UDACITY

‹ Return to "Data Engineering Nanodegree" in the classroom

DISCUSS ON STUDENT HUB

Data Pipelines with Airflow

审阅

代码审阅 1

HISTORY

Meets Specifications

Congratulations on completing the project! You should be very proud of your accomplishments in building an ETL pipeline using AirFlow. Please keep learning 😊👏

General

DAG can be browsed without issues in the Airflow UI

DAG opens up as expected

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.

All tasks have a dependency and good work done.

Dag configuration

DAG contains default_args dict, with the following keys:

DAG contains default_args dict, with the following keys:

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

DAG contains default_args dict.

Especially the depends_on_past and catchup params are the ones that are great to understand to avoid unnecessary backfills when adding new DAGs. Nice job on configuring those args!

The DAG object has default args set

Default args are perfectly binded.

The DAG should be scheduled to run once an hour

perfect, DAG scheduled to hourly basis.

Staging the data

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

Good job on the staging operator, but you implemented only for JSON, you should also implement for Json files.

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

Nice and dynamic copy statement!

The operator contains logging in different steps of the execution

Suggestion
Could also add a log statement to inform that everything was successful

The SQL statements are executed by using a Airflow hook

Good, Hooks are the best and make DB operations easy.

Loading dimensions and facts

Dimensions are loaded with on the LoadDimension operator

Facts are loaded with on the LoadFact operator

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

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

The params to switch between modes is there, nice one!
Suggestion
You should make this as a boolean variable. For example:
mode=False,

Data Quality Checks

Data quality check is done with correct operator

The DAG either fails or retries n times

The operator now raises and error if the DQ test fails, good stuff!

Suggestion
It is important the the operator fails if the checks do not pass and logs the results in a visible manner so next steps can be taken to fix the issues.
Also the approach of first running all the checks and then collecting the results and checking the errors is a smart way to go and you can catch all the errors for pipeline at once!
You could also add add helpers separate class like sql_queries.py for the data quality check and collect all the statements in one place. This would keep the DAG and task clean and make it easy to add additional test by just changing a one file