CS3219 OTOT Task B

- **Name**: Yusuf Bin Musa
- **Matric. Number**: A0218228E
- **Repo Link**: https://github.com/yusufaine/OTOT-B

Task B1: Simple Backend

Requirements

1. MongoDB (Mongo Atlas was used to host the database),
2. `.env` file that follows the format as shown:

   ```
   PORT=<number>
   MONGO_URI=mongodb+srv://<user>:<password>@<database_url>
   ```

3. Docker for easy installation -- this would be used.
4. Postman is used to (manually) test CRUD for the API.

Local Deployment

Build the image from the given `Dockerfile` and map the ports accordingly.

```
# create yarn-lock and node_modules
yarn

# build the docker image
docker build . -t otot-b1-demo
docker run -dp <local port>:<exposed port> otot-b1-demo

# verify that it's running correctly
docker ps -a
```

Cloud Deployment

Google Cloud Platform was chosen as free credits were given for it. Cloud Run was the used over App Engine as there was no *need* to have this service up and running 24/7.

The official documentation from Google to assist the deployment but some deployment steps are included.

Optionally, Secret Manager was used also used to manage `.env` secrets without explicitly sharing the file.

GCP Requirements

1. Google account
2. Google Cloud CLI installed (recommended, but it's possible to do this via web GUI)

Deployment Steps

1. Create a Google Cloud project
2. Ensure that billing is enabled

- It is quite unlikely that we would be billed as the first 180,000 vCPU seconds are free, though I recommend to delete the instance just to be safe.

3. Set the project ID (project name is NOT project ID) for the deployment.
4. Create a `.gcloudignore` to ignore unnecessary files that are not needed for deployment.
    - Do NOT add `.env` if Secret Manager is not planned to be used.

```
# get project ID
gcloud projects list
gcloud config set project <projectId>

# setting defaults
gcloud config set run/region asia-southeast1 # deploy to Singapore by default

# deploy to Cloud Run at the root of the project directory
gcloud run deploy <service-name> --source .

# Alternatively provide the necessary params.
# I excluded `.env` as seen in `.gcloudignore` as I was using Secret Manager
(optional)

# gcloud run deploy task-b1-otot --source . \
#  --update-secrets OTOT_B_MONGO_PORT=OTOT_B_MONGO_PORT:latest \
#  --update-secrets OTOT_B_MONGO_URI=OTOT_B_MONGO_URI:latest \
#  --update-secrets OTOT_B_SOURCE=OTOT_B_SOURCE:latest
```

Troubleshoot Cloud Run Deployment

1. Unable to build container
    - Try to build and run `Dockerfile` locally and ensure that it works.
    - `.dockerignore` file is optional, but ensures that only the necessary files are copied over (non-documentation related).
2. "Successfully deployed on GCP Run but I'm unable to access the link that was generated"
    - Ensure the following settings in `Cloud Run` > `SERVICE_NAME` > `Triggers`
        - `Ingress` : Allow all traffic
        - `Authentication` : Allow unauthenticated invocations
            - This is set to false if the default is chosen in the deployment step.
3. How do I use Secret Manager with GCP Cloud Run?
    - You can refer to the official documentation.
    - Exclude `.env` and other unnecessary files in `.gcloudignore` .
    - Verify that secrets are loaded in: `Cloud Run` > `SERVICE_NAME` > `Revisions` > Select running revision > `Containers` > `Environment variables`
    - TLDR via gcloud CLI:

```
# Create secret (no initial value)
gcloud secrets create <SECRET_NAME>

# Add initial/update secret
echo -n "SECRET_VALUE" | gcloud secrets versions add <SECRET_NAME> --
data-file=-

# Verify
gcloud secrets versions access <VERSION> --secret="SECRET_NAME"

# Example
# gcloud secrets versions access <N> --secret="SECRET_NAME"
# gcloud secrets versions access latest --secret="SECRET_NAME"
```

Testing via Postman

Import the Postman collection via this link.

Swap the current values of `host` and `deployed` to test the local/deployed environment.

Explanation of HTTP requests:

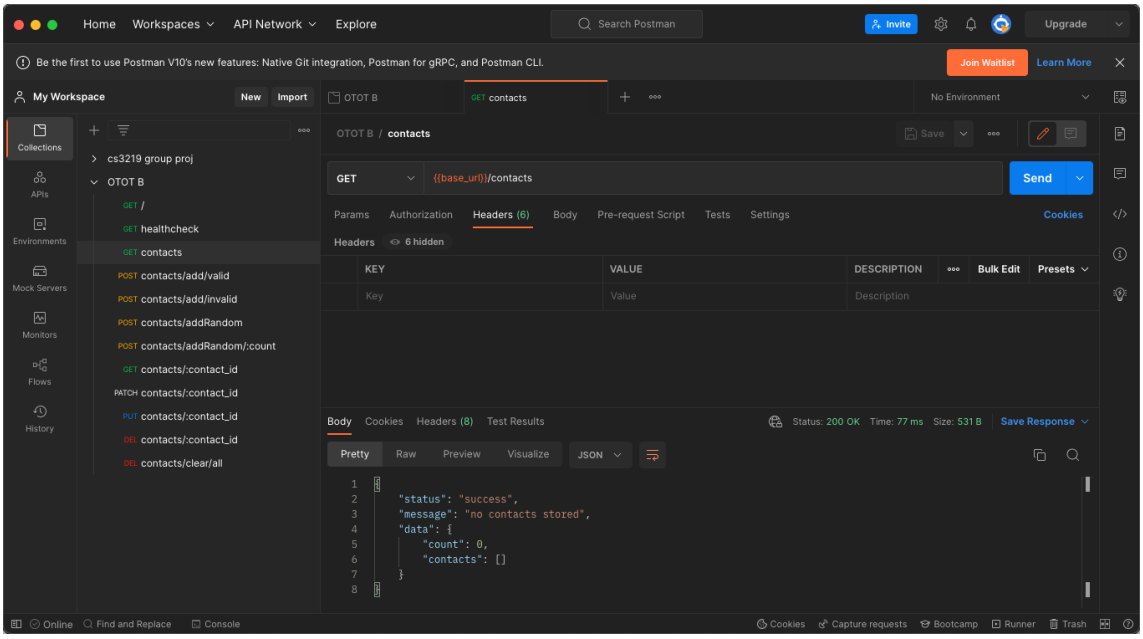
- `hello world` : test that Express is working
- `router test` : test that `router.use()` is working
- `contacts` : list all contacts (might be empty initially)
- `contacts/add/valid` : adds a valid user as seen in `body`
- `contacts/add/invalid` : tries to add an invalid user as seen in `body` , name is missing
- `contacts/addRandom` : adds a valid random user by using `@fakerjs` , mainly for testing
- `contacts/addRandom/:count` : similar to `addRandom` , but for multiple people
- `contacts/:contact_id` : Throws error if (mongo) ID does not exist
  - `GET` : returns contact with the specified ID -- refer to `contacts`
  - `PATCH` / `PUT` : updates a specific field given an ID
  - `DELETE` : removes the specified contact from the database
- `contacts/clear/all` : helper endpoint to wipe the entire contacts database

Note:

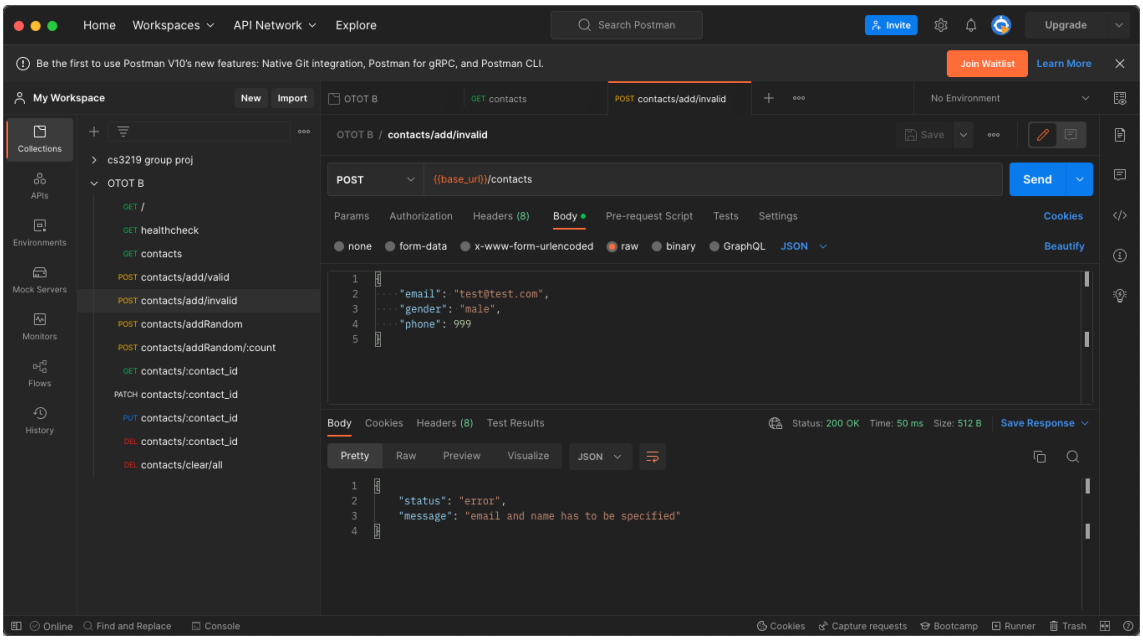
- `params` and `body` would need to be updated manually, where they are necessary.
- The default mongo ID is used for demonstration purposes. Realisitically, the entries would likely be indexed by phone number and checked for uniqueness.

Screenshots

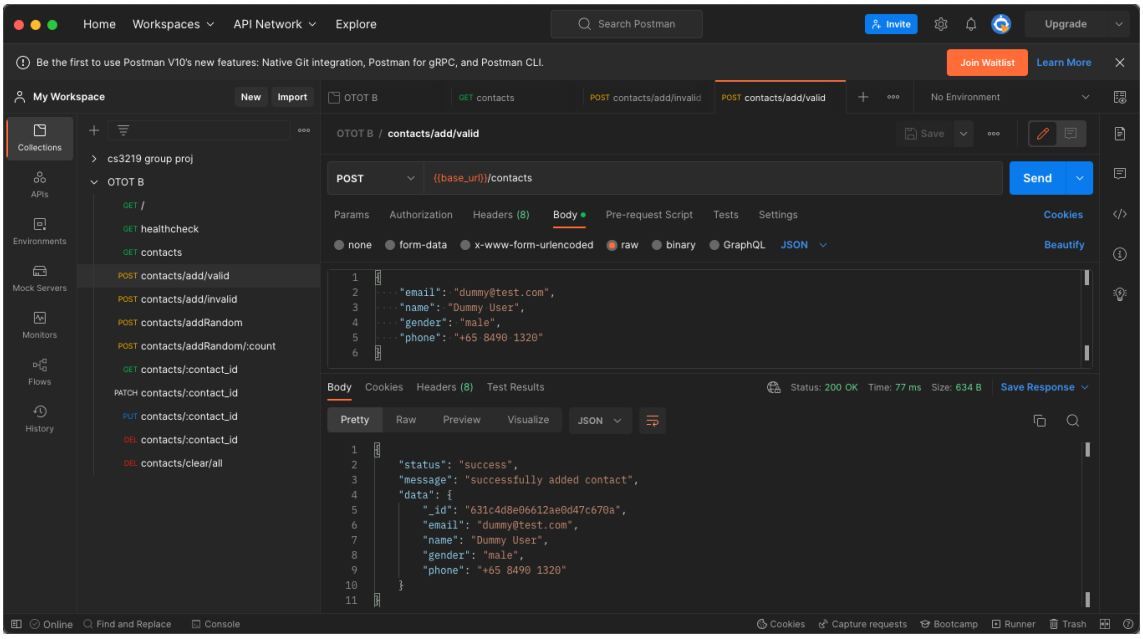
`GET` : get contacts given empty database



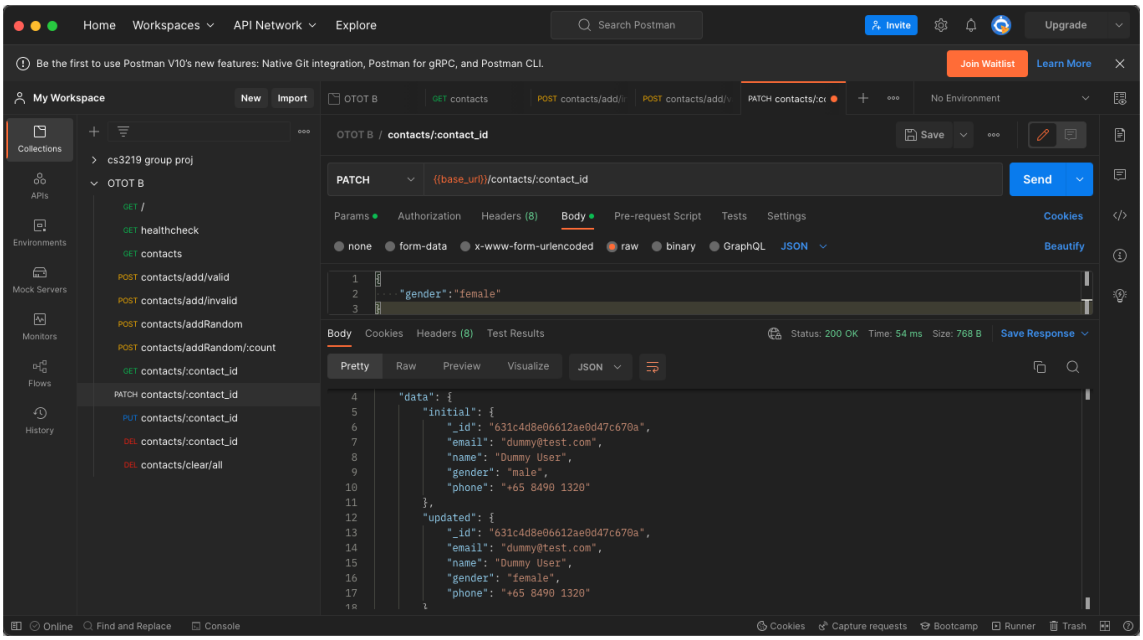
`POST` : invalid contact (missing name)



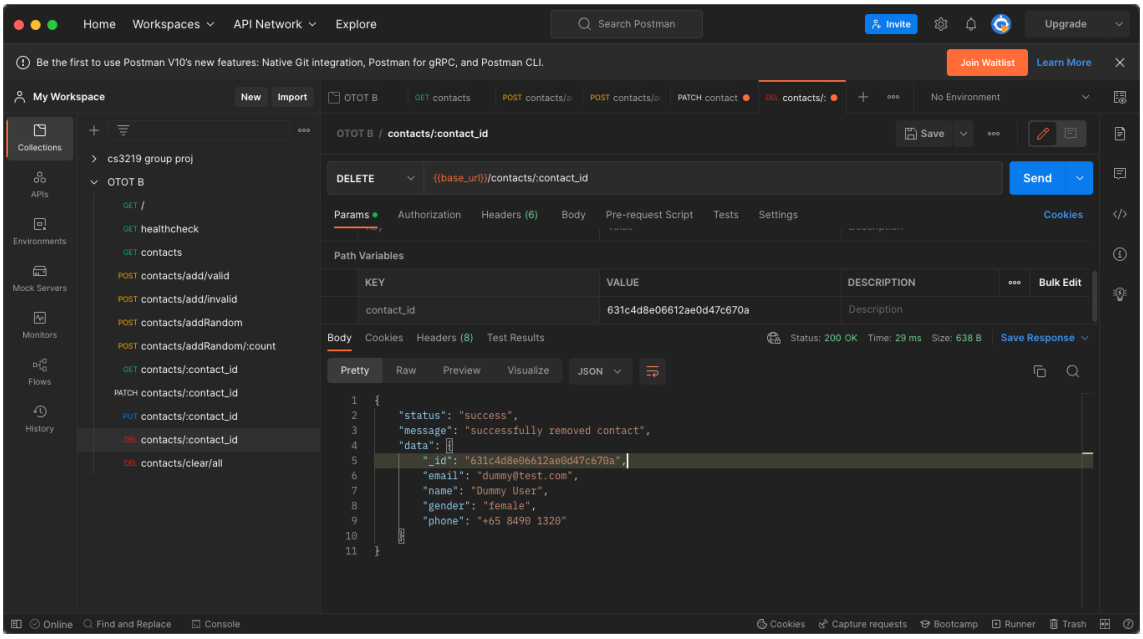
`POST` : valid contact



`PATCH` / `PUT` : update value of contact



`DELETE` : remove specified contact



Task B2.1: Testing through Continuous Integration

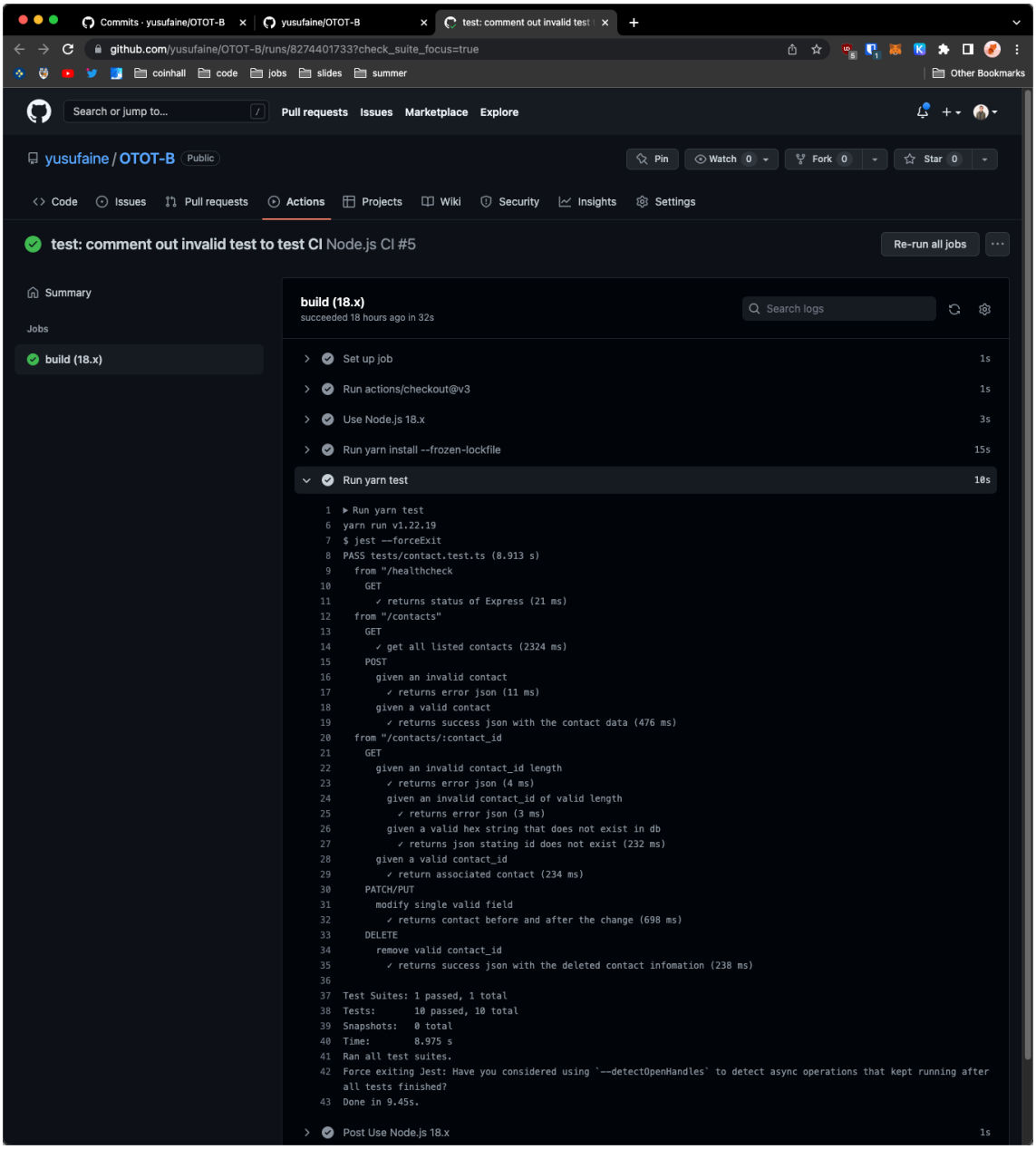
This section briefly goes over how Jest and Superset were used to test the application locally as well as through CI tools (Github Action).

Summary of CI

- For testing, Jest was used along with Superset to test the REST endpoints.
- For these series of tests, the actual database that is deployed is being used as there were issues with using a mock or in-memory database.
    - As such, the `--forceExit` flag was needed as the database connection was still open.
- While `PATCH`, and `PUT` serves provides only the modified and full data with the modified values respectively, they serve the same function in this task (as I only found out the difference after the fact).
- The Postman collection can be found here with the variables already set as well as some helper endpoints that can be used to populate the database.
- The Github Action yaml file can be viewed here.

Local test output

Testing via [Github Action]



Task B2.2: Deploying through Continuous Deployment

Building from the previos sections, this section briefly goes over how Github Action is used as a CD tool, paired with Google's Cloud Run to ensure that the latest (tested) version is deployed live.

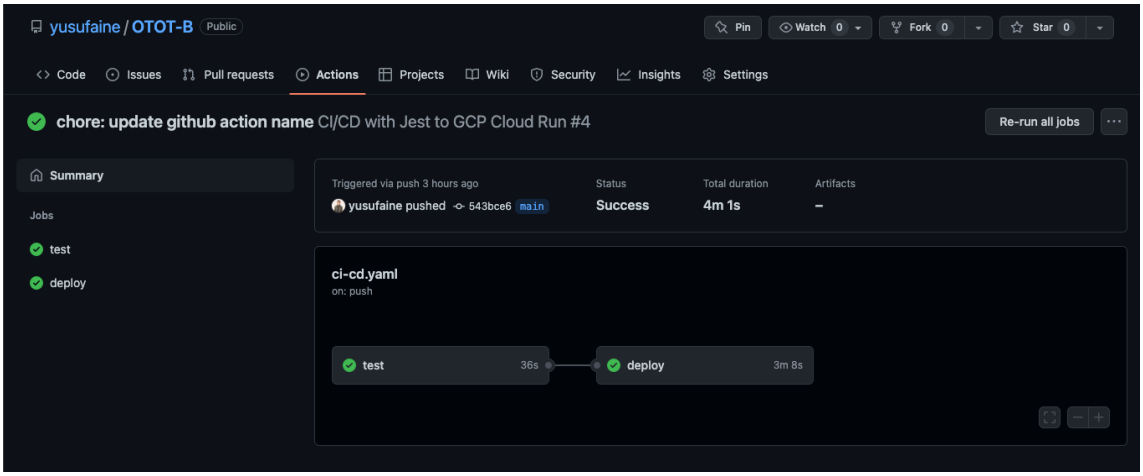
Summary of CD

- With reference to the Github Action yaml, the deployment step had to somehow execute after all the tests ran successfully, and build and/or deploy the repo/ `Dockerfile` .
- While Google Cloud Run allows to setup continous deployment fairly easily, deployment via `Dockerfile` was preferred as the process was more straight-forward.
    - Cloud Run's CD creates a Cloud Build trigger that can launch when something is being pushed to the monitored branch of the repo (requires Github authorisation).

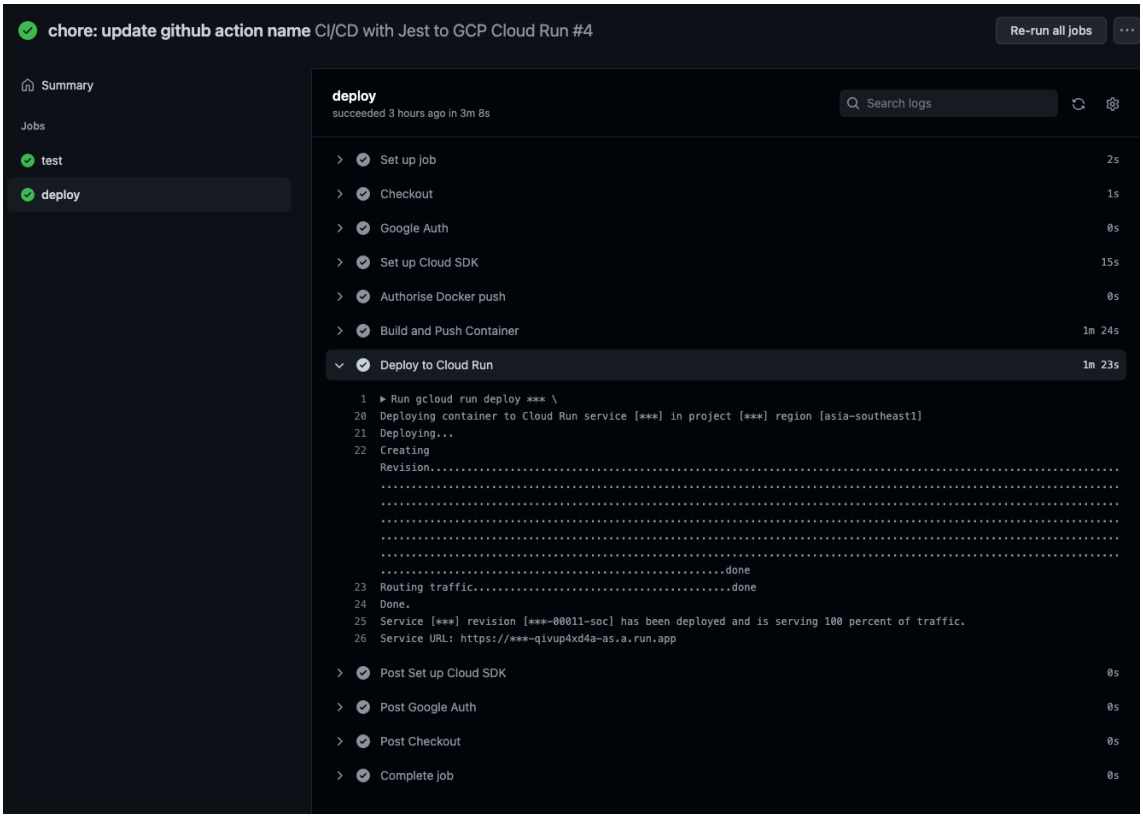
    - The build step is not able to be observed from Github's interface, only through GCP Cloud Build.
    - There was not a clear way to conditionally build -- if the testing step fails, it still gets built and deploys to Cloud Run.
- CD via Github Actions is also a little invovled as it requires setting up a few resources that allows Github to access the GCP project so that it can deploy the `Dockerfile` accordingly.
    - This Google community tutorial was used to setup the GCP Deploy account and obtain the necessary credential.
    - The official Github Action from Google was also referenced.

Continuous Deployment Screenshots

Deploy only after successful test



Github Action CD Logs



Deployed via Google's Service Account

task-b1-otot-00011-soc

Deployed by Github-CICD@cs3219-361805.iam.gserviceaccount.com using gcloud

CONTAINERS | CONNECTIONS | SECURITY | YAML

General

CPU allocation	CPU is only allocated during request processing
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

Max. instances	100

Main container : task-b1-otot-1 (Port 8080, 1 CPU, 512MiB memory)

Image URL	gcr.io/cs3219-361805/task-b1-otot@sha256:52e4d9b...
Port	8080
Build	(no build information available)
Source	(no source information available)
Command and arguments	(container entrypoint)
CPU limit	1
Memory limit	512MiB

Environment variables (3)

Volume mounts (0)