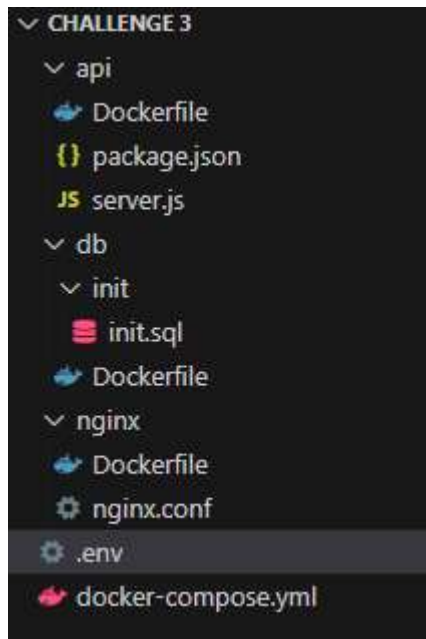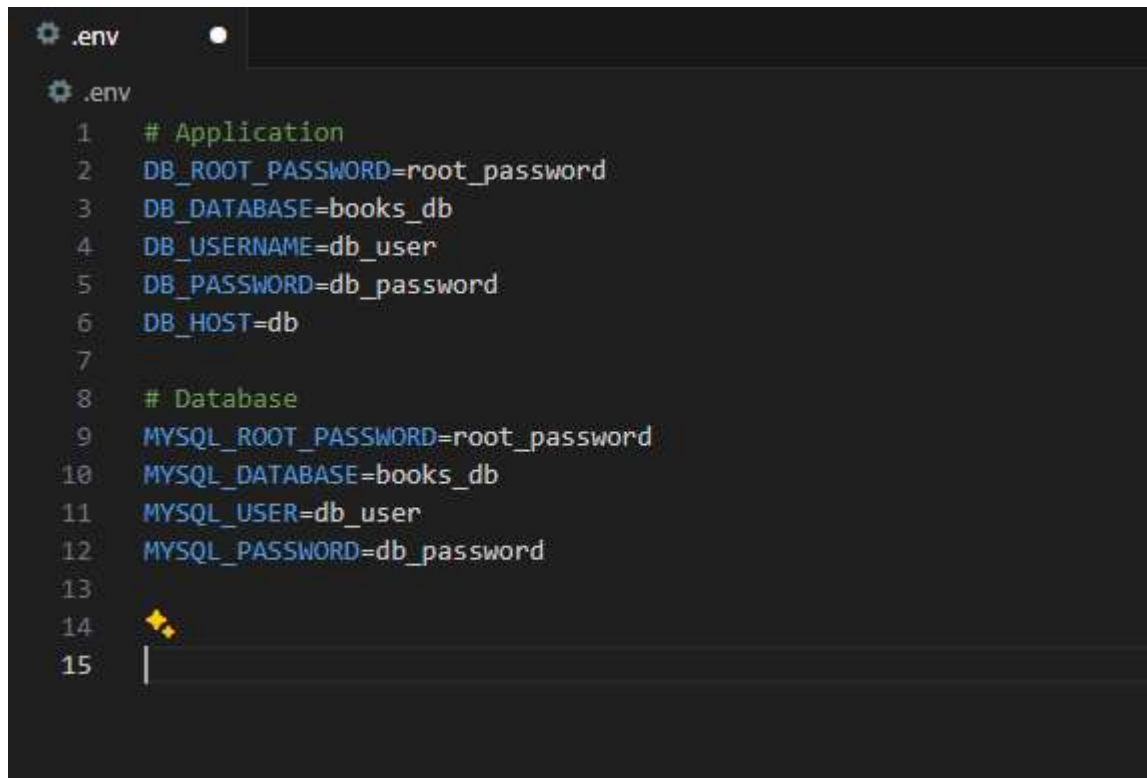# Docker Challenge 3
Abdel Mouzahir

For Docker Challenge 3, we are tasked with connecting a NGinx app to a simple database, and displaying the information. To start, we were given the NGinx, the database, and the api files and their respective dockerfiles.

This is the overall file layout:

Note that the .env, and the docker-compose file were not included in the package.
Once the package was extracted, we were responsible for setting the .env variables and writing the docker-compose file. The .env file should look something like this:

```
.env        ●

.env
  1   # Application
  2   DB_ROOT_PASSWORD=root_password
  3   DB_DATABASE=books_db
  4   DB_USERNAME=db_user
  5   DB_PASSWORD=db_password
  6   DB_HOST=db
  7
  8   # Database
  9   MYSQL_ROOT_PASSWORD=root_password
 10   MYSQL_DATABASE=books_db
 11   MYSQL_USER=db_user
 12   MYSQL_PASSWORD=db_password
 13
 14   ✦
 15   |
```

After we have made our .env file and saved all of our variables, we can continue on to write out docker-compose file. It should look something like this
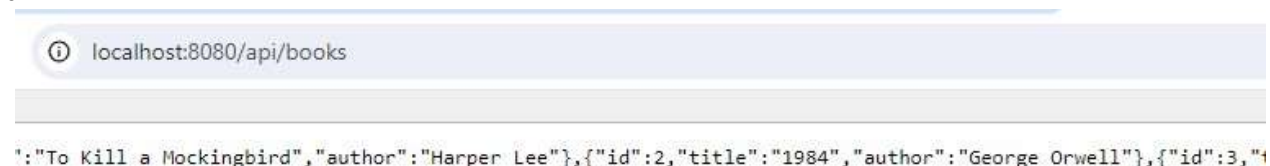
🐳 docker-compose.yml

```yaml
1
2    services:
3      db:
4        build:
5          context: .
6          dockerfile: db/Dockerfile
7        environment:
8          MYSQL_ROOT_PASSWORD: root_password
9          MYSQL_DATABASE: books_db
10         MYSQL_USER: db_user
11         MYSQL_PASSWORD: db_password
12       volumes:
13         - db-data:/var/lib/mysql
14
15     node-service:
16       build:
17         context: .
18         dockerfile: ./api/Dockerfile
19       environment:
20         DB_HOST: db
21         DB_USERNAME: db_user
22         DB_PASSWORD: db_password
23         DB_DATABASE: books_db
24       depends_on:
25         - db
26       ports:
27         - "3000:3000"
28
29     nginx:
30       build:
31         context: .
32         dockerfile: nginx/Dockerfile
33       depends_on:
34         - node-service
35       ports:
36         - "8080:80"
37
38   volumes:
39     db-data:
40
```

This file describes what services we are building, and where to find them. For example for the database (db), the build section describes where to find the dockerfile to build the database from, and in this case it is the db/Dockerfile in the directory. The environment variables are used to build the database (not to be confused with the .env). The volumes mounts a volume to the /var/lib/mysql MariaDB directory. This is how the information is stored. The other two services are largely the same, but the only difference being that node-service and nginx have ports. This first number is the port of the host, and the second is the port of the container.

Now that we have our files ready, we can simply build and run our container. The command for building and running the docker-compose file, in the directory of the docker-compose file we type the command 'docker-compose up --build', which first builds the container, then runs it. It should take a few minutes for the container to build and then to run.

Once the container is running, we can go to port 8080 of the localhost in a browser. If everything is correct, and the build was successful, you should be able to see the items in the database in json format.


```
':"To Kill a Mockingbird","author":"Harper Lee"},{"id":2,"title":"1984","author":"George Orwell"},{"id":3,"t
```

Additionally, you can test the containers by navigating to the correct folder in the command line, and typing in docker-compose ps, which should give you something like this:

Here we can visibility see all 3 services are correctly running.

As for trouble shooting, it is imperative that the file layout in the directory is correct, and that the dockerfile addresses are correct. The initial package had docker/ in front of the COPY command, which I deleted, and then the relative addresses worked fine. Here are the corrected dockerfile addresses:

```
.env           Dockerfile  X

api >  Dockerfile > ...
  1      # Use the official Node.js image as base
  2      FROM node:alpine
  3
  4      # Set the working directory in the container
  5      WORKDIR /app
  6
  7      # Copy package.json and package-lock.json
  8      COPY api/package*.json ./
  9
 10      # # Install dependencies
 11      RUN npm install
 12
 13      # Copy the rest of the application
 14      COPY api/* .
 15
 16      # Expose port 3000 to the outside world
 17      EXPOSE 3000
 18
 19      # Command to run the application
 20      CMD ["node", "server.js"]
 21      CMD ["sleep", "infinity"]
 22
```

```
db >  Dockerfile > ...
  1      FROM mariadb
  2
  3      COPY db/init/init.sql /docker-entrypoint-initdb.d
  4
```

```
nginx >  Dockerfile > ...
  1      FROM nginx
  2      RUN rm /etc/nginx/conf.d/default.conf
  3      COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
```

Another trouble shooting aspect was ensuring that the .env file and the docker-compose file had the same environment variables. Any variance and the container will not work, as the variables must be the same. If you are still struggling, here are some commands for debugging:

Docker-compose logs db
Docker-compose logs node-service
Docker-compose logs nginx

These will show you the logs of the of the build process, and you can see any problems as they arise.
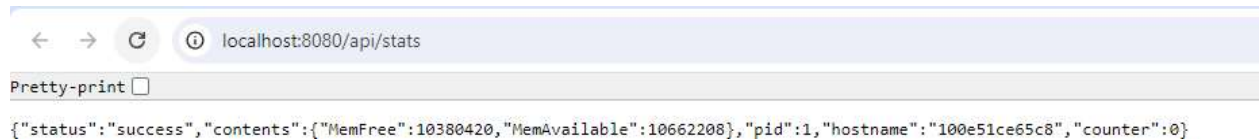
Docker Challenge 4

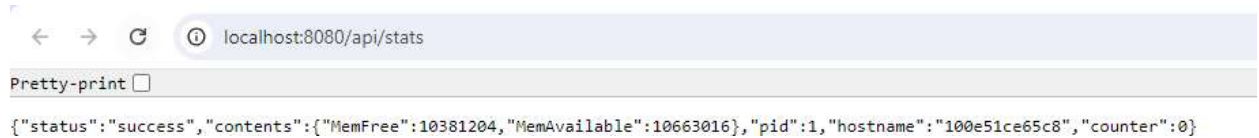The docker challenge 4 require us to scale the application we built in challenge 3 up. Going to the url http://localhost:8080/api/stats with the app running, we get this message:

localhost:8080/api/stats

Pretty-print ☐

{"status":"success","contents":{"MemFree":10388332,"MemAvailable":10668580},"pid":1,"hostname":"100e51ce65c8","counter":0}

If we do this a couple more times we get:

localhost:8080/api/stats

Pretty-print ☐

{"status":"success","contents":{"MemFree":10380420,"MemAvailable":10662208},"pid":1,"hostname":"100e51ce65c8","counter":0}

localhost:8080/api/stats

Pretty-print ☐

{"status":"success","contents":{"MemFree":10381204,"MemAvailable":10663016},"pid":1,"hostname":"100e51ce65c8","counter":0}

As we can see, the host name remains the same. We know this is actually a new request by the change in memory, and not just a frozen browser. The hostname we get for this is "100e51ce65c8"

After changing our docker-compose file slightly to let docker handle the ports instead of explicitly declaring them, like this:

```
node-service:
  build:
    context: .
    dockerfile: ./api/Dockerfile
  environment:
    DB_HOST: db
    DB_USERNAME: db_user
    DB_PASSWORD: db_password
    DB_DATABASE: books_db
  depends_on:
    - db
```

We then could use the command 'docker-compose up - -scale node-service=3 -d' to scale up the application. When we look at the hostname in the localhost, we see that is has changed to "d50ab7dff472"
When we run the command docker-compose ps in the command line we can see the three different node-services running, as shown here:

```
C:\Users\mz-ab\OneDrive\Desktop\sem4\operating sys\mid-term-docker-challenge-Abdel-Mouzahir\challenge3> docker-compose ps
NAME                      IMAGE                     COMMAND                 SERVICE        CREATED             STATUS                PORTS
challenge3-db-1           challenge3-db             "docker-entrypoint.s…"  db             4 hours ago         Up About a minute     3306/tcp
challenge3-nginx-1        challenge3-nginx          "/docker-entrypoint.…"  nginx          About a minute ago  Up About a minute     0.0.0.0:8080->80/tcp
challenge3-node-service-1 challenge3-node-service   "docker-entrypoint.s…"  node-service   2 minutes ago       Up About a minute     3000/tcp
challenge3-node-service-2 challenge3-node-service   "docker-entrypoint.s…"  node-service   50 seconds ago      Up 46 seconds         3000/tcp
challenge3-node-service-3 challenge3-node-service   "docker-entrypoint.s…"  node-service   51 seconds ago      Up 44 seconds         3000/tcp

C:\Users\mz-ab\OneDrive\Desktop\sem4\operating sys\mid-term-docker-challenge-Abdel-Mouzahir\challenge3> |
```

This should be all you need to scale your application as needed.

References:

https://github.com/infiniflow/ragflow/issues/329

https://hub.docker.com/_/mariadb

https://nginx.org/en/docs/beginners_guide.html