Create a Service Spanning Multiple Containers

PostgreSQL Creation:

1. docker pull postgres
2. docker network create doknet
3. docker run --network doknet --name db -e POSTGRES_PASSWORD=mysecret -d postgres

PostgreSQL Init:

1. Create init.sql as follows:

```
 CREATE TABLE IF NOT EXISTS pathcount (
  path TEXT PRIMARY KEY,
  count INT DEFAULT 0
);
```

2. docker run -i --rm --network doknet -e PGPASSWORD=mysecret postgres psql -h 172.18.0.2 -U postgres < init.sql

Create WebApp:

1. Create a .env file called exp.env with all the environment variables required to connect to db as follows:

```
POSTGRES_URL=172.18.0.2
POSTGRES_USER=postgres
POSTGRES_PW=mysecret
POSTGRES_DB=postgres
```

2. Now write up the Dockerfile with the following contents:

```
from ubuntu:latest

RUN ["apt-get", "update"]
RUN apt-get update \
&& apt-get install -y software-properties-common vim \
&& add-apt-repository ppa:jonathonf/python-3.6 \
&& apt-get update -y \
&& apt-get install -y build-essential python3.6 python3.6-dev python3-pip
python3.6-venv \
&& pip3 install --upgrade pip
COPY . /app
WORKDIR /app
RUN pip install --upgrade pip
RUN pip install -r requirements.txt
```

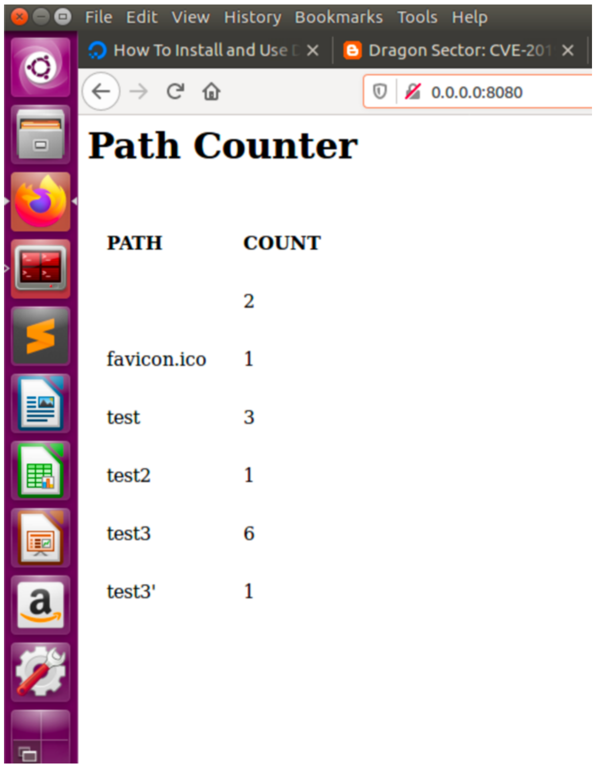
```
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

3. Then our app.py is the flask server with the following code:

```
from flask import Flask, request, jsonify, render_template
import psycopg2,os


#Get Env Variables
def get_env_variable(name):
    try:
        return os.environ[name]
    except KeyError:
        message = "Expected environment variable '{}' not set.".format(name)
        raise Exception(message)

# POSTGRES SQL ENV
POSTGRES_URL = get_env_variable("POSTGRES_URL")
POSTGRES_USER = get_env_variable("POSTGRES_USER")
POSTGRES_PW = get_env_variable("POSTGRES_PW")
POSTGRES_DB = get_env_variable("POSTGRES_DB")


app = Flask(__name__)

#App Routing

@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def index(path):
        try:
                conn = psycopg2.connect(dbname = POSTGRES_DB, user =
POSTGRES_USER,password = POSTGRES_PW, host = POSTGRES_URL, port=5432)
        except:
                print("I am unable to connect to the database.")
        cur = conn.cursor()
        c = cur.execute(sql.SQL("INSERT INTO {} VALUES (%s,1) on
CONFLICT(path) DO UPDATE  SET count = pathcount.count + 1  RETURNING
count;").format(sql.Identifier('pathcount')),[path,])

        try:
                cur.execute("SELECT * from pathcount ORDER BY path;")
        except:
                print("cant print out pathcount")
        rows = cur.fetchall()
        conn.commit()
        cur.close()
        conn.close()
        #for row in rows:
```

```
        #       print(row[0])
        return render_template("index.html",rows = rows)
        #return "ok"

if __name__ == '__main__':
        app.run(host='0.0.0.0',port=8080)
```

4. Run the following command to build the docker image (including the dot):
   docker build -t server .
5. Then run the following command to create the container:
   docker run -d --network doknet --name service -p 8080:8080 --env-file=exp.env server
6. Now if you go to the browser with the following url '127.0.0.1:8080' you should see the web app running with the various paths shown.



Path Counter

PATH	COUNT
	2
favicon.ico	1
test	3
test2	1
test3	6
test3'	1

Security Questions:

1. Why did we create a special network instead of exposing the host network?
    - In order to keep the docker network isolated from the host network. This acts as a defence-in-depth strategy if the container were to get compromised and the attacker is trying to move across the network. The idea is that it is easier to prune a malicious docker network and image rather then replacing the host network and devices.

2. Why didn't we use exposed ports everywhere (that they exist)?

        - Open ports that are not required can be easily accessed my a malicious attacker using tools like nmap, etc. These ports can act as a point of enrty for the attacker to take control of the container.

3. What could happen if you didn't use SQL parameters, but relied on string formatting for setting the path in your queries?
    - The Web Application would be vulnerable to SQL Injection Attack and can lead to data leakage and in some cases credential leakage too.
4. Why is that particularly important in this setup? What makes those parameters potentially dangerous?
    - Since our web app takes strings directly from the url and puts them into a DB, we already know that the webapp is not performing any sanization of input. This can be used by the attacker to an advantage because the attaker can inout anything and see clearly how the database is reacting to the input.
5. The bridge network we define only works on a single host. What would you have to do to make these containers talk to each other if they were running on *different host machines*?
    - If we consider two different hosts with different subnet space, the method to connect the containers in each of those nodes to the same network is. To use comething called 'overlay network'. This method is mainly used in swarms for different nodes to have containers talking to each other.
    - The command used to create this Overlay Network is : docker network create -d overlay <network name>
6. What parts of this did you wish were simpler? Which parts seemed unnecessarily difficult?
    - Recreating containers from images while debugging was a little redundant.

References:

[1] Github Code for Web app : https://github.com/nithinrajs/psql-docker