Student: Marta Muñoz San Román

Docker – self guided exercise for a Java app

2) **Run using volumes non-interactively**

First, let's create a volume to save the Java application

```
docker volume create java-vol
```

```
marta@marta-VirtualBox:~$ sudo docker volume create java-vol
[sudo] password for marta:
java-vol
```

Before running a container, we should check where it is really located with docker inspect.

```
docker inspect java-vol
```

```
marta@marta-VirtualBox:~$ sudo su
root@marta-VirtualBox:/home/marta# docker inspect java-vol
[
    {
        "CreatedAt": "2021-12-16T16:40:02+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/java-vol/_data",
        "Name": "java-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

In my particular case, the route is: "`/var/lib/docker/volumes/java-vol/_data`"

Then, you should copy the Java file into the volume folder.

I have copied the Main.java in the Text Editor of Linux, saving it in my Desktop and then copy it with cp and the route of "`/var/lib/docker/volumes/java-vol/_data`"

```
root@marta-VirtualBox:/home/marta# cp Desktop/Main.java /var/lib/docker/volumes/java-vol/_data
root@marta-VirtualBox:/home/marta# ls /var/lib/docker/volumes/java-vol/_data
Main.java
```

At this point we can compile the Java file associating the volume location.

```
docker run --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk javac /usr/src/myapp/Main.java
```

```
root@marta-VirtualBox:/home/marta# docker run --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk javac /usr/src/myapp/Main.java
```

It is not necessary to create an image, but the main disadvantage is that we need to create two different containers to compile and run. The first container should have created the **Main.class** in the volume folder.

```
root@marta-VirtualBox:/home/marta# ls /var/lib/docker/volumes/java-vol/_data
Main.class  Main.java
```

Finally, we can run the compiled file.

```
docker run --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk java /usr/src/myapp/Main.java
```

```
root@marta-VirtualBox:/home/marta# docker run --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk java /usr/src/myapp/Main.java
Hello, World
```

3) **Run using volumes interactively**

Using the same volume as in part 2, it is possible to run a container with an interactive bash from an openjdk image (default option without bash opens a jshell, whose commands we have not studied).

```
docker run -it --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk bash
```

```
root@marta-VirtualBox:/home/marta# docker run -it --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk bash
bash-4.4#
```

Then, we can compile and run from bash using the javac and java command in the directory /usr/src/myapp in which we have the volume.

```
bash-4.4# cd /usr/src/myapp

bash-4.4# javac Main.java

bash-4.4# java Main.java

Hello, World
```

```
root@marta-VirtualBox:/home/marta# docker run -it --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk bash
bash-4.4# cd /usr/src/myapp
bash-4.4# javac Main.java
bash-4.4# java Main.java
Hello, World
```

Then, we finish the bash process with the corresponding **exit** command.

```
root@marta-VirtualBox:/home/marta# docker run -it --rm --name my-running-app --mount source=java-vol,destination=/usr/src/myapp openjdk bash
bash-4.4# cd /usr/src/myapp
bash-4.4# javac Main.java
bash-4.4# java Main.java
Hello, World
bash-4.4# exit
exit
```

Regardless of the alternative, this example shows the utility of Docker. We do not need to install Java in our computer. And every employee has the same container with the same configuration.