**Design approach**

1. Fetching from SQS table :

I chose to read one message at a time from the queue, flatten the message into a python dictionary. The keys of python dictionary are the columns that are in the user\_logins table. While creating the dictionary, we apply base64 encoding on the PII fields.

Once, the dictionary is created, we delete the message from the queue.

1. Writing to the postgres table

The dictionary from previous process is the input to this function. From this dictionary, we write the values as a record in the postgres database

1. Putting it all together

We call fetchDataFromSQS() function and it returns dictionary with one record and we write the record into the postgres database using function writeToPostgres(). fetchDataFromSQS() is a generator function which returns one dictionary at a time.

We continuously poll the queue so that in event of new message getting added to the queue, we can quickly write it to the postgres database. This loop will run infinitely and the process never suspends.

**Questions on the decisions made:**

1. How will you read messages from the queue?

We created a generator function fetchDataFromSQS() which generates a dictionary containing all column values. If a particular column value is not present in the SQS message, the value is None in the dictionary

1. What type of data structures should be used?

We have used a dictionary to represent the message from SQS queue

1. How will you mask the PII data so that duplicate values can be identified?  
   To mask the PII data, I chose to use the SHA512 hashing because hashing techniques are irreversible. This offers protection to the data. Duplicate values can be identified as same value will always result in the same hashed result.
2. What will be your strategy for connecting and writing to Postgres?

For writing to Postgres database, we use the library psycopg2 using which we create connection, execute SQL query to insert the record, commit the results and finally, close the connection. Since, from the table definition all columns allow Null data, if any of the fields from the message is Null it will be inserting into the postgres table

1. Where and how will your application run?

The application can be run using the docker compose.yml file provided. We have listed the steps to run below.

**Steps to run code**

1. Clone the git repository  
   git clone <https://github.com/mihirrane/fetch-data-engineering-takehome-test.git>
2. Open config.ini and add values for password in postgres credentials
3. Navigate to the folder in which the repository is cloned. Open command line inside the folder and run  
   docker compose up
4. Check the container id of the service which is running the python script. Open another command line within the folder and run  
   docker ps  
   A screen shot of a computer

   Description automatically generated  
   Copy the container id corresponding to the image fetch-sqs\_to\_postgres
5. The container is running infinitely but we need to enter inside the container to run the python script  
   docker exec -it container\_id bash
6. Bash shell will open. Run command  
   python main.py
7. The data will be written to postgres. To check, note the container id of postgres container, fetchdocker/data-takehome-postgres
8. Run command  
   docker exec -it container\_id bash
9. Login to postgres using command:  
   psql -d postgres -U postgres -p 5432 -h localhost -W  
     
   Type password which is given in the provided document
10. When the credentials are accepted, postgress command shell will open
11. Type the query -   
    select \* from user\_logins;  
    You will see all the rows, 100 in our case

**Questions asked**

1. How would you deploy this application in production?

The application is hosted on docker hub. We need to build a deployment configuration using docker-compose.yml. Then we need to use a container orchestration tool such as docker swarm or Kubernetes to monitor the docker containers. We need a host for these docker containers such as AWS, Azure or any other cloud services.

We attempted to have the code in a separate docker image from the docker image of SQS and Postgres but we were finding it difficult to make the docker containers communicate with each other. We tried using ‘depends\_on’ and ‘links’ in the docker compose file but it did not work.

We also tried using linking the networks: default in SQS and python script container. It did not work. In production, the SQS link will not have ‘localhost’ in it. It will be deployed on AWS server and thus, this network issue will not persist.  
  
A screenshot of a computer program

Description automatically generated

1. What other components would you want to add to make this production ready?

To make this production ready, the AWS SQS will not be hosted on localhost and the python script will be accessed from its image and not using the volume mount. Ideally, each piece should be its separate image and they should communicate with each other through the docker compose file.

1. How can this application scale with a growing dataset.

The application reads one message at a time from SQS. With a growing dataset, it will still read one message, so the performance is unaffected with scale. But instead of using SQS, if Kafka were used as a system to store the message, we can use multiple workers and create partitions to process messages faster.

1. How can PII be recovered later on?   
   We can use different encryption algorithms such as RSA, AES, Blowfish to encrypt the data. With the usage of private key only the person who has access can see the original value of the fields.
2. What are the assumptions you made?

* AWS credentials were not provided so we added dummy credentials -   
  aws\_access\_key\_id=foobar

aws\_secret\_access\_key=foobar  
  
The error I got was - botocore.exceptions.NoCredentialsError: Unable to locate credentials

* Region for the AWS server was not provided   
  So I added region\_name=us-east-1  
    
  The error I got was botocore.exceptions.NoRegionError: You must specify a region.
* For app version, from the DDL I understood its integer column so I selected the first number before the first dot to assign to app\_version. For eg. 23.0.1 will be 23
* Since all columns are nullable if the ‘body’ key is present in the message, we will insert the values of all columns within it. If the body Json is empty, we will insert all values as Null
* If the body is not present in the message JSON, no insertion will be made and the log will record the messageId for future reference
* The wait time is set to 20 seconds to wait for a message from SQS. After that control will enter the infinite loop and again attempt for 20 seconds

**Folder contents**

1. config.ini  
   Contains the configuration details for the postgres database and the SQS credentials
2. sqs\_to\_postgress.py  
   Python script which contains the class SQSToPostgres. The methods in this class are encode(), decode(), fetchDataFromSQS() and writeToPostgres()
3. main.py

Python script which reads the credentials from config.ini and calls the methods from SQSToPostgres class to read data from SQS and write to postgres

1. unit\_tests.py

Python script which consists of all unit tests

1. requirements.txt

File contains the details of all libraries that are used in all code files

1. Dockerfile

Dockerfile contains instructions to create the docker image