ACIT4640 - in class evaluation

In class evaluation: Docker

In this evaluation, you will have to:

- create `Dockerfile` and `docker-compose.yml` files for a web application
- design and implement a CI/CD pipeline to build and publish images on the Docker Hub

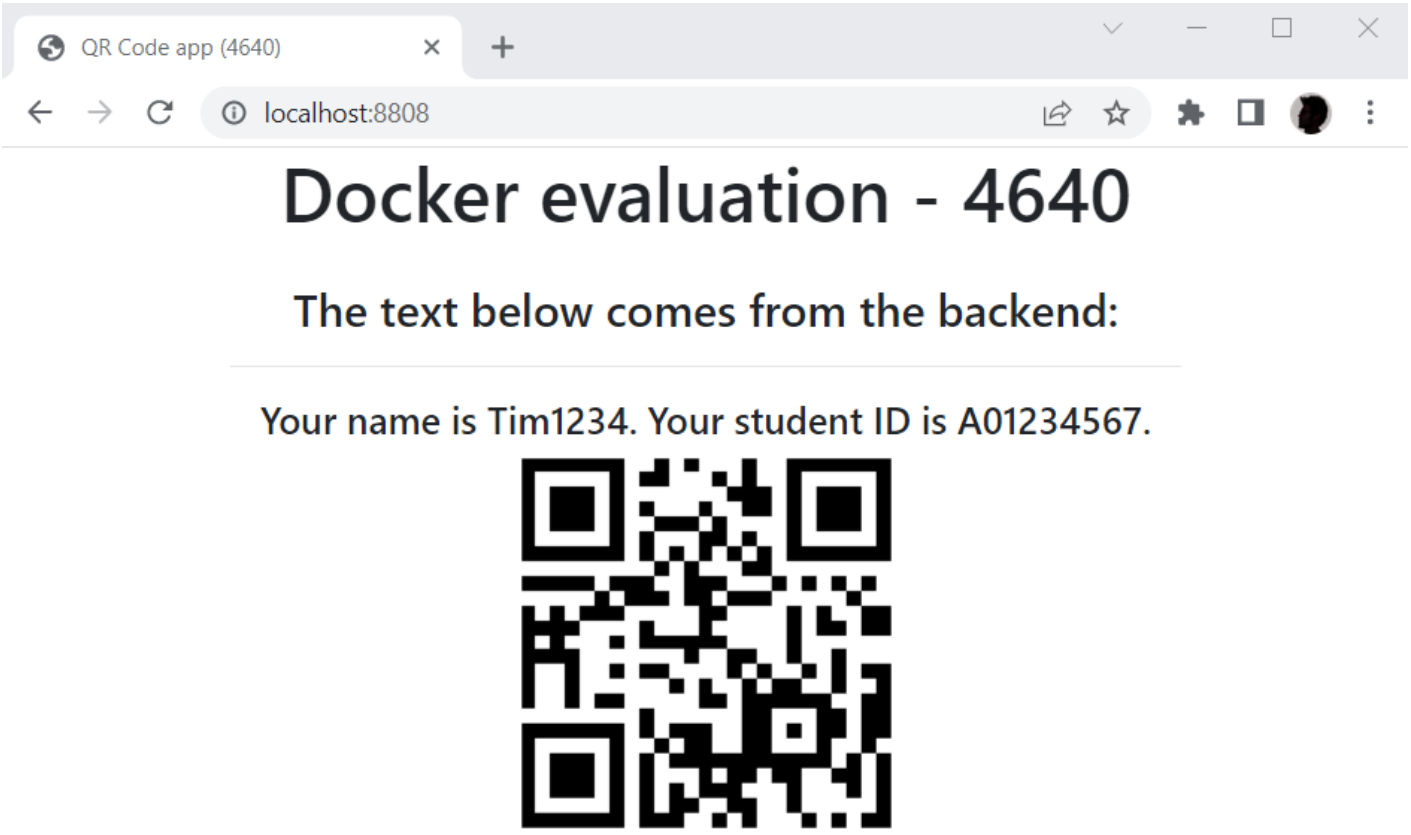
The souce code for the application is available on D2L.

Application structure

The application is built with Flask and requires Python 3. It has two components:

- the frontend (an HTML file, and static files for CSS and JavaScript)
- the API/backend (runs with Python)

When deployed, the application displays a webpage similar to:



Make the backend work

Instructions to setup the Flask application

- Create a user to run the application.
- The application has a single dependency: `flask`. Make sure you `pip install flask`.
- Copy the files in the `app` folder to a relevant location in your container.
- The application requires a plain text file called `student.txt` **in the same folder as the Python files**.
- The `student.txt` must contain 2 lines:
  - the first line is the name of the student
  - the second line is the student ID
- The application can be run with `python web.py` (adjust the path as necessary).
- It will listen on all interfaces, on port 5000.

Instructions: Dockerfile for the application

- Write a `Dockerfile` that executes the steps above.
- Your container must dynamically create the `student.txt` **at runtime**, based on the following environment variables:
  - `STUDENT_NAME` for the name of the student (line 1)
  - `STUDENT_ID` for the student ID (line 2)
- Your container must not use bind mounts.

Make the static web server work

- Create a `Dockerfile` for the static web service.
- The container may use `nginx`.
- It must serve all the files located in the `html` folder (HTML files + static files for CSS and JS).
- It must also forward all HTTP requests whose URL begins with `/eval` to the Python API.
- The Python API URL must be configurable **at runtime** using the environment variable `API_URL`.
- Your container must not use bind mounts.

Create a repository and push your files to it

- Create a Git repository (you may use GitHub, GitLab, or any provider - as long as you can run CI/CD pipelines with Docker). **Your repository must be public**.
- Commit and push your files:
  - all the required application files
  - all your `Dockerfile` and supporting files (startup scripts, templates, configuration files, etc).

Develop a CI/CD pipeline that builds and pushes Docker images

- Create a CI/CD pipeline in your git repository.
- Upon any commit on any branch, the pipeline must:
  - build both images (application and static web service)
  - push them to the Docker Hub under your account
  - each image must be distinct
    - do not use tags / labels but make two separate images, each with their own name
    - For example:
      - `tim/eval4640-web:latest`
      - `tim/eval4640-app:latest`

Test and run your CI/CD pipeline

- Commit to your repository. You may make an empty commit.

    - for example: `git commit --allow-empty -m "Run pipeline"`)
- The pipeline must run successfully.
- The images must be created and available on the Docker Hub.

Create the `docker-compose.yml` file

- Create a Docker Compose file that runs the two containers.
- You must use the images that were build and pushed to the Docker Hub at the previous step.

A starting point for your `docker-compose.yml` is below (**note: the file is not complete - you must make changes to it!**):

```
---
version: '3'

services:
    web:
        image: timg/eval4640-web:latest
    app:
        image: timg/eval4640-app:latest
```

Submission & grading

Create a ZIP file containing:

- the `docker-compose.yml` file
- a copy of your git repository files
- a text file containing:
    - the link to your git repository
    - the link to your Docker Hub images
    - the link to the application when running the Docker infrastructure with your docker-compose file (ex: `http://localhost:12345`)
- a screenshot showing the app running in a browser on your computer, with your name, student ID, QR code and page URL visible

> Names are up to you, as long as they are consistent and the setup works. The Docker Hub images must have been built and published during the exam time for your submission to be valid.

Grading rubric

Item	Marks

Item	Marks
The app is running correctly when running `docker-compose up -d` using images from the Docker Hub	4
Frontend container setup	2 (total)
- nginx configuration	1
- container parametrization	1
Application container setup	3 (total)
- application setup	2
- container parametrization	1
CI/CD pipeline	1
docker-compose file and variables	2
Setup quality and best practices (`EXPOSE`, file structure, simplicity)	2
TOTAL	**14**