Docker Assignment DSA3101

AY 23/24 Sem I

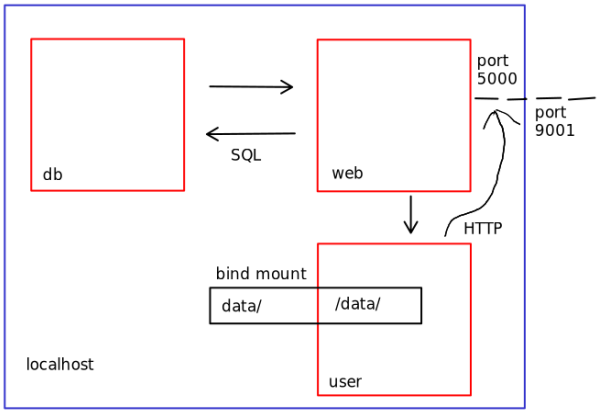
Contents

Learning Outcomes

1. To familiarise with writing Dockerfiles and Docker compose YAML files.
2. To get comfortable reading about how to
   - run images from Docker Hub.
   - run a useful Linux utility program.
3. To generate basic API endpoints in a Flask Application.
4. Navigate through a linux environment and locate relevant files
5. To practice breaking a problem down, testing individual parts and then putting them together.

Overview

The task is to get these three containers up and running:



1. `db`: A MySQL database containing information about customers. This is a subset of the database that we had for our *Exploring Data* class. The image containing the database can be found from this Docker

Hub site

2. `web`: A Flask application serving on port 5000 with 4 end points:
   - `/` - displays the landing page for the Solace Trust website
   - `/customers` - displays the table of customers that is currently in our database in a webpage
   - `/get_customers_gender_segment` - returns customers from a select query based on two particular columns: gender and segment.
   - `/add_customer` - a POST request that inserts a row into the database.
   - `/delete_customer` - a DELETE request that deletes the stated customer from our database
3. `user`: A user container running a cron job that will run a python script to upload a csv file into the database every day at 2359hrs.

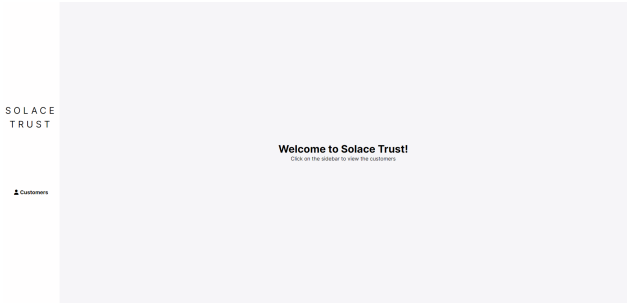


Figure 1: Landing and customer pages

Figure 2: Landing and customer pages

File descriptions

These are the files provided to you:

```
|- customers.py
|- data/
|- docker-compose.yml
|- flask_dockerfile
|- init.sh
|- requirements.txt
|- static/
|- templates/
|- upload.py
|- user_dockerfile
|- sample_crontable.txt
```

1. `customers.py` contains the Flask application. It will have to be copied into the flask image.
2. `data/` is a folder that needs to be bind-mounted to the user container. It contains a sample `users.csv`.

3. `docker-compose.yml` should contain the specifications to get the three containers up and running.
4. `flask_dockerfile` contains instructions to build the Flask application.
   - Note that it is not named Dockerfile, but this is ok.
5. `init.sh` needs to be copied into the user image.
6. `requirements.txt` needs to be copied into the Flask image.
7. `static/` and `templates/` are directories that need to be copied into the Flask image. They assist in generation of the HTML pages displaying the records in the database.
8. `upload.py` contains the python code to upload a csv file into the database. This file will have to be copied into the user image, and will be executed by the cron job at 2359 every day.
9. `user_dockerfile` contains instructions to build the user image.
   - Again, note that it is not named Dockerfile, but this is ok.
10. `sample_crontable.txt` contains a sample crontable. It has to be modified before you use it.

Cron Job

A **cron job** is a task automated using **cron**, which is a scheduler tool on Linux. With `cron`, we can automate tasks to run periodically (for e.g. yearly, monthly, or even every minute). This could be particularly useful when we want to create backup, update software or even monitor software.

Here is how cron works:

1. `cron` runs in the background on Linux.
2. We add entries to a *cron table*, that specifies what program to run, and at what frequency.

There are two ways to create and entry in the cron table:

a) Using `crontab -e` to manually edit the cron table, or
b) Providing a text file to `crontab`.

Tasks

1. Add the decorators to `customers.py`, and the SQL queries for `/get_customers_gender_segment` and `/delete_customer`.
2. Complete the `docker-compose.yml` file to get the containers running and working together. Refer to the diagram in the Overview section above.
3. Complete `flask_dockerfile` according to the instructions in the file.
4. Complete `user_dockerfile` according to the instructions in the file.
5. In order to get the cron job working, the cron table on the user container needs to be updated. One approach is to
   - Modify `sample_crontable.txt` to run `upload.py` every day at 2359hrs.
   - Copy this modified file into the container.
   - Use `crontab` on the container to load the file into the cron table.
   - You don't have to modify the time/timezone on the containers.

   There are other approaches to editing the cron table, that do not use a text file as input. You can find them online.

Place the above 4 files (and others that your method needs e.g.another requirements file, or the modified `sample_crontable.txt`) into a zip file and submit them as your assignment **on Canvas**.

The deadline for this assignment is **Friday Oct 6th 2359hrs**. Late assignments (no matter how close to the deadline) and email submissions will not be graded.

Notes/Hints

- Only the Flask container should be accessible to the outside world, and only through port 9001.

- The database image does not need any further modification; it just needs to be pulled and run.
- Do not delete or modify existing lines in any of the files. You only need to add new ones.
- There is no naming convention for the zip file to be submitted. If you made a mistake and need to submit a new version, you can do so. We will only grade the latest one.
- Make sure that you test all your end points, and that the cron job is working before you submit your files.

References

1. More on cron:
   - https://www.hostinger.com/tutorials/cron-job
   - https://www.digitalocean.com/community/tutorials/how-to-use-cron-to-automate-tasks-ubuntu-1804
2. Docker compose reference:
   - https://docs.docker.com/compose/compose-file/compose-file-v3/
   - Examples using docker compose
3. Dockerfile reference:
   - https://docs.docker.com/engine/reference/builder/