

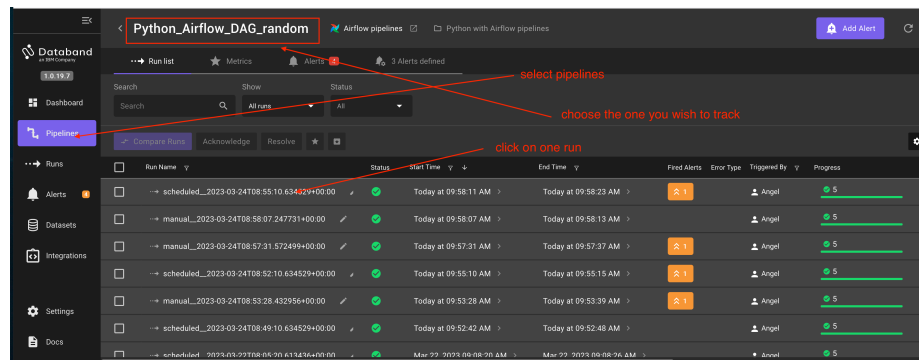
# Alerts and exceptions

In this chapter, we will explore the basic ways to handle exceptional situations and how Databand helps find the root cause very quickly.

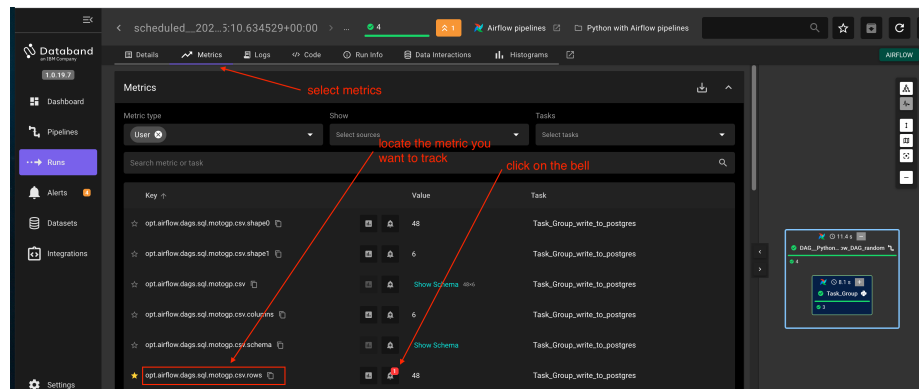
## 1. Customized alerts and incident management

Apart from the predefined alerts that Databand includes, we will see how to create a customized alert on a specific metric that may be important to track.

An intuitive way of defining an alert starts by clicking on any run of a pipeline we want to track:



On the metrics track, we locate the specific parameter that will trigger the alerts:



Next, we define the trigger condition and the threshold. In our case, an alert will be triggered if the last step of the pipeline writes more than 45 rows into the destination table.

**Alert asset**

Pipeline  
Python\_Airflow\_DAG\_random from Airflow pipelines

Task  
write\_to\_postgres

**Alert condition**

Metric  
opt.airflow.dags.sql.motogp.csv.rows

Operator  
greater than or equals

Value  
45

Last 10 runs overview

| Run start time       | Value |
|----------------------|-------|
| Today at 09:58:11 AM | 48    |
| Today at 09:58:07 AM | 0     |

Back Save alert

We can let the pipeline run a few times and, if the trigger condition is met, we will start to see new entries in the Alerts section of the Databand GUI:

**Alerts**

3 Alerts defined

| Search                   | Alert type         | Source   | Project                   | Pipeline          | Status    | Severity              | Triggered Time   | Collaborators |
|--------------------------|--------------------|--|---------------------------|-------------------|-----------|-----------------------|------------------|---------------|
| 4 alerts                 | Acknowledge        | Resolve  | Last seen                 |                   |           |                       |                  |               |
| <input type="checkbox"/> | Task custom metric | Metric opt.airflow.dags.sql.motogp.csv.rows equal or greater than 45 in task write_to_postgres | Python_Airflow_DAG_random | write_to_postgres | Triggered | No assigned receivers | No collaborators | View Details  |
| <input type="checkbox"/> | Task custom metric | Metric opt.airflow.dags.sql.motogp.csv.rows equal or greater than 45 in task write_to_postgres | Python_Airflow_DAG_random | write_to_postgres | Triggered | No assigned receivers | No collaborators | View Details  |
| <input type="checkbox"/> | Task custom metric | Metric opt.airflow.dags.sql.motogp.csv.rows equal or greater than 45 in task write_to_postgres | Python_Airflow_DAG_random | write_to_postgres | Triggered | No assigned receivers | No collaborators | View Details  |
| <input type="checkbox"/> | Task custom metric | Metric opt.airflow.dags.sql.motogp.csv.rows equal or greater than 45 in task write_to_postgres | Python_Airflow_DAG_random | write_to_postgres | Triggered | No assigned receivers | No collaborators | View Details  |

Records per page: 25 14 of 4

If we click on any of the alerts we will see the normal details of the run but we want to highlight the buttons displayed on the right top corner. They are intended to act as incident management



You can click on the Resolve button to delete the alert entry (not the alert definition) or on Acknowledge to investigate the cause of the alert and let it in the list.



Notice that there is other ways of defining customized alerts. For example, like shown on the next picture:



## 2. Exceptions

In this section you can play with the test pipelines and introduce errors to how Databand will report them.

In case of a problem, Databand will mark the runs with the status "Failed":



In this case, you the cause of the problem is that a column does not exist, as indicated in the first line of the log.



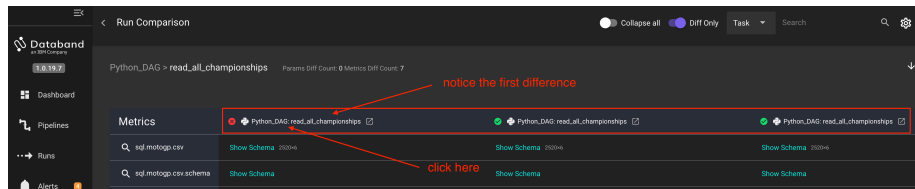
If you choose another failed run, you may see a different error. In that case, that the table does not exist, as displayed in the first line of the log.



Another great way to determine the cause of an error is to select a few runs and click on "Compare Runs" to see the differences between a good runs and bad ones.



The first difference can be found on one of the tasks, which has failed:



By clicking on the failed task, you can see immediately the cause of the error.



The second difference would be more difficult to find manually but Databand shows the line of the code that causes the error and compares it with the same line of another run that worked well:

Run Comparison

Python\_DAG = select\_one\_year

Params Diff Count: 0 Metrics Diff Count: 7

notice that is section is coloured differently

| Metrics | sql.motogp.csv.columns | sql.motogp.csv.rows | sql.motogp.csv.schema | sql.motogp.csv.shape0 | sql.motogp.csv.shape1 | sql.motogp.csv.read_rows |
|---------|------------------------|---------------------|-----------------------|-----------------------|-----------------------|--------------------------|
| 6       | 6                      | 12                  | None                  | 12                    | 6                     | 12                       |
| 12      | 12                     | 12                  | None                  | 12                    | 6                     | 12                       |
| None    | None                   | None                | None                  | None                  | None                  | None                     |
| 12      | 12                     | 12                  | 12                    | 12                    | 6                     | 12                       |
| 12      | 12                     | 12                  | 12                    | 12                    | 6                     | 12                       |
| 12      | 12                     | 12                  | 12                    | 12                    | 6                     | 12                       |
| 12      | 12                     | 12                  | 12                    | 12                    | 6                     | 12                       |

see the difference of in the code

Code diff for: Task Source Full Module

```

PYTHON_DAG_SELECT_ONE_YEAR
1 # Pass
2 def select_one_year(all_data):
3
4 # begin logging
5 with dataset_log_logger(motogp_file,
6 "read",
7 with_schema=True,
8 with_previous=True,
9 with_stats=True,
10 with_histogram=True
11
PYTHON_DAG_SELECT_ONE_YEAR
1 # operation to be logged - select the Season 2022 in pandas
2 oneyear = all_data[all_data.Season.eq(2022)]
3
4 oneyear = all_data[all_data.Season.eq(2022)]
5
6 # end logging

```

Next Section: Customized Metrics

Previous Section: DataStage pipelines

Return to main