Name: Aditya Sharma
Roll No.: CH21B006

DA5402: Assignment #4

Task 1 [20 points]

Create a docker container of your favorite database. Add scripts to initialize your database with the necessary credentials and create the necessary table(s) to save the data with the above-listed columns. So, when the container is up (during the service creation), the database should get created with the supplied credentials, tables are to be created. The database gets created when the docker container is created for the first time. When the container is restarted, there should be a checker that validates the existence of the database and the required tables. If the validation fails, the db & table creation should be redone again. If there is some issue with the initialization, the container should not start up. The required configuration settings from setting up the db should be via environment variables.

File Structure:

```
Assignment4/
├── docker-compose.yml
├── .env
├── .gitignore
├── .gitattributes
├── task1_database/
│   ├── Dockerfile
│   ├── entrypoint.sh
│   ├── db_init.py
│   └── check_tables.py
```

docker-compose.yml:

- References the task1_database folder for building the image and environment variables from .env.

.env:

- Stores configuration settings like POSTGRES_USER, POSTGRES_PASSWORD, and POSTGRES_DB.
- It is not pushed in the GitHub Classroom repository for security reasons.

.gitignore:

- Contains *.env* for the same reason as above.

.gitattributes:

- It contains the line **.sh text eol=lf* to ensure that the shell scripts are handled consistently across all operating systems.

task1_database/Dockerfile:

- Uses the official postgres:latest image as a base.
- Installs psycopg, feedparser and other requirements, copies over scripts, and overrides the entrypoint so we can run custom checks.

task1_database/entrypoint.sh:

- A custom shell script that first checks if the database and table exist through chack_tables.py. If not, it initializes them through db_init.py
- After validation, it hands off control to the official Postgres entrypoint, ensuring the container only starts if setup succeeds.

task1_database/db_init.py:

- A Python script that can connect to the Postgres database, create the required rss_items table if missing, and log the process.
- It is useful for the first initialization of the database.

task1_database/check_tables.py:

- Another Python script to verify that the table and database exist on container restarts.
- If the table is missing, it recreates it or logs errors to prevent partial startup.

How to run code for task_1:

docker compose up --build
or
docker compose build
docker compose up

Task 2 [20 points] + Task 3 [10 Points]

Build an application (a python runnable script here) that would fetch the news articles from the configured RSS feed and push the required fields into the containerized database. The URL of the RSS feed, the dictionary paths of the required fields are to be configurable via environment variables. This way, we can configure the application to fetch from any RSS feed from https://rss.feedspot.com/indian_news_rss_feeds/. For now, we are hanging on to The Hindu. But the expectation is to make it work for others such as, TOI,

Indian Express, NDTV, India Today, News18, etc. Dockerize the application. The application should poll the RSS feed for updates every 10 mins (configurable) and fetch the feed when the data has changed. Mind that data change does not mean

We got two containerized applications. Use docker-compose to create a multi-application (services) dockerization such that the containerized RSS reader app could push the data into the containerized DB. Upon docker-compose up, both services should be created and applications should be started for RSS reading and db storage. Docker-compose down should stop services and remove them. Likewise, docker-compose start and stop should have the necessary functionalities.

File Structure:

```
Assignment4/
├── docker-compose.yml
├── .env
├── .gitignore
├── .gitattributes
├── task1_database/
│   ├── Dockerfile
│   ├── entrypoint.sh
│   ├── db_init.py
│   ├── check_tables.py
│   └── ...
├── task2_rss_reader/
│   ├── Dockerfile
│   ├── rss_reader.py
│   ├── requirements.txt
```

docker-compose.yml (updated):

- References the task1_database folder as well as the task2_rss_reader folder for building the image and environment variables from the updated .env.
- Ensures both containers start together (using *depends_on*) and share environment variables via .env.

.env (updated):

- Now includes RSS_FEED_URL and POLL_INTERVAL environment variables for the RSS reader.
- It is not pushed in the GitHub Classroom repository for security reasons.

task2_rss_reader/Dockerfile:

- Builds a Python container that installs dependencies from requirements.txt
- Sets the default command to run your rss_reader.py script on startup.

task2_rss_reader/rss_reader.py:

- This is the main Python script that periodically fetches the RSS feed from a URL (configurable via .env).
- Parses each entry and inserts the required fields (title, timestamp, link, etc.) into the database.

task2_rss_reader/requirements.txt:

- Lists the Python dependencies for the RSS reader (feedparser==6.0.10, psycopg2==2.9.6).

How to run code fort task 1+2 i.e. task 3:

docker compose up --build
or
docker compose build
docker compose up

Brownie [10 points]

Create a simple containerized web application to read the data from the database with suitable date filters (default is today's news). The web application should show the Title, Image & Summary from the database for every record in the table. Additionally, upon clicking the news title or image, you should open another tab with the actual news using the URL that you got from the db table. The web application should also become a part of the docker-compose environment, which will now have three services. Up, down, start and stop commands of docker-compose should have the necessary functionalities.

File Structure:

```
Assignment4/
├── docker-compose.yml
├── .env
├── .gitignore
├── .gitattributes
├── task1_database/
│   ├── Dockerfile
```

```
│   ├── entrypoint.sh
│   ├── db_init.py
│   ├── check_tables.py
├── task2_rss_reader/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── rss_reader.py
└── brownie_web_app/
    ├── Dockerfile
    ├── requirements.txt
    ├── main.py
    └── templates/
        └── index.html
```

docker-compose.yml (updated):

- References the task1_database folder, task2_rss_reader folder and the brownie_web_app folder for building the image and environment variables from the .env.

brownie_web_app/Dockerfile:

- Installs dependencies from requirements.txt and copies main.py plus the templates/ folder.
- Exposes a port (5000) and runs main.py via Python.

brownie_web_app/reuiqrements.txt:

```
fastapi==0.95.0
uvicorn==0.21.1
jinja2==3.1.2
psycopg2==2.9.6
```

brownie_web_app/main.py:

- A FastAPI application that queries the rss_items table, filtering by date.
- Displays title, image, and summary in HTML, linking to the article's original URL.
- It ends with

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=5000)
```

- This helps in running the web app using unicorn when python main.py is implemented in the command line.

brownie_web_app/templates/index.html:

- Jinja2 template that loops over the articles from the database, showing each one's title, image, and summary.
- Clicking on the title or image opens the original news link in a new tab.

How to run all the code files:

docker compose up --build
or
docker compose build
docker compose up

- After doing this, we can open the port using http://localhost:5000/.
- The default date filter is today. It can be changed by adding ?date_str=YYYY-MM-DD in the URL.

Below is an image of how it showed up on my system: