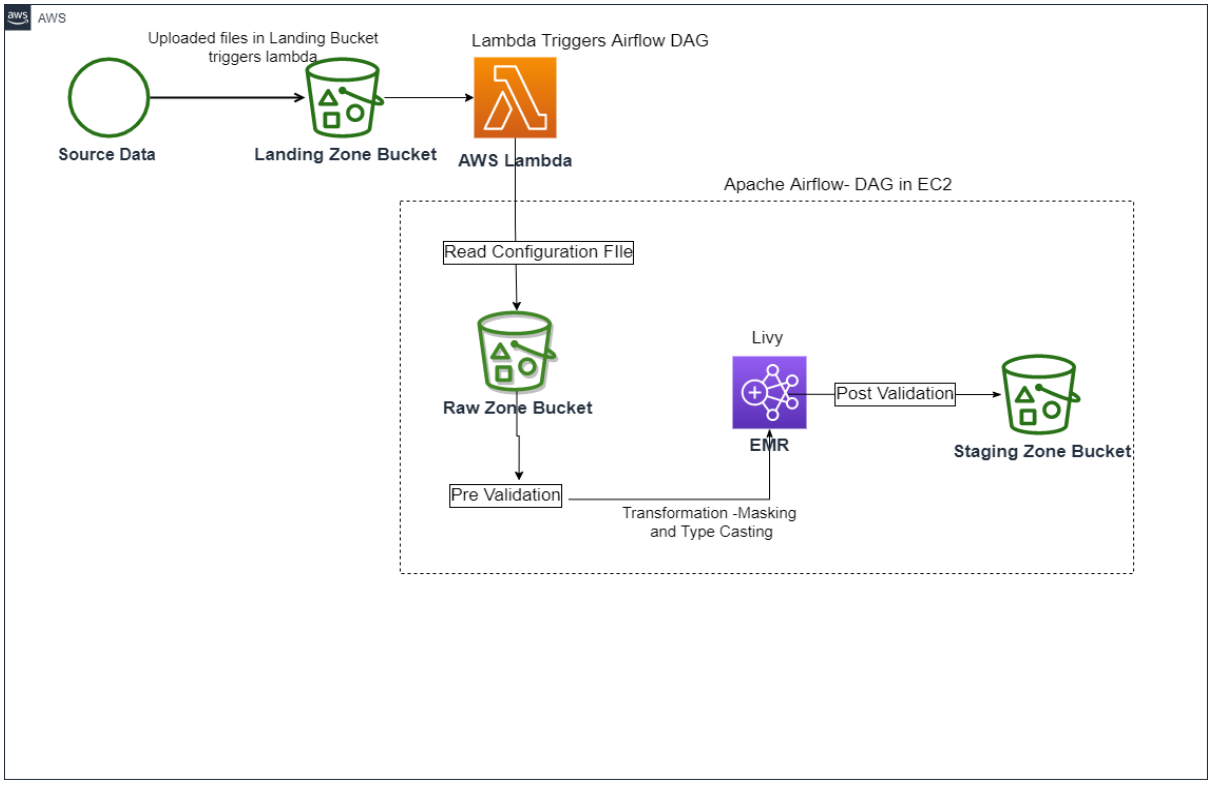
Documentation- POC PROJECT

Milestone 1:

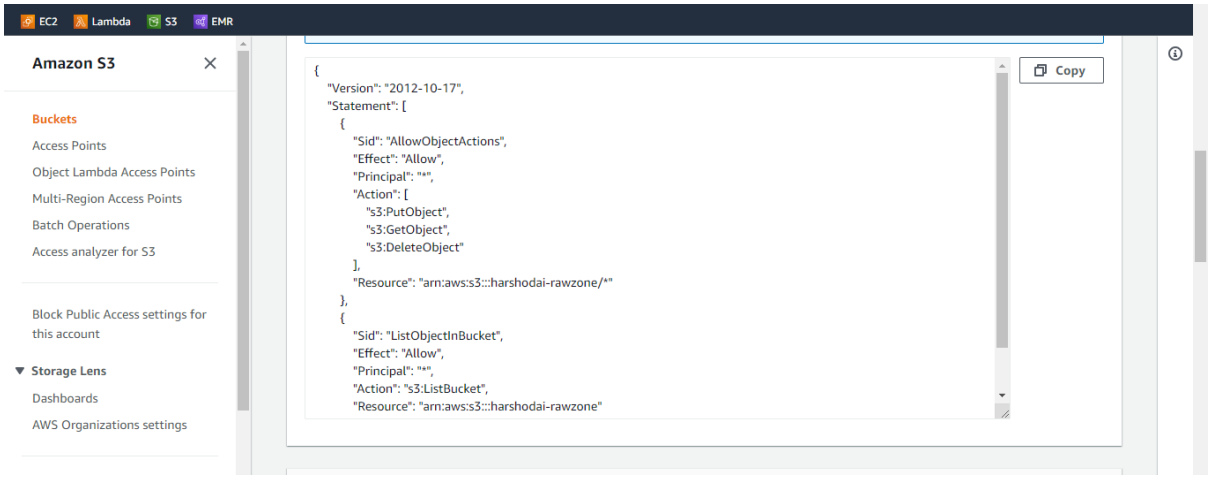
- Created a high-level architectural diagram with efficient solution to solve the problem.
- The Algorithm includes steps like creating a S3 buckets for Data Lake namely Raw Zone, Standing Zone, Landing Zone.
- The Landing Zone is an S3 Bucket which is used for storing raw data from Source Data and when the files are uploaded in this bucket, AWS Lambda is triggered.
- With the help of Lambda, it triggers a DAG (Directed Acyclic Graphs) which helps in organizing the flow of execution in AWS EC2.
- When Airflow is triggered, then the configuration files are read and are stored in a Raw Zone bucket is created with a restriction to avoid all public access and only the authorised personal must get access to it.
- Then the Process of Pre-Validation occurs
- Then Transformation like Masking, Type Casting and Precision up to 7 decimal places are done in EMR(Elastic Map Reduce) where a spark job needs to be done under the supervision of Livy
- Then all the files are then to be uploaded in S3 bucket (Staging Zone)

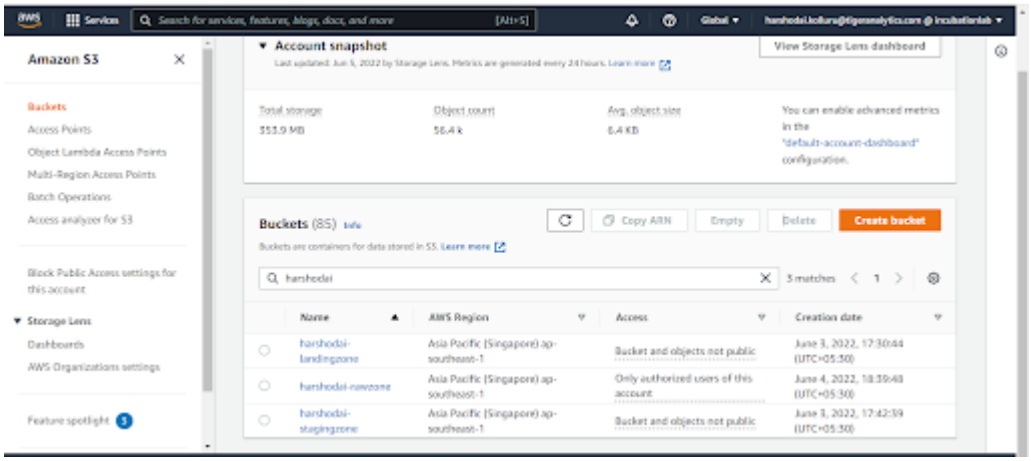


Milestone 2:

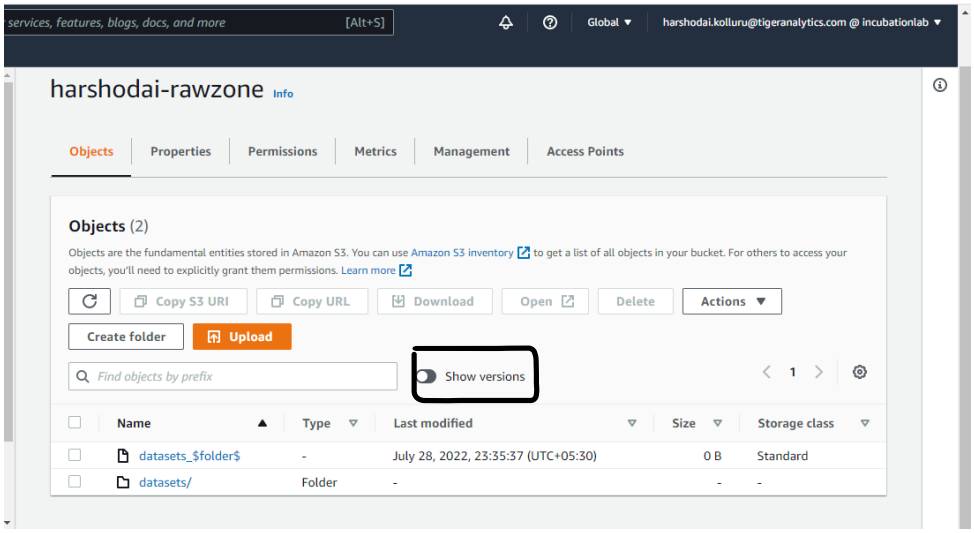
- Creation of three S3 buckets for data lake- Landing Zone, Staging Zone, Raw Zone.
- Block public access for Landing Zone and Staging Zone and in Raw Zone which needs to have restricted access to only authorized users we need to change the bucket policy by generating policy in the edit section. Later we need to block all public access for the Raw Zone. This will be done in Properties tab in S3 console
- We need to add Life cycle management for all the three buckets under management tab in S3 console
- While creating Life cycle management change the current buckets to standard IA for 60 days and apply the permanent deletion of incomplete Downloads
- Creating Sample datasets for actives and viewership by using Faker library in python in any of IDLE and also the data in csv format
- Change the format of the csv files to parquet format using the pandas library in python.

- Add the bucket policy listed below:





For enabling versioning:



MILESTONE 3:

- To create a Spark Job in Pyspark that is able to read data from the raw zone and after the transformation put the data in the staging zone.
- The Pyspark code must be able to implement casting, masking on the required columns
- App configuration and spark configuration are configured in json format includes the location of the file in S3 buckets and also includes the source and destination paths along with the transformation columns and PII columns

app_config - Notepad

File Edit Format View Help

```
{

        "ingest-Actives": {
                 "source": {
                        "data-location": "s3://harshodai-landingzone/datasets/actives1.parquet",
                        "file-format": "parquet"
                },

                "destination": {
                        "data-location": "s3://harshodai-rawzone/datasets/actives1",
                        "file-format": "parquet"
                },

                "masking-cols": [],
                "transformation-cols": [],
                "partition-cols": []


        },
        "masked-Actives": {
                "source": {
                        "data-location": "s3://harshodai-rawzone/datasets/actives1",
                        "file-format": "parquet"
                },
                "destination": {
                        "data-location": "s3://harshodai-stagingzone/result/actives1",
                        "file-format": "parquet"
                },
                "masking-cols": ["advertising_id", "user_id"],
                "transformation-cols": {
                        "user_latitude":"DecimalType,7",
                        "user_longitude":"DecimalType,7",
                        "location_source":"StringType"
                },
                "partition-cols": ["month","date"]
```

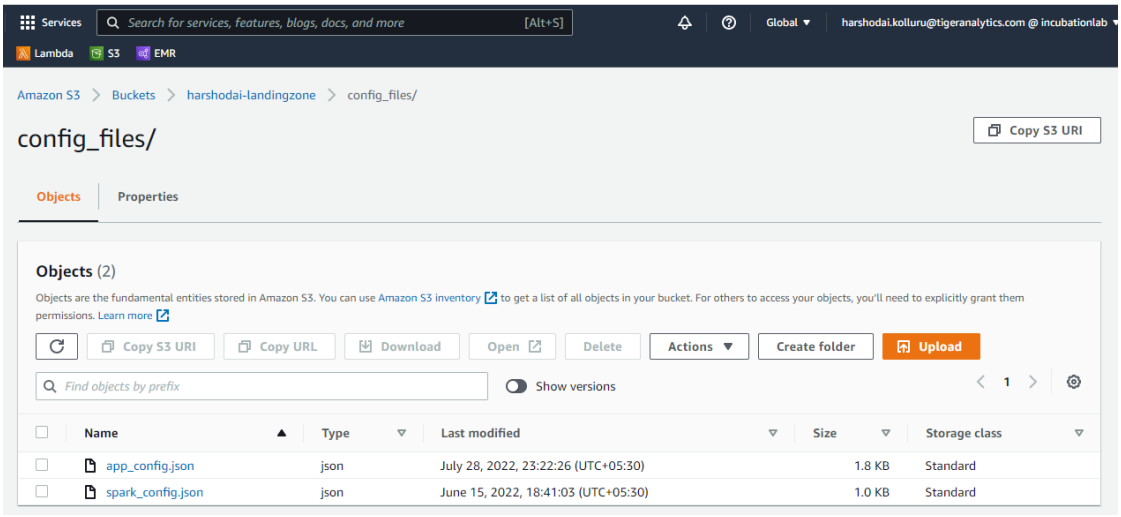
- Spark Configuration includes the software configuration

spark_config - Notepad

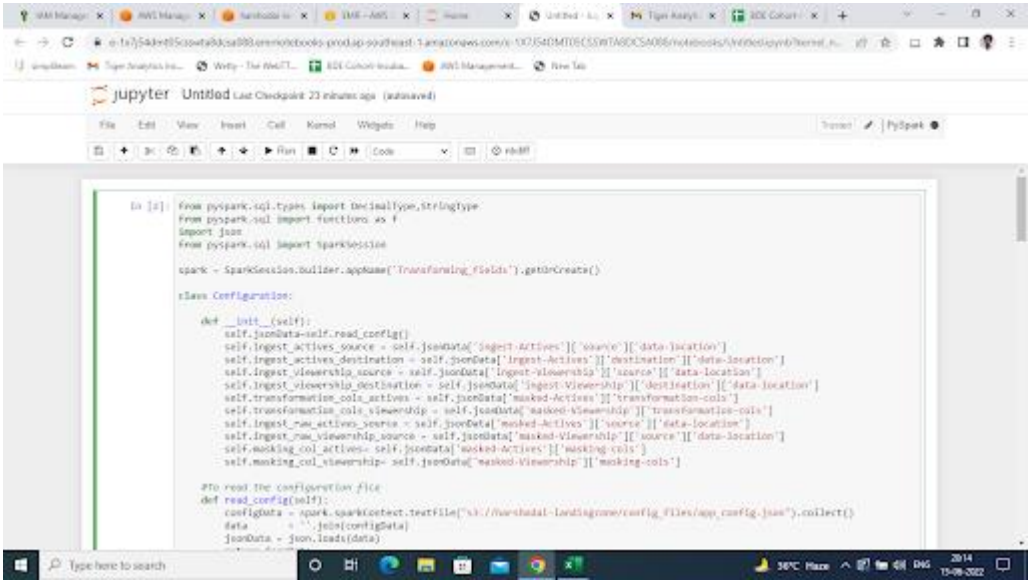
File Edit Format View Help

```
[
  {
    "Classification": "spark",
    "Properties": {
      "spark.path":"s3://harshodai-landingzone/config_files/app_config.json",
      "spark.master":"yarn",
      "spark.submit.deployMode":"cluster",
      "spark.driver.memory":"4g", //amount of memory to use for the driver process
      "spark.executor.memory":"6g", //amount of memory to use per executor process
      "spark.driver.cores":1, //number of cores to use for the driver process only in cluster mode
      "spark.executor.cores":2,
      "spark.dynamicAllocation.enabled" : true,
      "spark.dynamicAllocation.initialExecutors":10, //intial number of executors to run
      "spark.dynamicAllocation.minExecutors":10,
      "spark.dynamicAllocation.maxExecutors":100,
      "spark.sql.shuffle.partitions":500,
      "spark.hadoop.fs.s3a.block.size":"512M"
    }
  }
]
```

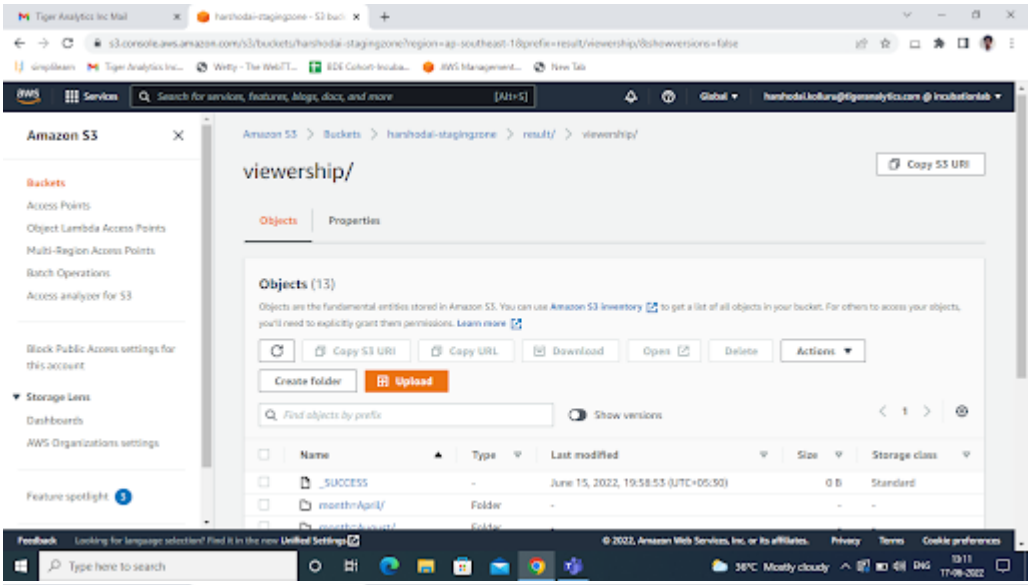
- Save these configuration files in S3 landing zone bucket

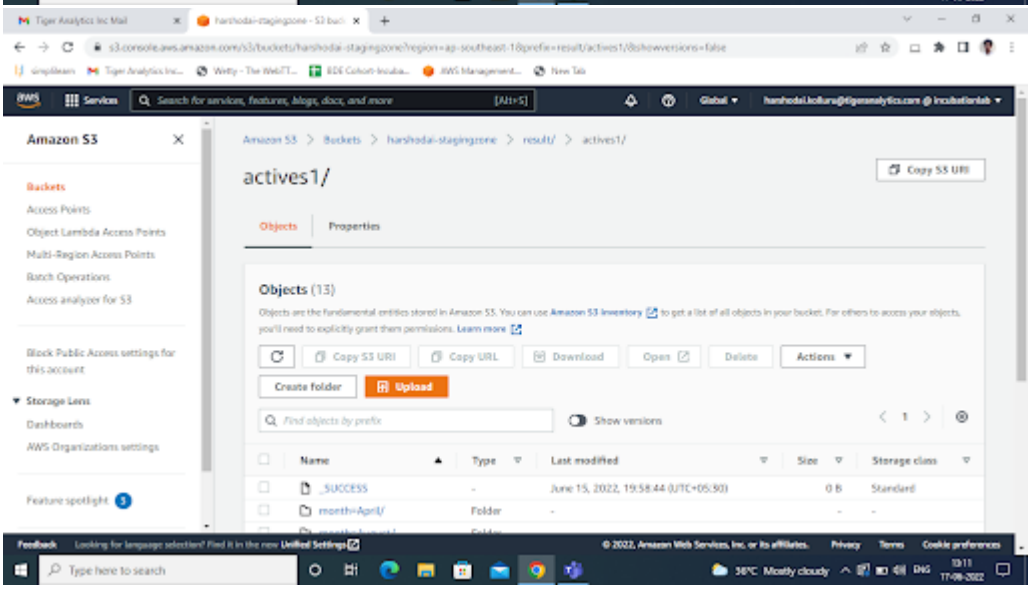


- Create a EMR cluster with a key pair and after creating an emr cluster create a jupyter notebook and paste the spark job



- After running the code see the results in Staging zone bucket for the two datasets: Actives and Viewership



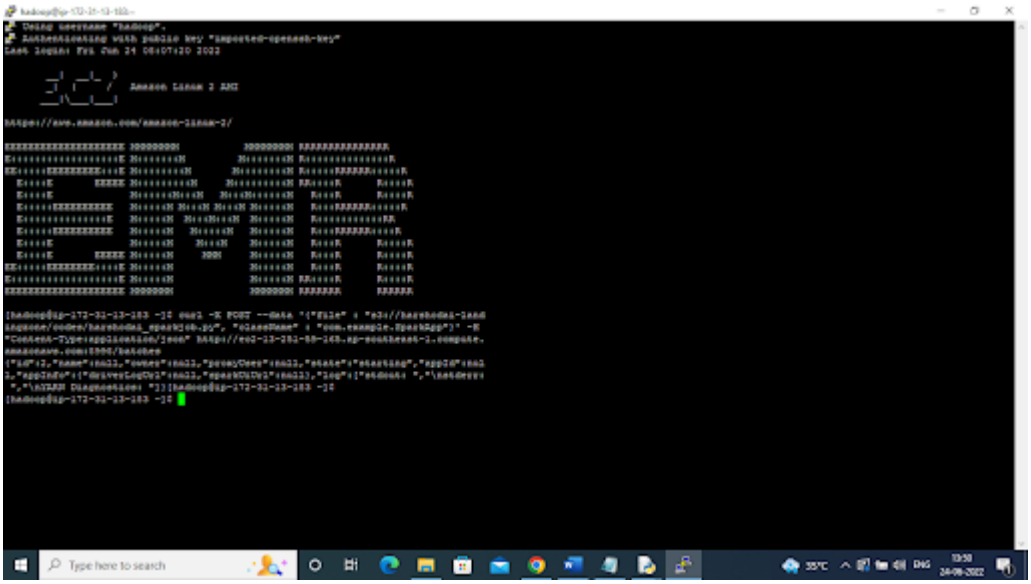


- Terminate the cluster and notebook after the successful output

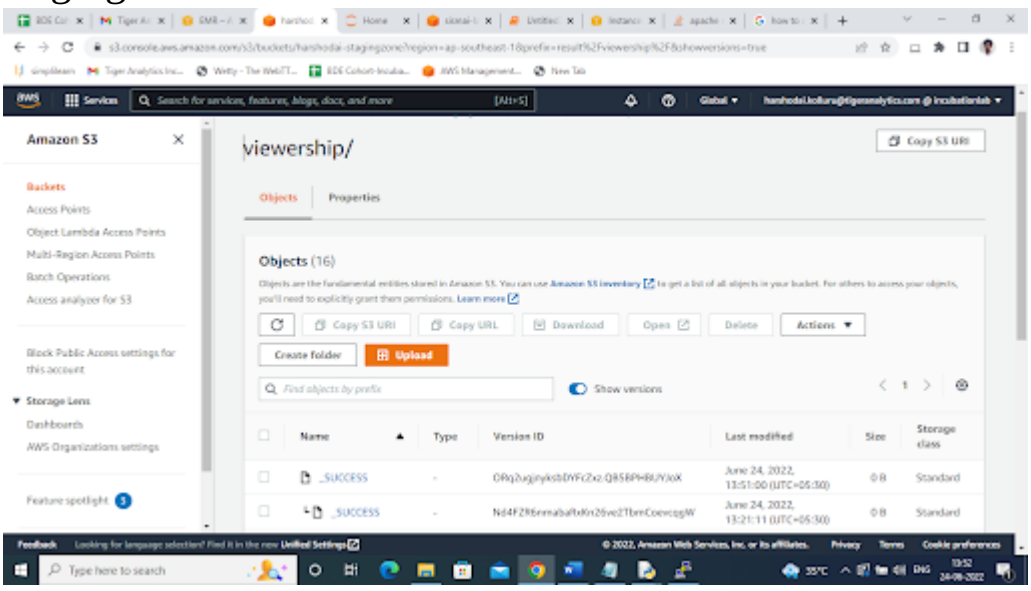
MILESTONE 4:

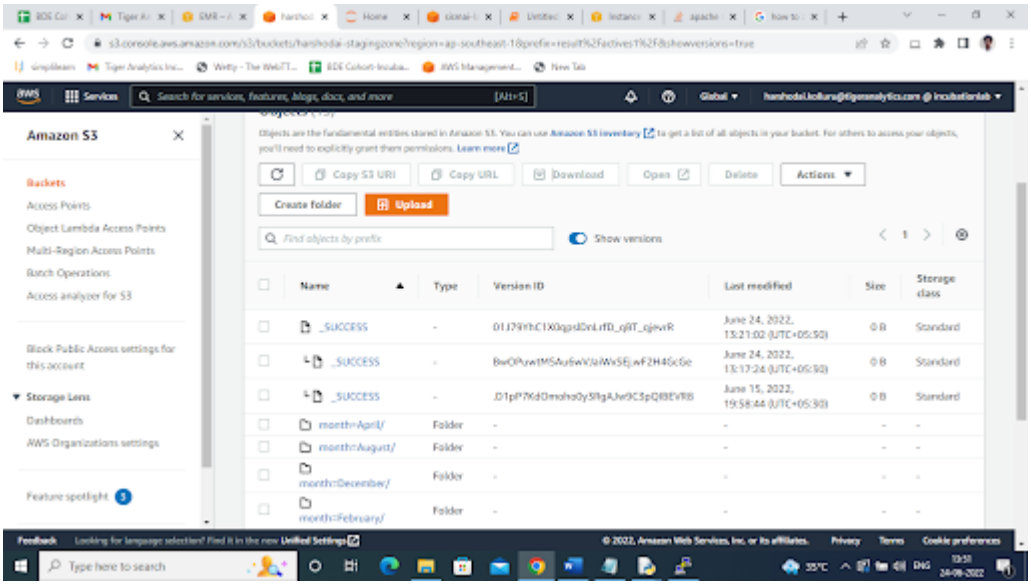
- Implementation of Spark Job using Livy (REST API interface) using Curl commands
- Create a EMR cluster and add Livy to our cluster and goto connect to instance using SSH
- **curl -X POST --data '{"file" : "s3://harshodai-landingzone/codes/harshodai_sparkjob.py", "className" : "com.example.SparkApp"}' -H "Content-Type:application/json" http://ec2-13-251-59-165.ap-southeast-1.compute.amazonaws.com:8998/batches**

- Open and initate the putty with the public IPv4 address of cluster (e.g.:hadoop@ip-xxx-xx-xxx)and add the above command
- Upon writing the command it will be implemented successfully and the output will be updated in staging zone



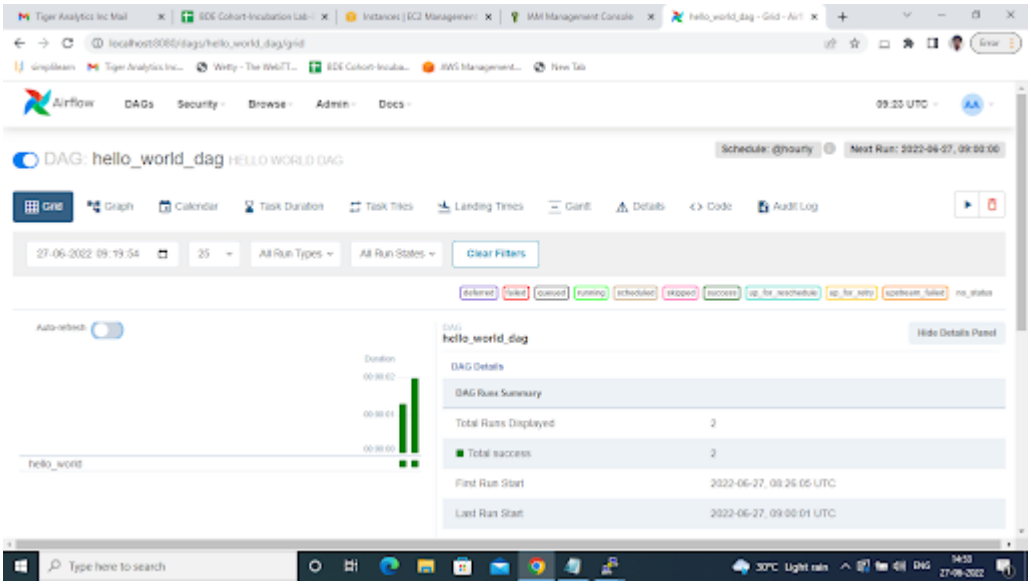
- The output versions of actives and viewership will be seen in staging zone



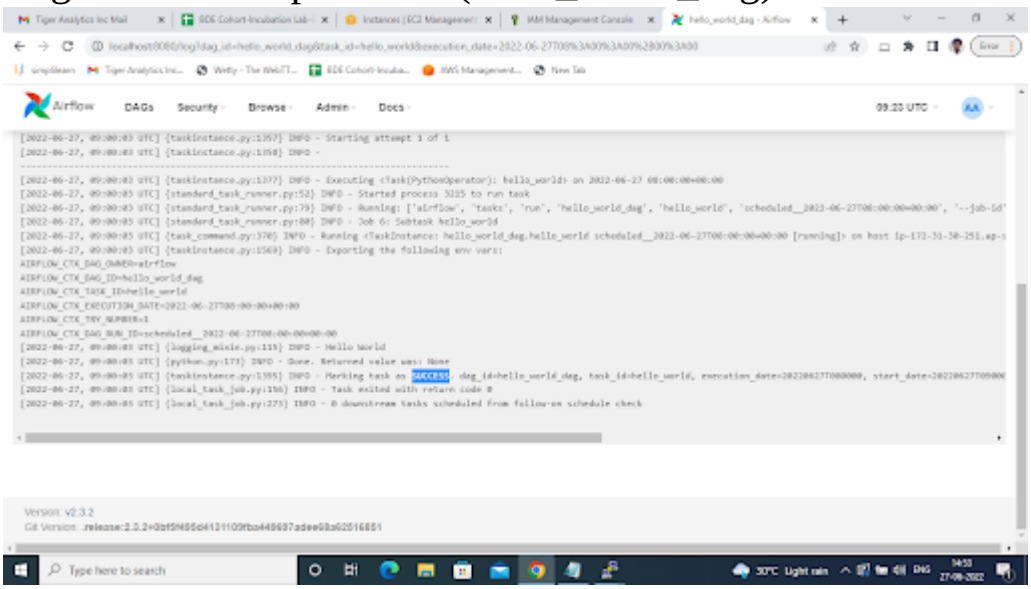


MILESTONE 5:

- To set up Airflow on EC2 select ubuntu as the primary server and select t3.medium for stable processing and optimization
- Upon successful completion of EC2 instance connect using putty and install anaconda using the wget command from the anaconda website
- After installation of anaconda in a python virtual environment install airflow using “pip install airflow”
- After successful installation you can access the virtual environment by “source activate <name of your virtual environment”
- Add an Inbound rule in EC2 security group for accessing the port 8080 for airflow in SSH connection
- To open the airflow server type the command airflow webserver in putty virtual environment
- To schedule the task open a duplicate session and type airflow scheduler and add the public DNS to tunnel for reducing the connection issues.
- Write a sample DAG in the jupyter notebook provided by anaconda
- So the DAG will be triggered and the status will be shown
-



Logs for the sample DAG (hello_world_dag)



Milestone 6:

- To create a high-level DAG capable of the following requirements:
- Read config file
- Copy raw data as it is from the landing zone and copy to raw zone
- Pre-validation
- Submit EMR-Spark with Livy (to transform)
- Copy transformed data to a staging zone
- Post-validation

- Upon initiating the DAG code using python programming language and opening the airflow webserver and scheduler
- Trigger the DAG using the trigger with config and add the following files along with S3 source paths in json format as below:

  {

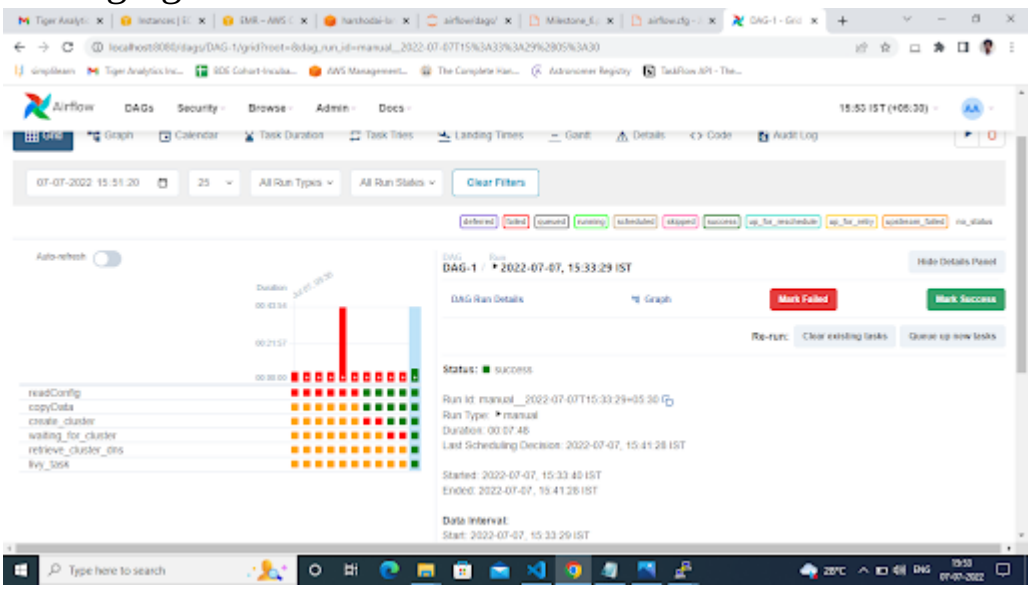
  "app_config": "s3://harshodai-landingzone/config_files/app_config.json",

  "dataset_path": "s3://harshodai-landingzone/datasets/actives1.parquet",

  "spark_code_path": "s3://harshodai-landingzone/codes/harshodai_sparkjob.py",

  "spark_configuration": "s3://harshodai-landingzone/config_files/spark_config.json"

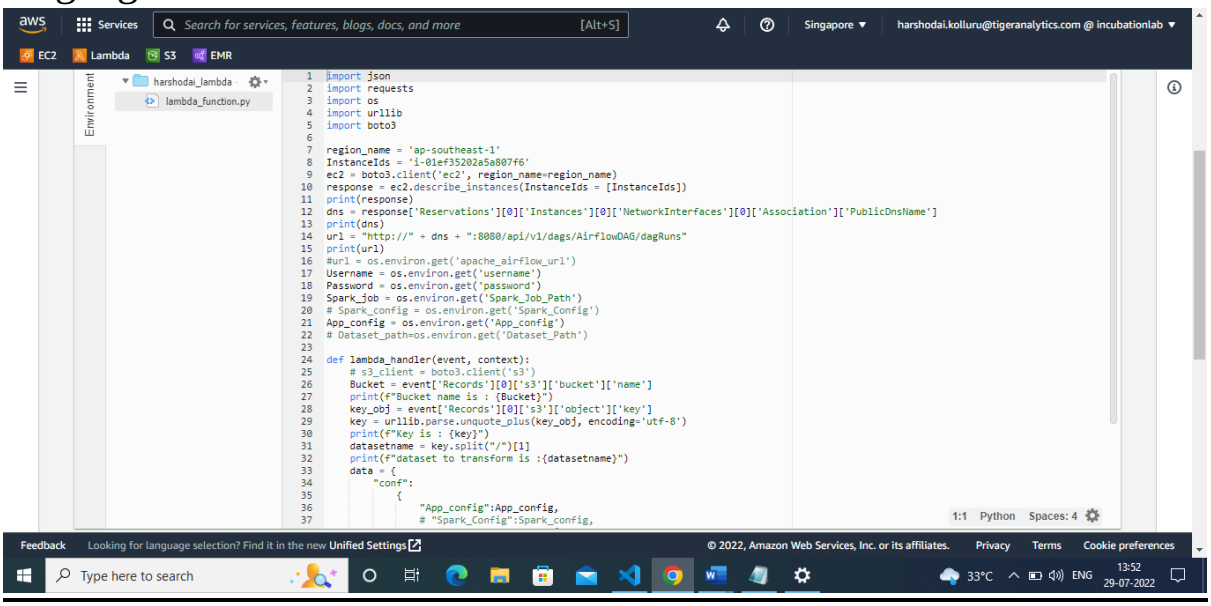
  }
- So the DAG will be triggered and the results will be updated in S3 staging zone bucket

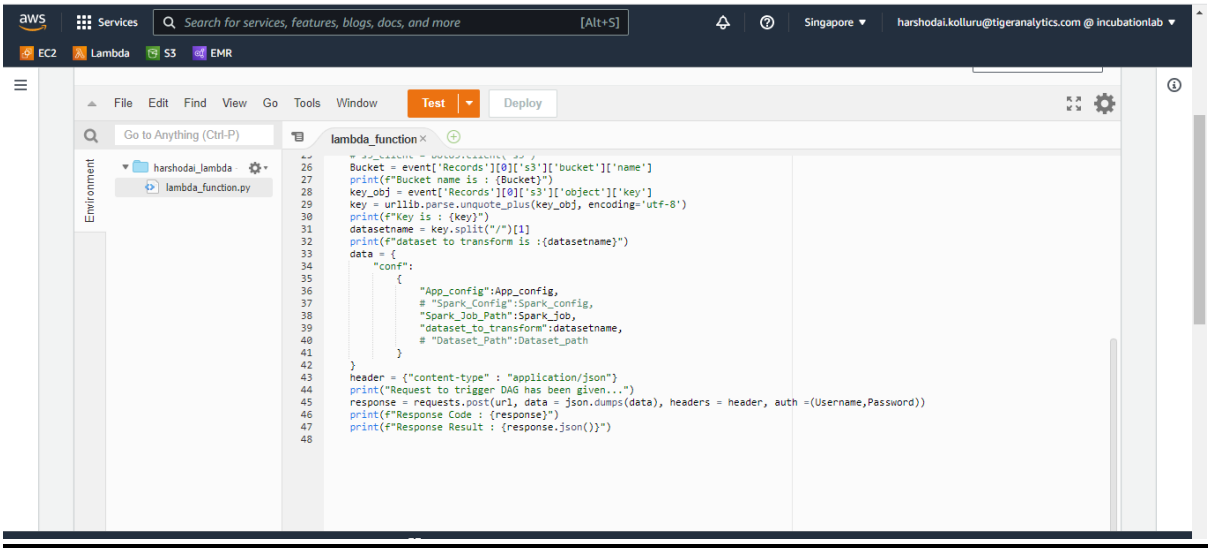


Milestone 7:

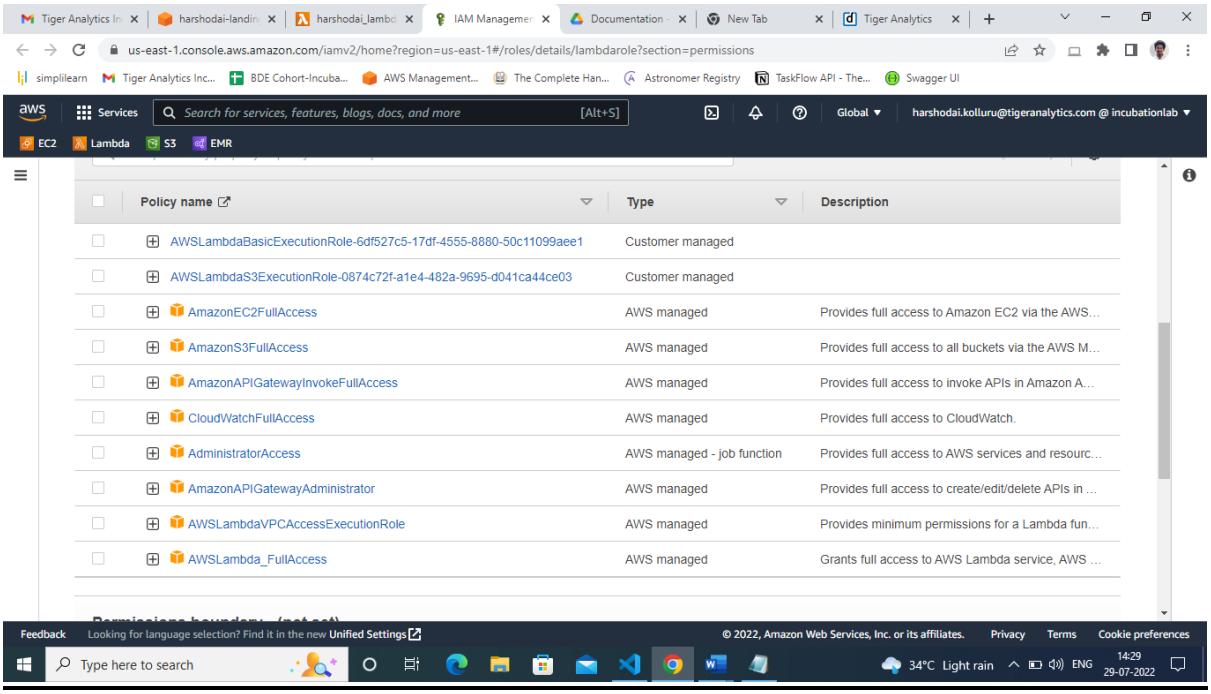
- In this milestone we use AWS Lambda to trigger the DAG whenever a dataset is inserted into the landing zone bucket

- We use the boto3, requests modules etc and obtain the environmental variables by using os module (os.environ.get("variable name listed"))
- We specify the configuration variables and the bucket and key values in the lambda code console using python programming language

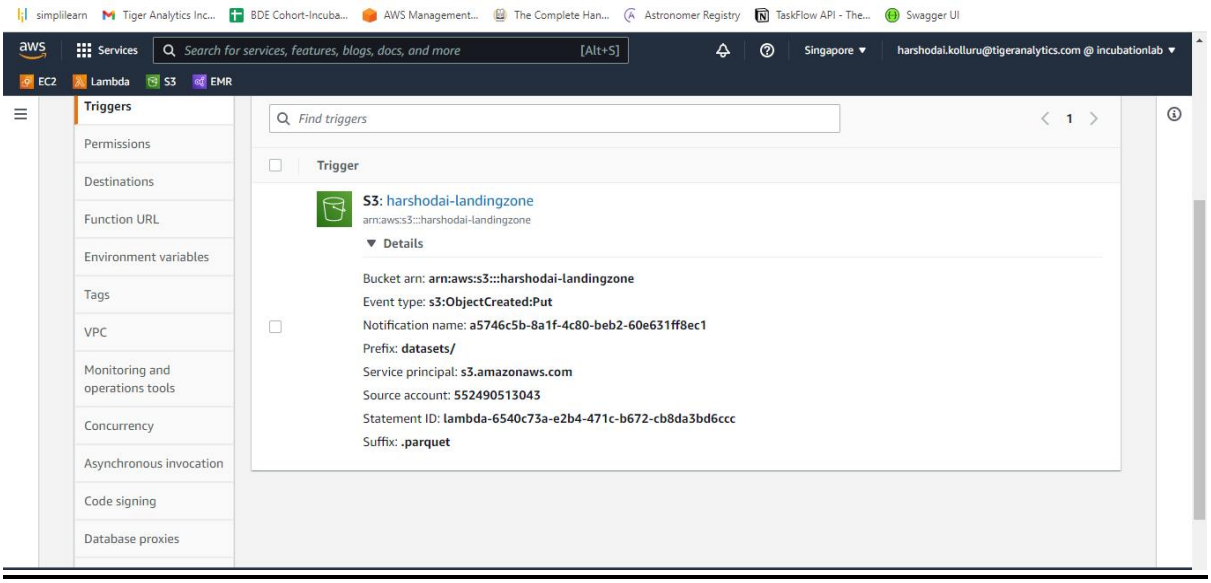




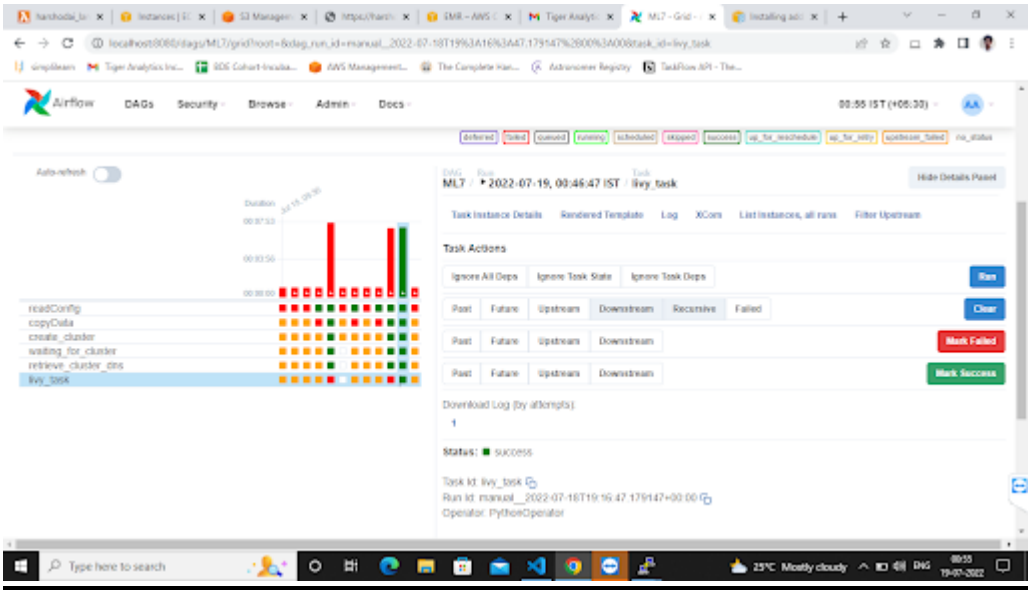
- Assign required IAM roles for lambda



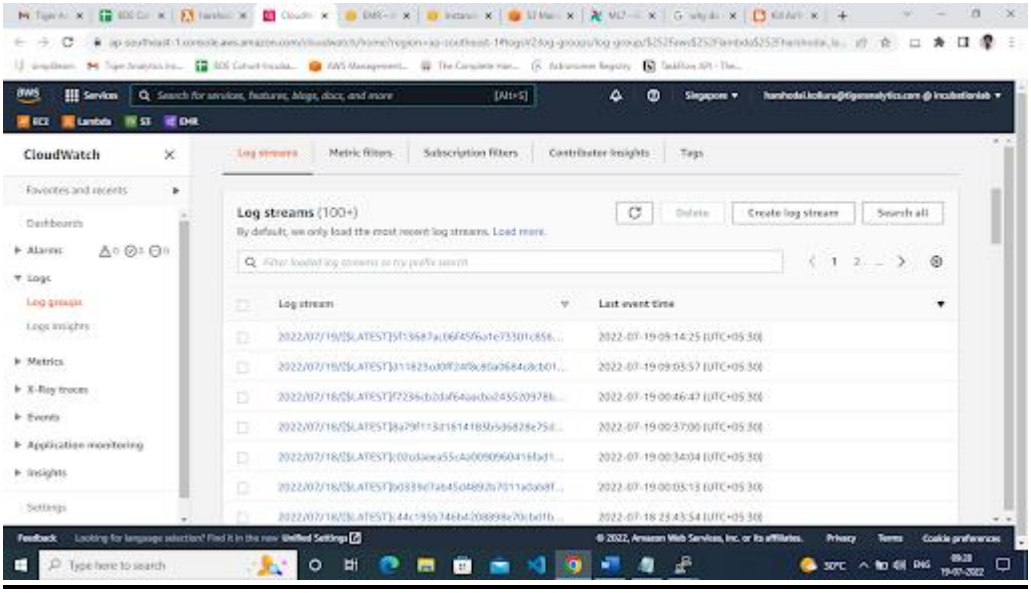
- Create a trigger for landing zone S3 bucket



- After adding these upload a file in the landing zone under datasets/ folder so the DAG will be triggered



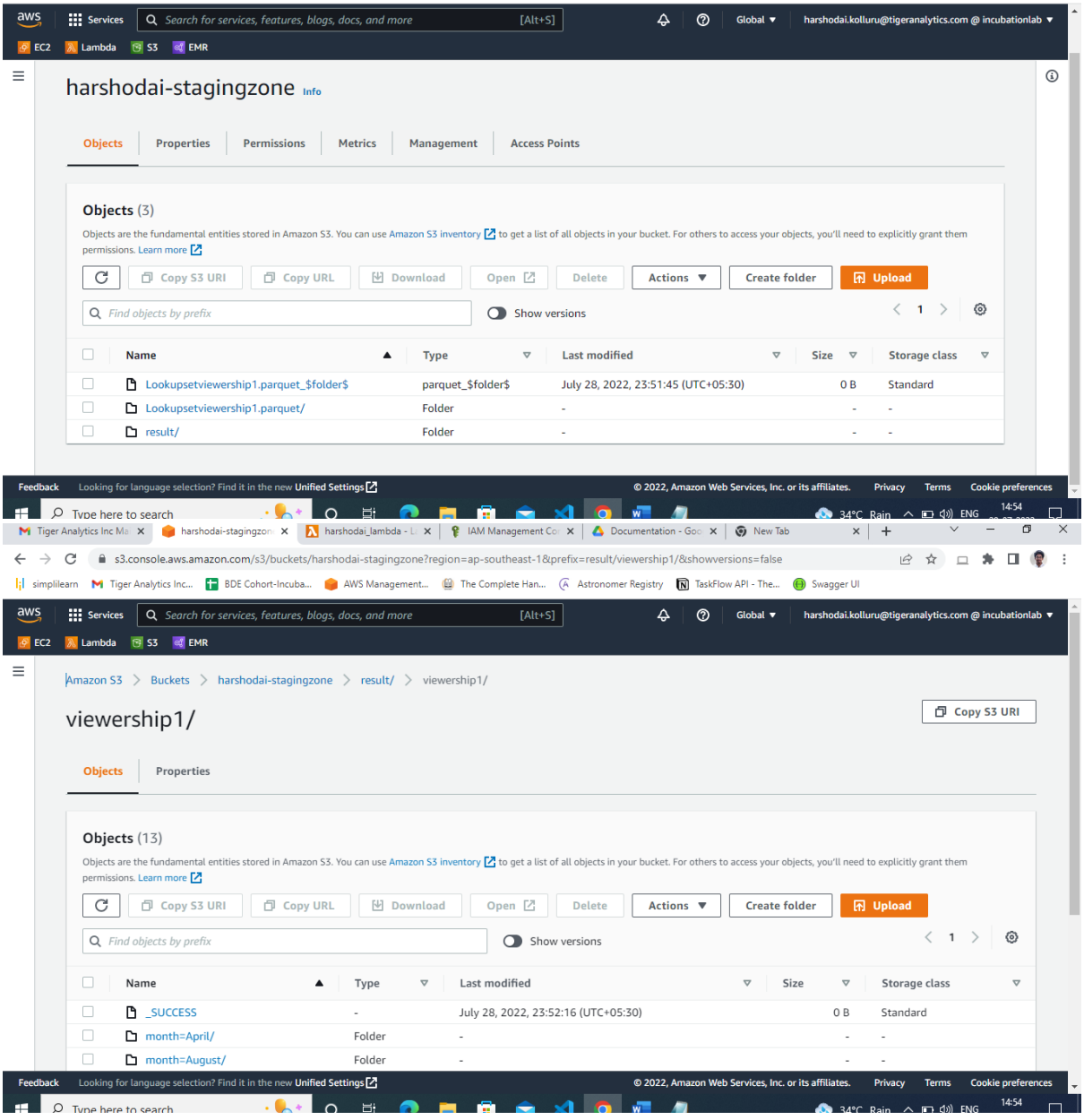
- We can monitor the logs in Cloud watch



Milestone 8:

- Implement data availability check and count check i.e checking the count of both rows and columns and data type check, SCD2 implementation
- The parquet files are read using s3fs module and pyarrow module (ParquetDataset) and later importing them into dataframe by using pandas library
- Count check is implemented by checking the count of the rows and columns of the 2 buckets

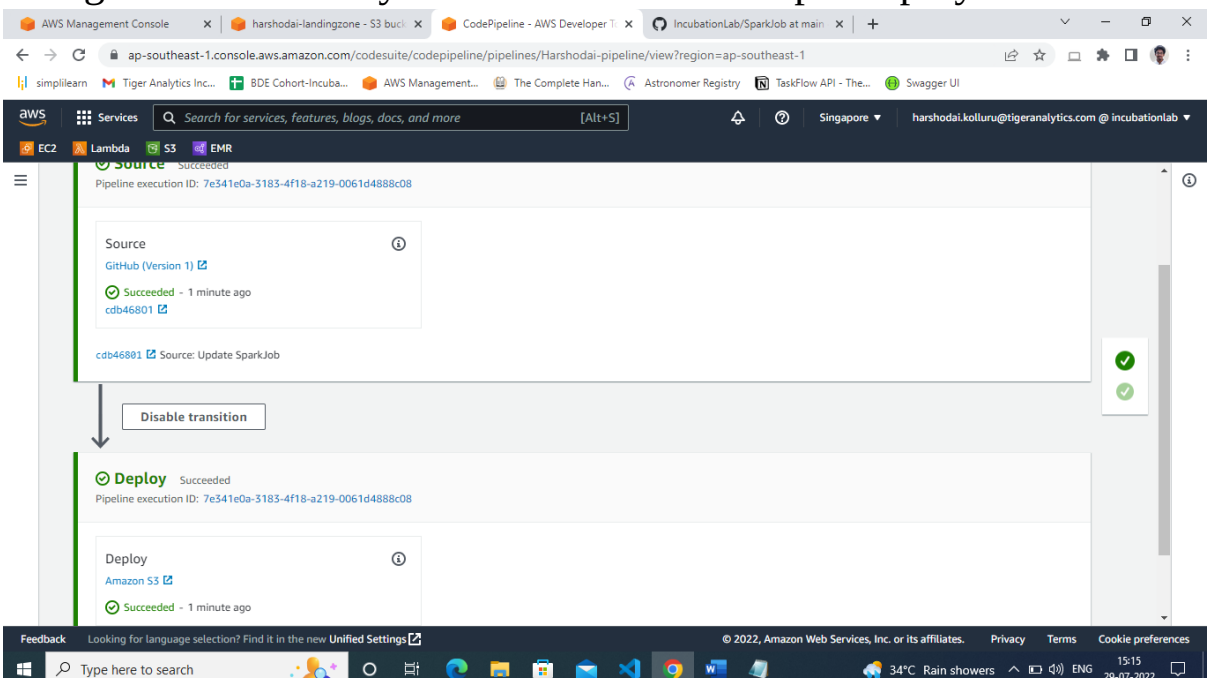
- Data type check is implemented by checking the required column datatype and by importing DecimalType and StringType from pyspark.sql module
- Implementation of SCD2 is done in the spark job by taking the required columns for creating masked and unmasked columns
- Upon successful completion, the results will be uploaded to staging zone



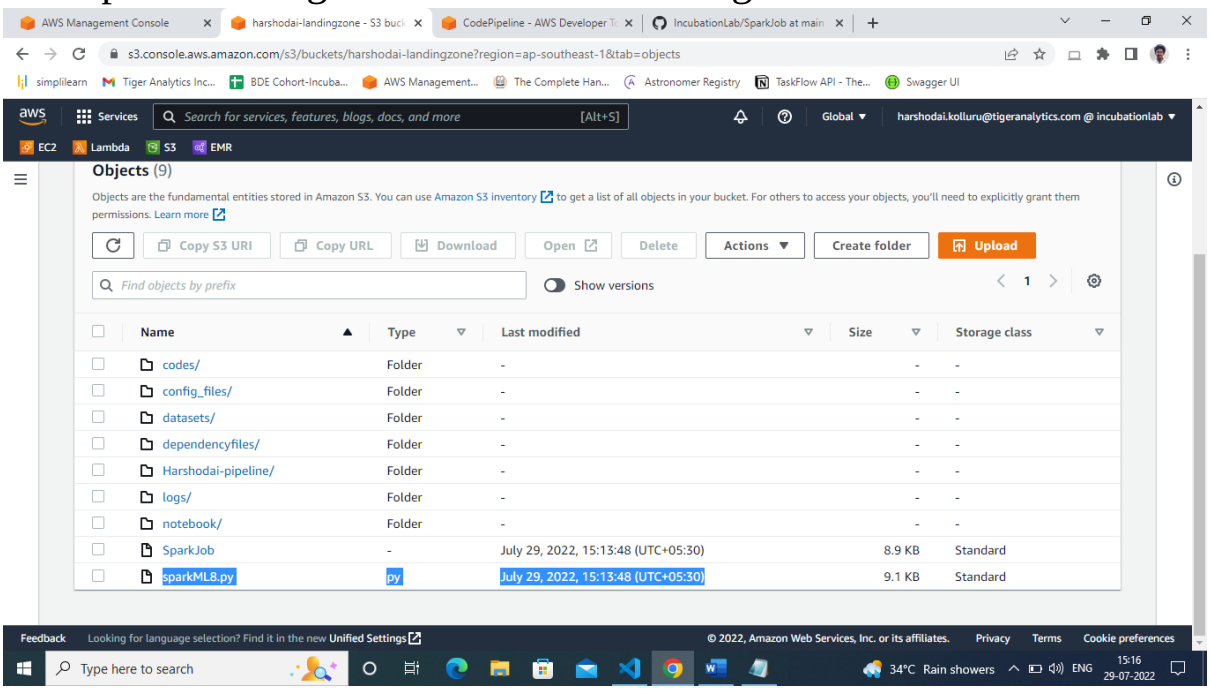
Milestone 9:

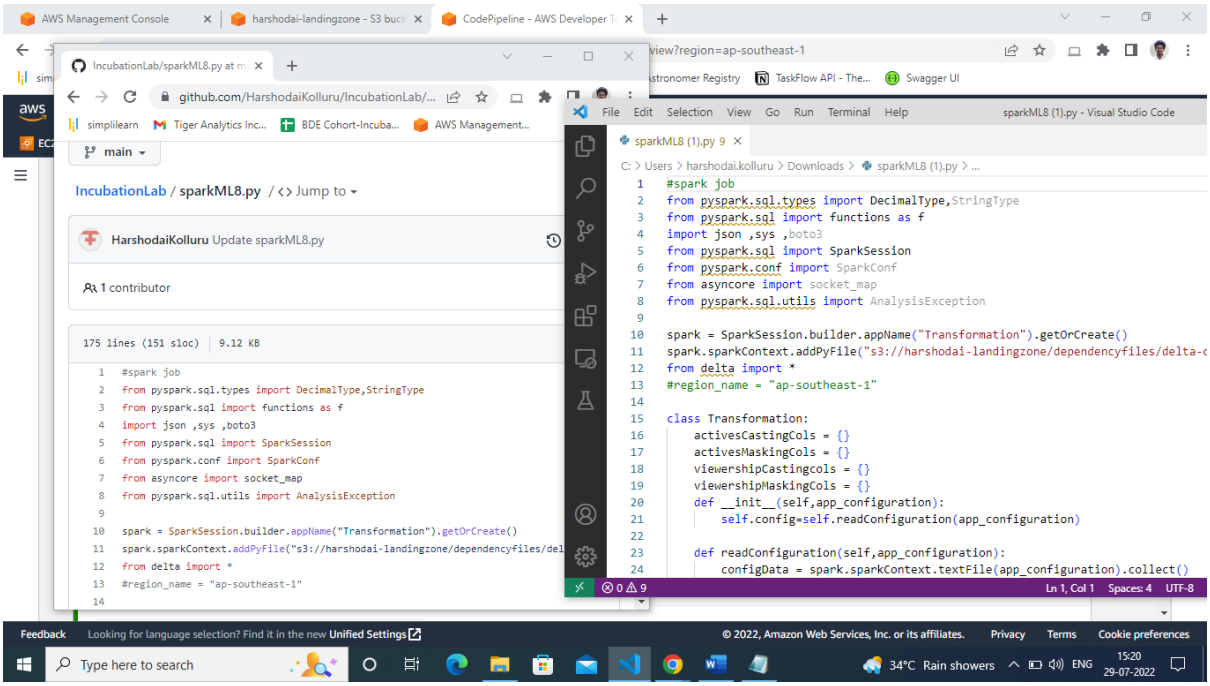
- To implement a pipeline by using AWS code pipeline

- Create a basic pipeline and by adding source as github(version 1) and need to be deployed in landing zone bucket even though changes are added they are reflected in the output deployment



- The updated changes can be seen in landing zone




IncubationLab/sparkML8.py at m
github.com/HarshodaiKolluru/IncubationLab/...
main
IncubationLab / sparkML8.py / Jump to
HarshodaiKolluru Update sparkML8.py
1 contributor
175 lines (151 sloc) 9.12 KB
#spark job
from pyspark.sql.types import DecimalType,StringType
from pyspark.sql import functions as f
import json ,sys ,boto3
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf
from asyncore import socket_map
from pyspark.sql.utils import AnalysisException
spark = SparkSession.builder.appName("Transformation").getOrCreate()
spark.sparkContext.addPyFile("s3://harshodai-landingzone/dependencyfiles/del
from delta import *
#region_name = "ap-southeast-1"
sparkML8 (1).py - Visual Studio Code
sparkML8 (1).py 9
C: > Users > harshodai.kolluru > Downloads > sparkML8 (1).py > ...
#spark job
from pyspark.sql.types import DecimalType,StringType
from pyspark.sql import functions as f
import json ,sys ,boto3
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf
from asyncore import socket_map
from pyspark.sql.utils import AnalysisException
spark = SparkSession.builder.appName("Transformation").getOrCreate()
spark.sparkContext.addPyFile("s3://harshodai-landingzone/dependencyfiles/delta-c
from delta import *
#region_name = "ap-southeast-1"
class Transformation:
activesCastingCols = {}
activesMaskingCols = {}
viewershipCastingcols = {}
viewershipMaskingCols = {}
def __init__(self,app_configuration):
self.config=self.readConfiguration(app_configuration)
def readConfiguration(self,app_configuration):
configData = spark.sparkContext.textFile(app_configuration).collect()
Ln 1, Col 1
Spaces: 4
UTF-8
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
34°C Rain showers
ENG
15:20
29-07-2022