Lab Exercise: Working With Docker Images

Preamble

This lab exercise will demonstrate how it's possible to use containers to create Docker images, as well as how to retrieve information related to images located in the Docker host's local repository.

Step 1 – find and retrieve an image

Using the `docker search` command, we will find all Ubuntu images on the Docker Hub, that have at least a 50-star rating. Execute the following command to retrieve a set of images:

```
$ docker search -s=50 ubuntu
```

The command will return something similar to this:

```
NAME             DESCRIPTION                       STARS     OFFICIAL   AUTOMATED
ubuntu           Ubuntu is a Debian-based Li...    2904      [OK]
ubuntu-upstart   Upstart is an event-based r...    57        [OK]
```

How many different tagged images reside in the `library/ubuntu` repository? Use the Docker Hub website to determine this (https://registry.hub.docker.com).

We can pull the latest version of the image down to our local repository, using the `docker pull` command:

```
$ docker pull ubuntu
```

How many image layers are there, and what is the cumulative size of the image?

Step 2 – create a new image from a container

We will use the Ubuntu image as the base for creating an image for the NoSQL database server, MongoDB, which we will call `jbloggs/mongo`. First we need to update the package lists supplied in the Ubuntu image, before installing the `curl` utility to enable us to download MongoDB, by executing the following command:

```
$ docker run ubuntu sh -c 'apt-get update'
```

We can create our image by committing this initial change with the following command:

```
      $ docker commit $(docker ps -lq) jbloggs/mongo
```

Use the `docker images` command to make sure the image has been created.

Now we'll install the `curl` utility, and update our image accordingly:

```
      $ docker run jbloggs/mongo sh -c 'apt-get install -y curl'
      $ docker commit $(docker ps -lq) jbloggs/mongo
```

Next, we can use the `curl` utility to download MongoDB and commit the changes to our image, using the following commands:

```
      $ docker run jbloggs/mongo sh -c 'curl -L \
      https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.0.8.tgz \
      -o /tmp/mongo.tgz'
      $ docker commit $(docker ps -lq) jbloggs/mongo
```

The archive needs to be extracted to a preferred location, using the following:

```
      $ docker run jbloggs/mongo sh -c 'tar -xvf /tmp/mongo.tgz -C \
      /usr/local --strip-components=1'
      $ docker commit $(docker ps -lq) jbloggs/mongo
```

Now that MongoDB has been installed, we can do some tidying up, by removing the redundant `curl` utility and its dependent packages, and the MongoDB archive:

```
      $ docker run jbloggs/mongo sh -c 'apt-get remove -y curl && \
      apt-get autoremove -y && apt-get clean autoclean && \
      rm -rfv /var/lib/apt/lists/* /tmp/*'
      $ docker commit -a 'Joe Bloggs' $(docker ps -lq) jbloggs/mongo
```

Notice that we have chosen to specify the author of our image during this commit.

How big is the new image that we've just created, and how big are the composite layers?

Check that Joe Bloggs has been recorded as the author in the image's metadata, by executing the following command:

```
      $ docker inspect -f '{{.Author}}' jbloggs/mongo
```

Step 3 – flatten an image

In creating our `jbloggs/mongo` image, as well as downloading the MongoDB archive, and expanding it, we also installed the `curl` utility. We removed `curl` and the MongoDB archive, but these remain in the image, so we will now perform a `docker export` and `docker import` in order to flatten the image.

First we will export the last container's filesystem:

```
$ docker export $(docker ps -lq) | gzip -c > mongo.tgz
```

Having successfully exported the container's filesystem, we can now import it to see what effect the export has had on re-establishing the image:

```
cat mongo.tgz | docker import - jbloggs/mongo
```

How big is the flattened image? How many layers does it have now?

Remove all of the redundant containers – try and do this with a single, compound command.

Step 4 – tag an image

If we now execute the `docker images` command, we can see that the unflattened image that we exported is now a dangling image. We can return this image to our repository, but we need to give it a tag in order to differentiate it from the `jbloggs/mongo:latest` image we created from our import. Execute the following command to tag our original image as `jbloggs/mongo:original` (replacing `<image>` with the ID of the dangling image):

```
docker tag <image> jbloggs/mongo:original
```