Part 1

Running and stopping containers

Next we will start using a more useful image than hello-world. We can run ubuntu just with `docker run ubuntu`.

```
$ docker run ubuntu
  Unable to find image 'ubuntu:latest' locally
  latest: Pulling from library/ubuntu
  83ee3a23efb7: Pull complete
  db98fc6f11f0: Pull complete
  f611acd52c6c: Pull complete
  Digest: sha256:703218c0465075f4425e58fac086e09e1de5c340b12976c
  Status: Downloaded newer image for ubuntu:latest
```

Anticlimactic as nothing really happened. The image was downloaded and ran and that was the end of that. It actually tried to open a shell but we will need to add a few flags to interact with it. `-t` will create a tty.

```
$ docker run -t ubuntu
  root@f83969ce2cd1:/#
```

Now we're inside the container and if we input `ls` and press enter... nothing happens. Because our terminal is not sending the messages into the container. The `-i` flag will instruct to pass the STDIN to the container. If you're stuck with the other terminal you can just stop the container.

```
$ docker run -it ubuntu
  root@2eb70ecf5789:/# ls
  bin   boot   dev   etc   home   lib   lib32   lib64   libx32   media   r
```

Great! Now we know at least 3 useful flags. `-i` (interactive), `-t` (tty) and `-d` (detached).

Let's throw in a few more and run a container in the background:

```
$ docker run -d -it --name looper ubuntu sh -c 'while true; do d
```

> **Quotes**
>
> If you are command prompt (Windows) user you must use double quotes around the script i.e. `docker run -d -it --name looper ubuntu sh -c "while true; do date; sleep 1; done"`. The quote or double-quote may haunt you later during the course.

- The first part, `docker run -d`. Should be familiar by now, run container detached.
- Followed by `-it` is short for `-i` and `-t`. Also familiar, `-it` allows you to interact with the container by using the command line.
- Because we ran the container with `--name looper`, we can now reference it easily.
- The image is `ubuntu` and what follows it is the command given to the container.

And to check that it's running, run `docker container ls`

Let's follow `-f` the output of logs with

```
$ docker logs -f looper
  Thu Feb  4 15:51:29 UTC 2021
  Thu Feb  4 15:51:30 UTC 2021
  Thu Feb  4 15:51:31 UTC 2021
  ...
```

Let's test pausing the looper without exiting or stopping it. In another terminal run `docker pause looper`. Notice how the logs output has paused in the first terminal. To unpause run `docker unpause looper`.

Keep the logs open and attach to the running container from the second terminal using 'attach':

```
$ docker attach looper
  Mon Jan 15 19:26:54 UTC 2018
  Mon Jan 15 19:26:55 UTC 2018
  ...
```

Now you have process logs (STDOUT) running in two terminals. Now press control+c in the attached window. The container is stopped because the process is no longer running.

If we want to attach to a container while making sure we don't close it from the other terminal we can specify to not attach STDIN with `--no-stdin` option. Let's start the stopped container with `docker start looper` and attach to it with `--no-stdin`.

Then try control+c.

```
$ docker start looper

$ docker attach --no-stdin looper
  Mon Jan 15 19:27:54 UTC 2018
  Mon Jan 15 19:27:55 UTC 2018
  ^C
```

The container will continue running. Control+c now only disconnects you from the STDOUT.

To enter a container, we can start a new process in it.

```
$ docker exec -it looper bash

  root@2a49df3ba735:/# ps aux

  USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TI
```

```
root         1  0.2  0.0   2612  1512 pts/0    Ss+  12:36   0
root        64  1.5  0.0   4112  3460 pts/1    Ss   12:36   0
root        79  0.0  0.0   2512   584 pts/0    S+   12:36   0
root        80  0.0  0.0   5900  2844 pts/1    R+   12:36   0
```

From the `ps aux` listing we can see that our `bash` process got PID (process ID) of 64.

Now that we're inside the container it behaves as you'd expect from ubuntu, and we can exit the container with `exit` and then either kill or stop the container.

Our looper won't stop for a SIGTERM signal sent by a stop command. To terminate the process, stop follows the SIGTERM with a SIGKILL after a grace period. In this case, it's simply faster to use kill.

```
$ docker kill looper
$ docker rm looper
```

Running the previous two commands is basically equivalent to running `docker rm --force looper`

Let's start another process with `-it` and add `--rm` in order to remove it automatically after it has exited. The `--rm` ensures that there are no garbage containers left behind. It also means that `docker start` can not be used to start the container after it has exited.

```
$ docker run -d --rm -it --name looper-it ubuntu sh -c 'while tr
```

Now let's attach to the container and hit control+p, control+q to detach us from the STDOUT.

```
$ docker attach looper-it

  Mon Jan 15 19:50:42 UTC 2018
  Mon Jan 15 19:50:43 UTC 2018
  ^P^Qread escape sequence
```

Instead, if we had used ctrl+c, it would have sent a kill signal followed by removing the container as we specified `--rm` in `docker run` command.

Exercise 1.3: Secret message

Now that we've warmed up it's time to get inside a container while it's running!

Image `devopsdockeruh/simple-web-service:ubuntu` will start a container that outputs logs into a file. Go inside the container and use `tail -f ./text.log` to follow the logs. Every 10 seconds the clock will send you a "secret message".

Submit the secret message and command(s) given as your answer.

Exercise 1.4: Missing dependencies

Start a ubuntu image with the process `sh -c 'echo "Input website:"; read website; echo "Searching.."; sleep 1; curl http://$website;'`

You will notice that a few things required for proper execution are missing. Be sure to remind yourself which flags to use so that the read actually waits for input.

Note also that curl is NOT installed in the container yet. You will have to install it from inside of the container.

Test inputting `helsinki.fi` into the application. It should respond with something like

```
<html>
  <head>
    <title>301 Moved Permanently</title>
  </head>

  <body>
    <h1>Moved Permanently</h1>
```

```
    <p>The document has moved <a href="http://www.helsinki.f
  </body>
</html>
```

This time return the command you used to start process and the command(s) you used to fix the ensuing problems.

- This exercise has multiple solutions, if the curl for helsinki.fi works then it's done. Can you figure out other (smart) solutions?

You have reached the end of this section! Continue to the next section:

→ 3. In-depth dive to images

In this part:

Source code of the material

Report issue Edit page

About the material and its creators.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI