Sensor_pipeline_dag.py

```python
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.