Fetch Rewards
Data Engineering Take Home: ETL off a SQS Queue
Name: Gopichand Tadapaneni
Gmail: gtadapaneni@gmail.com

Project link(GitHub):
https://github.com/gopichowdary3/ETL-off-a-SQS-Queue.git

Thank you for giving me the opportunity.

In this Test, I have addressed the following questions:
1. How will I read messages from the queue?

I used Boto3, which is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python. It allows me to write software that makes use of services like Amazon S3, Amazon EC2, and others. Specifically, I used the 'receive_message' function provided by Boto3 to read messages from the queue.

2. What type of data structures should I use?

I chose to use Python's built-in data structures like dictionaries and lists for this project. For each record, I extracted the message from SQS and loaded it into a dictionary because JSON maps naturally to a dictionary. I then appended each dictionary into a list, which allowed me to bulk insert records into the Postgres database.

3. How will I mask the PII data so that duplicate values can be identified?

For masking the PII (Personally Identifiable Information), I decided to use a hashing function. This would allow me to preserve user privacy while still being able to identify duplicate values. Specifically, I used the SHA-256 hashing function, which returns a fixed size, unique hash value for each unique input.

4. What will be my strategy for connecting and writing to Postgres?

I used psycopg2, a PostgreSQL database adapter for Python. After establishing a connection to the PostgreSQL server, I used the 'executemany' function of the psycopg2 cursor object to bulk insert the list of records into the database. To ensure that the database connection does not close prematurely, I committed the transaction only after all the operations are done.

5. Where and how will my application run?

The application runs on an AWS EC2 instance. I used a Python script which can be executed from the command line. I could also schedule it to run at regular intervals using a cron

job. For a more advanced setup, I could also use containerization technologies like Docker, and orchestration tools like Kubernetes or Amazon ECS, which would provide additional benefits such as easier scaling, better resource management, and self-healing.

Questions:

1. How would I deploy this application in production?

I would use a cloud service like AWS to host and run the application. AWS provides managed services for both the queuing service (SQS) and the PostgreSQL database, which reduces the management overhead. The application would be containerized using Docker for better portability and scalability. The Docker containers could be orchestrated using a service like Kubernetes or AWS ECS. For continuous integration and deployment, I might use a service like Jenkins or AWS CodePipeline.

2. What other components would I want to add to make this production ready?

Monitoring & Alerting: I would use services like Amazon CloudWatch or Prometheus and Grafana to monitor the application. This would include tracking metrics like the number of messages processed per minute, the database write rate, etc. Any errors or abnormal behavior would trigger alerts.

Logging: Detailed logs would be maintained for debugging purposes. Services like AWS CloudWatch Logs or the ELK stack (Elasticsearch, Logstash, Kibana) could be used for this.

Error Handling & Retry Mechanisms: The application should be able to handle any errors that occur during the processing of messages or writing to the database. In case of transient issues, a retry mechanism could be useful.

Security: I would also ensure that all the necessary security measures are in place. This includes managing access controls, securing the data in transit and at rest, etc.

3. How can this application scale with a growing dataset?

As the volume of data grows, both the application and the database would need to scale. The application could be scaled out horizontally by adding more instances or containers as needed. This would allow it to process more messages in parallel. For the database, I might consider partitioning the data or using a managed service that can automatically scale, like Amazon RDS.

4. How can PII be recovered later on?

As part of the data privacy measures, the PII data is hashed before being stored. Since hashing is a one-way function, the original PII cannot be recovered from the hash. If there's a requirement to retrieve the original data later, I would need to use a different method to anonymize the data, such as encryption with a secure key. The original data can then be recovered by decrypting the data using the key.

5. What are the assumptions I made?

- I assumed that all messages in the queue are properly formatted JSON and that they contain all the required fields.
- I assumed that the SQS service is able to handle the incoming message load and that the PostgreSQL database has enough capacity to store the processed records.
- I also assumed that the EC2 instance where the application is running has sufficient resources (CPU, memory, network bandwidth) to handle the processing load.
- Additionally, I assumed that the hashing function used to mask the PII data provides a sufficient level of data privacy.