FILL IN : Lab 2

*Lovepreet Singh*

Questions from the lab

*In the lab assignment, you’ll see several questions in red boxes. Paste those questions and their respective answers below. Make sure your answer is concise and well-formatted. You may submit this as e.g. a screenshot of a filled-out cell in a copy of the Notion document (e.g. with code, so that code formatting is maintained).*

**Q1 :** Add your own comments to explain in your own words what this Dockerfile does.

**A1 :** It installs Flask, uWSGI and sets everything up.

**Q2 :** To obtain an image, we have to create a so-called build. The basis for this is a Dockerfile. With which command can you do this execute this as well.

**A2 :** docker build -t my-flask-app web

**Q3 :** Reflect on whether it is feasible to minimise the number of layers. Once you have made your assessment, take the necessary steps to reduce them.

You need to reduce the Docker Layers in your Dockerfile by removing or combining certain layers.

**A3** : Uses --no-cache-dir with pip to reduce image size. Keeps the order of instructions optimal for layer caching.

**Q4 :** Can you figure out a way to replace the previous commands (docker build and docker history) with an image name instead of a container\_id ?

Try to tag your image as following: your-name/frontend. You do not have to specify any version (by default that will be latest). The advantage of Tagging would be that your image now has a specific tag it can use.

**A4 :** docker history singh-lovepreet/frontend

docker run singh-lovepreet/frontend

**Q5 :** What does the -p and -it (can also be written as -i -t) flags do? You should search for this in the official Docker documentation.

**A5 :** -p maps port 5000 on your computer to port 5000 in the container, so you can access the app from your browser.

-it allows you to interact with the container’s terminal (it combines --interactive and --tty).

**Q6 :** In your opinion, is the build process fast? Remember you just added a small extra endpoint to the application …

**A6 :** It could be faster

**Q7 :** What did you change to the Dockerfile to make it build faster?

**A7 :** look into the Dockerfile (combining some steps, ... )

**Q8 :** Take a look at prestart.sh and explain what's going on in this script.

**A8 :** The script ensures the database schema is current, then starts the main application process.

**Q9 :** What is the advantage of an official image? What other types are there?

**A9 :** The advantage of using an official image is that it is maintained, tested, and regularly updated by Docker or the upstream project’s maintainers.

**Q10 :** Will this work? What's going wrong?

**A10 :** POSTGRES\_DATABASE should be POSTGRES\_DB for the official Postgres image.

POSTGRES\_HOST and POSTGRES\_PORT are not used by the official Postgres image; they are typically used by applications connecting to Postgres, not by the database itself.

If your-name/api is not a Postgres image, these environment variables may not have any effect unless your app is coded to use them.

If you want to run a Postgres container, use the official image: postgres.

If you want to run your own API that connects to Postgres, you should only set the environment variables your app expects.

**Q11 :** Why can't you reach the API from the front-end? How could you solve this? You don't have to implement your solution.

**A11 :** Network/Host issues

**Q12 :** What does each line do?

**A12 :**

# Defines the services (containers) to be run.

# Name of the service (here, "api").

# Builds the Docker image from the Dockerfile in the ./api directory.

# Maps ports between the host and the container.

# Maps port 8080 on the host to port 8080 in the container.

# Sets environment variables for the container.

# Sets the PostgreSQL username.

# Sets the PostgreSQL password.

# Sets the PostgreSQL database name.

# Sets the PostgreSQL port.

# Sets the PostgreSQL host (likely another service).

# Sets the application port to 8080.

**Q13 :** For the API, you utilised ports while for the frontend, you used expose. Could you explain the distinction between the two, and what factors might lead you to avoid using ports for the frontend?

**A13 :** Use ports when you need external access; use expose for internal, container-to-container communication only.

* If the frontend is only meant to be accessed by other containers (like a reverse proxy or API), not directly from your host.
* To reduce security risks by not exposing unnecessary ports to the outside world.
* To keep your host environment cleaner and avoid port conflicts.

**Q14 :** Is the postgres:17.0-alpine the same as the latest version?

**A14 :** No, the code mentions the 17 version specifically not the latest ( if 1 is available )

**Q15 :** What purpose do [volumes](https://docs.docker.com/storage/volumes/) serve? Reflect on the concepts covered in the theory session. Additionally, consider the topic of layers. Can you recall which layers allow data to be written, and which ones do not?

**A15 :** Volumes in Docker are used to persist data generated by and used by containers. They allow data to be stored outside the container’s writable layer, so the data remains available even if the container is removed or recreated. This is especially important for databases like PostgreSQL, where you don’t want to lose your data when updating or restarting containers.

**Q16 :** What are the steps to initiate the entire setup, including the building of the necessary image(s)?

**A16 :** Steps to Initiate the Entire Docker Compose Setup

1. Open a terminal\*\* and navigate to the directory containing your `compose.yaml` file:

cd c:\VSCode\docker-new\api

2. Build images and start all services:

docker compose up --build

3. (Optional) To run in detached mode (in the background):

docker compose up --build -d

4. To stop and remove all running containers, networks, and volumes created by `up`

docker compose down

**Q17 :** Paste you compose.yaml file. Start all the services, probably best in the background (-d) and verify if everything works as expected (docker compose ps).

**A17 :** it works.

**Q18 :** What is a proxy?

**A18 :** A proxy is an intermediary server that sits between clients (such as browsers) and other servers. It forwards client requests to the appropriate server and then returns the server’s response back to the client.

**Q19 :** What did you change to compose.yaml?

**A19 :**

**Q20 :** Push the images on Docker Hub. Paste your repository URL.

**A20 :**

**Q21 :** It will become apparent that there's an issue. Identify the specific malfunction and its underlying cause.

**A21 :**

Questions to answer for every lab

**What did you learn?**

*Fill in your three take aways that you learned during this lesson.*

* fdgh

**Givethree interesting exam questions about the contents of the lab and/or the theory**.

*Thinking about this will make sure you remember the key take-aways and important details better and longer.*

* C
* sdf

**Check the following:**

* I have made the entire lab assignment (be careful, some labs consist out of two or more Notion documents!).
* I have answered all the questions from the lab assignment.
* I have submitted my code as a zip file and/or as a link to a *public* Git repository.
* <For labs on Azure> I have shut down any resources that are in use, in order to avoid unexpected costs.