Orchestrate a batch ETL Data Pipeline with Airflow

Overview

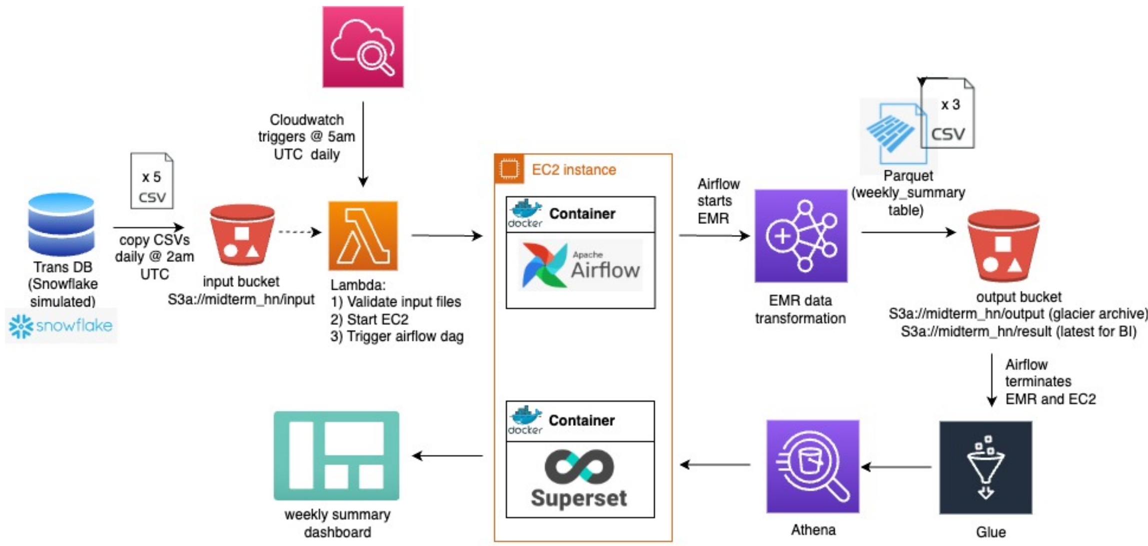
In this guide you will create an ETL (extract/transform/load) data pipeline from data ingestion through to a BI dashboard output. Apache Airflow is a popular open-source data engineering tool that can manage workflows, and specifically ETL/ELTs so we'll use Airflow for this orchestration. This is a great first end to end system for those new to data engineering to build.

This pipeline will be configured to run a daily batch load which is typical of taking transactional data in an (OLTP – Online Transaction processing) database and transforming it to analytical (OLAP – online analytical processing) data for consumption by a business for analysis and decision making.

Here, the input will be daily sales/inventory transactional tables coming from a relational database, the data will be transformed, stored in a data warehouse (S3 here), and then read to a BI intelligence dashboard.

External users will be able to view the dashboard to help make business decisions based on inventory metrics.

Architecture



The architecture above shows the workflow and technologies we will implement. We'll focus on AWS services although Azure and GCP could be used equivalently.

The high level daily workflow:

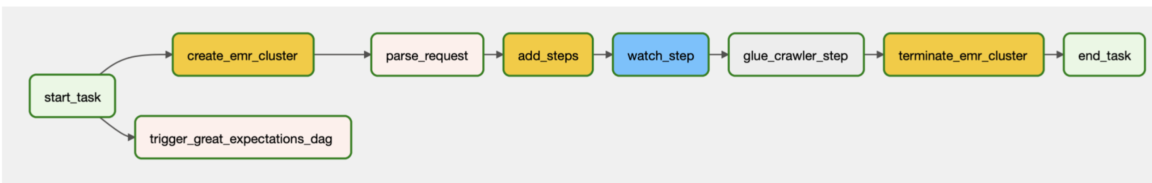
1. The OLTP database (simulated here by Snowflake) will send CSV formatted transactional data tables to an AWS S3 bucket daily @ 2am UTC.
2. Cloudwatch will trigger a lambda function daily at a later time (5am UTC) to kick-off the workflow by performing the functions:
   - Validate the input files (checks that files with today's date in the filename are loaded). If files are not complete then an email is sent indicating so and this will be the end of the workflow. Otherwise continue.
   - Start the EC2 instance if it is not already running.
   - Trigger the airflow dag.
3. The airflow workflow will then execute by:
   - starting the AWS EMR for the compute cluster,
   - the transformation will be performed on the data using Spark, and then the output saved to S3 (in parquet and .csv formats),

- then the AWS Glue Crawler will run to refresh the data in Athena (here used to simulate a data lake).
- EMR will be shutdown at the end of the dag workflow.

4. The user, often a business analyst, can view the output in Superset on a weekly summary dashboard.

Airflow Dags

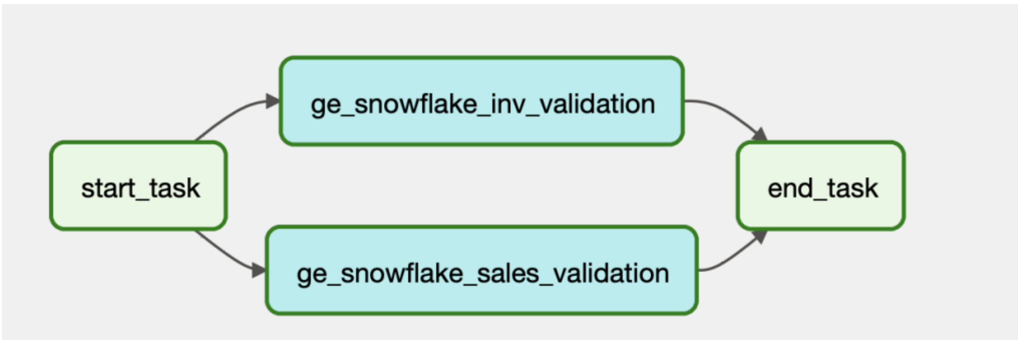
The main airflow dag follows the path below laid out with the following operators.



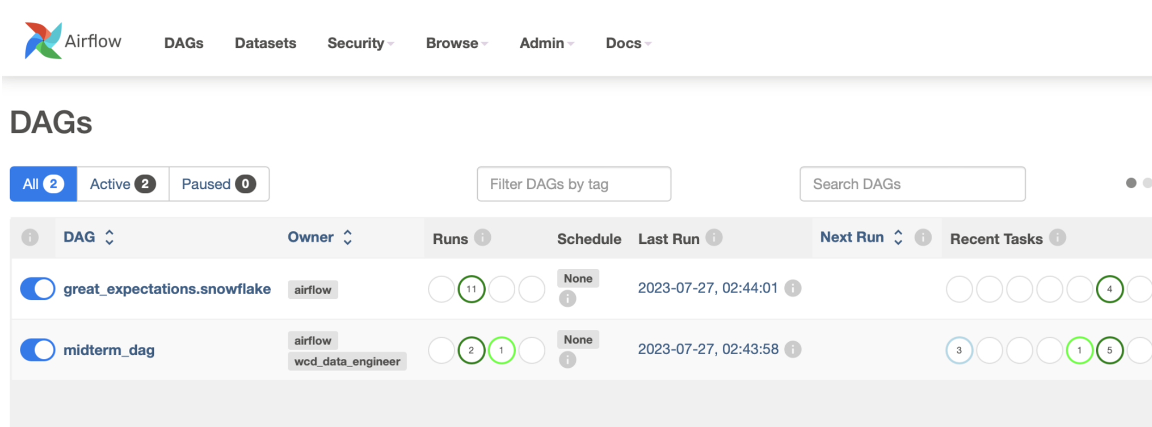
The dag operator steps here are:

1. **Start_task** – This is a dummy task marking the start of the workflow.
2. **Create_EMR_cluster** – The EMR cluster is created (code with 1 master and 2 cores, logging set to S3).
3. **Parse_request** – A python operator that retrieves the S3 input file data and stores this to airflow xcom (so that it can shared with another operator) .
4. **Add_steps** – Adds an EMR step to process the tranformation (passing in the transformation pyspark code file)
5. **Watch_step** – Waits for the previous step to complete. EMR takes a while to startup and run so except to wait a while (enough to go make coffee).
6. **Glue_crawler_step** – Runs the glue crawler to read the output data and update for Athena.
7. **Terminate_EMR_cluster** – As expected it stops and deletes the EMR cluster (saves cost as running EMR when not in use will run up costs).
8. **End_task** – dummy task marking end of the workflow.

You'll also notice that in parallel **trigger_great_expectations_dag** also runs. This is a bonus addition which runs Great Expectations unit test suites to perform data validation on the incoming data files. There are 2 operators here validating the main fact tables: Inventory (with ge_snowflake_inv_validation) and Sales (with ge_snowflake_sales_validation).



Here's a screenshot of the Airflow UI interface showing the 2 dags:



Installation and Setup

Now that you have an overview of the architecture, let's dive into the installation. Note the assumptions for these instructions are:

1. The transactional database (here simulated by Snowflake) will be setup already. You can set up what database you would like. We'll start at the point of the input files (here 5 CSV transaction tables) placed in the input S3 bucket. If you need addition instructions on setting up a simulated snowflake database contact me for them.
2. You have knowledge of how to use the AWS console and how to set AWS IAM roles and permissions to allow AWS components to communicate.

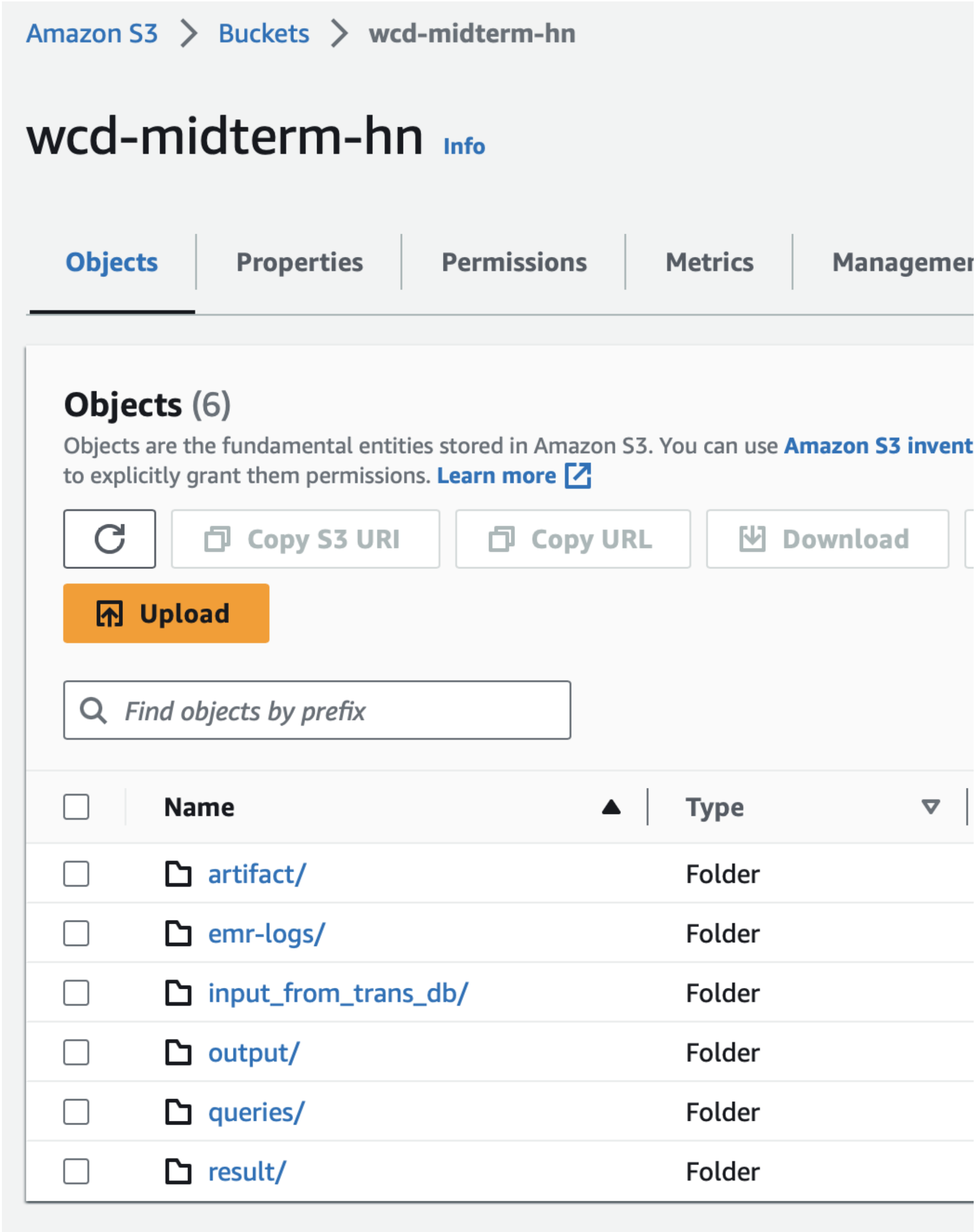
The files needed for this project can be found in **this GitHub repository < https://github.com/hil22/WCD/tree/fb2f03b4f2fb4c60e9f9d984e843647640a0b3b3/midterm>** .

a) Create the S3 bucket structure

Start by creating the follow S3 bucket and folders below, and populating with the files shown. The wcd_midterm-transform_sales_inventory_data.py file is in the GitHub respository, while the .csv files are the input files corresponding to your transactional database which are table named and end in a {date} in the form YY_MM_DD.

```
{S3_bucket_name}
|- artifact
        |- wcd_midterm-transform_sales_inventory_data.py
|- input_from_trans_db
        |- calendar_{date}.csv
        |- product_{date}.csv
        |- sales_{date}.csv
        |- store_{date}.csv
        |- inventory_{date}.csv
|- output
|- result
|- emr-logs
```

Here's a screenshot of my S3 bucket:



For reference, here is the schema for input table files, but you can replace these files with your own inputs:

Fact tables :

SALES	1.1M Rows
TRANS_ID	NUMBER(38,0)
PROD_KEY	NUMBER(38,0)
STORE_KEY	NUMBER(38,0)
TRANS_DT	DATE
TRANS_TIME	NUMBER(38,0)
SALES_QTY	NUMBER(38,2)
SALES_PRICE	NUMBER(38,2)
SALES_AMT	NUMBER(38,2)
DISCOUNT	NUMBER(38,2)
SALES_COST	NUMBER(38,2)
SALES_MGRN	NUMBER(38,2)
SHIP_COST	NUMBER(38,2)

INVENTORY	1.2M Rows
CAL_DT	DATE
STORE_KEY	NUMBER(38,0)
PROD_KEY	NUMBER(38,0)
INVENTORY_ON_HAND_QTY	NUMBER(38,2)
INVENTORY_ON_ORDER_QTY	NUMBER(38,2)
OUT_OF_STOCK_FLG	NUMBER(38,0)
WASTE_QTY	NUMBER(38,2)
PROMOTION_FLG	BOOLEAN
NEXT_DELIVERY_DT	DATE

Dimension tables :

STORE	151 Rows
STORE_KEY	NUMBER(38,0)
STORE_NUM	VARCHAR(30)
STORE_DESC	VARCHAR(150)
ADDR	VARCHAR(500)
CITY	VARCHAR(50)
REGION	VARCHAR(100)
CNTRY_CD	VARCHAR(30)
CNTRY_NM	VARCHAR(150)
POSTAL_ZIP_CD	VARCHAR(10)
PROV_STATE_DESC	VARCHAR(30)
PROV_STATE_CD	VARCHAR(30)
STORE_TYPE_CD	VARCHAR(30)
STORE_TYPE_DESC	VARCHAR(150)
FRNCHS_FLG	BOOLEAN

PRODUCT	1.2K Rows
PROD_KEY	NUMBER(38,0)
PROD_NAME	VARCHAR(16777216)
VOL	NUMBER(38,2)
WGT	NUMBER(38,2)
BRAND_NAME	VARCHAR(16777216)
STATUS_CODE	NUMBER(38,0)
STATUS_CODE_NAME	VARCHAR(16777216)
CATEGORY_KEY	NUMBER(38,0)
CATEGORY_NAME	VARCHAR(16777216)
SUBCATEGORY_KEY	NUMBER(38,0)
SUBCATEGORY_NAME	VARCHAR(16777216)

CALENDAR	10.6K Rows
CAL_DT	DATE
CAL_TYPE_DESC	VARCHAR(20)
DAY_OF_WK_NUM	VARCHAR(30)
DAY_OF_WK_DESC	VARCHAR(16777216)
YR_NUM	NUMBER(38,0)
WK_NUM	NUMBER(38,0)
YR_WK_NUM	NUMBER(38,0)
MNTH_NUM	NUMBER(38,0)
YR_MNTH_NUM	NUMBER(38,0)
QTR_NUM	NUMBER(38,0)
YR_QTR_NUM	NUMBER(38,0)

b) Set up the Lambda function

Create the lambda function through the AWS console. It will have 2 files : **lambda_function.py < https://github.com/hil22/WCD/blob/f5052258cefbe03049f18739ca755bc0047ada1a/midterm/lambda/lambda_function.py>** and **send_mail.py < https://github.com/hil22/WCD/blob/f5052258cefbe03049f18739ca755bc0047ada1a/midterm/lambda/send_email.py>**. You may want to update the references and logic in the code as it references the specific 5 input CSV files and checks they exist in the correct bucket before continuing.

c) Set up CloudWatch

Create a Cloudwatch schedule with a cron expression to trigger the lambda daily @ 5am. If you're not familiar with Cloudwatch, here's a **tutorial video < https://www.youtube.com/watch?v=-v4LMV5DAD4>** on how to set this.

d) Set up the EC2 instance

- Create an EC2 with instance type at least T2.large and Amazon Linux installation (to follow the commands listed here, or your own preference and adjust the install packages).

- On the EC2 instance create the following file structure populating with files from the Guthub repo:

```
{project_dir_name}
|- docker-compose.yaml
|- docker_boot.service
|- Dockerfile
|- dags
        |- midterm_dag.py
        |- snowflake_ge.py
|- Superset
```

- Install Docker using the **AWS instructions here < https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html>** .
- Install Docker-compose. Here are the instructions for amazon linux which uses the yum package installer (or for linux ubuntu use apt).

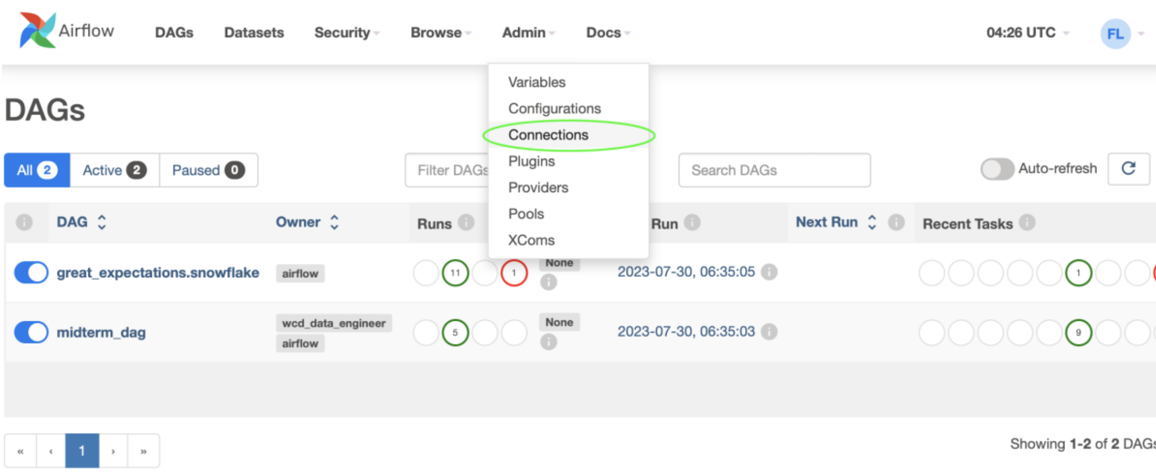
```
> sudo yum update
> sudo yum upgrade
> sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
> sudo chmod +x /usr/local/bin/docker-compose
> docker-compose version
```

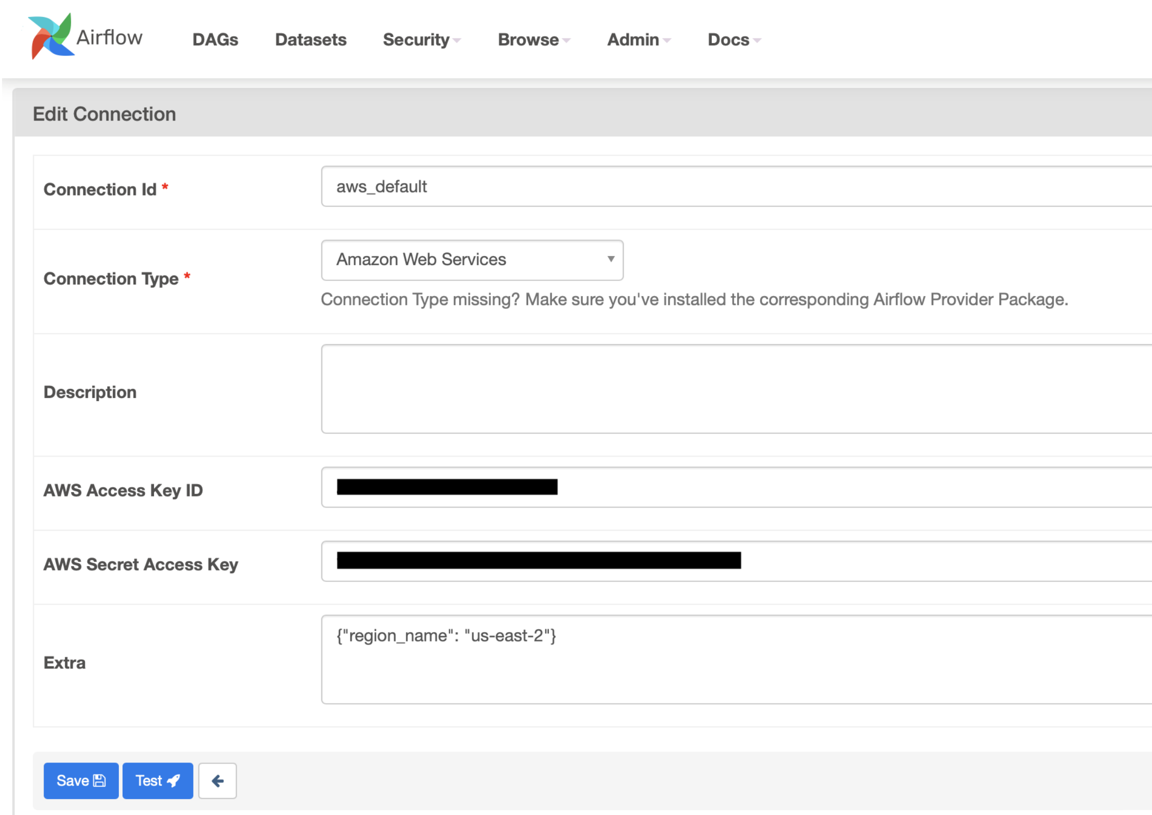
e) Install and setup Airflow

- From the main project folder, follow the code below to install Airflow with the docker-compose file, then update the database, initialize the database, and create a login,

```
> docker-compose up -d
> sudo docker exec -it airflow_lab_webserver_1 airflow db upgrade
> sudo docker exec -it airflow_lab_webserver_1 airflow db init
> sudo docker exec -it airflow_lab_webserver_1 airflow users create -u admin -p admin -f firstname -l lastname -r Admin -e admin@airflow.com
```

- Validate that airflow is running by listing the running containers with **docker ps**.
- Open the airflow UI by opening a web browser viewing port 8080 i.e. URL **{EC2_instance_public_ip}:8080**.
- Add an xcom connector for the AWS connection named aws_default. This will contain the access key and secret key to allow Airflow to connect to the services in your AWS user account. Screenshot below of where in airflow UI to add a connector, and then the entry (scratching out my private info).





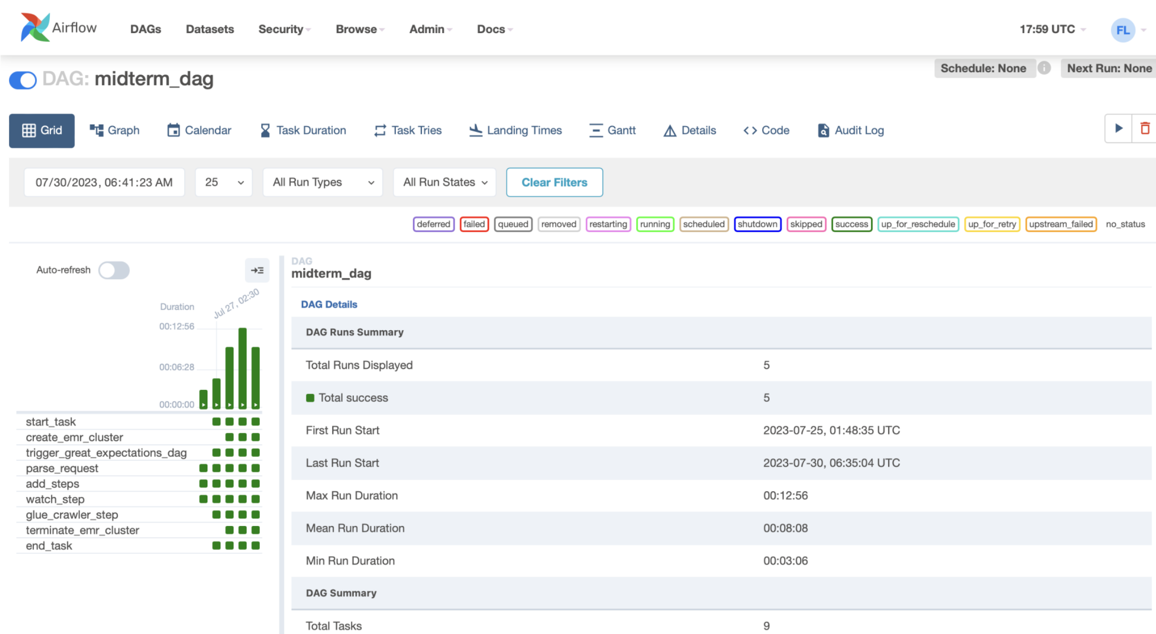
f) Test the workflow so far

Run the workflow to ensure everything created so far works, mainly that the transformation executes and generates the desired output file tables.

1. Trigger the input database to send input files to the S3 input folder with today's date.
2. Trigger a lambda test run.

3. Validate the s3 **result** and **output** folders are populated with the correct files. (In my example this would be weekly_summary_{date}.csv, calendar_{date}.csv, product_{date}.csv, store_{date}.csv

In airflow you should see the Operator steps all turn green as they execute like below.



g) Athena and Glue

Now that we have the output files correctly in the data warehouse now, we can run the Glue Crawler over the **result** folder to feed into Athena. AWS instructions to do so **here < https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html#:~:text=On%20the%20AWS%20Glue%20service,Data%20Crawler%20%2C%20and%20choose%20Next.>** .

The output schema if you used my GitHub transformation code (and feel free to replace with your own logic in the **wcd_midterm-transform_sales_inventory_data.py < https://github.com/hil22/WCD/blob/b454a11d324d6ccf55f46266458 99eec55998601/midterm/transformation/wcd_midterm-transform_sales_inventory_data.py>** file) will look like this:

Schema (16)

View and manage the table schema.

Filter schemas

#	Column name	Data type
1	wk_timestamp	timestamp
2	store_key	string
3	prod_key	string
4	total_sales_qty	double
5	total_sales_amount	double
6	avg_sales_price	double
7	total_sales_cost	double
8	stock_level_eow	double
9	order_level_eow	double
10	percent_in_stock	double
11	total_low_stock_impact	double
12	potential_low_stock_impact	double
13	no_stock_impact	double
14	low_stock_instances	bigint
15	no_stock_instances	double
16	weeks_on_hand_stock_can...	double

h) Install Superset

- Now to install Superset which is the business intelligence dashboard tool we'll use to view the data. Go into the superset folder and create a file named **Dockerfile** containing the following lines.

```
FROM apache/superset:9fe02220092305ca8b24d4228d9ab2b6146afed6

USER root
RUN pip install "PyAthena>1.2.0"

USER superset
```

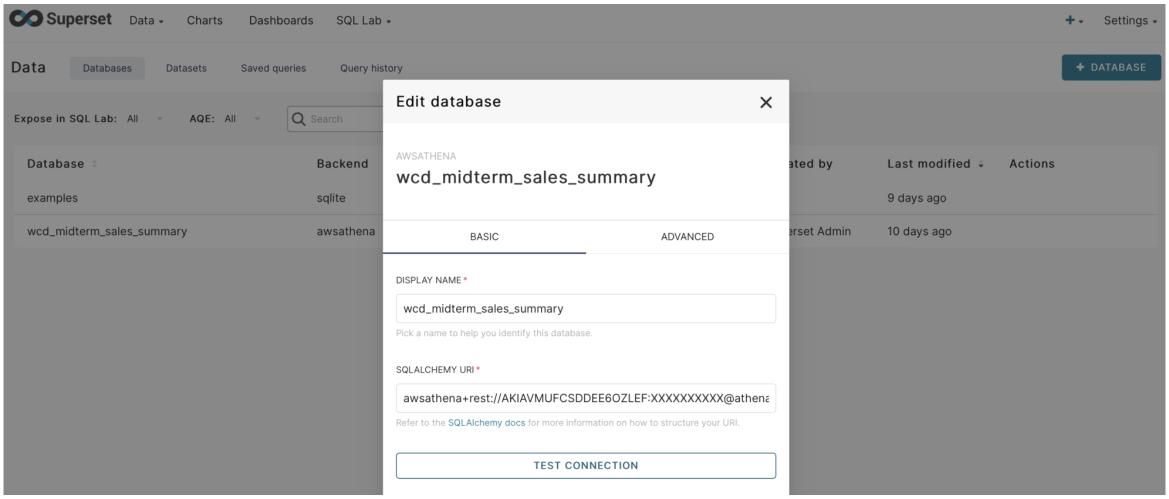
- Now deploy the Superset image and set a local admin account

> ```
> > docker run -d p 8088:8088 --name superset apache/superset
> > docker exec -it superset superset fab create-admin \
>                --username admin \
>                --firstname Superset \
>                --lastname Admin \
>                --email admin@superset.com \
>                --password admin
> > docker exec -it superset superset db upgrade
> > docker exec -it superset superset init
> ```

- In Superset connect to the Athena database using the following connection string

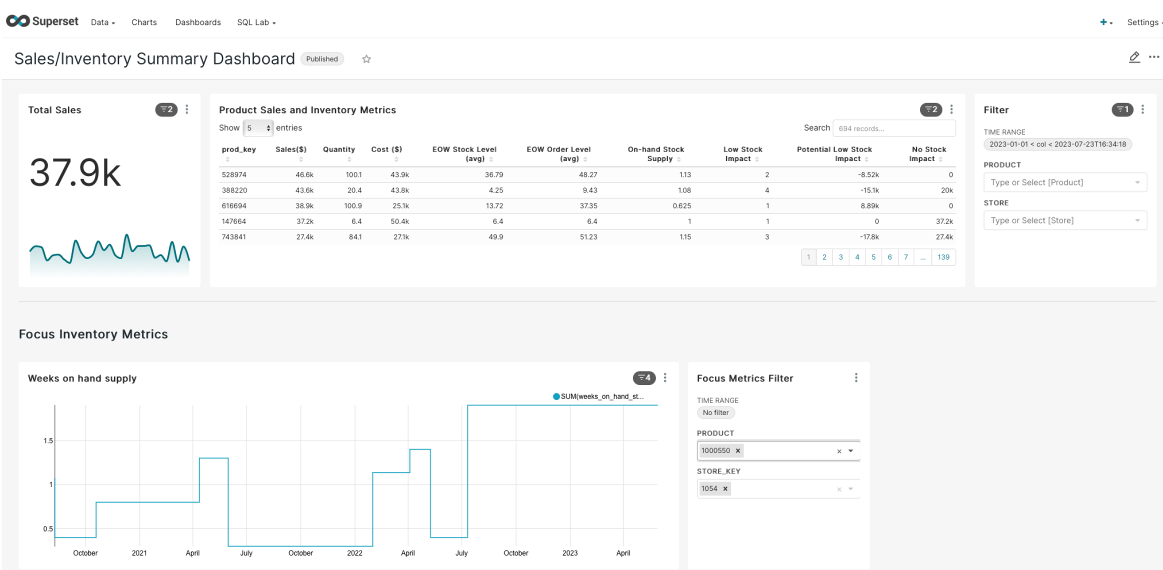
```
awsathena+rest://{aws_access_key_id}:
{aws_secret_access_key}@athena.
{region_name}.amazonaws.com/{schema_name}?s3_staging_dir=
{s3_staging_dir}&work_group=primary
```

Here's a screenshot where to edit the database connection string in Superset:



I'll let you figure out how to build a Superset dashboard as it's fairly straightforward and will probably just take a day to figure out. Any other Business Intelligence tool can also be used.

Here's a view of the Superset dashboard I created which you can mimic and/or expand upon:



i) Run the workflow

Now we have completed the creation of a ETL data pipeline. Let the workflow run off the scheduler for a few days to ensure all works correctly.

Conclusion

Congratulations on complete a full ETL data pipeline! I'm sure there are will be many small issues and AWS permissions to correct along the way which makes it all the more rewarding when it finally works. Feel free to contact me for assistance if you're stuck.

Bonus: Add data validation with Great Expectations

Install and run great expectations:

- From project directory {project_dir_name} create a gx dir (**mkdir gx**)
- Copy **Docker.gx < https://github.com/hil22/WCD/blob/b454a11d324d6ccf55f4626645899eec55998601/midterm/docker/Dockerfile.gx>** into the gx dir
- Build and run the image from the gx dir:

```
> docker build -f Dockerfile.gx . -t gx/gxdemo:local
> docker run -it -v $(pwd):/app -p 8888:8888 gx/gxdemo:local
```

Create the new test suites:

```
> docker exec -it bash
in container> great_expectations suite new
```

In the interactive questions that you're prompted for use the following:

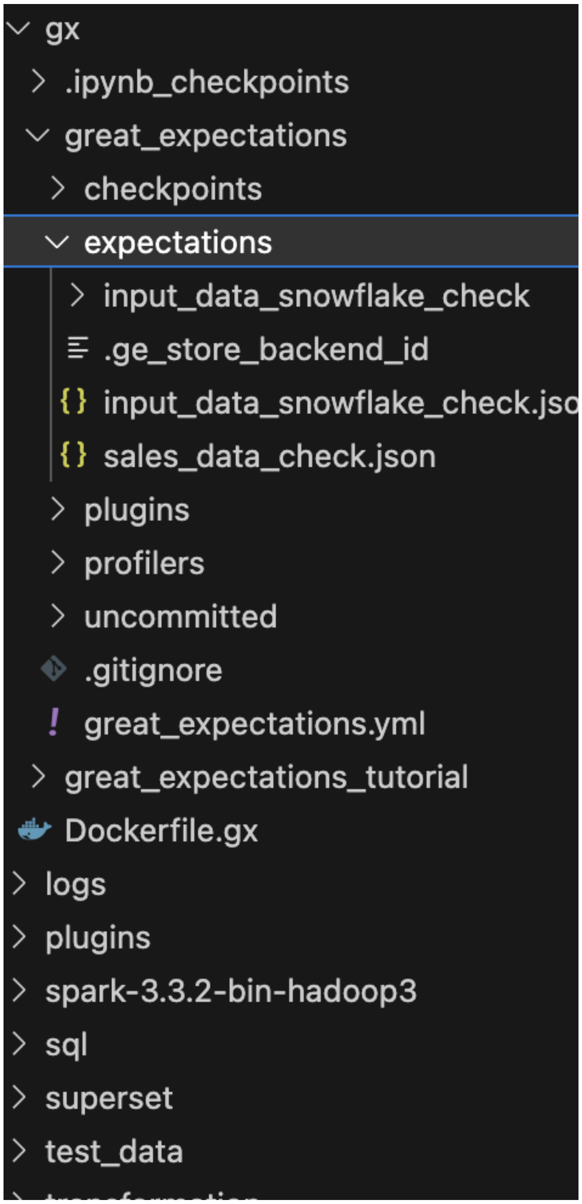
name: input_data_snowflake_check
select 1 for manual

Repeat to create the **sales_data_check** suite

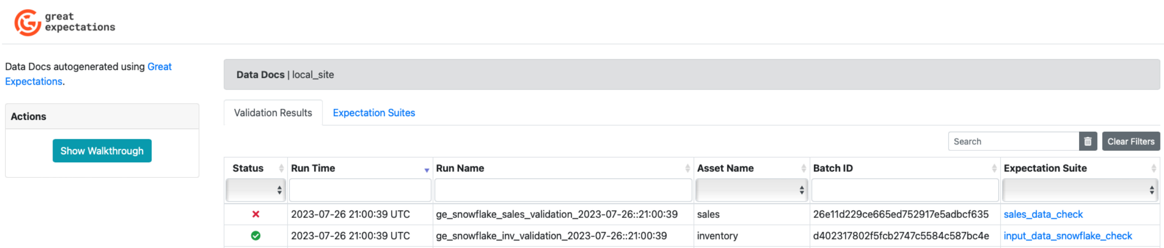
Copy over the pre-made test suites (or just edit the Jupyter notebook suites if you prefer):

1. Change directory to **gx/great_expectations/uncommitted**. This is where the jupyter notebook files are for the GE suites.
2. Replace the new suite files with the github files: [edit_input_data_snowflake_check.ipynb <https://github.com/hil22/WCD/blob/b454a11d324d6ccf55f4626645899eec55998601/midterm/gx_suites/edit_input_data_snowflake_check.ipynb>] and [edit_sales_data_check.ipynb <https://github.com/hil22/WCD/blob/b454a11d324d6ccf55f4626645899eec55998601/midterm/gx_suites/edit_sales_data_check.ipynb>].
3. In jupyter notebook run the 2 suite files which generates the json code suite file.

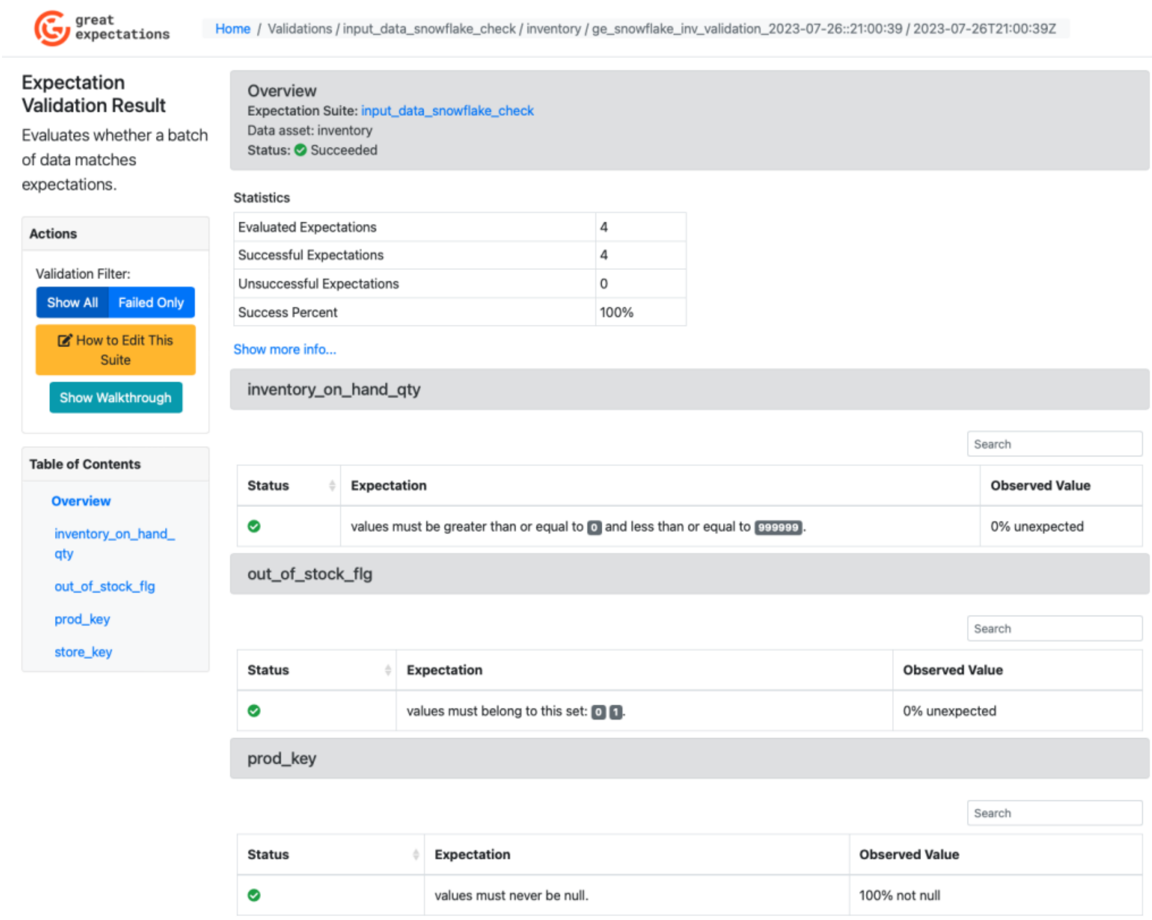
Here's what the directory structure will look like. In the **expectations** folder there will be two json files which will be the unit test code used by Great Expectations (**input_data_snowflake_check.json** and **sales_data_check.json**).



Now you can rerun the airflow pipeline including the **trigger_great_expectations_dag**. In the **gx/uncommitted** directory there will be a **data_docs** folder. Download this folder and view the contained **index.html** file. Here is what the Great expectations main test result page looks like showing the 2 test suites:



and drilling into one of the test suites the result looks like:



 Up ↑