Exercise: Docker Intro Hands-On (BONUS)

Conventions

When you see anything enclosed in "less than" and "greater than" symbols, like `<this>`, interpret what's in those brackets yourself - do not copy and paste `<this>`. Here's an example - when you see:

```
ps <Hit TAB><hit Enter>
```

You should press these keys:

p s spacebar tab enter

Here's another example - when you see:

```
echo <your name>
```

Please type `echo`, then hit the spacebar, then type your name, then hit Enter.

Steps

1. Start up a fresh lxterminal by clicking the "sparrow" logo in the bottom-left corner of the screen, clicking run, typing `lxterminal` and hitting enter. Alternatively, use the hot key sequence below:

```
<hold down Alt><hit F2>lxterminal<HIT the enter key>
```

2. Log in to the docker virtual machine with password `logidebtech`:

```
ssh user@docker
logidebtech
```

3. Sudo to root with the same password `logidebtech`:

```
sudo su -
logidebtech
```

4. We can start using Docker by executing a single client command:

   ```
   docker run -it centos:7 /bin/bash
   ```

5. Your machine will now pull an official Centos 7 image from Docker Hub and starts a container using this as the filesystem, running `/bin/bash` as

its only process. Once the container starts, we'll get a shell. Let's get a process list with `ps`:

```
ps -ef
```

6. Observe that you have only one process besides the bash shell: a ps command. Here was the output on a sample machine:

```
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 17:31 pts/0    00:00:00 /bin/bash
root          14       1  0 17:31 pts/0    00:00:00 ps -ef
```

7. Let's detach from this container's terminal. While holding down Ctrl-P, hit Q:

```
<Hold down the CTRL and P keys><Hit the Q key>
```

8. Get a list of running containers on the system with the `docker ps` command.

```
docker ps
```

9. In the first column, you'll see a container ID, a random SHA-256 hash string. Docker also creates a random human-friendly name, which you'll find in the last column. Note the name of your new container. In the example below, the container is named "upbeat_borg."

```
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          PORTS     NAMES
498e5be7a23b   centos:7   "/bin/bash"   14 seconds ago   Up 13 seconds             eloquent_
```

10. Re-enter the container by attaching a TTY to that container's first process. We'll use Docker's command completion by hitting tab instead of typing a container name, allowing docker to fill in the name of the only running container:

```
docker attach <Hit TAB>
```

11. Let's kill this container off, so we can start a new one with a name that we choose. If we exit the container's PID 1 process, the container will stop.

```
exit
```

12. Let's run a docker ps to list running containers:

```
docker ps
```

13. Observe that the container is no longer in the output - here's the sample output from our example system:

```
CONTAINER ID   IMAGE      COMMAND       CREATED         STATUS         PORTS     NAMES
```

14. The container is still present. We are able to investigate it if we want. For now, let's list all containers, including those no longer running:

```
docker ps -a
```

15. Observe that the container's status column says that it exited some minutes ago. Here's our sample output:

```
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS                     PORTS
498e5be7a23b   centos:7   "/bin/bash"   41 seconds ago   Exited (0) 4 seconds ago
```

16. Let's remove this container from the system. The command below will use `docker ps -a`, place the container ID into a variable called $ctr, then remove (delete) the container. You'll see the container's ID output if the container is removed successfully.

```
ctr=$(docker ps -a | grep centos:7 | awk '{print $1}')
docker rm $ctr
```

17. Let's start a new container using the `centos:7` image, but pass in a `--name` flag to name the container `one_shell`.

```
docker run -it --name=one_shell centos:7 /bin/bash
```

18. Let's create a filesystem change in this container now, to explore union-mounted filesystems.

```
echo "kube" >/etc/changedfile.txt
```

19. Let's detach from this container's terminal. While holding down Ctrl-P, hit Q.

```
<Hold down the CTRL and P keys><Hit the Q key>
```

20. Next, start another container using the `centos:7` image.

```
docker run -it --name=second_container centos:7 /bin/bash
```

21. Check for an `/etc/changedfile.txt` file in this container - you should receive an error saying the file does not exist.

```
ls -l /etc/changedfile.txt
```

22. Let's create that file, but with different contents:

```
echo "1234567890" >/etc/changedfile.txt
```

23. Detach from this container's terminal. While holding down Ctrl-P, hit Q.

```
<Hold down the CTRL and P keys><Hit the Q key>
```

24. Let's stop both containers. This may take a few seconds – Docker will first send a SIGTERM to the container's PID 1 bash shell, wait a bit, then send a SIGKILL:

```
docker stop one_shell second_container
```

25. We know both containers had their own filesystem. The filesystem contents will be deleted when we delete the containers, unless we persist those contents by committing them to an image repository (repo). Let's commit

each of them to a different image repo on this system. First commit the `one_shell` container's contents to the `contents_were_kube` image repo:

```
docker commit one_shell        contents_were_kube
```

26. Commit the `second_container` filesystem contents to the `contents_were_digits` local repo:

```
docker commit second_container contents_were_digits
```

27. Now get a list of the container images on this system:

```
docker images
```

28. Notice that there are three images cached on this system now:

```
REPOSITORY             TAG       IMAGE ID       CREATED              SIZE
contents_were_digits   latest    e2a8b024356b   About a minute ago   204MB
contents_were_kube     latest    28d108028ae4   About a minute ago   204MB
centos                 7         eeb6ee3f44bd   6 months ago         204MB
```

29. Let's use the `contents_were_digits` image to start a new container.

```
docker run -it --name=newctr contents_were_digits /bin/bash
```

30. Check out the contents of the `changedfile.txt` file - you should find the text "1234567890" in this file.

```
cat /etc/changedfile.txt
```

31. Now detach from the container. While holding down Ctrl-P, hit Q.

```
<Hold down the CTRL and P keys><Hit the Q key>
```

32. Stop and remove the `newctr` container, using the `-f` (force) option on `docker rm`:

```
docker rm -f newctr
```

33. Let's look at those two container images we committed using the `docker history` command. First, look at the `contents_were_kube` image.

```
docker history contents_were_kube
```

34. Look at the layers in this image. Here's our sample output:

```
IMAGE          CREATED         CREATED BY                                      SIZE      CO
28d108028ae4   3 minutes ago   /bin/bash                                       5B
eeb6ee3f44bd   6 months ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      6 months ago    /bin/sh -c #(nop)  LABEL org.label-schema.sc...   0B
<missing>      6 months ago    /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4...   204MB
```

31. Compare them to the layers in the `centos:7` image that your machine has cached:

```
docker history centos:7
```

35. Here's the output of that on our test system:

```
IMAGE          CREATED        CREATED BY                                      SIZE      COMM
eeb6ee3f44bd   6 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      6 months ago   /bin/sh -c #(nop)  LABEL org.label-schema.sc...   0B
<missing>      6 months ago   /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4...   204MB
```

36. Note that the `contents_were_kube` image has all the layers from `centos:7` and then one more layer. That layer contains the minimal filesystem changes we made in the `one_shell` container. In our example output, this top layer takes up 5 bytes, which seems to correspond to the bytes "`kube<EOF>`".

37. Take a look at the image layer list on the `contents_were_digits` image with `docker history`.

```
docker history contents_were_digits
```

38. Notice how the `contents_were_digits` differs from the `contents_were_kube` image only in one layer. The differing layer has a slightly different size, reflecting the size of contents we put in the `/etc/changedfile.txt` file. In `contents_were_kube` on our test system, this layer was 5 bytes, whereas it was 11 bytes in `contents_were_digits`, corresponding to the bytes "`1234567890<EOF>`." Here's the same output from our test system:

```
IMAGE          CREATED        CREATED BY                                      SIZE      COMM
e2a8b024356b   10 minutes ago /bin/bash                                       11B
eeb6ee3f44bd   6 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      6 months ago   /bin/sh -c #(nop)  LABEL org.label-schema.sc...   0B
<missing>      6 months ago   /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4...   204MB
```

39. Let's see how these layers are stored on the filesystem. We'll be exploring `/var/lib/docker/overlay2/`. Let's use `docker inspect` on the `contents_were_digits` image, along with the `jq` (json query) utility. We'll learn a ton about how to use `jq` later in this class.

```
docker inspect contents_were_digits | jq '.[] | .GraphDriver'
```

40. Here's the output from our test system. Note the four paths listed here - you can read more about them in the documentation for overlayfs.

```
{
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/36ab5f1c64f60e31db2554a97e824c48a32d83f77ad9b9221b
    "MergedDir": "/var/lib/docker/overlay2/f07b162a86e9d9ee810fca80561e691caf0e3759a0b2fe12f
    "UpperDir": "/var/lib/docker/overlay2/f07b162a86e9d9ee810fca80561e691caf0e3759a0b2fe12f
    "WorkDir": "/var/lib/docker/overlay2/f07b162a86e9d9ee810fca80561e691caf0e3759a0b2fe12f3
  },
  "Name": "overlay2"
}
```

41. We're going to look at what files are changed in the very top (last-added) layer of the image, which is called `UpperDir`. Let's save that directory name in the $dir variable.

```
dir=$(docker inspect contents_were_digits | jq -r '.[] | .GraphDriver.Data.UpperDir')
```

42. Change your working directory to that one.

```
cd $dir
```

43. List the entire contents of that directory:

```
find
```

44. Note that there's an `etc` directory and just one file: `changedfile.txt`. Here's the output from our test system:

```
.
./etc
./etc/changedfile.txt
```

45. Display the contents of that file:

```
cat etc/changedfile.txt
```

46. Note that the file contains the contents of the `changedfile.txt` file that we changed in each container. Here's the output from our test system:

```
1234567890
```

47. Look at the history for the `contents_were_digits` image one more time with `docker history`:

```
docker history contents_were_digits
```

48. Notice that the `diff/` directory that we just looked at is the upper layer of this image. When you start a container from the image, this `diff` directory's contents are mounted onto the root of the container, on top of whatever files the previous layer contained. Union-mounting allows this upper layer to only contribute/overrule the exact files to the filesystem that this upper layer contains.

49. For extra credit, repeat the process for the `LowerDir` layer. You'll find this command helpful:

```
cd $(docker inspect contents_were_digits | jq -r '.[] | .GraphDriver.Data.LowerDir')
```