**Steps for Creating Catalog Tables:**

The step-by-step process to define catalog tables using s3 as source.
Upload data to s3. We will be using JSON Data.
Create Crawler
Provide Name
Configure IAM Role
I download GitHub activity data so that we can upload to s3 to learn data engineering using AWS Analytics Services.We can download the files using instructions provided as part of gharchive.org.

mkdir ~/Downloads/ghactivity
cd ~/Downloads/ghactivity
wget https://data.gharchive.org/2021-01-13-{0..23}.json.gz
wget https://data.gharchive.org/2021-01-14-{0..23}.json.gz
wget https://data.gharchive.org/2021-01-15-{0..23}.json.gz

You can use s3 web console to create a bucket and then copy the data into the bucket. Based up on the bandwidth, this action will take a considerable amount of time.

Make sure to have a bucket by name itv-github
Make sure to create folder by name landing/ghactivity/

**Create Glue Catalog Table – ghactivity:**
create Glue Crawler as well as Glue Catalog Table for GitHub Activity data under itvghlandingdb.
We will give the name as GHActivity Landing Crawler.
Create a new role by name AWSGlueServiceRole-GitHub. It will create a role and attach policy to provide access to relevant s3 buckets.
Data is in s3 in this location - s3://itv-github/landing/ghactivity/
When we run crawler it will create a table with name ghactivity in itvghlandingdb.
Crawls the files and sample data from the files.
Infer schema using folder names as well as attributes from our JSON files.
Creates a table in the configured database with the schema inferred from the data and folders.
Here are the instructions to validate the table in Glue Catalog.

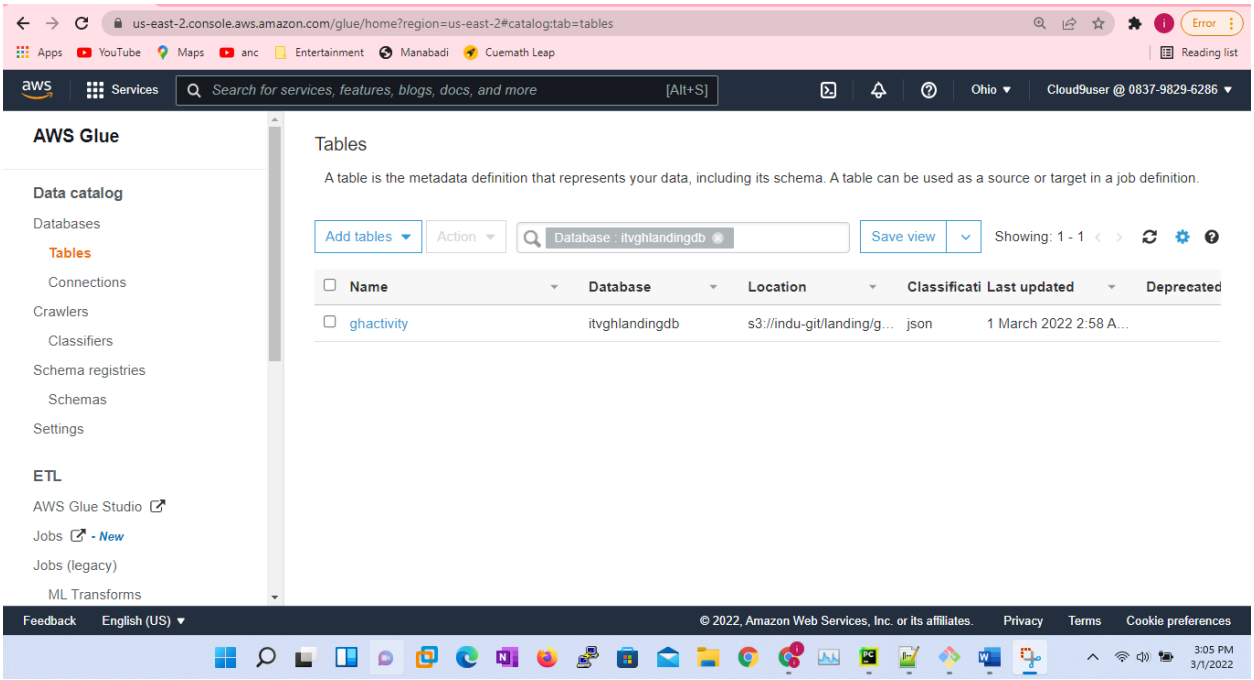
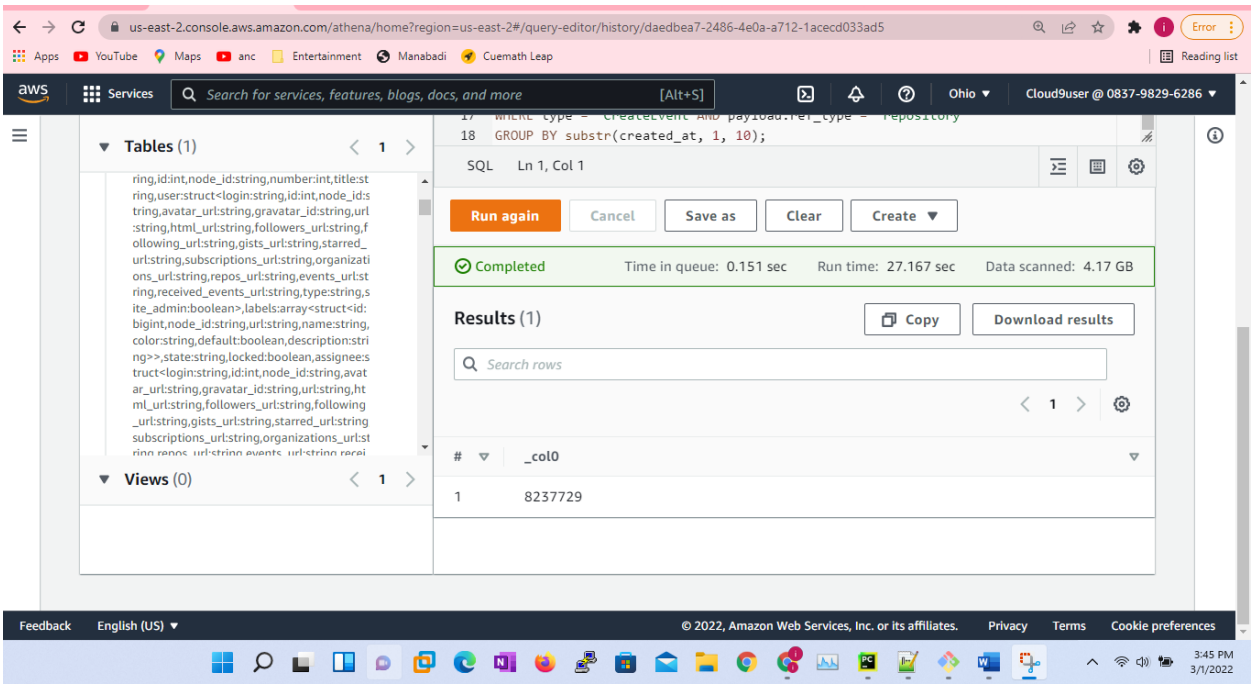


*Figure 1Created the catalog tabel*

Go to the database and check if the table is created or not.
Click on the table and check the columns as well as their data types.
Once the table is created, we can run queries using services like Athena or process data using services like Glue Jobs, EMR or even third party services like Databricks.
Upload files into the folder using AWS s3 Web Console

**Running Queries using Athena – ghactivity:**

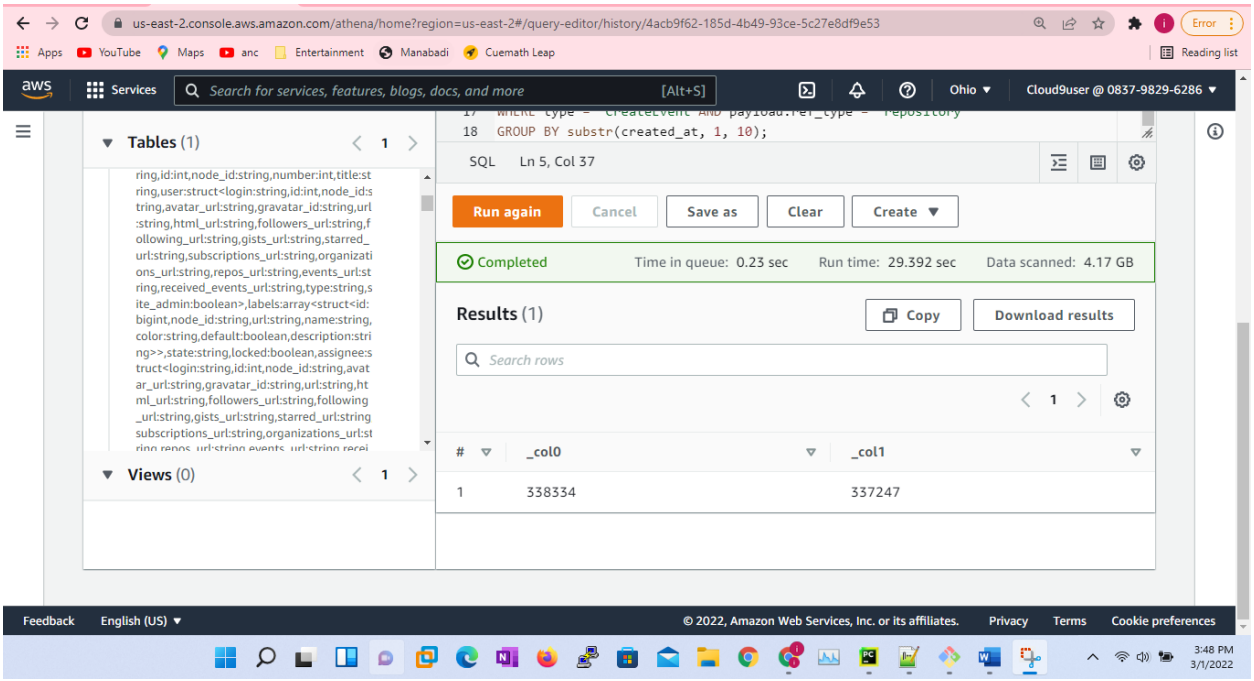
run following queries using Athena to ensure that data is copied and tables can be queried.Get the number of records from the table.
SELECT count(1) FROM ghactivity;




Get the number of new repositories added.

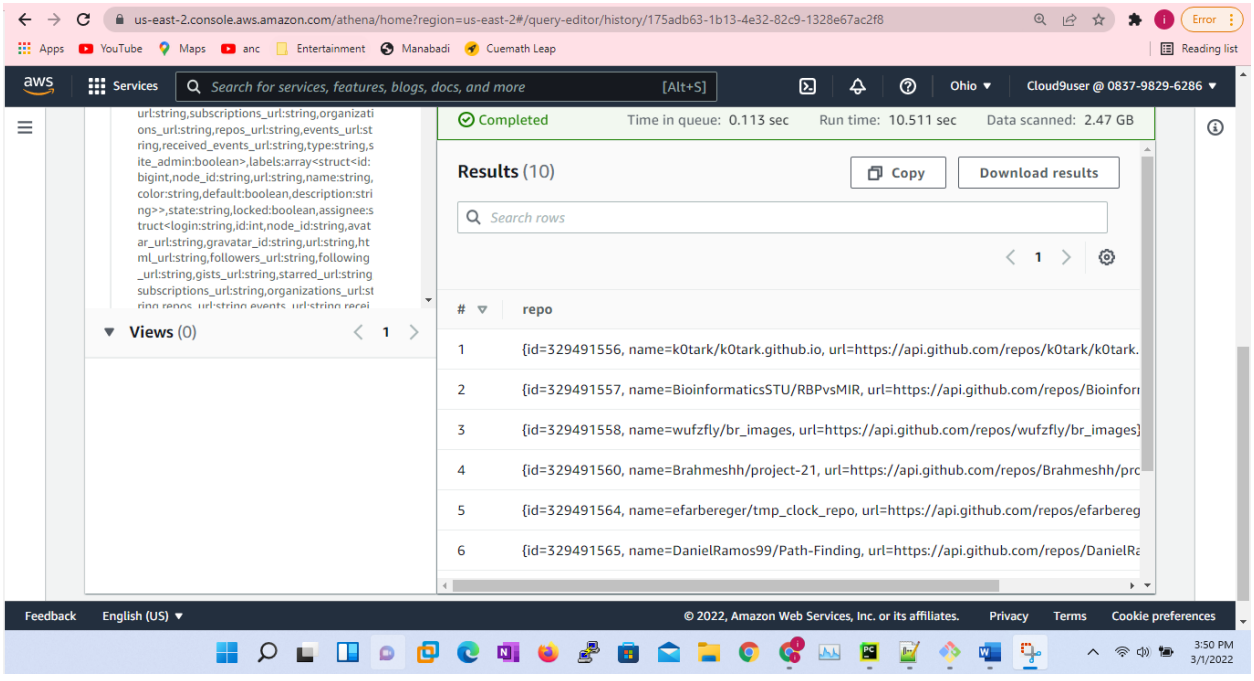
SELECT count(1), count(distinct repo.id) FROM ghactivity

WHERE type = 'CreateEvent'
AND payload.ref_type = 'repository';




Preview repo related details using repo column of type struct.

SELECT repo.* FROM ghactivity
WHERE type = 'CreateEvent'
AND payload.ref_type = 'repository'
LIMIT 10;



**Crawling Multiple Folders:**

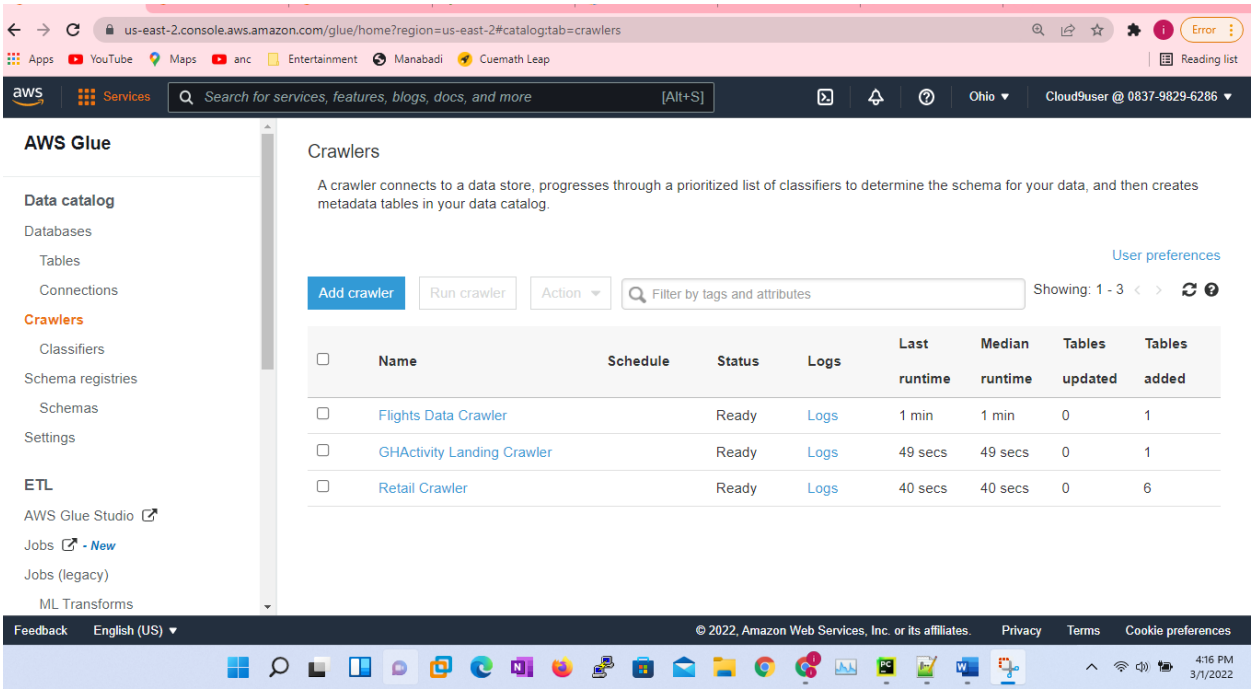
we can crawl multiple folders and create multiple tables using Glue Crawler.

We will use retail_db data using JSON format. You can get the files from our GitHub Repository.

It contains 6 folders with 1 file each.

We need to add a data source for each table as part of the crawler definition.

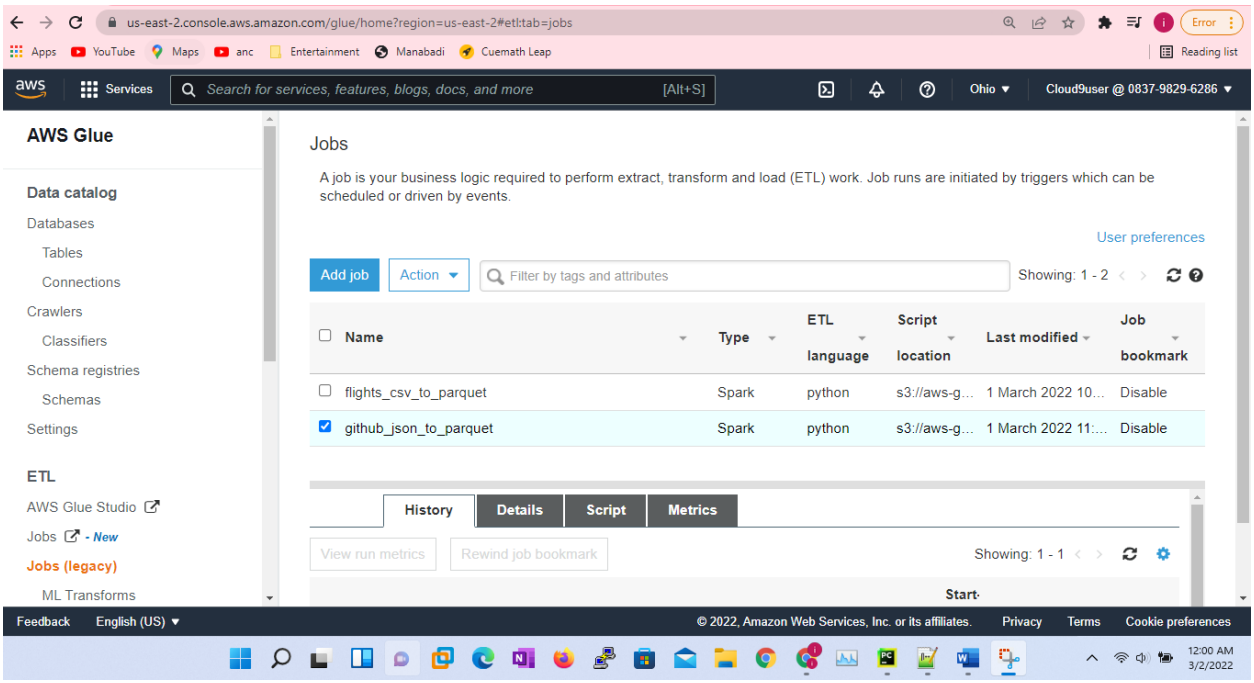
Once we create the crawler and run it, it will take care of creating 6 tables




**Generate baseline Glue Job:**

Let us generate a baseline Glue Job. We will go through all the steps that are involved in generating a baseline Glue Job.

- Give a name to the job.
- Assign appropriate role with all permissions to read and write the data.
- Configure source and target s3 locations.
- Configure locations related to logs.




Let us run the baseline Glue Job before customizing it. It will prove that all the permissions are working as expected.

- Monitor the job to track progress and troubleshoot if there are any errors.
  Once the job is completed, make sure that data is copied into the target location.
  We can crawl the target location to create a Glue Catalog table and query using Athena.

Make sure to delete the data in the target location as we would like to customize the job to partition the data.

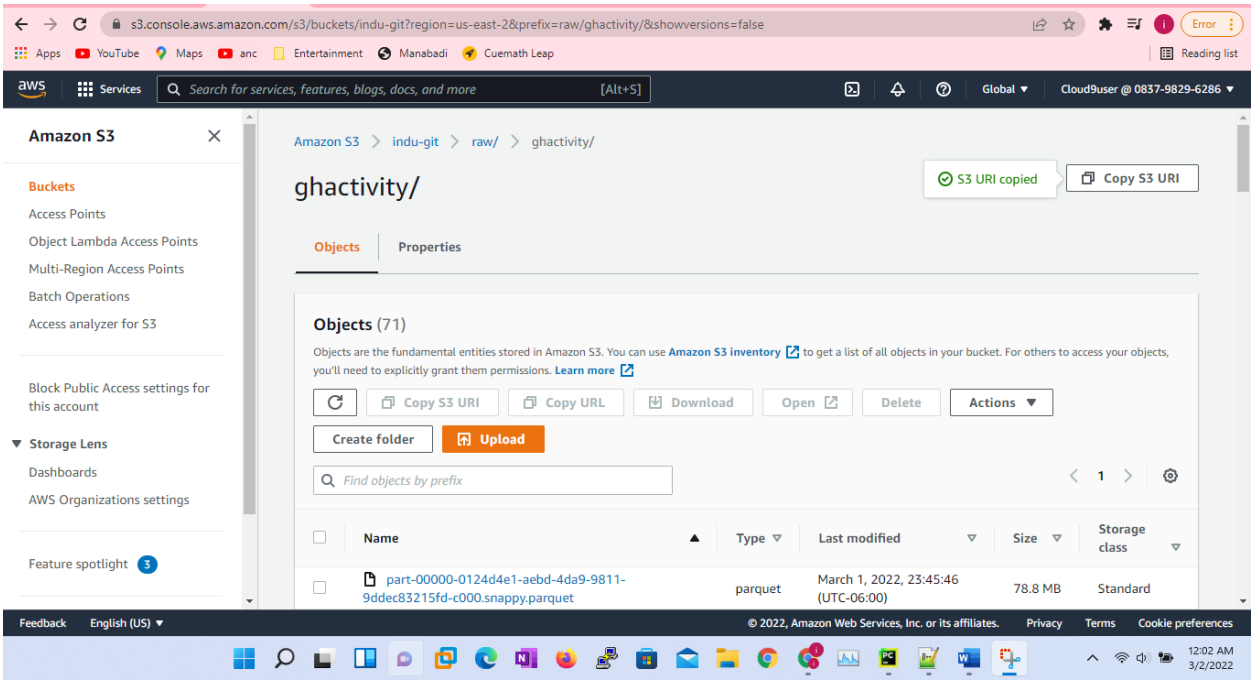
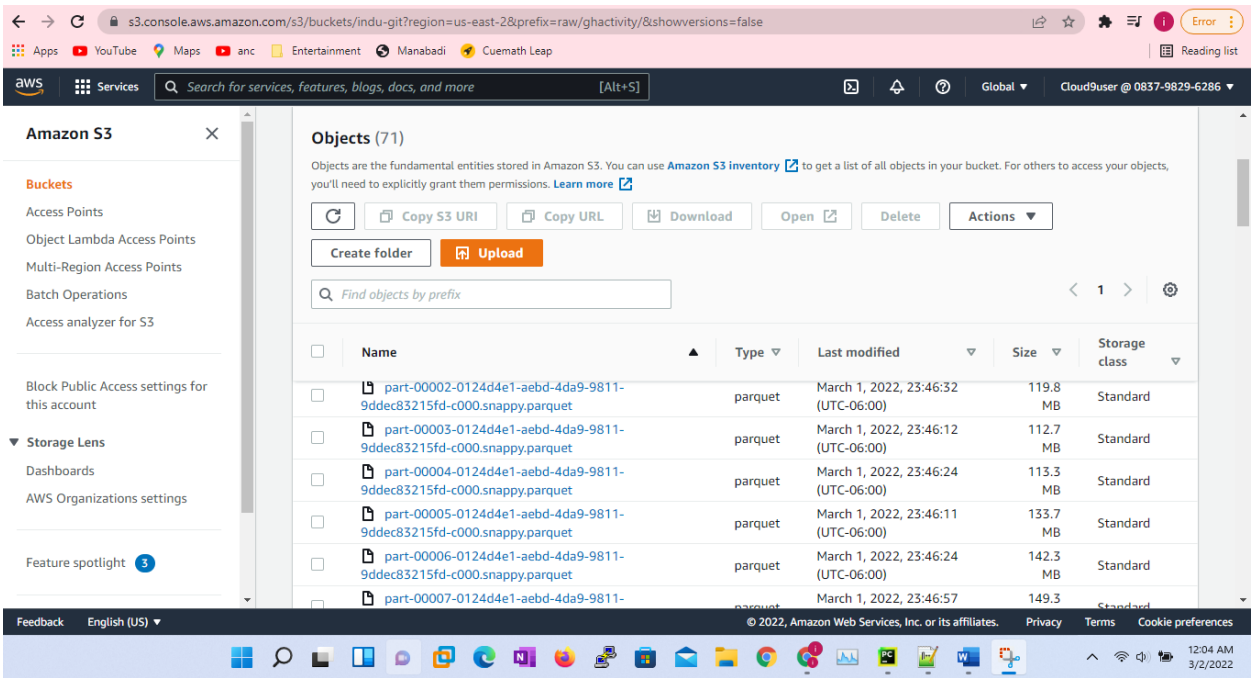



*Figure 2Data is copied into the target location.*




**Glue Script for Partitioning Data:**

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from pyspark.sql.functions import date_format, substring
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext. \
  create_dynamic_frame. \
  from_catalog(
    database = "itvghlandingdb",
    table_name = "ghactivity",
    transformation_ctx = "datasource0"
  )

df = datasource0. \
  toDF(). \
  withColumn('year', date_format(substring('created_at', 1, 10), 'yyyy')). \
  withColumn('month', date_format(substring('created_at', 1, 10), 'MM')). \
  withColumn('day', date_format(substring('created_at', 1, 10), 'dd'))

dyf = DynamicFrame.fromDF(dataframe=df, glue_ctx=glueContext, name="dyf")

datasink4 = glueContext. \
  write_dynamic_frame. \
```

```
  from_options(frame=dyf,
    connection_type="s3",
    connection_options={"path": "s3://itv-github/raw/ghactivity/",
      "compression": "snappy",
      "partitionKeys": ["year", "month", "day"]},
    format="glueparquet",
    transformation_ctx="datasink4")

job.commit()
```