DOC1 Exercise(s) 4

Contents

Manual docker commands in the shell

Note: All of (sub-) part of this exercise should be done in a WSL (Linux) terminal window running Bash.

Make sure you have started Docker Desktop (called DD in the following), and use this command to make sure we have an Ubuntu-based image (**if you have not already done that**):

```
> docker pull ubuntu:latest
```

4.1.1 – What about the image?

Try to answer these questions:

1) What does the docker command above do?
2) Using the Docker registry (https://hub.docker.com/search?q), can you find out which specific versions of Ubuntu this image is based on?

4.1.2 – Find out what images and containers you have

Use DD to get an overview of the images and containers you have available locally. Find out how to list the same information in the shell using docker commands (hints: Google "docker list images" and "docker list containers".

4.1.3 – Create a new container

Create a new container:

```
> docker container create --name exercise4 -it ubuntu /usr/bin/bash
```

Use DD and a docker command to ascertain that you now have a new container called *exercise1*.

4.1.4 – Run the new container (start and stop)

Run the new container:

```
> docker start -i exercise4
```

What happens? (what's the current folder, what application are you running inside the container, etc?)

Press ^D (Control-D) – what happens? (remember that ^D means end-of-file/EOF in Unix/Linux)

Start the container again like before. Then create a folder and a file like this:

```
mkdir opt
cd opt
echo "This is a file" > info.txt
```

Exit the container again with ^D. Start it again – is your file (info.txt) still there?

4.1.5 – Using volume(s) with a container

If you delete your container from before (exercise1), all the changes created inside would be lost. So now we will try to create a docker volume that can be used by a container to store things in a way that allows us to save data so that they can survive deletion of the container.

Step 1, create a docker volume:

```
❯ docker volume create vol_exercise4
```

Just for fun, use DD to create another volume called *vol_todelete*.

List your volumes:

```
❯ docker volume ls
```

Can you see both volumes? Now delete the volume called *vol_todelete* from inside DD.

Step 2, create *and* start a new container called *exercise4_1_5* using the docker run command like this (in a single long line):

```
❯ docker run -it -v vol_exercise4:/mydata --name exercise4_1_5 ubuntu
  bash
```

The container will start, running a bash shell, and the volume is mounted in /mydata.

Create a file in the /mydata folder:

```
# cd mydata
# echo "This file should not be deleted" > info.txt
# cat info.txt
```

Now quit the container and delete it with DD! While you are in DD check that the volume is still there.

Step 3, create and start new container called exercise4_1_5B:

```
❯ docker run -it -v vol_exercise1:/mydata --name exercise4_1_5B ubuntu
  bash
```

Is the /mydata/info.txt file available in the new container?

4.1.6 – Now, all by yourself, create a new volume and container

For this exercise, look at the previous examples and use the excellent Docker documentation at https://docs.docker.com/reference/, especially https://docs.docker.com/engine/reference/commandline/create/ to find out how the *docker create* command works and

1. Create a new volume called *vol_exercise4_1_6*.
2. Create a new container (using the *docker create* command) called *con_exercise4_1_6*. The new container should have the *vol_exercise4_1_6* mounted in a folder called */workspace*.
3. Start your spanking new container so that it runs a bash shell.
4. ------- *Give yourself a high-five when you get to this point*
5. Still inside the container, update your container and install the micro editor like this:

```
# apt update
# apt install micro
```

6. Use micro to create a file inside the */workspace* folder. The file should be called *systeminfo.txt*, and the text should hold (each on a separate line):
   a. Your initials
   b. The results of running the command ***date*** inside the container
   c. The results of running the command ***uname -a*** inside the container

   Hint to step 6: First write your initials in the first line of the file, save the file and exit micro. Then run a command and select/copy (Ctrl-C) the output, start micro on the file again and paste the output into micro (Ctrl-V). Rinse and repeat for each command.

7. Exit the container.
8. Jump into DD, find the volume (*vol_exercise4_1_6*) and look at the contents. Save the file systeminfo.txt to somewhere on your “normal” Windows file system.
9. Be ready to show the contents of the file as a badge of merit for being promoted from the rank of *ContainerNewbie* to *ContainerApprentice*.

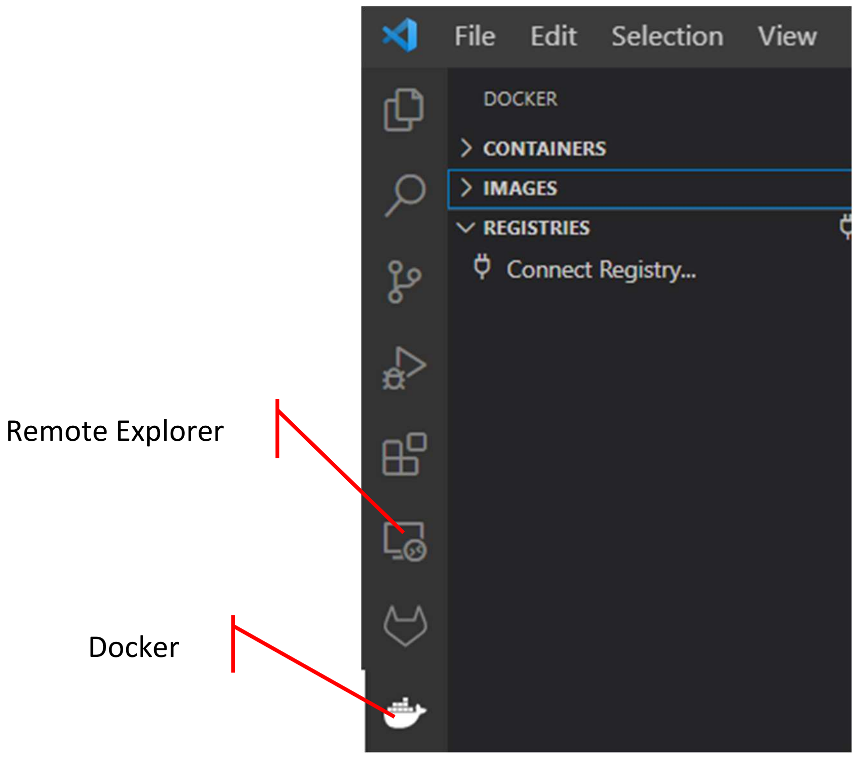
Exercise 4.2 – Working with containers using VSCode

Note: All of (sub-) part of this exercise should be done with VSCode. Make sure you have these extensions installed (VSCode may suggests some of these automatically):

Extension title	*Extension identifier*
Docker for Visual Studio Code	`ms-azuretools.vscode-docker`
Remote Development (pack of 4)	`ms-vscode-remote.vscode-remote-extensionpack`
Visual Studio Code Remote Explorer	`ms-vscode.remote-explorer`

4.2.1 – Playing with the Docker extension

Once you install the extensions above, there will be added some icons to the left in VSCode:



The next part assumes that you have pulled some images (i.e. you have some images cached locally), created some containers and some volumes. If not, just make a few.

Now, simply click on the Docker extension icon and play around. Try to find out exactly what you can see (images, containers etc.) and what you can do for each type of docker object (start, stop, run, delete etc.).

4.2.2 – Attaching to any container

VSCode allows us to attach to any container and look inside as well as change the content of the container. For this sub-exercise follow these steps

Step 1: Create a new container called exercise22 like this (we'll use the Alpine Linux for this):

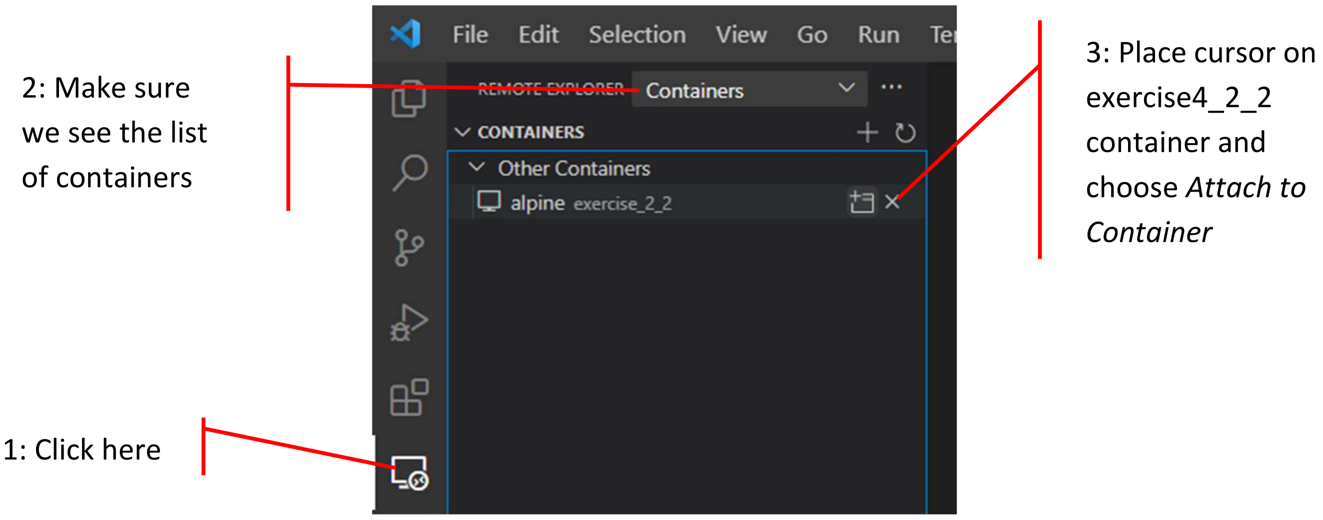
```
❯ docker create -i -t --name exercise4_2_2 alpine
```

Step 2: Start the container in a shell and peek around:

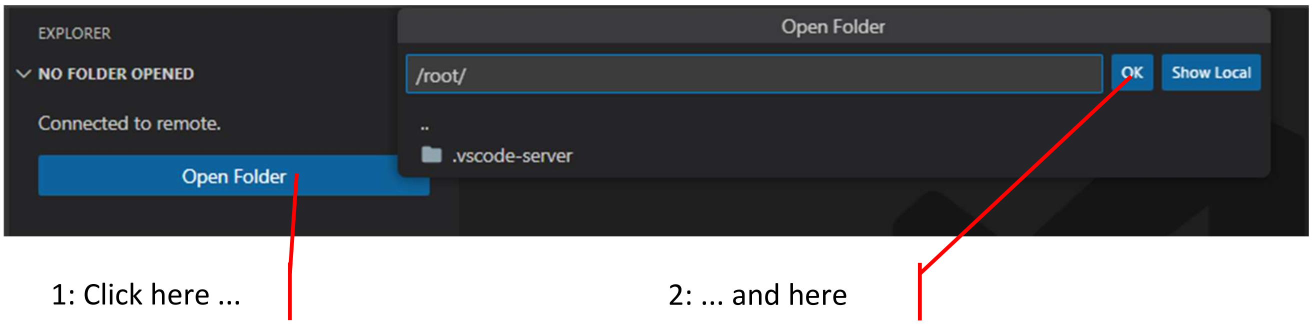
```
❯ docker start -i exercise4_2_2
```

Change to the "home folder" */root* and list the contents of the folder with *ls -al*. Notice what files and folders are there, you can also simply copy/paste the output of *ls -al* for later comparison. Now exit the container.

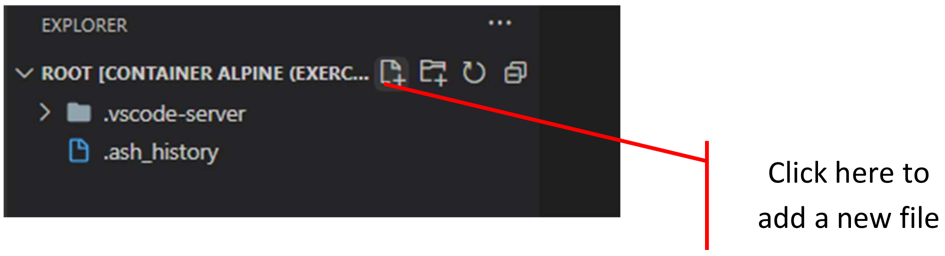
Step 3: Start VSCode and this time click on the *Remote Explorer* icon and attach to the container:


2: Make sure we see the list of containers
1: Click here
3: Place cursor on exercise4_2_2 container and choose *Attach to Container*
File
Edit
Selection
View
Go
Run
REMOTE EXPLORER
Containers
CONTAINERS
Other Containers
alpine exercise_2_2

Step 4: Open the "home folder" /root as our project folder:



Now you have full, normal access to the folder /root inside the container. Try it out by adding a new file in the folder (and put some text in the file):


EXPLORER
ROOT [CONTAINER ALPINE (EXERC...
.vscode-server
.ash_history
Click here to add a new file

Step 5: Close your remote session by choosing *Close Remote Connection* from the File menu.

Step 6: Enter the container from your shell (like in step 2) and find the file you just created with VSCode. View the contents with the *less* command. Apart from the file, what else has changed in the /root folder (compare with what you noted in step 2)?