EXERCISE: Play with Docker classroom

https://training.play-with-docker.com/beginner-linux/#Task0

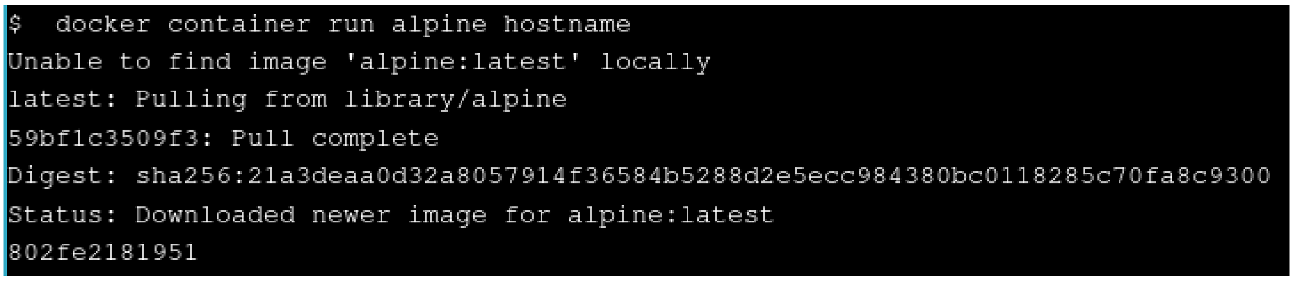
Task 1: Run some simple Docker containers

Run a single task in an Alpine Linux container

In this step we're going to start a new container and tell it to run the hostname command. The container will start, execute the hostname command, then exit.

1. **Run the following command in your Linux console:**

   ```
   docker container run alpine hostname
   ```



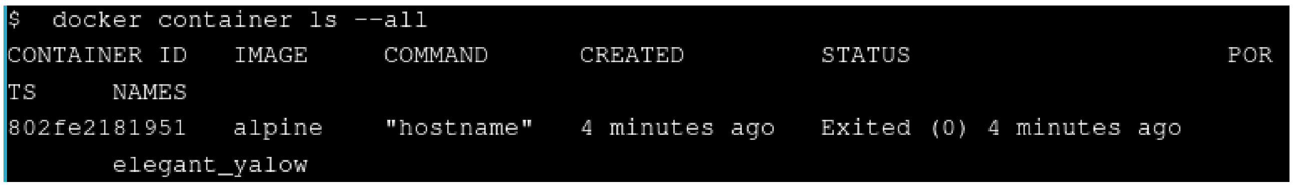
```
$  docker container run alpine hostname
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
59bf1c3509f3: Pull complete
Digest: sha256:21a3deaa0d32a8057914f36584b5288d2e5ecc984380bc0118285c70fa8c9300
Status: Downloaded newer image for alpine:latest
802fe2181951
```

The output below shows that the alpine:latest image could not be found locally. When this happens, Docker automatically *pulls* it from Docker Hub, and downloads it. After the image is pulled, the container's hostname is displayed (888e89a3b36b).

2. **Docker keeps a container running as long as the process it started inside the container is still running. In this case the hostname process exits as soon as the output is written. This means the container stops. However, Docker doesn't delete resources by default, so the container still exists in the Exited state.**

   ```
   docker container ls --all
   ```

List all containers:



```
$  docker container ls --all
CONTAINER ID   IMAGE     COMMAND      CREATED          STATUS                        POR
TS      NAMES
802fe2181951   alpine    "hostname"   4 minutes ago    Exited (0) 4 minutes ago
       elegant_yalow
```

As it's shown above, my Alpine Linux container is in the Exited state.

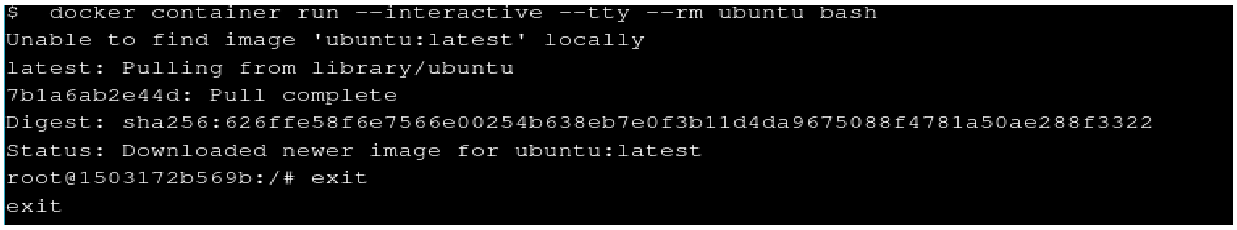
NOTES: The container ID ***is*** the hostname that the container displayed. In the example above it's 888e89a3b36b.

Run an interactive Ubuntu container

You can run a container based on a different version of Linux than is running on your Docker host. In the next example, we are going to run an Ubuntu Linux container on top of an Alpine Linux Docker host (Play With Docker uses Alpine Linux for its nodes).

1. **Run a Docker container and access its shell.**

   ```
   docker container run --interactive --tty --rm ubuntu bash
   ```



```
$  docker container run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
7b1a6ab2e44d: Pull complete
Digest: sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322
Status: Downloaded newer image for ubuntu:latest
root@1503172b569b:/# exit
exit
```

   In this example, we're giving Docker three parameters:

   - --interactive says you want an interactive session.
   - --tty allocates a pseudo-tty.
   - --rm tells Docker to go ahead and remove the container when it's done executing.

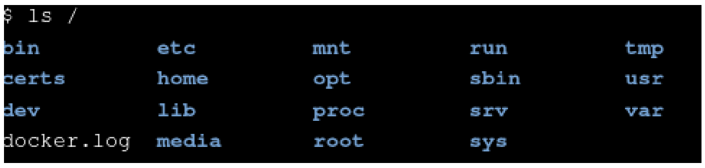
   The first two parameters allow you to interact with the Docker container.

   We're also telling the container to run bash as its main process (PID 1).

   When the container starts you'll drop into the bash shell with the default prompt root@<container id>:/#. Docker has attached to the shell in the container, relaying input and output between your local session and the shell session in the container.

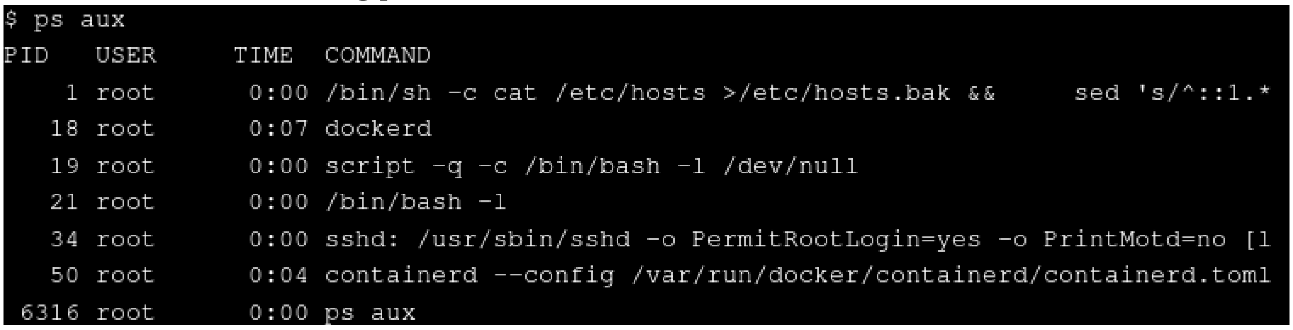
2. **Run the following commands in the container:**

   `ls /` → to list contents of the root directory of the container



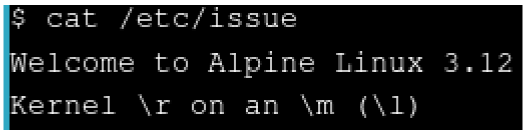
```
$ ls /
bin         etc         mnt         run         tmp
certs       home        opt         sbin        usr
dev         lib         proc        srv         var
docker.log  media       root        sys
```

   `ps aux` → to show running processes in the container



```
$ ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c cat /etc/hosts >/etc/hosts.bak &&     sed 's/^::1.*
   18 root      0:07 dockerd
   19 root      0:00 script -q -c /bin/bash -l /dev/null
   21 root      0:00 /bin/bash -l
   34 root      0:00 sshd: /usr/sbin/sshd -o PermitRootLogin=yes -o PrintMotd=no [l
   50 root      0:04 containerd --config /var/run/docker/containerd/containerd.toml
 6316 root      0:00 ps aux
```

   `cat /etc/issue` → to show which Linux distribution the container is runnning



```
$ cat /etc/issue
Welcome to Alpine Linux 3.12
Kernel \r on an \m (\l)
```

3. **Type `exit` to leave the shell session. This will terminate the `bash` process, causing the container to exit.**

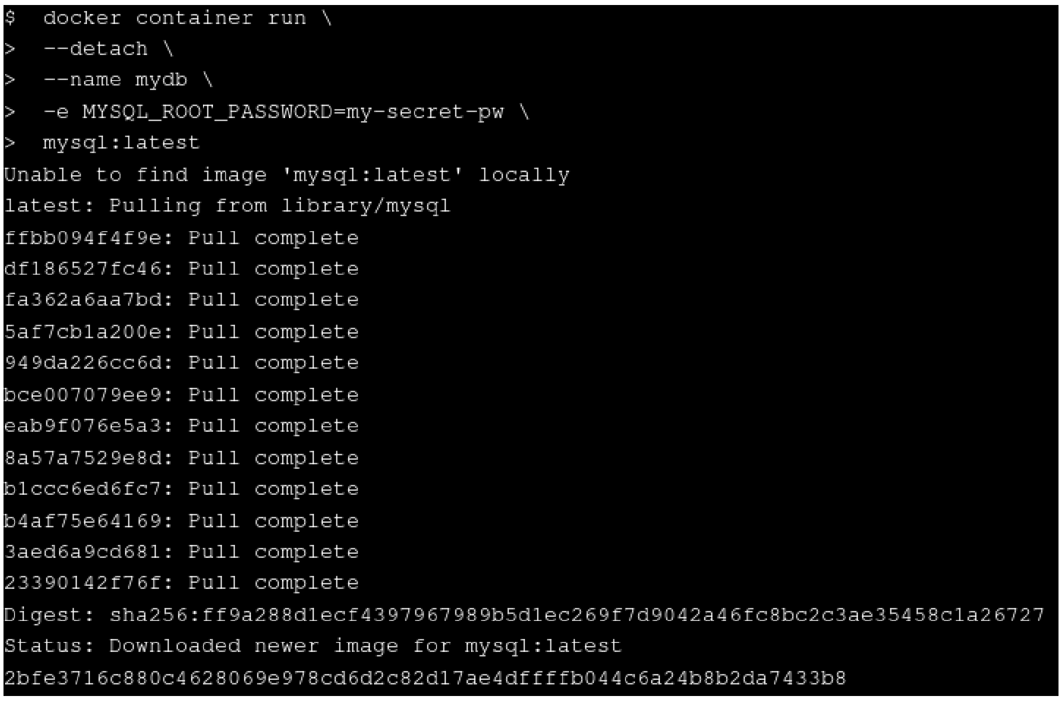
   ```
   Exit
   ```

Run a background MySQL container

Background containers are how you'll run most applications. Here's a simple example using MySQL.

1. **Run a new MySQL container with the following command.**

   `docker container run \`
   `--detach \` → will run the container in the background.
   `--name mydb \` → will name it **mydb**.
   `-e MYSQL_ROOT_PASSWORD=my-secret-pw \` → will use an environment variable to specify the root password (NOTE: This should never be done in production).
   `mysql:latest`

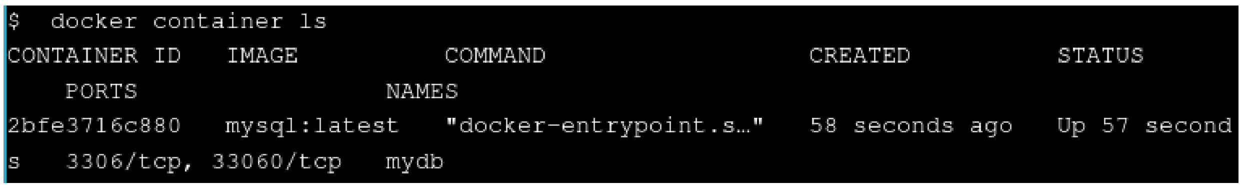
   

   ```
   $  docker container run \
   >  --detach \
   >  --name mydb \
   >  -e MYSQL_ROOT_PASSWORD=my-secret-pw \
   >  mysql:latest
   Unable to find image 'mysql:latest' locally
   latest: Pulling from library/mysql
   ffbb094f4f9e: Pull complete
   df186527fc46: Pull complete
   fa362a6aa7bd: Pull complete
   5af7cb1a200e: Pull complete
   949da226cc6d: Pull complete
   bce007079ee9: Pull complete
   eab9f076e5a3: Pull complete
   8a57a7529e8d: Pull complete
   b1ccc6ed6fc7: Pull complete
   b4af75e64169: Pull complete
   3aed6a9cd681: Pull complete
   23390142f76f: Pull complete
   Digest: sha256:ff9a288d1ecf4397967989b5d1ec269f7d9042a46fc8bc2c3ae35458c1a26727
   Status: Downloaded newer image for mysql:latest
   2bfe3716c880c4628069e978cd6d2c82d17ae4dffffb044c6a24b8b2da7433b8
   ```

   As the MySQL image was not available locally, Docker automatically pulled it from Docker Hub.

2. **List the running containers.**

   ```
   docker container ls
   ```

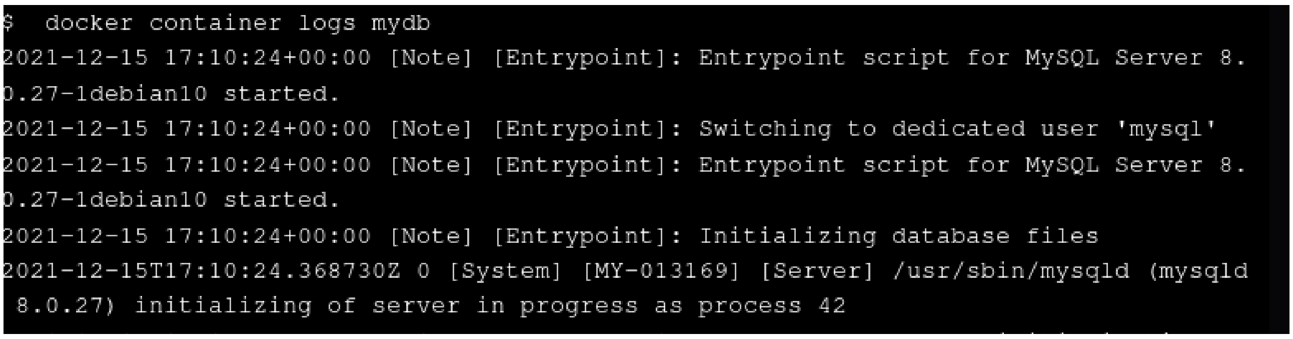


```
$  docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
    PORTS                 NAMES
2bfe3716c880   mysql:latest   "docker-entrypoint.s…"   58 seconds ago   Up 57 second
s   3306/tcp, 33060/tcp   mydb
```

The container is running.

3. **You can check what's happening in your containers by using a couple of built-in Docker commands: docker container logs and docker container top.**

```
docker container logs mydb
```

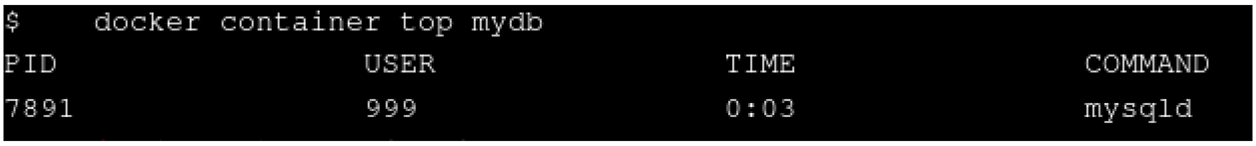


```
$  docker container logs mydb
2021-12-15 17:10:24+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.
0.27-1debian10 started.
2021-12-15 17:10:24+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2021-12-15 17:10:24+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.
0.27-1debian10 started.
2021-12-15 17:10:24+00:00 [Note] [Entrypoint]: Initializing database files
2021-12-15T17:10:24.368730Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld
 8.0.27) initializing of server in progress as process 42
```

This shows the **logs** from the MySQL Docker container.

- Let's look at the processes running inside the container:

```
docker container top mydb
```



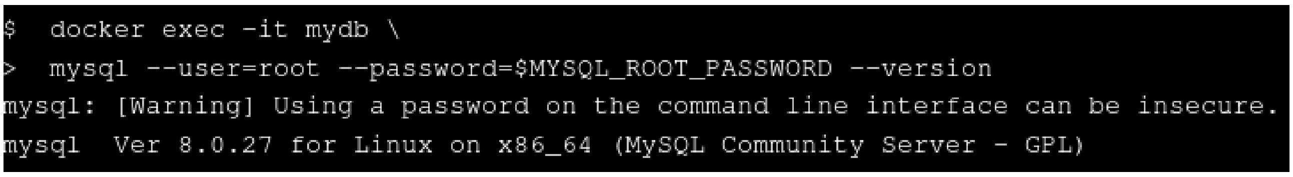
```
$    docker container top mydb
PID                   USER                  TIME                  COMMAND
7891                  999                   0:03                  mysqld
```

The MySQL daemon (mysqld) is running in the container.

4. **List the MySQL version using docker container exec.**

docker container exec allows you to run a command inside a container. In this example, we'll use docker container exec to run the command-line equivalent of mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version inside our MySQL container.

```
docker exec -it mydb \
mysql --user=root --password=$MYSQL_ROOT_PASSWORD –version
```



```
$  docker exec -it mydb \
>  mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql  Ver 8.0.27 for Linux on x86_64 (MySQL Community Server - GPL)
```

MySQL version number, as well as a handy warning is shown.

5. **You can also use docker container exec to connect to a new shell process inside an already-running container. Executing the command below will give you an interactive shell (sh) inside your MySQL container.**

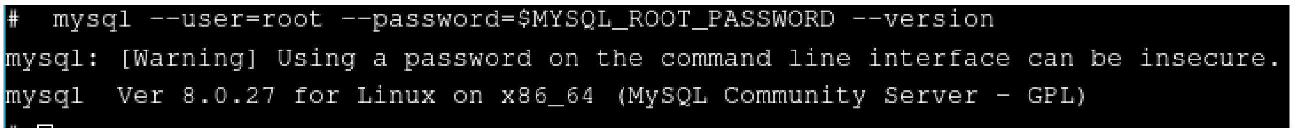
```
docker exec -it mydb sh
```

```
$  docker exec -it mydb sh
```

6. **Notice that your shell prompt has changed. This is because your shell is now connected to the `sh` process running inside of your container.**

Let's check the version number by running the same command again, only this time from within the new shell session in the container.

```
mysql --user=root --password=$MYSQL_ROOT_PASSWORD –version
```

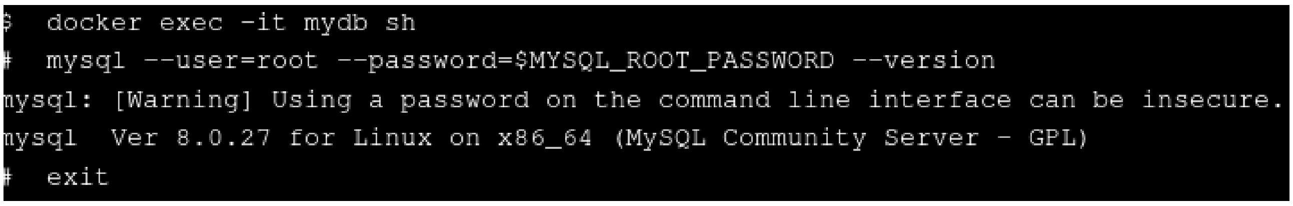


```
#  mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql  Ver 8.0.27 for Linux on x86_64 (MySQL Community Server - GPL)
```

The output is the same as before.

7. **Type exit to leave the interactive shell session.**

   ```
   exit
   ```



```
$  docker exec -it mydb sh
#  mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql  Ver 8.0.27 for Linux on x86_64 (MySQL Community Server - GPL)
#  exit
```