Return to "Data Engineering Nanodegree" in the classroom

Data Pipelines with Airflow

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

IMPRESSIVE!!
You did very well on this and passed the project successfully from the first trail! Congratulations :)
Thanks a lot for your effort and commitment to get the project completed. wish you the best of luck in future submissions and projects :)

General

✓ **DAG can be browsed without issues in the Airflow UI**

Correct! Dag loaded and Browsed successfully through the interface without any issues

✓ **The dag follows the data flow provided in the instructions, all the tasks have a dependency and DAG begins with a start_execution task and ends with a end_execution task.**

Good work! DAG follows the required data flow provided. All functions have the correct dependencies, star point and end point are present as requested in the instructions

Dag configuration

✓ **DAG contains default_args dict, with the following keys:**

- Owner
- Depends_on_past
- Start_date
- Retries
- Retry_delay
- Catchup

All default arguments are defined correctly as required in the default_args dict

✓ **The DAG object has default args set**

DAG defined in the right manner, and the default_args dict was assigned in the right manner to the DAG's default_args parameter

✓ **The DAG should be scheduled to run once an hour**

Excellent! scheduling was done using `0 * * * *`. You can also use the `@hourly` expression

Staging the data

✓ **There is a task that to stages data from S3 to Redshift. (Runs a Redshift copy statement)**

Very good! the StageToRedshiftOperator is present as required and it's running the copy statements from S3 o AWS Redshift

✓ **Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

Great work! The copy statement is parametrized and not hardcoded, so it can load multiple tables with different config and not tied to only table.

✓ **The operator contains logging in different steps of the execution**

Good work but only two lines of logging is present. Logging should be enhanced to be able to monitor the different functionalities occurring within your operator and to be able to debug your code in case of any issue as this will minimize the time required to resolve an issue if any.

✓ **The SQL statements are executed by using a Airflow hook**

Nice work on this. Used the PostgresHook to define your connection

Loading dimensions and facts

✓ **Dimensions are loaded with on the LoadDimension operator**

Nice work! Dimensions are loaded using the LoadDimensionOperator operator successfully

✓ **Facts are loaded with on the LoadFact operator**

Correct! You used the loadFactOperator operator in the correct manner required.

✓ **Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

You did well on this by passing SQL queries as a parameter with the table and connection ID to both operators. This makes both operators as dynamic execution engine for queries and not dedicated to load only one table.

✓ **The DAG allows to switch between append-only and delete-load functionality**

You did very well on this by using the truncate_table flag, according to which you decide to truncate prior insertion or not based on the loading mode required; append or truncate-insert

Data Quality Checks

✓ **Data quality check is done with correct operator**

Correct! you created an operator for data quality to run after loading is finished.

✓ **The DAG either fails or retries n times**

Correct! exceptions are raised in case the expected result was not met, this will fail the task and will lead to retry logic as per the config passed to the dag.

✓ **Operator uses params to get the tests and the results, tests are not hard coded to the operator**

Excellent! Exactly as required. You passed both tests and expected results as parameters to the operator. This makes your operator dynamic and able to run any tests in a dynamic manner. You can also add another field in the dict you are passing which is the comparison operator; according to which you decide how to compare between the result generated and expected result.

For example

```
dq_checks=[{'check_sql': "SELECT COUNT(*) FROM users WHERE userid is null", 'expected_result': 0, comaprison:'='},
                       {'check_sql': "SELECT COUNT(*) FROM songs WHERE songid is null", 'expected_result': 0 , comparison:'>'}]
```

DOWNLOAD PROJECT

RETURN TO PATH