TP: Docker

First steps with containers : docker run (and pull)

0. (if you work on the university machine) : Execute the instruction `dockersh` in a shell: this is very specific to university paris-saclay, which has so many users that its UNIX user ids exceed what docker can manage, and also because granting each individual student access to docker would not be very practical anyway. The dockersh instruction changes your user to "dockerlocal", the only user which is allowed to run docker (and which has a low-enough userid). Beware: dockerlocal does not have the right to read your files and files downloaded from a browser will have your rights, not those of dockerlocal (a solution willbe to download any file you need in the shell as dockerlocal through wget, the file then belongs to dockerlocal. An alternative solution would be using chmod to let anybody process the file). You will probably face this minor issue for the last question in this lab, and we may face it later on in other labs.

1. Start a container (no need to first download the image explicitly) that runs the latest version of the hello-world image:

   `docker run hello-world:latest`

   If the image has not been pulled (downloaded) before by yourself or somebody else, you may observe the following instruction, which indicates that "docker run" performed automatically a "docker pull" first.

   ```
   Unable to find image 'hello-world:latest' locally
   latest: Pulling from library/hello-world
   ```

2. Pull the latest version of the ubuntu Docker image

   We essentially care about images from the official Docker Hub registry in these lectures. The image is generally described on docker hub and sometimes provides information about how to run the image, though the ubuntu page https://hub.docker.com/_/ubuntu is of no practical use for us.

3. Start a container running this ubuntu image; the container must start an interactive bash shell inside (launch container with `-it`).

   - Actually, the ubuntu image executes the command bash when starting the container, like the hello-world image executes its program /hello. So you need not specify the command "bash" (you may, but it's not necessary).
   - The `-it` is necessary. If you omit the `-i` option the container exits immediately. If instead you omit the `-t` option it stays up, but you don't attach a pseudo-TTY; you could open another terminal, then attach in this terminal a pseudo-tty to the container with `docker exec -t my-ubuntu-container bash`, but this is wasteful.
   - Again, if we had not pulled the image first, the `docker run` instruction would perform the pull anyway.

4. [On the bash that runs inside the container] list the processes running inside the container with `ps`. [Outside] Open a shell outside the container and list processes running on host machine.

   You should observe more processes and higher PIDs on the host machine.

5. [In another terminal] Create another container, also with a bash attached, of the same ubuntu image, and give it an explicit name, ex: `myubuntu`. Then check that a file created on container `myubuntu` (`touch myfile.txt`) does not appear in the first ubuntu container.

- `--name myubuntu` will set the name to `myubuntu` instead of using an auto-generated name.
- The regex that describes valid container names is: `^[a-zA-Z0-9][a-zA-Z0-9_.-]+$`
- You should generally name explicitly your containers. It simplifies container management. Tip: the shell may be able to auto-complete container's names when you identify containers by their name in docker instructions.
- We can consider (for now, as a simplification) that each container essentially has its own file-system, though it would be more accurate to describe each container as a writable layer on top of the image layers.

At PUIO, multiple students share the same docker machine. So you may find that other students already created a container with the name you had in mind; just pick another name (ex: you may call the container `ubuntu_your_surname`).

Executing tasks from outside the container, life of a container

6. Find out which instructions can be performed in docker using `docker help`; deduce how you can list images, and list containers. Obtain more information about the command `docker ps` and deduce how to list the docker containers present on the machine including the ones that are not presently running.

   Any `docker ...` instruction is to be executed from a shell on the host machine. Not from a shell inside a container: we will never run docker inside a docker container, though it would technically be possible.

7. Start a container `my_alp` running the alpine image in the background (`-d`). Check the image size.

   - The alpine image is a very lean image, so is often used as a base image in docker containers.
   - This image executes the command `/bin/sh`. The image does contain the bash implementation of shell, but only ash (bin/sh and bin/ash both link to /bin/busybox).

8. [From a terminal outside the container] Execute instructions on your container `my_alp` from outside the container: first create a file a.txt on the container. Then list the files at the root.

   We expect an answer using two distinct `docker exec` instructions. The two instructions could however be packed into a single shell instruction provided as a command-string operand (`-c`) rather than from the standard input; `docker exec my_alp sh -c "instruction 1; instruction2"`

9. (Optional) Execute and understand the following two instructions:

```
docker exec -e VAR_A=1 -e VAR_B=2  my_alp  env
docker exec -w /usr/bin  my_alp  sh -c 'printenv'
```

10. Interrupt the container with `docker stop my_alp`. Check its status, and check that you cannot perform instructions on the container. Then restart the container with `docker restart my_alp`. Check that the file you created is still in the container.

    When you leave your session, containers will be interrupted. In theory interrupted containers are persisted on the machine and can be restarted when you reconnect. However, we sometimes aggressively cleanup the docker machine (among others, because students often forget to do it) so there is no guarantee that you will recover your containers. Consider containers at PUIO as temporary and unreliable.

11. Remove the hello-world container you created, with `docker rm`. and remove the running ubuntu containers.

Exchanging data with containers

12. Copy a file from your host machines into the container `my_alp`.

    - `docker cp` actually copies recursively : if you provide a directory as a source, the whole directory is copied.
    - a path must necessarily be specified on the container (unlike in `scp` file transfer where the home directory is used by default if no directory is specified in the target).