Software Engineering Lab Sheet 6: solutions

1. Running Docker locally

   (b) *Pull* and *run* the Docker image for `alpine`, version `3.4`. Use the arguments `-it` to make the shell interactive, and add the command to execute (`sh`) to the end of the call.

```
$ docker pull alpine:3.4
3.4: Pulling from library/alpine
3690ec4760f9: Already exists
Digest: sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c
Status: Downloaded newer image for alpine:3.4

$ docker run -it alpine:3.4 sh
  now we are in the alpine VM
  # echo "Hello from Alpine!"
  Hello from Alpine!
  # uname -a
  Linux 5836550c9811 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64 Linux
  # exit
```

   (c) After exiting the shell, list your images, list all your containers and remove the newly created alpine container. Check the container list again to make sure it is removed.

```
list images
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              3.4                 baa5d63471ea        8 weeks ago         4.803 MB

list Docker containers - calling without -a only shows the running ones
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
1852c92f23d8        alpine:3.4          "sh"                3 seconds ago       Exited (0) 1 seco

take ID of container from above, and remove based on it
$ docker rm 1852c92f23d8
1852c92f23d8

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

(d) Write a shell file that prints "Hello World". *Build* a Docker image based on `alpine:3.4` which executes this shell file. Check that the image is successfully built.

```
write hello world script and Dockerfile. contents should be as below
$ cat hello.sh
echo Hello World!
$ cat Dockerfile
  # start with alpine image
  FROM alpine:3.4
  # copy script to root of image
  COPY hello.sh /
  # set executable flag of script to true
  RUN chmod a+x /hello.sh
  # when container of this image is run, execute script
  CMD ./hello.sh

build image with name "helloworld" based on Dockerfile in current folder (.)
$ docker build -t helloworld .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM alpine:3.4
 ---> baa5d63471ea
Step 2 : COPY hello.sh /
 ---> 49a0f2ab42c9
Step 3 : RUN chmod a+x /hello.sh
 ---> Running in b90171e8a85b
 ---> 86eeccb952b7
Step 4 : CMD ./hello.sh
 ---> a58289733353
Successfully built a58289733353

check to see image now exists
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
helloworld          latest              a58289733353        3 seconds ago       4.803 MB
alpine              3.4                 baa5d63471ea        8 weeks ago         4.803 MB
```

(e) *Run* the image with `--rm`. This automatically removes the container when it exits.

```
$ docker run --rm helloworld
Hello World!
```

2. Uploading to the Docker Hub

   (a) Create an account on the Docker Hub. Proceed to login from command line.

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a
Docker ID, head over to https://hub.docker.com to create one.
Username: <docker hub username here>
Password: <docker hub password here>
Login Succeeded
```

   (b) Build your "Hello World" image again, but this time give it the name `<username>/helloworld`.

```
notice how every step below uses a cache and doesn't rebuild until necessary
$ docker build -t csabasulyok/helloworld .
Sending build context to Docker daemon 6.144 kB
Step 1 : FROM alpine:3.4
 ---> baa5d63471ea
Step 2 : COPY hello.sh /
 ---> Using cache
 ---> 49a0f2ab42c9
Step 3 : RUN chmod a+x /hello.sh
 ---> Using cache
 ---> 86eeccb952b7
Step 4 : CMD ./hello.sh
 ---> Using cache
 ---> a58289733353
Successfully built a58289733353
```

   (c) *Push* the new image to the Docker Hub. Check the website and make sure it is there: `https://hub.docker.com/r/<username>/helloworld/`

```
$ docker push csabasulyok/helloworld
The push refers to a repository [docker.io/csabasulyok/helloworld]
1f7470572997: Pushed
011b303988d2: Pushed
latest: digest: sha256:afe958aabcecd8bc87e8a3063626e3be7ba7d9813624793fe5b82c357d2dff17 size: 94
```

3. Running a web server in Docker

   (b) *Build* a Docker image based on `node:4.7-alpine` that executes your JS file. *Expose* the port `8080` in the Dockerfile.

```
new Dockerfile should look something like this
$ cat Dockerfile
  # start with node image this time
  FROM node:4.7-alpine
  # the VM's firewall should allow access to port 8080
  EXPOSE 8080
  # create directory /usr/src on VM
  RUN mkdir /usr/src
  # copy JS file
  COPY server.js /usr/src
  # directory to be in when VM starts
  WORKDIR /usr/src
  # run node when VM starts, with argument server.js
  CMD node server.js

build new image with name "nodeserver"
$ docker build -t nodeserver .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM node:4.7-alpine
 ---> cd3c1431054d
Step 2 : EXPOSE 8080
 ---> 3707fb428841
Step 3 : RUN mkdir /usr/src
 ---> f23953a0d37a
Step 4 : COPY server.js /usr/src
 ---> 103f29ef4ab0
Step 5 : WORKDIR /usr/src
 ---> 313dd1e14c2d
Step 6 : CMD node server.js
 ---> f36cec1e4cae
Successfully built f36cec1e4cae
```

   (c) *Run* your new image, giving it the argument to forward the port (`-it -p 8080:8080`). You should see a message when accessing `http://localhost:8080`.

```
$ docker run --rm -it -p 8080:8080 nodeserver
Server listening on: http://localhost:8080
```

(d) Change line 5 (defining the PORT) in the JS file to `const PORT=process.env.PORT;`. This makes it want to receive the port from an environment variable. Define an environemnt variable in the Dockerfile and expose the same port. Rebuild and run the image with a different port.

```
after changing the scripts, the Dockerfile should look like this
$ cat Dockerfile
  FROM node:4.7-alpine
  # change exposed port to 12345
  EXPOSE 12345
  # add same port as environment variable
  ENV PORT=12345
  RUN mkdir /usr/src
  COPY server.js /usr/src
  WORKDIR /usr/src
  CMD node server.js

rebuild image
$ docker build -t nodeserver .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM node:4.7-alpine
 ---> cd3c1431054d
...
Successfully built f36cec1e4cae

now run exposing the new port
$ docker run --rm -it -p 12345:12345 nodeserver
Server listening on: http://localhost:12345
```

4. Deploying the web server to the cloud

(a) Create an account on Heroku. Proceed to login from command line using `docker login registry.heroku.com`. For this login, you need to use your e-mail as user and your **public API key** as password. You can find the key here.

```
add URL as parameter; you can also add username and password using -u and -p
$ docker login registry.heroku.com
Username: <e-mail here>
Password: <api key from Heroku here>
Login Succeeded
```

(c) Recreate your NodeJS image with the name `registry.heroku.com/ubbse2016-<username>-node/web`.

```
the app name on Heroku and the one in the name must coincide
$ docker build -t registry.heroku.com/ubbse2016-scim0864-node/web .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM node:4.7-alpine
 ---> cd3c1431054d
...
Successfully built f36cec1e4cae
```

(d) *Push* the new image. Your code should now be live under `https://ubbse2016-<username>-node.herokuapp.com`.

```
$ docker push registry.heroku.com/ubbse2016-scim0864-node/web
The push refers to a repository [registry.heroku.com/ubbse2016-scim0864-node/web]
658e9eaf2600: Pushed
162277144b5b: Pushed
85f850abdf89: Pushed
011b303988d2: Pushed
latest: digest: sha256:7d8960f2a32f4cfae768f6c537fcafb7a9e449f50a29288f2234ae2ca708e6e3 size: 115
```