Part 1 - Docker introduction lab

Prerequisites

This set of instructions requires that docker is already installed and docker commands can be run from a bash shell. You can get more information at the Docker website.

Note: This demo assumes that you are running in a "clean" environment. Clean means that you have not used docker with the images in this demo. This is important for someone who is using docker for the first time, so they can see the activity as images are downloaded.

Working with docker

1. Launch a shell and confirm that docker is installed. The version number isn't particularly important.

```
$ docker -v
Docker version 17.06.1-ce, build 874a737
```

2. As with all new computer things, it is obligatory that we start with "hello-world"

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b04784fba78d: Pull complete
Digest:
sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26f
f74f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the
Docker Hub.
 3. The Docker daemon created a new container from that image which
runs the
```

```
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client,
which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container
with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide
```

Notice the message `Unable to find image 'hello-world:latest' locally` First, you see that the image was automatically downloaded without any additional commands. Second, the version `:latest` was added to the name of the image. We did not specify a version for this image.

3. Rerun "hello-world" Notice that the image is not pulled down again. It already exists locally, so it is run.

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working
correctly.

      [output truncated]
```

4. It already exists locally and `docker images` will show us that image.

```
$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
hello-world         latest    1815c82652c0   2 months ago    1.84kB
```

5. From where was the `hello-world` image pulled? Go to `https://hub.docker.com/_/hello-world/` (that is an underscore `_` character in the URL:) and you can read about this image. Docker-hub is a repository that holds docker images for use. Docker-hub is not the only repository, IBM Cloud can serve as a docker repository.

6. This image is atypical. When an image is run, it usually continues to

run. The running image is called a container. Let us run a more typical image; this image contains the noSQL database "couchDB".

```
$ docker run -d couchdb
Unable to find image 'couchdb:latest' locally
latest: Pulling from library/couchdb
ad74af05f5a2: Downloading [==>                        ]
2.702MB/52.61MB
ffdd0c835430: Download complete
d922980c187f: Downloading [===>                       ]
2.661MB/43.77MB
affbf57fdbcf: Verifying Checksum
0ddcd7e9244b: Download complete
34473f480310: Downloading [=============>           ]
2.26MB/8.236MB
78a52d457cb5: Waiting
```

The output above was captured while the image was still downloading from docker-hub. When the download is down you don't see anything from the container, like with hello-world. Instead you see a long hex id like `2169c6b42e5c590229c5c86f5ed3596b1b56c2366378914b082e5b000752bd34`. This is the id of the container.

7. Here's how you would see the running container. Notice only the first part of that long hex id is displayed. Typically this is more than enough to uniquely identify that container. `docker ps` provides information about when the container was created, how long it has been running, then name of the image as well as the name of the container. Note that each container must have a unique name. You can specify a name for each container as long as it is unique.

```
$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
2169c6b42e5c  couchdb   "tini -- /docker-e..."   8 minutes ago   Up
8 minutes  5984/tcp  nervous_poincare
```

8. An image can be run multiple times. Launch another container for the couchdb image.

```
$ docker run -d couchdb
f9885aaf0a96742119462208dce611018ab2104737adf3485d6fc4e7642b104b
```

9. Now we have two containers running the couchdb database. Did you notice how quickly the second instance started? There was no need to download the image this time. The id of the container is show after is has started.

```
$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED         STATUS        PORTS
NAMES
f9885aaf0a96  couchdb   "tini -- /docker-e..."   2 minutes ago   Up 2 minutes  5984/tcp
brave_booth
2169c6b42e5c  couchdb   "tini -- /docker-e..."   22 minutes ago  Up 22 minutes 5984/tcp
nervous_poincare
```

9. The containers look similar, but they have unique names and unique ids. Stop the most recent container and then check to see what's running.

```
$ docker stop f9885aaf0a96
f9885aaf0a96

$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
2169c6b42e5c  couchdb   "tini -- /docker-e..."   25 minutes ago  Up
25 minutes 5984/tcp  nervous_poincare
```

10. Stop the other container and see what is running.

```
$ docker stop 2169c6b42e5c
2169c6b42e5c

$ docker ps
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS        PORTS     NAMES
```

11. Notice the image still exists.

```
$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
couchdb           latest    7f8923b03b7f   5 weeks ago     225MB
hello-world       latest    1815c82652c0   2 months ago   1.84kB
```

11. Did you forget about the hello-world image? Go ahead and delete the couchdb image and double check that it is gone.

```
docker rmi couchdb
Error response from daemon: conflict: unable to remove repository reference "couchdb"
(must force) - container 2169c6b42e5c is using its referenced image 7f8923b03b7f
```

12. Oops, we can't delete that image until we delete the "couchdb" container. Note the `docker ps -a` will show us all the containers, not just the ones that are running.

```
$ docker ps -a
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS                     NAMES
f9885aaf0a96  couchdb     "tini -- /docker-e..." 22 minutes ago
Exited 2 minutes ago     brave_booth
9cd02a3f2eec2 couchdb     "tini -- /docker-e..." 22 minutes ago
Exited 4 minutes ago     silly_ritchie
b44146cfad65  hello-world "/hello"               An hour ago
Exited About an hour ago elated_engelbart
8d71894c865b  hello-world "/hello"               2 hours ago
Exited 2 hours ago       stoic_lamport
```

13. Delete the couchdb containers, delete the couchdb image, and make sure it is gone. You can leave hello-world.

```
$ docker rm f9885aaf0a96
f9885aaf0a96

$ docker rmi couchdb
Untagged: couchdb:latest
Untagged:
couchdb@sha256:eb463cca23b9e9370afbd84ae1d21c0274292aabd11b2e5b904d
4be2899141ff
Deleted:
sha256:7f8923b03b7f807ffbd51ff902db3b5d2e2bbbc440d72bc81969c6b05631
7c8a
Deleted:
sha256:d53bc50464e197cbe1358f44ab6d926d4df2b6b3742d64640a2523e46401
04c4
Deleted:
sha256:851748835e9443fa6b8d84fbfada336dffd1ba851a7ed51a0152de3e3115
b693
Deleted:
sha256:feb87fb4c017e2d01b5d22dee3f23db2b3f06a0111a941dda926139edc02
```

```
7c8e
Deleted:
sha256:e00c9e10766f6a4d24eff37b5a1000a7b41c501f4551a4790348b94ff179
ca53
Deleted:
sha256:b64ffebe7ca9cec184d6224d4546ccdfecce6ddf7f5429f8e82693a8372c
f599
Deleted:
sha256:f78934f92a8a0c822d4fc9e16a6785dc815486e060f60b876d2c4df19255
84d8
Deleted:
sha256:491c6d0078fa4421d05690c79ffa4baf3cdeb5ead60c151ab64af4fb6d4d
93dc
Deleted:
sha256:2c40c66f7667aefbb18f7070cf52fae7abbe9b66e49b4e1fd740544e7cea
ebdc
```

```
$ docker ps -a
CONTAINER ID  IMAGE     COMMAND                  CREATED
STATUS                     NAMES
b44146cfad65  hello-world "/hello"               An hour ago
Exited About an hour ago elated_engelbart
8d71894c865b  hello-world "/hello"               2 hours ago
Exited 2 hours ago       stoic_lamport

$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
hello-world       latest    1815c82652c0   2 months ago   1.84kB
```

Note: Docker images and containers can be referenced by name or by id. This demonstration has shown only the rudimentary capabilities of docker.

Part 2 - WebSphere Liberty Docker lab

WebSphere Liberty running in docker

1. Launch a shell and confirm that docker is installed. The version number isn't particularly important.

```
$ docker -v
Docker version 17.06.1-ce, build 874a737
```

2. Pull the latest docker image for WebSphere Liberty. This may take a few

minutes to complete. You will notice that docker will download all of the layers of images that the Liberty image is built upon.

```
$ docker pull websphere-liberty
Using default tag: latest
latest: Pulling from library/websphere-liberty
  [output removed]

Digest:
sha256:60c00ecb8b3f74f7a2a4533bad57e9684a2df8f957919f332909bd55b248
4817
Status: Downloaded newer image for websphere-liberty:latest
```

3. When the image download has completed, you can verify that it has been downloaded by using the command `docker images`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
websphere-liberty   latest              01128080ee00        3 weeks
ago         437MB
```

4. You can also verify that no other images (containers) are running with the `docker ps` command.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
```

5. One thing you should look at in this demo is how little memory and CPU are required to run docker images. In this example you'll use `uptime` to look at system load to get a general idea how hard the CPU is working. Take a look at the load averages below. It isn't really important that you understand exactly what these numbers mean. This article explains what the load averages mean. From left to right the load average numbers are for 1, 5, and 15 minutes. What you will be checking is what the load averages look like before and after running multiple docker containers. You should write down the load averages, because one of the steps will clear your screen. Notice in the output here, that the one minute average load is slightly above three, and the five minute and fifteen minutes averages are a little less than three.

```
$ uptime
11:33  up 6 days, 20:24, 3 users, load averages: 3.28 2.91 2.83
```

6. In this step you will start up multiple instances of the WebSphere Liberty docker image. This command relies on having a bash shell to run. You will be launching five separate instances of the container. The liberty server will listen on port 9080. The `-p $i:9080` maps this port to another port. The first image will listen on port 80, the second on port 81, and so on. You can change these ports if they have a conflict on your machine. As each container is started, the unique ID of that container is returned. Notice that each container starts rather quickly, another good feature of docker containers.

```
$ for i in 80 81 82 83 84; do docker run -d -p $i:9080  websphere-
liberty:webProfile7 ; done
webProfile7: Pulling from library/websphere-liberty
d5c6f90da05d: Already exists
1300883d87d5: Already exists
c220aa3cfc1b: Already exists
2e9398f099dc: Already exists
dc27a084064f: Already exists
155fe9cd6124: Already exists
974b2337a80b: Already exists
0ad69ad38c5e: Already exists
21f9c31bf2e9: Already exists
453240fee003: Already exists
a2c82cb1af29: Already exists
c5ae97216ae8: Already exists
c941b70b6812: Already exists
b3b2397ccd01: Already exists
Digest:
sha256:dca823c618a7d4a481eeee7acd59b5bdae5d5775ee69015a76aa53df1580
ce28
Status: Downloaded newer image for websphere-liberty:webProfile7
4fbf4b2d5440491fcd5b04ee8c8eec2a478a5db7913dea2e7715078182443c12
be78761830eacd254c0a252942b9b801381281af9deb9a1ef4f65b3d35756b07
7ce18cb57aef7b4c6b38e2c006300c5d008889bcefd53d616f60541c5bfe0c16
c74159638e585207857d08f4d093757e786835606653331472c7731a396f410f
251ed1f67a23b67a3893b3880d13fc622c3a307d14c6d91c6847b2d33658bc34
```

7. Docker provides the `docker stats` command so that you can see the resources used by each docker instance. Run this command quickly after all of the containers are launched so you can see the CPU and memory spike when the containers are first started. Notice that after a short period of time, the CPU and memory use drops significantly to low

usage. This should give you a better feeling for the efficiency of docker containers. The `docker stats` command loops continuously, so you will need to stop it. -c

```
$ docker stats
CONTAINER         CPU %       MEM USAGE / LIMIT     MEM %      NET
I/O        BLOCK I/O        PIDS
4fbf4b2d5440      68.96%      49.14MiB / 1.952GiB   2.46%      578B
/ 0B      0B / 766kB       35
be78761830ea      64.32%      48.07MiB / 1.952GiB   2.41%      578B
/ 0B      0B / 643kB       36
7ce18cb57aef      82.19%      62.59MiB / 1.952GiB   3.13%      578B
/ 0B      0B / 1.28MB      36
c74159638e58      86.43%      67.93MiB / 1.952GiB   3.40%      712B
/ 0B      0B / 2.1MB       37
251ed1f67a23      88.11%      63.91MiB / 1.952GiB   3.20%      822B
/ 0B      123kB / 2.1MB    38

CONTAINER         CPU %       MEM USAGE / LIMIT     MEM %      NET
I/O        BLOCK I/O        PIDS
4fbf4b2d5440      0.43%       114.1MiB / 1.952GiB   5.71%      986B
/ 0B      4.1kB / 6.8MB    42
be78761830ea      0.59%       106.2MiB / 1.952GiB   5.31%
1.06kB / 0B    0B / 8.59MB      45
7ce18cb57aef      0.98%       112.3MiB / 1.952GiB   5.62%
1.06kB / 0B    0B / 8.46MB      42
c74159638e58      0.52%       107.2MiB / 1.952GiB   5.36%
1.12kB / 0B    0B / 7.74MB      42
251ed1f67a23      0.39%       102.5MiB / 1.952GiB   5.13%
1.3kB / 0B     123kB / 7MB      43
```

8. You can rerun `uptime` to compare the load before there were five docker instances running. Notice in this example there is virtually no difference in the load.

```
$ uptime
11:37  up 6 days, 20:30, 3 users, load averages: 2.71 3.01 2.96
```

9. Open a browser and go to `http://localhost` you will see the liberty welcome page. You should also verify that `http://localhost:81` loads as well as ports 82, 83, and 84.

10. You can see the running containers with the `docker ps` command. This will also give you the container ID for each container.

```
$ docker ps
CONTAINER ID    IMAGE                              COMMAND
CREATED
4fbf4b2d5440    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
be78761830ea    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
7ce18cb57aef    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
c74159638e58    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
251ed1f67a23    websphere-liberty:webProfile7
"/opt/ibm/docker/d..."   2 hours ago
```

11. Before moving on you should stop the five Liberty instances. You can use the `docker stop` command with all five of the container IDs on the same line. As each container is stopped, that ID is returned.

```
$ docker stop 4fbf4b2d5440 be78761830ea 7ce18cb57aef c74159638e58
251ed1f67a23
4fbf4b2d5440
be78761830ea
7ce18cb57aef
c74159638e58
251ed1f67a23
```