Building data pipelines with Amazon EMR and MWAA

Leonardo Gomez
Senior Analytics Specialist

How we want our data pipelines

But data pipelines can be complex...

1 **Different Technologies**
- Spark
- Hive
- Presto
- Pig
- Many more

2 **Different Services**
- EMR
- Glue
- Athena
- Redshift
- RDS

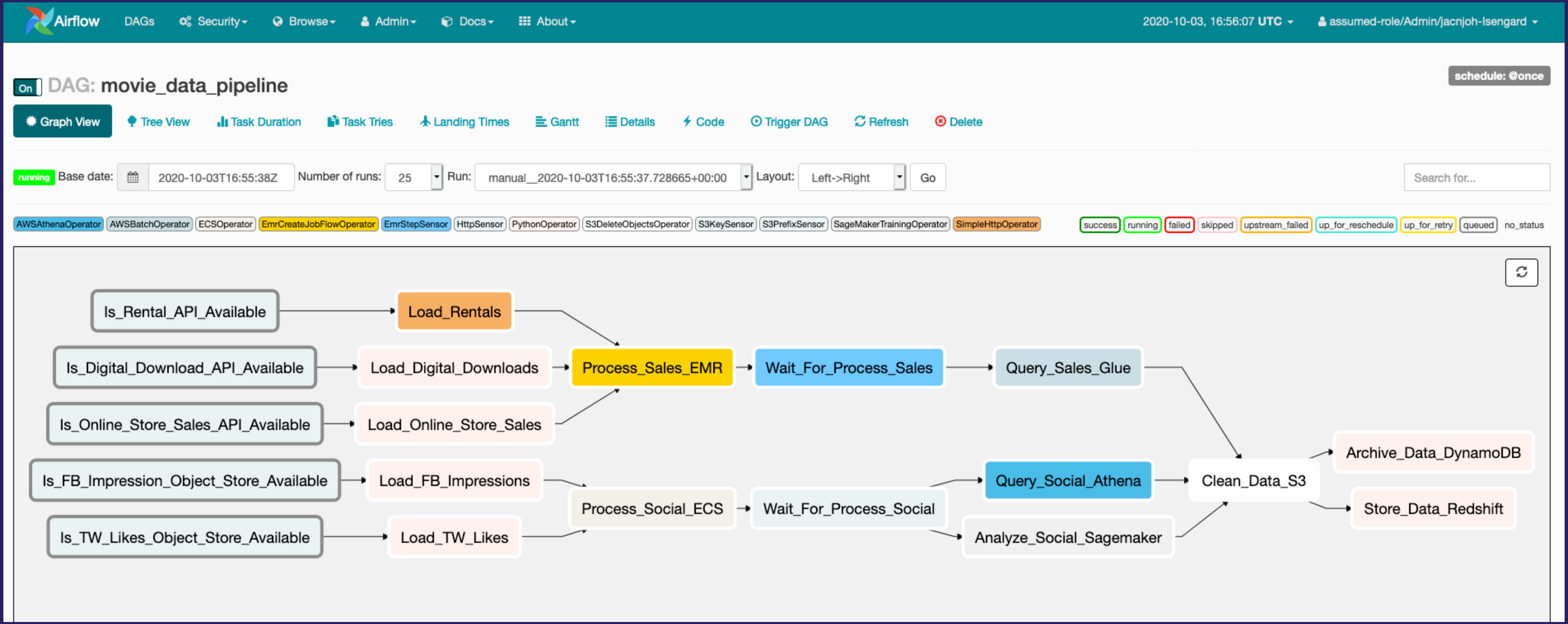
3 **Different Versions**
- Spark 2.x
- Spark 3.x

4 **Workflow Complexity**
- Dependencies
- Conditions
- Error handling
- Retry policies

Real world pipelines

We need orchestration

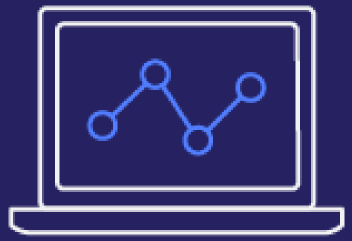
What is Apache Airflow?



Apache Airflow components

Scheduler

Worker



Web Server

Meta Database

Apache Airflow key concepts

DAG
Collections of tasks and describe how to run a workflow written in Python

Tasks
A Task defines a unit of work within a DAG; it is represented as a node in the DAG graph.

Operators
Atomic components in a DAG describing a single task in the pipeline.

Sensors
Special types of operators whose purpose is to wait on some external or internal trigger

Hooks
Provide a uniform interface to access external services like S3, MySQL, Hive, EMR, etc.

Scheduling
The DAGs and tasks can be run on demand or can be scheduled to be run at a certain frequency.

How does it work?

The workflows you build with Apache Airflow are called DAG, and each step of your workflow is called a task

When you execute your DAG, the workflow moves from one task to the next based on dependencies

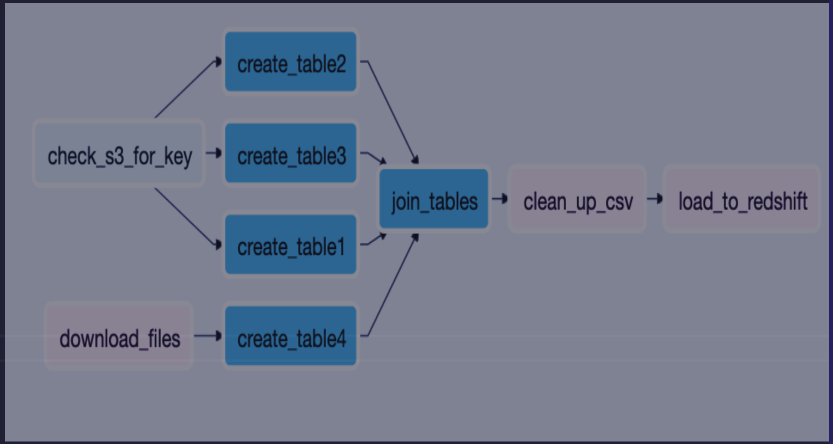
You can reuse components, easily edit the sequence of tasks, or swap out the code called by tasks as your needs change

Directed acyclic graph (DAG)

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.operators import HttpSensor, S3KeySensor
from airflow.contrib.operators.aws_athena_operator import AWSAthenaOperator
from airflow.utils.dates import days_ago
........
redshift_dbuser='awsuser'
redshift_table_name='movie_demo'
........

def download_zip():
    s3c = boto3.client('s3')
    indata = requests.get(download_http)
    n=0
    with zipfile.ZipFile(io.BytesIO(indata.content)) as z:
        zList=z.namelist()
........
with DAG(
    dag_id='movie-list-dag-v1.0',
    default_args=DEFAULT_ARGS,
    dagrun_timeout=timedelta(hours=2),
    start_date=days_ago(1),
    schedule_interval='* 10 * * *',
    tags=['athena','redshift'],
    catchup=False
) as Dag

    download_files = PythonOperator(
        task_id="download_files",
        python_callable=download_zip
    )
    create_table1 = AWSAthenaOperator(task_id="create_table1",query=create_athena_movie_table_query,
     database=athena_db, output_location='s3://'+s3_bucket_name+"/"+athena_results+'create_athena_movie_table')
.....
    download_files >> create_table4 >> join_tables
```



Challenges with self-managed Apache Airflow

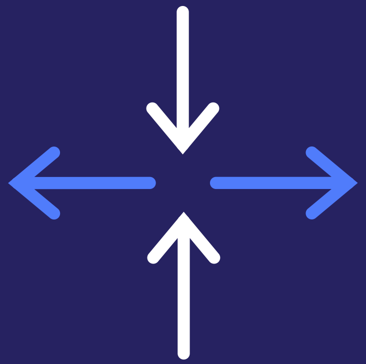
Setup | Scaling | Security | Upgrades | Maintenance

Solution

Amazon Managed Workflows for Apache Airflow (MWAA)

Setup

- Deploy Airflow Rapidly using AWS Console, AWS CLI, AWS API, or AWS CloudFormation
- Same Open-source Airflow

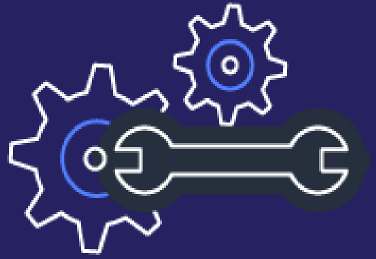


Scaling

- Seamless Worker Scaling
- Uses Celery Executor
- Amazon ECS on AWS Fargate

Security

- Integrated with AWS IAM
- VPC only or Public Airflow UI
- Workers and Scheduler run in customer VPC



Upgrades

- Maintenance windows for upgrades
- Snapshot and rollback in case of failure

Maintenance

- Monitoring with CloudWatch
- Multi-AZ
- Automatic restart on failure

How does Amazon MWAA work?

Create an MWAA Environment

Upload Airflow DAG (Directed Acyclic Graph) to Amazon S3

Access the Airflow UI

Amazon EMR

Big data analytics using open-source frameworks: Apache Spark, Presto, Trino, Hadoop, Hive, HBase & Flink

Differentiated performance for runtimes

Performance-optimized runtime for popular frameworks like Spark and Hive with 100% open-source API compatibility

Latest open-source features

New open-source features available within 30 days of release in open source

Best price-performance for big data analytics

Reduce cost using Amazon EC2 Spot, Amazon EMR managed scaling, and per-second billing

Self-service data science

Data science IDE with EMR Studio and deep integration with Amazon SageMaker Studio provides ability to use open-source UX and frameworks to build, visualize, and debug applications

Run workloads on Amazon EC2, Amazon EKS, or on premises

EMR provides flexibility to run big data workloads on EC2, EKS, and on premises with AWS Outposts

S3 data lake integration

Fine-grained access controls with AWS Lake Formation and Apache Ranger, and integrations with Apache HUDI to enable Amazon S3 data lake use cases

Amazon EMR deployment options

Amazon EMR on Amazon EC2

Choose instances that offer the best price performance for your workload

MWAA Supported

Amazon EMR on Amazon EKS

Automate provisioning, management, and scaling of Apache Spark jobs on Amazon EKS

MWAA Supported

Amazon EMR Serverless

Run applications using open source frameworks like Apache Spark, Hive, and Presto without having to configure, optimize, operate, or secure clusters

Amazon EMR on AWS Outposts

Set up, manage, and scale Amazon EMR in your on-premises environments, just as you would in the cloud

Cost-optimization options

Performance optimizations

- Runtime improvements
- Transactions in data lakes

Compute optimizations

- Graviton instances
- Spot Instances
- Instance fleets

Cluster management

- Managed scaling
- Cluster auto-termination

Containerization

Consolidate analytics and other workloads on Amazon EKS using Amazon EMR on Amazon EKS

EMR + Airflow allows

1 Different Technologies

- Spark
- Hive
- Presto
- Many more

2 Different Cost Options

- Spot Instances
- Instance fleets
- Instance groups

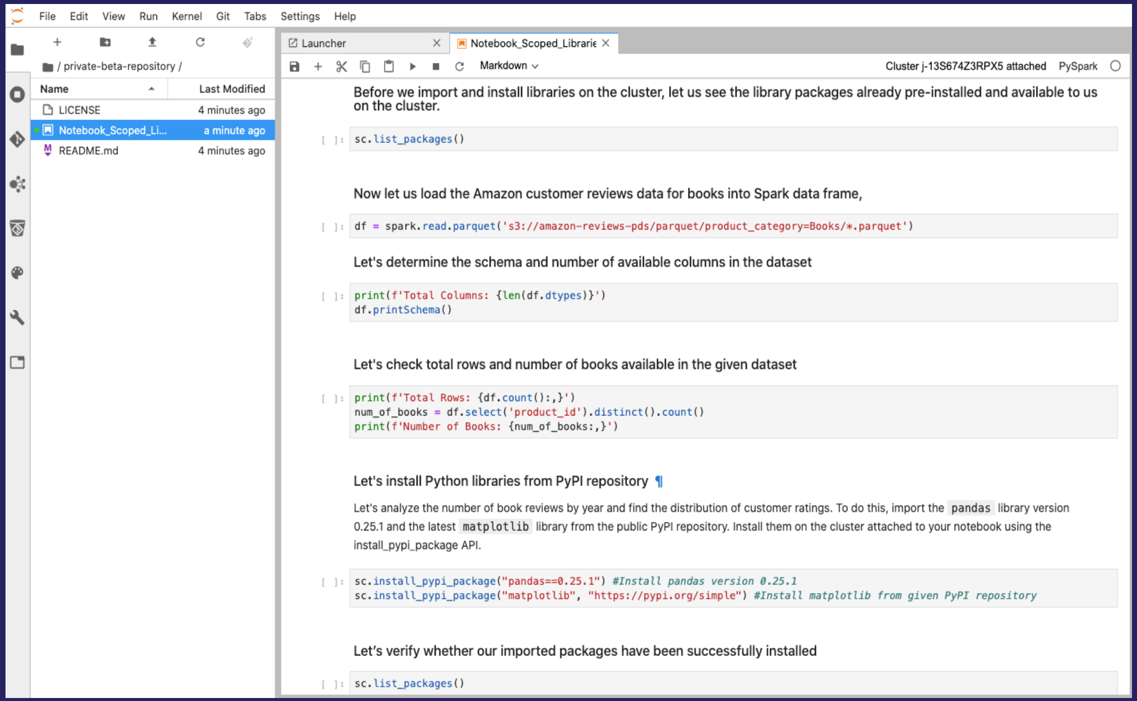
3 Different Versions

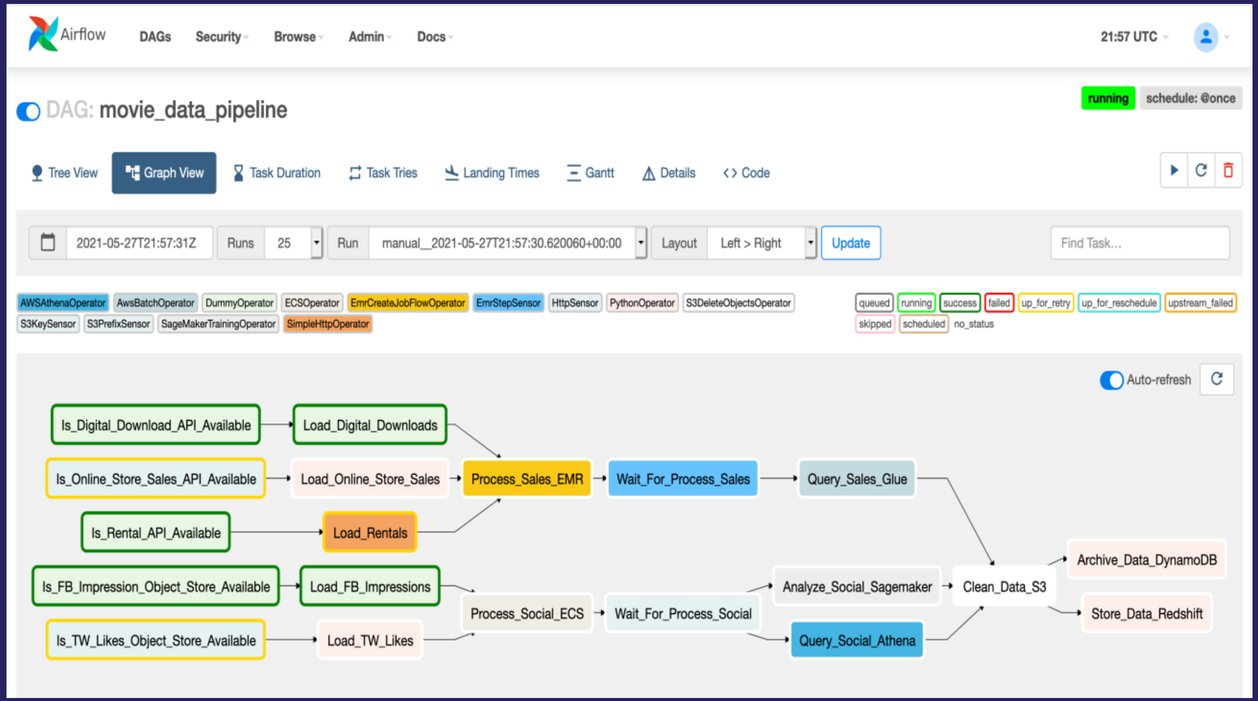
- EMR 5.x - Spark 2.x
- EMR 6.x - Spark 3.x

4 Workflow Flexibility

- EMR managed scaling
- EMR on EKS
- EMR on EC2
- EMR Serverless

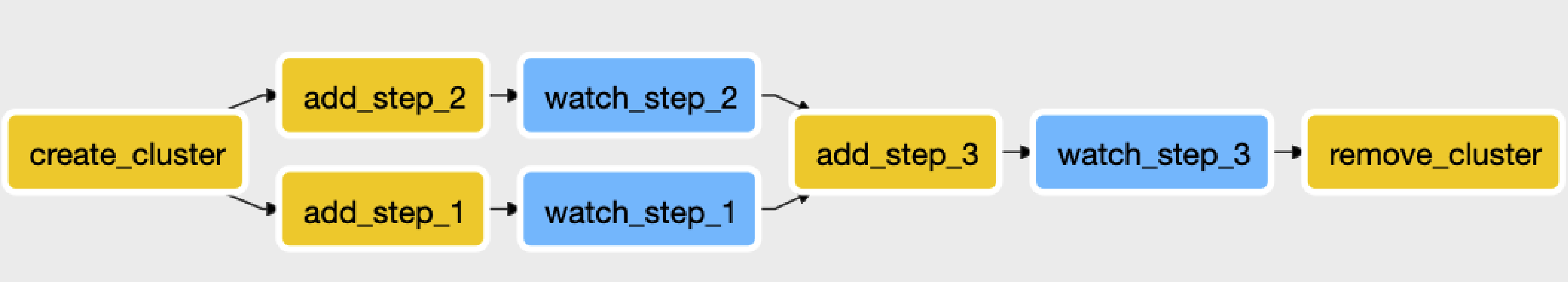
EMR Notebooks + Airflow





Airflow DAG to launch EMR

Example workflow:



- Airflow Operators
  - Create EMR with EmrCreateJobFlowOperator
  - Add steps to EMR with EmrAddStepsOperator
  - Watch progress of steps with EmrStepSensor
  - Terminate EMR with EmrTerminateJobFlowOperator

- Airflow Xcoms (Cross Communication)
  - Pass values between Airflow tasks

EmrCreateJobFlowOperator

```
JOB_FLOW_OVERRIDES = {
    "Name": "Data-Pipeline-" + execution_date,
    "ReleaseLabel": "emr-5.29.0",
    "LogUri": "s3://{}/logs/emr/".format(S3_BUCKET_NAME),
    "Instances": {
        "InstanceGroups": [
            {
                "Name": "Principal nodes",
                "Market": "ON_DEMAND",
                "InstanceRole": "MASTER",
                "InstanceType": "m5.xlarge",
                "InstanceCount": 1
            },
            {
                "Name": "worker nodes",
                "Market": "ON_DEMAND",
                "InstanceRole": "CORE",
                "InstanceType": "m5.xlarge",
                "InstanceCount": 2
            }
        ],
        "TerminationProtected": False,
        "KeepJobFlowAliveWhenNoSteps": True
    }
}

cluster_creator = EmrCreateJobFlowOperator(
    task_id='create_emr_cluster',
    job_flow_overrides=JOB_FLOW_OVERRIDES,
    aws_conn_id='aws_default',
    emr_conn_id='emr_default',
    dag=dag
)
```

EmrAddStepsOperator

```
SPARK_TEST_STEPS = [
   {
       'Name': 'setup - copy files',
       'ActionOnFailure': 'CANCEL_AND_WAIT',
       'HadoopJarStep': {
           'Jar': 'command-runner.jar',
           'Args': ['aws', 's3', 'cp', '--recursive', S3_URI, '/home/hadoop/']
       }
   },
   {
       'Name': 'Run Spark',
       'ActionOnFailure': 'CANCEL_AND_WAIT',
       'HadoopJarStep': {
           'Jar': 'command-runner.jar',
           'Args': ['spark-submit',
                    '/home/hadoop/nyc_aggregations.py',
                    's3://{}/data/transformed/green'.format(S3_BUCKET_NAME),
                    's3://{}/data/aggregated/green'.format(S3_BUCKET_NAME)]
       }
   }
]

step_adder = EmrAddStepsOperator(
    task_id='add_steps',
    job_flow_id="{{ task_instance.xcom_pull('create_emr_cluster', key='return_value') }}",
    aws_conn_id='aws_default',
    steps=SPARK_TEST_STEPS,
    dag=dag
)
```

EmrStepSensor

```
step_checker = EmrStepSensor(
    task_id='watch_step',
    job_flow_id="{{ task_instance.xcom_pull('create_emr_cluster', key='return_value') }}",
    step_id="{{ task_instance.xcom_pull('add_steps', key='return_value')[0] }}",
    aws_conn_id='aws_default',
    dag=dag
)
```

EmrTerminateJobFlowOperator

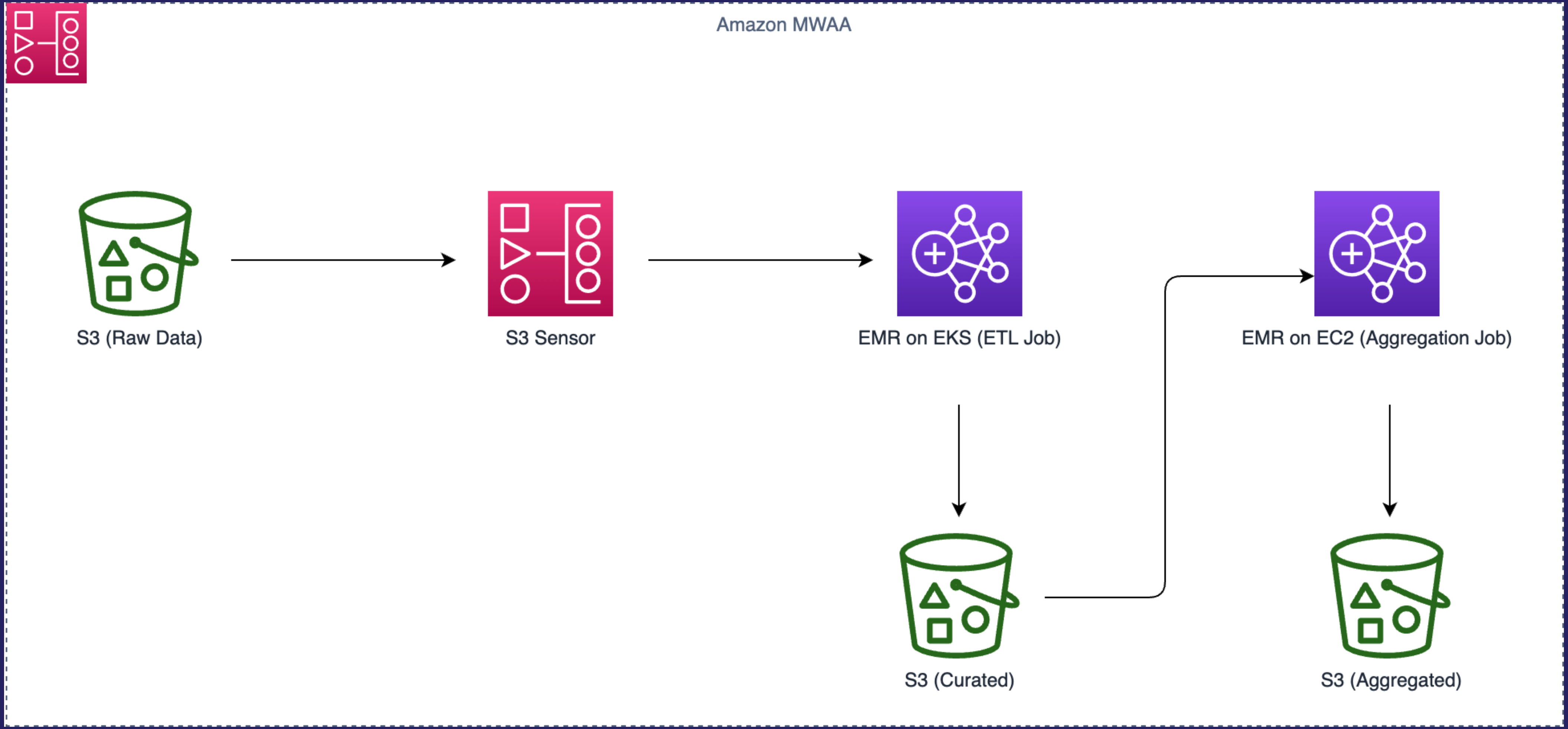
```
cluster_remover = EmrTerminateJobFlowOperator(
    task_id='remove_cluster',
    job_flow_id="{{ task_instance.xcom_pull('create_emr_cluster', key='return_value') }}",
    aws_conn_id='aws_default',
    dag=dag
)
```

EMRContainerOperator

```
eks_job_starter = EMRContainerOperator(
    task_id="emr_eks_job",
    virtual_cluster_id=VIRTUAL_CLUSTER_ID,
    execution_role_arn=JOB_ROLE_ARN,
    release_label="emr-6.3.0-latest",
    job_driver=JOB_DRIVER_ARG,
    name="data_aggregation.py",
    dag=dag
)
```

DEMO

Demo use case



Thank you!