Data Pipelines with Airflow

Meets Specifications

Brilliant Udacity Learner,

Congratulations! 🎉
You've successfully passed all the specification in this submission. I must admit that the structure of this project implementation is impressive! You should be proud of the work done as you seem to have a good hold on **Apache Airflow** and you've created your own custom operators to perform tasks such as staging the data, filling the data warehouse, and running checks on the data.

It was my pleasure reviewing this wonderful project. Please continue with this same spirit of hard work in the projects ahead.

Extra Materials

Below are some additional links to help you deepen your understanding on the related concepts:

- Airflow: Lesser Known Tips, Tricks, and Best Practises
- Airflow Tips, Tricks & Pitfalls
- Getting started with Apache Airflow
- DAG Writing Best Practices in Apache Airflow
- Scheduling Tasks in Airflow
- 88 companies who use Airflow in their tech stacks
- Apache Airflow Grows Up!
- Data Traffic Control with Apache Airflow

General

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

Awesome! DAG can be browsed without issues in the Airflow UI. 👏🏻

Suggestions

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. Depending on your set up, using a subdag operator could make your DAG cleaner.

- Using SubDAGs in Airflow
- Subdag Operator examples

✓ **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.**

Excellent work! The DAG's graph view all the task have a dependency and DAG begins with a `start_execution` task and ends with a `end_execution` task. 👍🏻

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 as required.

Notes

If a dictionary of default_args is passed to a DAG, it will apply them to any of its operators. This makes it easy to apply a common parameter to many operators without having to type it many times.

External Resources

External Resources

- Backfill and Catchup
- How to prevent airflow from backfilling dag runs?

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

Nice work! The DAG object contains a binding to the default args. 👏🏽

```
dag = DAG( . . .
          default_args=default_args
```

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

Good work scheduling the DAG to run once an hour as required.

`schedule_interval='0 * * * *'`

External Resources

- Airflow Scheduling & Triggers
- Scheduling Tasks in Airflow

Staging the data

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

Nice work! The stage operator contains a condition to test the file type if it is JSON or CSV. 👏🏽

Suggestions

I would also suggest adding a template field that would allow to load timestamped files from S3 based on the execution time and run backfills.

```
template_fields = ("s3_key",)
```

Extra Materials

- Templating and Macros in Airflow
- How to add template variable in the Operator task

✓ **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 as opposed to static SQL statements.

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

`logging.info` shows the status of staging load. Nice work! 👍🏽

```
self.log.info("Copying data from S3 to Redshift")
```

Suggestions

- Here is a nice documentation about Logging in Airflow.
- A nice discussion about Adding logs to Airflow Logs.

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

Good job connecting to the database via an Airflow hook. 👏🏽

```
aws_hook = AwsHook(self.aws_credentials_id)
credentials = aws_hook.get_credentials()
redshift = PostgresHook(postgres_conn_id=self.redshift_conn_id)
```

Suggestions

Hooks are interfaces to external platforms and databases, implementing a common interface when possible and acting as building blocks for operators.

- Automate AWS Tasks using Airflow Hooks
- Source code for airflow.hooks.S3_hook

Loading dimensions and facts

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

Splendid! There is a Separate functional operator for dimensions LoadDimensionOperator.

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

There is a separate functional operator for facts LoadFactOperator as well. 👏🏻

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

The parameters was used to add some dynamic functionality to the operators and allow to run various SQL statements instead of hard coded SQL statement. Well done! ⭐

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

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

Suggestions

Dimension loads are often done with the truncate-insert pattern where the target table is emptied before the load. Thus, having a parameter that allows switching between insert modes when loading dimensions. Fact tables are usually so massive that they should only allow append type functionality. That is why there is a need to check for parameter value passed and if it says delete before insertion, delete the values from table before insertion or append otherwise.

An ideal condition could be:

```
If operation truncate: Delete and then insert the values in the dimension table
If operation append: Only insert in the dimension table
```

Data Quality Checks

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

Great work! The operator that runs a check on the fact or dimension table(s) after the data has been loaded is DataQualityOperator. The data quality operator looks awesome. It is simple but still allows you to do import checks on the data and catch the possible data quality issues as soon as possible.

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

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**

Nice work! The Operator uses params to get the tests and the results and you've passed an array of checks and runs them in a loop. Well done! ⭐