = scores\_df.withColumn("reviews", when(scores\_df.Score > 3, "Positive").when(scores\_df.Score == 3, "Neutral").otherwise("Negative"))

scores\_df = scores\_df.drop("Score")

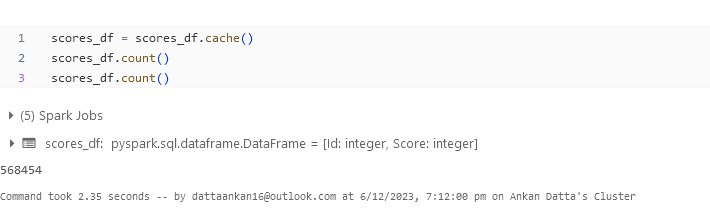
display(scores\_df)



  
  
scores\_df = scores\_df.cache()

scores\_df.count()

scores\_df.count()



**Writing the data in the staging folder**

reviews\_df.write\

.format("parquet")\

.mode("overwrite")\

.option("path", "/mnt/finefooddata/staging/reviews\_filtered")\

.save()

scores\_df.write\

.format("parquet")\

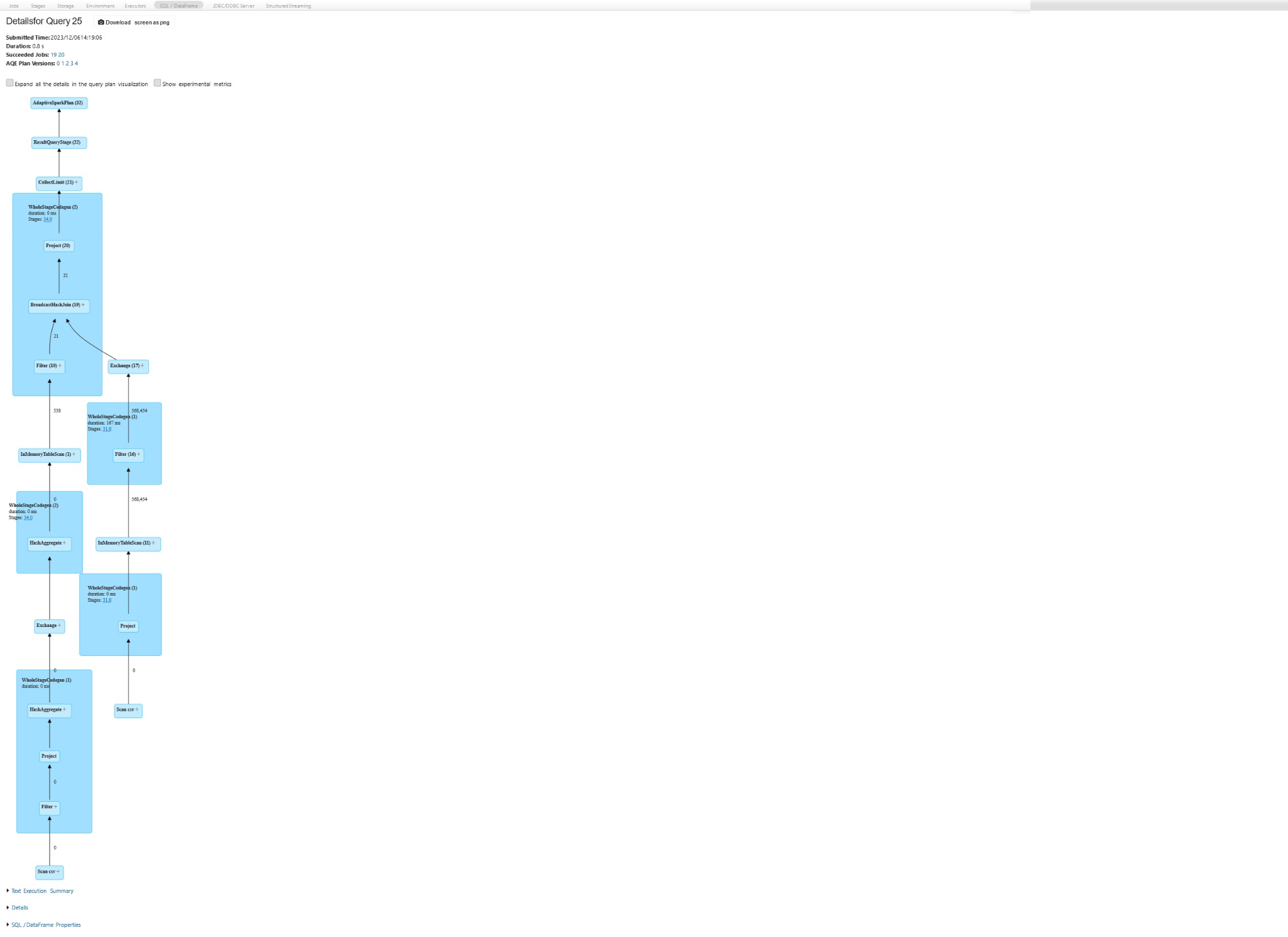
.mode("overwrite")\

.option("path", "/mnt/finefooddata/staging/scores\_filtered")\

.save()

Join Plan With df and sql, both show the same join plan  
  
joined\_df = reviews\_df.join(scores\_df, reviews\_df.Id == scores\_df.Id, "inner").drop(scores\_df.Id)

joined\_df.show()



reviews\_df.createOrReplaceTempView("reviews\_tt")

scores\_df.createOrReplaceTempView("scores\_tt")  
spark.sql("select \* from reviews\_tt re inner join scores\_tt ss on re.Id == ss.Id").show()

**5. Disable the broadcast join by changing the threshold and perform a join**

**again. Now see what join strategy was used.**spark.conf.set('spark.sql.autoBroadcastJoinThreshold',-1)

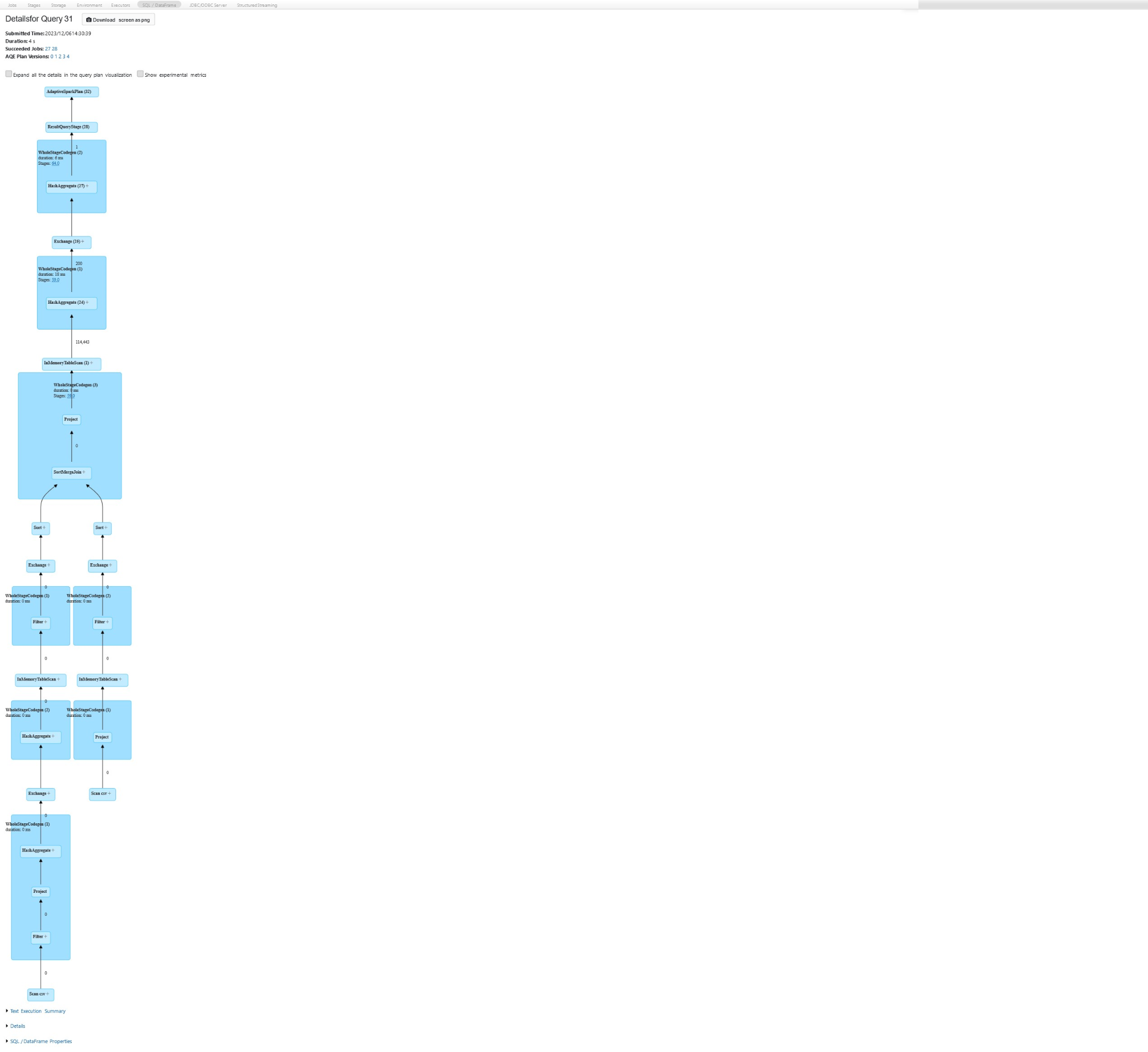
spark.conf.get('spark.sql.autoBroadcastJoinThreshold')

  
joined\_df = reviews\_df.join(scores\_df, reviews\_df.Id == scores\_df.Id, "inner").drop(scores\_df.Id)

joined\_df = joined\_df.cache()

joined\_df.count()

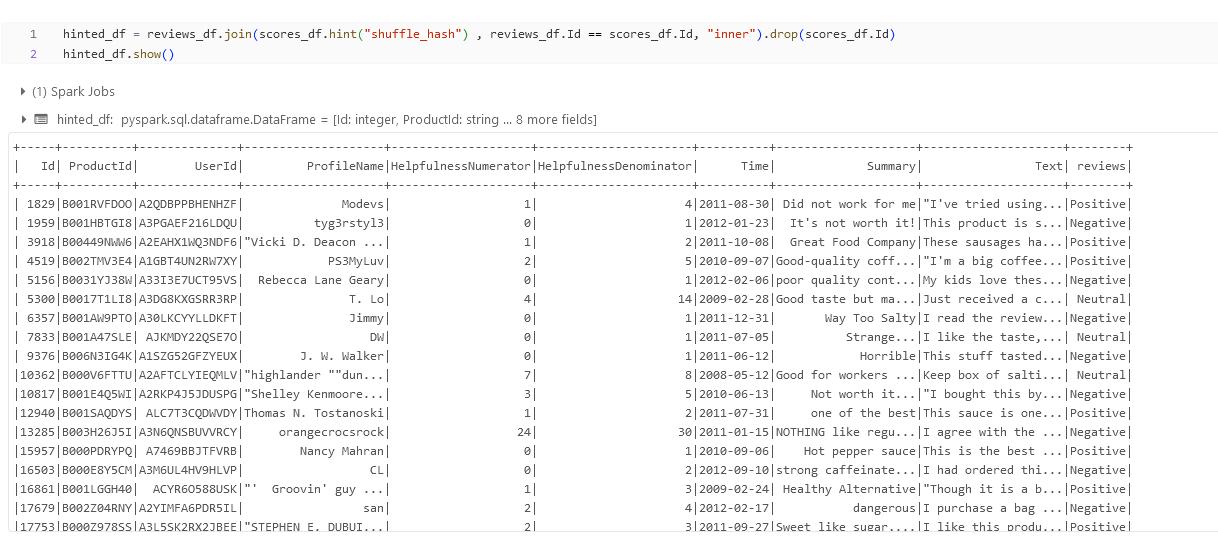
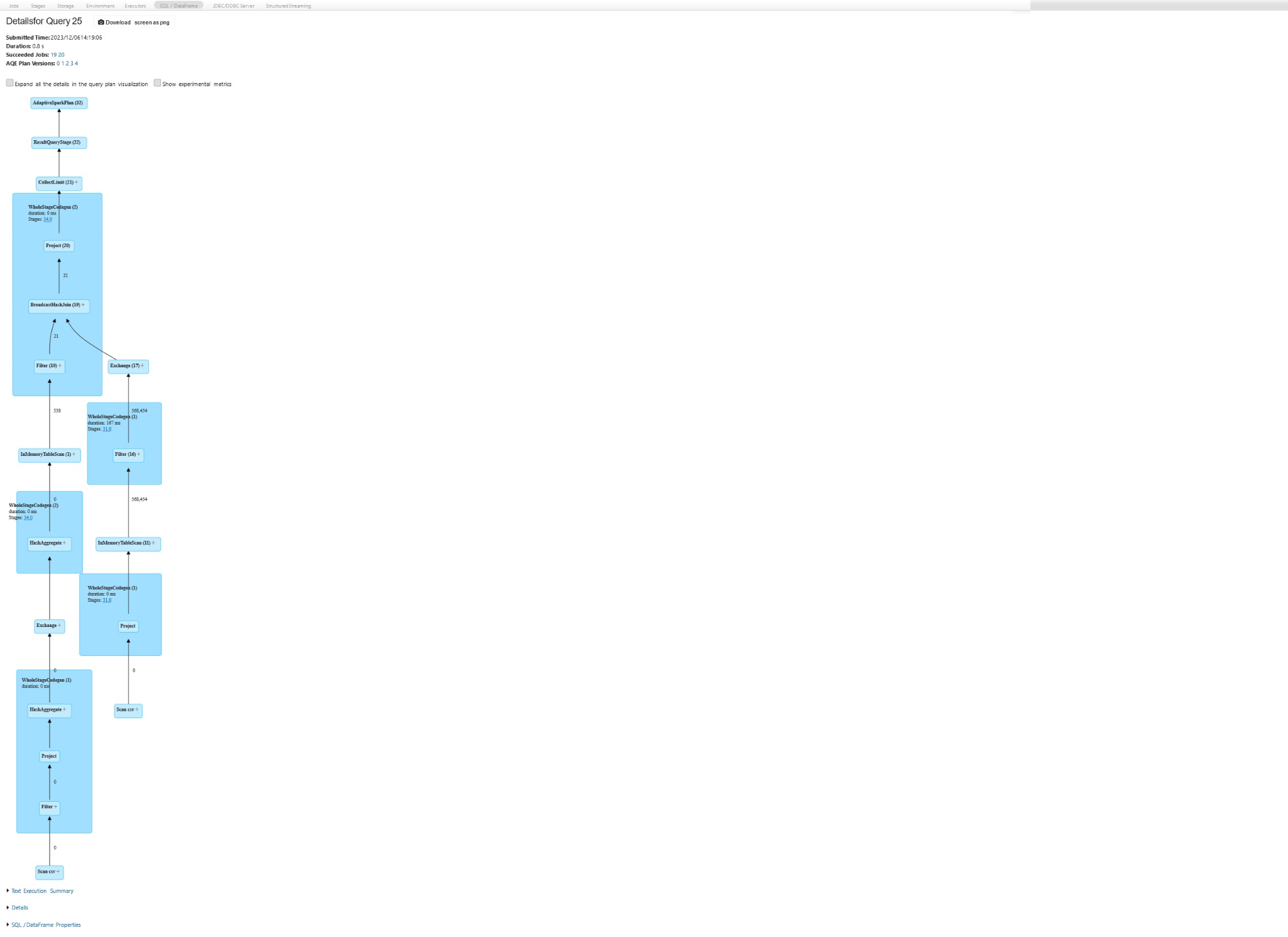
joined\_df.count()

****

**6. Give a hint for shuffle hash join and invoke the join again and check the**

**spark UI for the join strategy used**hinted\_df = reviews\_df.join(scores\_df.hint("shuffle\_hash") , reviews\_df.Id == scores\_df.Id, "inner").drop(scores\_df.Id)

hinted\_df.show()

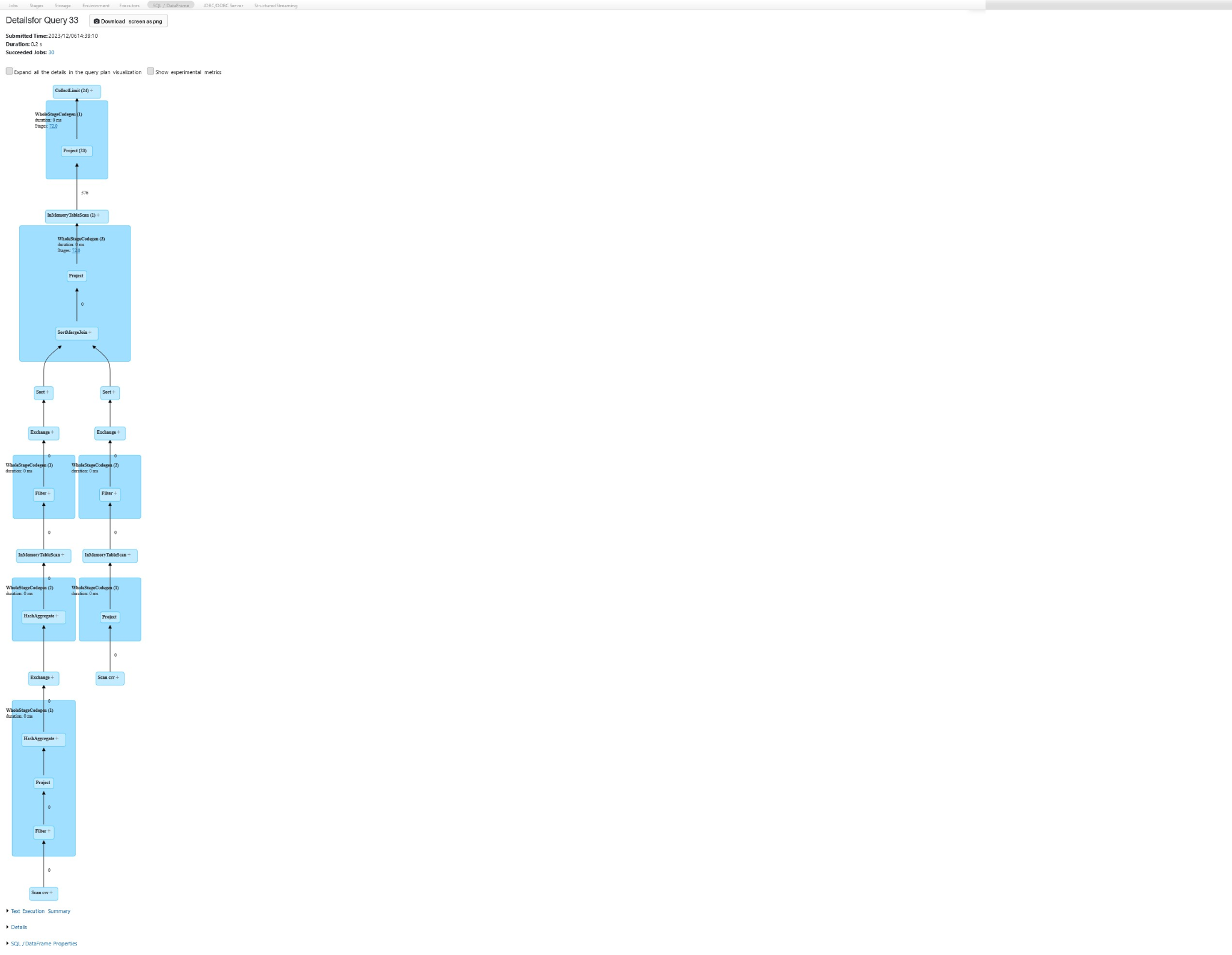
**7. By default the AQE was enabled, disable the AQE and perform a join again**

**and see if there is a change in number of shuffle partitions.**spark.conf.set("spark.sql.adaptive.enabled",False)

spark.conf.get("spark.sql.adaptive.enabled")

joined\_without\_aqe = reviews\_df.join(scores\_df, reviews\_df.Id == scores\_df.Id, "inner").drop(scores\_df.Id)

joined\_without\_aqe.show()

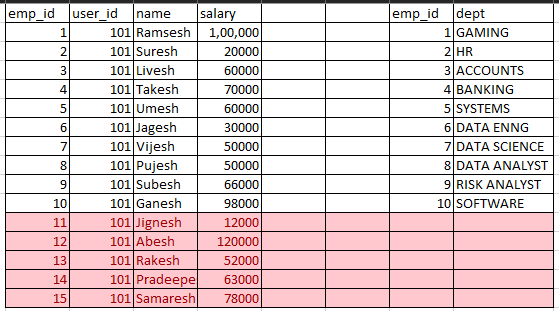
**  
**

**8. You need to explain left outer and semi join with relevant usecase, consider**

**any datasets of your choice which make sense. demonstrate it by running in**

**the notebook.**

* Here is what our data looks like :

****

* **Left join :**

test\_join = left\_df.join(right\_df, left\_df.emp\_id == right\_df.emp\_id, "left").drop(right\_df.emp\_id)

test\_join.show()

****

* **Semi Join :**  This join shows the matching records from both the tables or the records where at least one emp\_id with salary has a specific department  
    
  test\_join = left\_df.join(right\_df, left\_df.emp\_id == right\_df.emp\_id, "semi").drop(right\_df.emp\_id)

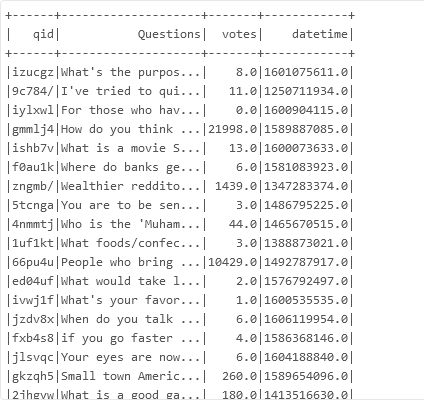
test\_join.show()



**9. Consider any 2 datasets of your choice (ideally 2 large datasets) which are**

**more than 1 GB each in our case. Demonstrate a bucket sort merge join**

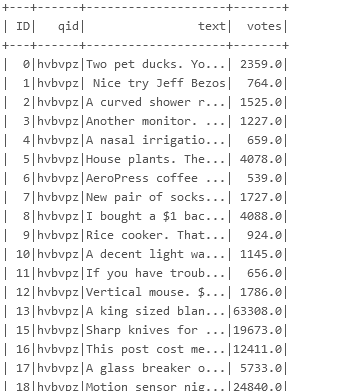
* Dataset : <https://www.kaggle.com/datasets/rodmcn/askreddit-questions-and-answers/code>
* We can not directly do bucketing in databricks because in databricks, all tables are saved as delta live tables and for some reason it is not possible to do bucketing here.
* So, it is better to write the files in the disc are Parquet the read those parquet files and then o bucketing
* dbutils.fs.mount(source = "wasbs://reddit@finefooddata.blob.core.windows.net", mount\_point = "/mnt/finefooddata/reddit", extra\_configs = {"fs.azure.account.key.finefooddata.blob.core.windows.net":"BFXqu52ydqWJ4WSTchnAGvGLdKY69OxYB20CpMxmSraxSZBbQlQBEggovAbc81OcrQtWX8ZBfoLF+AStQkhkwQ=="})
* questions\_schema = "qid string, Questions string, votes float, datetime string"
* reddit\_questions = spark.read.format("csv").schema(questions\_schema).option("sep",";").load("dbfs:/mnt/finefooddata/reddit/reddit\_questions.csv", skiprows = 1)
* reddit\_questions.show()



* reddit\_questions.write.format("parquet").option("path", "/mnt/finefooddata/reddit/questions").save()
* reddit\_answers\_long\_schema = "ID int, qid string, text string, votes float"
* reddit\_answers\_long = spark.read.format("csv").option("sep",";").schema(reddit\_answers\_long\_schema).load("dbfs:/mnt/finefooddata/reddit/reddit\_answers\_long.csv", skiprows = 1)

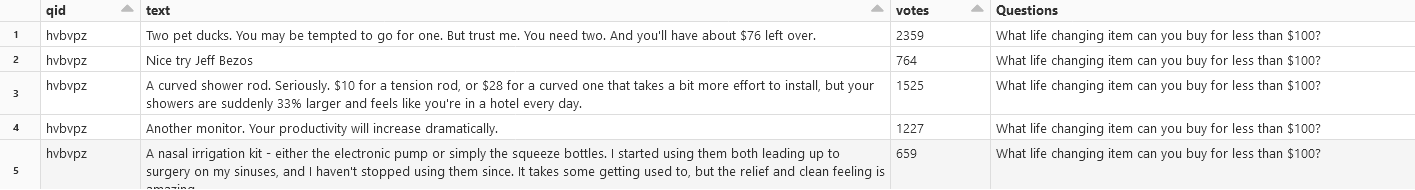
reddit\_answers\_long = reddit\_answers\_long.cache()

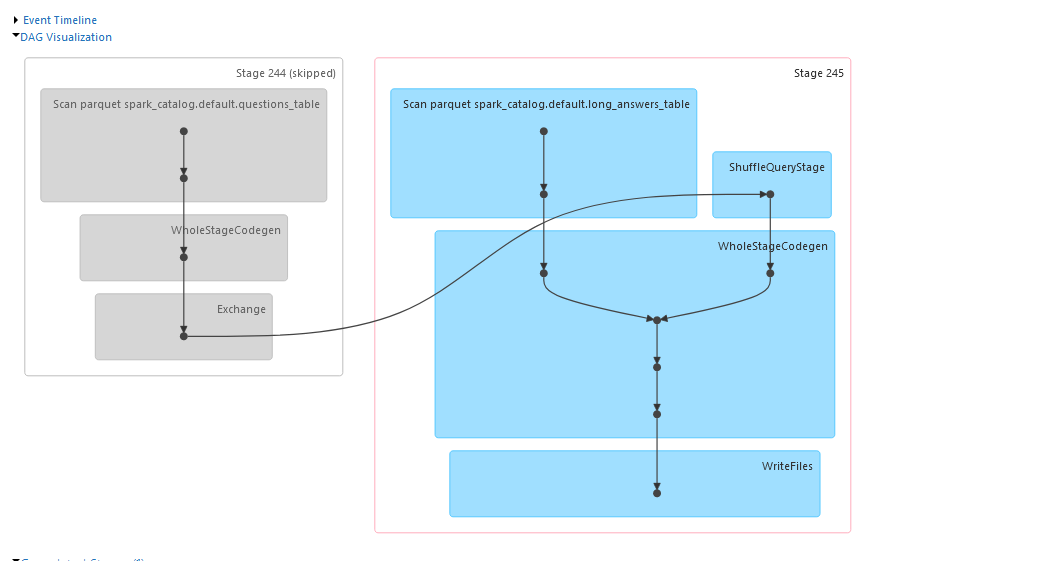
reddit\_answers\_long.show()



* reddit\_answers\_long.write.format("parquet").option("path", "/mnt/finefooddata/reddit/long\_answers").save()
* reddit\_answers\_long.write.format("parquet").bucketBy(8,"qid").option("path", "/mnt/finefooddata/reddit/short\_answers/bucket/long\_answers").saveAsTable("long\_answers\_table")
* reddit\_questions.write.format("parquet").bucketBy(8,"qid").option("path", "/mnt/finefooddata/reddit/short\_answers/bucket/questions\_table").saveAsTable("questions\_table")
* spark.sql("select la.qid, la.text, la.votes, qt.Questions from long\_answers\_table la left join questions\_table qt on la.qid==qt.qid").write.format("parquet").mode("overwrite").option("path", "/mnt/finefooddata/reddit/joined\_stage\_1").saveAsTable("joined\_1\_table")
* %sql

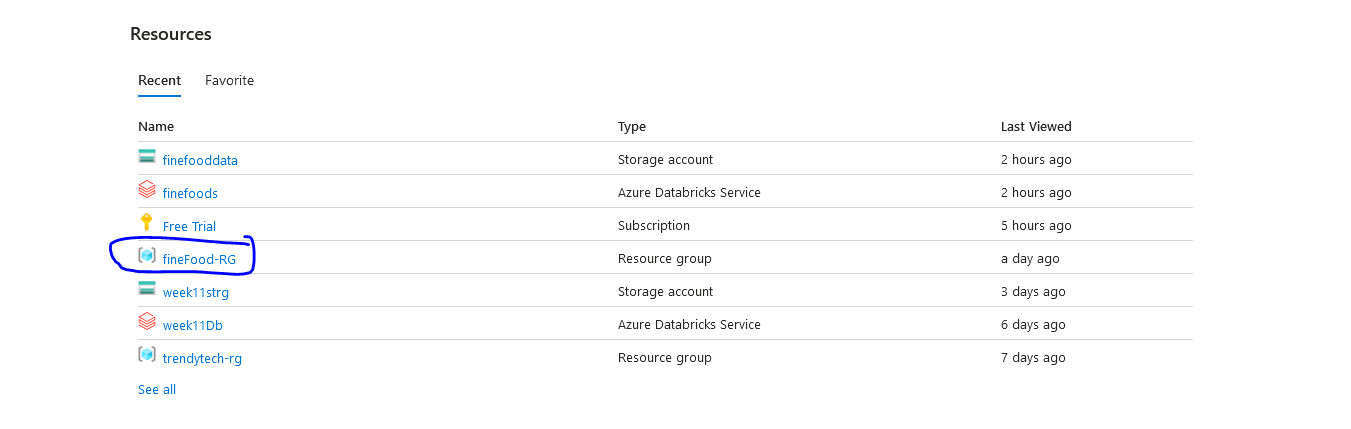
select \* from joined\_1\_table

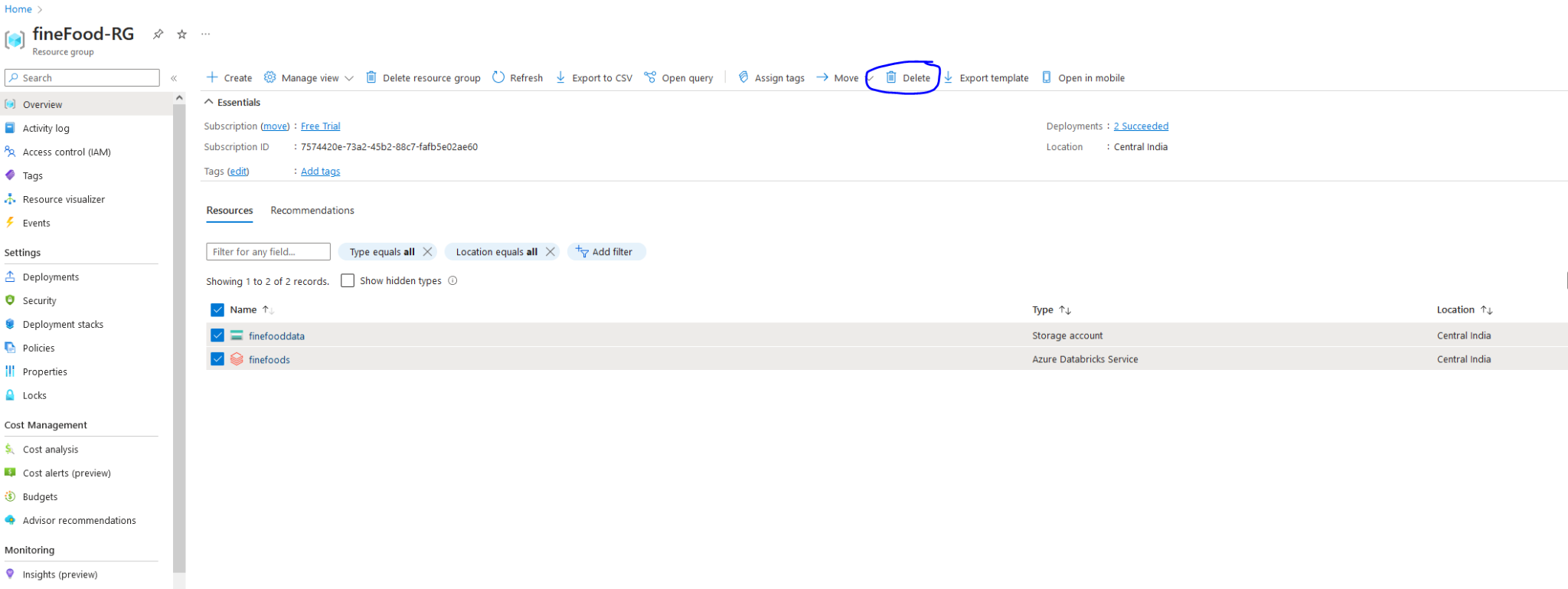
* 



Command took 27.30 seconds -- by dattaankan16@outlook.com at 7/12/2023, 3:04:39 pm on Ankan Datta's Cluster

**10. Delete the resources that you have created**

****

****