Lab 4
REST web service in a Docker container

This lab will show you how you can run your Spring Boot REST service in a Docker container.

On top of that, we will verify how secure the container is, using an open source tool named Claire.

The starting point for this lab is to have the provided VirtualBox machine up-and-running:

- You are logged in under user/password: developer/welcome01
- You have updated the labs running the `git pull` command in the lab workspace directory `/home/developer/projects/SIGSpringBoot101`

Note that in this lab, we will NOT use Eclipse STS.

1. Building the Docker container

We will use the service that we developed in lab 3. You can find the completed code in:

`/home/developer/projects/SIGSpringBoot101/lab 4/dronebuzzers`

The following steps will be taken:

- Step 1: change the Maven pom.xml
  The Maven pom.xml has to be changed so it can create a Docker container
- Step 2: add the Dockerfile
  The Dockerfile contains the Docker container definition
- Step 3: build the container
  Use Maven to create a Docker container

Step 1: change the Maven pom.xml

The Maven file pom.xml will have to undergo two changes:

1. Add a property 'docker.image.prefix', that specifies the Docker image prefix name
2. Add the Spotify plugin for building the Docker container

@1:add the property line in the <properties> tag:

```
<docker.image.prefix>docker</docker.image.prefix>
```

@2: add the plugin in the <plugins> tag:

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.3.4</version>
    <configuration>
        <repository>${docker.image.prefix}/${project.artifactId}</repository>
    </configuration>
</plugin>
```

The resulting pom.xml file can also be found in:

`/home/developer/projects/SIGSpringBoot101/lab 4/input/pom.xml`

Examining the pom.xml file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://mav
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dronebuzzers.parts</groupId>
    <artifactId>dronebuzzers-rest-service</artifactId>
    <packaging>jar</packaging>
    <name>dronebuzzers-rest-service</name>
    <version>1.0.0</version>
    <properties>
        <java.version>1.8</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <springfox-version>2.7.0</springfox-version>
        <docker.image.prefix>docker</docker.image.prefix>
    </properties>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.4.RELEASE</version>
    </parent>
    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>dockerfile-maven-plugin</artifactId>
                <version>1.3.4</version>
                <configuration>
                    <repository>${docker.image.prefix}/${project.artifactId}</repository>
                </configuration>
            </plugin>

        </plugins>
    </build>
    <dependencies>
        <dependency>
```

image prefix is 'docker'

Defines the name for the Docker image, here: docker/dronebuzzers-rest-service

plugin for building Docker container images

Step 2: add the Dockerfile

In order to build a Docker container, we need a file named Dockerfile that specifies the container.

Copy the Dockerfile from the *input* to the project directory *lab 4/dronebuzzers*:

In a terminal window, navigate to the directory `lab 4/dronebuzzers`. Then perform a copy action

```
cp ../input/Dockerfile .
```

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 4/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$ pwd
/home/developer/projects/SIGSpringBoot101/lab 4/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$ cp ../input/Dockerfile .
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$
```

Have a look at the Dockerfile:

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 4/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$ more Dockerfile
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD target/dronebuzzers-rest-service-1.0.0.jar app.jar
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$
```

The keywords in the Dockerfile have the following meaning:

Keyword	Meaning
FROM	sets the base image for the container. This means that our container will be built 'on top of' the FROM container
VOLUME	external mounted volume. This makes the indicated volume write its data on the host machine
ADD	adds files/directories from the source to the container image
ENV	environment variable
ENTRYPOINT	command that the container will run when started

So for our container:

Keyword	Value	Meaning
FROM	openjdk:8-jdk-alpine	base image for the container is openjdk:8-jdk-alpine

VOLUME	/tmp	The container's /tmp volume will write its data to the host. Unless explicitly defined otherwise during start-up of the container, the volume will end up in `/var/lib/docker/volumes`
ADD	target/dronebuzzers-rest-service-1.0.0.jar app.jar	adds the Spring Boot application to the container image under the name app.jar
ENV	JAVA_OPTS=""	Sets the environment variable JAVA_OPTS to "" in the container
ENTRYPOINT	["sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar"]	Upon start, the container will run a shell that executes our application (app.jar)

Step 3: build the container

The Docker container image is built with the command `mvn install dockerfile:build`

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 4/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$ mvn install dockerfile:build
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building dronebuzzers-rest-service 1.0.0
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ dronebuzzers-rest-service ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ dronebuzzers-rest-service ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ dronebuzzers-rest-service ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/developer/projects/SIGSpringBoot101/lab 4/dronebuzzers/src/t
```

Near the end of the execution of the command, the Docker file execution is clearly visible:

```
[INFO] Image will be built as docker/dronebuzzers-rest-service:latest
[INFO]
[INFO] Step 1/5 : FROM openjdk:8-jdk-alpine  1
[INFO]
[INFO] Pulling from library/openjdk
[INFO] Image ff3a5c916c92: Already exists
[INFO] Image 5de5f69f42d7: Pulling fs layer
[INFO] Image fd869c8b9b59: Pulling fs layer
[INFO] Image 5de5f69f42d7: Downloading
[INFO] Image 5de5f69f42d7: Verifying Checksum
[INFO] Image 5de5f69f42d7: Download complete
[INFO] Image 5de5f69f42d7: Extracting
[INFO] Image 5de5f69f42d7: Pull complete
[INFO] Image fd869c8b9b59: Downloading
[INFO] Image fd869c8b9b59: Verifying Checksum
[INFO] Image fd869c8b9b59: Download complete
[INFO] Image fd869c8b9b59: Extracting
[INFO] Image fd869c8b9b59: Pull complete
[INFO] Digest: sha256:4cd17a64b67df1a929a9c6dedf513afcdc48f3ca0b7fddee6489d0246a14390b
[INFO] Status: Downloaded newer image for openjdk:8-jdk-alpine
[INFO]  ---> 224765a6bdbe
[INFO] Step 2/5 : VOLUME /tmp  2
[INFO]
[INFO]  ---> Running in 374f6cd07b42
[INFO] Removing intermediate container 374f6cd07b42
[INFO]  ---> e7a2416dcca7
[INFO] Step 3/5 : ADD target/dronebuzzers-rest-service-1.0.0.jar app.jar  3
[INFO]
[INFO]  ---> 728349a67c9e
[INFO] Step 4/5 : ENV JAVA_OPTS=""  4
[INFO]
[INFO]  ---> Running in 432dfbf76515
[INFO] Removing intermediate container 432dfbf76515
[INFO]  ---> 5797e91fef31
[INFO] Step 5/5 : ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]  5
[INFO]
[INFO]  ---> Running in b942d5305850
[INFO] Removing intermediate container b942d5305850
[INFO]  ---> d4c25bee0522
[INFO] Successfully built d4c25bee0522
[INFO] Successfully tagged docker/dronebuzzers-rest-service:latest
[INFO]
[INFO] Detected build of image with id d4c25bee0522
[INFO] Building jar: /home/developer/projects/SIGSpringBoot101/lab 4/dronebuzzers/target/dronebuzzers-rest-service-1.0.0-docker-info.jar
[INFO] Successfully built docker/dronebuzzers-rest-service:latest
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:06 min
[INFO] Finished at: 2018-03-31T14:56:26Z
[INFO] Final Memory: 52M/254M
[INFO] ------------------------------------------------------------------------
```

The openjdk:8-jdk-alpine container image consists of sever layers that are downloaded

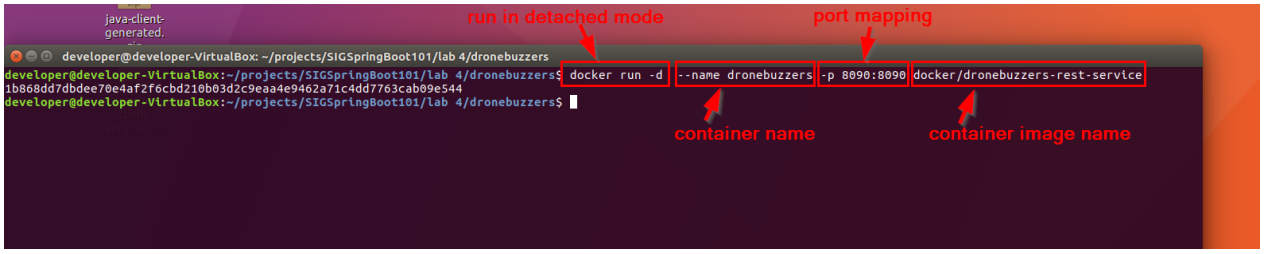
Name as configured in pom.xml, including a 'latest'tag

Check that the Docker container image has been created with the command `docker images`:

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 4/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
docker/dronebuzzers-rest-service latest             1cb9ab207e74        About a minute ago  123MB
rabbitmq                        management          2f415b0e9a6e        3 weeks ago         151MB
openjdk                         8-jdk-alpine        224765a6bdbe        2 months ago        102MB
hello-world                     latest              f2a91732366c        3 months ago        1.85kB
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 4/dronebuzzers$
```

2. Running the Docker container

Now, the container that we've built can be started with the `docker run` command:



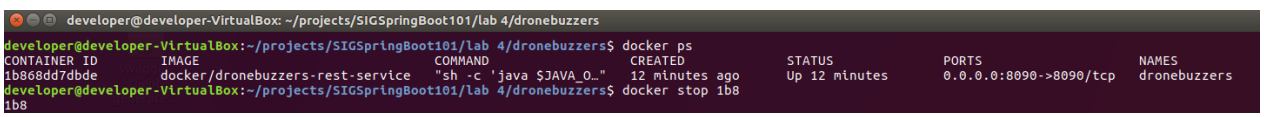
The command to be used:

```
docker run -d  --name dronebuzzers -p 8090:8090 docker/dronebuzzers-rest-service
```

- -d: the container runs in de-tached mode, i.e. in the background
- -p 8090:8090: specifies port exposure, i.e. how a local host port (first 8090)is mapped to the internal container port (second 8090)

Note: if you still have a Spring Boot application running from Eclipse, listening to port 8090, then you will not be able to run the container; in that case, either stop the application in Eclipse to change the first 8090 in the command above in some other port number.

Should you later on want to stop the container: look up the Container ID with the *docker ps* command. Then stop the container with the *docker stop* command. Notice that you only have to enter the first couple of characters of the Container ID.



There is a good Docker cheat sheet: https://github.com/wsargent/docker-cheat-sheet

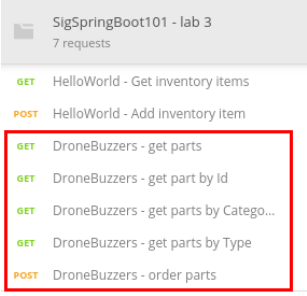
3. Testing the Docker container

Once the container is up and running, we can test it. We will do that with the same Postman tests from lab 3.

For testing, start Postman and import the Collection of Postman tests for lab 3 from location

```
/home/developer/projects/SIGSpringBoot101/lab 3/postman
```

Test the interface with the last 5 requests in the Postman collection:



The responses in Postman will not look any different now they come from the SpringBoot application running inside a Docker container compared with before when the application was executed from within Eclipse.

At this moment, the container is running inside the same VM as Eclipse. However, we can move that container virtually anywhere (where we can run a Docker container) and access the REST API there.