Creating a Search Engine with TF-IDF Algorithm for Wikipedia Data using Apache Spark on AWS EMR Studio to Analyze and Rank Relevant Documents.

Naveen Raju Sreerama Raju Govinda Raju
naveenraju100@gmail.com | +1 224 706 7718 | +91 9886157101 |
https://www.linkedin.com/in/naveen-raju-s-g-bb1486124
https://naveenrajusg.github.io/Portfolio/

Table of Contents

1) Overview

2) Create and configure Amazon EMR Studio
- **A) Studio name and Description**
- **B) Networking and Security**
- **C) Studio service role**
  - **I) Create an IAM role**
  - **II) Configure Studio Service role**
  - **III) Create a S3 bucket**
  - **IV) Continue configuring Studio Service Role**

3) Create Amazon EMR Workspace

4) Cluster creation

5) Attach cluster to workspace and launch PySpark Notebook

6) Code TF-IDF PySpark code

7) Deleting Resources
- **A) Delete workspace**
- **B) Terminate Cluster**
- **C) Delete EMR Studio**
- **D) Delete S3 buckets**

1) Overview

Apache PySpark:-

Apache PySpark is an open-source distributed computing framework built on top of Apache Spark and designed for processing and analyzing large-scale data sets. It provides a Python API that allows developers to harness the power of Spark's distributed computing capabilities to perform data transformations, data manipulation, and machine learning tasks efficiently and at scale. In our project, we utilized PySpark on EMR Studio to implement a TF-IDF algorithm for preparing and analyzing Wikipedia data, ultimately building a custom search engine for relevant document retrieval.

TF-IDF (Term Frequency - Inverse Document Frequency) :

Term Frequency just measures how often a word occurs in a document. (A word that occurs frequently is probably important to that document's meaning)

Number of times term t appears in document d
Total number of terms in document d

$$\text{Measure of term frequency of term t in a document d} = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

Document Frequency is how often a word occurs in an entire set of documents, I.e., all of Wikipedia or every web page. (This tells us about common words that just appear everywhere o matter what the topic, like "a", "the", "and", etc.)

$$\text{Measure of relevancy of a word to a document} = \frac{Term\ Frequency}{log\ (Document\ Frequency)}$$

Or

Measure of relevancy of a word to a document = Term Frequency * log (Inverse Document Frequency)

That is, it takes how often the word appears in a document, over how often it just appears everywhere. That gives you a measure of how important and unique this word is for this document.

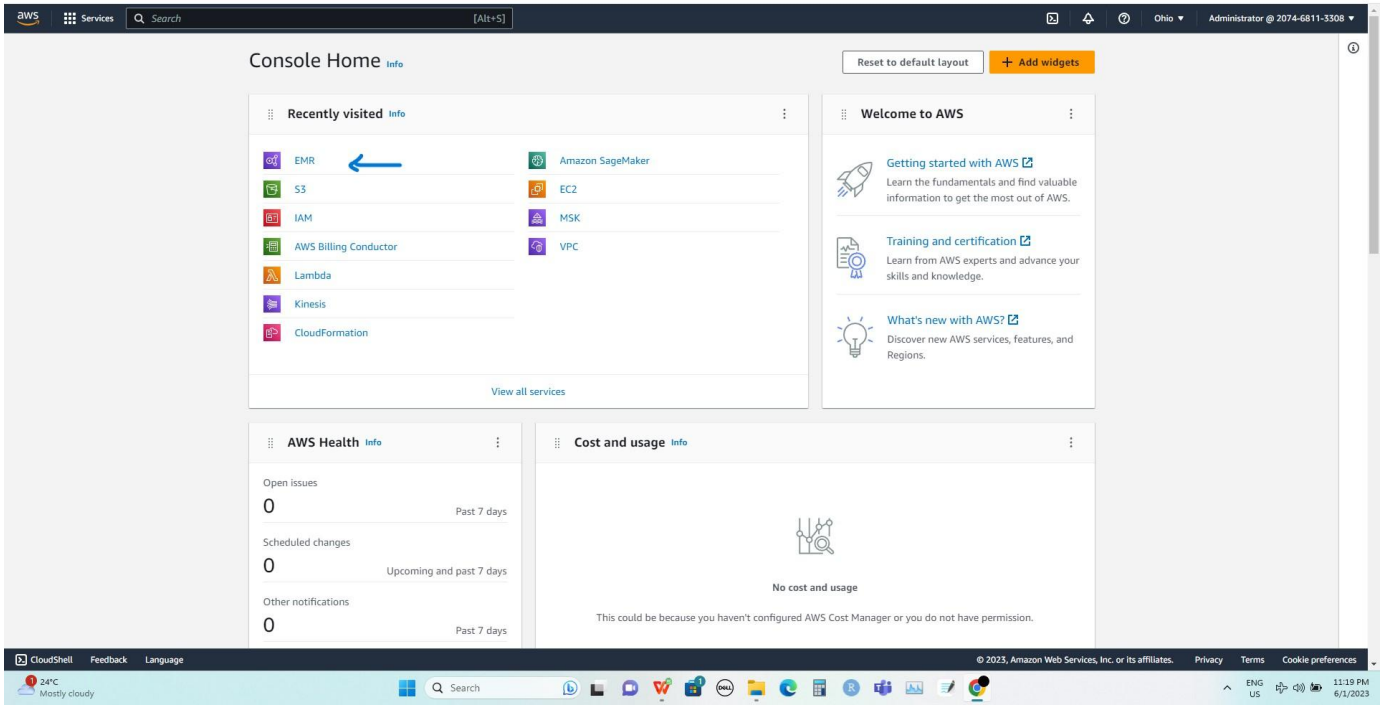
Usually log of the IDF is used, since word frequencies are distributed exponentially. Using log gives us a better weighting of a words overall popularity.

The rationale behind using the log transformation for IDF is to dampen the effect of extremely high and low IDF values. It helps to normalize the IDF scores and makes them more manageable. The IDF value of a term increases logarithmically with the number of documents in the collection that contain the term.

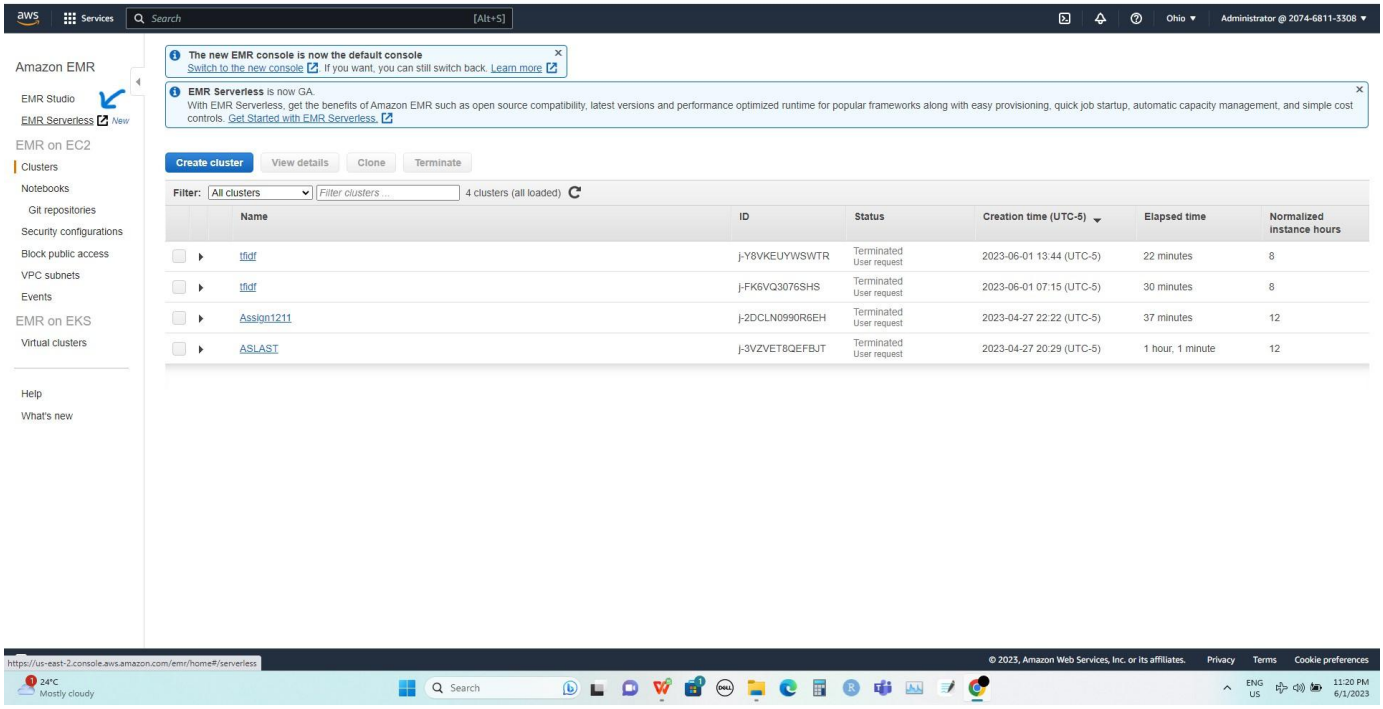
The formula for IDF is typically given as:
$IDF(t) = \log(N / (DF(t)))$
where:
N is the total number of documents in the collection.
DF(t) is the document frequency of term "t," i.e., the number of documents that contain the term.
By taking the logarithm, we can handle a wide range of values and prevent the IDF from being dominated by very large or very small numbers.

2) Create and configure Amazon EMR Studio

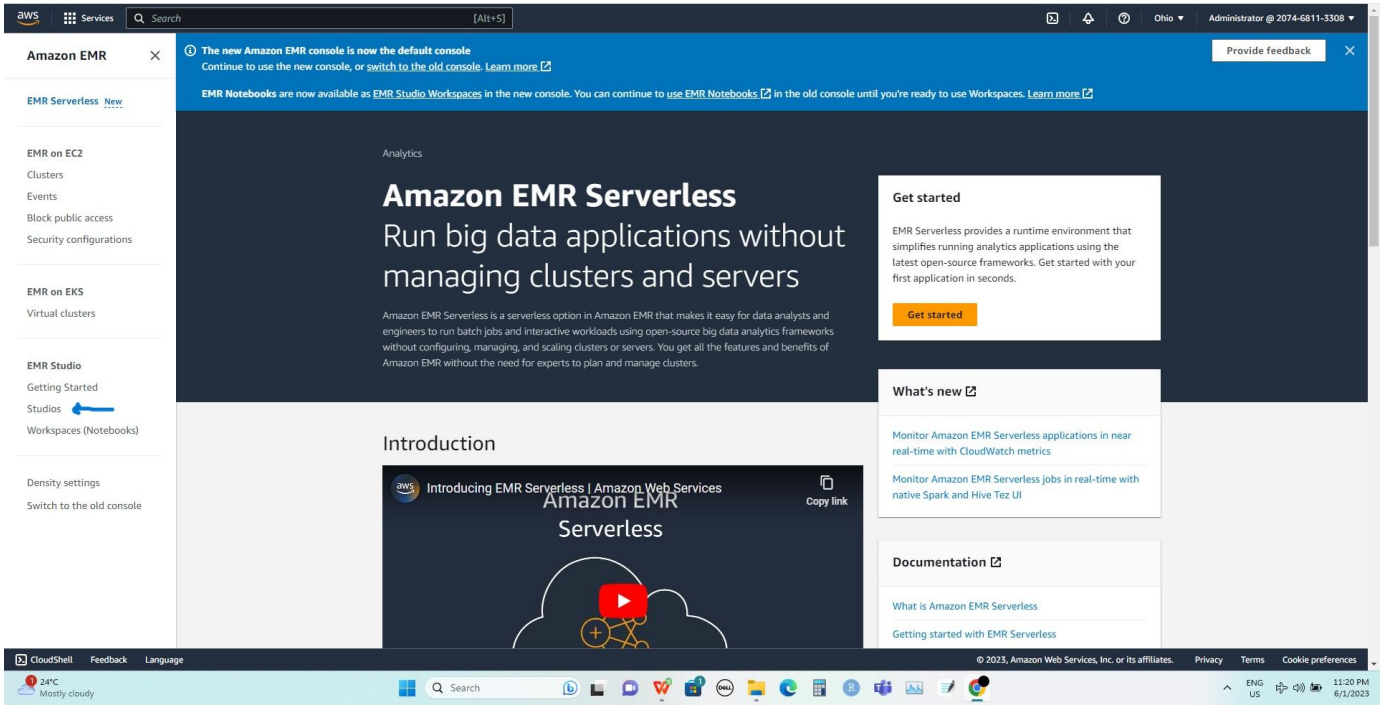
From AWS console navigate to EMR



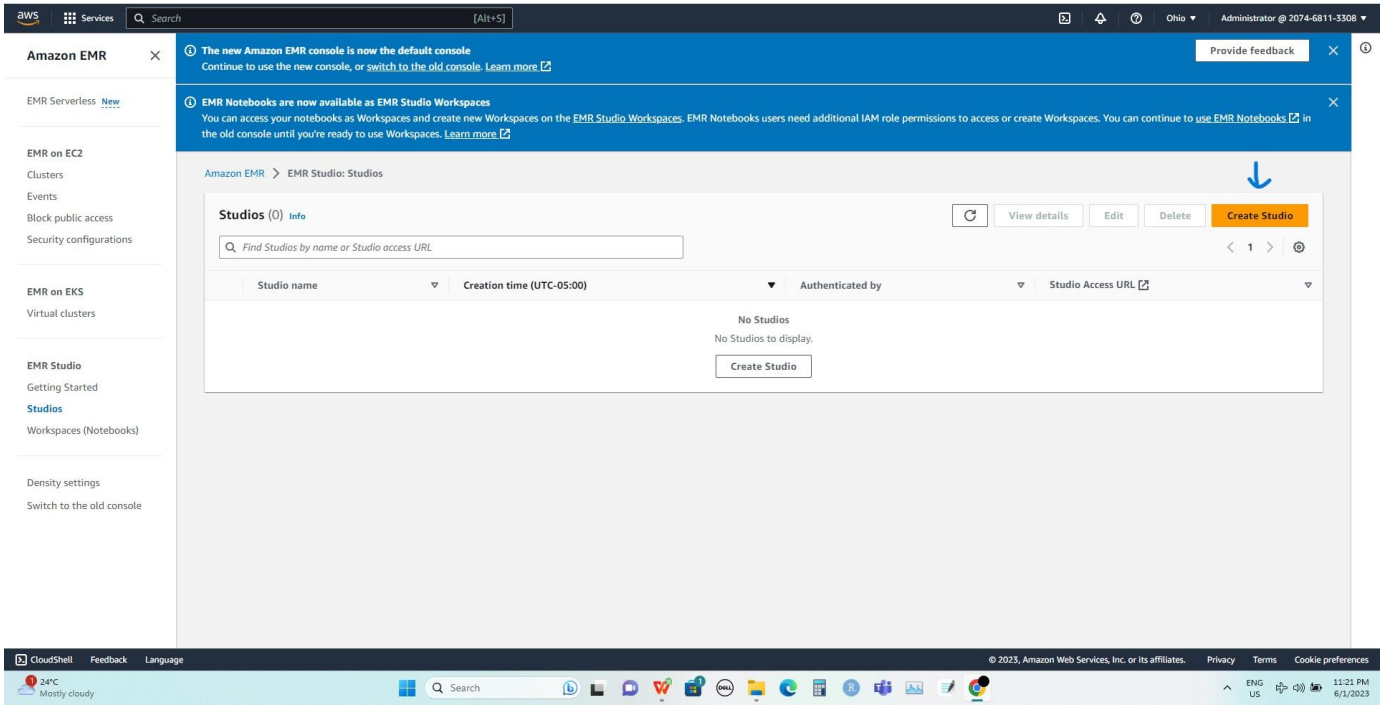
From AWS EMR console click on EMR serverless



Navigate to EMR Studio - > Studios



Click on Create Studio

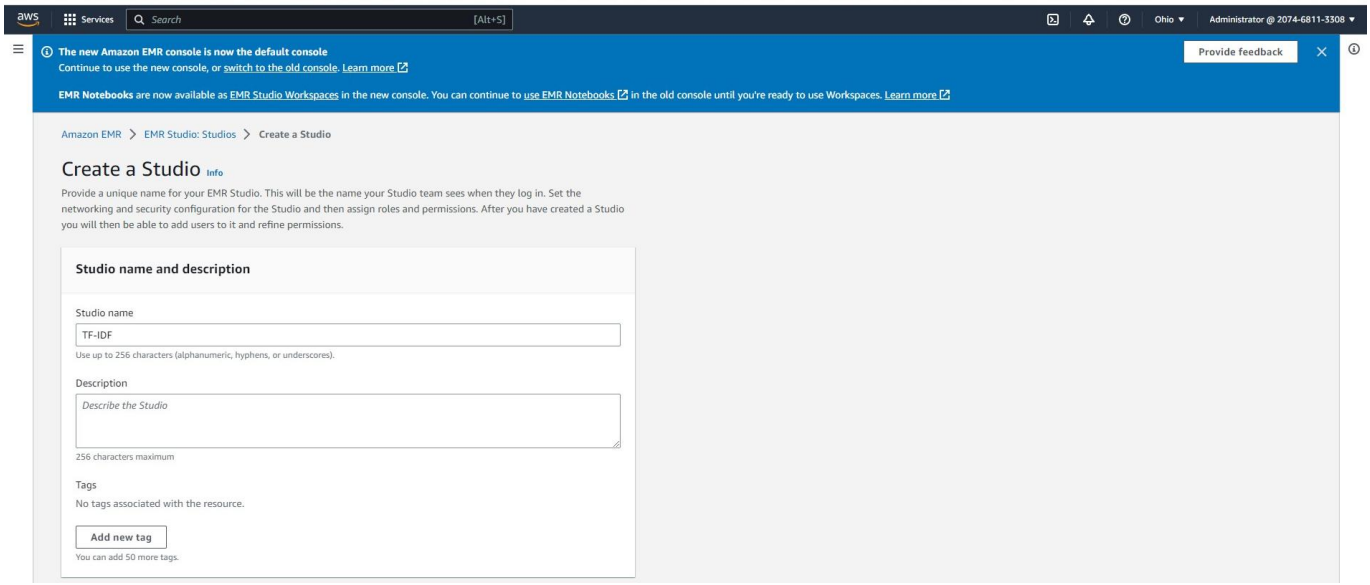


A) <u>Studio name and Description:</u>

<u>Set configuration of a Studio:</u>

<u>Studio name and Description</u>

Studio name : TF-IDF



B) <u>Networking and Security</u>

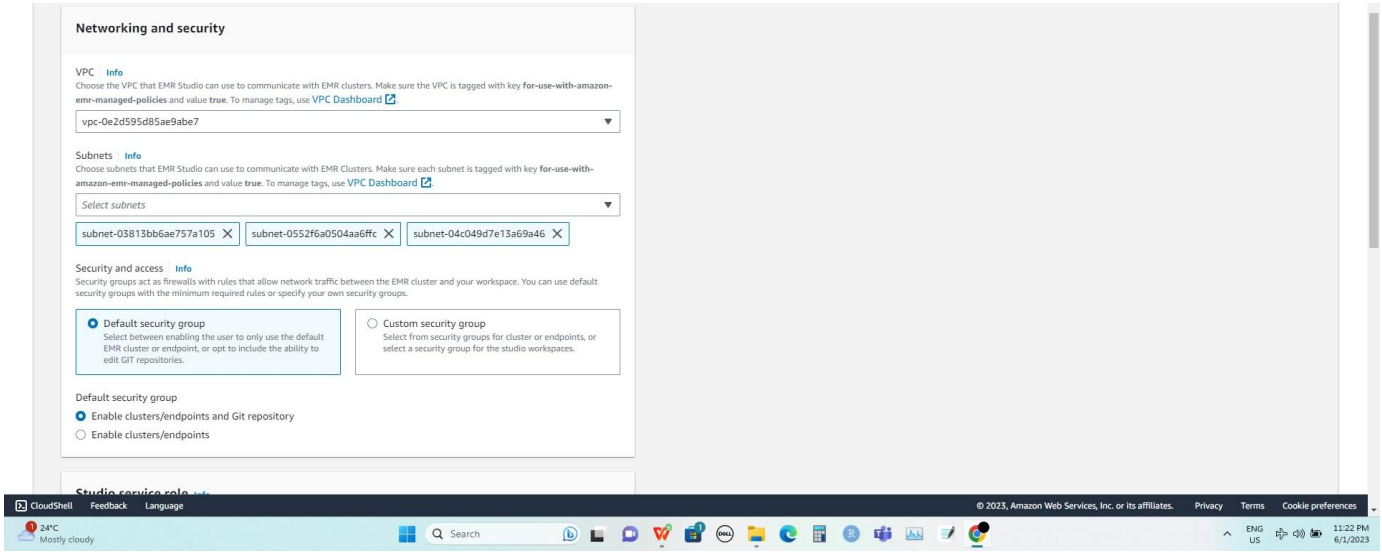
VPC:
VPC (Choose the VPC that EMR Studio can use to communicate with EMR clusters.
From the dropped down select VPC that is usually associated with our AWS account)

Subnets : Choose subnets that EMR Studio can use to communicate with EMR Clusters.
From the dropped down select all 3 subnets associated with our account.

Security and access:
Security group act as firewalls with rules that allow network traffic between the EMR duster and your workspace. You can use default arty groups with the minimum required nudes

Select Default security group

Deafult security group:
Select Enable clusters/endpoints and Git repository



C) Studio service role

Authentication: AWS Identity and Access Management (IAM)

Service role:

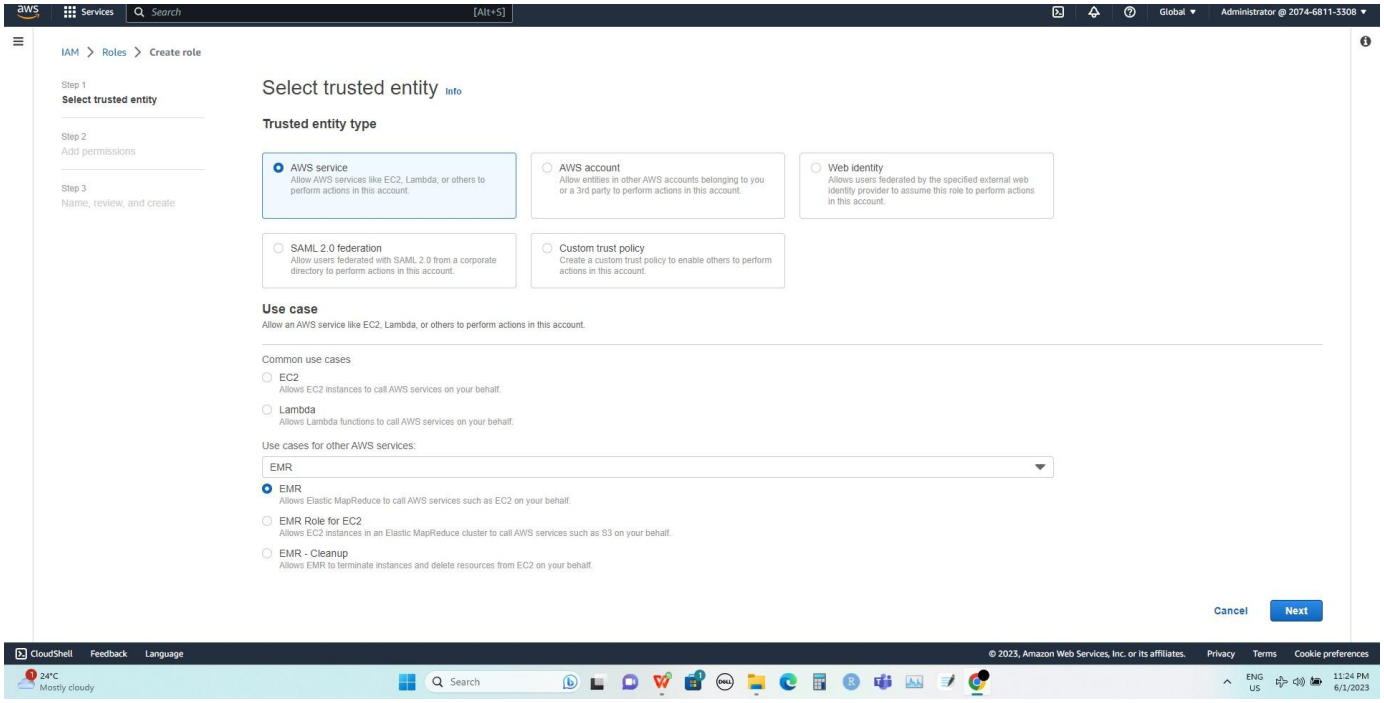
The service role defines the allowable actions for EMR Studio when provisioning resources. Examples of such actions included attaching a workspace to a cluster or accessing the S3 backup location for workspace.

I) Create an IAM role

From AWS console navigate to Identity and Access Management (IAM) - > Roles - > Create role

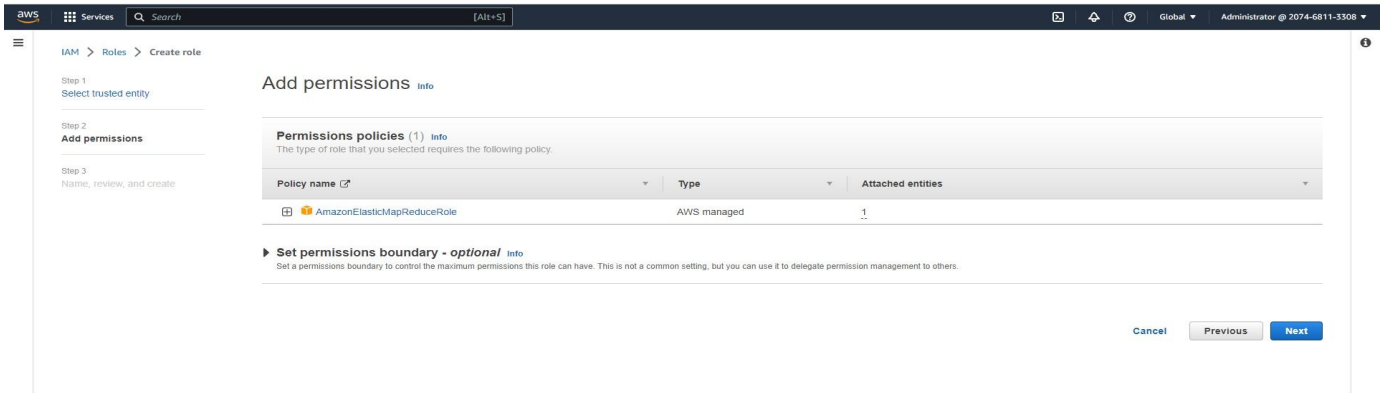
Step A - Select trusted entity:

Trusted entity type : AWS service (Allows AWS services like EC2, Lambda, or others to perform actions in this account)
Use case : EMR (Allows EC2 instance in an Elastic MapReduce cluster to call AWS services such as S3 on your behalf.)



Step B - Add permisions

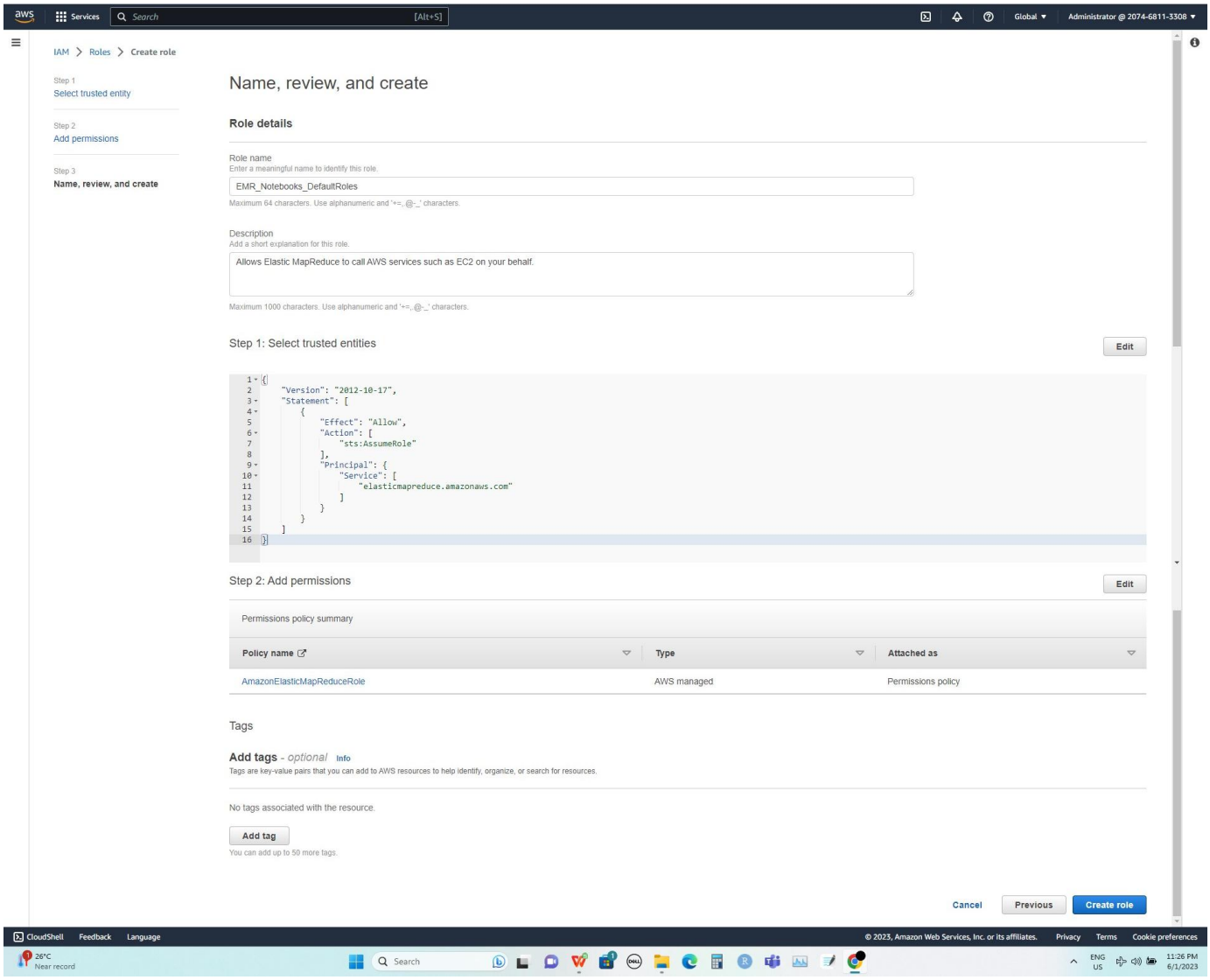
Permission Policies:
Policy name - AmazonElasticMapReduceRole (would have been auto selected)



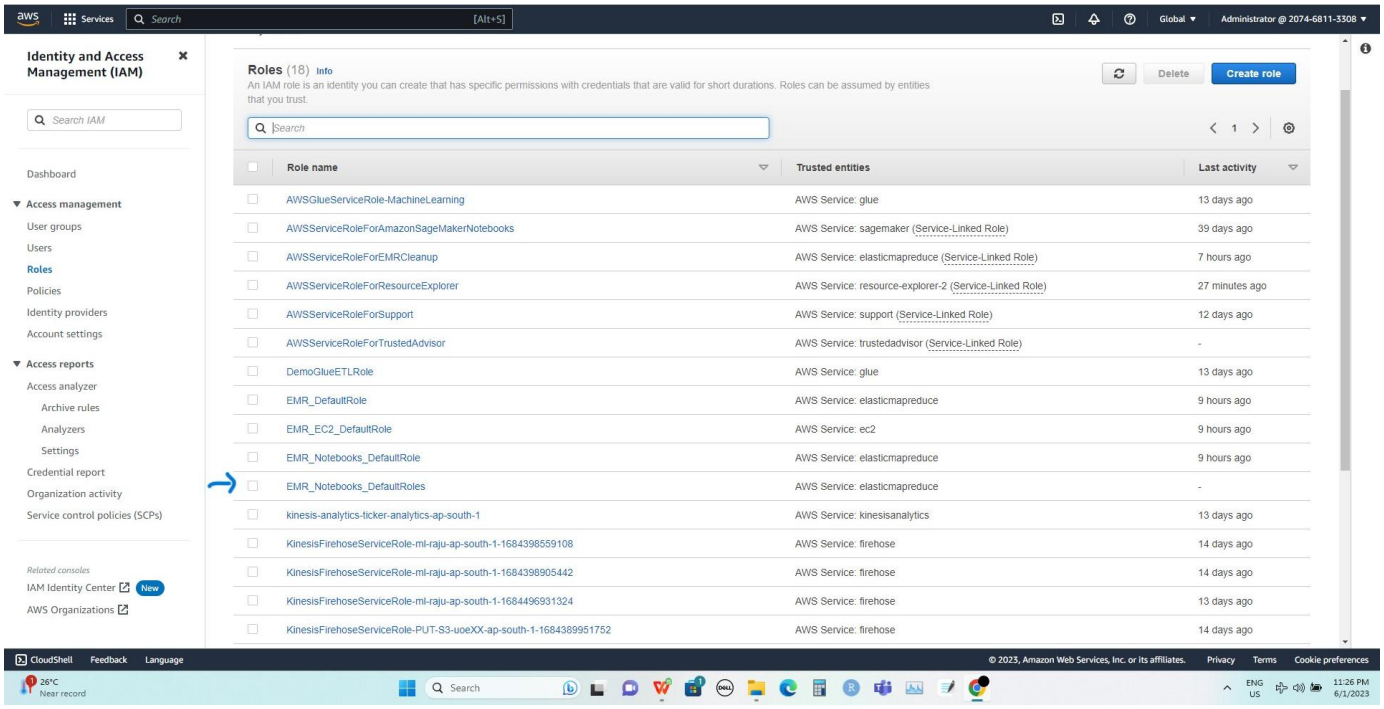
Step C - Name, review and create

Role name - EMR_Notebooks_DefaultRoles

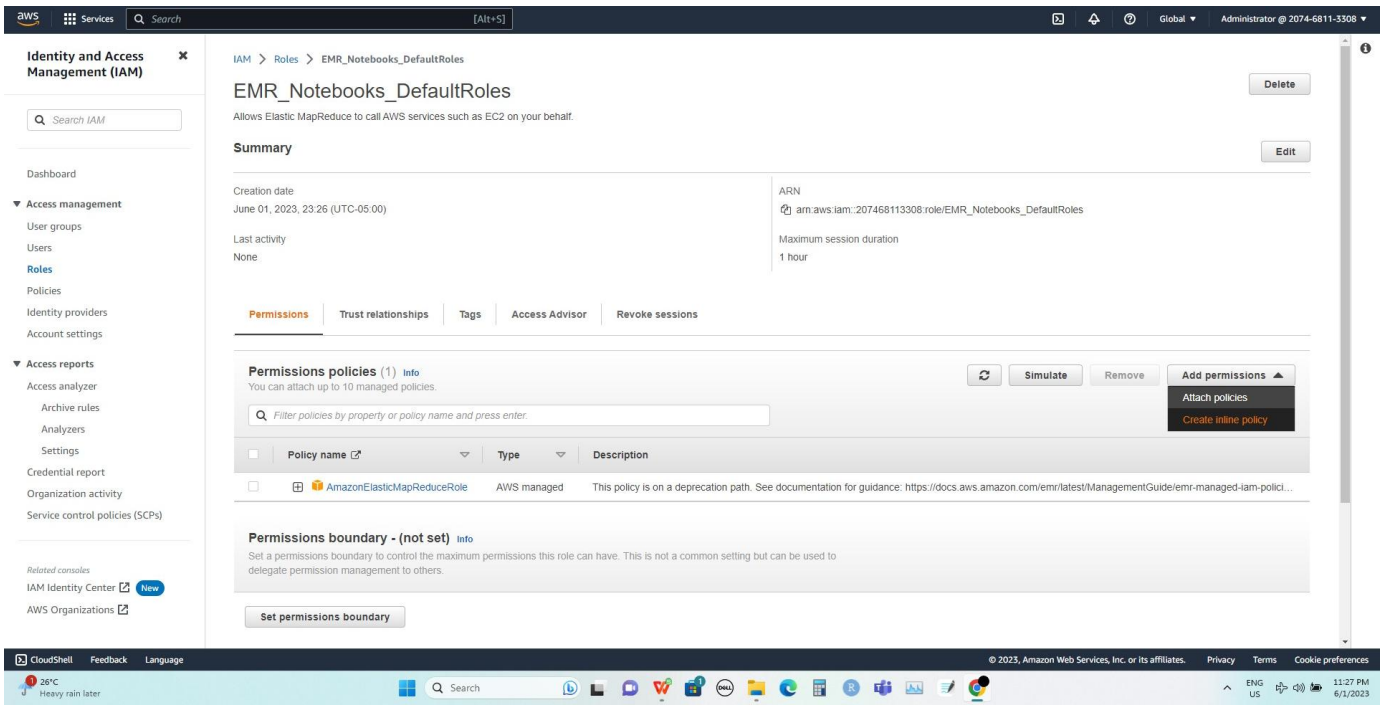
Click on Create role.



Navigate to IAM - > Roles - > EMR_Notebooks_DefaultRoles

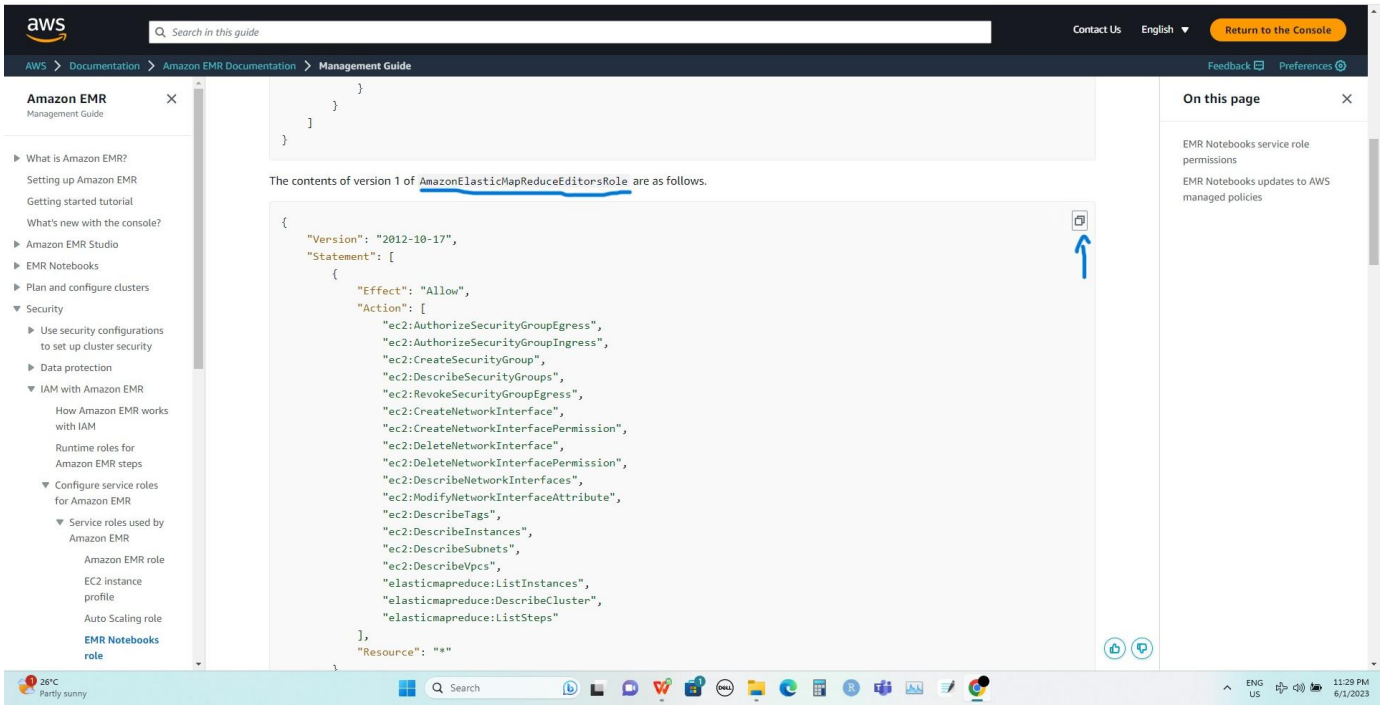


Click on attach policies

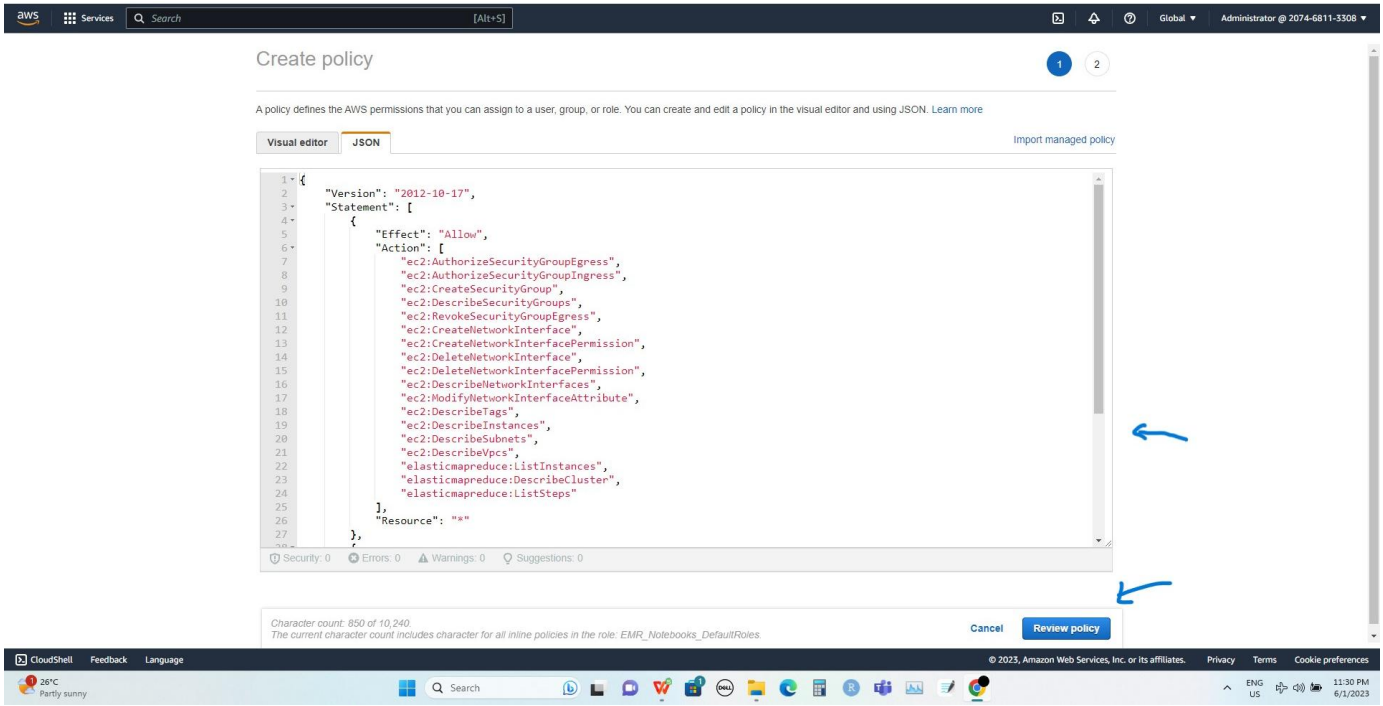


Visit the following web page : https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-managed-notebooks-service-role.html to fetch JSON's related to AmazonElasticMapReduceEditorsRole.

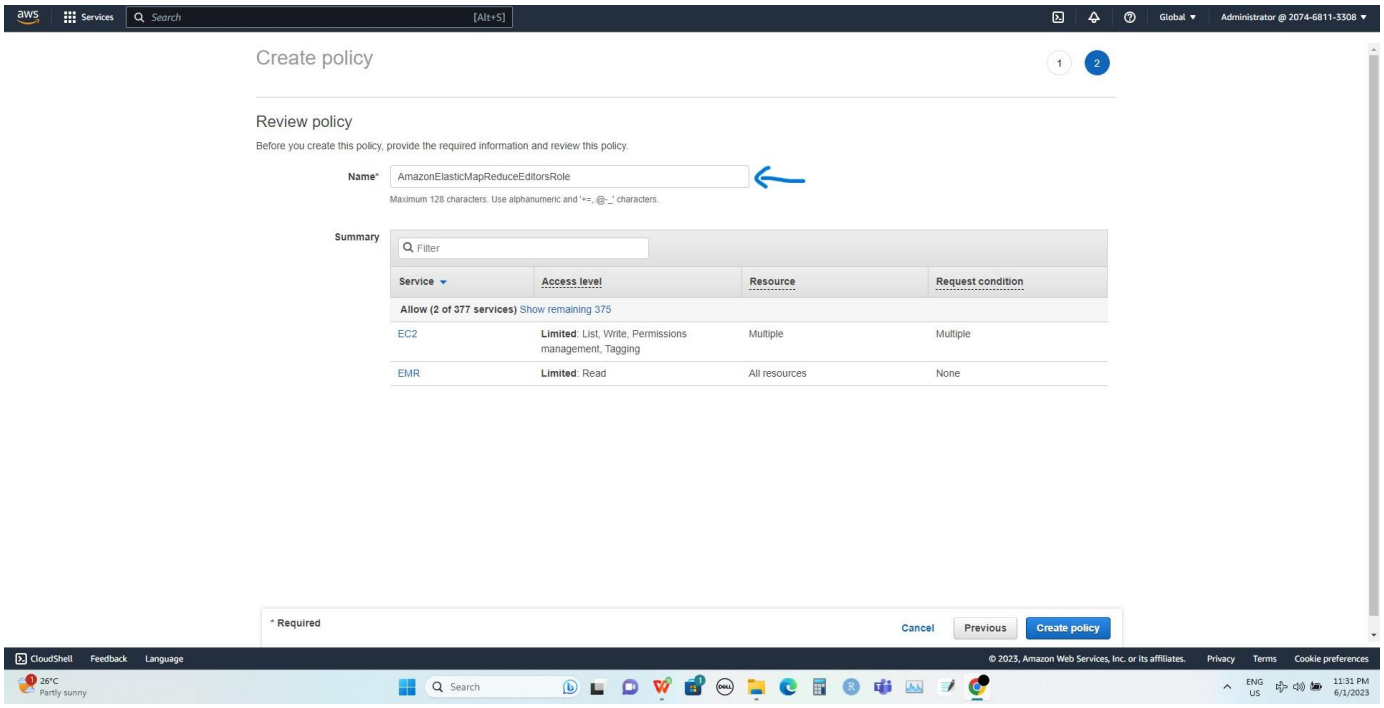
Copy the Json related to AmazonElasticMapReduceEditorsRole:



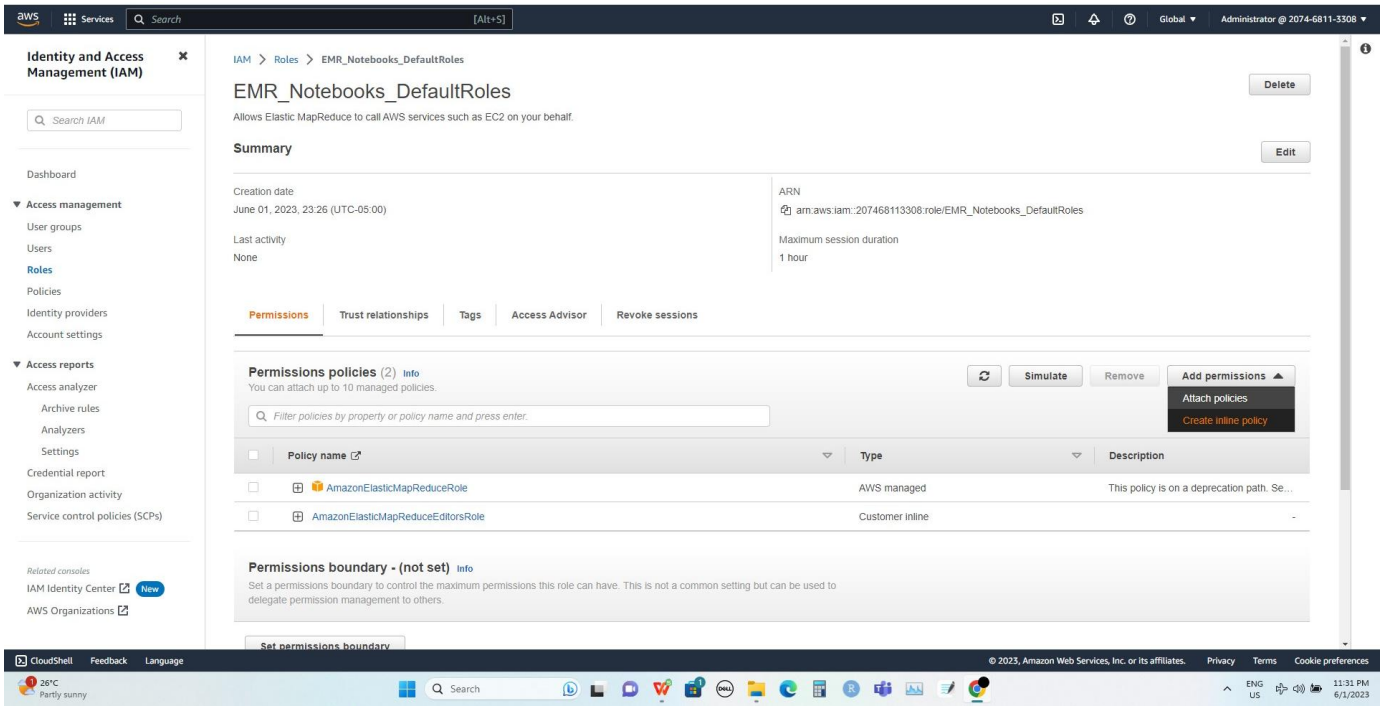
Continue in Attache policies web page:
Paste the AmazonElasticMapReduceEditorsRole Json and click on Review Policy



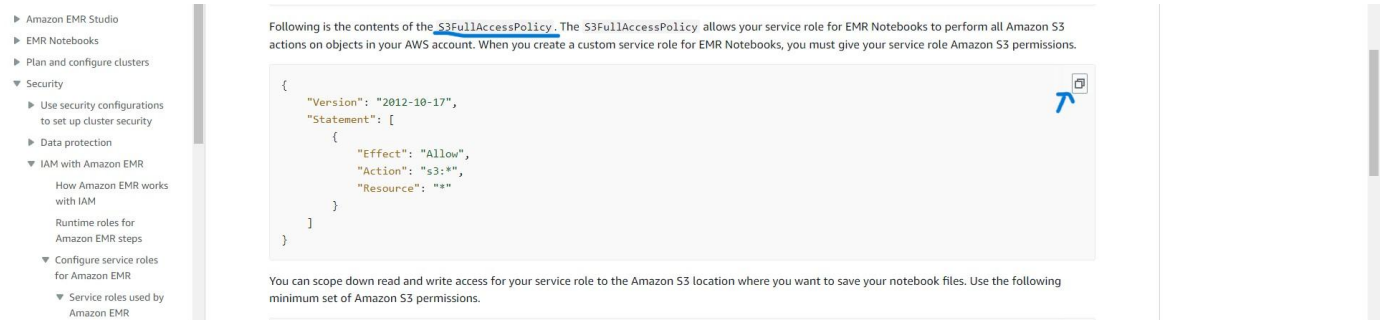
Name : AmazonElasticMapReduceEditorsRole



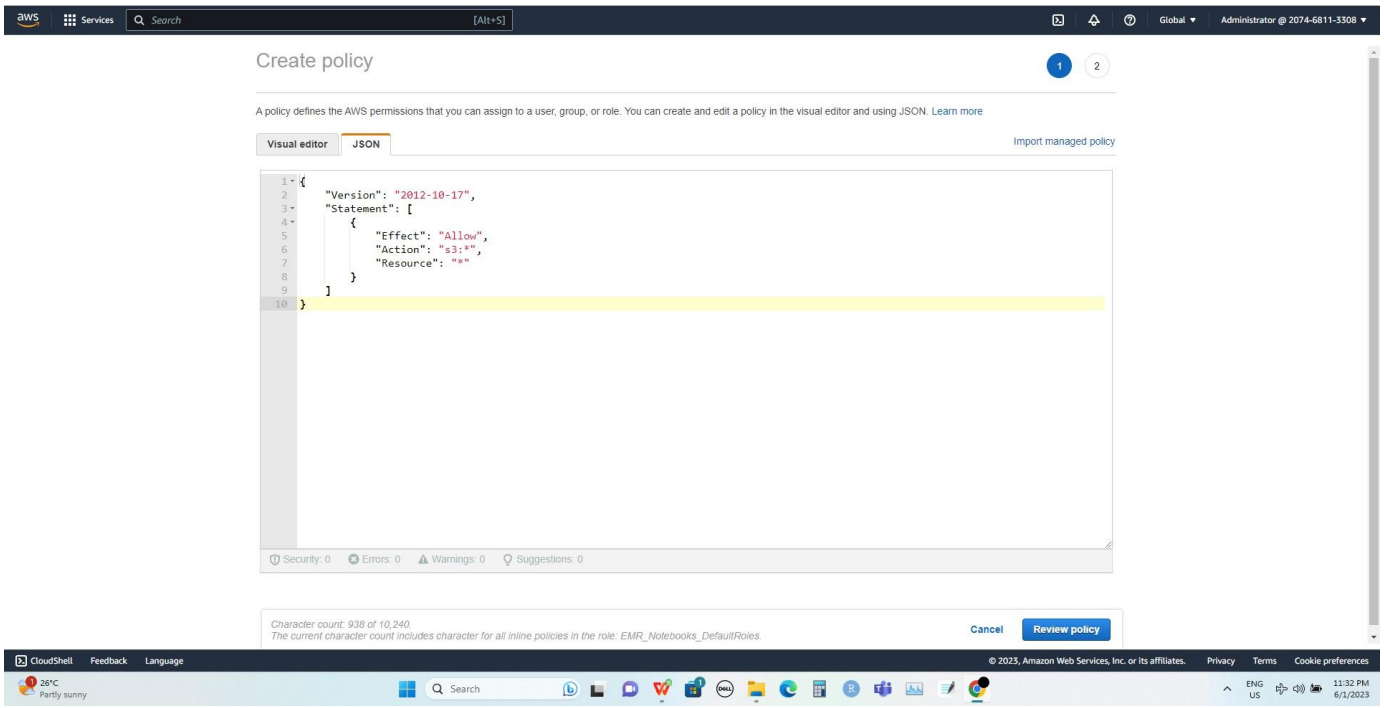
Again click on Attach Policy:

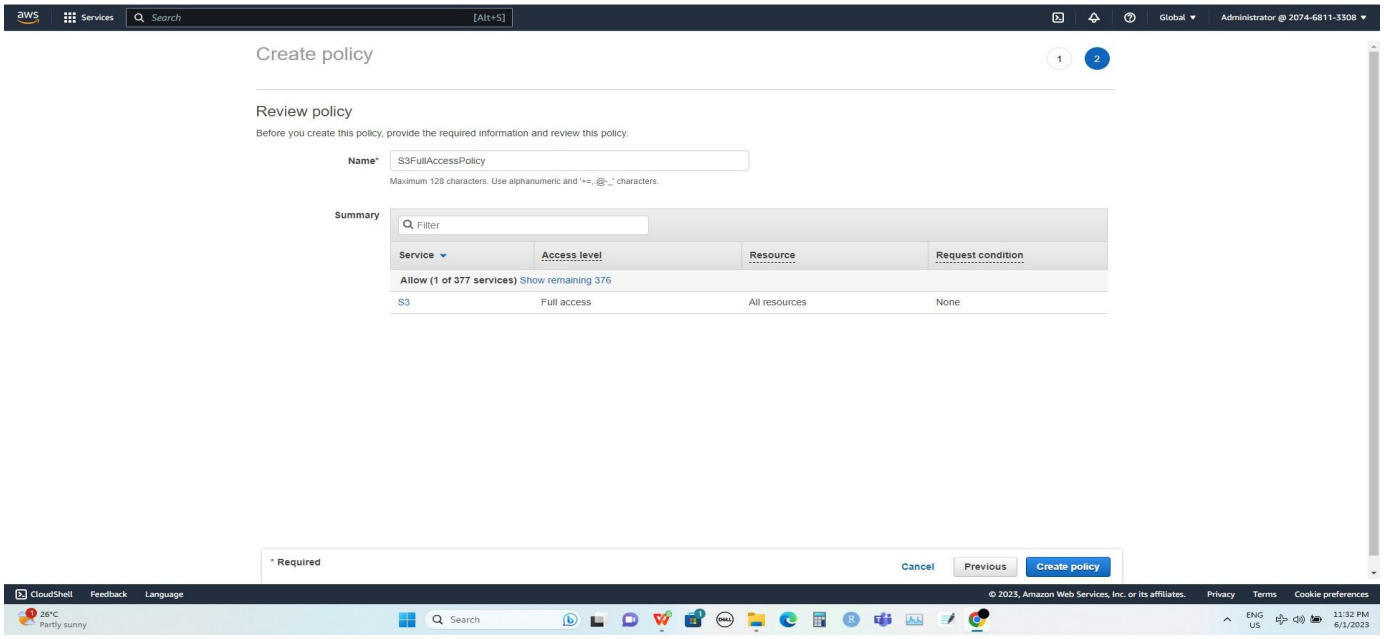


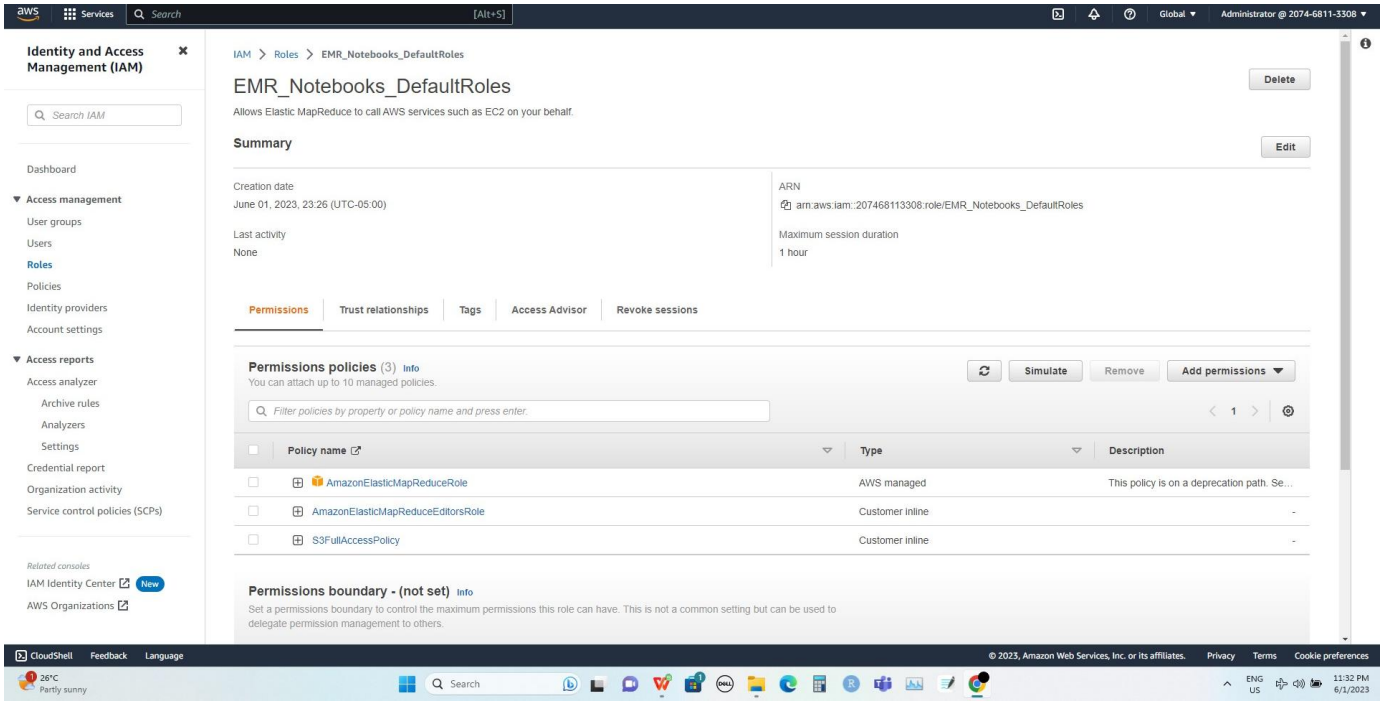
Visit the following web page : https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-managed-notebooks-service-role.html to fetch JSON's related to S3FullAccessPolicy. Copy the JSON.



Continue in Attache policies web page:
Paste the S3FullAccessPolicy Json and click on Review Policy





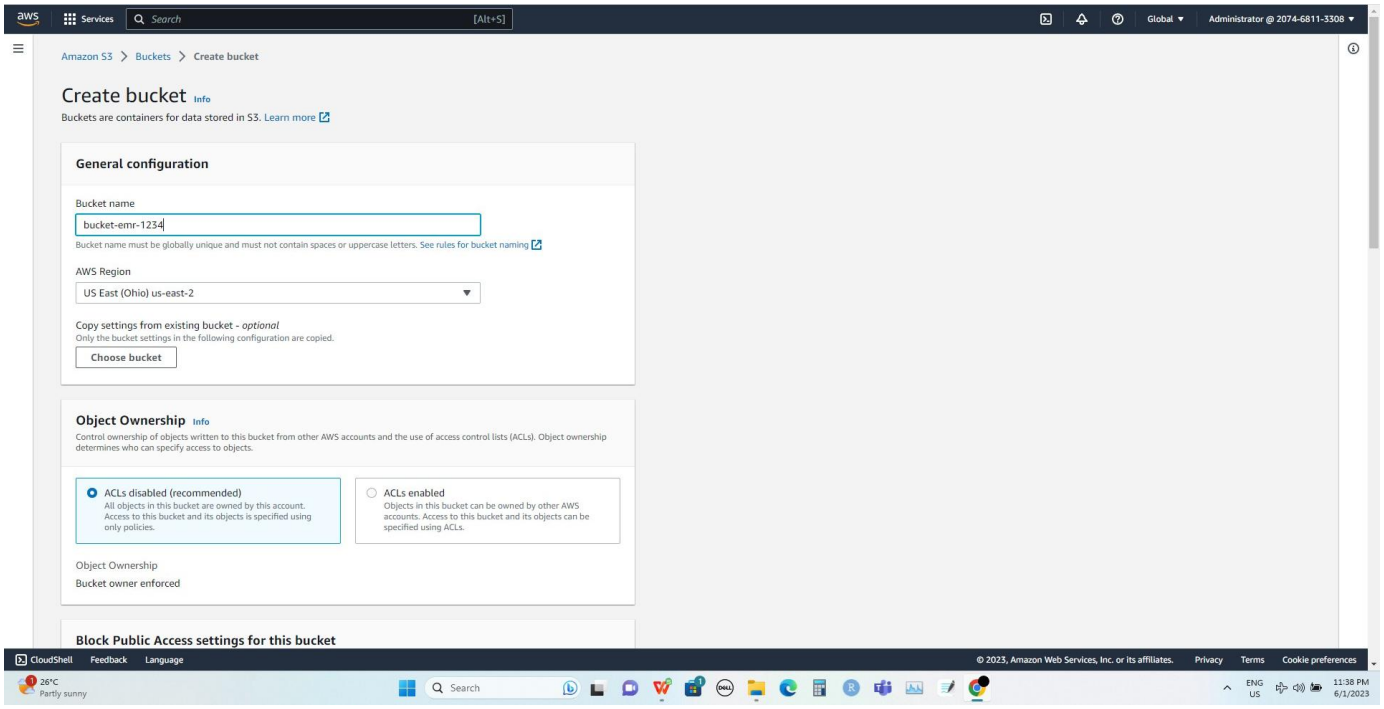


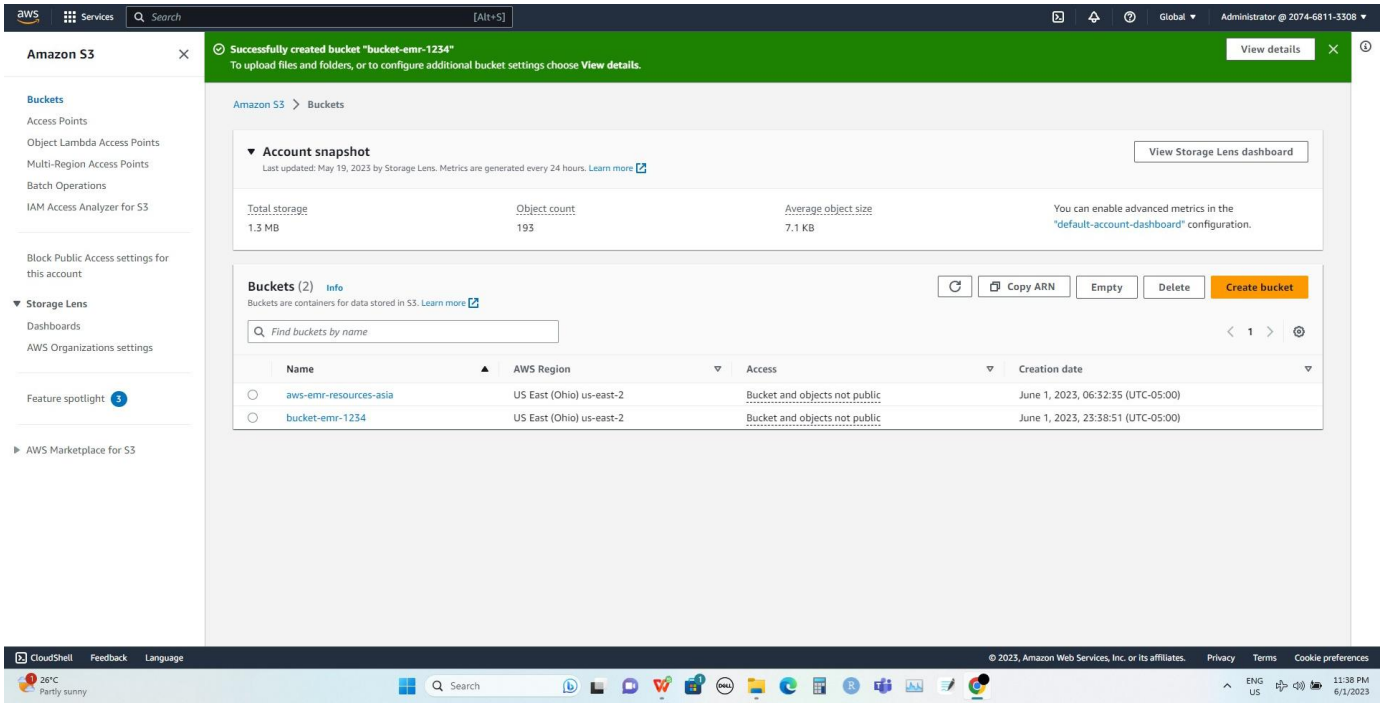
II) Configure Studio Service role

Service role : EMR_Notebooks_DeafultRoles

III) Create a S3 bucket

Create S3 bucket named “bucket-emr-123”

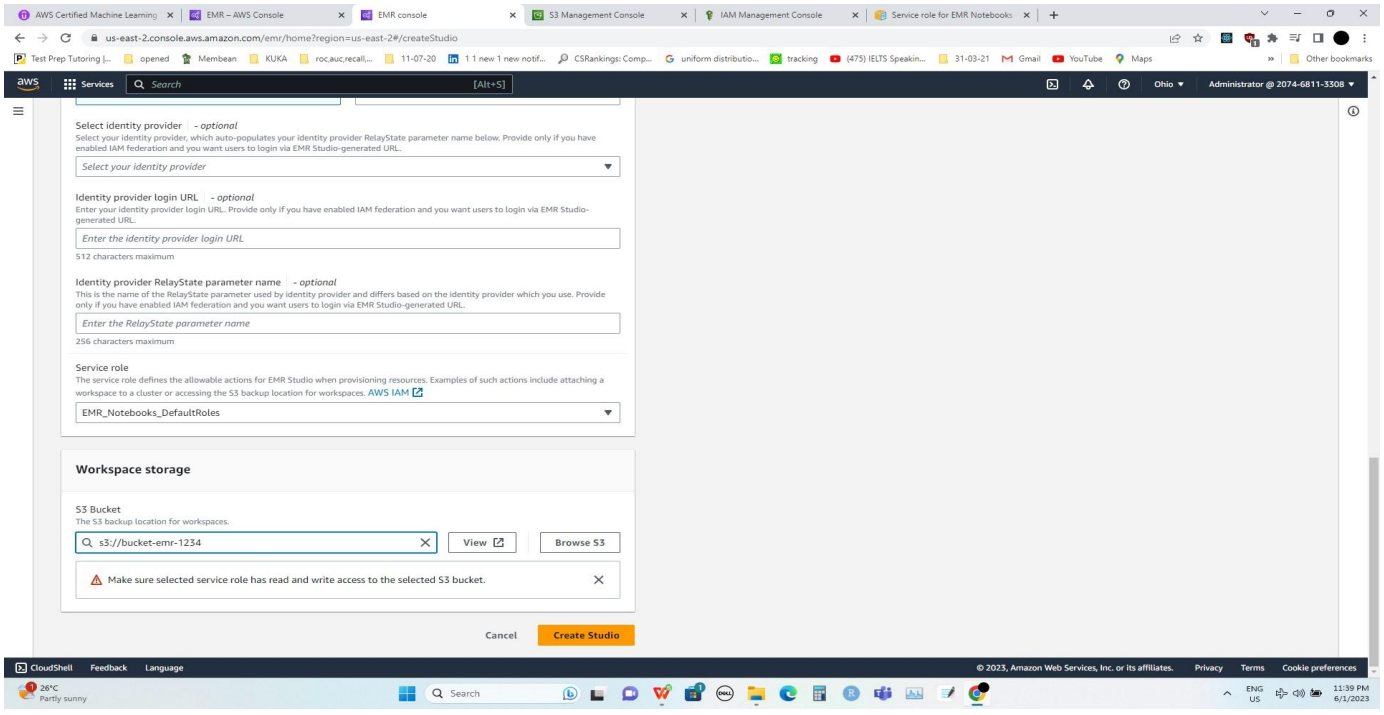




IV) Continue configuring Studio Service Role

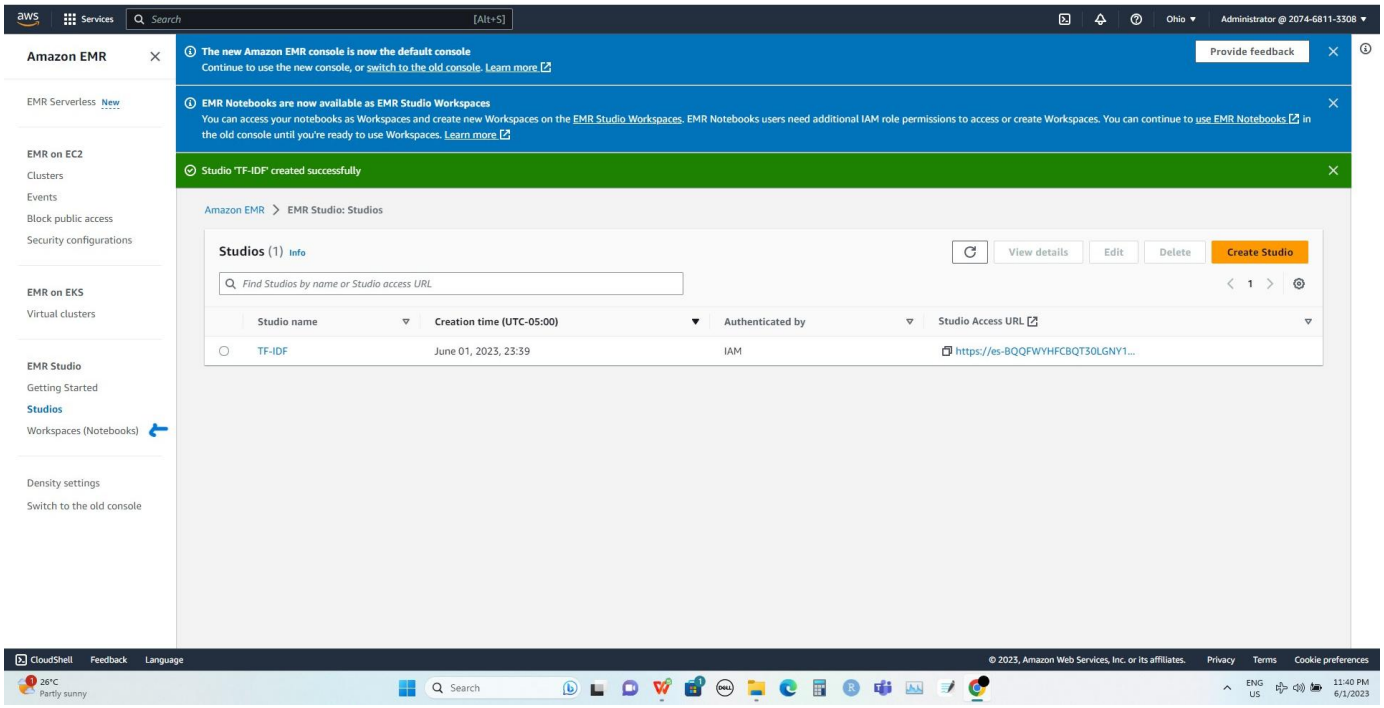
Select S3 bucket create in last step : s3://bucket-emr-1234

Click on Create Studio

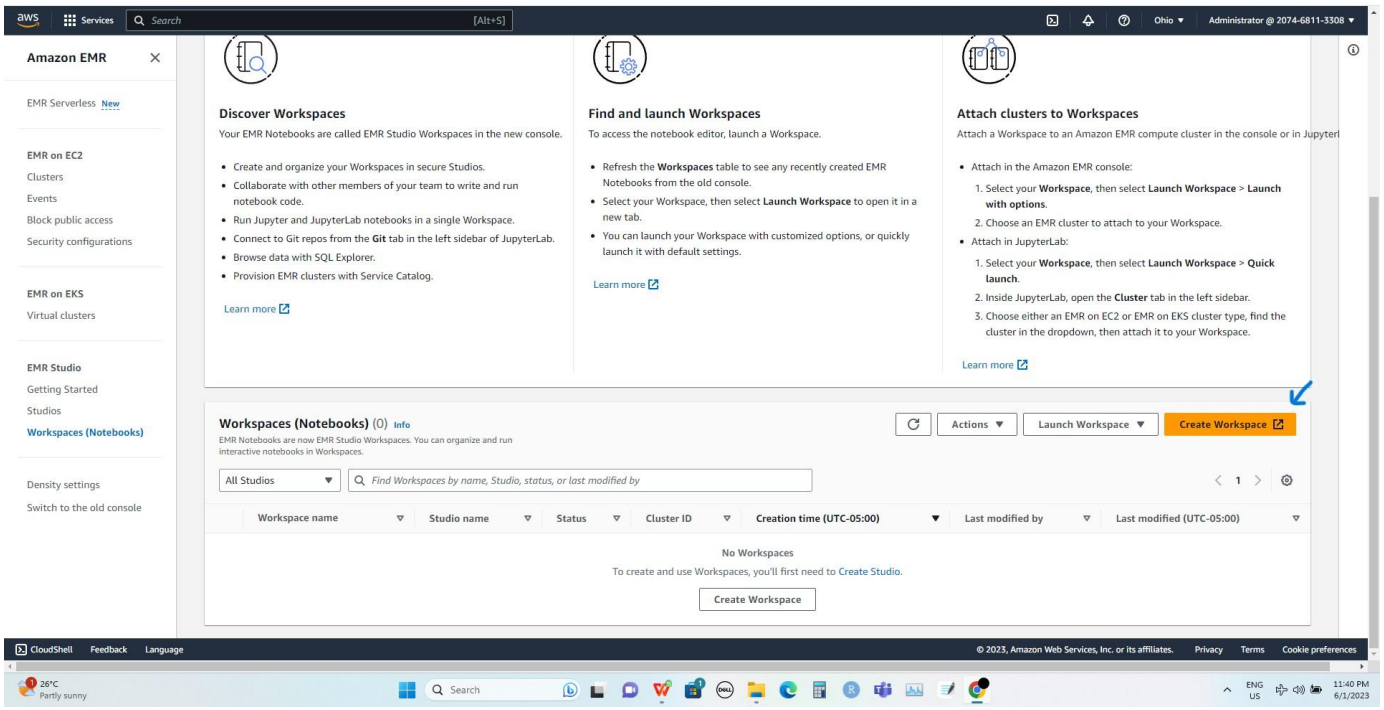


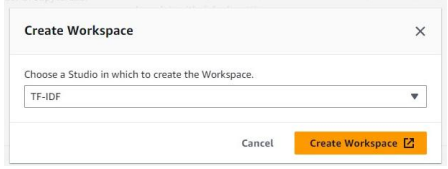
3) Create Amazon EMR Workspace

On Amazon EMR console click on “Workspaces(Notebooks)”



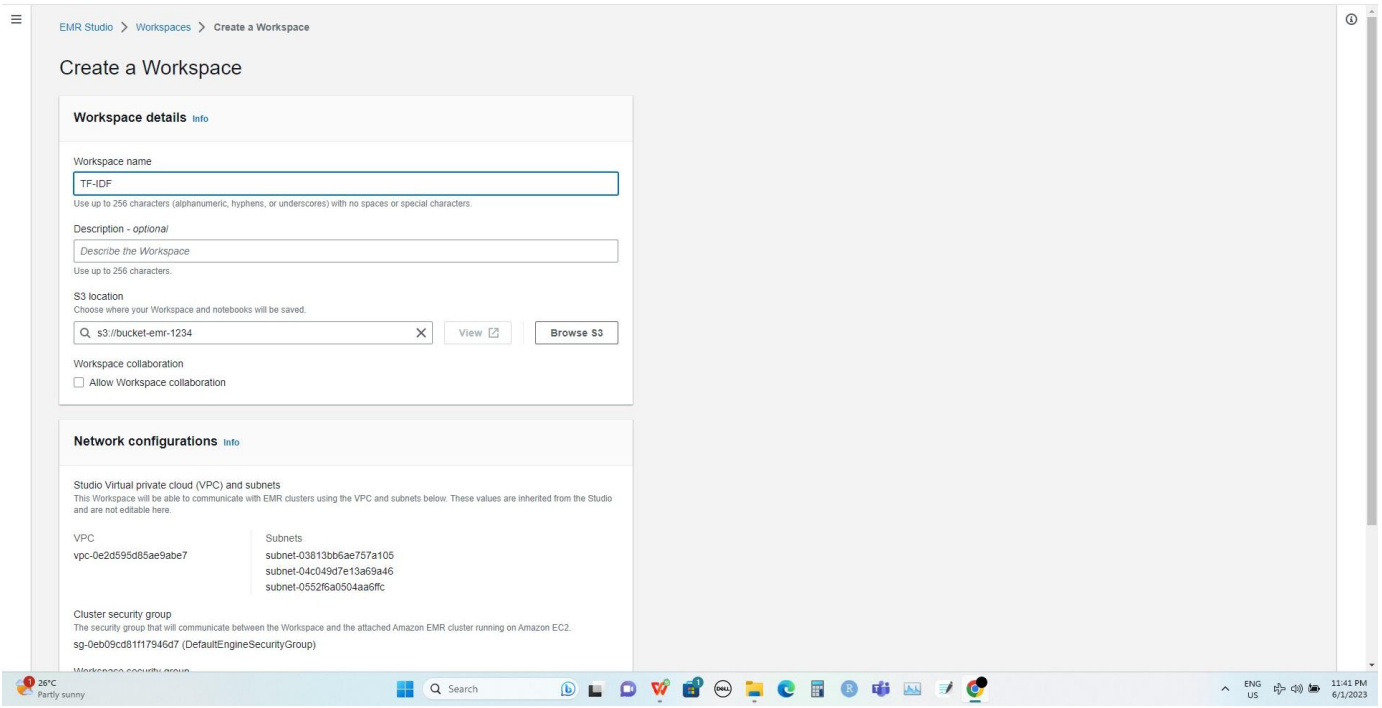
Click on “Create Workspace”
Basically, we create a workspace and then create notebooks inside it.
Choose a Studio in which to create the Workspace : TF-IDF



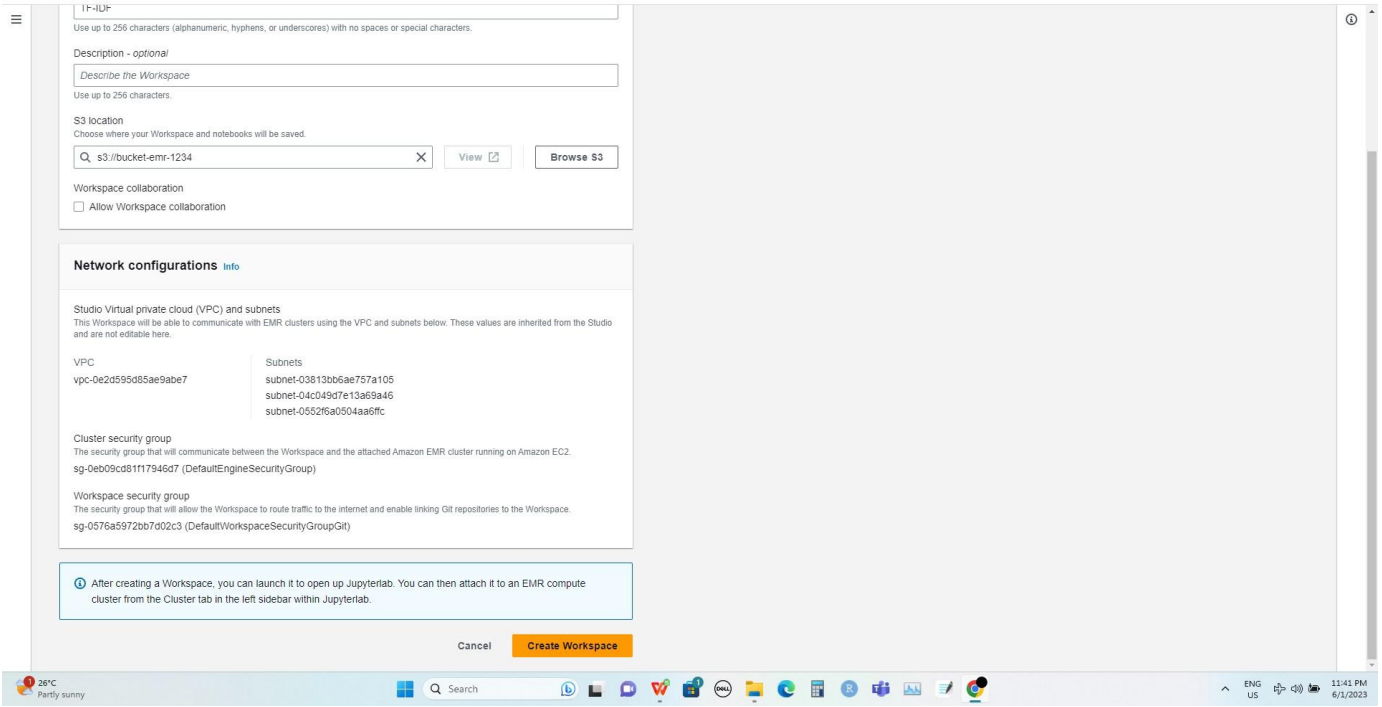


Workspace details:

Workspace name : TF-IDF
S3 location (place where workspace and notebooks will be saved) : s3://bucket-emr-1234



Click on Create Workspace

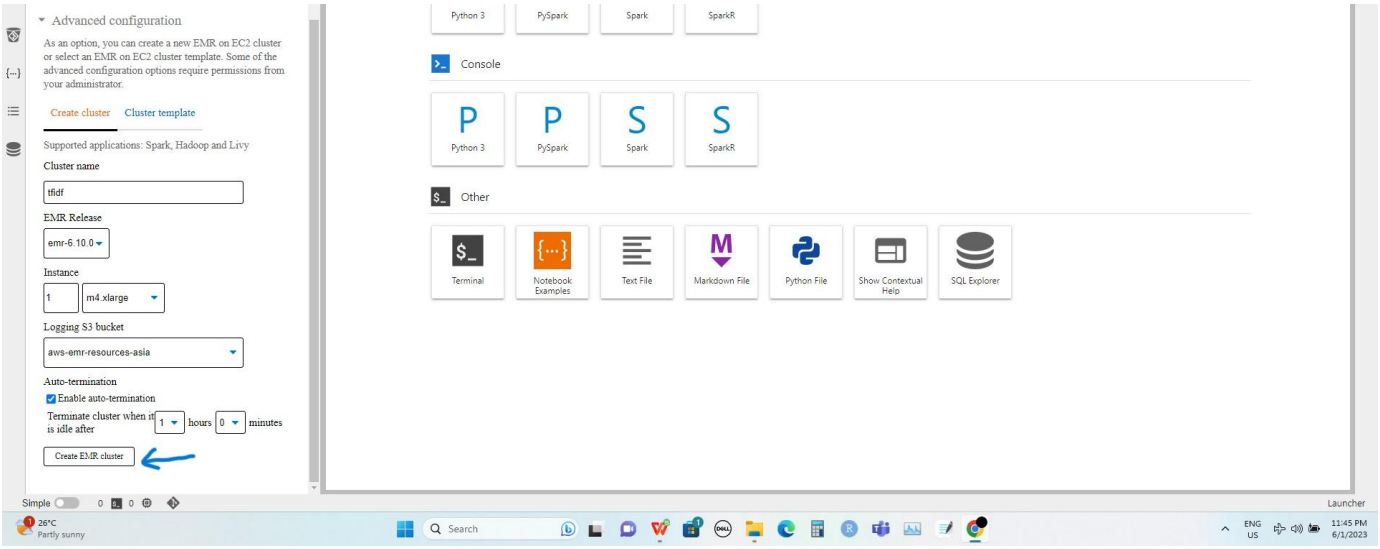


4) Cluster creation

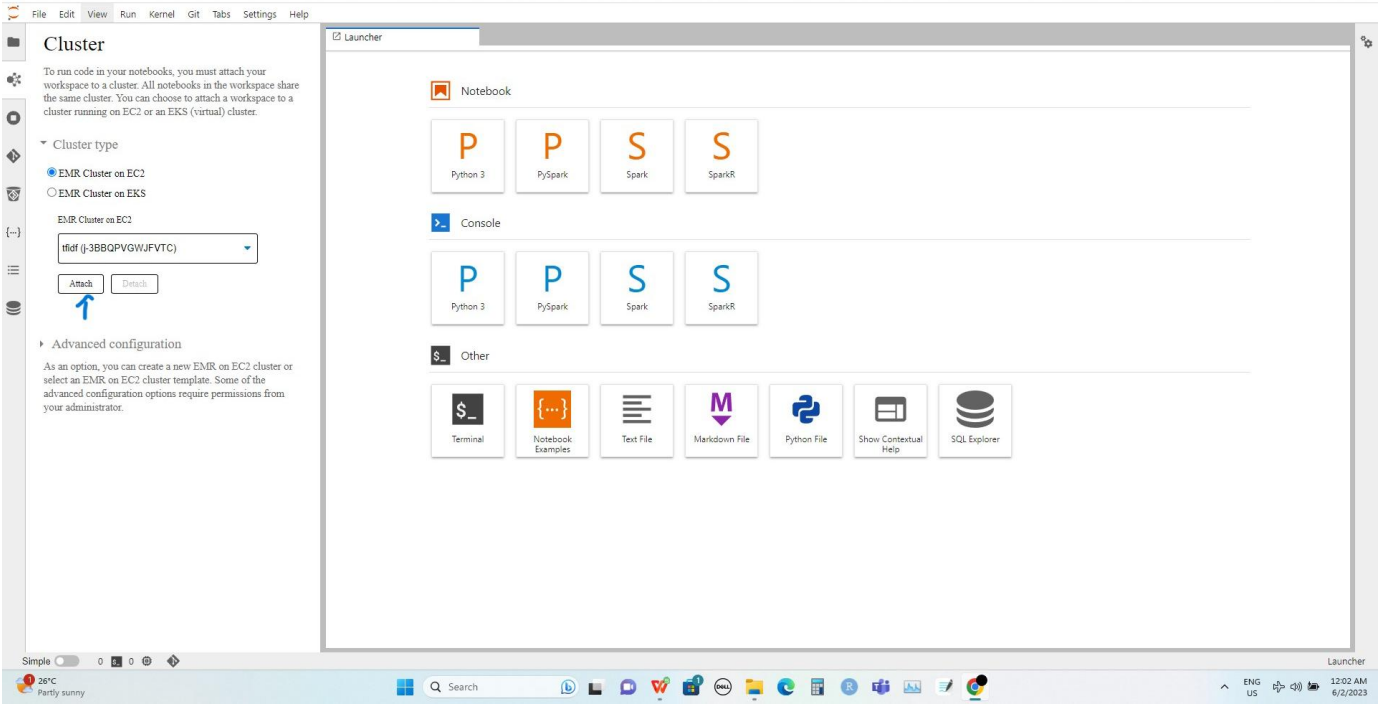
To run code in notebooks, we must first attach workspace to cluster. All notebooks in the workspace share the same cluster. We can choose to attach a workspace to a cluster on EC2 or an EKS (virtual) cluster.

In the first there will be no EC2 cluster. So under Advance Configuration drop down we should create one.

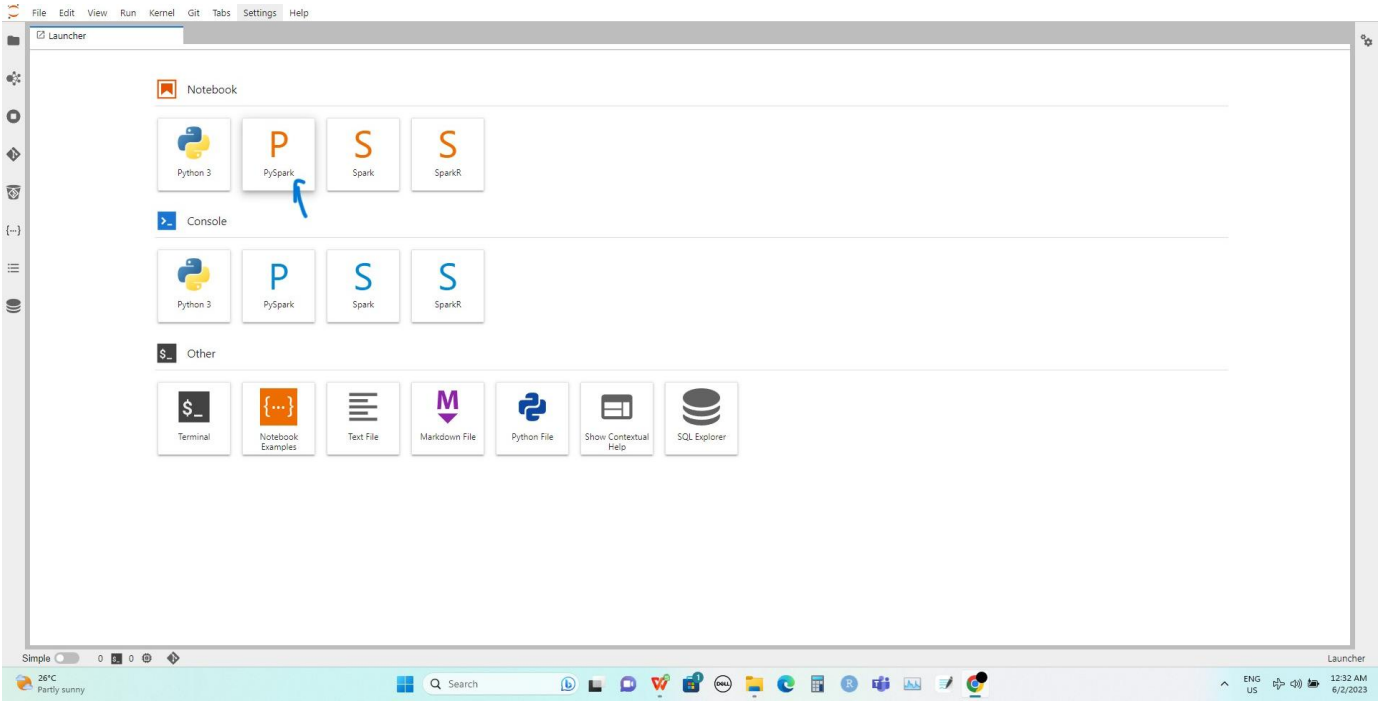
Cluster name : tfidf
EMR release : emr- 6.10.0
Instance : 1, m5.xlarge
Logging S3 bucket : aws-emr-resources-asia
Auto-termination : Enable auto-termination (Terminate clsuter when it idle for 1 hours)
Click on Create EMR cluster



5) Attach cluster to workspace and launch PySpark Notebook



Choose PySpark as launcher:

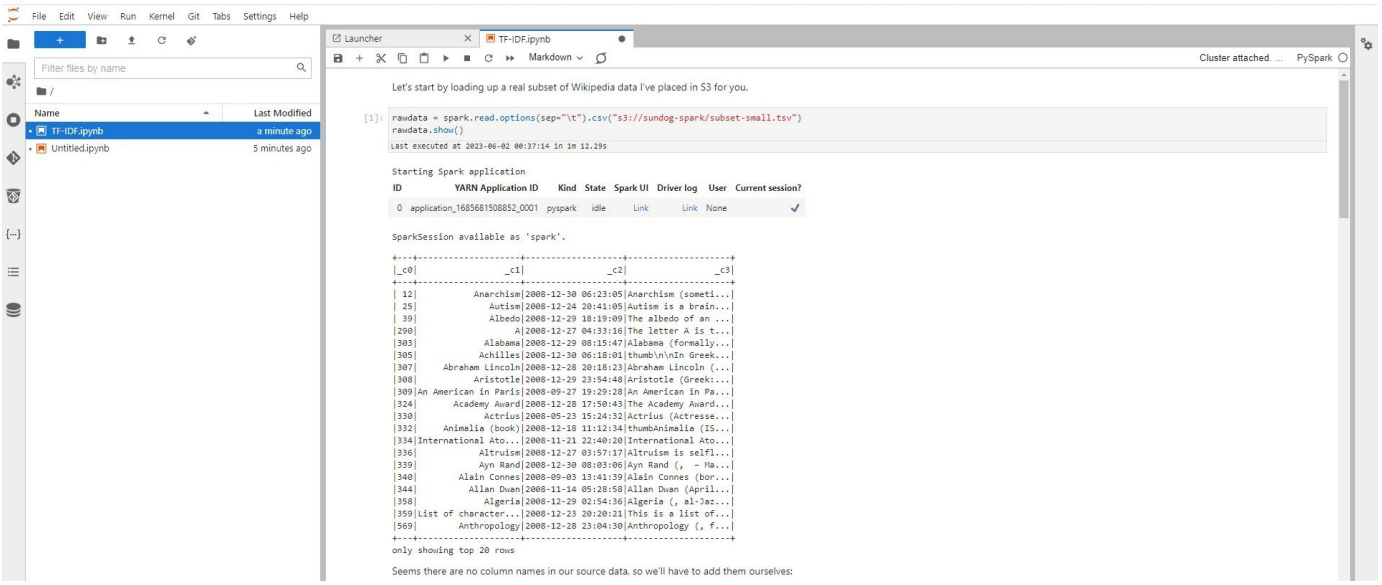


6) Code TF-IDF PySpark code

Code TF-IDF in TF-IDF.ipynb:

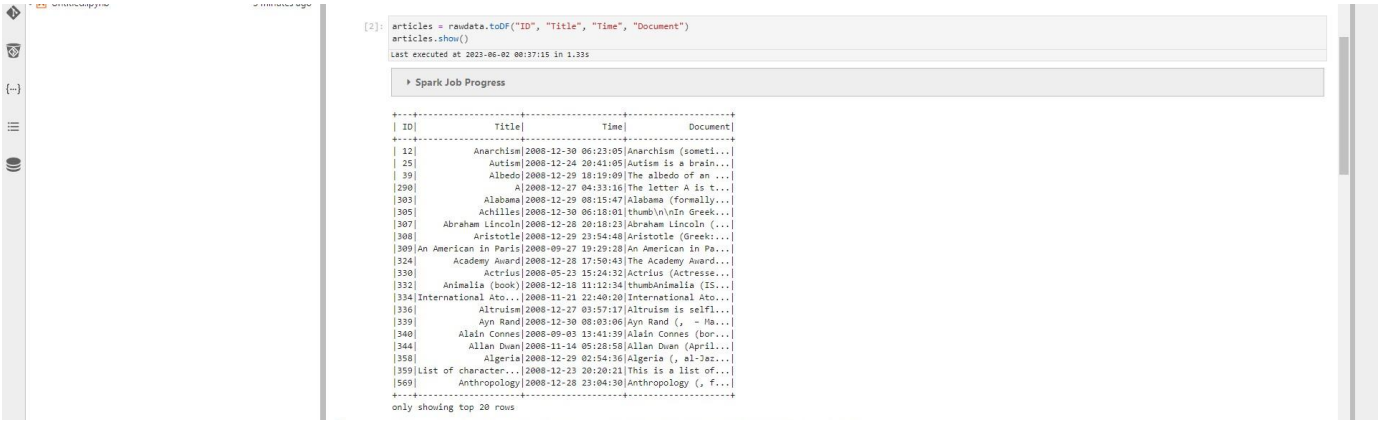
```
rawdata = spark.read.options(sep="\t").csv("s3://sundog-spark/subset-small.tsv")
rawdata.show()
```

Explanation: Loading a real subset of Wikipedia data from public S3 bucket and print the dataset.



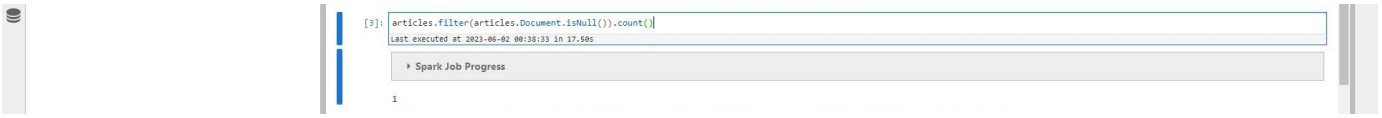
articles = rawdata.toDF("ID", "Title", "Time", "Document")
articles.show()

Explanation: Seems there are no column names in our source data, so we will add them.



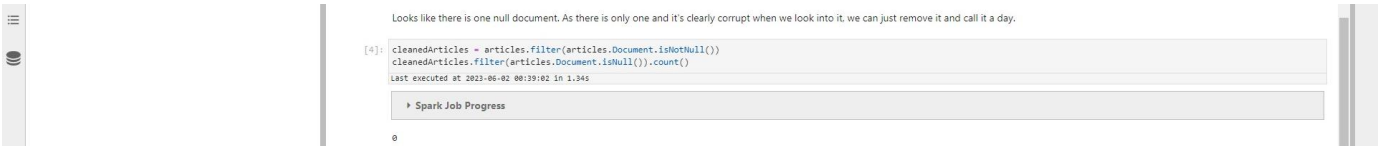
articles.filter(articles.Document.isNull()).count()

Explanation: We should "clean" our data because TF/IDF can't handle null documents, so we should check for Null.



cleanedArticles = articles.filter(articles.Document.isNotNull())
cleanedArticles.filter(articles.Document.isNull()).count()

Explanation: There is one null document.So we will remove it. And then count the number of Null records.



from pyspark.ml.feature import HashingTF, IDF, Tokenizer

tokenizer= Tokenizer(inputCol="Document", outputCol="words")

Explanation:

TF/IDF wants numbers, not words. So now we need to pre-process our data before we can run any fun algorithms on it. We'll first tokenize the articles to split them up into words, and store them in a sparse vector that is now a numeric representation of the words in each article.

"Tokenizer” is the class being instantiated.
"Tokenizer” class from Apache Spark’s MLlib library. It is used for tokenizing text documents into individual words.

inputCol="Document" specifies the name of the input column in the DataFrame that contains the text documents to be tokenized. In this case, the input column is expected to be named "Document

"outputCol =words" specifies the name of the output column in the DataFrame where the tokenized words will be stored. In this case, the output column will be named "words"

wordsData = tokenizer.transform(cleanedArticles)

Explanation:

Once we have defined ‘Tokenizer’ instance you can apply it to a DaraFrame using the ‘transform()’ method.

This will create a new DataFrame called ‘tokenizedData’ that contains the original columns from ‘dataframe’, as well as an additional column called ‘words’ that contain the tokenize words.

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures")
featurizedData = hashingTF.transform(wordsData)
featurizedData.show()

Explanation:

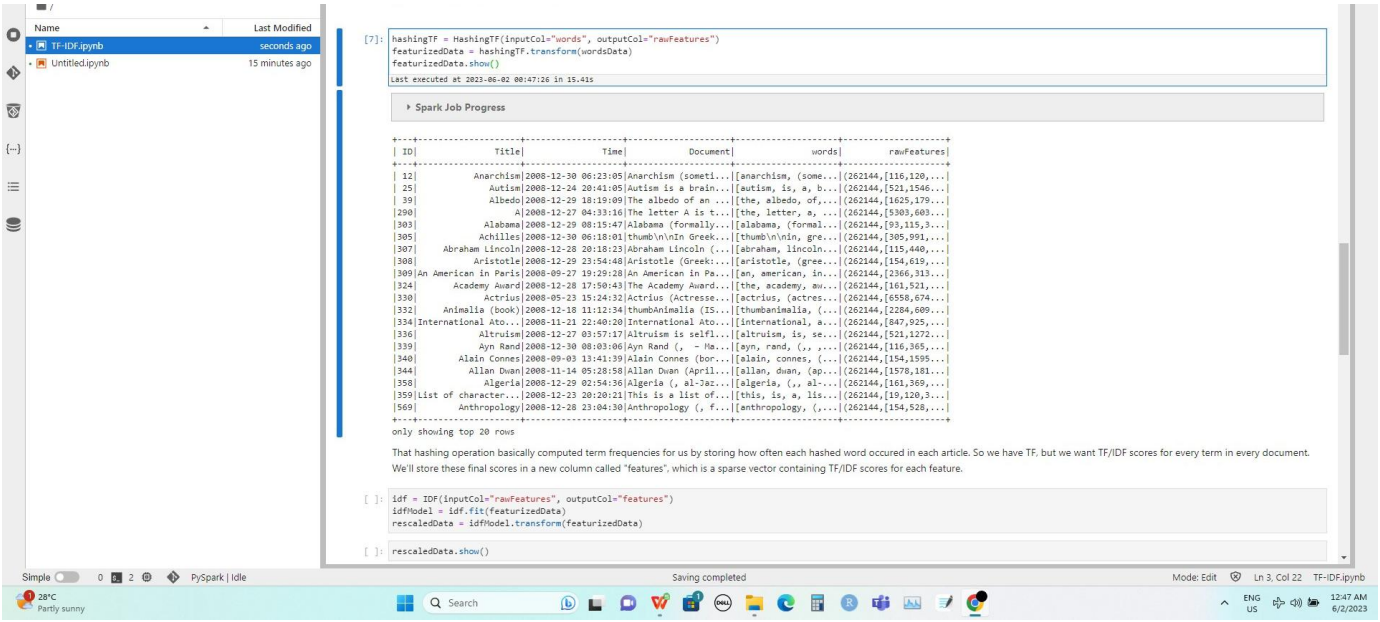
HashingTF is a class from Apache Spark’s MLIB library, it is a feature extraction method that converts a collection of text documents into a numerical feature vector representation.

"HashingTF" is the class being instantiated.

inputCol="words" specifies the name of the input column in the DataFrame that contains the tokenized words. In our case, it seems that the previous step produced a column named "words" containing the tokenized words.

"outputCol="rawFeatures" specifies the name of the output column in the DataFrame where the feature vectors will be stored. In this case, the output column will be named "rawFeatures.

Once we have defined the "Hashing TF instance, we can apply it to a DataFrame using the transform() method, similar to how it was done with the ‘Tokenizer’. The input DataFrame ‘wordsData’ will be transformed into a new DataFrame ‘featurizedData’, which will contain the original columns along with an additional column called "rawFeatures” that contains the feature vectors generated by the "HashingTF transformation. "HashingTF method uses the hashing trick to map tokens to numerical indices. The number of features in the output vector is determined by the ‘numFeatures’ parameter, which has a default value of 2^18 = 262,144. We can adjust this parameter when instantiating the "HashingTF" object if needed.



idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
rescaledData.show()

Explanation:

IDF is the class being instantiated

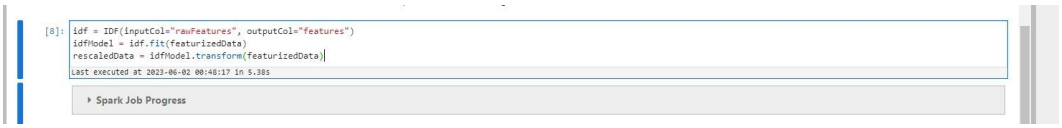
"inputCol="rawFeatures" specifies the name of the input column in the DataFrame that contains the raw feature vectors. In your case, in previous step we produced a column named "rawFeatures" containing the feature vectors generated by the "HashingTF transformation.

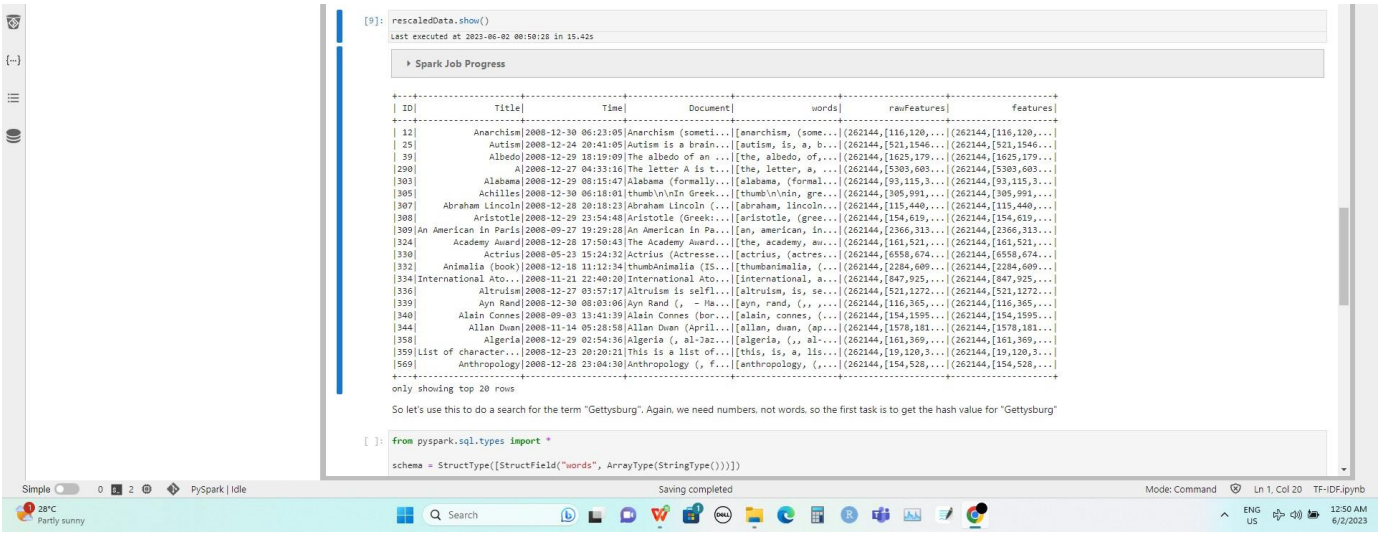
outputCol="features" specifies the name of the output column in the DataFrame where the IDF-weighted feature vectors will be stored. In this case, the output column will be named "features"

Once we have defined the IDF instance, we can apply it to a DataFrame using the fit() method to compute the IDF weights and the transform() method to apply the IDF transformation. The input DataFrame 'featurizedData' will be transformed into a new
DataFrame ‘rescaledData’, which will contain the original columns along with an additional column called "features" that contains the IDF-weighted feature vectors.

The fit() method is used to compute the IDF weights based on the input DataFrame. It returns an "IDFModel" object that can be used to transform other DataFrames with the same schema

It's important to note that the IDF weights are typically computed based on a collection of documents to provide a measure of the informativeness of each term. The IDF class in Spark MLlib assumes that each row in the Input DataFrame represents a document, and the IDF weights are computed based on the entire dataset.






```
from pyspark.sql.types import *

schema = StructType([StructField("words", ArrayType(StringType()))])

df = spark.createDataFrame(([[["gettysburg"]]]), schema).toDF("words")
df.show()
```

Expalanation:The above code defines schema for the DataFarme. Then, it creates a DataFrame called ‘df’ with a single row that contains the word “gettysburg” as the value in the “word” column.


```
gettysburg = hashingTF.transform(df)
gettysburg.show()
```

Expalanation:It applies ‘hashing tf’ transformation on the DataFrame ‘df’ to obtain the feature vector representation of the word “gettysburg”. The resulting DataFrame is assigned to ‘gettysburg’.


```
featureVec = gettysburg.select('rawFeatures').collect()
```

Expalanation:It selects the “rawFeatures” column from the ‘gettysburg’ DataFrame and collects all the rows into Python list called ‘featureVec’.

```
featureVec = gettysburg.select('rawFeatures').collect()
print(featureVec)
```

Last executed at 2023-06-02 00:57:26 in 793ms

Spark Job Progress

gettysburgID = int(featureVec[0].rawFeatures.indices[0])

Expalanation: It retrieves the first element of 'featureVec', which corresponds to the feature vector for the word "gettysburg". It then extracts the index of the first non-zero element in the feature vector using 'indices[0]' and converts it to an integer.

```
gettysburgID = int(featureVec[0].rawFeatures.indices[0])
print(gettysburgID)
```

Last executed at 2023-06-02 00:57:46 in 97ms

```
116775
```

We have the number that represents "Gettysburg". Now we can add another column named "score" that just extracts the TF/IDF value for Gettysburg for each document.

```
from pyspark.sql.types import FloatType
from pyspark.sql.functions import udf

termExtractor = udf(lambda x: float(x[gettysburgID]), FloatType())
gettysburgDF = rescaledData.withColumn('score', termExtractor(rescaledData.features))

gettysburgDF.show()
```

```
from pyspark.sql.types import FloatType
from pyspark.sql.functions import udf

termExtractor = udf(lambda x: float(x[gettysburgID]), FloatType())
gettysburgDF = rescaledData.withColumn('score', termExtractor(rescaledData.features))

gettysburgDF.show()
```

Last executed at 2023-06-02 00:59:26 in 21.39s

Spark Job Progress

```
+---+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------+
| ID|               Title|                Time|            Document|               words|         rawFeatures|            features|   score|
+---+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------+
| 12|           Anarchism|2008-12-30 06:23:05|Anarchism (someti...|[anarchism, (some...|(262144,[116,120,...|(262144,[116,120,...|     0.0|
| 25|              Autism|2008-12-24 20:41:05|Autism is a brain...|[autism, is, a, b...|(262144,[521,1546...|(262144,[521,1546...|     0.0|
| 39|              Albedo|2008-12-29 18:19:09|The albedo of an ...|[the, albedo, of,...|(262144,[1625,179...|(262144,[1625,179...|     0.0|
|290|                   A|2008-12-27 04:33:16|The letter A is t...|[the, letter, a, ...|(262144,[5303,603...|(262144,[5303,603...|     0.0|
|303|             Alabama|2008-12-29 08:15:47|Alabama (formally...|[alabama, (formal...|(262144,[93,115,3...|(262144,[93,115,3...|     0.0|
|305|            Achilles|2008-12-30 06:18:01|thumb\n\nIn Greek...|[thumb\n\nin, gre...|(262144,[305,991,...|(262144,[305,991,...|     0.0|
|307|     Abraham Lincoln|2008-12-28 20:18:23|Abraham Lincoln (...|[abraham, lincoln...|(262144,[115,440,...|(262144,[115,440,...|30.695974|
|308|           Aristotle|2008-12-29 23:54:48|Aristotle (Greek:...|[aristotle, (gree...|(262144,[154,619,...|(262144,[154,619,...|     0.0|
|309|An American in Paris|2008-09-27 19:29:28|An American in Pa...|[an, american, in...|(262144,[2366,313...|(262144,[2366,313...|     0.0|
|324|       Academy Award|2008-12-28 17:50:43|The Academy Award...|[the, academy, aw...|(262144,[161,521,...|(262144,[161,521,...|     0.0|
|330|             Actrius|2008-05-23 15:24:32|Actrius (Actresse...|[actrius, (actres...|(262144,[6558,674...|(262144,[6558,674...|     0.0|
|332|     Animalia (book)|2008-12-18 11:12:34|thumbAnimalia (IS...|[thumbanimalia, (...|(262144,[2284,609...|(262144,[2284,609...|     0.0|
|334|International Ato...|2008-11-21 22:40:20|International Ato...|[international, a...|(262144,[847,925,...|(262144,[847,925,...|     0.0|
|336|            Altruism|2008-12-27 03:57:17|Altruism is selfl...|[altruism, is, se...|(262144,[521,1272...|(262144,[521,1272...|     0.0|
|339|            Ayn Rand|2008-12-30 08:03:06|Ayn Rand (,  - Ma...|[ayn, rand, (,, ,...|(262144,[116,365,...|(262144,[116,365,...|     0.0|
|340|       Alain Connes|2008-09-03 13:41:39|Alain Connes (bor...|[alain, connes, (...|(262144,[154,1595...|(262144,[154,1595...|     0.0|
|344|          Allan Dwan|2008-11-14 05:28:58|Allan Dwan (April...|[allan, dwan, (ap...|(262144,[1578,181...|(262144,[1578,181...|     0.0|
|358|             Algeria|2008-12-29 02:54:36|Algeria (, al-Jaz...|[algeria, (,, al-...|(262144,[161,369,...|(262144,[161,369,...|     0.0|
|359|List of character...|2008-12-23 20:20:21|This is a list of...|[this, is, a, lis...|(262144,[19,120,3...|(262144,[19,120,3...|     0.0|
|569|        Anthropology|2008-12-28 23:04:30|Anthropology (, f...|[anthropology, (,...|(262144,[154,528,...|(262144,[154,528,...|     0.0|
+---+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------+
only showing top 20 rows
```

Now have to sort our articles by score, and we'll have the most relevant articles for 'Gettysburg'.

```
sortedResults = gettysburgDF.filter("score > 0").orderBy('score', ascending=False).select('ID', 'Title', 'Document', 'score')
sortedResults.show(truncate=100)
```

```
sortedResults = gettysburgDF.filter("score > 0").orderBy('score', ascending=False).select('ID', 'Title', 'Document', 'score')
sortedResults.show(truncate=100)
```

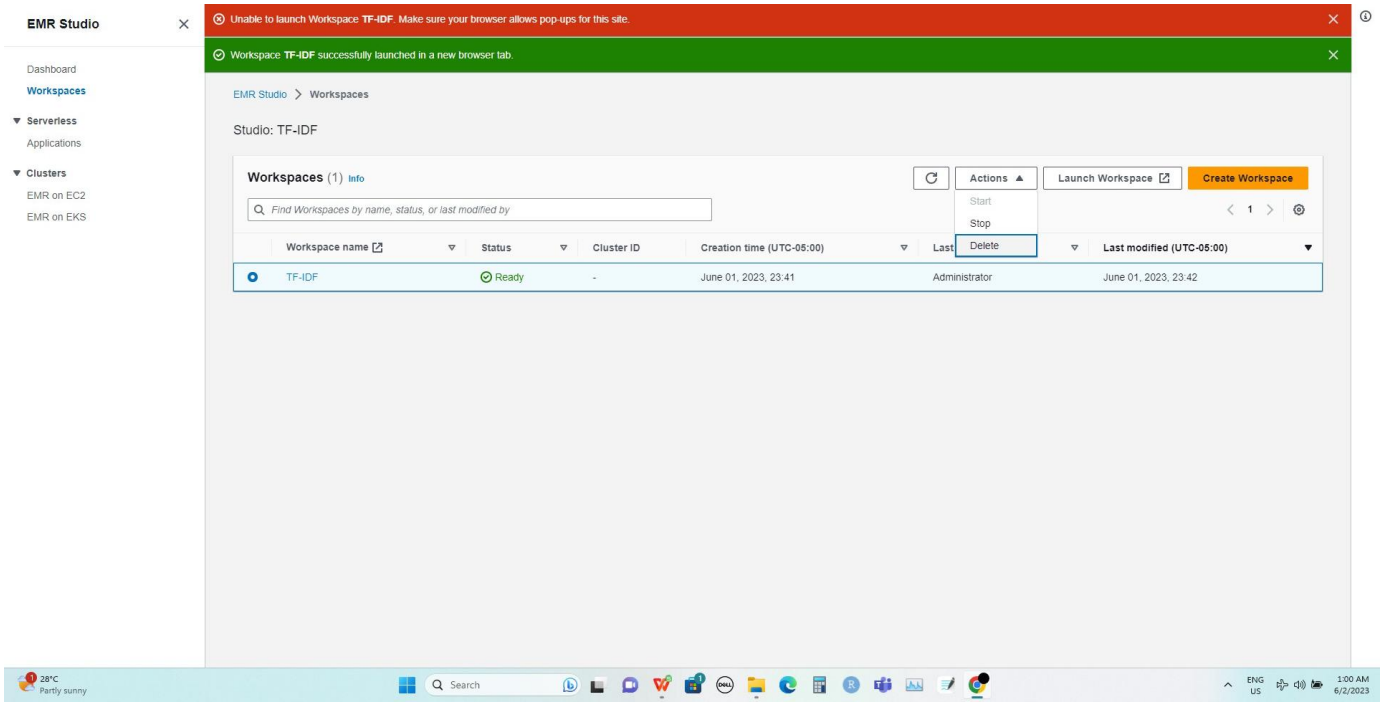
Last executed at 2023-06-02 00:59:46 in 5.36s

Spark Job Progress

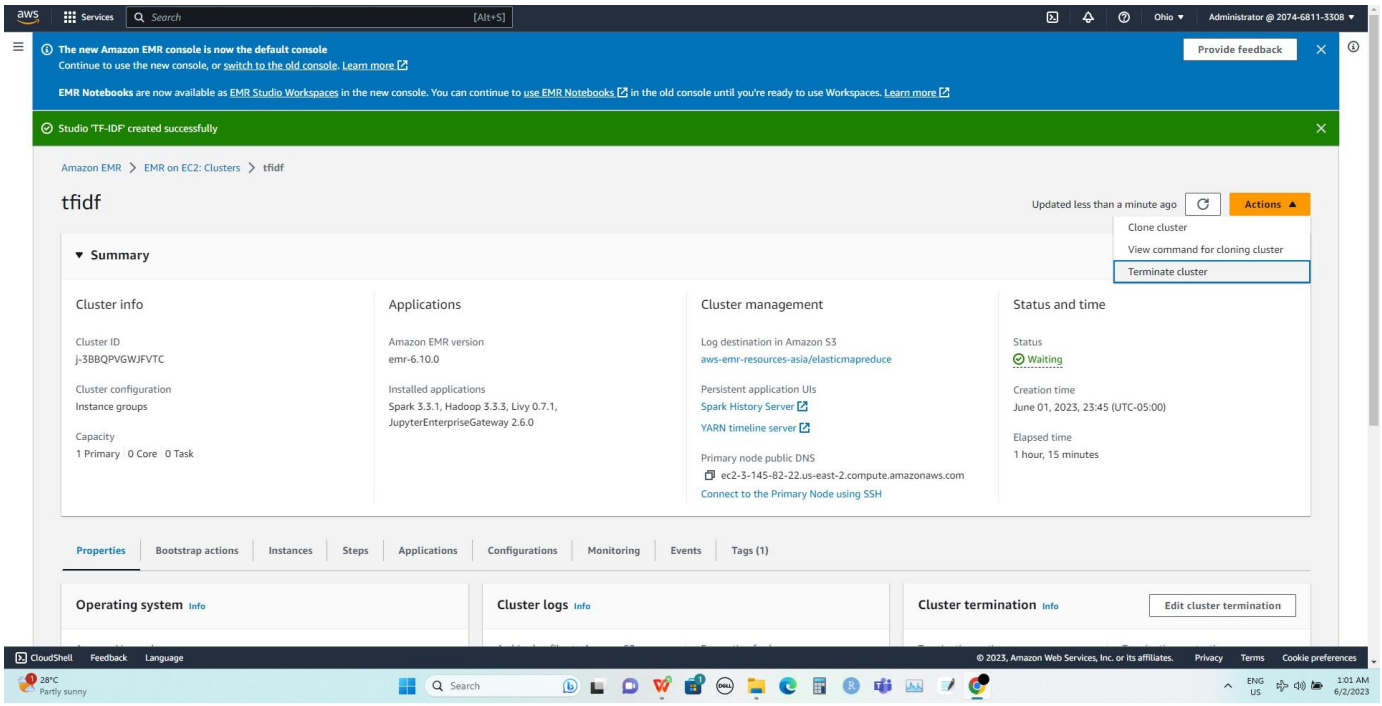
```
+----+----------------+----------------------------------------------------------------------------------------------------+--------+
|  ID|           Title|                                                                                            Document|   score|
+----+----------------+----------------------------------------------------------------------------------------------------+--------+
| 307|  Abraham Lincoln|Abraham Lincoln (February 12, 1809 - April 15, 1865) was the sixteenth President of the United St...|30.695974|
|1135|  Abner Doubleday|Abner Doubleday (June 26, 1819 - January 26, 1893) was a career United States Army officer and Un...|25.579979|
| 863|American Civil War|The American Civil War (1861-1865), also known as the War Between the States and several other na...|15.347987|
|1998|     Austin, Texas|Austin is the capital of the U.S. state of Texas and the county seat of Travis County. Situated i...| 5.115996|
|2085|          Assyria|Assyria was a geographic region on the Upper Tigris river, named for its original capital, the an...| 5.115996|
+----+----------------+----------------------------------------------------------------------------------------------------+--------+
```

7) Deleting Resources

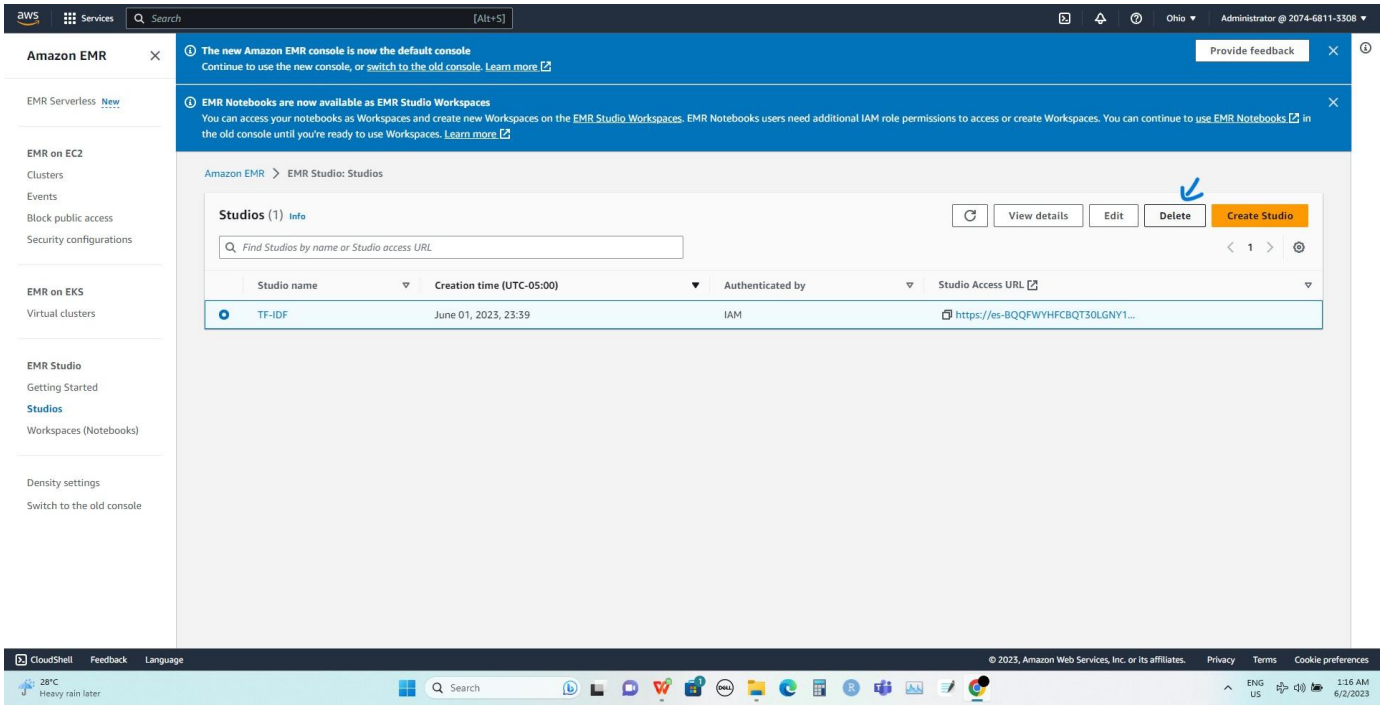
A) Delete workspace:



B) Terminate Cluster



C) Delete EMR Studio



D) Delete S3 buckets