Decisions Made During Development

How will you read messages from the queue?

We use the boto3 library to interact with the AWS SQS queue. The get_messages_from_queue function is responsible for fetching messages from the queue.

What type of data structures should be used?

The script primarily uses dictionaries to store message data and tuples for inserting data into the PostgreSQL database.

How will you mask the PII data so that duplicate values can be identified?

The PII data, such as ip and device_id, are hashed using the SHA-256 algorithm. This ensures that while the actual data is masked, duplicate values can still be identified by the same hash output.

What will be your strategy for connecting and writing to Postgres?

We use a context manager (get_db_connection) to handle PostgreSQL database connections. The data is inserted into the user_logins table using the write_to_postgres function.

Where and how will your application run?

The application runs locally and uses Docker to simulate the AWS environment with LocalStack. PostgreSQL also runs in a Docker container.

Next Steps

1. **Improve Error Handling**: Add more robust error handling and logging to cover all possible failure points.
2. **Enhance Security**: Secure the credentials and sensitive information using environment variables or AWS Secrets Manager.
3. **Automated Testing**: Implement unit tests and integration tests to ensure the functionality works as expected.
4. **Deployment**: Create scripts for deploying this setup in a production environment, possibly using AWS ECS or Kubernetes.
5. **Scaling**: Implement mechanisms for handling a larger volume of messages and data, such as batching messages or using a message processing service.

Assumptions

1. The SQS queue and PostgreSQL are running locally using Docker and LocalStack.
2. The PII data needs to be masked but still allow for duplicate identification.
3. The script will be run manually or triggered by a cron job or other scheduling system.

Questions Answered

How would you deploy this application in production?

This application can be deployed using AWS ECS or Kubernetes for container orchestration. The SQS and PostgreSQL services would be managed by AWS. CI/CD pipelines can be set up using Jenkins, GitHub Actions, or AWS CodePipeline.

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

1. **Monitoring and Logging**: Use AWS CloudWatch for logging and monitoring.
2. **Error Handling and Alerts**: Integrate with AWS SNS or another notification service for alerting on errors.
3. **Scaling Mechanisms**: Implement auto-scaling for handling larger datasets and traffic spikes.

How can this application scale with a growing dataset? The application can scale by:

1. **Horizontal Scaling**: Adding more instances to handle increased load.
2. **Message Batching**: Processing messages in batches to optimize throughput.
3. **Database Optimization**: Using indexing and optimizing queries to handle larger datasets.

How can PII be recovered later on? To recover PII, you would need to store the original PII data in a secure manner, possibly encrypted in another database or service that has restricted access.