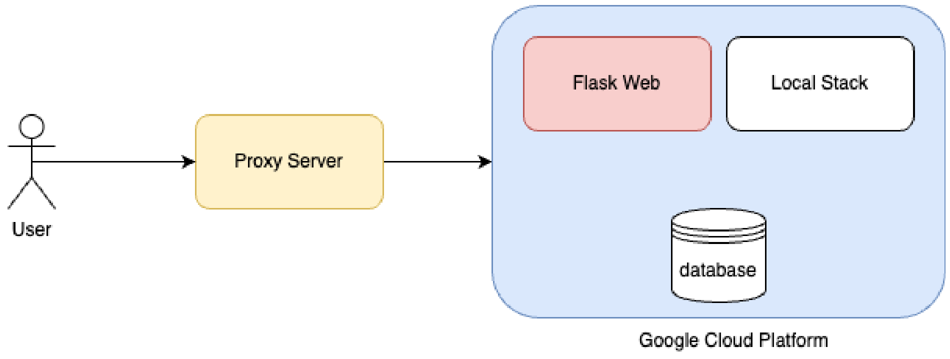
Questions

How would you deploy this application in production?

I've currently followed a simple architecture to deploy this to Google Cloud Platform .

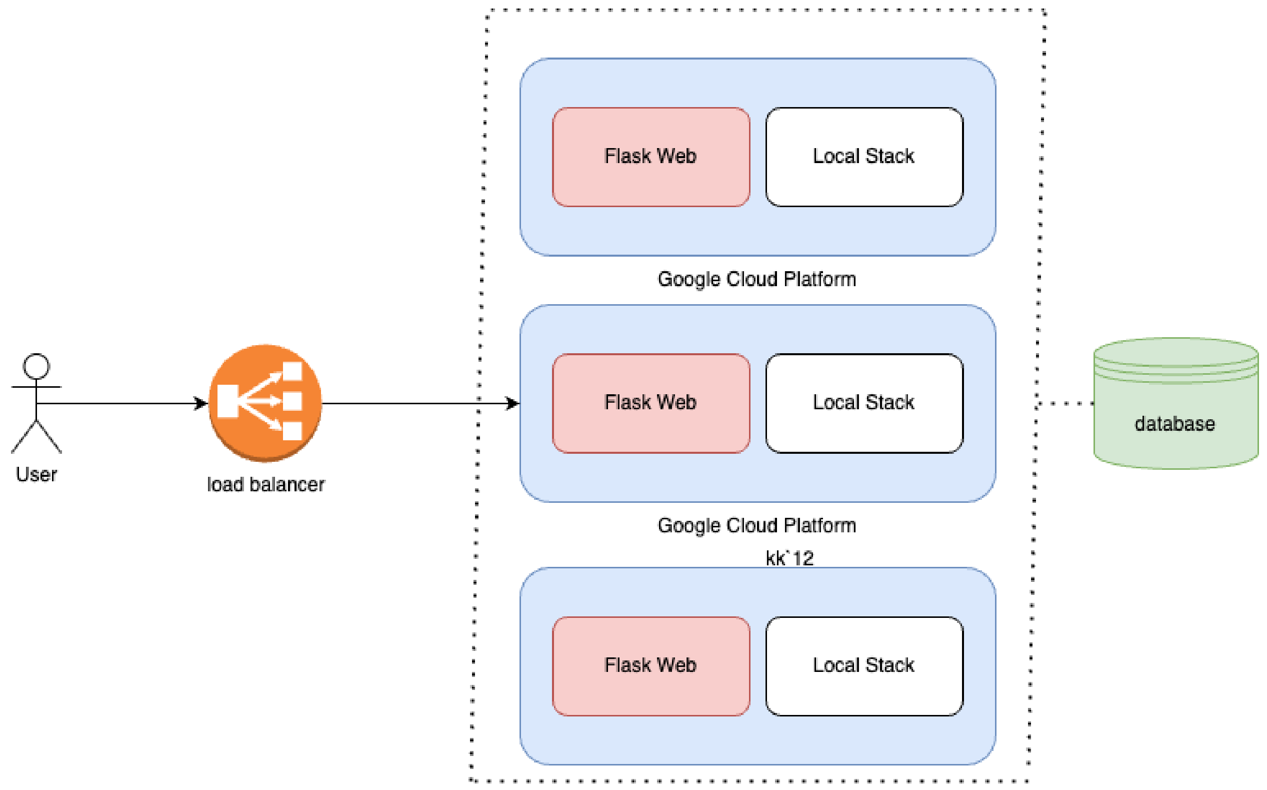
1. I created a medium size VM in https://console.cloud.google.com , setup my application inside the VM and security rules to expose the external IP to the world
2. I created a Proxy Server https://fetch-rewards-siddarth-sairaj.netlify.app that route requests to Web Server Running the Consumer and Web backend on GCP .



Current Deployed Architecture @ https://fetch-rewards-siddarth-sairaj.netlify.app/

But this architecture has few flaws, Some of the important ones that needs to be addressed are :

1. There is no Load Balancer currently, so it doesn't scale for more requests. I would set up Load Balancers with scaling rules to scale to more VMs for processing a high number of requests if necessary. Alternatively AWS Lambda/Glue is another popular alternative to help manage scaling ETL.
2. Running PostgreSQL in Docker requires manual management and scalability efforts, whereas a managed service offers automated backups, patches, and scaling out-of-the-box. (taking an assumption that the provided postgres image sets up DB locally and not as a method to connect to an external RDS) . I would move it to an external RDS Database and Connect accordingly.
3. There are currently no CI/CD Pipelines to roll out changes, I would integrate Github CI/CD to roll out changes to the Servers
4. There are currently no measures to monitor logs, I would use a cloud monitoring system for tracking the status of the service and also track logs generated from the console.
5. There are currently no Security measure to restrict Access, I would introduce IAM Access to restrict the actions performed to authorized individuals



Modified Architecture

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

1. Load Balancer (Alternatively AWS Lambda or Glue)
2. RDS Database
3. Cloud Instance VMS
4. Web Backend to Support Visualization and Accessing Data (Optional)
5. CI/CD Workflow
6. CloudWatch for logs monitoring

How can this application scale with a growing dataset?

By Leveraging Load Balancer with autoscaling groups (ideally on request count) and scale to more requests accordingly as ***described in above Architecture*** to distribute the workload to multiple instances .Alternatively use **AWS Lambda** or **Glue** to handle **ETL** and **Scaling** for us.

How can PII be recovered later on?
There are few ways to do this :

1. **Using Cryptography :** Cryptography involves using algorithms to transform data into a format that can't be understood without a specific key, ensuring data confidentiality. In Python, to mask (or encrypt) data, you'd typically use a library like cryptography. First, you'd generate a key, then use this key to encrypt the data into a ciphertext. To unmask

(or decrypt) the data, you'd use the same key to reverse the encryption process, converting the ciphertext back into its original format. It's essential to securely manage and store the keys, as access to the key means access to the original data.

2. **Using Base64** : Base64 encoding is a technique that converts binary data into a string of ASCII characters, primarily for safe transmission over protocols that are ASCII-friendly. In Python, the base64 module provides methods for encoding and decoding data using Base64. While this can be used to represent binary data as strings, it's important to note that Base64 is not a cryptographic method; it merely disguises data in a way that is easily reversible. To "mask" data with Base64, one would encode the binary or byte data into a Base64 string. To "unmask" or retrieve the original data, one would decode the Base64 string back to its binary or byte form. However, because this process is easily reversible without any keys, Base64 should not be relied upon for data confidentiality or security purposes.

What are the assumptions you made?

1. I've inferred that the "app_version" should not be treated as an integer due to its format (e.g., 2.0.1). Consequently, I have modified the Postgres table to store it as a string.
2. Regarding the boto3 region setup, the region wasn't initially specified, leading to an error: `botocore.exceptions.NoRegionError: You must specify a region`. To resolve this, I have manually set the region to "us-west-2".
3. My approach to data masking was based on the premise that visual obfuscation was the primary requirement. Thus, I implemented **base64** encoding. If a higher degree of confidentiality is required, I recommend transitioning to more robust encryption methods such as **SHA256** or other **cryptographic** algorithms.
4. It was ambiguous as to the number of requests that needed processing. To ensure accuracy, I am obtaining this specification directly from the user.
5. I have chosen to iteratively process the **SQS** events without deletion.
6. In the absence of provided **AWS access** and secret access keys, I have defaulted to a predetermined value.