**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

1. Notice that there is a Dockerfile in there that we did not previously use.  
   Take a look at this Dockerfile:  
     
   buster-slim is the cut down version of a Debian distro. 15.12 is the latest node release at the time of writing.
2. Also look at .dockerignore  
     
   
3. 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.
4. Now build the docker image (with the right place for yruserid).  
   docker build -t *<yruserid>*/purchase-ts:0.0.1 .

*Don’t forget the .*

1. Make sure the postgres container is still running.
2. 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!)

1. Try out the service by browsing [http://localhost:8000/purchase](http://localhost/purchase)
2. If you like you could rerun the tests against the service.
3. Now kill the service and the postgres database (Ctrl-C on both)
4. 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

1. Let’s remove them.   
     
   docker rm purchase postgres

purchase

postgres

1. 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**

****

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.

1. 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.
2. Change the docker userid on line 5 to match yours.
3. In the terminal window type:  
     
   docker-compose up --build
4. You should see something like:  
   
5. Test out the service on [http://localhost:8000/purchase](http://localhost/purchase)
6. This is great for development - we can simply rebuild the environment completely whenever there are changes.
7. You can stop the system by hitting Ctrl-C, and then typing:  
   docker-compose down
8. 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.
9. 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)

1. Now you can pull the containers from the hub:  
     
   docker-compose pull
2. 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.

1. 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.