Exercise 9

Dockerising our POResourceMS microservice

Prior Knowledge
Unix command-line
Previous exercises

Learning Objectives
Creating a docker image for Java
Linking docker images

Software Requirements
- Docker
- Docker-compose
- Eclipse
- Nano text editor

Overview
We are going to take the shadow JAR executable from Exercise 7 and get it to run in a Docker container, talking to Redis running in another docker container.

Steps:

1. Go back to your Exercise 7 directory
   cd ~/ex7/POResourceMS

2. Notice that there is a Dockerfile in there that we did not previously use. Take a look at this Dockerfile:

```
FROM alpine:3.3
MAINTAINER Paul Fremantle (paul@fremantle.org)

RUN apk --update add openjdk8-jre

RUN mkdir -p /home/root/po
ADD build/libs/POResourceMS-all.jar /home/root/po
EXPOSE 8080
ENV REDIS_HOST "redis"
ENTRYPOINT java -jar /home/root/po/POResourceMS-all.jar
```

   It is all pretty self-explanatory. The only new thing is that we are setting an environment variable in the docker container of REDIS_HOST.

3. Now we need to change where the POResource code is going to find its redis server. Look at this line in OrderRedis.Java

```
public static JedisPool pool = new JedisPool(new JedisPoolConfig(),
        System.getenv().containsKey("REDIS_HOST") ?
        System.getenv("REDIS_HOST") : "localhost");
```

   This will see if there is an environment variable REDIS_HOST and if so try to connect to that. If not, it will default to localhost. Hence we need to make sure the REDIS_HOST env is set in our Docker config or runtime.

   Now look at the Dockerfile again. You can see that the REDIS_HOST is set so that it will look for a server called *redis.*

4. Build and test this to check it is still working.
   `gradle clean build`

5. Now build the shadowJar:
   `gradle clean shadowJar`

6. Now build the docker image (with the right place for yruserid).
   `sudo docker build -t <yruserid>/poresourcems .`

7. Stop the local redis server on Ubuntu:
   `sudo service redis-server stop`

8. Validate it is really stopped by trying redis-cli and hoping it fails!

9. Start the redis docker image with the following command:
   `sudo docker run -d --name redis -p 6379:6379 redis:3.2.0-alpine`

   This gives the container a name of redis, which we can use to link to it from the Java container.

10. We can now start the Java container:
    `sudo docker run -d -p 80:8080 --link redis:redis <yrid>/poresourcems`

    The important new setting is the --link redis:redis, which means that there will be a DNS host entry in this container for redis, which will point to the redis container. (see point #3)

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

12. Have a look at the file **docker-compose.yml**

    This is a config file for Docker Compose (see https://docs.docker.com/compose/). Docker compose is a really useful tool for managing combinations of Docker instances. It can also work with another tool called Docker Swarm to run clusters. There are other tools that do this sort of thing too (e.g. Kubernetes) but that is beyond the scope of this Module!

    Here is an annotated version of the file:

```
# supports version 2 of the docker compose language.
# Requires docker compose 1.6 or later
version: '2'

#define the services (e.g. containers) we need
services:
  # the Java container is our POResourceMS app, whose
  # dockerfile is in the current directory
  java:
    # build the Dockerfile
    build: .
    # expose port 8080 as port 80
    ports:
     - "80:8080"
    # we require access to the redis service
    links:
     - "redis:redis"
    # depends on redis
    depends_on:
     - redis
  # the redis service is just redis
  redis:
    image: redis
```

13. 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.

14. Before we start this, we would like to completely clean out docker, so you can see this relies on nothing we have previously built or pulled. Run these three commands.

```
sudo docker kill $(sudo docker ps -q)
sudo docker rm $(sudo docker ps -a -q)
sudo docker rmi $(sudo docker images -q)
```

15. In the terminal window type:

```
sudo docker-compose up
```


16. You should see something like:

```
Creating network "poresourcems_default" with the default driver
Creating poresourcems_redis_1
Creating poresourcems_java_1
Attaching to poresourcems_redis_1, poresourcems_java_1
redis_1  |                 _._
redis_1  |            _.-``__ ''-._
redis_1  |       _.-``    `.  `_.  ''-._           Redis 3.2.0 (00000000/0) 64 bit
redis_1  |   .-`` .-```.  ```\/    _.,_ ''-._
redis_1  |  (    '      ,       .-`  | `,    )     Running in standalone mode
redis_1  |  |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
redis_1  |  |    `-._   `._    /     _.-'    |     PID: 1
redis_1  |   `-._    `-._  `-./  _.-'    _.-'
redis_1  |  |`-._`-._    `-.__.-'    _.-'_.-'|
redis_1  |  |    `-._`-._        _.-'_.-'    |           http://redis.io
redis_1  |   `-._    `-._`-.__.-'_.-'    _.-'
redis_1  |  |`-._`-._    `-.__.-'    _.-'_.-'|
redis_1  |  |    `-._`-._        _.-'_.-'    |
redis_1  |   `-._    `-._`-.__.-'_.-'    _.-'
redis_1  |       `-._    `-.__.-'    _.-'
redis_1  |           `-._        _.-'
redis_1  |               `-.__.-'
redis_1  |
redis_1  | 1:M 31 May 14:20:44.979 # WARNING: The TCP backlog setting of 511 cannot be
enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis_1  | 1:M 31 May 14:20:44.979 # Server started, Redis version 3.2.0
redis_1  | 1:M 31 May 14:20:44.980 * The server is now ready to accept connections on port
6379
java_1   | 2016-05-31 14:20:47.057:INFO:oejs.Server:main: jetty-9.1.z-SNAPSHOT
java_1   | 2016-05-31 14:20:47.143:INFO:oejs.ServerConnector:main: Started
ServerConnector@62230c58{HTTP/1.1}{0.0.0.0:8080}
```

17. Test out the service on http://localhost/purchase

18. You can stop the system by hitting Ctrl-C.

19. Congratulations, this lab is complete.