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 neeeded 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 expore 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-
```

```
OC_LOGIN_COMMAND='oc login --token=sha256~6Xs6va20JZ2CFhS61HN6bpQC2z075XZbhIJt3tZ8L6w --serv
$OC_LOGIN_COMMAND
```

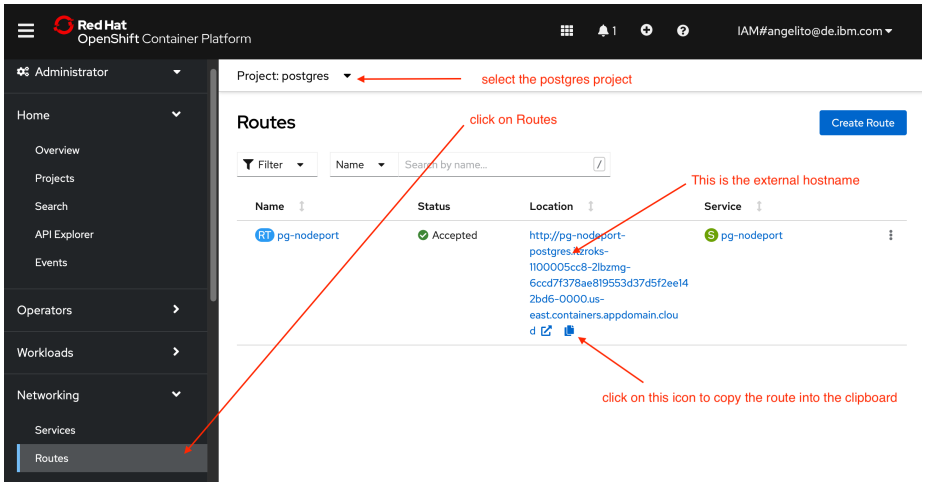
Then, we retrieve the external hostname (or the route or the IP Address) and the port of the Postgres database in the cluster.

```
# 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 -f
echo ------ The following two lines are a hostname '(or IP Address or route)' and a port num
echo $externalhostnamepostgres
echo $externalportpostgres
echo ------ paste these values in the part of the code that defines the Postgres connection
```

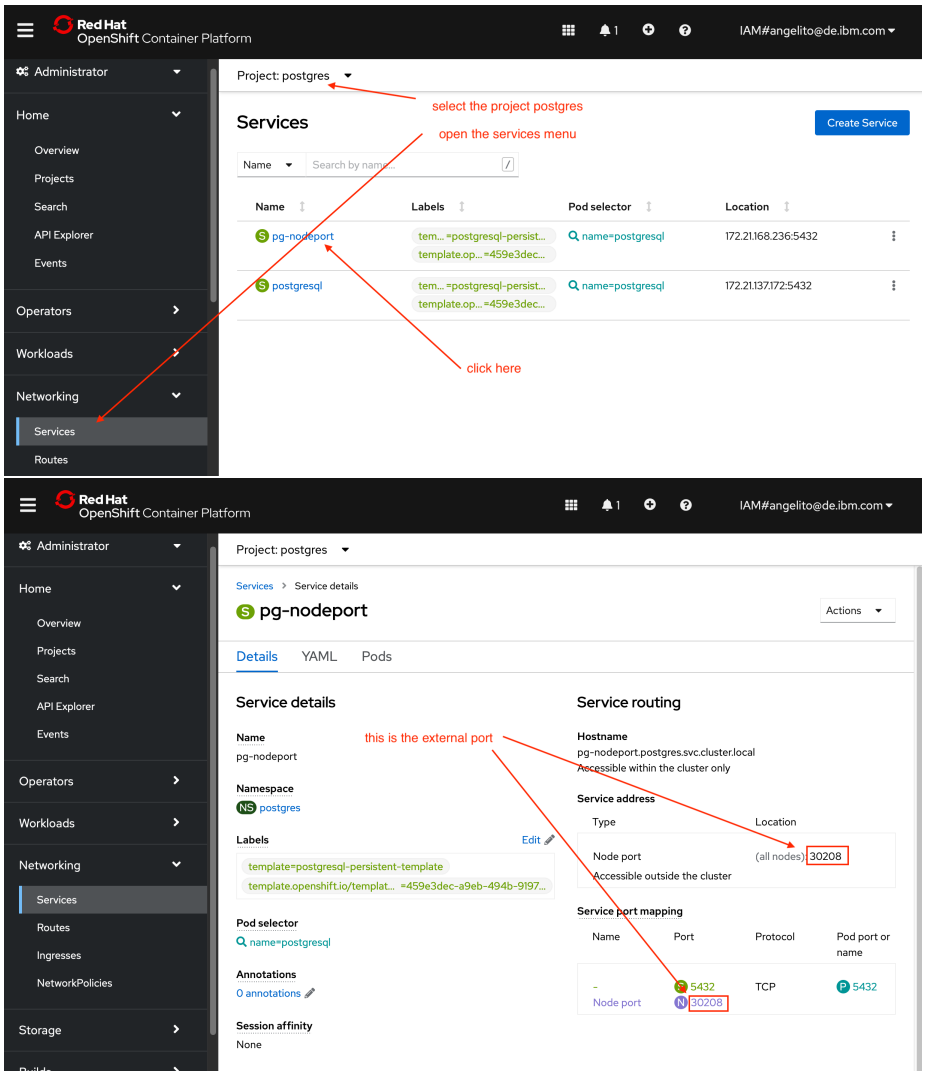
```
Now using project "postgres" on server "https://c103-e.us-south.containers.cloud.ibm.com:325
------ The following two lines are a hostname (or IP Address or route) and a port number ---
pg-nodeport-postgres.itzroks-1100005cc8-myv0re-4b4a324f027aea19c5cbc0c3275c4656-0000.us-sout
32322
------ paste these values in the part of the code that defines the Postgres connection as ir
```

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:



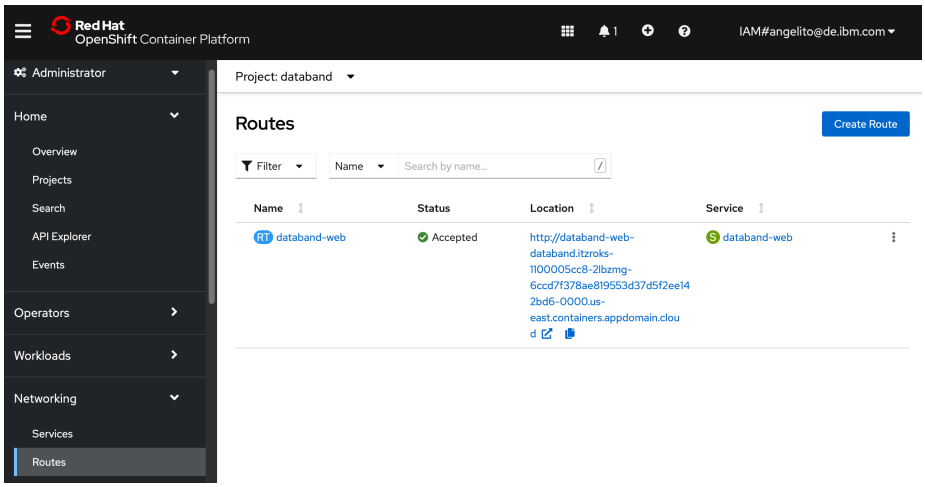
Two more steps are necessary in order to see the external port:



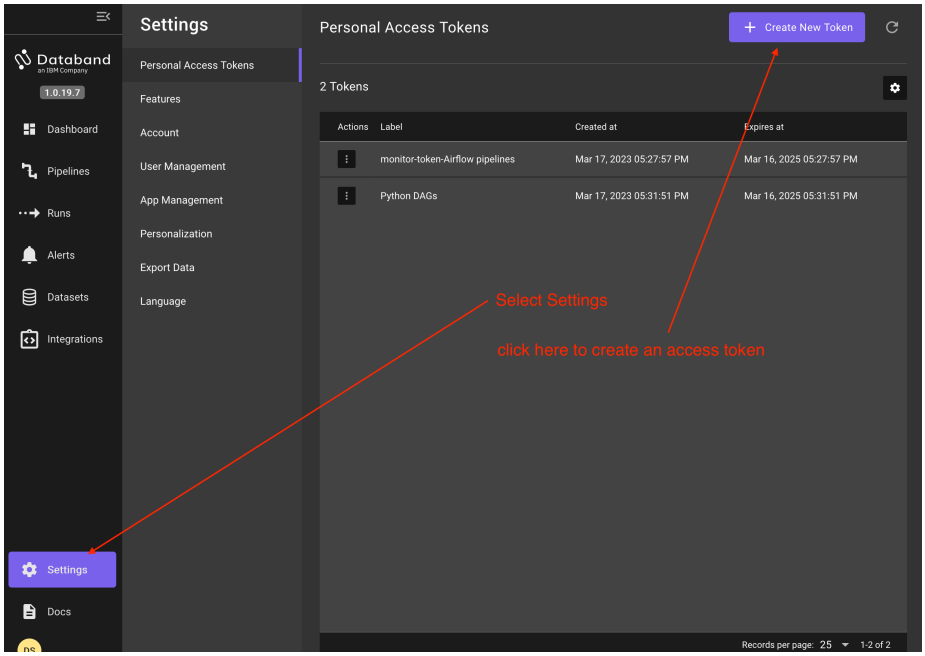
The next thing is to get the connection parameters for Databand. The hostname (the route in OpenShift's dialect)

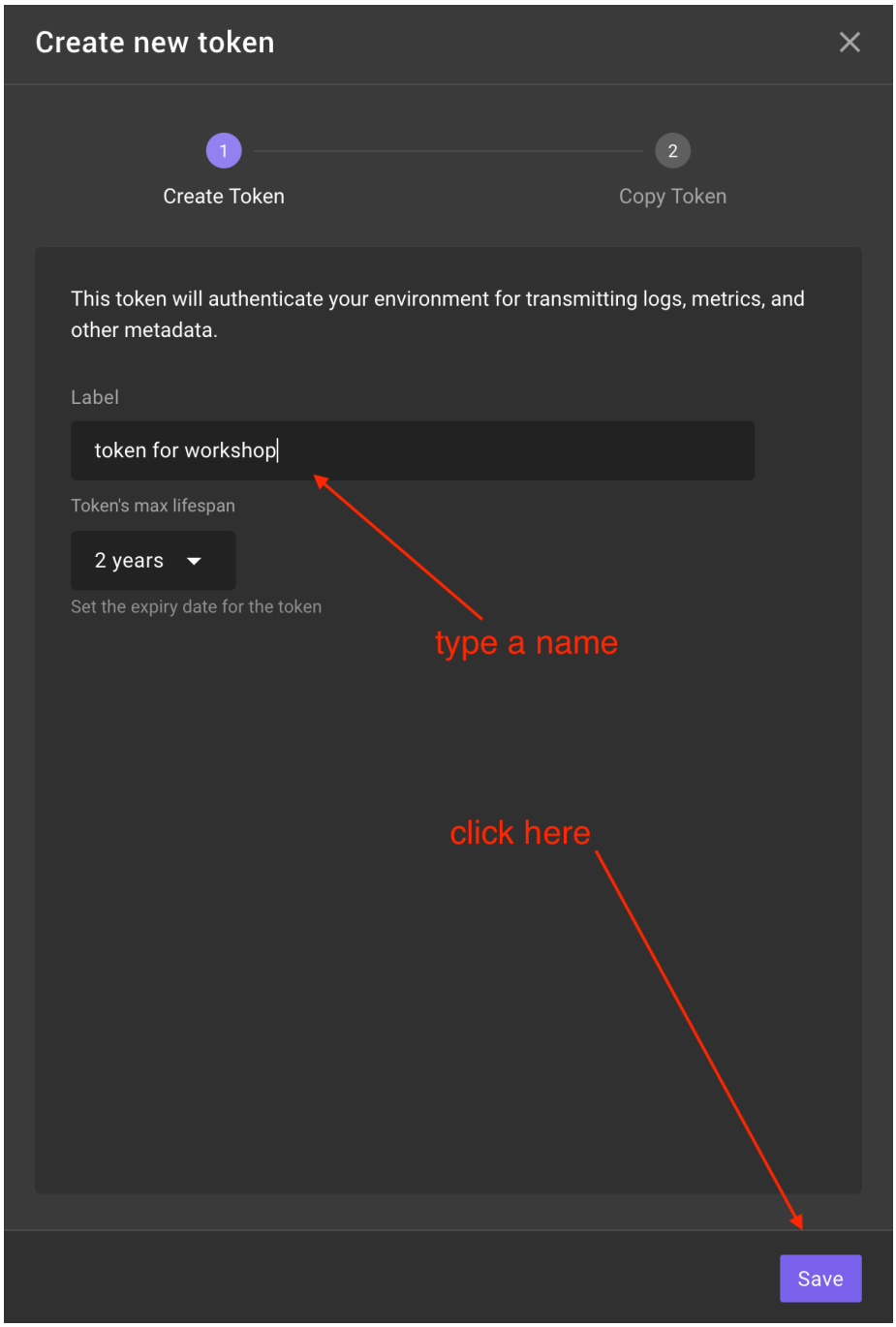
```
# Run this cell to get the hostname and the port of Databand
oc project databand
externalhostnamedataband=$(oc get routes | grep databand | awk '{print $2}')

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
```

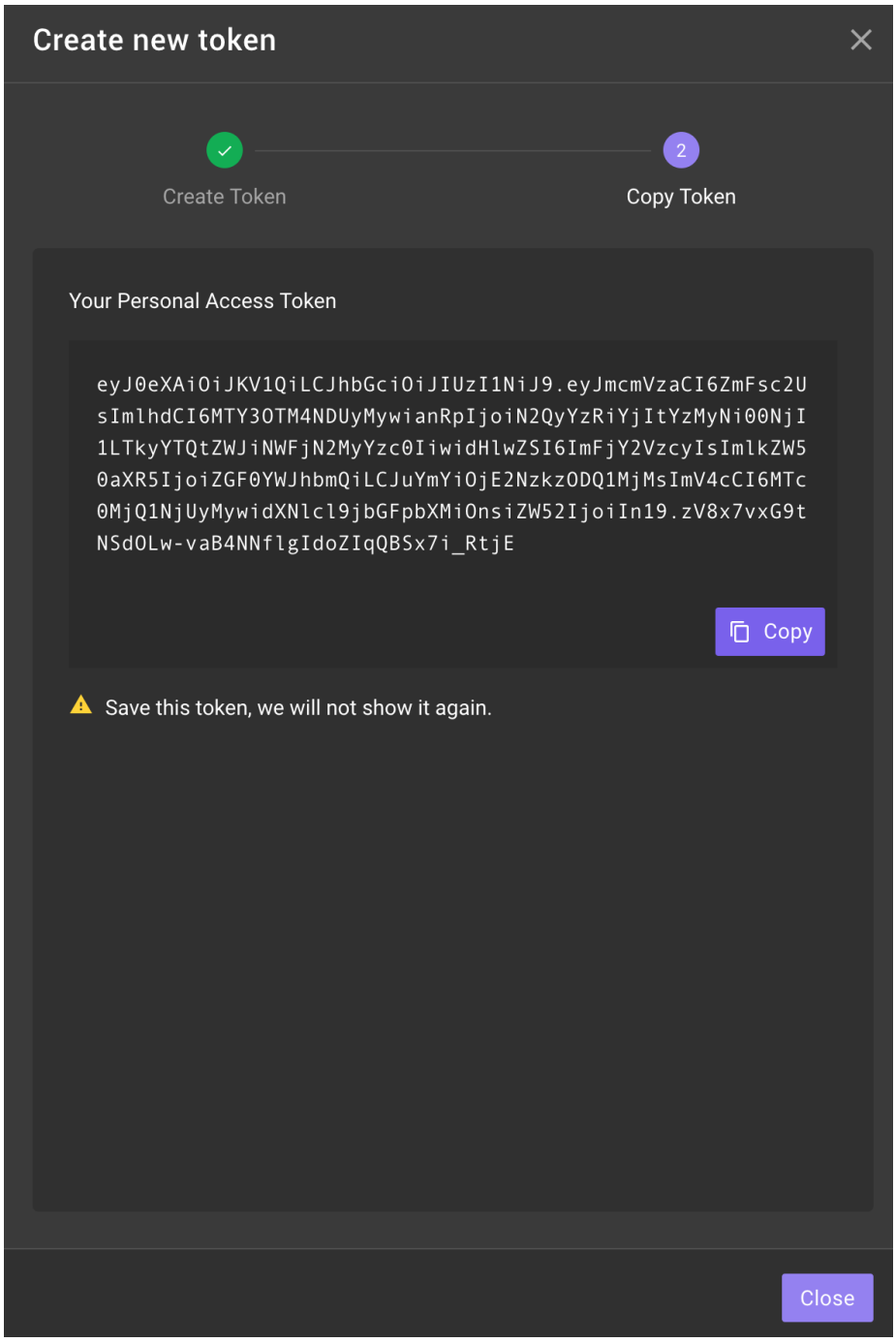
If you want to verify the name, you can see it in the Openshift console as well



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
type a name
click here
Save

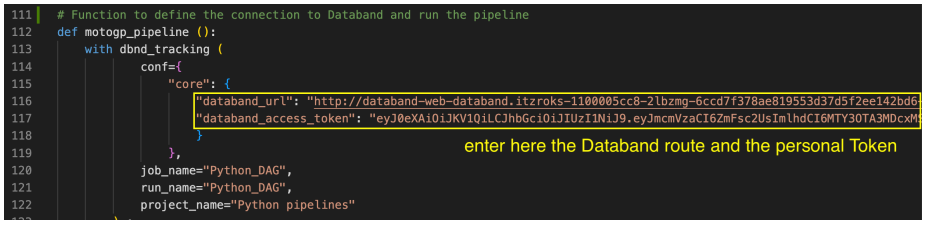

Create new token
Create Token
2
Copy Token
Your Personal Access Token
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2U
sImlhdCI6MTY3OTM4NDUyMywianRpIjoiN2QyYzRiYjItYzMyNi00NjI
1LTkyYTQtZWJiNWFjN2MyYzc0IiwidHlwZSI6ImFjY2VzcyIsImlkZW5
0aXR5IjoiZGF0YWJhbmQiLCJuYmYiOjE2NzkzODQ1MjMsImV4cCI6MTc
0MjQ1NjUyMywidXNlcl9jbGFpbXMiOnsiZW52IjoiIn19.zV8x7vxG9t
NSdOLw-vaB4NNflgIdoZIqQBSx7i_RtjE
Copy
Save this token, we will not show it again.
Close

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://...`

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://..`:



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:

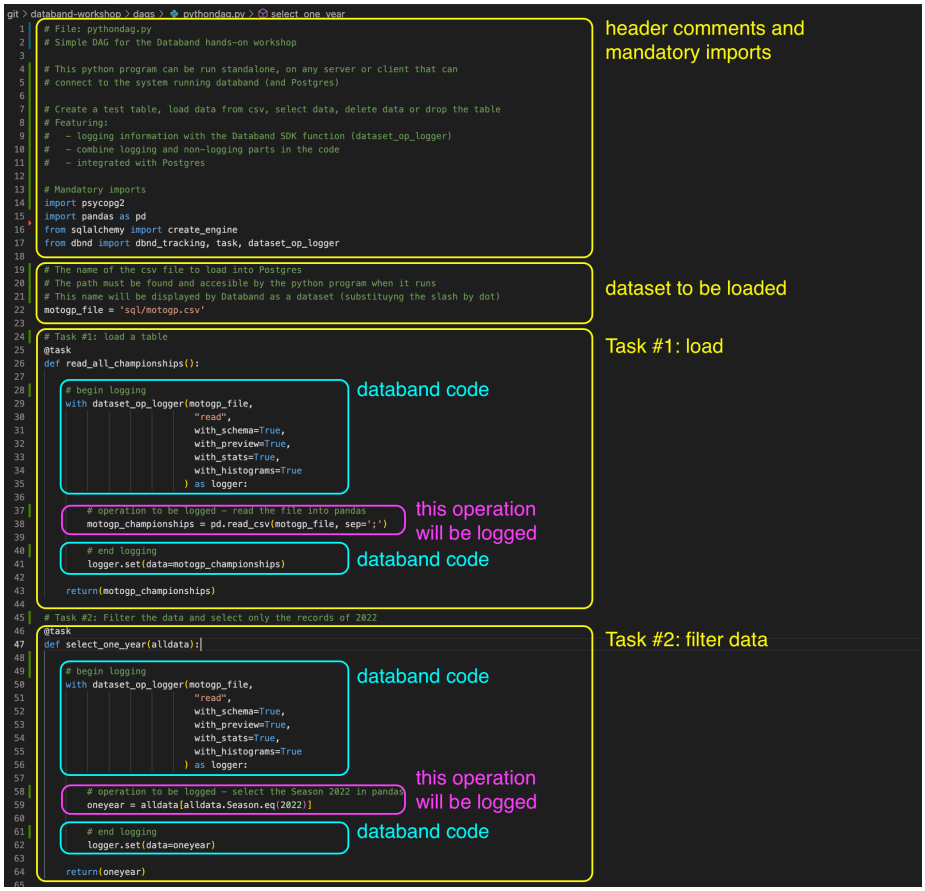
```
# This cell runs the pipeline
python3 ./pythondag.py
```

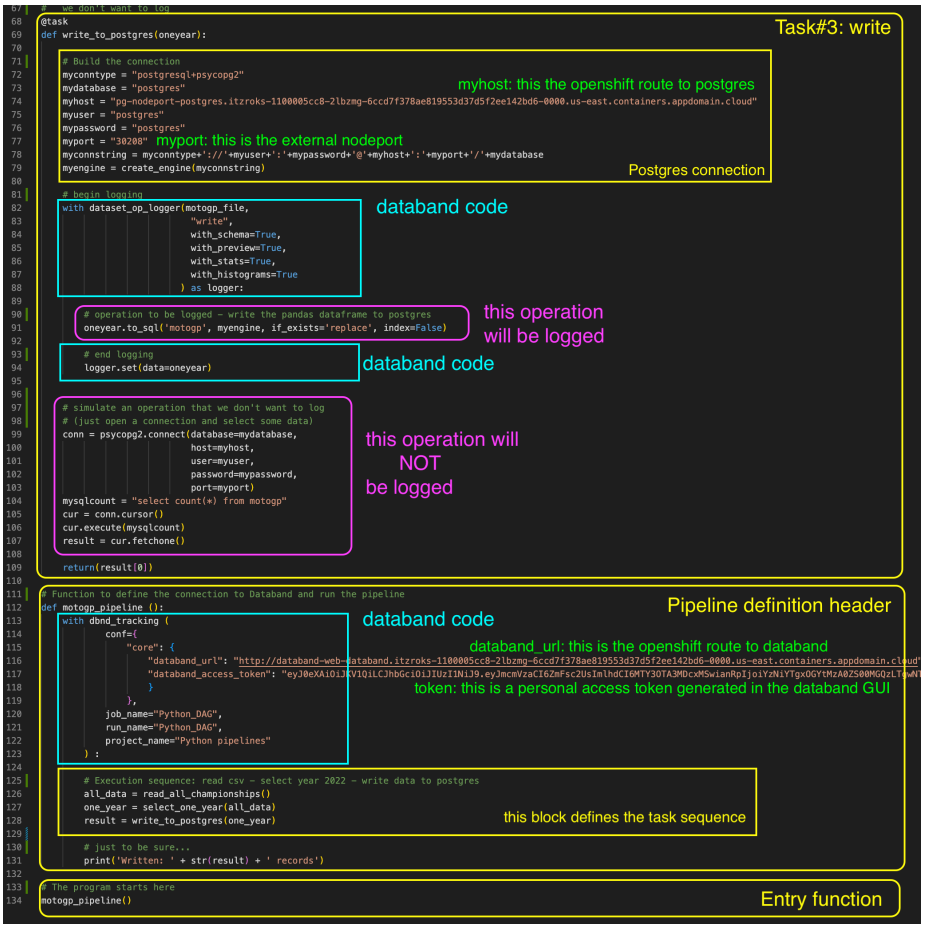
The expected output is no longer than two lines:

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

2. Explore the code structure

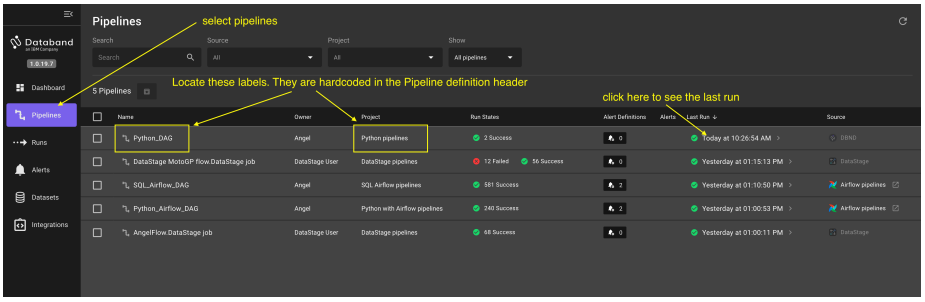
This sample python pipeline is described in the following pictures. Notice the cyan blocks, which are the specific code of Databand that we need to add to the program in order to generate performance data that can be displayed in the Databand GUI. You may also see an operation in the Task#3 that will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



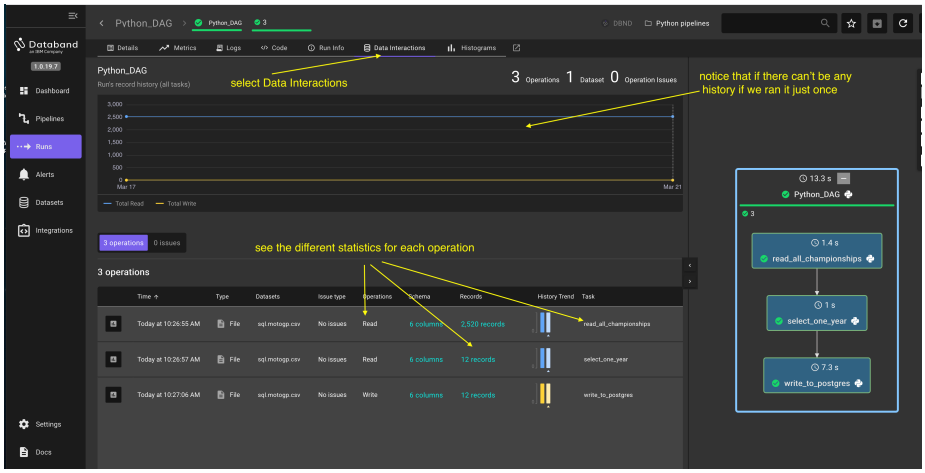


3. Display the performance data available in Databand

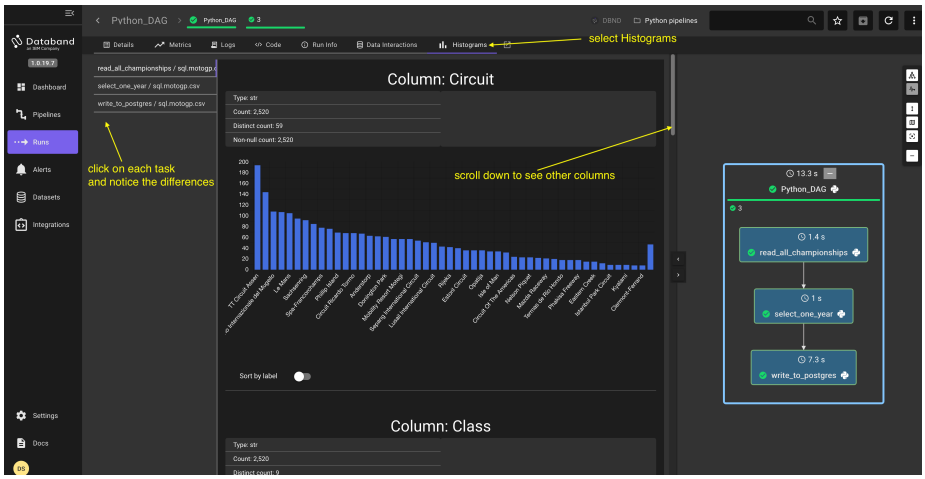
This time, you will see a lot of information collected by Databand. Start by selecting the `Python_DAG` pipeline in the project `Python pipelines`, the names that were given in the code.



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 ocassionally 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 informoation 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