Running your first container from image

Now that you have everything setup, it's time to get your hands dirty. In this section, you are going to run an Alpine Linux container (a lightweight linux distribution) on your system and get a taste of the `docker container run` command.

To get started, let's run the following in our terminal:

```
$ docker pull alpine
```

> **Note:** Depending on how you've installed docker on your system, you might see a `permission denied` error after running the above command. You may need to prefix your `docker` commands with `sudo` as stated before. Alternatively you can create a docker group to get rid of this issue.

The `pull` command fetches the alpine **image** from the **Docker registry** and saves it in your system. You can use the `docker image ls` command to see a list of all images on your system.

```
$ docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
alpine                  latest              c51f86c28340        4 weeks ago
1.109 MB
hello-world             latest              690ed74de00f        5 months ago
960 B
```

1.1 docker container run

Great! Now let's run a Docker **container** based on this image. To do that you are going to use the `docker container run` command.

```
$ docker container run alpine ls -l
total 48
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 bin
drwxr-xr-x    5 root     root           360 Mar 18 09:47 dev
drwxr-xr-x   13 root     root          4096 Mar 18 09:47 etc
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 home
drwxr-xr-x    5 root     root          4096 Mar  2 16:20 lib
......
......
```

What happened? Behind the scenes, a lot of stuff happened. When you call `run`, 1. The Docker client contacts the Docker daemon 2. The Docker daemon creates the container and then runs a command in that container. 3. The Docker daemon streams the output of the command to the

Docker client

When you run `docker container run alpine`, you provided a command (`ls -l`), so Docker started the command specified and you saw the listing.

Let's try something more exciting.

```
$ docker container run alpine echo "hello from alpine"
hello from alpine
```

OK, that's some actual output. In this case, the Docker client dutifully ran the `echo` command in our alpine container and then exited it. If you've noticed, all of that happened pretty quickly. Imagine booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast!

Try another command.

```
$ docker container run alpine /bin/sh
```

Wait, nothing happened! Is that a bug? Well, no. These interactive shells will exit after running any scripted commands, unless they are run in an interactive terminal - so for this example to not exit, you need to `docker container run -it alpine /bin/sh`.

You are now inside the container shell and you can try out a few commands like `ls -l`, `uname -a` and others. Exit out of the container by giving the `exit` command.

Ok, now it's time to see the `docker container ls` command. The `docker container ls` command shows you all containers that are currently running.

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Is *this* a bug? Also no; when you wrote `exit` in the shell, the process stopped. No containers are running, you see a blank line. Let's try a more useful variant: `docker container ls -a`

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
36171a5da744        alpine              "/bin/sh"                5 minutes ago
Exited (0) 2 minutes ago                        fervent_newton
a6a9d46d0b2f        alpine              "echo 'hello from alp"   6 minutes ago
Exited (0) 6 minutes ago                        lonely_kilby
ff0a5c3750b9        alpine              "ls -l"                  8 minutes ago
Exited (0) 8 minutes ago                        elated_ramanujan
c317d0a9e3d2        hello-world         "/hello"                 34 seconds ago
```

```
Exited (0) 12 minutes ago                       stupefied_mcclintock
```

What you see above is a list of all containers that you ran. Notice that the `STATUS` column shows that these containers exited a few minutes ago.

Try using the `run` command again with the `-it` flag, so it attaches you to an interactive tty in the container. You can run as many commands in the container as you want! Take some time to run your favorite commands. (Remember, you can write `exit` when you want to quit.)

Naming your container

Take a look again at the output of the `docker container ls -a`:

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
36171a5da744        alpine              "/bin/sh"                5 minutes ago
Exited (0) 2 minutes ago                        fervent_newton
a6a9d46d0b2f        alpine              "echo 'hello from alp"   6 minutes ago
Exited (0) 6 minutes ago                        lonely_kilby
ff0a5c3750b9        alpine              "ls -l"                  8 minutes ago
Exited (0) 8 minutes ago                        elated_ramanujan
c317d0a9e3d2        hello-world         "/hello"                 34 seconds ago
Exited (0) 12 minutes ago                       stupefied_mcclintock
```

All containers have an **ID** and a **name**. Both the ID and name is generated every time a new container spins up with a random seed for uniqueness. If you want to assign a specific name to a container then you can use the `--name` option. That can make it easier for you to reference the container going forward.

Summary

That concludes a whirlwind tour of the `docker container run` command which would most likely be the command you'll use most often. It makes sense to spend some time getting comfortable with it. To find out more about `run`, use `docker container run --help` to see a list of all flags it supports. As you proceed further, we'll see a few more variants of `docker container run`.