

# Python pipelines

## 1. Run the sample python pipeline

In this section, a simple python program will read a small csv file and it will load it in a Postgres database. No Airflow and no other tool is needed for this program to run properly. It can be executed on your local laptop if you have a python environment and installed the necessary packages. Additionally, we will need to capture some parameters of our deployment on OpenShift and modify four lines of the code.

### 1.1. Install python packages

We need the following python packages in the sample pipeline:

- **pandas**: to create a dataframe where the csv data will reside
- **psycopg2**: to access Postgres
- **sqlalchemy**: to move a pandas dataframe into Postgres without warnings or errors
- **dbnd**: to collect performance data that will be pushed to the Databand system

Run the next cell to install these prerequisites:

```
# This cell installs the packages needed to run pythondag.py
pip install psycopg2-binary
pip install pandas
pip install sqlalchemy
pip install dbnd
# This last installation may be necessary on your particular environment
# if you see errors in the execution of the python program
# pip install more-itertools==9.1.0
```

### 1.2. Retrieve deployment parameters for the python pipeline

We need the following parameters for our deployment:

- The IP Address and the port of the Postgres service (route and nodeport)
- The IP Address of the Databand service (route)
- A personal API key to access the Databand service

Let's begin with the Postgres parameters. We can retrieve them from the command line or by accessing the OpenShift interface. We will expose both ways in order to verify that produce the same results.

As usual, we start by logging to the cluster and selecting the Postgres project

```
# 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-l
```

```
OC_LOGIN_COMMAND='oc login --token=sha256~6Xs6va20JZ2CFhS61HN6bpQC2z075XZbhIJt3tZ8L6w --server=https://c103-e.us-south.containers.cloud.ibm.com:3232'
$OC_LOGIN_COMMAND
```

Then, we retrieve the external hostname (or the route or the IP Address) and the port of the Postgres database in the cluster.

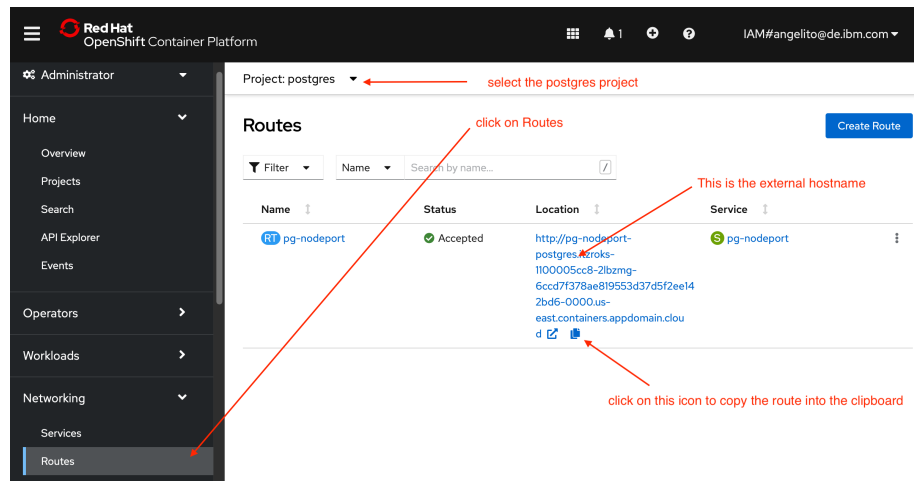
*# Run this cell to get the hostname and the port of Postgres*

```
oc project postgres
externalhostnamepostgres=$(oc get routes | grep nodeport | awk '{print $2}')
externalportpostgres=$(oc get svc | grep NodePort | awk '{print $5}' | cut -f2 -d':' | cut -f1 -d'/')
echo ----- The following two lines are a hostname '(or IP Address or route)' and a port number
echo $externalhostnamepostgres
echo $externalportpostgres
echo ----- paste these values in the part of the code that defines the Postgres connection
```

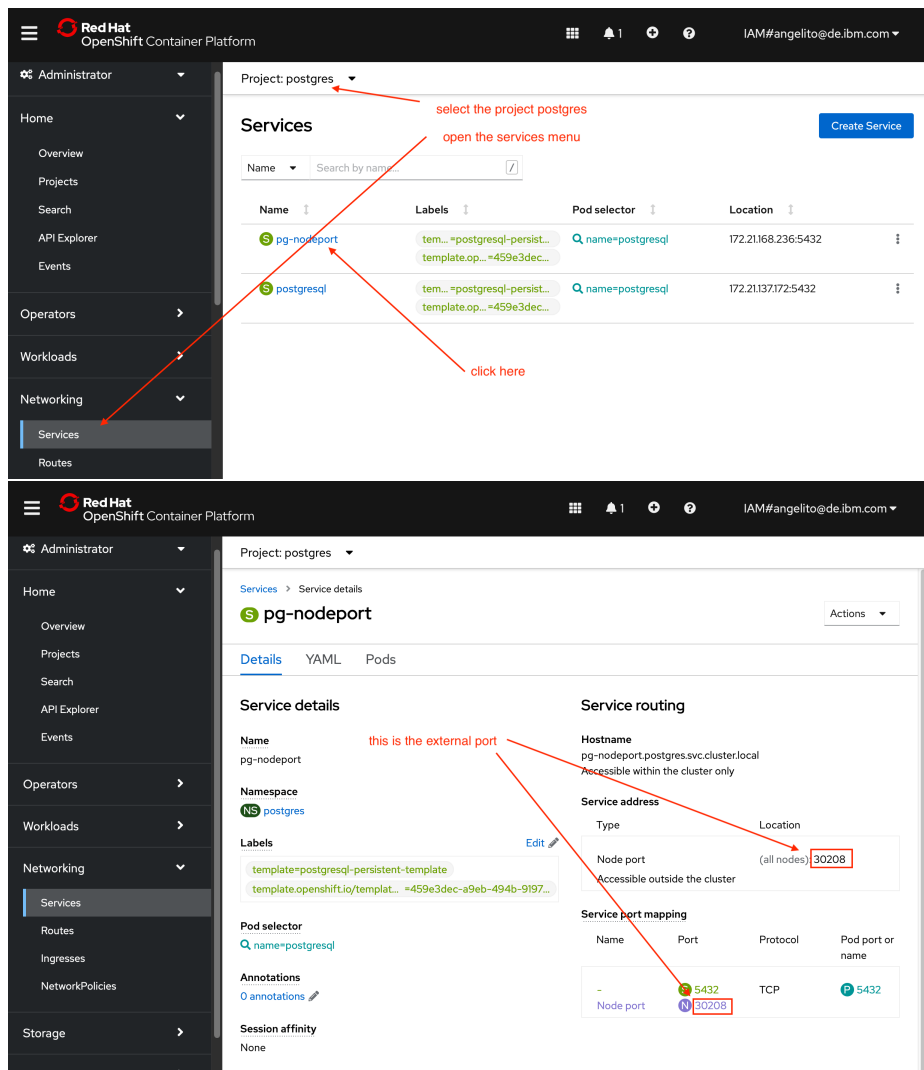
Now using project "postgres" on server "https://c103-e.us-south.containers.cloud.ibm.com:3232" ----- The following two lines are a hostname (or IP Address or route) and a port number ----- pg-nodeport-postgres.itzroks-1100005cc8-myv0re-4b4a324f027aea19c5cbc0c3275c4656-0000.us-south.containers.appdomain.cloud:3232 ----- paste these values in the part of the code that defines the Postgres connection as in

Notice that this data is different from the internal parameters we needed to retrieve in the Section 7. At that time, we wanted to bind Airflow with Postgres, which can be done inside the cluster and the external names were not used.

You may want to verify that these parameters match the the values displayed in the OpenShift console. The external hostname of Postgres can be found as follows:



Two more steps are necessary in order to see the external port:



The next thing is to get the connection parameters for Databand. The hostname (the route in OpenShift's dialect)

*# Run this cell to get the hostname and the port of Databand*

oc project databand

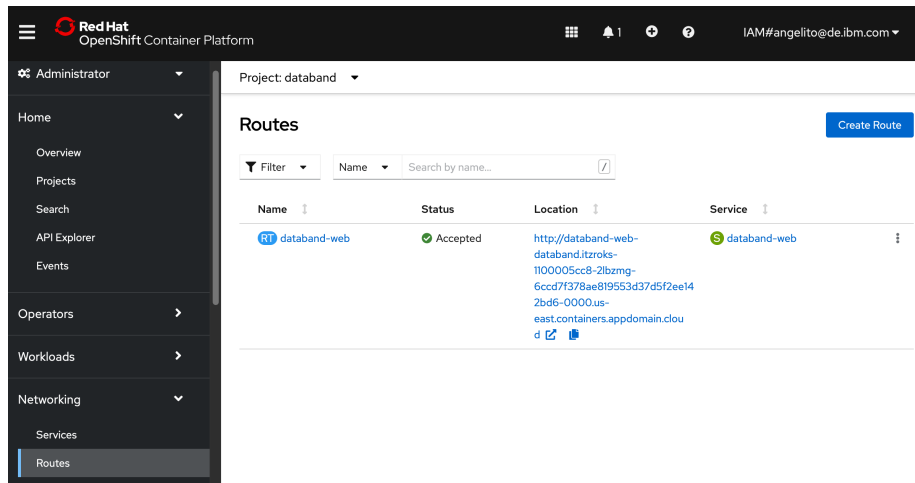
externalhostnamedataband=\$(oc get routes | grep databand | awk '{print \$2}')

echo ----- The following line is the external hostname of databand -----

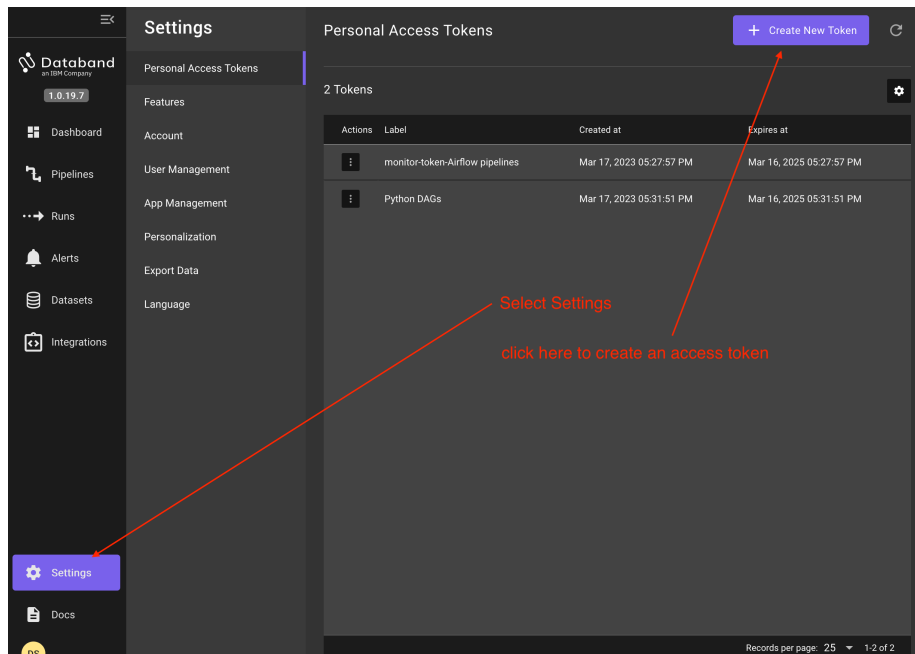
echo \$externalhostnamedataband

echo ----- paste this value in the part of the code that defines the Databand connection as

If you want to verify the name, you can see it in the Openshift console as well



Finally, we need to generate an API key in the Databand user interface.



Create new token

×

1

Create Token

2

Copy Token

This token will authenticate your environment for transmitting logs, metrics, and other metadata.

Label

token for workshop

Token's max lifespan

2 years ▾

Set the expiry date for the token

click here

Save



### 1.3. Edit the pythondag.py file

Now, you need to edit the `pythondag.py` file. Locate the portion of the code as shown on the picture and replace the values between the double quotes with the parameters collected above. Ensure that the variable `myhost` does not begin with `http://...`

```

68 @task
69 def write_to_postgres(oneyear):
70
71     # Build the connection
72     myconntype = "postgresql+psycopg2"
73     mydatabase = "postgres"
74     myhost = "pg-nodeport-postgres.itzroks-1100005cc8-2lbzm-gccdf7378ae819553d7d5f2ee142b6-0000.us-east.containers.appdomain.v0.amazonaws.com"
75     myuser = "postgres"
76     mypassword = "postgres"
77     myport = "30208"
78     myconnstring = myconntype+'://'+myuser+':'+mypassword+'@'+myhost+':'+myport+'/'+mydatabase
79     myengine = create_engine(myconnstring)
80

```

There is no need to modify other parameters in this section if you didn't change the database name, userid or password.

In the same file, some lines of code down, the Databand section must be edited analogously but, this time, please ensure that the route of Databand begins with `http://...`

```

111 # Function to define the connection to Databand and run the pipeline
112 def motogp_pipeline ():
113     with dbnd_tracking (
114         conf={
115             "core": {
116                 "databand_url": "http://databand-web-databand.itzroks-1l00005cc8-2lbzmng-6ccd/f7378ae819553d37d5f2ee142bd6",
117                 "databand_access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmZmaCI6ZnFsc2UsImhhdiI6MTY3OTAzMDcxMCMxMCBmdC"}
118         },
119         enter here the Databand route and the personal Token
120     ),
121     job_name="Python_DAG",
122     run_name="Python_DAG",
123     project_name="Python pipelines"

```

Save the file and we are ready to test the program.

#### 1.4. Invoke the program

We must be located in the directory where the python program is. You may need to modify the cd command.

```
pwd
cd ../dags
ls -l pythondag.py
```

If everything went well, you should see something like this:

Now, you can run the program:

```
# This cell runs the pipeline
python3 ./pythondag.py
```

The expected output is no longer than two lines:

The actual performance data will be displayed on Databand as explained in the sections below.

## 2. Explore the code structure

This sample python pipeline is described in the following pictures. Notice the cyan blocks, which are the specific code of Databand that we need to add to the program in order to generate performance data that can be displayed in the Databand GUI. You may also see an operation in the Task#3 that will not be logged because it is outside of the monitoring block.

```

1 # File: pythondag.py
2 # Simple DAG for the Databand hands-on workshop
3
4 # This python program can be run standalone, on any server or client that can
5 # connect to the system running databand (and Postgres)
6
7 # Create a test table, load data from csv, select data, delete data or drop the table
8
9 # Featuring:
10 # - logging information with the Databand SDK function (dataset_op_logger)
11 # - combine logging and non-logging parts in the code
12 # - integrated with Postgres
13
14 # Mandatory imports
15 import psycopg2
16 import pandas as pd
17 from sqlalchemy import create_engine
18 from dbnd import dbnd_tracking, task, dataset_op_logger
19
20 # The name of the csv file to load into Postgres
21 # The path must be found and accessible by the python program when it runs
22 # This name will be displayed by Databand as a dataset (substituting the slash by dot)
23 motogp_file = 'sql/motogp.csv'
24
25 @task
26 def read_all_championships():
27     # begin logging
28     with dataset_op_logger(motogp_file,
29                           "read",
30                           with_schema=True,
31                           with_preview=True,
32                           with_stats=True,
33                           with_histograms=True
34                           ) as logger:
35         # operation to be logged - read the file into pandas
36         motogp_championships = pd.read_csv(motogp_file, sep=';')
37         # end logging
38         logger.set(data=motogp_championships)
39     return(motogp_championships)
40
41 # Task #2: Filter the data and select only the records of 2022
42
43 @task
44 def select_one_year(alldata):
45     # begin logging
46     with dataset_op_logger(motogp_file,
47                           "read",
48                           with_schema=True,
49                           with_preview=True,
50                           with_stats=True,
51                           with_histograms=True
52                           ) as logger:
53         # operation to be logged - select the Season 2022 in pandas
54         oneyear = alldata[alldata.Season.eq(2022)]
55         # end logging
56         logger.set(data=oneyear)
57     return(oneyear)
58
59
60
61
62
63
64
65

```

**header comments and mandatory imports**

**dataset to be loaded**

**Task #1: load**

**databand code**

**this operation will be logged**

**databand code**

**Task #2: filter data**

**databand code**

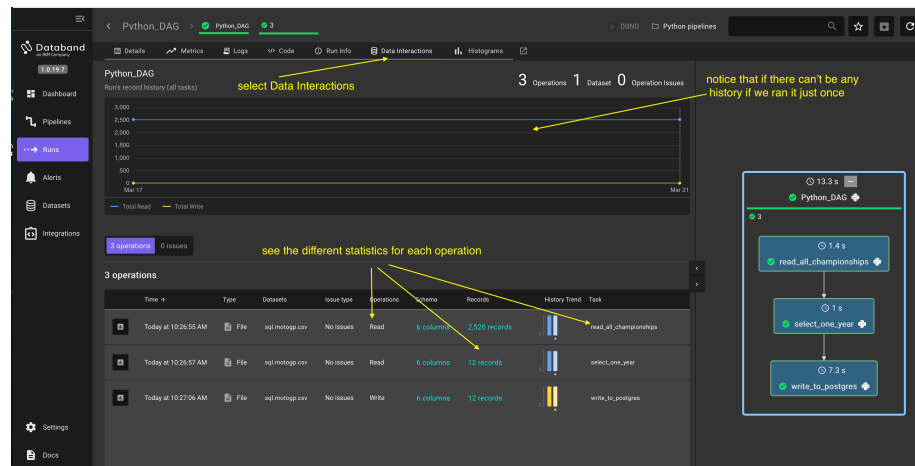
**this operation will be logged**

**databand code**

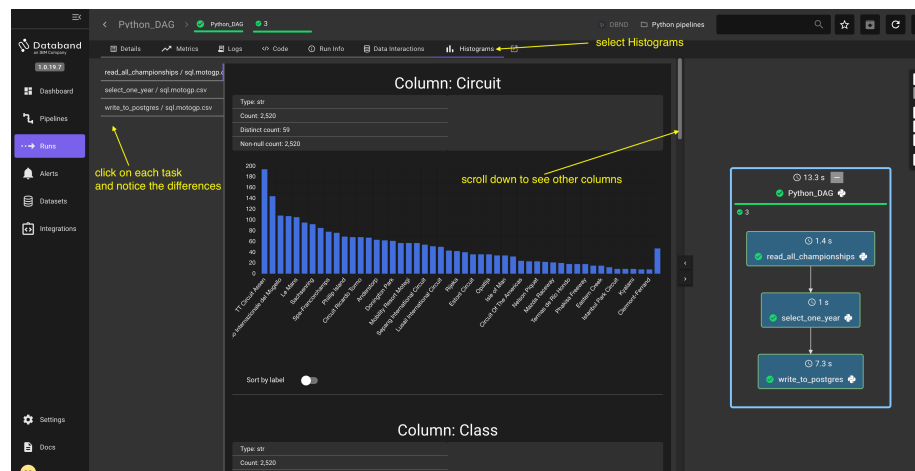




You may wonder why there is no historical information but it is normal because this python pipeline was not scheduled to run periodically, but just occasionally by hand. Indeed, it was run only two times. However, this was enough to collect the data relevant to the datasets, even grouped by the three different tasks of the pipeline.



You may also want to take a look at the histogram information to find out how the value distribution changes during the task runtime.



This information is good for individual executions. In the next sections, we will focus on displaying historical information.

Next Section: Python on Airflow pipelines

Previous Section: Python pipelines

[Return to main](#)