Intro to Docker workshop - worksheet 1

This document will guide you through the basic docker commands and options. By the end of this worksheet, you should have a working knowledge of how to pull containers from the registry and use them for your own purposes.

Follow the instructions and think about the questions made at each step. If you need help, feel free to call for help from the instructors.

- So you ran **docker run hello-world** and you got the correct output. Good! Now, see the the line toward the end of the output where it says something like

  "To try something more ambitious, you can run an Ubuntu container with:
  $ docker run -it ubuntu bash"

  Do it. **docker run -it ubuntu bash**

  After downloading the necessary images, you will be in a regular Ubuntu terminal, with access to regular Ubuntu linux commands. However, if you navigate around the filesystem, you'll see that you don't have access to stuff outside the container. Oh, in case you hadn't figured it out you're inside your ubuntu container. Well, "inside" isn't really accurate… In reality, your *stdin* and *stdout* (and *stderr*) were redirected to the containers input and output. You can type **exit** to leave.

- You can use **docker ps** to see which containers are active. If you do, you will notice that you probably don't have an ubuntu image running, even though you were just using it.

  Go ahead and **docker ps -a** and you'll see your container has been stopped. Why do you think that is?

  To avoid a clutter of containers potentially taking up your precious disk space, you can run **docker container prune** to get rid of all stopped containers.

- Go ahead bring up another ubuntu container, but this time we want to stay "out" of it. So **docker run ubuntu** would be enough…
  You've probably realised by now that it's not enough. Your container gets created and is immediately stopped. Why do you think that happens?

  To run an ubuntu container that will stay up without any running commands, you can run **docker run -td ubuntu** . With this you've essentially got a black box ubuntu container, where you can run commands. Get the container's id from **docker ps** if you hadn't caught it and try **docker exec {id} uname -a** .

- Another cool thing is that you can copy stuff into your containers. Create a text file (**echo “Hello” > test.txt**) and then try **docker cp text.txt {id}:/** you can verify that it worked with **docker exec {id} cat /text.txt** .

- Now for the wonder of shared volumes! Say you want to work on a C script using the linux API. It’s a pain to have to use a cli text editor within the container and it’s also a pain to have to copy the file in every time you need to test something.

  The solution is share the project folder between the host filesystem and the container’s. To do this, you can run **docker run -it -v /path/to/folder {id}:/destination/path ubuntu**

  Let’s try it! If you still have your file from the previous exercise, do **docker run -it -v /path/to/the/folder/with/the/file {id}:/home ubuntu** . Inside the container, run **tail -f /path/to/file** and make some changes to it outside the container.

  You’ll see that your changes are updated in real time. Cool, right? If you change it inside the container, the changes will also appear in real time on the host filesystem.

- Finally, you can make a new image out of the changes you’ve made to an existing container. For example, if you created a file or installed something on your ubuntu container, you can run **docker commit {id} {image name}** to have the container’s state immortalized in it’s own image.

  Using this, you can set up an image with your own setup, which makes it easier to deploy or develop your projects.

  Bear in mind, though, that you have at your disposal millions of plug-and-play containers you can download, with most of the functionalities you might ever need. You can search for images with **docker search {what you want}** and you’ll get a ton of packages. Alternatively, just google it!

- **FINAL EXERCISE:** See if you can use what you’ve learned until now to create an image which serves a static web page locally. (hint: Start with an image that already has a web server)

- Other cool commands:
  - **docker image ls** - look at the images you have locally
  - **docker image rm -** delete images
  - **docker image rm $(docker image ls -q)** - remove all images