Data Engineer Take Home Assessment

Name	**Jia Then**
Position	Data Engineer

Github Repository

For more detailed solutions and commit review, kindly refer to https://github.com/jia-von and I will grant access to me repository.

```
git clone https://github.com/jia-von/data_engineer_take_home_assessment.git

cd data_engineer_take_home_assessment
```

Directory Structure

```
data_engineer_take_home_assessment
|-- architecture
|-- figures (this directory contains figures for documentation purposes)
|-- scripting
    |-- farm_api.json (JSON object file)
    |-- farm_function.py (Python script function)
    |-- scripting_solution_human_readable.md
|-- SQL
    |-- sql_solution_human_readable.md
    |-- sql_solution.sql (SQL queries)
```

SQL

All solutions to the SQL assessment are found in directory `SQL`. You may copy and paste queries from `sql_solution.sql` in a PostgreSQL database.

Scripting

All solutions for the scripting assessment are found in `scripting` directory. You may test the Python script.

```
$ cd scripting

# To run the script farm_function.py
# On Ubuntu you may need to use python3 command
$ python farm_function.py
```

Architecture

`architecture` directory contains no code therefore no commands or setup required.

This take home assignment has 3 parts: a SQL question, a scripting question and a high-level design question. Please complete all three questions.

SQL

Use SQL to solve this problem.
We are evaluating this from the lens that this is your best possible code.

Description
Attribution Data
Given Account and Source tables, write SQL scripts to transform:

1. When an account*'s utm_na*me = 'utmcsr', output its *utm_value* in a column called *utm_source*
2. When an account's *utm_name* = 'utmcmd', output its *utm_value* in a column called *utm_medium*

Input tables

accounts	
account_id	account_name
226	aaa
971	bbb
598	ccc
999	ddd

source		
account_id	utm_name	utm_value
226	utmcsr	facebook
226	utmcmd	cpc
971	utmcsr	google
971	utmcmd	cpc
598	utmcmd	cpc
598	utmcsr	google
999	utmcsr	twitter

Expected Output

account_name	utm_source	utm_medium
aaa	facebook	cpc
bbb	google	cpc
ccc	google	cpc
ddd	twitter	

SQL Assignment

Use SQL to solve this problem. We are evaluating this from the lens that this is your best possible code.

Problem:

Given Account and Source tables, write SQL scripts to transform: 1. When an account's utm_name = 'utmcsr', output its utm_value in a column called utm_source 2. When an account's utm_name = 'utmcmd', output its utm_value in a column called utm_medium

Setup:

PostgreSQL syntax and snake_case is used as nomenclature for this exercise.

1. PostgreSQL database was deployed locally to recreate the tables provided by Jobber.
2. Create tables 'accounts' and 'sources'.
3. Insert data into the tables.
4. Create primary and foreign key. Note: No explicit instruction was given in the exercise in regards to making relational database. Therefore an assumption was made based on one to many relationship, from 'accounts' to 'sources'.
5. Create a SQL query that will produce an output similar to 'expected output'. Note: No explicit instruction that I would need to create an 'expected output' table. Therefore, the solution for this exercise is a SQL query.

Solution

Re-create the table named 'accounts' and 'sources'

```sql
CREATE TABLE public.accounts (
    account_id int NULL,
    account_name varchar NULL
);

CREATE TABLE public.sources (
    account_id int NULL,
    utm_name varchar NULL,
    utm_value varchar NULL
);
```

Insert data into tables.

```sql
INSERT INTO public.accounts (account_id,account_name)
    VALUES (226,'aaa');
INSERT INTO public.accounts (account_id,account_name)
    VALUES (971,'bbb');
INSERT INTO public.accounts (account_id,account_name)
```

```
    VALUES (598,'ccc');
INSERT INTO public.accounts (account_id,account_name)
    VALUES (999,'ddd');

INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (226,'utmcsr','facebook');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (226,'utmcmd','cpc');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (971,'utmcsr','google');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (971,'utmcmd','cpc');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (598,'utmcmd','cpc');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (598,'utmcsr','google');
INSERT INTO public.sources (account_id,utm_name,utm_value)
    VALUES (999,'utmcsr','twitter');
```

Add primary and foreign keys.

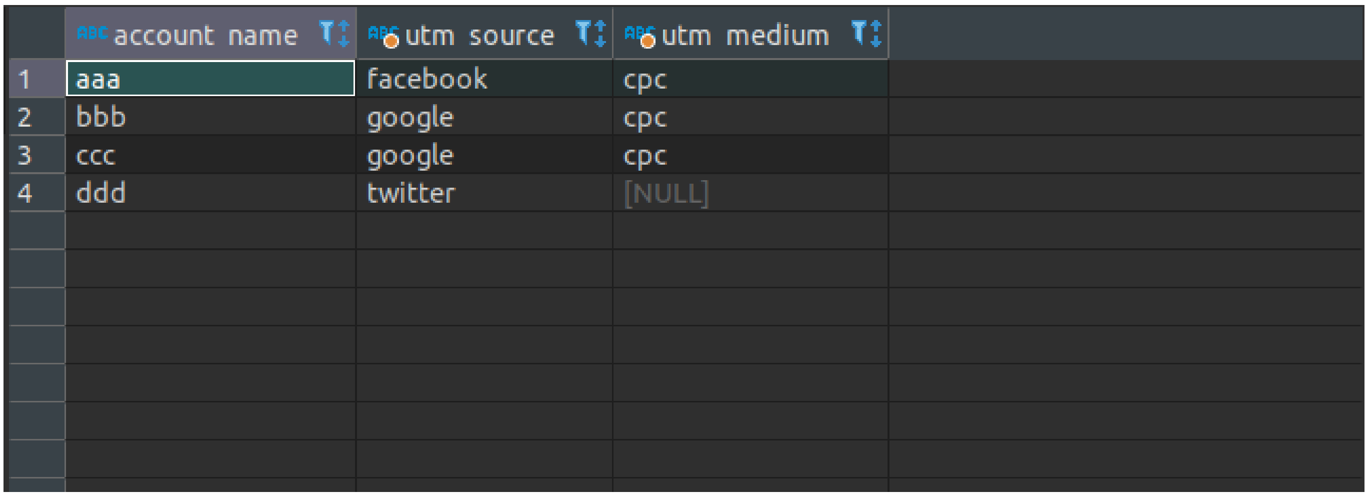
```
ALTER TABLE public.accounts ADD CONSTRAINT accounts_pk PRIMARY KEY
(account_id);

ALTER TABLE public.sources ADD CONSTRAINT sources_fk FOREIGN KEY
(account_id) REFERENCES public.accounts(account_id);
```

SQL queries to re-create 'expected output'

```
SELECT DISTINCT account_name, utm_source, utm_medium
FROM
    (
    SELECT utm_s.account_id, utm_source, utm_medium
    FROM
        (
        -- When an account's utm_name = 'utmcsr', output its utm_value in a
column called utm_source
        -- Alias, 'AS' was used to assign 'utm_value' with temporary name
'utm_source'. Refer: https://www.postgresqltutorial.com/postgresql-
tutorial/postgresql-column-alias/
        SELECT account_id, utm_value AS utm_source
        FROM sources
        WHERE utm_name = 'utmcsr'
        ) AS utm_s
        -- Since account with the name account_name 'ddd' or account_id
'999' do not have 'utmcmd' row, temporary tables 'utm_s' and 'utm_m' will
have none matching rows.
        -- Therefore, FULL OUTER JOIN was used to set NULL values for every
column of the table that does not have the matching row
```

```
        -- Refer: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-full-outer-join/
FULL OUTER JOIN
        (
        -- When an account's utm_name = 'utmcmd', output its utm_value in a column called utm_medium
        SELECT account_id, utm_value AS utm_medium
        FROM sources
        WHERE utm_name = 'utmcmd'
        ) AS utm_m
        ON utm_s.account_id = utm_m.account_id
    ) AS exp_out
LEFT JOIN accounts ON exp_out.account_id = accounts.account_id
```

Screenshot of output



	account_name	utm_source	utm_medium
1	aaa	facebook	cpc
2	bbb	google	cpc
3	ccc	google	cpc
4	ddd	twitter	[NULL]

Scripting

Use any scripting programming language to solve this problem (eg. JS, Python, etc.). Demonstrate with tests that your solution is correct.
We are evaluating this from the lens that this is your best possible code.

Description
Agriculture company
Your objective is to transform data coming from the Farm API before loading it into your database.
The data from the Farm API contains 2 fields:

- name
- inventory

Write a script function that should receive an array of JSON objects containing Farm API data. The function should return an array of JSON with *inventory* values transformed as shown below and added to a field called *inventory_level*:

- 0 or less: None
- 1 or 2: Low
- 3 or more: Normal

Example
Input:
[{name: 'Pig', inventory: 3}, {name: 'Cow', inventory: 4}, {name: 'Chicken', inventory: -1}, {name: 'Dog', inventory: 1}]

Output:
[{name: 'Pig', inventory_level: 'Normal'}, {name: 'Cow', inventory_level: 'Normal'}, {name: 'Chicken', inventory_level: 'None'}, {name: 'Dog', inventory_level: 'Low'}]

Scripting

Setup:

1. Change directory into `scripting` so you can execute `farm_function.py` script function.
2. `farm_api.json` was created to replicate an array of JSON objects containing Farm API data.
3. Example command to execute `farm_function.py` in shell:

```
python farm_function.py
```

Python script function

Create a function that accepts arguments, JSON object as strings.

```
def api_input(json_object_input):

    # Parse JSON string and this will result in Python dictionary data. Refer: https://www.w3schools.com/python/python_json.asp
    api_output = json.loads(json_object_input)

    # Calculate the length of an array to be used with the 'while' loop command below. Dynamically generated values allow functions to be reused.
    row_number = len(api_output)

    # Use 'while' loop to iterate through the array and checking values in "inventory" that satisfies these conditions. 0 or less: None, 1 or 2: Low, 3 or more: Normal
    i = 0
    while i < row_number:

        # Since 'api_output' is a dictionary, I use the built-in method 'get()' to obtain value from "inventory" key and set the values to 'int' data type.
        # Refer: https://www.w3schools.com/python/python_dictionaries_methods.asp
        x = int(api_output[i].get("inventory"))

        # I use 'if statement' to check whether the "inventory" values are 0 or less, 1 or 2, 3 or more.
        # If condition is true, use 'update()' method to insert new key-value pairs and remove specified key, "inventory".
        if x <= 0:
            api_output[i].update({"inventory_level": "None"})
            api_output[i].pop("inventory")
            i += 1
        elif x >= 3:
            api_output[i].update({"inventory_level": "Normal"})
            api_output[i].pop("inventory")
            i += 1
        else:
            api_output[i].update({"inventory_level": "Low"})
            api_output[i].pop("inventory")
            i += 1

    # Convert Python objects into JSON string using 'json.dumps()' method
and format the string with idents to produce easy to read format.
    # Refer: https://www.w3schools.com/python/python_json.asp
    print(json.dumps(api_output, indent=4))
```

Example 1.

The use of `api_input()` function with the given JSON objects.

- JSON objects: `[{"name": "Pig", "inventory": 3}, {"name": "Cow", "inventory": 4}, {"name": "Chicken", "inventory": -1}, {"name": "Dog", "inventory": 1}]`

```
api_input('[{"name": "Pig", "inventory": 3}, {"name": "Cow", "inventory":
4}, {"name": "Chicken", "inventory": -1}, {"name": "Dog", "inventory":
1}]')
```

Output:

```
Example 1. output:
[
    {
        "name": "Pig",
        "inventory_level": "Normal"
    },
    {
        "name": "Cow",
        "inventory_level": "Normal"
    },
    {
        "name": "Chicken",
        "inventory_level": "None"
    },
    {
        "name": "Dog",
        "inventory_level": "Low"
    }
]
Press Enter to continue...
```

Example 2.

The use of `api_input()` function with two additional objects.

- JSON objects: `[{"name": "Pig", "inventory": 3}, {"name": "Cow", "inventory": 4}, {"name": "Chicken", "inventory": -1}, {"name": "Dog", "inventory": 1}, {"name": "Donkey", "inventory": 6}, {"name": "Cat", "inventory": 4}]`

```
api_input('[{"name": "Pig", "inventory": 3}, {"name": "Cow", "inventory":
4}, {"name": "Chicken", "inventory": -1}, {"name": "Dog", "inventory": 1},
{"name": "Donkey", "inventory": 6}, {"name": "Cat", "inventory": 4}]')
```

Output:

```
Example 2. output:
[
    {
        "name": "Pig",
        "inventory_level": "Normal"
    },
    {
        "name": "Cow",
        "inventory_level": "Normal"
    },
    {
        "name": "Chicken",
        "inventory_level": "None"
    },
    {
        "name": "Dog",
        "inventory_level": "Low"
    },
    {
        "name": "Donkey",
        "inventory_level": "Normal"
    },
    {
        "name": "Cat",
        "inventory_level": "Normal"
    }
]
Press Enter to continue...
```

Example 3.

The function below uses JSON object file instead of string.

- File input: `farm_api.json`.

```
def api_input_json_file(api_file):

    # Assuming the file is saved in the same directory as farm_function.py
script. I open the file using 'open()' function.
    farm_api_file = open(str(api_file), "r")

    # Re-using 'api_input()' function defined previously.
    api_input(farm_api_file.read())
```

Example use of `api_input_json_file()`:

```
api_input_json_file("farm_api.json")
```

Output:

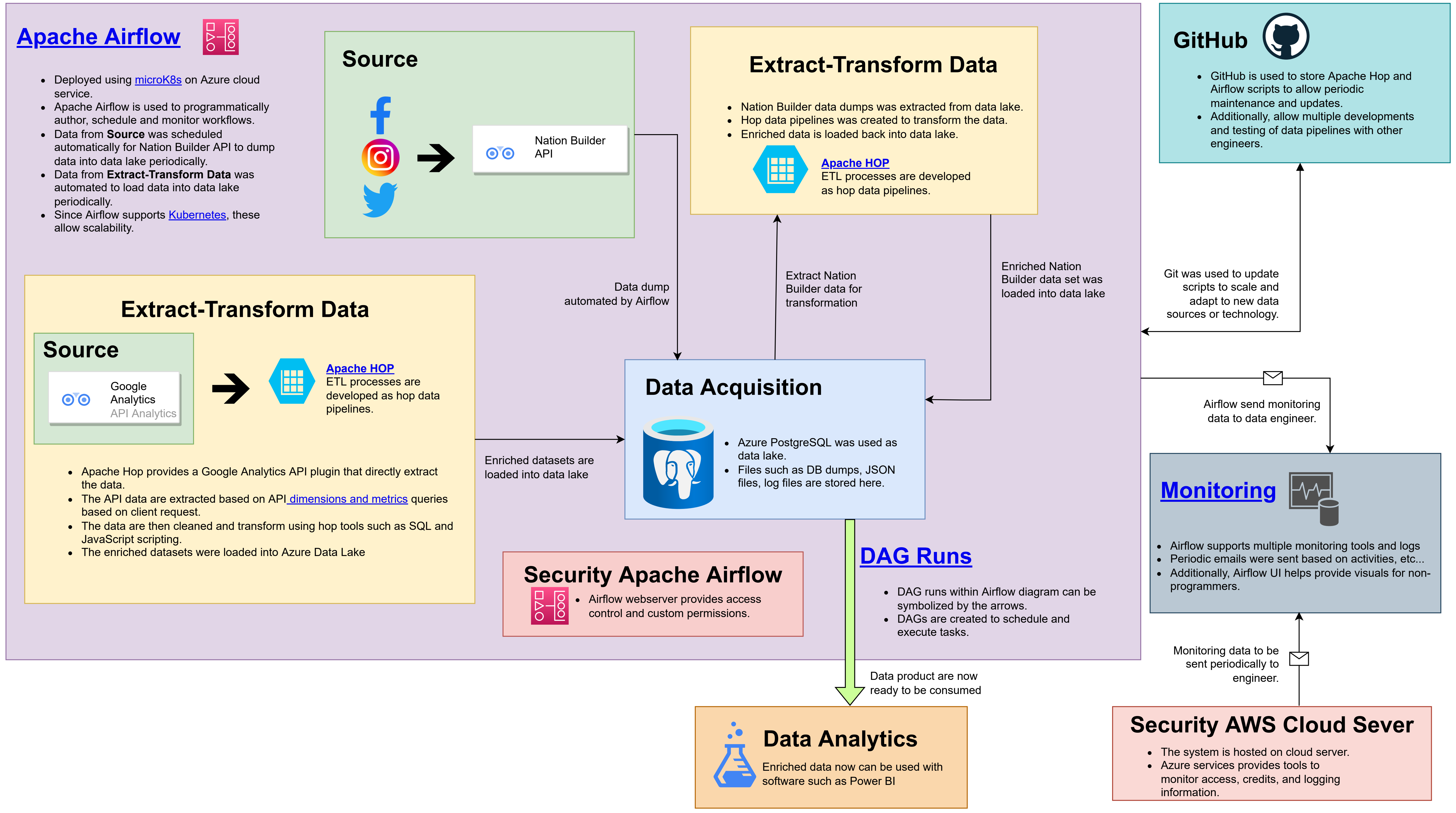
```
Example 3. output:
[
    {
        "name": "Pig",
        "inventory_level": "Normal"
    },
    {
        "name": "Cow",
        "inventory_level": "Normal"
    },
    {
        "name": "Chicken",
        "inventory_level": "None"
    },
    {
        "name": "Dog",
        "inventory_level": "Low"
    }
]
Press Enter to end...
```

High Level Design / Architecture

Modern applications generate a large amount and variety of data, across many systems. This data, if harnessed, can be a strategic and tactical asset. Please speak to the design of a data pipeline that would facilitate getting value from such data. Some elements we would like to hear about include:

- Extraction, transformation, loading and integration
- Enrichment and extension (semantic layer)
- Mitigating risk / single source of truth
- Scalability, availability and security
- Orchestration / workflow / performance
- Monitoring (failure, optimization, etc.)

A high level design is what we’re looking for here, but it would be great if you could include any relevant technologies you have experience with.


Source
Nation Builder API
Source
Google Analytics
API Analytics
Data dump automated by Airflow
Extract Nation Builder data for transformation
Enriched Nation Builder data set was loaded into data lake
Enriched datasets are loaded into data lake
Git was used to update scripts to scale and adapt to new data sources or technology.
Airflow send monitoring data to data engineer.
Monitoring data to be sent periodically to engineer.
Data product are now ready to be consumed

Apache Airflow

- Deployed using microK8s on Azure cloud service.
- Apache Airflow is used to programmatically author, schedule and monitor workflows.
- Data from **Source** was scheduled automatically for Nation Builder API to dump data into data lake periodically.
- Data from **Extract-Transform Data** was automated to load data into data lake periodically.
- Since Airflow supports Kubernetes, these allow scalability.

Extract-Transform Data

- Nation Builder data dumps was extracted from data lake.
- Hop data pipelines was created to transform the data.
- Enriched data is loaded back into data lake.

Apache HOP
ETL processes are developed as hop data pipelines.

GitHub

- GitHub is used to store Apache Hop and Airflow scripts to allow periodic maintenance and updates.
- Additionally, allow multiple developments and testing of data pipelines with other engineers.

Extract-Transform Data

Apache HOP
ETL processes are developed as hop data pipelines.

- Apache Hop provides a Google Analytics API plugin that directly extract the data.
- The API data are extracted based on API dimensions and metrics queries based on client request.
- The data are then cleaned and transform using hop tools such as SQL and JavaScript scripting.
- The enriched datasets were loaded into Azure Data Lake

Data Acquisition

- Azure PostgreSQL was used as data lake.
- Files such as DB dumps, JSON files, log files are stored here.

Monitoring

- Airflow supports multiple monitoring tools and logs
- Periodic emails were sent based on activities, etc...
- Additionally, Airflow UI helps provide visuals for non-programmers.

Security Apache Airflow

- Airflow webserver provides access control and custom permissions.

DAG Runs

- DAG runs within Airflow diagram can be symbolized by the arrows.
- DAGs are created to schedule and execute tasks.

Data Analytics

Enriched data now can be used with software such as Power BI

Security AWS Cloud Sever

- The system is hosted on cloud server.
- Azure services provides tools to monitor access, credits, and logging information.