Support Documentation

Jirathip Kunkanjanathorn - 32999860 A - 12345678
B - 12345678 C - 12345678 D - 12345678

Contents

Introduction

Documentation Page

Data Workflows

Data Collection

Data Processing

Data Ingestion

Backend

How to run the system

1. Clone the repository or from the zip file
2. Install Docker with Docker Compose
3. Run `docker-compose up -d` in the root directory (Compose V2 do not have - between docker-compose)
4. SSH into the backend container with `docker exec -it backend bash`
5. Migrate the database with `alembic upgrade head` inside the container
6. Insert the data with `python3 scripts/insert_data.py` inside the container

How to shutdown the system

1. Run `docker-compose down` in the root directory

How to test the system

Unit test using pytest with Docker locally

1. Run `docker-compose -d` in the root directory
2. Run `docker exec -it backend pytest` to run the test

Noted that this test will remove all the data in the database where the test is run. The better way to test is to use CI/CD with Github Action.

Unit test the system with Github Action

This workflow is triggered whenever there's a push to the main branch.

1. Checks out repository.
2. Builds and starts Docker Compose services using the dev configuration.
3. Runs pytest within backend service.
4. Shuts down and removes the containers afterward.

Test the system with Postman

How to deploy the system

Build and Push Images to Dockerhub with Github Action

Activated either manually or when there's a push to the deploy branch. The steps include:

1. Checking out repository.
2. Setting up QEMU & Docker Buildx.
3. Logging into Docker Hub using saved credentials.
4. Building Docker images from Dockerfiles (Dockerfile.db & Dockerfile.backend).
5. Pushing these images to Docker Hub.

Setting up GCP Compute Instance

Before deploying on GCP, ensure the VM instance ready. To set up a VM instance:

1. Go to the GCP Console at https://console.cloud.google.com/.
2. Navigate to the Compute Engine and then VM Instances.
3. Click on "Create Instance."
4. Fill out the necessary details like Name, Region, Zone, Machine type, etc.
5. In the Boot Disk section, select an Ubuntu as OS.
6. Under the Firewall settings, make sure to allow HTTP and HTTPS traffic if your application needs to be accessed over the internet.
7. Once filled out, click "Create" to instantiate your VM.
8. SSH into the instance and install Docker and Docker Compose. The instructions can be found here: https://docs.docker.com/engine/install/ubuntu/
9. Set up NGINX for reverse proxy in the instance.

   9.1 Install NGINX: `sudo apt install nginx`

   9.2 Create a new file in /etc/nginx/sites-available/ and name it settle-aid

   9.3 Copy the following configuration into the file:

   ```
   to be filled in later
   ```

   9.4 Create a symbolic link to the file in /etc/nginx/sites-enabled/:

   ```
   sudo ln -s /etc/nginx/sites-available/settle-aid /etc/nginx/sites-enabled/
   ```

   9.5 Test the configuration and restart NGINX:

   ```
   sudo nginx -t
   ```

```
sudo systemctl restart nginx
```

Deploying on GCP

1. SSH into GCP Instance: `gcloud compute ssh <instance-name> --zone <zone>`
2. Change directory: `cd ..` (Optional)
3. Make sure docker-compose.yaml is exist in the directory. The configuration for production is as following

```yaml
version: '3'
services:
  db:
    container_name: settle-aid-db
    image: jirathipk/postgres-vec-geo:latest
    restart: always
    environment:
      - POSTGRES_DB=database
      - POSTGRES_USER=db_user
      - POSTGRES_PASSWORD=password1234
    volumes:
      - database_volume:/var/lib/postgresql/data/
      - dbbackups_volume:/backups
  redis:
    container_name: settle-aid-redis
    image: redis:latest
    restart: always
    command: redis-server --requirepass topmelloredis --loglevel verbose
    volumes:
      - redis_volume:/data

  backend:
    image: jirathipk/settle-aid-backend:latest
    container_name: settle-aid-backend
    user: myuser
    ports:
      - "8000:8000"
    environment:
      - DATABASE_HOSTNAME=db
      - DATABASE_NAME=database
      - DATABASE_PORT=5432
      - DATABASE_PASSWORD=password1234
      - DATABASE_USERNAME=db_user
      - SECRET_KEY=f307994fc80461bf6161eb2bc66b75a6bf32b36b2e3ecd391c2cc17f2be99da0
      - REFRESH_SECRET_KEY=8dfb212cc55a09cb2f215d76619ba07e87c1c89ffc3aab4fe8f21d93e5394c5d
      - REFRESH_TOKEN_EXPIRE_DAYS=7
```

```
    - ALGORITHM=HS256
    - ACCESS_TOKEN_EXPIRE_MINUTES=30
    - MAPBOX_ACCESS_TOKEN=pk.eyJ1IjoiamlyYXRoaXAiLCJhIjoiY2xsdTBoNzQ3MHdndzNzc3luaW03YmNse
    - DOC_USERNAME=topmello
    - DOC_PASSWORD=da7da0df508738e37f18
    - REDIS_HOSTNAME=redis
    - REDIS_PORT=6379
    - REDIS_PASSWORD=topmelloredis
    - USER_CACHE_EXPIRY=3600
    - TRANSFORMERS_CACHE=/usr/src/app/transformers_cache
    - PYTEST_ADDOPTS="-o cache_dir=/usr/src/app/.pytest_cache"
  depends_on:
    - db
    - redis

pgbackups:
  container_name: settle-aid-db-backup
  image: prodrigestivill/postgres-backup-local
  restart: always
  user: postgres:postgres
  volumes:
    - dbbackups_volume:/backups
  links:
    - db
  depends_on:
    - db
  environment:
    - POSTGRES_HOST=db
    - POSTGRES_DB=database
    - POSTGRES_USER=db_user
```

4. Pull the Latest Docker Compose Configuration: `sudo docker-compose pull`
5. Start the Containers: `sudo docker-compose -p settle-aid up -d`

- The -p flag is to set a project name, which can be useful for running multiple environments on the same host
- The -d flag is to run the containers in the background

Actions for Developers:

- Modifications: If there are modifications or additions to packages, update requirements.txt so the Docker build process incorporates these changes.
- GitHub Workflows

How to add table to the database

How to backup and restore the database

Setup Automated Backups

We use the docker-postgres-backup-local image to facilitate our backup tasks. Once the service is started, this will automatically create backups of database daily and maintain.

https://github.com/prodrigestivill/docker-postgres-backup-local

Restore from fresh database

To restore a backup to a fresh database:

```
docker exec -it db psql \
--username=db_user \
--dbname=postgres -c "DROP DATABASE database;"

docker exec -it db psql \
--username=db_user \
--dbname=postgres -c "CREATE DATABASE database;"

docker exec -it db psql \
--username=db_user \
--dbname=database -c "CREATE EXTENSION IF NOT EXISTS postgis;"

docker exec -it db psql \
--username=db_user \
--dbname=database -c "CREATE EXTENSION IF NOT EXISTS vector;"

sudo docker exec \
-it db /bin/sh \
-c "zcat /backups/last/database-latest.sql.gz | \
psql --username=db_user --dbname=database -W"
```

Change permission for backups If necessary, adjust the permissions for the backup files.

```
docker exec -u root -it db-backup chown -R 999:999 /backups
```

Check backup volumes content To inspect the contents of the backup volume:

```
sudo docker run \
--rm -it \
-v settle-aid_dbbackups_volume:/volume_content alpine:latest /bin/sh
```

How to launch new service in the backend

What to do when the system is down

How to monitor the system

Frontend

How to run the system

How to shutdown the system

How to connect to the backend

How to deploy the system

Training and knowledge needed to operate the system

Change management

Resources

- GitHub Repository:
  - Frontend: to be filled
  - Backend: to be filled
  - Documentation page: to be filled
  - Data Wrangling: to be filled
- Project Documentation Page
- Backend API Documentation
- Backend Logging
- Backend UI for testing