Smart Cities Hackathon
Tier 3 Challenge 10

Tier 3: Challenge 10

Tier 3 Challenge 10: Python Docker and Scripting Challenge

This challenge will see you set up your environment for Python and create a Python script to retrieve the latest reading for a sensor of your choosing.

If you want to run the Python locally you can jump to Step 4. Otherwise, if you want to build as a Docker image, you can follow the following steps
https://code.visualstudio.com/docs/languages/python

https://code.visualstudio.com/docs/python/python-tutorial

Step 1

Go to the official Docker website: https://www.docker.com/products/docker-desktop
Download Docker Desktop for your operating system (Windows, macOS, or Linux).
Follow the installation instructions provided for your operating system.

Verify Docker installation:
After installing Docker Desktop, open a terminal or command prompt.

Install Docker desktop onto your machine.

Step 2

Create a new folder for your Tier 3 challenges.

Step 3

Create a **Dockerfile** and add the following elements.

```
# Use an official Python runtime as a parent image
FROM python:3.10-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy all Python files from the host's hackathon folder into the container's
/app folder
COPY hackathon/*.py /app/
```

Step 4

Next, create a **requirements file** and add the following contents.

```
matplotlib
requests
```

Step 5

Create a subfolder called hackathon. This is where you will save all of your Python files. Create a file called hello.py and add the following code.

```
# hello.py

def main():
   print("Hello, world!")

if __name__ == "__main__":
   main()
```

Step 6

Create a file called **compose.yml** and add the following contents.

```
services:
 python-app:
   build:
     context: .
     dockerfile: Dockerfile
   ports:
     - "80:80"
   volumes:
     - ./hackathon:/app
```

Step 7

Open a command prompt and run the following command:

```
docker compose run python-app python hello.py
```

This will build the Docker file initially. You should see the following output:

Hello, world!

Step 8

Log into https://core.aql.com/login and create a bearer token if you haven't created one from an earlier challenge. Copy this as you will need it for one of the following steps.

Step 9

Choose a series of sensor IDs from the sensors list - we will need these IDs for the coding element.

Step 10

Create a new Python file called challenge10.py.

Step 11

Enter the code below, then replace the token and IDs with the chosen values from the earlier steps.

```
import requests

# Constants
CORE_URL = 'https:/api.core.aql.com/v1/'
TOKEN = 'YOUR_TOKEN_ID'
SENSOR_IDS = [
   "REPLACE_WITH_SENSOR_ID",
   "REPLACE_WITH_SENSOR_ID"
]
```

Step 12

Let's add the loop to handle each sensor.

```
# Get data from each station
for sensor_id in SENSOR_IDS:
   print("Getting data from", sensor_id)
```

Run the challenge using:

```
docker compose run python-app python challenge10.py
```

You should see the sensor IDs echoed to the console output.

Step 13

Add a definition to query https://api.core.aql.com/doc/#api-Sensor-Sensor_data/latest with the bearer token created earlier.

```
def get_reading(sensor_id):
   url = f"{CORE_URL}sensors/{sensor_id}/sensor-data/latest"
   headers = {'Authorization': f'Bearer {TOKEN}'}
   try:
       response = requests.get(url, headers=headers)
       response.raise_for_status()
       return response.json()
   except requests.exceptions.HTTPError as e:
   print(e)
```

Add the following updates to the initial loop:

```
reading = get_reading(sensor_id);
print(reading)
```

Run the code using:

```
docker compose run python-app python challenge11.py
```

The latest readings should be displayed to the screen.

You have now completed this challenge. You could extend this challenge by calling https://api.core.aql.com/doc/#api-Sensor-Sensor_data/aggregate and returning a range of data and then plotting this to a web output.