# CSCI4145/5409: Docker 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.**

## Introduction

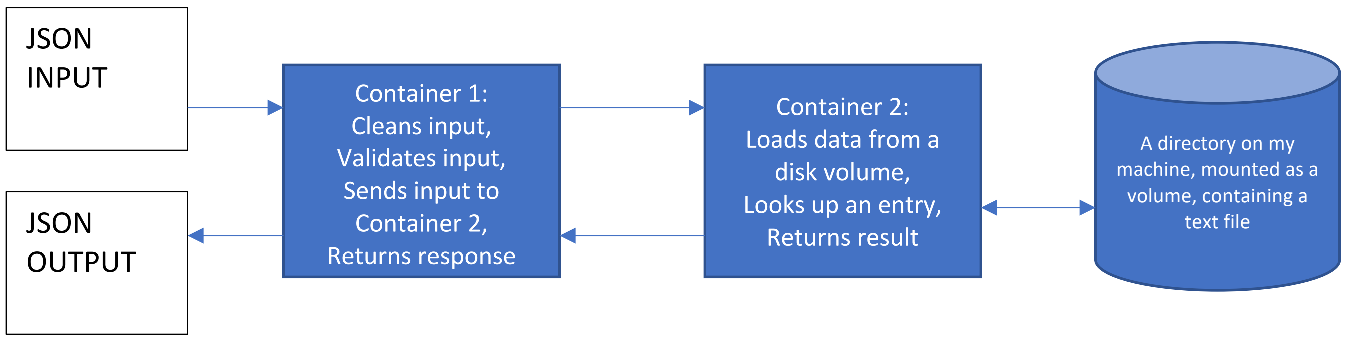
This assignment will measure your understanding of containerization, and specifically containerization done through Docker. The task you are asked to do is not complicated programmatically, however it should assess whether you have met the learning outcomes of understanding Docker and its usage. This 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
JSON OUTPUT
Container 1: Cleans input, Validates input, Sends input to Container 2, Returns response
Container 2: Loads data from a disk volume, Looks up an entry, Returns result
A directory on my machine, mounted as a volume, containing a text file

## JSON Input

Your first container will receive JSON with the following format:

```
{
   "word": "PoTaTo"
}
```

The intent of the message is for your microservice architecture to look up the definition for the word passed in.

## JSON Output

If the word provided via the input JSON is found in the dictionary, the definition is returned:

```
{
   "word": "PoTaTo",
   "definition": "a starchy plant tuber which is one of the most
important food crops, cooked and eaten as a vegetable."
}
```

If the word is provided, but not found in the dictionary, this message is returned:

```
{
   "word": "pota",
   "error": "Word not found in dictionary."
}
```

If the word is not provided, an error message is returned:

```
{
   "word": 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 5000** for JSON input sent via an **HTTP POST** to "**/definition**", e.g. "http://localhost:5000/definition"
2. Validate the input JSON to ensure a word was provided, if the "word" parameter is nil, return the invalid JSON input result.
3. Clean the input JSON to ensure the word passed to container 2 does not have any extra spaces, and is in a consistent format:
    a. Trim whitespace from the start and end of the word
    b. Convert the word to all lowercase

4. Send the "word" parameter 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 look up definitions. It must:

1. Mount the **host machine** directory '.' to a docker volume
2. Load the contents of dictionary.txt in the docker volume
3. Listen on an endpoint/port you define to respond to definition requests:
    a. Lookup the input word in the dictionary
    b. Return the definition in the appropriate JSON format, or, if the word is not found the word not found response (see errorresponses.json for exact response formats).

## Additional Requirements

1. You must push both your containers to a Dockerhub account you create (this is free)
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 5000 and you must mount the local volume '.' (the current directory) to get access to the dictionary.txt file I provide.
    a. **NOTE: The version keyword should be either '3' in your docker-compose.yml file or absent (indicating to use the latest version). You must ensure you use the latest spec for building your docker-compose.yml file.**

## 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 project). I care that you can meet the learning objectives defined at the top of this document, and I can verify this by simply running your containers and verifying responses.

Your submission will be marked by a Python script that I will write, the script will do the following:

1. Copy your docker-compose.yml file into a temp directory
2. Copy my version of dictionary.txt into the temp directory
3. Run "docker-compose up" in the temp directory
4. HTTP POST an **invalid** JSON input to http://localhost:5000/definition and verify that you return the invalid JSON input response in your JSON response
5. HTTP POST a **valid** JSON input (with crazy combinations of capital, lowercase and whitespace at the start and end of words) to http://localhost:5000/definition and verify that you return the correct definition in your JSON response

6. HTTP POST a **valid** JSON input with a word that does not exist in the dictionary and verify that you return the word not found JSON response
7. Run "docker-compose -v --rmi all" to shut things down and remove your images.

Your mark is entirely based on the success of steps 4, 5 and 6:

- **Pass all 3 = 100%**
- **Pass 2 = 80%**
- **Pass 1 = 60%**
- **Any other result = 0%**

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! 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 testing dictionary.txt 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

## How To Submit

Submit your **docker-compose.yml** (and **nothing else**) to the Brightspace submission folder. If it's not named docker-compose.yml it won't work and you will get 0, I have 175 students this semester, I'm not manually renaming all your files! ;)