**Exercise 8**

*Dockerising our Purchase 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 JAR executable from the previous exercises and get it to run in a Docker container, talking to Redis running in another docker container.

Steps:

1. Clone PSBComplete (as in exercise 6)  
     
   mkdir ~/ex8  
   cd ~/ex8  
   git clone <https://github.com/pzfreo/PSBComplete.git>

cd PSBComplete

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

FROM openjdk:8-jre-alpine

MAINTAINER Paul Fremantle (paul@fremantle.org)

RUN mkdir -p /home/root/purchase

ADD build/libs/purchase-0.0.2.jar /home/root/purchase

EXPOSE 8080

ENV REDIS\_HOST "redis"

ENTRYPOINT java -jar /home/root/purchase/purchase-0.0.2.jar

1. 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.*

1. Build and test this to check it is still working.  
   gradle clean build
2. Now build the docker image (with the right place for yruserid).  
   docker build -t *<yruserid>*/purchase-service .

*Don’t forget the .*

1. Stop the local redis server on Ubuntu:  
   sudo service redis-server stop
2. Validate it is really stopped by trying redis-cli and hoping it fails!
3. Start the redis docker image with the following command:  
   docker run -d --name redis -p 6379:6379 redis:3.2.8-alpine  
     
   This gives the container a name of redis, which we can use to link to it from the Java container.
4. We can now start the Java container:  
   docker run -d -p 80:8080 --link redis:redis <yrid>/purchase-service  
     
   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)
5. Try out the service by browsing <http://localhost/purchase>

1. 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:3.2.8-alpine

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. 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 this command.

docker rm --force $(docker ps -aq)

1. In the terminal window type:  
   docker-compose up
2. 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}

1. Test out the service on <http://localhost/purchase>
2. You can stop the system by hitting Ctrl-C.
3. Congratulations, this lab is complete.

**Extension**

There is a docker and docker-compose plugin for gradle. See if you can get gradle to build the docker image and push it for you.