Exercise 7

Dockerising our Purchase microservice

Prior Knowledge
Unix command-line
Previous exercises

Learning Objectives
Creating a docker image for our Typescript service
Linking docker images and Docker Compose

Software Requirements

- Docker
- Docker-compose

Overview
We are already using docker to run the Postgres database we need. This lab takes this further creating a simple docker microservice and deploying it with docker-compose.

Steps

1. I suggest you do this in the existing purchase-starter directory.

   However, if you want to take my answer to the previous exercises, you can clone it from Github here:
   ```
   cd ~
   git clone https://github.com/pzfreo/purchase-complete.git
   cd purchase-complete
   yarn install
   ```

2. Notice that there is a Dockerfile in there that we did not previously use. Take a look at this Dockerfile:
   ```
   FROM node:15.12.0-buster-slim

   RUN mkdir -p /usr/src/app
   WORKDIR /usr/src/app
   COPY . .

   RUN yarn install
   RUN yarn global add typescript ts-node tsoa

   EXPOSE 8000
   CMD ["yarn", "run", "start"]
   ```

   buster-slim is the cut down version of a Debian distro. 15.12 is the latest node release at the time of writing.

3. Also look at .dockerignore

```
# comment
node_modules*
build*
.git*
```

4. This tells docker to not add the node_modules or built components into the container - instead we will run yarn install and the build inside the container process.

5. Now build the docker image (with the right place for yruserid).
   `docker build -t <yruserid>/purchase-ts:0.0.1 .`

   Don't forget the .

6. Make sure the postgres container is still running.

7. We can now start the ts container (all on one line). Don't forget to use your docker userid where I've put <yruserid>.

   ```
   docker run --name purchase -p 8000:8000 --link postgres:postgres
   -e DBHOST="postgres" -ti <yruserid>/purchase-ts:0.0.1
   ```

   Explanation:

   ```
   --name purchase
   # give the docker container instance a name

   -p 8000:8000
   # expose 8000 from the container as 8000 locally

   --link postgres:postgres
   # make the postgres container available
   # as DNS postgres

   -e DBHOST="postgres"
   # set the env variable DBHOST inside the container
   # so that the ts code can pick this up

   -ti
   # run in interactive mode (enables Ctrl-C!)
   ```

8. Try out the service by browsing http://localhost:8000/purchase

9. If you like you could rerun the tests against the service.

10. Now kill the service and the postgres database (Ctrl-C on both)

11. This leaves the container instances ready to be re-instantiated, which we don't want.

    First list them:

    ```
    docker ps -a
    ```

    You should see:

    ```
    CONTAINER ID   IMAGE                          COMMAND
    CREATED          STATUS                             PORTS
    NAMES
    2879a75e8e58   pzfreo/purchase-ts:0.0.1
    "docker-entrypoint.s…"   29 seconds ago   Exited (1)
    17 seconds ago                purchase
    b3ba01bba5d4   postgres
    "docker-entrypoint.s…"   40 minutes ago   Exited (0)
    4 seconds ago                 postgres
    ```

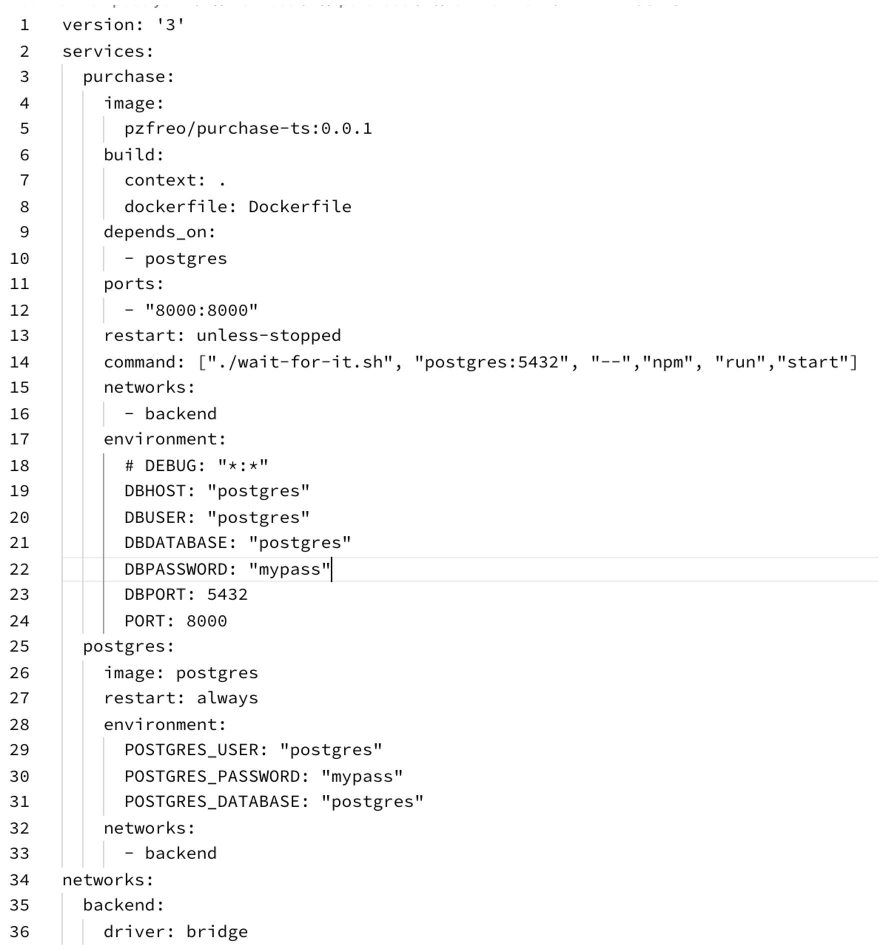
12. Let's remove them.

    ```
    docker rm purchase postgres

    purchase
    postgres
    ```

13. Now we can re-build this in a better way with Docker Compose.
    https://docs.docker.com/compose/

    Let's have a look at the file **docker-compose.yaml**



```
 1  version: '3'
 2  services:
 3    purchase:
 4      image:
 5        pzfreo/purchase-ts:0.0.1
 6      build:
 7        context: .
 8        dockerfile: Dockerfile
 9      depends_on:
10        - postgres
11      ports:
12        - "8000:8000"
13      restart: unless-stopped
14      command: ["./wait-for-it.sh", "postgres:5432", "--","npm", "run","start"]
15      networks:
16        - backend
17      environment:
18        # DEBUG: "*:*"
19        DBHOST: "postgres"
20        DBUSER: "postgres"
21        DBDATABASE: "postgres"
22        DBPASSWORD: "mypass"
23        DBPORT: 5432
24        PORT: 8000
25    postgres:
26      image: postgres
27      restart: always
28      environment:
29        POSTGRES_USER: "postgres"
30        POSTGRES_PASSWORD: "mypass"
31        POSTGRES_DATABASE: "postgres"
32      networks:
33        - backend
34  networks:
35    backend:
36      driver: bridge
```

Some notes on this.

Line 1: which version of docker-compose was this written for

Lines 3-24: The definition of our "purchase" container and service.

Most of this is obvious. However we are "overwriting" the start command to use a little script that waits until the db is active.
    https://github.com/vishnubob/wait-for-it
Because of the async way node works, the service would start up without this, but there is a danger that the web interface would be up before the database is.
Lines 10-11 ought to take care of this by defining a dependency but the database takes a while to get started.

Lines 24-33: setup of the Postgres service

The only other thing that's worth noting is the new network we are creating "backend". Lines 34-36

This is cool: basically the node service and db connect via this backend network, so the database is not accessible from the rest of the world.

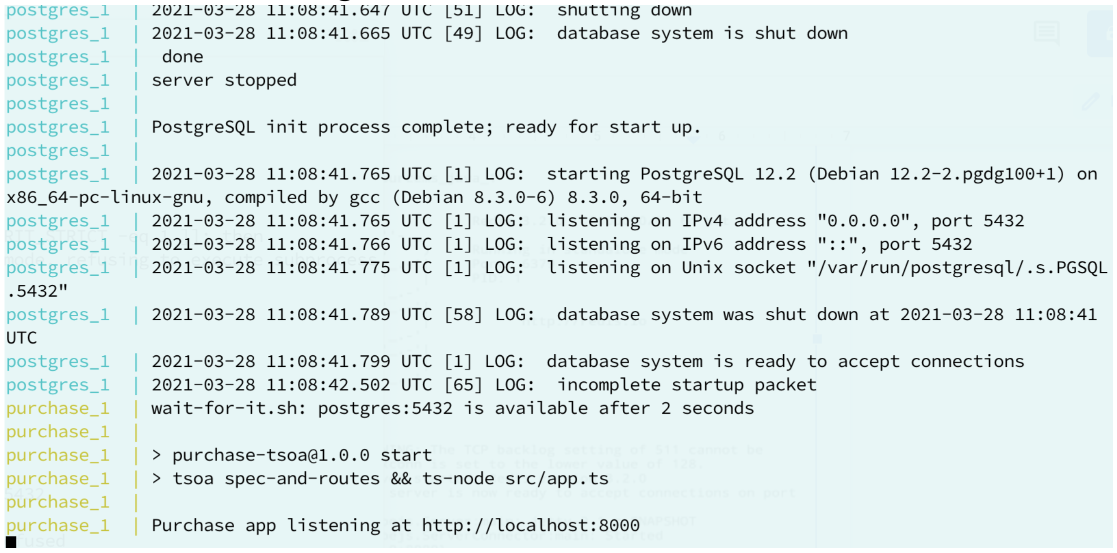
14. Docker compose also deals with a lot of other tricky issues, especially with inserting secrets (keys, passwords, etc) into your docker containers without pushing them into the docker hub.

15. Change the docker userid on line 5 to match yours.

16. In the terminal window type:

```
docker-compose up --build
```

17. You should see something like:



```
postgres_1  | 2021-03-28 11:08:41.647 UTC [51] LOG:  shutting down
postgres_1  | 2021-03-28 11:08:41.665 UTC [49] LOG:  database system is shut down
postgres_1  |  done
postgres_1  | server stopped
postgres_1  |
postgres_1  | PostgreSQL init process complete; ready for start up.
postgres_1  |
postgres_1  | 2021-03-28 11:08:41.765 UTC [1] LOG:  starting PostgreSQL 12.2 (Debian 12.2-2.pgdg100+1) on
x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
postgres_1  | 2021-03-28 11:08:41.765 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres_1  | 2021-03-28 11:08:41.766 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgres_1  | 2021-03-28 11:08:41.775 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL
.5432"
postgres_1  | 2021-03-28 11:08:41.789 UTC [58] LOG:  database system was shut down at 2021-03-28 11:08:41
UTC
postgres_1  | 2021-03-28 11:08:41.799 UTC [1] LOG:  database system is ready to accept connections
postgres_1  | 2021-03-28 11:08:42.502 UTC [65] LOG:  incomplete startup packet
purchase_1  | wait-for-it.sh: postgres:5432 is available after 2 seconds
purchase_1  |
purchase_1  | > purchase-tsoa@1.0.0 start
purchase_1  | > tsoa spec-and-routes && ts-node src/app.ts
purchase_1  |
purchase_1  | Purchase app listening at http://localhost:8000
```

18. Test out the service on http://localhost:8000/purchase

19. This is great for development - we can simply rebuild the environment completely whenever there are changes.

20. You can stop the system by hitting Ctrl-C, and then typing:
`docker-compose down`

21. We can build and push the images to the docker hub:

```
docker-compose build
docker-compose push
```

Note: you will need to be logged in to docker from the earlier exercise.

22. To prove this is using images built in the repository, we are going to clear out our local cache of images:

    ```
    #first kill all docker instances
    docker rm --force $(docker ps -aq)

    #now remove all images
    docker rmi --force $(docker images -aq)
    ```

23. Now you can pull the containers from the hub:

    ```
    docker-compose pull
    ```

24. Now try starting without building:

    ```
    docker-compose up --no-build
    ```

    This will be super quick now that all the build is done and the images pulled.

25. Congratulations, this lab is complete.

Extension:

Can you think of a way to speed up the startup even more? Try to update the Dockerfile to implement this strategy.