* A sensor is just like a normal task in a DAG, but instead of performing work (like SparkSubmitOperator runs a job), it waits for something external to happen.
* It’s useful when your workflow depends on external triggers such as:
  + A file arriving in S3
  + Another DAG/task finishing
  + A table partition becoming available
  + A time window being reached
* Airflow provides many built-in sensors, e.g.:
  + S3KeySensor → waits for a file in S3.
  + FileSensor → waits for a local/remote file to appear.
  + ExternalTaskSensor → waits for a task in another DAG to finish.
  + SqlSensor → waits until a SQL query returns rows.
  + TimeSensor → waits until a specific time

1. Job1 → check if student.csv exists in S3 (poc-test)
2. Job2 → transform data and push to S3 (dummy-trans-output)
3. Job2 → Job3 transition → wait for SQS message
4. Job3 → run Spark job, output goes to S3 (spark-out)
5. Job3 → Job4 transition → wait for file in S3 (spark-out)
6. Job4 → load into SQL DB
7. Job5 → send notification

Airflow DAG Code

from datetime import datetime, timedelta

from airflow import DAG

from airflow.operators.python import PythonOperator

from airflow.providers.amazon.aws.sensors.s3 import S3KeySensor

from airflow.providers.amazon.aws.sensors.sqs import SqsSensor

import boto3

# -------------------------------

# Python task implementations

# -------------------------------

def transform\_and\_push():

# Simulated transformation

print("Transforming data...")

s3 = boto3.client("s3")

# Example: just copy file from poc-test to dummy-trans-output

copy\_source = {"Bucket": "poc-test", "Key": "student.csv"}

s3.copy\_object(

CopySource=copy\_source,

Bucket="dummy-trans-output",

Key="transformed\_student.csv"

)

print("File pushed to dummy-trans-output/transformed\_student.csv")

def run\_spark\_job():

# Simulated Spark job (replace with SparkSubmitOperator if needed)

print("Running Spark job on transformed data...")

s3 = boto3.client("s3")

s3.put\_object(

Bucket="spark-out",

Key="spark\_output.csv",

Body="id,name,score\n1,Sid,90\n2,Ray,85"

)

print("Spark job output written to spark-out/spark\_output.csv")

def load\_to\_sql():

# Example: write into SQL DB (replace with actual DB hook/connection)

print("Loading data from spark-out/spark\_output.csv into SQL DB...")

def notify\_completion():

print("✅ Pipeline finished successfully!")

# -------------------------------

# DAG Definition

# -------------------------------

default\_args = {

"owner": "sid",

"depends\_on\_past": False,

"email\_on\_failure": False,

"email\_on\_retry": False,

"retries": 1,

"retry\_delay": timedelta(minutes=2),

}

with DAG(

dag\_id="student\_pipeline\_with\_sqs\_and\_s3",

default\_args=default\_args,

start\_date=datetime(2025, 8, 1),

schedule\_interval=None,

catchup=False,

tags=["s3", "sqs", "spark", "pipeline"],

) as dag:

# Job1 → Check if student.csv exists in poc-test bucket

job1\_check\_file = S3KeySensor(

task\_id="job1\_check\_student\_csv",

bucket\_name="poc-test",

bucket\_key="student.csv",

aws\_conn\_id="aws\_default",

poke\_interval=30,

timeout=600,

)

# Job2 → Transform and push to dummy-trans-output

job2\_transform = PythonOperator(

task\_id="job2\_transform",

python\_callable=transform\_and\_push,

)

# Transition Job2 -> Job3 → Wait for SQS message

wait\_for\_sqs = SqsSensor(

task\_id="wait\_for\_sqs\_message",

sqs\_queue="https://sqs.us-east-1.amazonaws.com/123456789012/my-queue",

aws\_conn\_id="aws\_default",

poke\_interval=30,

timeout=900,

)

# Job3 → Spark Job (output to spark-out bucket)

job3\_spark = PythonOperator(

task\_id="job3\_spark\_job",

python\_callable=run\_spark\_job,

)

# Transition Job3 -> Job4 → Wait for file in spark-out

wait\_for\_s3\_output = S3KeySensor(

task\_id="wait\_for\_spark\_output",

bucket\_name="spark-out",

bucket\_key="spark\_output.csv",

aws\_conn\_id="aws\_default",

poke\_interval=60,

timeout=1800,

)

# Job4 → Load into SQL DB

job4\_load\_sql = PythonOperator(

task\_id="job4\_load\_sql",

python\_callable=load\_to\_sql,

)

# Job5 → Notify Completion

job5\_notify = PythonOperator(

task\_id="job5\_notify",

python\_callable=notify\_completion,

)

# -------------------------------

# Task Dependencies

# -------------------------------

job1\_check\_file >> job2\_transform >> wait\_for\_sqs >> job3\_spark >> wait\_for\_s3\_output >> job4\_load\_sql >> job5\_notify

transform\_and\_push

import boto3

import pandas as pd

from io import StringIO

def transform\_and\_push():

s3 = boto3.client("s3")

# Step 1: Download file from poc-test bucket

bucket\_in = "poc-test"

key\_in = "student.csv"

obj = s3.get\_object(Bucket=bucket\_in, Key=key\_in)

data = obj["Body"].read().decode("utf-8")

# Step 2: Load into pandas

df = pd.read\_csv(StringIO(data))

# Example transformation: uppercase names + add status column

df["name"] = df["name"].str.upper()

df["status"] = ["PASS" if score >= 40 else "FAIL" for score in df["score"]]

# Step 3: Save transformed data into memory

csv\_buffer = StringIO()

df.to\_csv(csv\_buffer, index=False)

# Step 4: Upload to dummy-trans-output bucket

bucket\_out = "dummy-trans-output"

key\_out = "transformed\_student.csv"

s3.put\_object(

Bucket=bucket\_out,

Key=key\_out,

Body=csv\_buffer.getvalue()

)

print(f"✅ Transformed file written to s3://{bucket\_out}/{key\_out}")

**student.csv in poc-test**

id,name,score

1,ram,85

2,shyam,35

3,geeta,70

**transformed\_student.csv in dummy-trans-output**

id,name,score,status

1,RAM,85,PASS

2,SHYAM,35,FAIL

3,GEETA,70,PASS

run\_spark\_job

import boto3

import pandas as pd

from io import StringIO

def run\_spark\_job():

s3 = boto3.client("s3")

# Step 1: Download transformed file from dummy-trans-output

bucket\_in = "dummy-trans-output"

key\_in = "transformed\_student.csv"

obj = s3.get\_object(Bucket=bucket\_in, Key=key\_in)

data = obj["Body"].read().decode("utf-8")

# Step 2: Load into pandas (simulate Spark)

df = pd.read\_csv(StringIO(data))

# Example "Spark-like" processing: add grade column

def get\_grade(score):

if score >= 80:

return "A"

elif score >= 60:

return "B"

elif score >= 40:

return "C"

else:

return "F"

df["grade"] = df["score"].apply(get\_grade)

# Step 3: Save result

csv\_buffer = StringIO()

df.to\_csv(csv\_buffer, index=False)

# Step 4: Upload to spark-out bucket

bucket\_out = "spark-out"

key\_out = "spark\_output.csv"

s3.put\_object(

Bucket=bucket\_out,

Key=key\_out,

Body=csv\_buffer.getvalue()

)

print(f"✅ Spark job simulated. Output written to s3://{bucket\_out}/{key\_out}")

**dummy-trans-output/transformed\_student.csv**

id,name,score,status

1,RAM,85,PASS

2,SHYAM,35,FAIL

3,GEETA,70,PASS

**spark-out/spark\_output.csv**

id,name,score,status,grade

1,RAM,85,PASS,A

2,SHYAM,35,FAIL,F

3,GEETA,70,PASS,B

load\_to\_sql

import boto3

import pandas as pd

from io import StringIO

from airflow.providers.postgres.hooks.postgres import PostgresHook

def load\_to\_sql():

s3 = boto3.client("s3")

# Step 1: Download spark output file from S3

bucket\_in = "spark-out"

key\_in = "spark\_output.csv"

obj = s3.get\_object(Bucket=bucket\_in, Key=key\_in)

data = obj["Body"].read().decode("utf-8")

# Step 2: Load into pandas

df = pd.read\_csv(StringIO(data))

# Step 3: Connect to Postgres

pg\_hook = PostgresHook(postgres\_conn\_id="my\_postgres") # defined in Airflow Connections

engine = pg\_hook.get\_sqlalchemy\_engine()

# Step 4: Write DataFrame to table

df.to\_sql("students\_processed", engine, if\_exists="replace", index=False)

print(f"✅ Loaded {len(df)} rows into SQL table students\_processed")

**students\_processed**

id,name,score,status,grade

1,RAM,85,PASS,A

2,SHYAM,35,FAIL,F

3,GEETA,70,PASS,B

notify\_completion

import boto3

def notify\_completion():

sns = boto3.client("sns")

topic\_arn = "arn:aws:sns:us-east-1:123456789012:student-pipeline-topic"

sns.publish(

TopicArn=topic\_arn,

Message="✅ Student Pipeline finished successfully!",

Subject="Airflow Pipeline Success"

)

print("📢 SNS notification sent")