Random User Rest API - ETL Pipeline - Design Document

ETL Pipeline Development Steps:

1. Extraction Layer : Extraction layer's deliverable will be a python class module that will accept 2 parameters [first parameter will be the URL of the Rest API that needs to be queried and the second parameter would be, how many user records to retrieve].

2. Transform Layer : Transform layer which is a python class module will accept the Extracted Raw json data from the Extraction Layer and the necessary transformations, flattening the nested json fields etc will be handled inside this transformation layer

3. Load Layer : Load Layer, a python class module will accept the transformed/flattened data and write this data out to a CSV file. The Destination location this CSV file will be stored will again vary based on the requirement. This could mean storing the csv file in AWS s3 or in a local disk and this can vary based on the business requirements and future use cases.

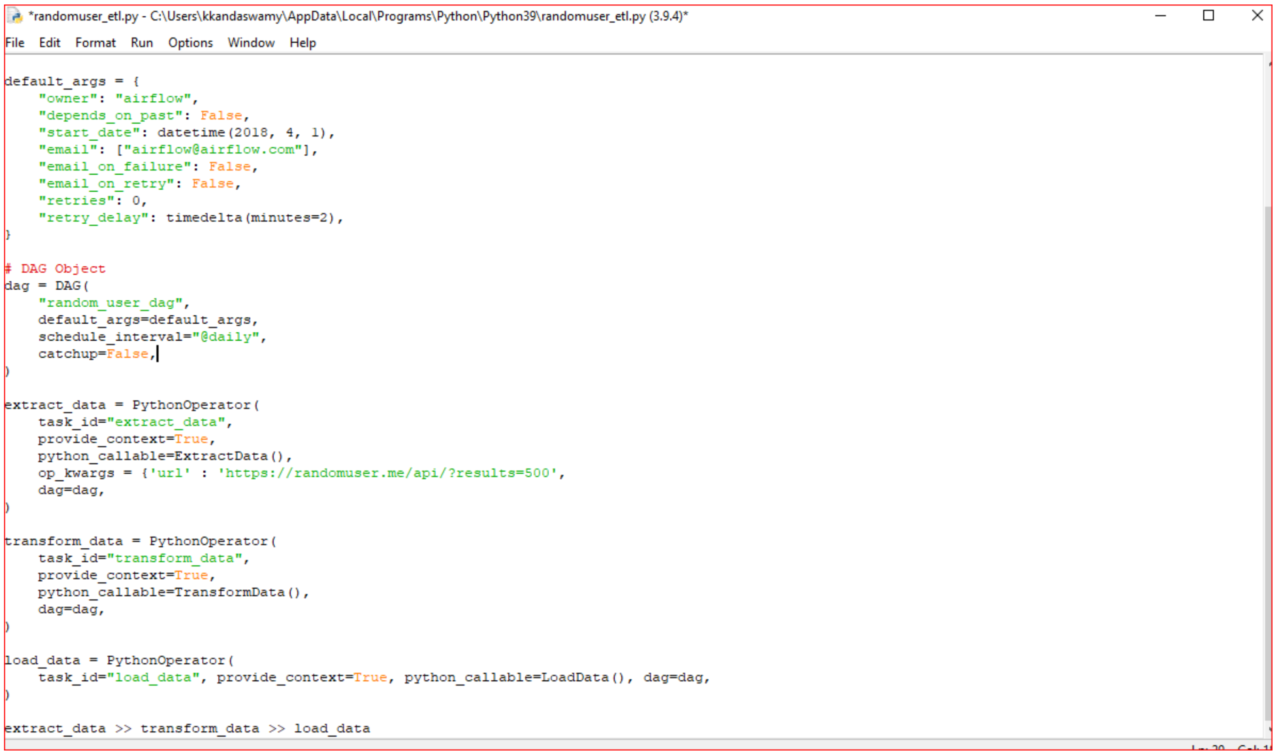
ETL Orchestration/Scheduling:

Orchestrating/Automating/Scheduling the above ETL pipeline that processes raw json data extracted from RandomUser.me API, can be achieved by using the Apache Airflow

Setting up an Airflow Workflow, involves creating a DAG, that will run a set of Airflow tasks, at the specified interval, in the order it's configured to run inside of the DAG.

To Automate the above ETL pipeline, we will have to create a new airflow DAG that will have a set of individual tasks that will call each of the (Extract, Transform, Load python modules) ETL modules in the order specified.

We can pass in the parameters for each of the ETL layers using airflow's op_kwargs and leverage the PythonOperator and xcom(xcom push/pull) for passing data between Airflow tasks


randomuser_etl.py - C:\Users\kkandaswamy\AppData\Local\Programs\Python\Python39\randomuser_etl.py (3.9.4)
File Edit Format Run Options Window Help

```python
default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2018, 4, 1),
    "email": ["airflow@airflow.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 0,
    "retry_delay": timedelta(minutes=2),
}

# DAG Object
dag = DAG(
    "random_user_dag",
    default_args=default_args,
    schedule_interval="@daily",
    catchup=False,
)

extract_data = PythonOperator(
    task_id="extract_data",
    provide_context=True,
    python_callable=ExtractData(),
    op_kwargs = {'url' : 'https://randomuser.me/api/?results=500',
    dag=dag,
)

transform_data = PythonOperator(
    task_id="transform_data",
    provide_context=True,
    python_callable=TransformData(),
    dag=dag,
)

load_data = PythonOperator(
    task_id="load_data", provide_context=True, python_callable=LoadData(), dag=dag,
)

extract_data >> transform_data >> load_data
```

ETL Testing Layer :

We could do two types of tests to ensure that the ETL process is working as intended

1. Code Testing : This is to ensure that the pipeline code produces the expected outputs for specific inputs.Code testing ensures that pipeline code produces the expected outputs for specific inputs. This helps us catch bugs proactively before they impact downstream data users.

2. Data Testing : - ensures that the data meets our expectations at particular points in the ETL process. Data testing ensures that the data meets our expectations at particular points in the ETL process. Data testing helps catch invalid records and unexpected changes in source data and sometimes also helps catch bugs in code.

As part of testing the processed “RandomUser.me” api data in csv file, i would do the following tests

1. Count Level QA: A python module that will compare the count of records between source and destination. The count level QA would help us catch any bugs in the code by showing the count discrepancy between source and destination data.

2. Field level QA : A python module that will check the field level population percentage between Source and Destination Data. This python module can get the population percentage of each of the fields in the source and destination datasets and in case field population percentage of any given field, between source and destination exceeds a set threshold, we could then check to see, why the differences are seen to uncover any potential issues in the source/destination datasets

Design/Business considerations:

Since the dataset being parsed is user data, decisions have to be made to decide if the **PII fields** in the user data need to be masked or encrypted before it can be stored in the CSV format.

These decisions need to accommodate the data privacy, storage policies in place, in that organization, and the end consumers of this data. If the end goal is, to encrypt the PII fields, we then have to update the “Transformation Layer” to accommodate this change

The other design consideration is, using login_uuid as a primary key vs using an auto increment key in the User Table.

There are pros and cons using either of these as a primary key.