CSCI4145/5409 - Summer 2024
A1- Docker

Important note

Carefully reading the assignment description is important, as even small oversights could lead to zero marks for your assignments. Take your time to scrutinize every word and ensure you comprehend all the requirements. If you have any doubts, don't hesitate to ask questions in the Teams channel dedicated to this assignment.

Reminder

This is an individual assignment. You are not allowed to collaborate with anyone else when completing this assignment. You can borrow code and configuration snippets from internet sources that are not from students in this class, however that code must be cited and include comments for how you have modified the original code and does not count as code you have written.
While I've mentioned that you're allowed to use ChatGPT or similar tools for your term project, I strongly advise coding the assignments independently. These tasks are simple enough for you to complete without assistance. If you rely on tools to generate code for these relatively straightforward programming assignments, it could impede your ability to qualify for entry-level developer positions in the industry. This is an opportunity for you to practice and improve your skills. However, if you still choose to use ChatGPT or similar tools for assignments, that's your decision and acceptable.
I will not accept negotiations regarding marks lost due to your errors.

Introduction

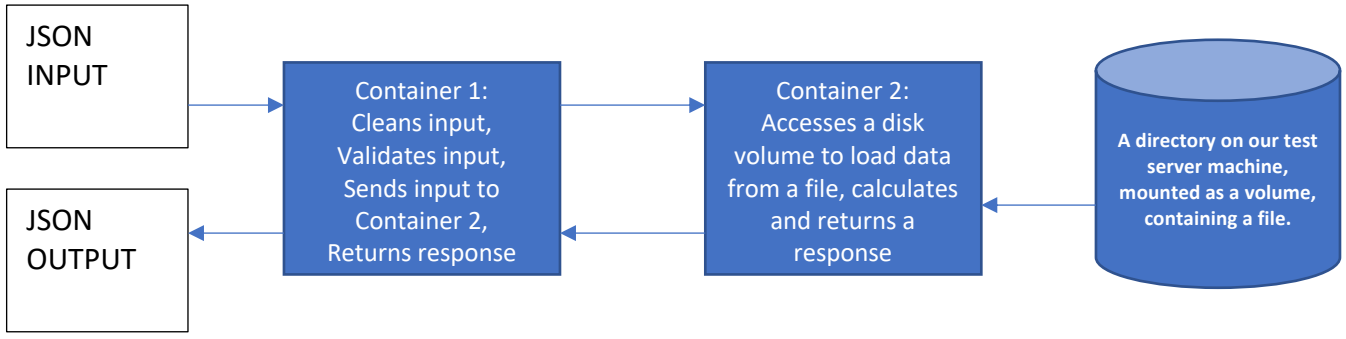
This assignment will measure your understanding of containerization, and specifically containerization done through Docker. The task you are asked to do is not complicated programmatically, in fact it's a bit silly and a made-up exercise to give you Docker experience, however its goal is to assess whether you have met the learning outcomes of understanding Docker and its usage. This assignment assures us that you have attended the Docker tutorials and learned its usage or found some other way to learn Docker.

Learning Outcomes

- You have a successful working install of Docker
- How to build a container
- How to open ports and communicate with other containers through a docker network
- Using JSON as a text-based data interchange format
- Making small webservices with existing official Docker images
- Creating Dockerfile's and learning Docker commands used in container app development
- Using docker compose to build multi-container microservice based architectures
- Developing the courage to dive into complicated cloud computing tools

Requirements

You will build two simple webapp containers that communicate to each other through a docker network to provide more complex functionality, a very small microservice architecture. When you are finished your system will look and function like this:



JSON Input

Your first container will receive JSON with the following format:

```
{
   "file": "file.dat",
   "product": "wheat"
}
```

The intent of the message is for your microservice architecture to perform a calculation and return a response. In this case we want you to load the file passed in from the mounted disk volume, parse the file as a comma-separated-value (CSV) file, and sum all values where the 'product' column matches the value passed in the product argument.

For example, if file.dat contained:

```
product,amount
wheat,10
wheat,20
oats,5
barley,6
```

The JSON above should cause your program to return 30 as the sum.

JSON Output

IF the filename provided via the input JSON is found in the mounted disk volume, the calculation is performed (**note: sum's value is an integer not a string)**:

```
{
   "file": "file.dat",
   "sum": 30
}
```

If a filename is provided, but the file contents cannot be parsed due to not following the CSV format described above, this message is returned:

```
{
   "file": "file.dat",
   "error": "Input file not in CSV format."
}
```

Note: CSV format refers to contents of the file and not the file extension.

If a filename is provided, but not found in the mounted disk volume, this message is returned:

```
{
   "file": "file.dat",
   "error": "File not found."
}
```

If the file name is not provided, an error message is returned:

```
{
   "file": null,
   "error": "Invalid JSON input."
}
```

Container 1

Your first container's role is to serve as an orchestrator and gatekeeper, making sure that the input into the system is clean and valid. It must:

1. **Listen on exposed port 6000** for JSON input sent via an **HTTP POST** to "**/calculate**", e.g. "http://localhost:6000/calculate"
2. Validate the input JSON to ensure a file name was provided, if the "file" parameter is null, return the invalid JSON input result.
3. Verify that the file exists, if it does not exist return the file not found error message.
4. Send the "file" and "product" parameters to container 2 (you don't have to use JSON to do this, do it however you like, but I recommend JSON) and return the response from container 2.

Container 2

The second container's role is to listen on another port and endpoint that you define within your docker network for requests to calculate product sums. It's a bit silly to separate them for such a trivial task, but we're trying to get you to learn multi-container environments and so you must divide your program into these two containers. Your second container must:

1. Mount the **host machine** directory '.' to a docker volume
2. Listen on an endpoint/port you define to respond to calculate requests:
    a. Load the file into memory
    b. Parse the CSV file with a CSV library of your choosing for whatever programming language you are using inside your docker containers
    c. Calculate the sum of all rows matching the given product parameter
    d. Return the sum in the appropriate JSON format, or an error indicating the file is not a proper CSV file in the appropriate JSON format.

Additional Requirements

1. You must push both your containers to a Dockerhub account **you** create (this is free), **if you don't push them to Dockerhub we won't have them and won't be able to run your program, so test this.**
2. You must prepare a docker-compose.yml file that defines a docker network and runs the two containers from your dockerhub deploy, remember container 1 must be listening on local port 6000 and you must mount the local volume '.' (the current directory) to get access to the files I provide.
    a. **NOTE: The version keyword should be ABSENT in your docker-compose.yml file (indicating to use the latest version). You must ensure you use the latest spec for building your docker-compose.yml file.**

How To Submit

Each of you is provided with a git repository for individual assignments in this course. You can find your repository on gitlab: https://git.cs.dal.ca. To setup your repository, go to gitlab and click on your project. Follow the steps described in your repo on Gitlab to initialize the repo.

Create a folder in your repository labeled **A1**. Put your docker-compose.yml file in the A1 folder. Also include the source code for the code running in your docker containers. Your source code can be in subfolders if you like but your docker-compose.yml file must be in the root A1 folder. If your docker-compose.yml file is not named properly (i.e., it's named something else) the server's tests won't work, and you will get 0. The server cannot not manually rename your files.

Make sure you commit your files to the **main** branch and push your changes to gitlab. Also, make sure to keep **main** branch as default.

A common mistake students make here is they submit a docker-compose.yml file they used to build their container instead of a docker-compose.yml file that pulls the container from Dockerhub. Your program just won't work if you do this.

Marking Rubric

In this class I'm not very concerned about the quality of the code you write because this is not a programming course, if you write bad quality code it is you that will suffer in maintaining and supporting it (especially on your term assignment). I care that you can meet the learning objectives defined at the top of this document, and that you satisfy the requirements. I can verify this by simply running your containers and verifying responses.

Your submission will be marked by our class assignment server that our scripting TA will program and maintain. You will need to send a POST request with some JSON to a URL we provide to you that begins a chain of events:

1. The server pulls your individual git repository to a temporary folder
2. In the server's temporary folder it will change directory to your repository's **A1** folder.

3. Copy docker-compose.yml and some test files into a test directory
4. Run "docker-compose up" in the test directory
5. HTTP POST an **invalid** JSON input to http://localhost:6000/calculate and verify that you return the invalid JSON input response in your JSON response
6. HTTP POST a **valid** JSON input to http://localhost:6000/calculate and verify that you return the correct sum in your JSON response
7. HTTP POST a **valid** JSON input with a file name that does not exist in the mounted volume and verify that you return the file not found JSON response
8. HTTP POST a **valid** JSON input with a file that contains data not in the correct CSV format and verify that you return the input file not in correct format response.
9. Run "docker-compose down -v --rmi all" to shut things down and remove your images.
10. Delete your repository and perform cleanup

Your mark is entirely based on the success of the tests conducted in steps 5, 6, 7 and 8:

- **Pass all 4 = 100%**
- **Pass 3 = 80%**
- **Pass 2 = 70%**
- **Pass 1 = 50%**
- **Any other result = 0%**

Note: this includes inability to run your code because it isn't in the right place or doesn't work or you don't return your JSON response in the correct format.

JSON to send to My App's /docker/start

```
{
    "banner": "<Replace with your Banner ID, e.g., B00123456>",
    "csid": "<Replace with your CSID, same as your repository name>"
}
```

Note: The application processes requests by placing them in a queue, when you send a request to /docker/start, your request will be put in the processing queue and will be evaluated in order along with other student's submissions. This takes time, for this reason it is extremely important that you do not wait until the last minute to submit your assignment if you want to receive your results before the deadline. Waiting until the last minute means you'll only have one chance to get it right.

JSON to send to My App's /docker/results

When you want to check the results of our server's tests of your submission you'll post to our server's /docker/results endpoint with the following JSON:

```
{
    "banner": "<Replace with your Banner ID, e.g., B00123456>",
    "all": <true to get all evaluated requests' feedback; false to
retrieve feedback for the last evaluated request>
}
```

The server will respond with the current state of your submission, either returning the feedback from your last call to /docker/start or, if you specify true for the “all” parameter it will return the results of all your submissions to /docker/start.
Professionals read requirements carefully and ensure that their programs meet requirements exactly. Especially when we are defining the intercommunication mechanisms when we are building microservice architectures we need to ensure we stick to our expected inputs and outputs. Pay very close attention to the message formats above and ensure you match them identically. ***We will be grading your submissions with an automated script on our testing server, if you do things like leave off periods or change the names of keywords or put your error messages in the sum keyword or other silly things that aren’t in the requirements you will get zero. Test your work! Ensure your meet requirements.***

Because your mark is entirely results based it makes sense for you to spend time testing to ensure your docker-compose.yml is properly configured to work on my machine! It is **very** common for students to make silly mistakes and submit files that they used for development instead of the final files and things like that. Therefore, I recommend that you:

- Use the 'docker image ls' and 'docker image rm' commands to delete your local images used during development / testing.
- Copy your docker-compose.yml to a temporary folder
- Place a test file in the folder
- Run 'docker-compose up' to verify that your images download properly from dockerhub
- Then use a tool like Postman to POST some testing JSON input to your container and verify that you receive the correct responses