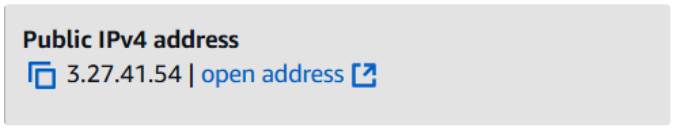
Hosting Service Access

Access to deployment tools is handled by an administrator account in AWS with access to a EC2 compute instance. The repository can be cloned using in line commands and then hosted by changing the “GLOBAL” file to include the public port for the ec2 instance. Any custom links can be made to work by adding a reroute to the public hosting id through the hosting portal.

1. Clone the latest repository of the AI_Usescale_APP

2. Locate the Public IPv4 address for the EC2 instance and edit the Globals.jsx file within ./client/src/GLOBALS



3. Install tmux and create two terminals for frontend and backend Hosting

   tmux is a utility package used to simultaneously host both the frontend and backend off the singular EC2 instance

   ”sudo apt install tmux”

4. Create a backend compartment
   ”tmux new -s backend”

   Change into the home directory and initialize database
   ”python3 sqlite.py”

Run the backend server on the EC2 instance

flask run --host=0.0.0.0 --port=5000

Use the following shortcut to exit the compartment

”Ctrl + B → D”

5. Create a frontend compartment

   ”tmux new -s frontend”

   Run the frontend server on the EC2 instance

```
[ec2-user@ip-172-31-25-145 client]$ npm run dev -- --host 0.0.0.0

  VITE v7.1.3  ready in 268 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: http://172.31.25.145:5173/
  ➜  press h + enter to show help
```

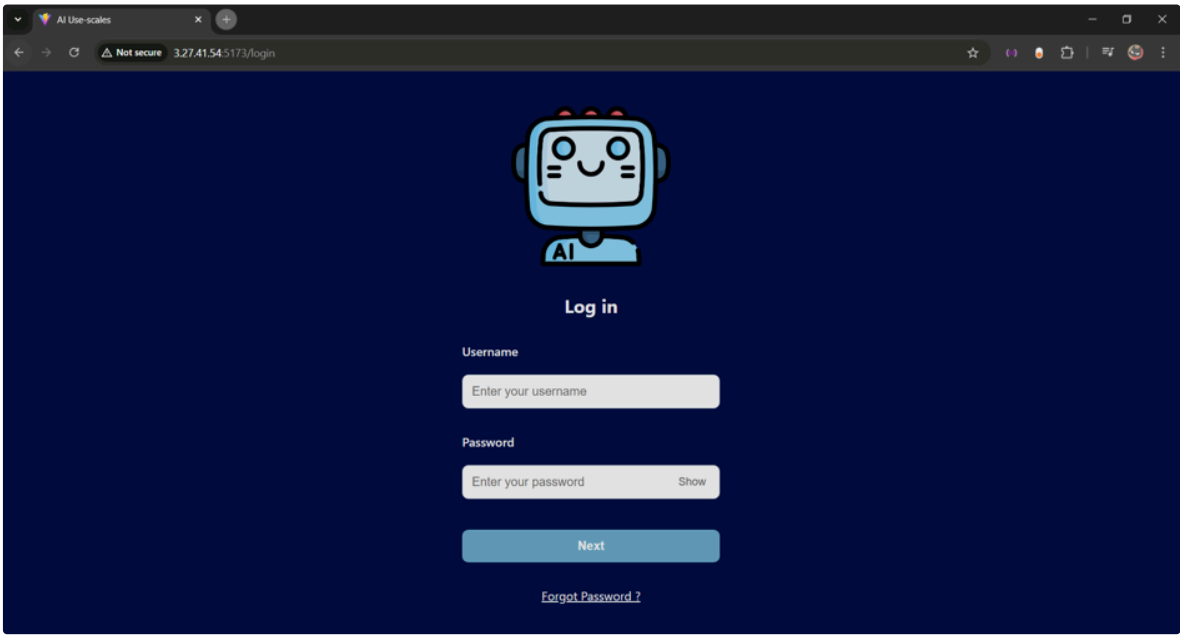
   The development modes used for testing but for deployment it is recommended to build and serve for deployment.

Final product

The program can now be accessed via the following link

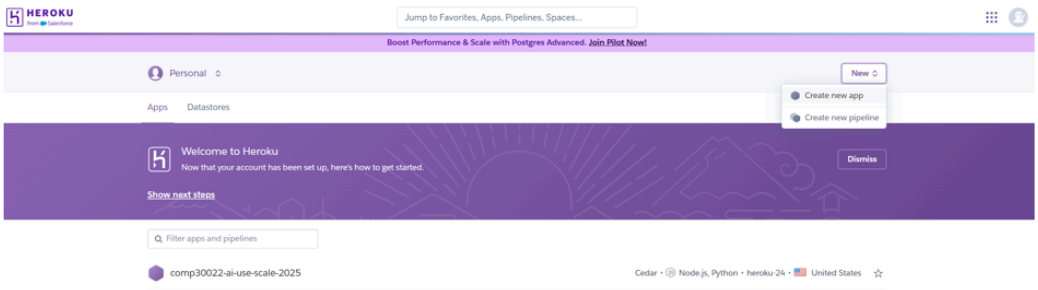
(public IP):5173

To use a custom link to redirect to the program a third-party DNS provider must be used, and the steps will depend on their setup.



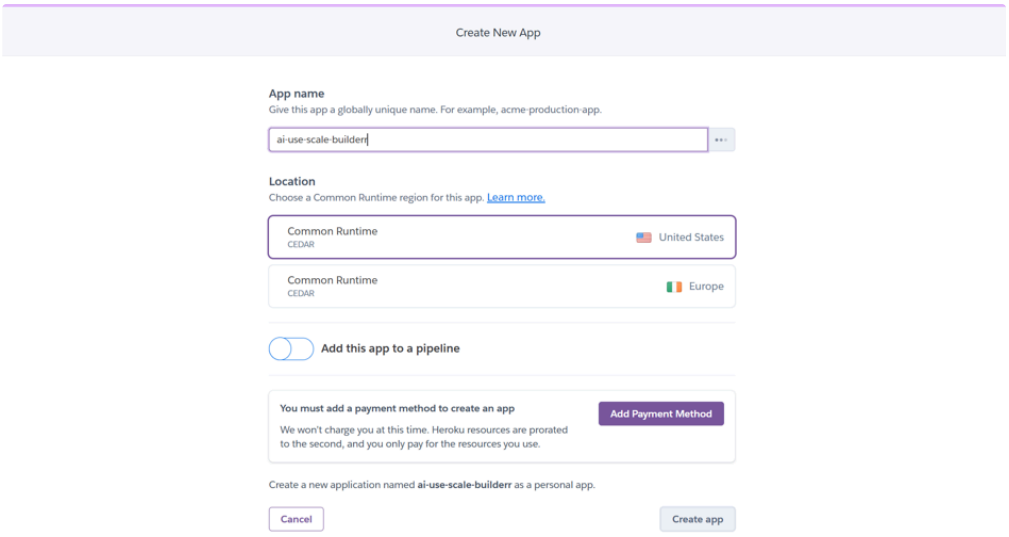
Heroku

- Once ownership has been transferred, the user can setup and deploy onto **Heroku** following these steps
- As the owner of the repository
  - I will create my own **Heroku** account Heroku
  - Create new app



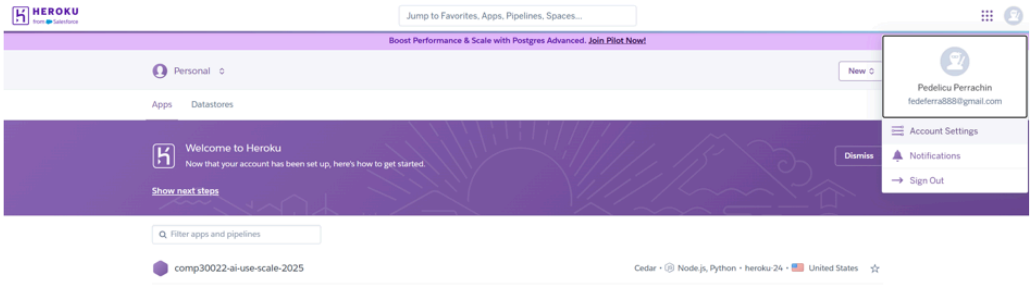
Heroku home page once logged in: click on New → Create new app

  - add in an appropriate unique name for it like: ***ai-use-scale-builder***
  - add payment method (Heroku mandates that)
  - then create the app



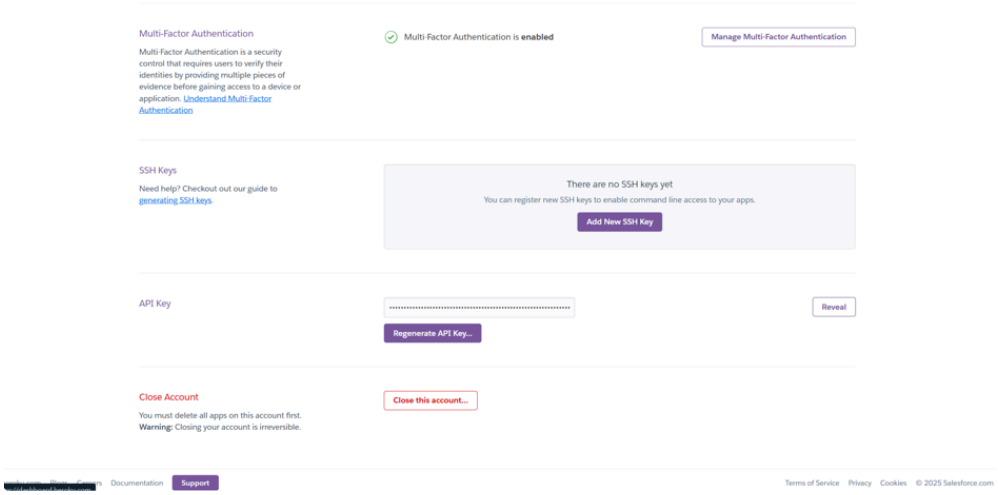
add a name for the app → add payment method → choose runtime → create app

- Click on Profile icon on the top right corner of the page



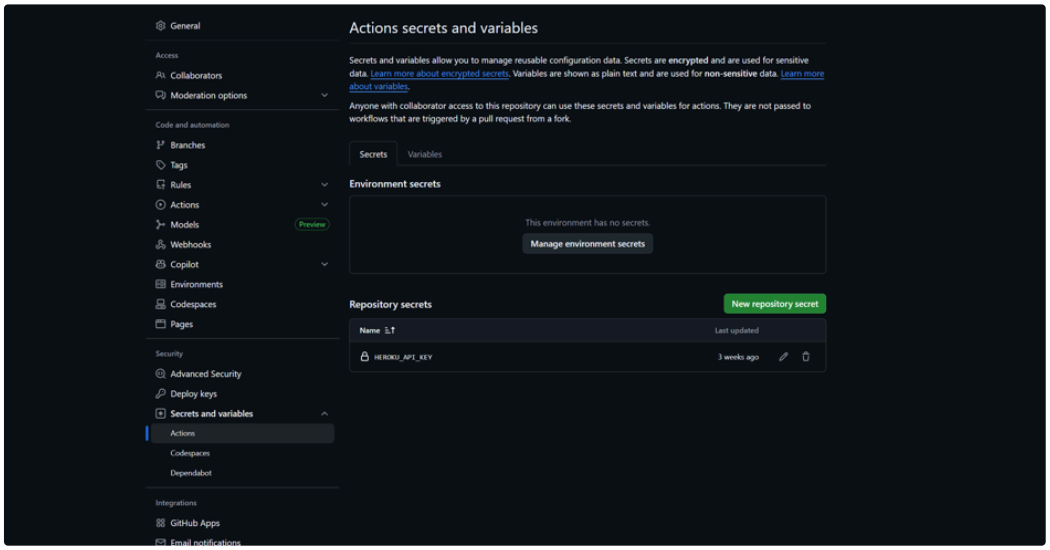
Click on top right corner (Profile icon) → Account Settings

- open account settings
- drag down to see API Key
  - click reveal and copy your API Key



Find API Key section → press to reveal it → copy it

- Go into the Settings page of the github repository
  - find secrets and variables
    - Actions
    - And add in a **New Repository Secret**



Open settings in the repository page → find Secrets and variables → Actions → New repository secret

  - The Heroku CLI | Heroku Dev Center
    - make sure to have installed Heroku CLI
- Head back into your github repository
  - make sure there is a deployment workflow file under **.github/workflows**

```yaml
name: Deploy

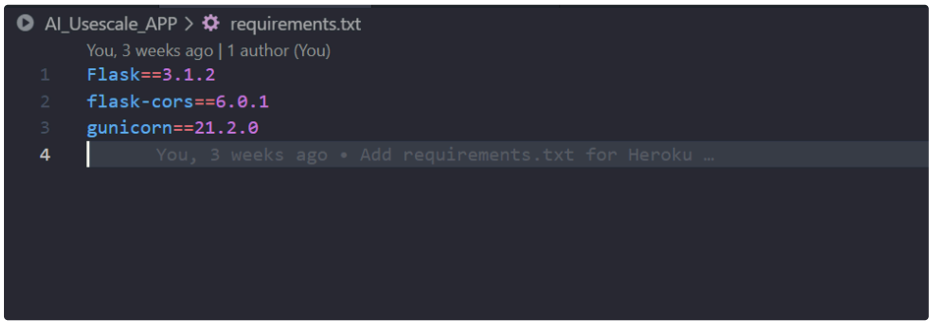
on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Install Heroku CLI # <- IMPORTANT!!! Make sure the cli is installed before using the action
        run: |
          curl https://cli-assets.heroku.com/install.sh | sh
      - uses: akhileshns/heroku-deploy@v3.14.15 # This is the action
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "your-app" # Must be unique in Heroku
          heroku_email: "your heroku email address @ yay .com"

```

- Default workflow: checks out repository → installs heroku cli → and deploys via api key, app name, account details

- in the terminal
  - ```
    heroku login
    ```
    - login to heroku with your account
  - ```
    heroku apps
    ```
    - make sure you can find your newly created app
  - ```
    heroku buildpacks:clear
    heroku buildpacks:add heroku/nodejs
    heroku buildpacks:add heroku/python
    ```
    - set up the buildpacks for your app **Order Matters:**
      - Node → builds your React frontend (npm install + npm run build)
      - Python → runs your Flask backend
  - ```
    heroku buildpacks
    ```
    - run this and *make sure* that nodejs comes first then python, like this:
      - ```
        1. heroku/nodejs
        2. heroku/python
        ```
  - ```
    pip freeze > requirements.txt
    ```
    - in the root, pip freeze and make sure **flask**, **flask-cors**, and **gunicorn** are in the file (everything else can be removed, if not used in the project)



-
- ```
  touch Procfile
  ```
  - create **Procfile** in the root of the project
  - add in the exact text:
    - ```
      web: gunicorn app:app
      ```
      - this tells Heroku to run app.py with the instance app (from app = Flask(__name__, static_folder="client/dist", static_url_path="") within app.py)
- ```
  touch .python-version
  ```
  - create file to specify the python version
  - in this instance, we used 3.12

  -

- in order the serve the react build from flask
  - we needed to add a few modifications in app.py
  - **it has already been done for you**, but the details will be included as screenshots for reference

    - added the static_folder where the react build lies so Heroku and Flask knows where it is, no static_url_path needed though
  ```

```python
@app.route("/", defaults={"path": ""})      You, 3 weeks ago • Serve react build from Flask
@app.route("/<path:path>")
def serve_react(path):
    import os
    from flask import send_from_directory

    # if the requested path is a real stored build file in client/dist, serve it directly
    if path != "" and os.path.exists(os.path.join(app.static_folder, path)):
        return send_from_directory(app.static_folder, path)
    else:  # otherwise, serve index.html so React Router can handle it
        return send_from_directory(app.static_folder, 'index.html')

@app.errorhandler(404)
def not_found(e):
    from flask import send_from_directory
    import os
    # if refresh page, serve index.html instead of showing Flask 404
    return send_from_directory(app.static_folder, "index.html")
```

- two new route handlers were added to allow Flask to correctly serve a React build

```
You, 3 weeks ago | 2 authors (You and one other)
// const HOST = "http://localhost:5000";
const IS_PROD = process.env.NODE_ENV === "production";

const HOST = IS_PROD ? "" : "http://localhost:5000";

export default HOST;      superderped, 2 months ago • login and template loading implemented …
```

- inside frontend **client/src/GLOBALS/Global.jsx**, the host has been updated to run both locally and on deployed server

    - **NOTE:** this is now done automatically because the Nodejs buildpack has been added into the Heroku app settings
        - ```
          cd client
          ```
            - enter the frontend folder
        - ```
          npm run build
          ```
            - and build the React frontend
    - now that everything is done, you can git add, commit, and push all and any of your updates and they will all be auto deployed into the **Heroku** app

- Manual Deployment Option
    - See link for more details 👇
    - Deploying with Git | Heroku Dev Center