UDACITY

‹ Return to "Data Engineering Nanodegree" in the classroom

Data Pipelines with Airflow

REVIEW

CODE REVIEW 3

HISTORY

Meets Specifications

Keen Learner,
Congratulations on completing the project! You should be very proud of your accomplishments in building an awesome ETL pipeline with Airflow! You have now the knowledge of how to build dynamic and reusable ETL pipelines and how to make sure the data quality meets specifications. The work done demonstrates a good understanding of the concepts covered in the project. Continue with this hard work and good luck moving forward.

Extra Material

You can check out the following links I find useful for further learning.

- Data pipelines, Luigi, Airflow: everything you need to know
- Airflow: Lesser Known Tips, Tricks, and Best Practises
- DAG Writing Best Practices in Apache Airflow
- Airflow Tips, Tricks & Pitfalls

General

DAG can be browsed without issues in the Airflow UI

The DAG looks just like expected and the task dependencies follow the required data flow. Great job!
To make the DAG even more compact, you could try to use the `SubDag` operator with the dimension loads and hide the repetitive parts behind that.

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.

Nice work on coding the DAG and the required operators. The code looks very clean and is easy to follow.

Tips

One thing to enhance here in the future could be to actually upload the Airflow logs to S3 for example so the logs would not fill the disk on the executor or worker machines. The logging to S3 can be achieved by modifying the Airflow configurations.

Dag configuration

DAG contains default_args dict, with the following keys:

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

Good job defining the `default_args` dictionary.

The DAG object has default args set

Good job here✅

The DAG should be scheduled to run once an hour

Good job scheduling the Dag to `@hourly`, running once in an hour as expected.

Staging the data

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

✓
On running the DAG, data is loaded to the staging tables in Redshift.

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

Good job dynamically generating copy statement using params like `s3_key` , `s3_bucket` and `table` as opposed to static SQL statements.

The operator contains logging in different steps of the execution

`logging.info` shows the progress of staging load.

The SQL statements are executed by using a Airflow hook

Good job connecting to the database via an Airflow hook.

Loading dimensions and facts

Dimensions are loaded with on the LoadDimension operator

Good! Separate functional operator for dimensions(`LoadDimensionOperator`).

Facts are loaded with on the LoadFact operator

Good! Separate functional operator for facts (`LoadFactOperator`).

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

Good job dynamically generating copy statement using params like `s3_key`, `s3_bucket` and `table` as opposed to static SQL statements.

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

Well done!

Data Quality Checks

Data quality check is done with correct operator

Nice!

The operator that runs a check on the fact or dimension table(s) after the data has been loaded is `DataQualityOperator`.

The DAG either fails or retries n times

Good job setting the DAG to either fail or retry 3 times.

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

Well done!👏

DOWNLOAD PROJECT

3 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review