Case study - Data Engineering

Hello

We're super happy you got there!

At this stage of our recruitment process we would like to present some bread and butter problems that you will definitely deal with working as a Data Engineer in Docplanner.

Deadline is up to you. But, **please share the date you are ready to discuss your solution as soon as you evaluate the task and your availability.** Remember to send the solution minimum 24 hours before the meeting, so we can prepare questions. If you spot any incomprehensible parts or you need clarifications, don't hesitate to contact us via email or schedule a call.

Task description

Your assignment will consist of a few tasks in data engineering scope. You will have an opportunity to prove yourself in DataOps, processing, data architecture and some hacking :) We assume that you are familiar with such technologies like: Docker, Airflow, AWS S3 and Postgres.

The main goal of this project is to obtain source data from climate observations, process it and load to a database in proposed data structure, all using Airflow.

Obligatory tasks

- Create a DAG in Airflow that will:
  - obtain data from the source, save raw to S3,
  - process, clean, prepare data in S3,
  - load data to the final tables in Postgres.
- Prepare data structure in relation to the data warehouse architecture good practices.
- Design your workflow as a daily process. Would it be better to load full set of data everytime or incrementally?

Additional tasks

- Change Airflow's default build-in metastore from Sqlite to Postgres or MySql (as a separate service in `docker-compose`),
- Write unit tests for your solution.

Data

The source data are daily climate observations from The Global Historical Climatology Network. - Endpoints for data are located here. - The Global Historical Climatology Network data is of the interest, thus additional endpoint parameter `?datasetid=GHCND` will be useful for almost all endpoints. - Data should be collected granularly by **station id** & **observation day** for **all available data categories**. - The amount of data might be overwhelming time- and download-wise if backfilling from ~1945. It's enough to fetch just a small subset. We don't expect whole dataset to be processed but a good architecture design.

Environment

The environment consists of dockerized Airflow, LocalStack AWS S3 and Postgres. Airflow, the workflow scheduler comes from an official image, almost untouched. LocalStack service provides an easy-to-use test/mocking framework for developing Cloud applications. Postgres used for data storage and DWH structure.

Requirements (tested on)

- Unix system (Linux, MacOS)
- Python 3.9
- IDE e.g. PyCharm
- Docker (if you use a computer with a new Apple Silicon processor you may want to select the "Use the new Virtualization framework" option that can be found in the "Experimental features" section of the Docker Desktop preferences)
- Git
- AWS CLI

Setup

This tutorial assumes that you want to setup the project in your home directory `~/`.

1. Change to your home directory `cd ~`
2. Clone repo `git clone git@github.com:DocPlanner/de-recruitment.git`
3. Switch to the project directory `cd de-recruitment`
4. Create virtualenv (if you don't have it yet) `python3 -m virtualenv venv/`
5. Activate `source venv/bin/activate`
6. Install needed packages `pip3 install -r requirements_dev.txt`
7. You are good to go. Have fun!

Run

See docker folder to run Airflow locally. 1. Switch to directory using `cd docker` 2. To run docker containers use `docker-compose up -d --build`. Airflow webserver will be accessible under `localhost:8080` after a while. The default user is `airflow` with password `airflow`. 3. To shut them down use `docker-compose down`

Tips

- If you have any issues - ask. Checking how you communicate with the team is an important part of the trial day.
- You can talk with us about any part of the task - how it works, architecture, code or env problems - you name it :)
- If you are out of time but feel that you could do something differently - leave a proper comment in the code.
- Final database schema doesn't have to contain only one dimension, create dictionary tables.
- Configuration Reference will be useful for Airflow configurations.
- Postgres is available on `localhost:5433` with user `postgres` and pwd `postgres`. You can connect it either via `psql -h localhost -p 5433 -U postgres` or through your preferred IDE. We've added it in Airflow as `docplanner_dwh` connection.
- You can interact with AWS S3 using awslocal that was installed in your python virtual env. We've added it in Airflow as `docplanner_aws` connection.
- You can add some example DAGs and connections to Airflow by setting the values of AIRFLOW__CORE__LOAD_EXAMPLES and AIRFLOW__CORE__LOAD_DEFAULT_CONNECTIONS to True in the .env file.

Good luck!