



POC on Orchestration using Airflow

Apache Airflow

Airflow is a platform to programmatically author, schedule and monitor workflows.

Airflow is used to author workflows as Directed Acyclic Graphs (DAGs) of tasks.

When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative.

Amazon Managed Workflows for Apache Airflow (MWAA)

- MWAA is a Managed orchestration service for Apache Airflow that helps to setup and operate end-to-end data pipelines in the cloud at scale.
 - Apache Airflow is an open-source tool used to programmatically author, schedule, and monitor sequences of processes and tasks referred to as "workflows."
 - MWAA uses Airflow and Python to create workflows without having to manage the underlying infrastructure for scalability, availability, and security.
-

POC Objective

- Data pipeline creation using DAG file in Airflow environment
 - Creation of Airflow environment MWAA in AWS
 - AWS EMR creation, Spark Job submission and Status check as steps in DAG file
 - Spark Job - Retail data staged in S3 is read and transformed based on business KPIs and loaded into S3 tables
 - Transformed data is queried using AWS Athena
-

DAG (Directed Acyclic Graph)

- Collection of all the tasks to be run organised by their relationships and dependencies.
 - A DAG is defined in a Python script, which represents the DAGs structure as code.
-

About retail data set

This is a transactional data set which contains all the transactions for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

Data set path in S3

s3://retail-sankir retail-sankir bucket in AWS S3 has the retail data files

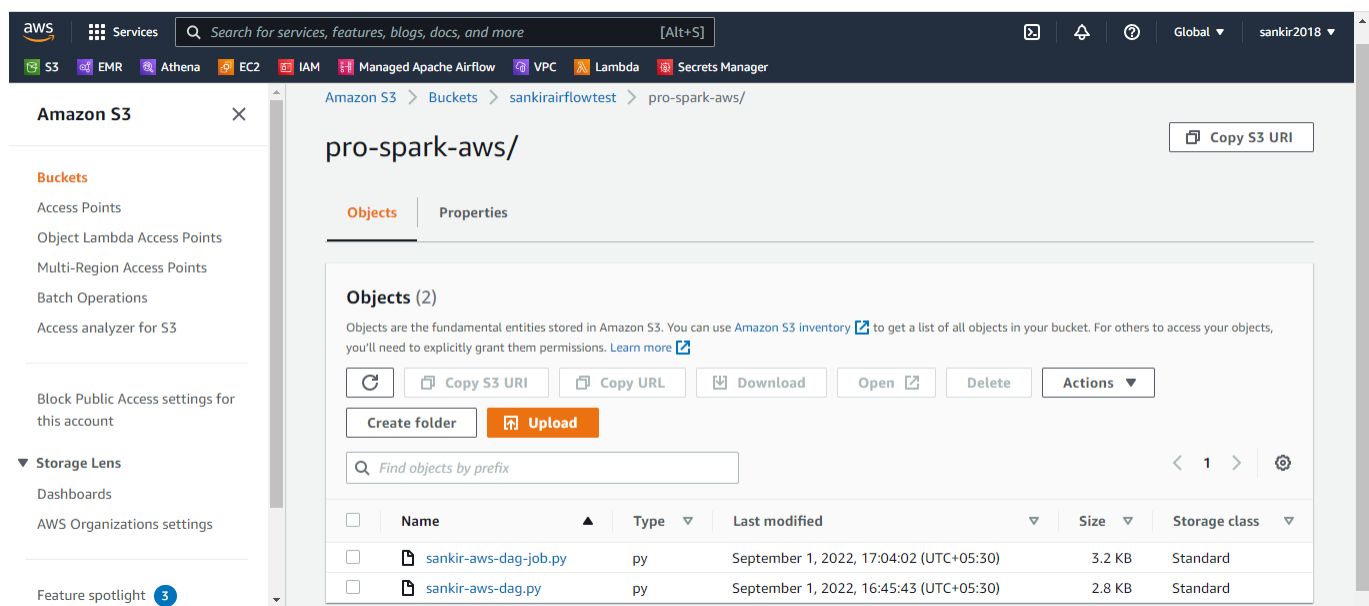
DAG folder path in S3

s3://sankirairflowtest/pro-spark-aws/

sankirairflowtest bucket in AWS S3 has the DAG folder, pro-spark-aws

s3://sankirairflowtest/pro-spark-aws/sankir-aws-dag-job.py

DAG folder has the DAG file, sankir-aws-dag-job.py



requirements.txt file

A requirements file is a list of all project's dependencies and specifies version of each dependency.

s3://sankirairflowtest/requirements.txt

sankirairflowtest bucket in AWS S3 has the requirements.txt. file.

Prerequisites for Managed Workflows for Apache Airflow (MWAA) environment creation

- S3 Storage bucket
 - DAG folder where python dag file is stored
 - requirements.txt specifying dependencies
 - IAM role
-

DAG file

The DAG file is as follows:

- Specify DAG_ID
- In Configurations, we specify EMR cluster name.
- Add JOB_FLOW_OVERRIDES to create EMR Cluster
- Add SPARK_STEPS to submit Spark job

Here is the summary of steps executed.

The data pipeline consists of steps like,

- creation of a Spark job cluster in EMR,
- assigning proper roles and permissions,
- submission of Spark job
- status check.

DAG file content

```
import os
from datetime import timedelta
from datetime import datetime as dt

from airflow import DAG
from airflow.operators.dummy import DummyOperator
from airflow.providers.amazon.aws.operators.emr_add_steps import
EmrAddStepsOperator
from airflow.providers.amazon.aws.operators.emr_create_job_flow import (
    EmrCreateJobFlowOperator,
)
from airflow.providers.amazon.aws.sensors.emr_step import EmrStepSensor
from airflow.utils.dates import days_ago

#Configurations
from datetime import datetime as dt
CLUSTER_SUFFIX = dt.now().microsecond
CLUSTER_NAME = "sankir-aws-emr-cluster-airflow-" + str(CLUSTER_SUFFIX)

DAG_ID = "a-job-sankir-aws-emr-01Sep"

DEFAULT_ARGS = {
    "owner": "sankir",
    "depends_on_past": False,
    "email": ["info@sankir.com"],
    "email_on_failure": False,
    "email_on_retry": False,
}

#Spark job submission
SPARK_STEPS = [
```

```

{
  "Name": "sankir-pro-spark-aws",
  "ActionOnFailure": "CONTINUE",
  "HadoopJarStep": {
    "Jar": "command-runner.jar",
    "Args": [
      "spark-submit",
      "--class",
      "com.sankir.smp.core.ApplicationMain",
      "--master",
      "yarn",
      "--deploy-mode",
      "cluster",
      "s3://retail-sankir/jar/pro-spark-aws-1.0-SNAPSHOT-twofiles-
nodebug.jar",
    ],
  },
},
]

# AWS EMR (Spark Cluster) Creation
JOB_FLOW_OVERRIDES = {
  "Name": CLUSTER_NAME,
  "ReleaseLabel": "emr-5.35.0",
  "Applications": [
    {"Name": "Spark"},
  ],
  "Instances": {
    "InstanceGroups": [
      {
        "Name": "Master nodes",
        "Market": "ON_DEMAND",
        "InstanceRole": "MASTER",
        "InstanceType": "m5.xlarge",
        "InstanceCount": 1,
      }
    ],
    "Ec2SubnetId": "subnet-0fcf1f4cf643496f1",
    "KeepJobFlowAliveWhenNoSteps": True,
    "TerminationProtected": True,
  },
  "VisibleToAllUsers": True,
  "JobFlowRole": "EMR_EC2_DefaultRole",
  "ServiceRole": "EMR_DefaultRole",
  "Tags": [
    {"Key": "Environment", "Value": "SanKir Retail AWS Environment"},
    {"Key": "Name", "Value": "SanKir EMR Airflow Demo"},
    {"Key": "Owner", "Value": "SanKir Team"},
  ],
}

with DAG(
    dag_id=DAG_ID,
    description="Run Spark app on Amazon EMR",

```

```

default_args=DEFAULT_ARGS,
dagrun_timeout=timedelta(hours=2),
start_date=days_ago(1),
schedule_interval=None,
tags=["emr airflow prospark", "spark"],
) as dag:
    begin = DummyOperator(task_id="begin_retail_de_pipeline")

    end = DummyOperator(task_id="end_retail_de_pipeline")

# for EMR (Spark Cluster) creation
    cluster_creator = EmrCreateJobFlowOperator(
        task_id="create_job_flow", job_flow_overrides=JOB_FLOW_OVERRIDES
    )

# for Spark Job submission
    step_adder = EmrAddStepsOperator(
        task_id="add_steps",
        job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}" ,
        aws_conn_id="aws_default",
        steps=SPARK_STEPS,
    )

# for status check
    step_checker = EmrStepSensor(
        task_id="watch_step",
        job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}" ,
        step_id="{{ task_instance.xcom_pull(task_ids='add_steps',
key='return_value')[0] }}" ,
        aws_conn_id="aws_default",
    )

#Steps executed
    begin >> cluster_creator >> step_adder >> step_checker >> end

```

MWAA Environment creation

Creating data pipeline using MWAA

Click on Create Airflow Environment

- Specify Name and Version
- Select the S3 bucket sankirairflowtest that has source code
- For DAG's folder - Select the bucket folder that has DAG code
- Select requirments.txt
- Create MWAA VPC
- Specify Stack and Environment name
- private and public subnets are listed here
- Click on Create stack.

Once stack is created

- Enable web server access as Public
- Specify security group
- Select the required Environment class that specifies system capacity
- Specify worker and Scheduler count
- Set the Airflow logging configuration to INFO level.
- Under permissions, select the execution role
- Click on Create Environment

Airflow environment is successfully created and when launched, it reads the DAG file.

aws

Services

Search for services, features, blogs, docs, and more

[Alt+S]

S3

EMR

Athena

EC2

IAM

Managed Apache Airflow

VPC

Lambda

Secrets Manager

Ohio

sankir2018

SankirAirflow0109 is being created. This takes 20-30 minutes.

Amazon MWAA > Environments > SankirAirflow0109

SankirAirflow0109

Edit

Delete



Open Airflow UI

Details

<div>Status</div> <div><div>Available</div></div>	<div>Airflow UI</div> <div><div>d20360e8-6a66-4610-8cac-f670f4970426.c5.us-east-2.airflow.amazonaws.com</div></div>
<div>ARN</div> <div><div>arn:aws:airflow:us-east-2:437389349595:environment/SankirAirflow0109</div></div>	<div>Weekly maintenance window start (UTC)</div> <div>Wednesday 12:30</div>

Last update

Trigger the DAG file


Airflow
DAGs
Security
Browse
Admin
Docs
10:23 UTC


DAGs

All 5
Active 0
Paused 5

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
<input type="checkbox"/> EMR-DAG-3 emr demo spark	SANKIR	<div><div></div><div></div><div></div></div>	None		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	<input type="button" value="▶"/> <input type="button" value="↺"/> <input type="button" value="🗑"/>	⋮
<input type="checkbox"/> example_emr_job_flow_automatic_steps-2 emr demo spark	SANKIR	<div><div></div><div></div><div></div></div>	None		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	<input type="button" value="▶"/> <input type="button" value="↺"/> <input type="button" value="🗑"/>	⋮
<input type="checkbox"/> pro-spark-aws-job-10-june emr airflow prospark spark	sankir	<div><div></div><div></div><div></div></div>	None		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	<input type="button" value="▶"/> <input type="button" value="↺"/> <input type="button" value="🗑"/>	⋮
<input checked="" type="checkbox"/> a-job-sankir-aws-emr-01Sep Run Spark app on Amazon EMR	sankir	<div><div></div><div></div><div></div></div>	None		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	<input type="button" value="▶"/> <input type="button" value="↺"/> <input type="button" value="🗑"/>	⋮
<input type="checkbox"/> sankir-retail-aws-emr-athena-ec2 emr airflow prospark spark	sankir	<div><div></div><div></div><div></div></div>	None		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	<input type="button" value="▶"/> <input type="button" value="↺"/> <input type="button" value="🗑"/>	⋮

«
<
1
>
»

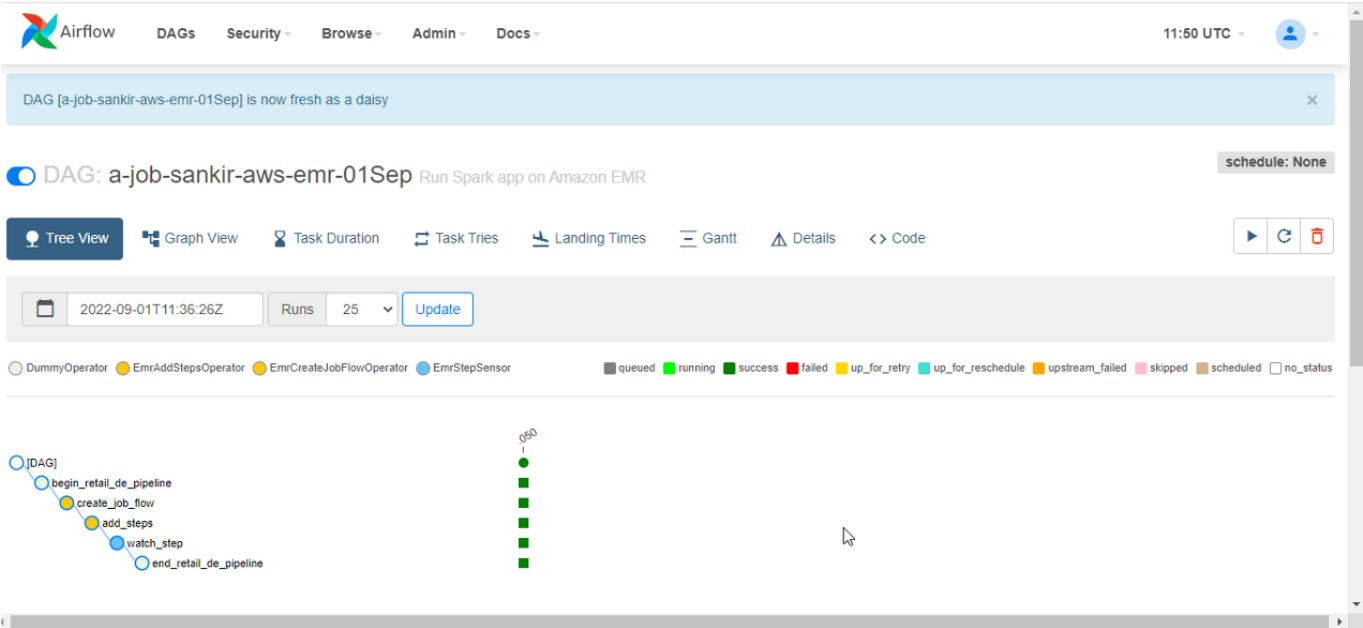
Showing 1-5 of 5 DAGs

- When the Airflow environment is launched, it reads the DAG file.
- Out of the List of DAG files, trigger the DAG file associated with this POC.
- First EMR is created and status check is success
- Next, Spark job is submitted status check is success

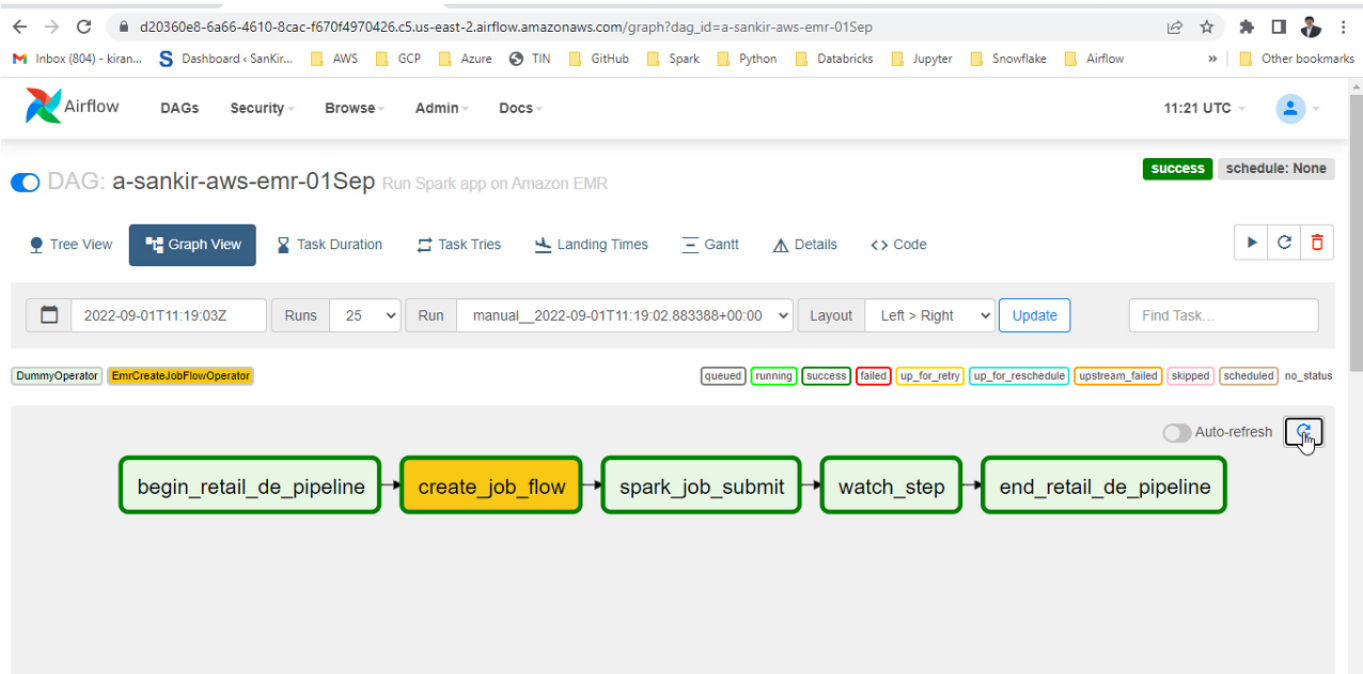
So EMR creation and Spark job submission tasks are successfully executed in MWAA environment using DAG.

Views in Airflow Environment

Here is the tree view of tasks in Airflow environment.

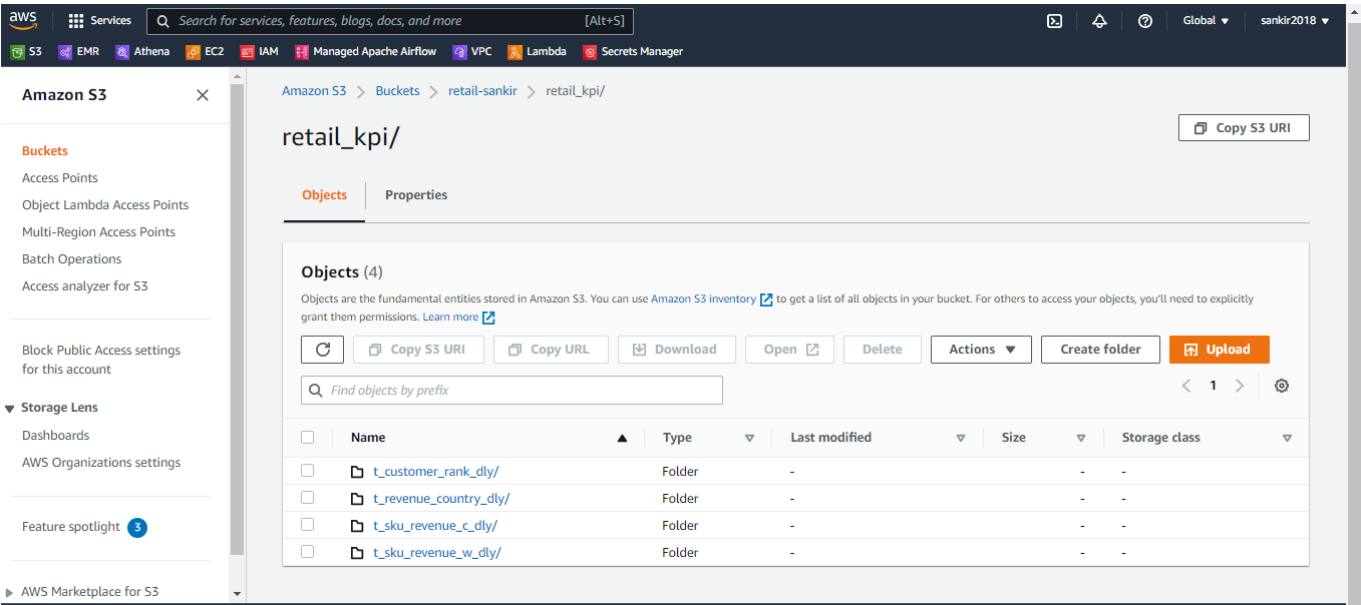


Here is the graph view of tasks in Airflow environment.



Transformed data in S3

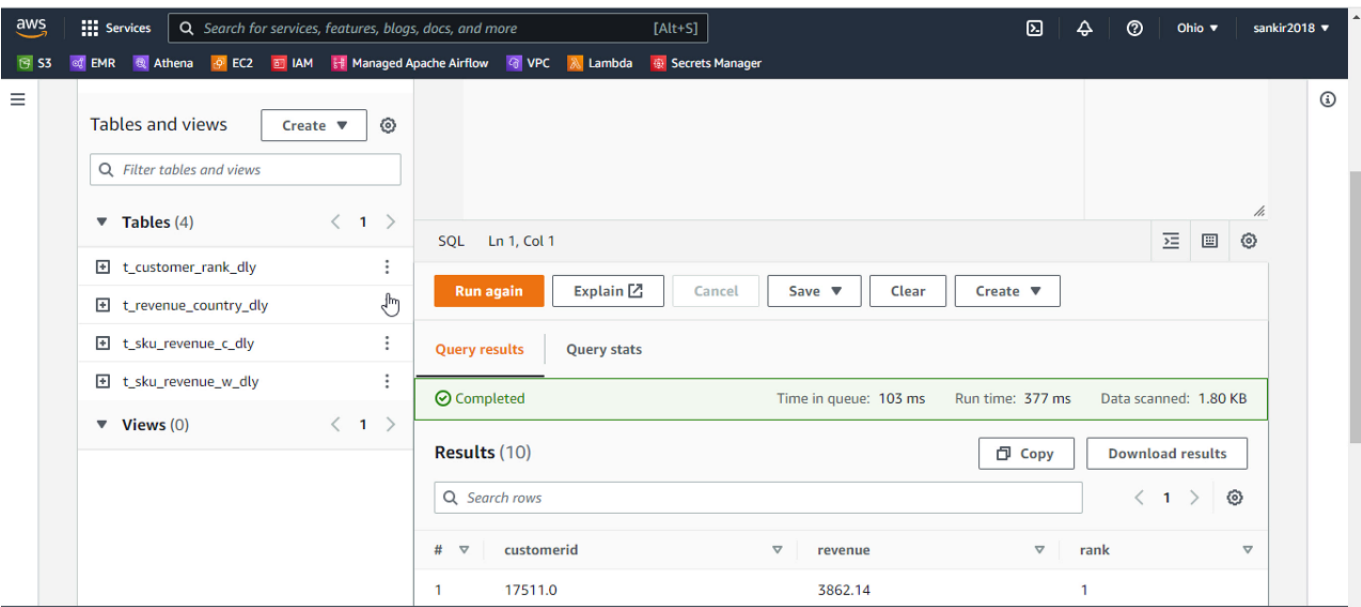
Transformed retail data is in AWS S3 output (KPI) tables



Query Transformed data in Athena

Transformed retail data is queried using AWS Athena.

Athena is a Serverless Interactive Query Service.



Technologies leveraged in POC

- Managed Apache Airflow (MWAA) with Airflow 2.0.2
 - IAM - Users, Groups, Roles and Responsibilities
 - VPC, Subnets, Route Tables, Elastic IPs, NAT Gateways
 - DAG - Directed Acyclic Graph
 - AWS EMR
 - AWS S3 Storage account
 - Apache Spark 2.4, Python 3.7
 - AWS Athena - Serverless Interactive Query Service.
-