CSCI5409 – Fall 2023
A1: Docker Assignment
Due date: 11:59pm, Sep 27, 2023

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.

Important notice: Thoroughly reading and complying with the assignment instructions is of utmost importance. Minor errors stemming from failure to follow the instructions can lead to a complete loss of marks. For instance, placing your code in the wrong folder can lead to a situation where you receive a grade of zero, even if your code is flawless.

Introduction

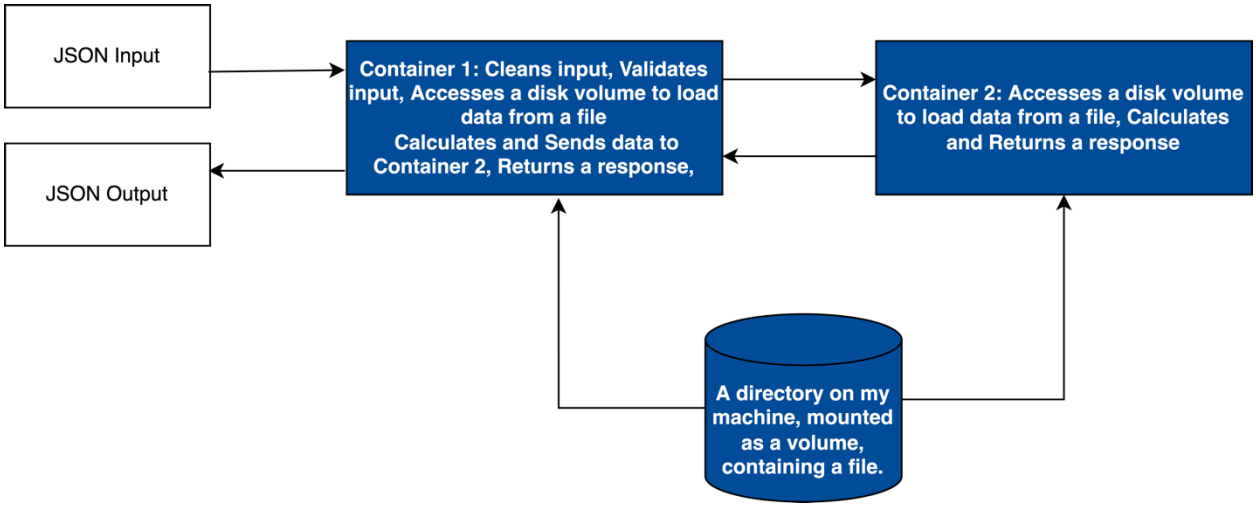
This assignment will measure your understanding of containerization, and specifically containerization done through Docker. The task you are asked to do is not complicated programmatically, however it should assess whether you have met the learning outcomes of understanding Docker and its usage. This 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 to retrieve latest location:

```
{
   "file": "file.dat",
   "name": "Ronald",
   "key": "location"
}
```

Your first container will receive JSON with the following format to retrieve latest temperature:

```
{
   "file": "file.dat",
   "name": "Ronald",
   "key": "temperature"
}
```

The intent of the message is for your microservice architecture to parse the data 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 return latest location ('latitude' and 'longitude') or return latest 'temperature'.

For example, if file.dat contained:

```
name,latitude,longitude,temperature
Reuel,45.5045,-73.5561,17
Ronald,44.6374,-63.5872,19
John,30.3284,35.4443,21
Ronald,44.6457,-63.5696,15
```

The JSON above would cause your program to return `44.6457, -63.5696` as the latest location or return `15` as the latest temperature.

JSON Output

If the filename provided via the input JSON is found in the mounted disk volume, the latest location is retrieved as (**note: latitude and longitude value is a float not a string)**:

```
{
   "file": "file.dat",
   "latitude": 44.6457,
   "longitude": -63.5696,
}
```

If the filename provided via the input JSON is found in the mounted disk volume, the latest temperature is retrieved as (**note: temperature value is an int not a string)**:

```
{
   "file": "file.dat",
   "temperature": 15
}
```

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."
}
```

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 and it can retrieve location information of a user. It must:

1. Mount the host machine directory '.' to a docker volume.
2. **Listen on exposed port 6000** for JSON input sent via an **HTTP POST** to "**/user-info**", e.g. "http://localhost:6000/user-info"
3. Validate the input JSON to ensure a file name was provided, if the "file" parameter is null, return the **invalid JSON input** result.
4. Verify that the file exists, if it does not exist return the **file not found** error message.
5. If the input JSON contains 'location' as key:
    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 Retrieve the latest location (latitude and longitude) of a user based on 'name' parameter.
    d Return the location value in the appropriate JSON format, or an error indicating the file is not a proper CSV file in the appropriate JSON format.
6. If the input JSON contains 'temperature' as key, then send the "file", "name" and "key" 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 retrieve temperature information. It must:

1. Mount the **host machine** directory '.' to a docker volume.
2. Listen on an endpoint/port you define to respond to user-info 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. Retrieve the latest temperature of a user based on 'name' parameter.
    d. Return the temperature value 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 Setup your repository, go to gitlab and click on your project. Follow the steps described 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 script won't work, and you will get 0. We will not manually rename your files.

Make sure you have you commit your files to the **main** branch and push your changes to gitlab. Also, make sure that the **main** branch is default branch. This must be done before the assignment deadline. Code pushed after the deadline will not be graded.

Marking Rubric

In this class I'm not very concerned about the quality of the code you write, 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 a Python script the scripting TA will write, the script will do the following:

1. Pull everyone's individual git repository
2. Change directory to your repository's **A1** folder.
3. Copy some test files into the directory
4. Run "docker-compose up" in the directory
5. HTTP POST an **invalid** JSON input to http://localhost:6000/user-info and verify that you return the invalid JSON input response in your JSON response
6. HTTP POST a **valid** JSON input (with `"key": "location"`) to http://localhost:6000/user-info and verify that you return the correct location information in your JSON response

7. HTTP POST a **valid** JSON input (with `"key": "temperature"`) to http://localhost:6000/user-info and verify that you return the correct temperature information in your JSON response
8. 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
9. 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.
10. Run "docker-compose down -v --rmi all" to shut things down and remove your images.

Your mark is entirely based on the success of the tests conducted in steps 5, 6, 7, 8 and 9:

- **Pass all 5 = 100%**
- **Pass 4 = 80%**
- **Pass 3 = 60%**
- **Pass 2 = 40%**
- **Pass 1 = 20%**
- **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.**

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, if you do things like leave off periods or change the names of keywords or put your error messages in the latitude, longitude or temperature keyword or other silly things that aren't in the requirements you will get zero and we will not give you an option to recover your grade.***

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