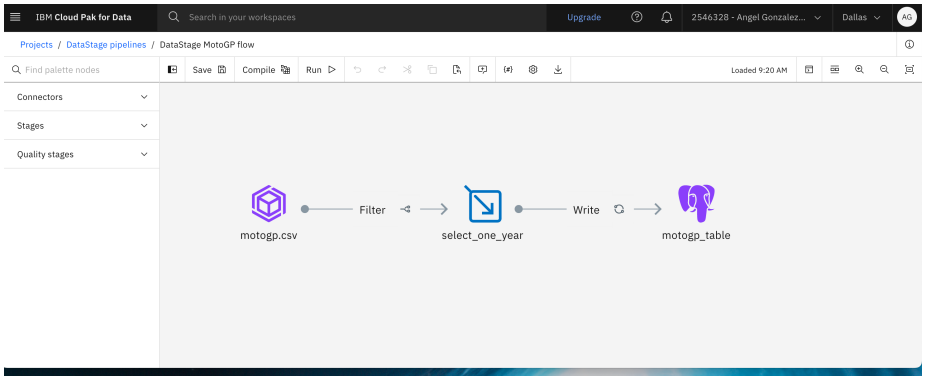
DataStage Pipelines

In this section we will build a DataStage job that performs the same operations as the two pipelines reviewed so far.

1. Build a DataStage Job

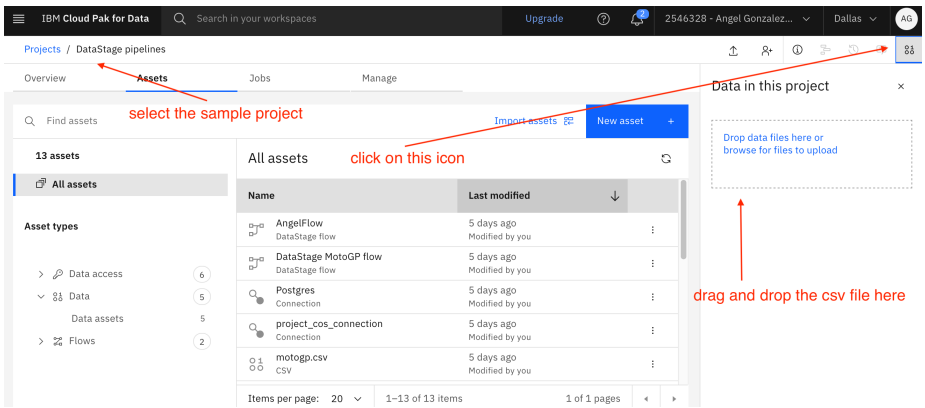
This is the DataStage job that we will create from scratch. As you see, it reads a csv file, performs some fltering and write the results on a Postgres :



1.1. Create a data asset for the input file motogp.csv

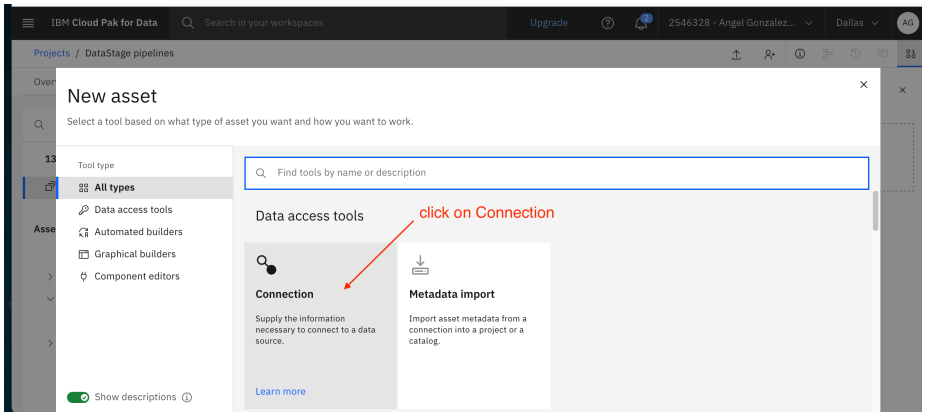
This git repository contains a small file motogp.csv (2500 records) that we will use as input file.

Go to the main project panel and upload the file into the project to create an data asset. Just follow the instructions on the following picture and leave all fields you see with the default values:

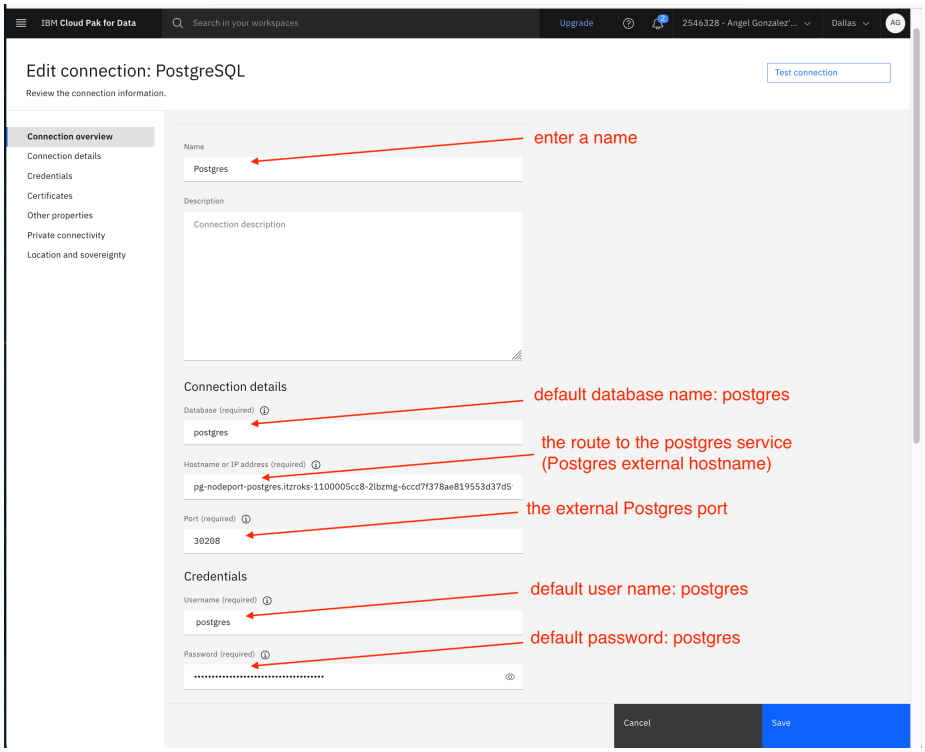


1.2 Create a connection asset for Postgres

On the same project main panel of the picture above, click on new asset. Select an asset connection:

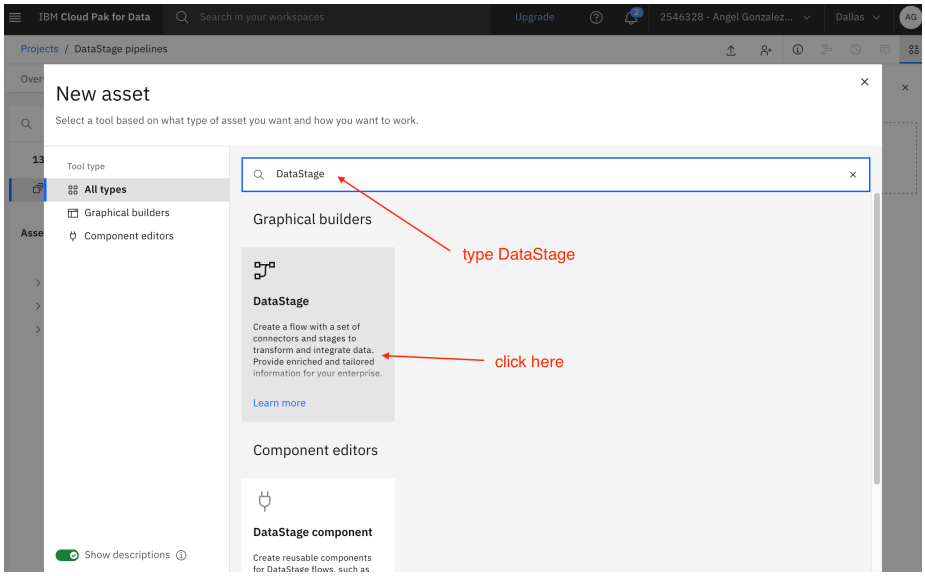


Type the required values as shown on the next screenshot and test the connection. Do not forget to save the asset.

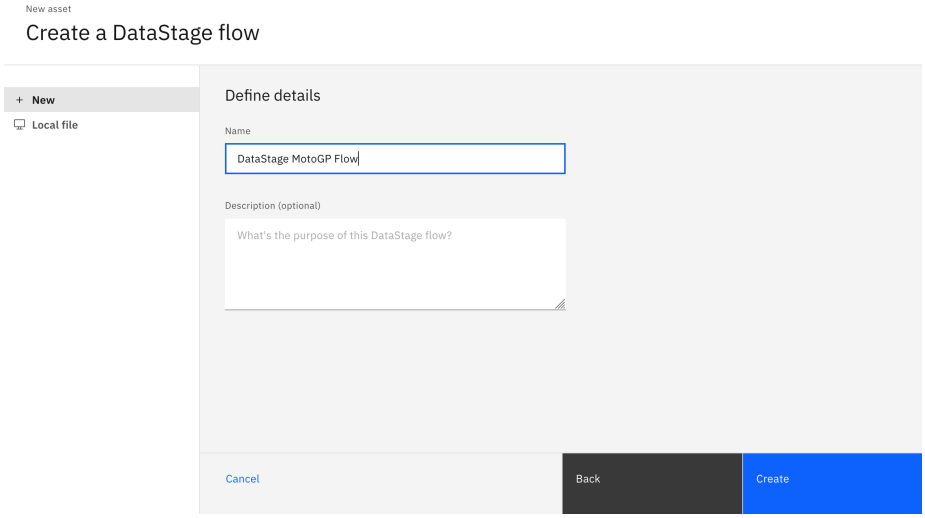


1.3 Create a DataStage flow

Again, on the main project pannel, click on new asset and search for DataStage:

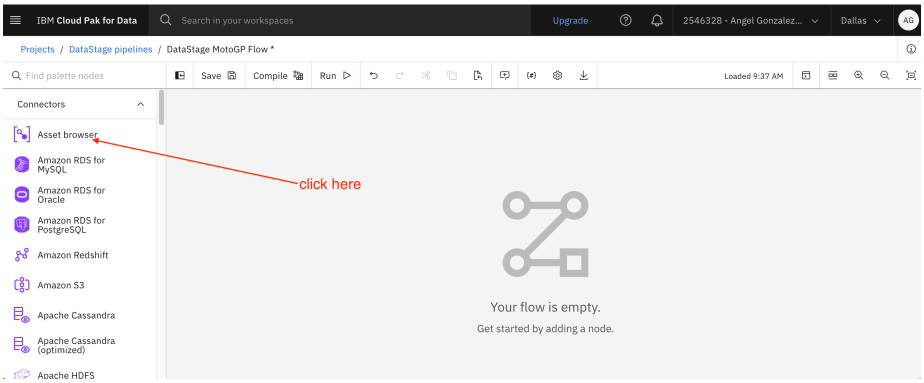


Type a name so that you can recognize it later in the Databand GUI and click on create.

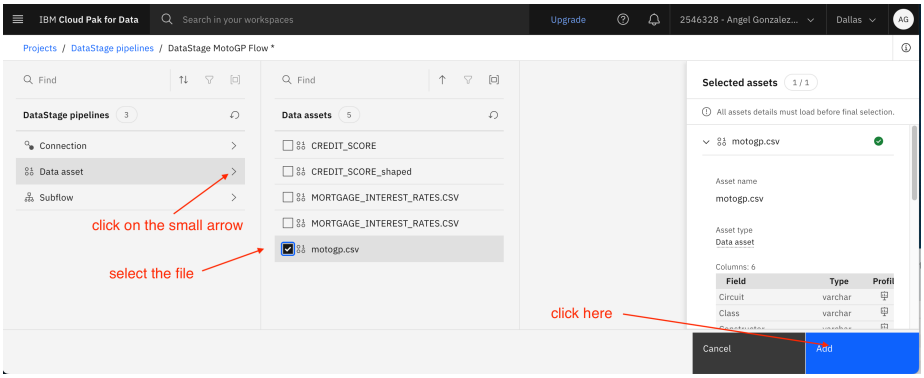


1.4 Add the csv file to the flow

An empty workspace will be shown after creating the job. Now, just click on asset browser:



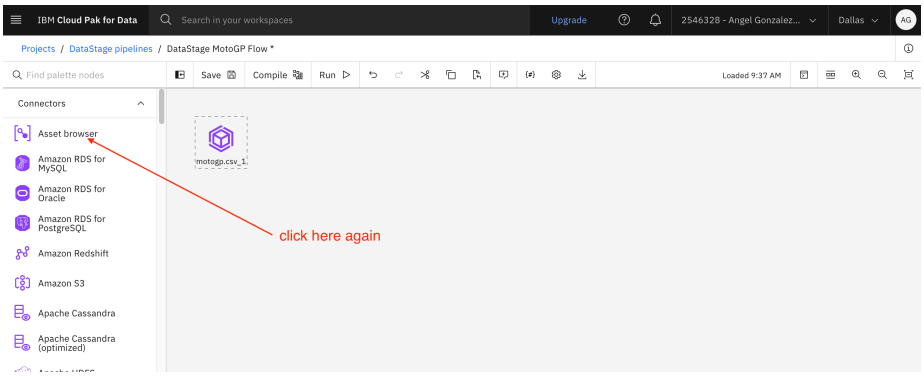
Add the `motogp.csv` file as instructed in the next screenshot:



The workspace will show an icon representing the input file.

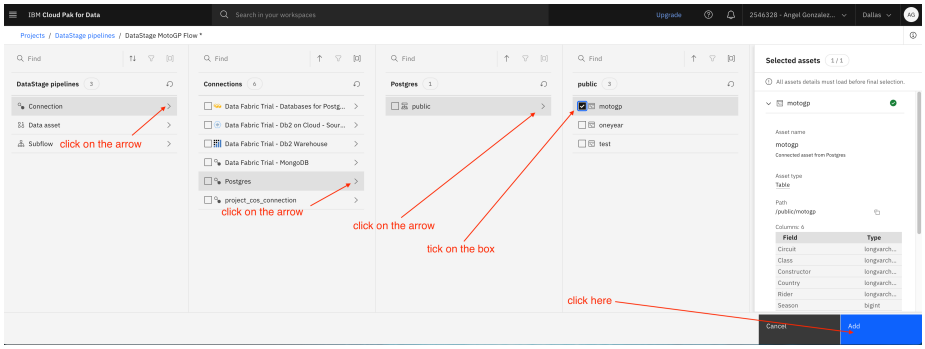
1.5 Add the Postgres table to the flow

In this step, we will add the destination of the data. As we have created an asset for Postgres, we can add it from the access browser again:



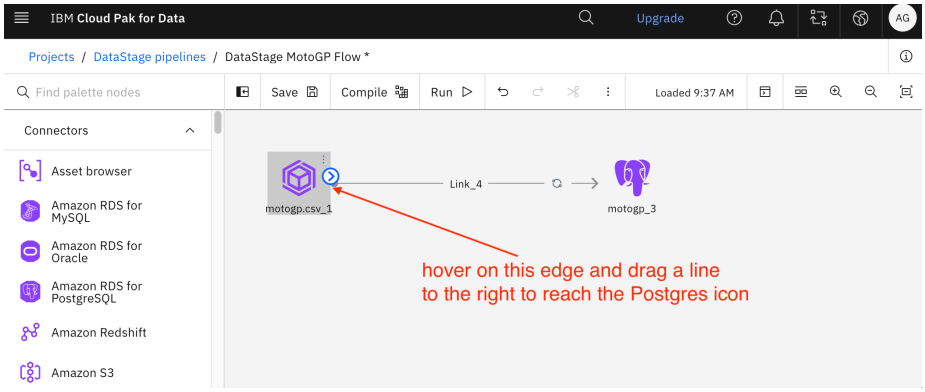
Follow the instructions on the screenshot to incorporte the `motogp` table to the

flow:



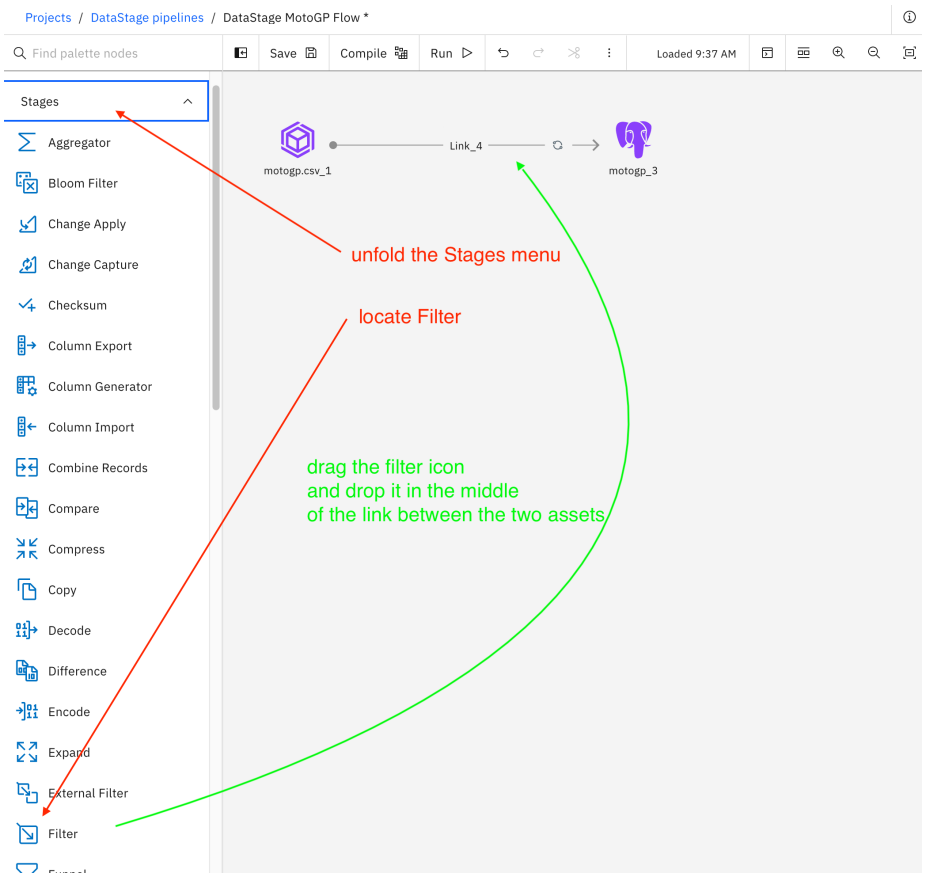
1.6 Connect the two assets

We will connect the input file with the table as follows:

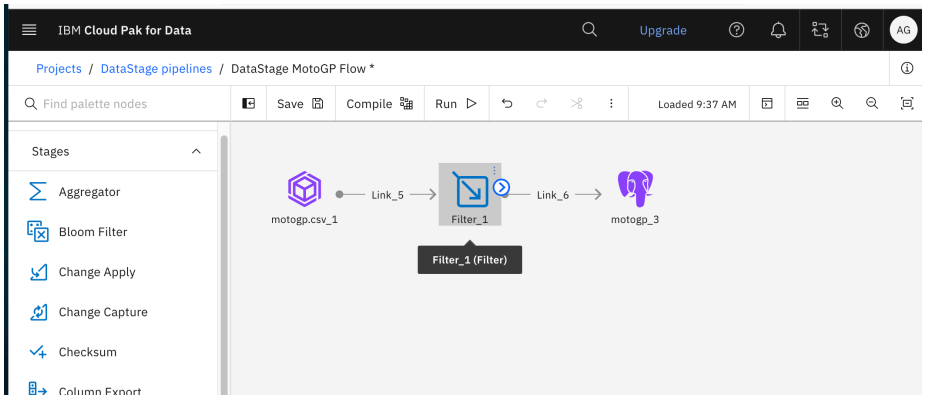


1.7 Add a filter

We don't want to load the full data, but only one season of the chapionships. So, we add a filter between the assets as follows:

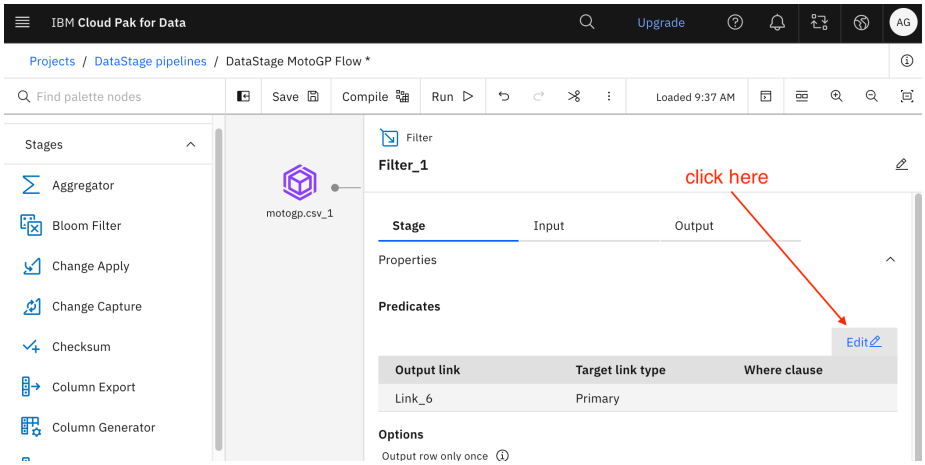


If you did it correctly, it will look like this:

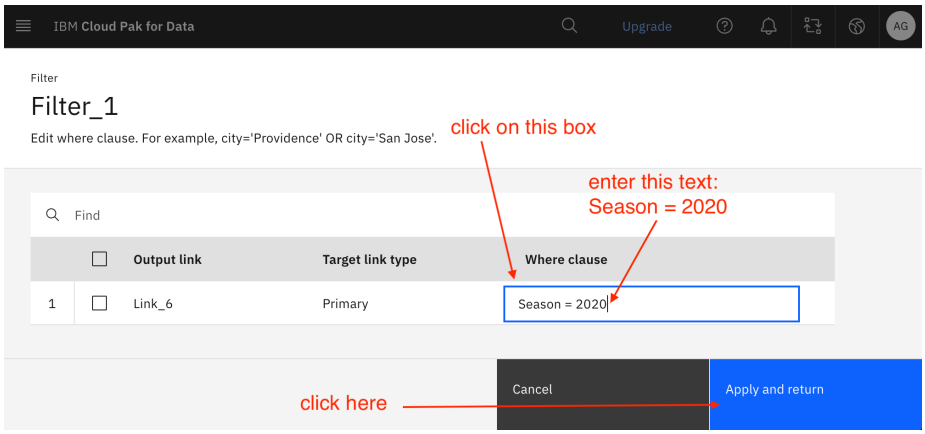


1.8 Specify the filter condition

We will select a portion of the csv file by typing an SQL-like clause, although the input data is not a relational table. Click on the filter icon and go to edit:

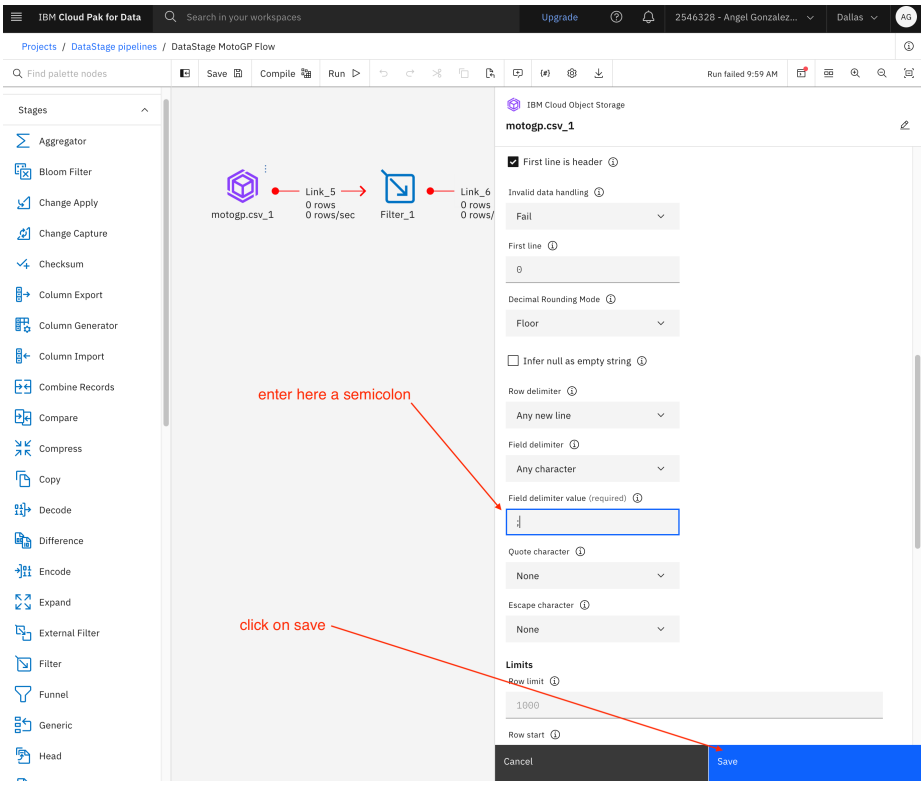


A new panel will open and that is the place to enter the filter condition:



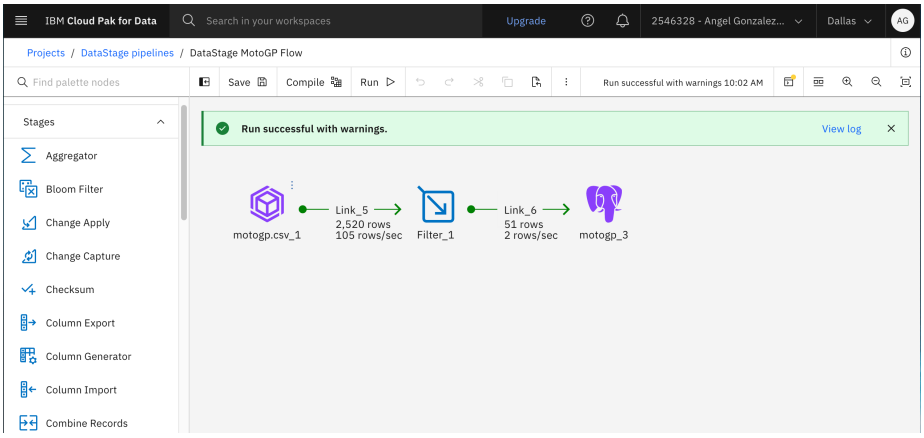
1.9. Correct the delimiter character option of the csv file

DataStage requires to enter the delimiter character of the CSV files. Just click on the motogp.csv icon of our flow, unfold the properties panel and find the place to specify a semicolon like follows:



10. Test the flow

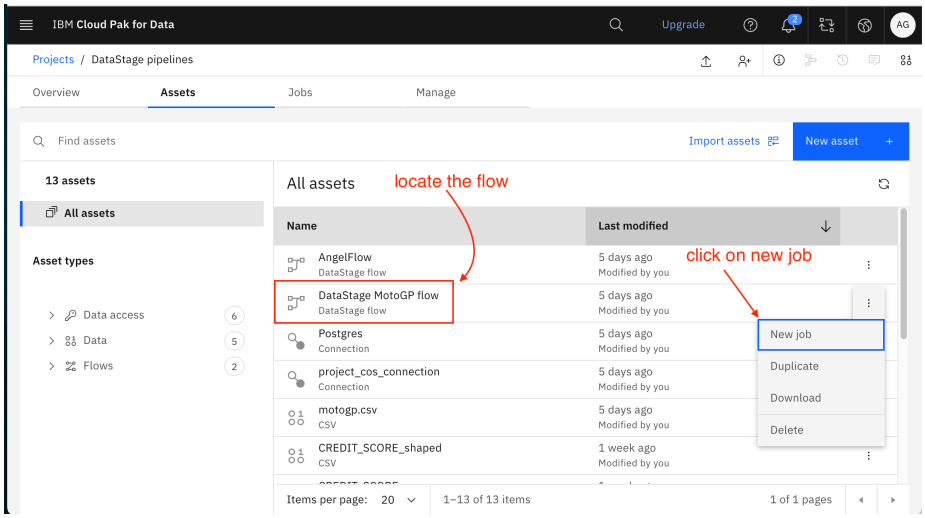
To run the flow just click on the arrow at the top and wait about one minute. If everything went well, you will see something like this:



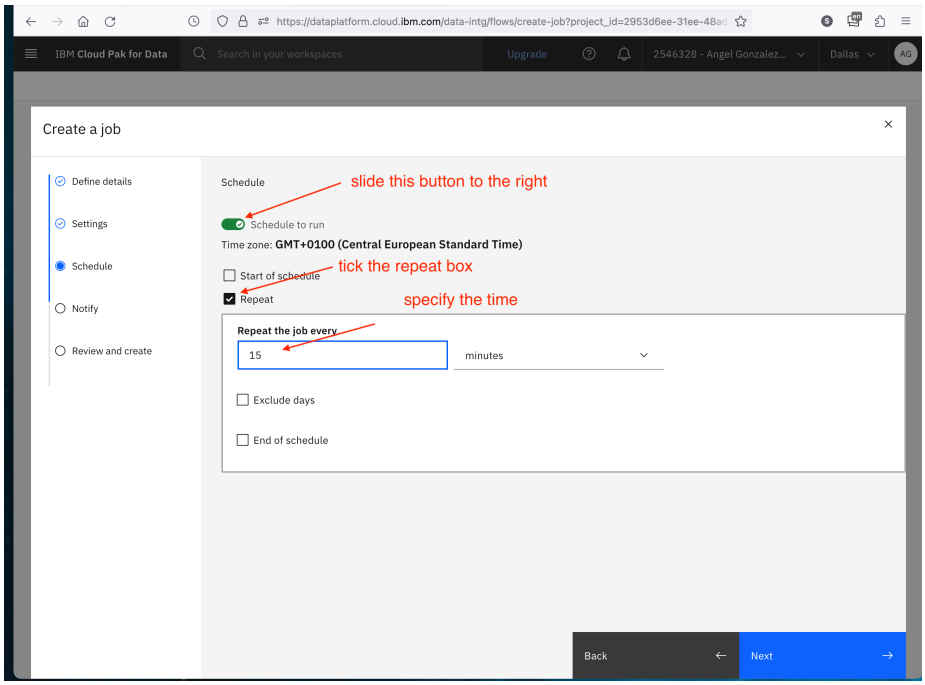
The warnings can be ignored. They simply indicate that the job will not be parallelized, which is OK for our data volume.

1.11. Schedule the flow

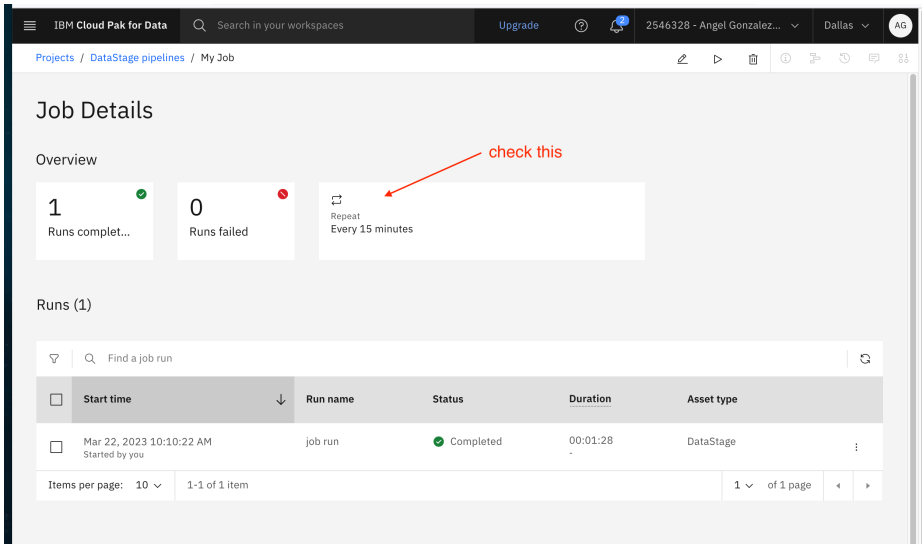
Go to the main project panel and proceed as indicated in the following screenshot:



The job schedule panel is very straightforward, just take care to specify the right schedule interval:

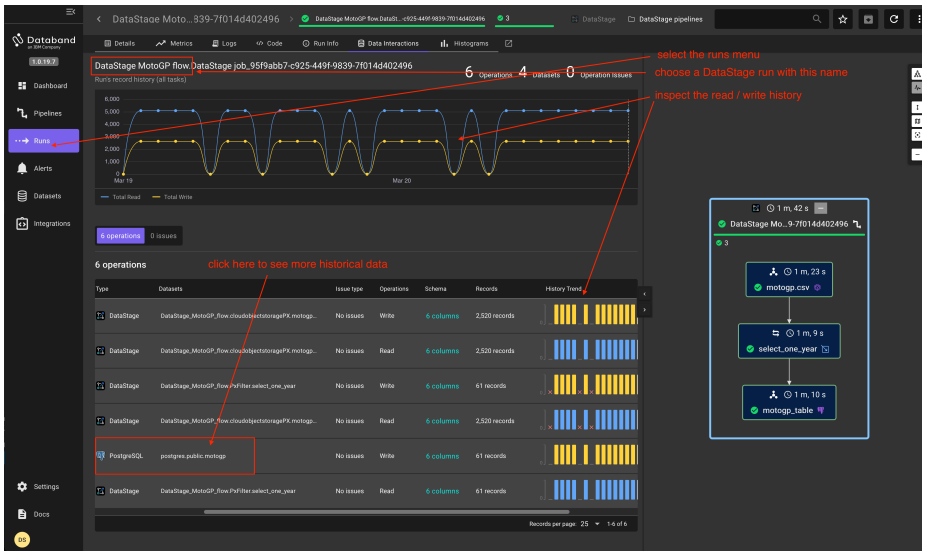


And verify that the job will run as specified:

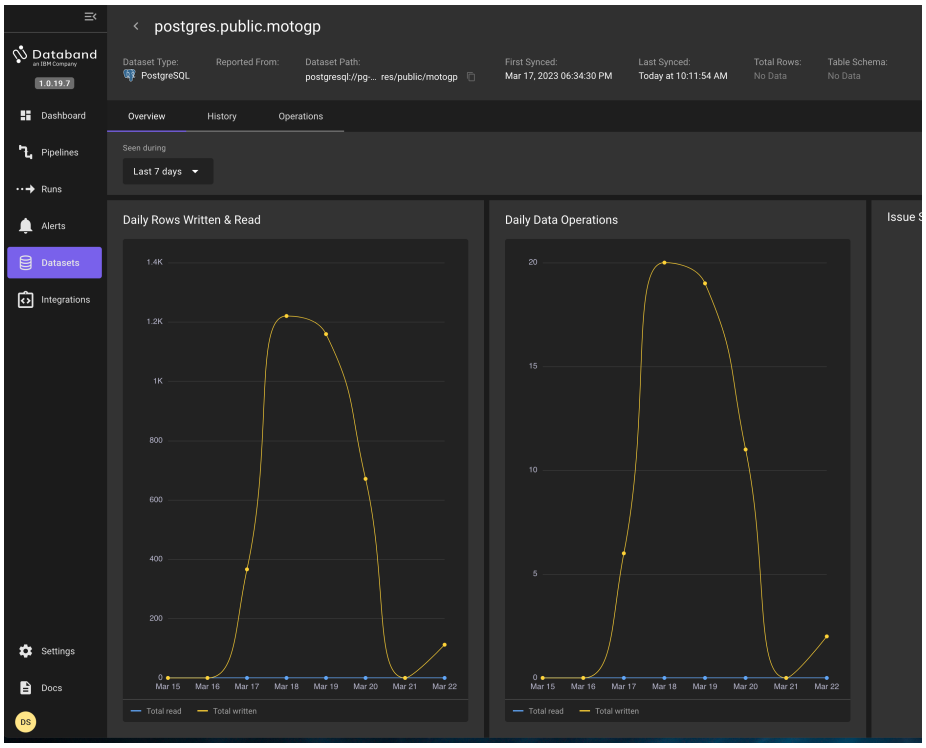


2. Observe performance data

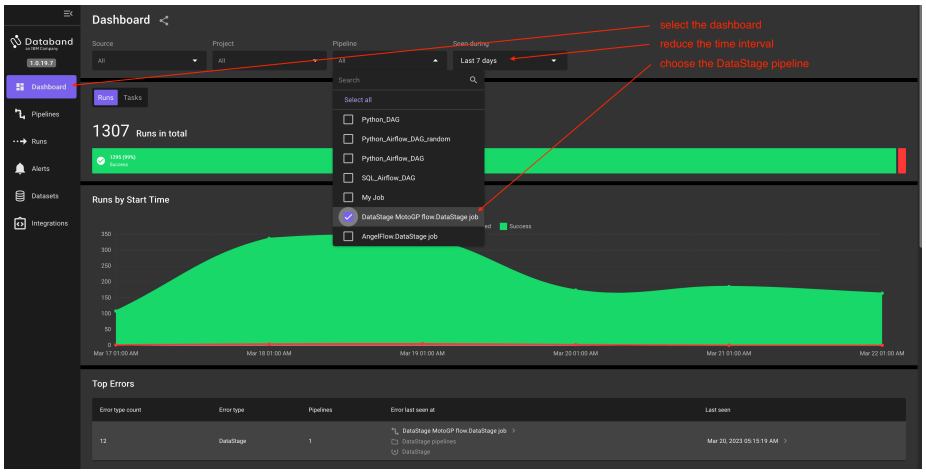
The DataStage performance data exhibit more variability due to the concurent execution of other jobs dropping randomly the destination table in Postgres, which generated errors and interruptions.



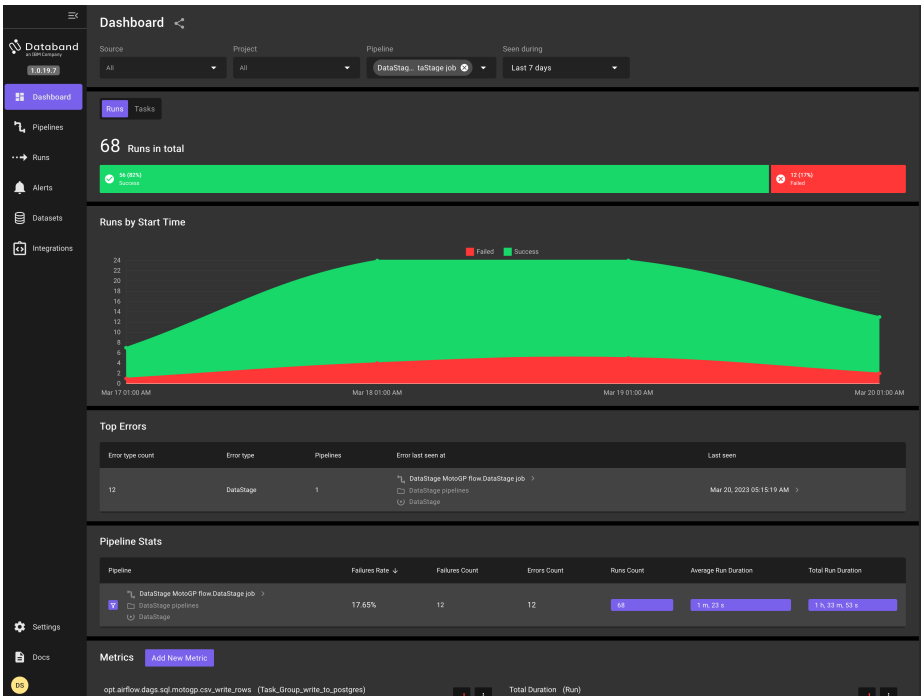
Notice how the destination table is explicitly labeled and how the historical data is shown:



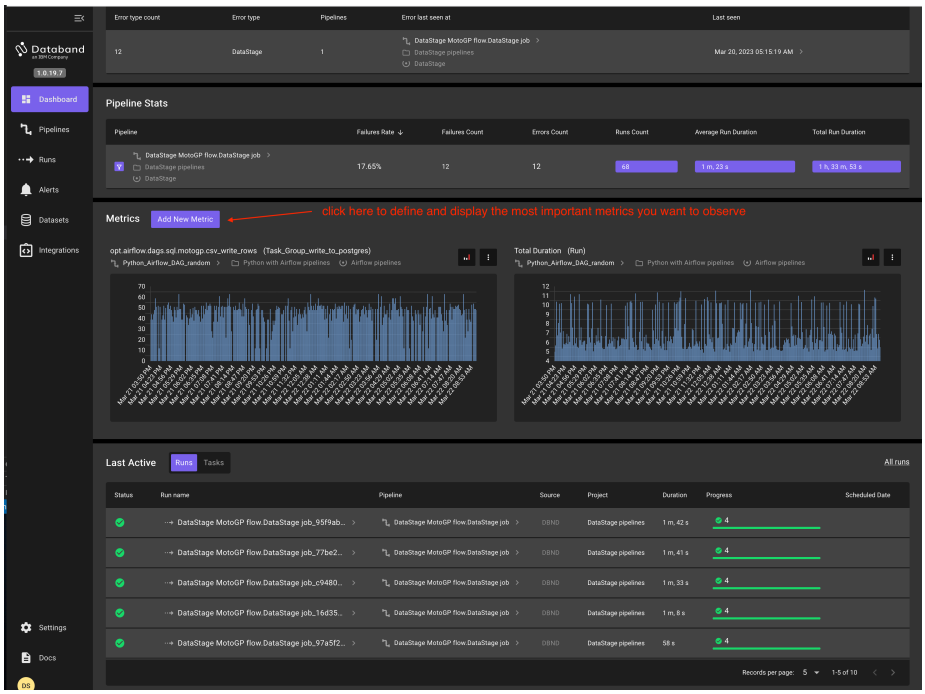
If you schedule at short intervals all the pipelines we built so far, a considerable number of runs and statistics will appear. Consider the use of filtering to find out what you are looking for:



See how we reduced the analysis from over 1000 pipelines to 68.



Remember that you can define the most important metrics to be shown in the dashboard:



We will explore the metrics and alerts in the next sections

Next Section: Alerts and Exceptions

Previous Section: Python on Airflow pipelines

Return to main