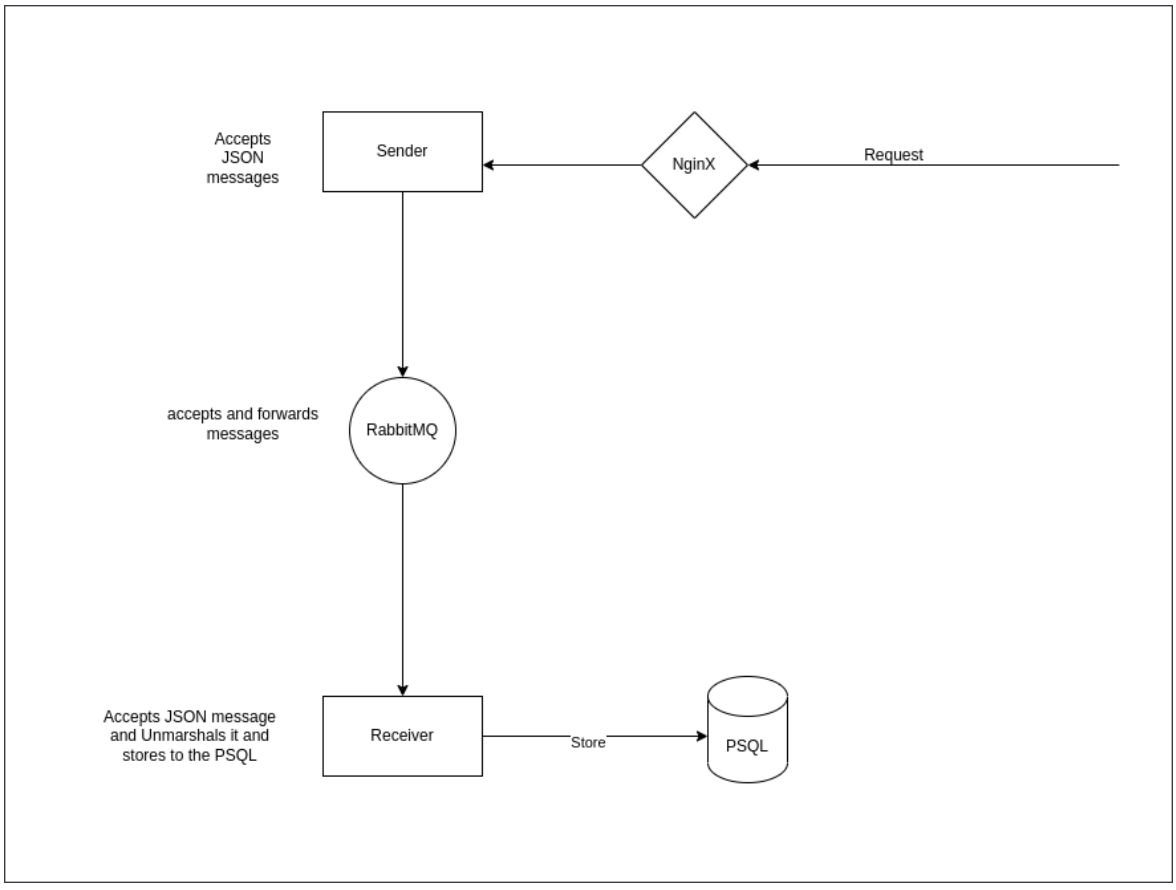
Final_Project

Created by	sarah nik
Created time	@September 15, 2023 6:23 PM

Overview Of Final Project



Part 1 & 2

- Docker file for Sender Service :

```
# base docker image to build our golang file
FROM golang:1.18-alpine AS builder

LABEL authors="sra"

# create working directory on our alpine image
RUN mkdir app

WORKDIR /app

ENV CGO_ENABLED 0
ENV GOPATH /go
ENV GOCACHE /go-build

COPY ./sender/go.mod ./sender/go.sum ./
RUN --mount=type=cache,target=/go/pkg/mod/cache \
    go mod download

COPY ./sender .

RUN --mount=type=cache,target=/go/pkg/mod/cache \
    --mount=type=cache,target=/go-build \
    go build -o result ./

FROM alpine:3.18.2

RUN mkdir final

WORKDIR /final

COPY --from=builder /app/result .

EXPOSE 9090


# main commands of our Dockerfile
CMD [ "./result" ]
```

Trick

- Here are two tricks to optimize image builds and reduce the need to re-download dependencies and rebuild the image .
    - **Cache Utilization :**

      Docker utilizes caching to cache build stages. If a stage has not changed, Docker uses the cache. Optimize the Dockerfile by placing less frequently changing steps (e.g., adding dependencies) towards the end of the Dockerfile.
    - **Multi-stage Builds**

      Utilize multiple stages in a Dockerfile to use temporary images for different steps (e.g., compiling code, building) and keep only the necessary information in the final image. This significantly reduces the final image size

- The receiver service, has similar docker file with the difference that this service does not need to expose port 9090 .

Part 3

Docker compose file

```
version: '3.8'

services:
  # sender
  sender:
    build: ./sender
    container_name: sender
    depends_on:
      - rabbit
    restart: on-failure
    networks:
      - sre-net
      - send-ng
    ports:
      - "9090:9090"
  # receiver
  receiver:
    build: ./receiver
    container_name: receiver
    depends_on:
      - db
      - rabbit
    restart: on-failure

    networks:
      - db-net
      - sre-net

  # database
  db:
    image: postgres:15.3-alpine
    container_name: db
    restart: always
    environment:
      POSTGRES_USER: receiver_user
      POSTGRES_PASSWORD: 2980
      POSTGRES_DB: tmp
    networks:
      - db-net
    volumes:
      - psql-vol:/var/lib/postgresql/data

  # nginx
  proxy:
    image: nginx:1.25.1
    container_name: ngx
    volumes:
      - type: bind
        source: ./nginx/nginx.conf
        target: /etc/nginx/conf.d/default.conf

    restart: on-failure

    ports:
      - "80:80"
    depends_on:
      - sender
    networks:
      - send-ng
    #rabbit
  rabbit:
    image: rabbitmq:3-management-alpine
    container_name: rbt
    networks:
      - sre-net
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest

    ports:
      - "5672:5672"
      - "15672:15672"
```

```
# networks
networks:
  sre-net:
  db-net:
  send-ng:

#volumes
volumes:
  psql-vol:
```

Services
sender
receiver
rabbitMQ
nginX
postgres

> We have 5 services here, The first service is the sender, which needs to be in the same network as the Nginx web server to establish a connection with it ; On the other hand, to complete the configurations in the main file for both sender and receiver programs, both programs should be able to establish a connection with RabbitMQ using the connection name. In a way, Docker DNS should work for them.

> On the other hand, the receiver program must be able to establish a connection with it via the database name, so that both the stability issue of the container's IP is resolved, and our database is on a local network. Therefore, these two services must be defined in a same network.

Networks	services
sre-net (stands for sender , receiver and rabbitMQ network)	sender - receiver - rabbitMQ
db-net	receiver - psql
send-ng	sender - nginx

---

nginx.conf

path : https://github.com/sarnik80/DevOps-Project/tree/main/nginx

```
upstream sender {
    server sender:9090;
}


server {
    listen 80;
    server_name localhost;
    location / {
        proxy_pass http://sender;
```

```
        proxy_http_version 1.1;
    }

}
```

- This file sets up an Nginx server as a reverse proxy to communicate with a sender service on port 9090 .
- upstream is intended as the destination for proxying and points to the address `sender:9090` on port 9090 .
- **In this configuration, our web server is listening on port 80 and forwards incoming requests to the sender service .**
- I created this file in the nginx directory to bind mount it to the default nginx configuration location of the docker container.

```
 proxy:
    image: nginx:1.25.1
    container_name: ngx

# bind nginx directory to the default path for config file
    volumes:
      - type: bind
        source: ./nginx/nginx.conf
        target: /etc/nginx/conf.d/default.conf

    restart: on-failure

    ports:
      - "80:80"
    depends_on:
      - sender
    networks:
      - send-ng
```

Postgresql

```
  # database
  db:
    image: postgres:15.3-alpine
    container_name: db
    restart: always
    environment:
      POSTGRES_USER: receiver_user
      POSTGRES_PASSWORD: 2980
      POSTGRES_DB: tmp
    networks:
      - db-net
    volumes:
      - psql-vol:/var/lib/postgresql/data
```

- For writing the necessary configurations for the database service, two important aspects were highly emphasized:

-
  1. Defining environment variables.
  2. Volume

- We set the values of these variables as configurations in the receiver program to enable it to connect to the database.

- Additionally, Docker creates a volume named 'psql-vol,' allowing us to access the stored data even after stopping that container.

```
// database config
  var (
    user     = "receiver_user"
    password = "2980"
    host     = "db"
    dbName   = "tmp"
    port     = "5432"
  )
```

RabbitMQ

```
  #rabbit
  rabbit:
    image: rabbitmq:3-management-alpine
    container_name: rbt
    networks:
      - sre-net
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest

    ports:
      - "5672:5672"
      - "15672:15672"
```

> `RABBITMQ_DEFAUILT_USER` ---> Sets the default RabbitMQ username to "guest.

> `RABBITMQ_DEFAULT_PASS` ---> Sets the default RabbitMQ password to "guest."

- "5672:5672" —> Map host port 5672 to container port 5672 for RabbitMQ
- "15672:15672" —> Map host port 15672 to container port 15672 for RabbitMQ management UI

```
// rabbitMQ config
  var (
    user       = "guest"
    password   = "guest"
    host       = "rbt"   // name of the rabbitMQ's container
    rabbitPort = "5672"
    queueName  = "hello"
  )
```

Part 4

Automation

```
name: Docker Build

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v2



      - name: Login to the GitHub container registry endpoint
        run: docker login --username sarnik80 --password ${{ secrets.GH_PAT }} ghcr.io


      - name: Build And Push Docker Image ( sender )
        run: |
          make build  DOCKER_IMAGE_NAME="ghcr.io/sarnik80/sender:latest" DOCKERFILE="./sender/Dockerfile"
          make push  PATH_TO_PUSH="ghcr.io/sarnik80/sender:latest"

      - name: Build And Push Docker Image ( receiver )
        run: |
          make build  DOCKER_IMAGE_NAME="ghcr.io/sarnik80/receiver:latest" DOCKERFILE="./receiver/Dockerfile"
          make push PATH_TO_PUSH="ghcr.io/sarnik80/receiver:latest"
```

- **We need to create a directory with the following structure in the project root.**

```
├── .github
│   └── workflows
│       └── main.yml
├── Makefile
```

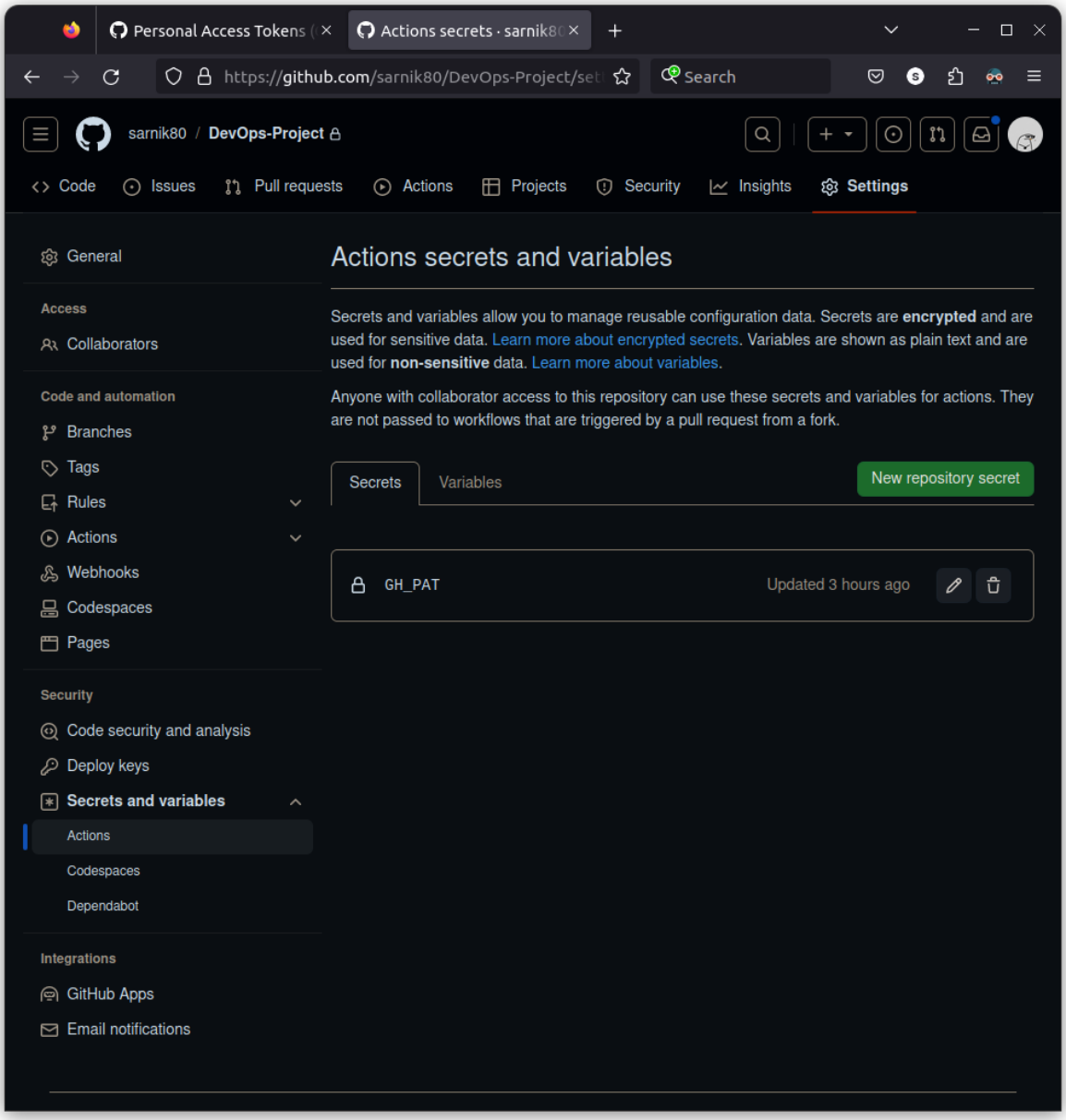
- In the **main.yml** file, we define the structure related to our pipeline.

- The job we defined consists of three steps:

```
uses: actions/checkout@v2
# this is going to pull our code from the repository into the actions runner
# so this way it can build the code
```

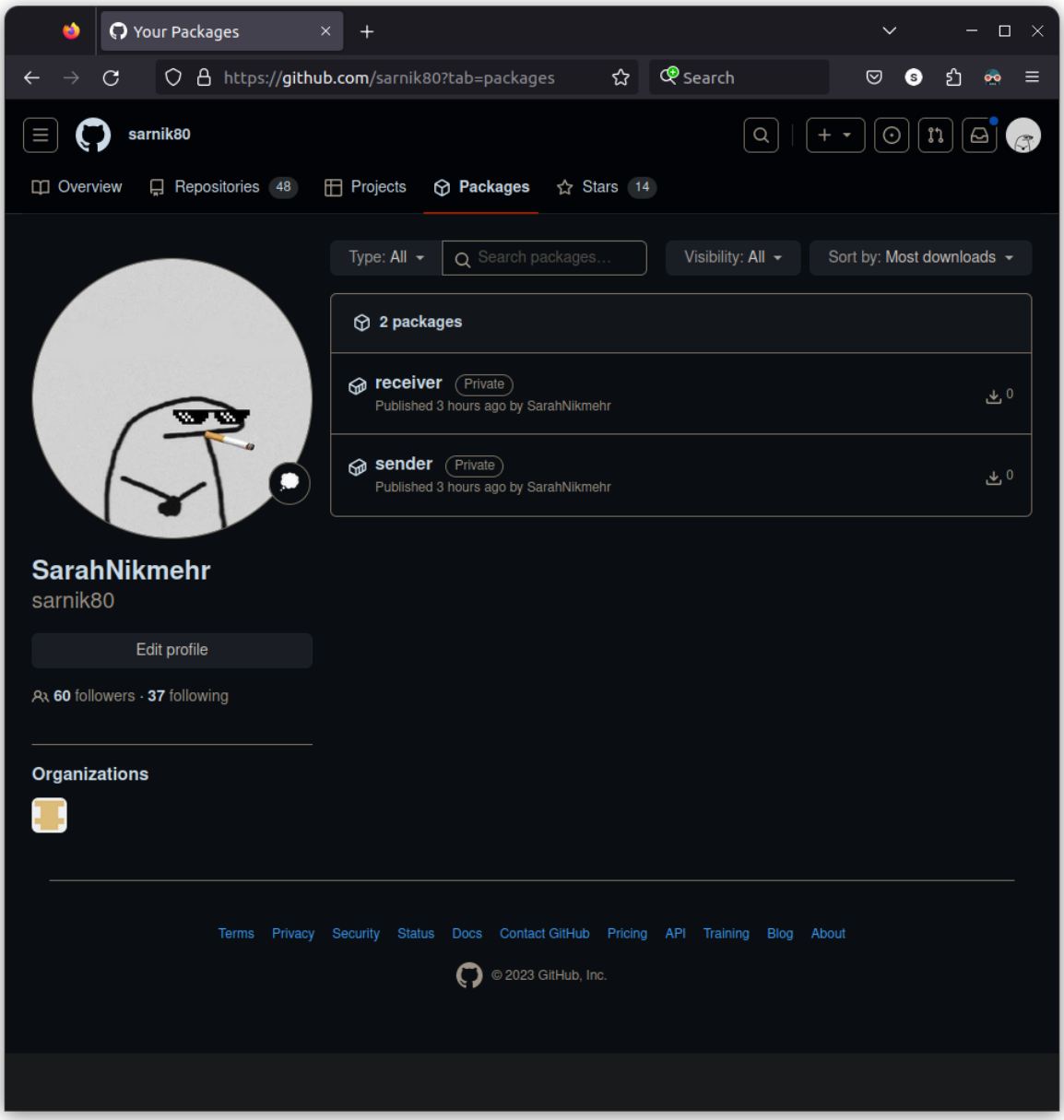
```
 name: Login to the GitHub container registry endpoint
       run: docker login --username sarnik80 --password ${{ secrets.GH_PAT }} ghcr.io

# instead of pasting in the personal access token we're going to use secret that we created on the github repository
```

- We add the personal access token in the repository in the following way after creating it, where we defined the pipeline. As a result, we no longer need to hardcode it in our file.




```
- name: Build And Push Docker Image ( sender )
  run: |
    make build  DOCKER_IMAGE_NAME="ghcr.io/sarnik80/sender:latest" DOCKERFILE="./sender/Dockerfile"
    make push  PATH_TO_PUSH="ghcr.io/sarnik80/sender:latest"
```

- Two final steps are used for building and pushing the images.
- Whenever new changes are pushed to the main branch, these images are built and pushed to GitHub.



[Repository-Link]