Python on Airflow pipelines

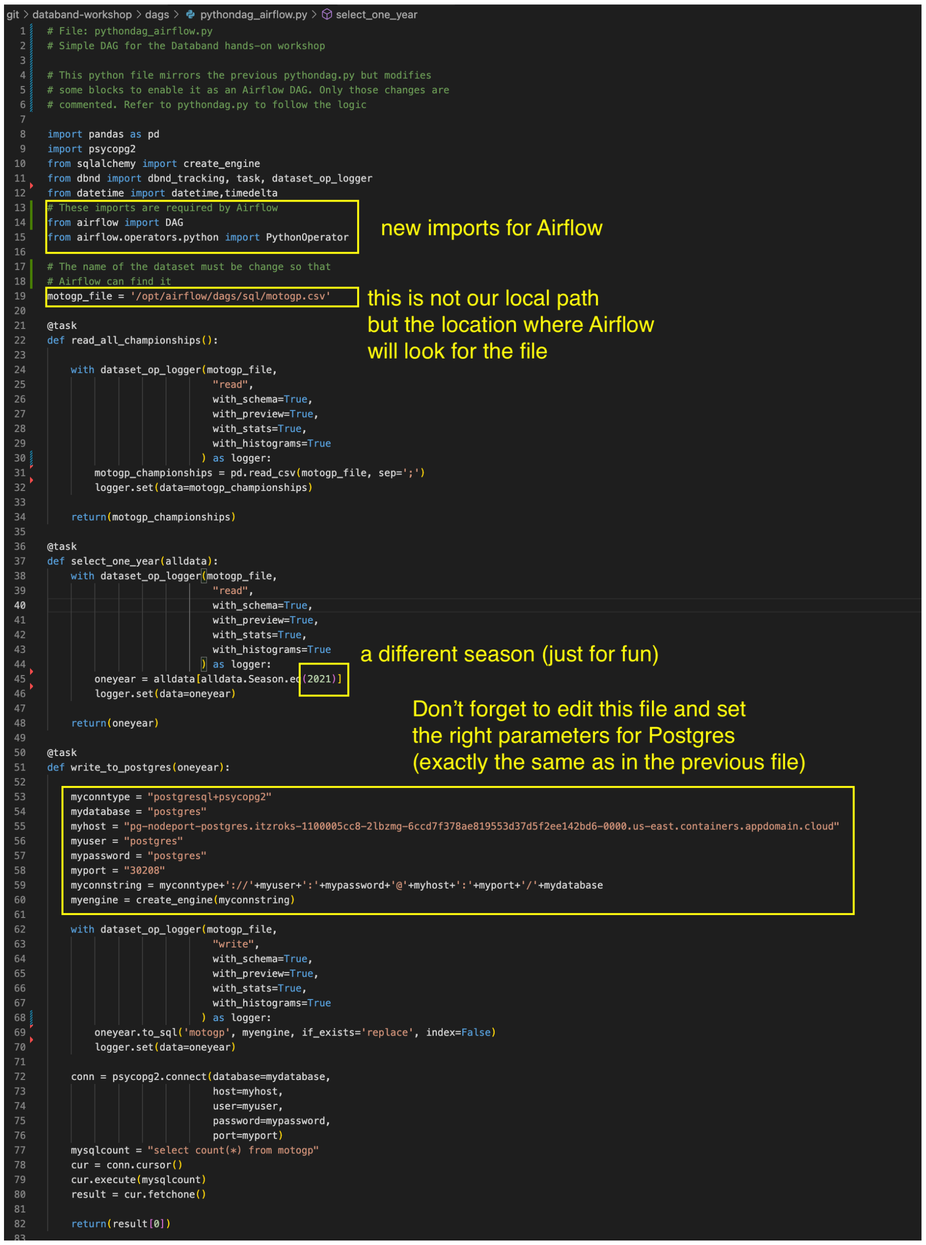
As we want to run pipelines many times to collect historical data, we need an scheduling mechanism. This is precisly what Airflow will do for us.

1. Add Airflow code to the pipeline

We will refactor the code of the previous section to run exactly the same pipeline as an Airflow DAG that can be scheduled for a few days to collect historical information.

1.1 Review the changes in the code

The necessary changes for enabling the python pipeline to run as an Airflow DAG are shown in the next cell



git > databand-workshop > dags > pythondag_airflow.py > select_one_year

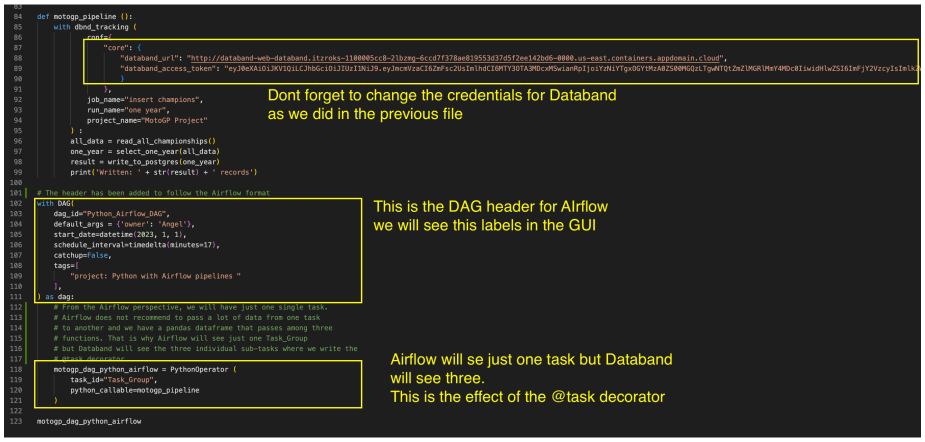
```python
# File: pythondag_airflow.py
# Simple DAG for the Databand hands-on workshop

# This python file mirrors the previous pythondag.py but modifies
# some blocks to enable it as an Airflow DAG. Only those changes are
# commented. Refer to pythondag.py to follow the logic

import pandas as pd
import psycopg2
from sqlalchemy import create_engine
from dbnd import dbnd_tracking, task, dataset_op_logger
from datetime import datetime,timedelta
# These imports are required by Airflow
from airflow import DAG
from airflow.operators.python import PythonOperator

# The name of the dataset must be change so that
# Airflow can find it
motogp_file = '/opt/airflow/dags/sql/motogp.csv'

@task
def read_all_championships():

    with dataset_op_logger(motogp_file,
                           "read",
                           with_schema=True,
                           with_preview=True,
                           with_stats=True,
                           with_histograms=True
                          ) as logger:
        motogp_championships = pd.read_csv(motogp_file, sep=';')
        logger.set(data=motogp_championships)

    return(motogp_championships)

@task
def select_one_year(alldata):
    with dataset_op_logger(motogp_file,
                           "read",
                           with_schema=True,
                           with_preview=True,
                           with_stats=True,
                           with_histograms=True
                          ) as logger:
        oneyear = alldata[alldata.Season.eq(2021)]
        logger.set(data=oneyear)

    return(oneyear)

@task
def write_to_postgres(oneyear):

    myconntype = "postgresql+psycopg2"
    mydatabase = "postgres"
    myhost = "pg-nodeport-postgres.itzroks-1100005cc8-2lbzmg-6ccd7f378ae819553d37d5f2ee142bd6-0000.us-east.containers.appdomain.cloud"
    myuser = "postgres"
    mypassword = "postgres"
    myport = "30208"
    myconnstring = myconntype+'://'+myuser+':'+mypassword+'@'+myhost+':'+myport+'/'+mydatabase
    myengine = create_engine(myconnstring)

    with dataset_op_logger(motogp_file,
                           "write",
                           with_schema=True,
                           with_preview=True,
                           with_stats=True,
                           with_histograms=True
                          ) as logger:
        oneyear.to_sql('motogp', myengine, if_exists='replace', index=False)
        logger.set(data=oneyear)

    conn = psycopg2.connect(database=mydatabase,
                           host=myhost,
                           user=myuser,
                           password=mypassword,
                           port=myport)
    mysqlcount = "select count(*) from motogp"
    cur = conn.cursor()
    cur.execute(mysqlcount)
    result = cur.fetchone()

    return(result[0])
```

new imports for Airflow

this is not our local path but the location where Airflow will look for the file

a different season (just for fun)

Don’t forget to edit this file and set the right parameters for Postgres (exactly the same as in the previous file)



```python
def motogp_pipeline ():
    with dbnd_tracking (
            conf={
                "core": {
                    "databand_url": "http://databand-web-databand.itzroks-1100005cc8-2lbzmg-6ccd7f378ae819553d37d5f2ee142bd6-0000.us-east.containers.appdomain.cloud",
                    "databand_access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTY3OTA3MDcxMSwianRpIjoiYzNiYTgxOGYtMzA0ZS00MGQzLTgwNTQtZmZlMGRlMmY4MDc0IiwidHlwZSI6ImFjY2VzcyIsImlk
                    }
                },
            job_name="insert champions",
            run_name="one year",
            project_name="MotoGP Project"
        ) :
        all_data = read_all_championships()
        one_year = select_one_year(all_data)
        result = write_to_postgres(one_year)
        print('Written: ' + str(result) + ' records')

# The header has been added to follow the Airflow format
with DAG(
    dag_id="Python_Airflow_DAG",
    default_args = {'owner': 'Angel'},
    start_date=datetime(2023, 1, 1),
    schedule_interval=timedelta(minutes=17),
    catchup=False,
    tags=[
        "project: Python with Airflow pipelines "
    ],
) as dag:
    # From the Airflow perspective, we will have just one single task.
    # Airflow does not recommend to pass a lot of data from one task
    # to another and we have a pandas dataframe that passes among three
    # functions. That is why Airflow will see just one Task_Group
    # but Databand will see the three individual sub-tasks where we write the
    # @task decorator
    motogp_dag_python_airflow = PythonOperator (
        task_id="Task_Group",
        python_callable=motogp_pipeline
    )

motogp_dag_python_airflow
```

Dont forget to change the credentials for Databand
as we did in the previous file
This is the DAG header for AIrflow
we will see this labels in the GUI
Airflow will se just one task but Databand
will see three.
This is the effect of the @task decorator

As you can see the fundamental changes are locatedat the bottom of the file where we write a new header for Airflow

1.2. Edit the file `pythondag_airflow.py` to include Postgres and Databand security parameters

All changes for Airflow are already done in `pythondag_airflow.py` but we need to enter the Postgres and Databand credentials for your particular environment. Please follow the same instructions as shown in the previous chapter under the paragraph `1.3.` No more changes are necessary.

1.3. Transfer `pythondag_airflow.py` to Airflow

As we modified the file, we need to transfer it to Airflow to be registered as a DAG. We begin with the usual login to the cluster

```
# Replace the command with your own one inside the single quotes and run the cell
# Example OC_LOGIN_COMMAND='oc login --token=sha256~3bR5KXgwiUoaQiph2_kIXCDQnVfm_HQy3YwU2m-
OC_LOGIN_COMMAND='oc login --token=sha256~6Xs6va20JZ2CFhS61HN6bpQC2z075XZbhIJt3tZ8L6w --serv
$OC_LOGIN_COMMAND
oc project airflow
```

We need to verify that the file `pythondag_airflow.py` is located in the DAGs directory.

```
# you may need to modify the cd command to place yourself in the DAGs directory
pwd
cd ../dags
ls -l
```

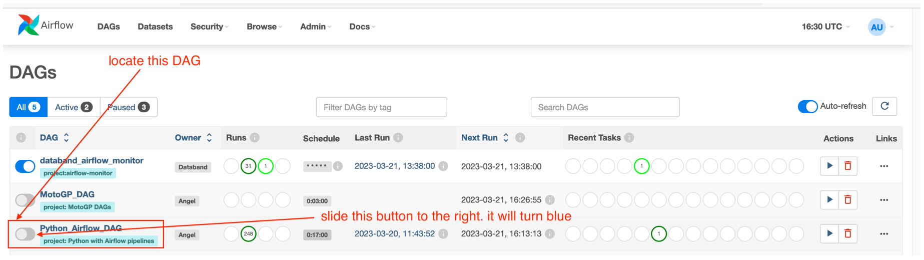
Look for a `pythondag_airflow.py` like this:

And then you can run this cell to transfer the file:

```
# Run this cell to copy the file to the openshift cluster
oc cp pythondag_airflow.py airflow-worker-0:dags/
```

1.4. Enable the run on Airflow

Once the file is transfered to Airflow you may need to wait about 5 minutes until the the DAG is visible. Then, you need to activate it:

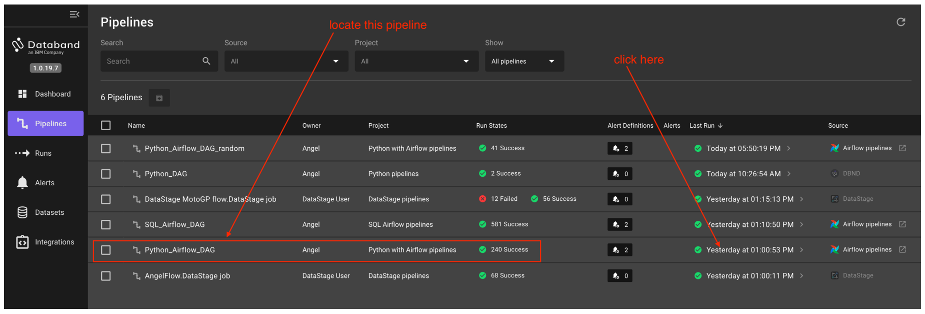


DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
databand_airflow_monitor project:airflow-monitor	Databand	31 1	*****	2023-03-21, 13:38:00	2023-03-21, 13:38:00	1		...
MotoGP_DAG project: MotoGP DAGs	Angel		0:03:00		2023-03-21, 16:26:55			...
Python_Airflow_DAG project: Python with Airflow pipelines	Angel	248	0:17:00	2023-03-20, 11:43:52	2023-03-21, 16:13:13	1		...

Now, the DAG is activated and will run every 17 minutes. Leave it running for a few days if you whish to see historical data or go to the next section where we see how it will look like.

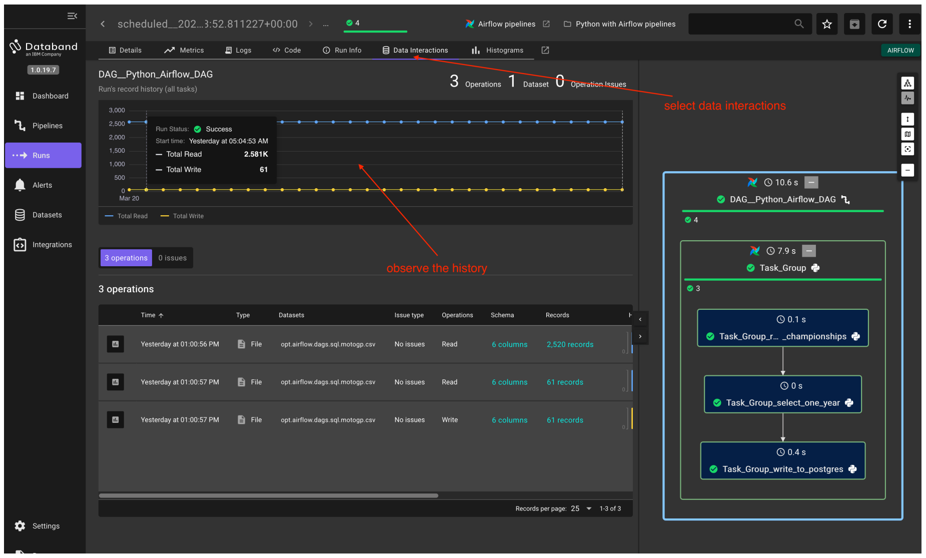
2. Display performance data with Databand

The new created pipeline can will be shown as `Python_Airflow_DAG` as it is hardcoded in the header section of the python code



Name	Owner	Project	Run States	Alert Definitions	Alerts	Last Run	Source
Python_Airflow_DAG_random	Angel	Python with Airflow pipelines	41 Success	2		Today at 05:50:19 PM	Airflow pipelines
Python_DAG	Angel	Python pipelines	2 Success	0		Today at 10:26:54 AM	DBND
DataStage MotoGP flow.DataStage job	DataStage User	DataStage pipelines	12 Failed 56 Success	0		Yesterday at 01:15:13 PM	DataStage
SQL_Airflow_DAG	Angel	SQL Airflow pipelines	581 Success	2		Yesterday at 01:10:50 PM	Airflow pipelines
Python_Airflow_DAG	Angel	Python with Airflow pipelines	240 Success	2		Yesterday at 01:00:53 PM	Airflow pipelines
AngelFlow.DataStage job	DataStage User	DataStage pipelines	68 Success	0		Yesterday at 01:00:11 PM	DataStage

The pipeline ran 240 times so far and the historical data can be shown like follows:



Run Status: Success
Start time: Yesterday at 05:04:53 AM
Total Read 2.581K
Total Write 61

3 operations | 0 issues

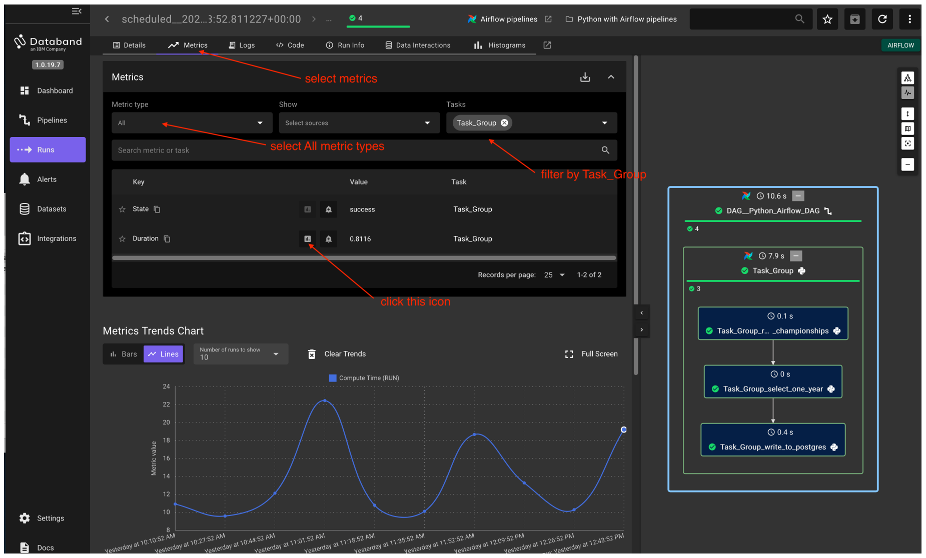
3 operations

Time	Type	Datasets	Issue type	Operations	Schema	Records
Yesterday at 01:00:56 PM	File	opt.airflow.dags.sql.motogp.csv	No issues	Read	6 columns	2,520 records
Yesterday at 01:00:57 PM	File	opt.airflow.dags.sql.motogp.csv	No issues	Read	6 columns	61 records
Yesterday at 01:00:57 PM	File	opt.airflow.dags.sql.motogp.csv	No issues	Write	6 columns	61 records

Records per page: 25 1-3 of 3

Unfortunately, when things run well, all graphics and trends look like very boring. In our case the number of rows written and read is the same during the whole history.

However, there are ways to see more exciting curves. Just proceed as instructed in the following picture:



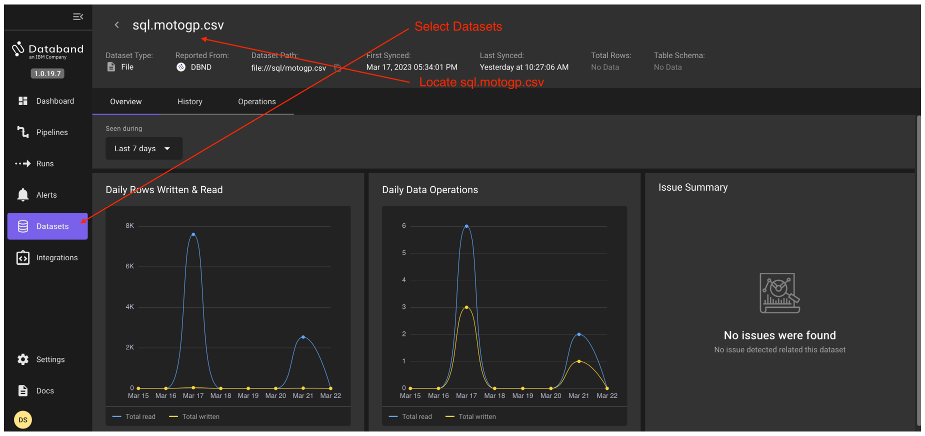
Metric type: All
Show: Select sources
Tasks: Task_Group

Key	Value	Task
State	success	Task_Group
Duration	0.8116	Task_Group

Records per page: 25 1-2 of 2

Number of runs to show: 10

Indeed, there are variations in the elapsed runtime caused by the concurrency of several jobs while the performance data was collected. You can now switch to the Datasets view and see the cumulated traffic of records in the last days:



sql.motogp.csv

Dataset Type: File
Reported From: DBND
Dataset Path: file:///sql/motogp.csv
First Synced: Mar 17, 2023 05:34:01 PM
Last Synced: Yesterday at 10:27:06 AM
Total Rows: No Data
Table Schema: No Data

Seen during: Last 7 days

No issues were found
No issue detected related this dataset

Next Section: DataStage pipelines
Previous Section: Python pipelines

Return to main