**Sensor\_pipeline\_dag.py**

from datetime import datetime, timedelta

from airflow import DAG

from airflow.operators.python import PythonOperator

from airflow.operators.dummy import DummyOperator

from airflow.models import Variable

from airflow.utils.task\_group import TaskGroup

# Default arguments for the DAG

default\_args = {

'owner': 'data\_engineering',

'depends\_on\_past': False,

'start\_date': datetime(2023, 1, 1),

'email': ['data-team@example.com'],

'email\_on\_failure': True,

'email\_on\_retry': False,

'retries': 3,

'retry\_delay': timedelta(minutes=5),

'execution\_timeout': timedelta(hours=2)

}

# Function to dynamically fetch pipeline configuration from Airflow Variables

def get\_pipeline\_config(\*\*context):

base\_config = {

'bucket\_name': Variable.get("SENSOR\_BUCKET"),

's3\_directory': f"sensor\_data/{context['ds\_nodash']}/",

'input\_files': Variable.get("SENSOR\_INPUT\_FILES", deserialize\_json=True),

'sensor\_patterns': Variable.get("SENSOR\_PATTERNS", deserialize\_json=True),

'write\_mode': 'append' if context['execution\_date'].day == 1 else 'overwrite',

'lookback\_days': int(Variable.get("LOOKBACK\_DAYS", default\_var=30)),

'jdbc\_fetch\_size': int(Variable.get("JDBC\_FETCH\_SIZE", default\_var=10000))

}

if Variable.get("USE\_AWS\_KEYS", default\_var=False):

base\_config.update({

'aws\_access\_key': Variable.get("AWS\_ACCESS\_KEY\_ID"),

'aws\_secret\_key': Variable.get("AWS\_SECRET\_ACCESS\_KEY")

})

context['ti'].xcom\_push(key='pipeline\_config', value=base\_config)

# Function to execute the PySpark pipeline

def execute\_pipeline(\*\*context):

from your\_package.pipeline import main\_with\_config

config = context['ti'].xcom\_pull(task\_ids='setup.get\_config', key='pipeline\_config')

main\_with\_config(config)

# Function to validate the output in S3

def validate\_output(\*\*context):

import boto3

s3 = boto3.client('s3')

bucket = Variable.get("SENSOR\_BUCKET")

prefix = f"sensor\_data/{context['ds\_nodash']}/"

objects = s3.list\_objects\_v2(Bucket=bucket, Prefix=prefix)

if not objects.get('Contents'):

raise ValueError(f"No output files found in s3://{bucket}/{prefix}")

# Define the DAG

with DAG(

'sensor\_data\_pipeline',

default\_args=default\_args,

schedule\_interval='0 2 \* \* \*',

catchup=False,

max\_active\_runs=1,

tags=['sensor', 'pyspark']

) as dag:

start = DummyOperator(task\_id='start')

end = DummyOperator(task\_id='end')

with TaskGroup('setup') as setup\_group:

get\_config = PythonOperator(

task\_id='get\_config',

python\_callable=get\_pipeline\_config

)

check\_resources = PythonOperator(

task\_id='check\_resources',

python\_callable=lambda: print("Resource check passed")

)

run\_pipeline = PythonOperator(

task\_id='execute\_pipeline',

python\_callable=execute\_pipeline,

execution\_timeout=timedelta(hours=1)

)

with TaskGroup('validation') as validation\_group:

validate\_data = PythonOperator(

task\_id='validate\_output',

python\_callable=validate\_output

)

log\_results = PythonOperator(

task\_id='log\_validation',

python\_callable=lambda: print("Validation complete")

)

# Define the task dependencies

start >> setup\_group >> run\_pipeline >> validation\_group >> end

**Purpose:** This DAG runs the PySpark sensor data pipeline daily at 2 AM.

**Key Features:**

* Dynamically loads config from Airflow Variables.
* Uses Task Groups (setup, validation) for clarity.
* Handles retries, execution timeout, and email alerts.
* Validates that output is actually written to S3.

**Main Tasks:**

* get\_config: Extracts config using Airflow Variables and pushes to XCom.
* check\_resources: Placeholder for resource availability checks.
* execute\_pipeline: Calls the PySpark pipeline main entrypoint.
* validate\_output: Verifies files exist in the S3 output directory.
* log\_validation: Logs a success message post-validation.

**Config\_manager\_dag.py**

from airflow import DAG

from airflow.operators.python import PythonOperator

from airflow.models import Variable

from datetime import datetime

# Function to update sensor patterns from external service

def update\_sensor\_patterns():

import requests

response = requests.get('https://config-service/sensor-patterns')

patterns = response.json()['patterns']

Variable.set("SENSOR\_PATTERNS", patterns, serialize\_json=True)

# Function to refresh DB credentials from AWS Secrets Manager

def refresh\_db\_credentials():

import boto3

client = boto3.client('secretsmanager')

secret = client.get\_secret\_value(SecretId='prod/db\_credentials')

Variable.set("DB\_CREDENTIALS", secret['SecretString'])

# Define the configuration management DAG

with DAG(

'pipeline\_config\_manager',

schedule\_interval='@weekly',

start\_date=datetime(2023, 1, 1),

catchup=False,

tags=['configuration']

) as dag:

update\_patterns = PythonOperator(

task\_id='update\_sensor\_patterns',

python\_callable=update\_sensor\_patterns

)

rotate\_credentials = PythonOperator(

task\_id='rotate\_db\_credentials',

python\_callable=refresh\_db\_credentials

)

update\_patterns >> rotate\_credentials

**Purpose:** Runs weekly to ensure your config is current and secure.

**Key Features:**

* Fetches new regex patterns from a config service.
* Refreshes credentials from AWS Secrets Manager.

**Main Tasks:**

* update\_sensor\_patterns: Pulls new regex patterns for sensors.
* rotate\_db\_credentials: Syncs database credentials into Airflow Variables.