Docker Hands-on Guide - Basic Commands

Now that you have installed Docker Toolbox, it is time to try out the basic commands that you can do via the docker client program.

Recap

Recollect that the Docker Toolbox consists of several tools but the main ones for our hands-on exercise here are:
1) Docker Machine
2) Docker Client
3) Docker Hub

Before you begin, ensure that you have the following:
1) Atleast one Docker Machine created. Use the docker-machine ls command to check that.
2) Ensure that the Docker Machine is started. Use docker-machine status command to check that. You should start the Docker Machine via the docker-machine start command.
3) Point your Docker client to the particular target Docker Machine. Use the following command :

   docker-machine env --shell cmd <target-machine-name>

   Then follow the set of instructions to set the Docker ENVIRONMENT variables so that your Docker client is pointing to that.

4) Ensure that your Docker client is able to communicate to the target Docker machine by giving the docker <commands>. For e.g. docker images.

   Now, when we work with the docker client, what is happening is that the commands are being sent to the Docker Engine running inside the Docker Machine, which then interprets the command and executes it for you.

docker client help

The docker client can understand several commands. And in this hands-on step, we are going to take a look at various commands that you will primary use while running docker.

To get help at any point in time try out the following command:

```
$ docker help
```

This will give you a full listing of commands that the docker client can understand. Take some time to go through this. Most of the commands are self-explanatory and are typical ones that you will use while dealing with containers.

At any point in time, if you need more help on any COMMAND, you can get that via the following:

```
$ docker COMMAND --help
```

Initial List of commands

The next few sections will take you through various commands and you should try out every single command. Before you try any of that, ensure that **boot2docker** VM has been started and you are inside the $ bash shell of the VM.

```
$ docker version
```

This will show you the current docker version.

Fun fact : Docker is written in [Go language](). The language is similar to C and has been the favorite of developers writing infrastructure software. Pick it up now !

```
$ docker info
```

This will show you several pieces of information about the OS.

Run a few Unix Commands/Utilities

Let's understand what we are doing here now. We are on a Windows machine and we wish to run a few Unix commands/utilities to get a bit familiar with them.

So this is what the steps look like with Docker now:

1. There is a useful Docker Image called **busybox** (just like we had hello-world) that someone has already created for Docker.
2. We will use the **docker run** command to run a container i.e. create an instance of that image.
3. By running, what we want to do is to go inside that container and run a few commands there.

Let us check some steps i.e. docker commands that we will run - not all are necessary but we are doing this to get you a bit familiar with the commands. We will be looking at some of these commands in more detail in subsequent sessions.

```
$ docker search busybox
```

This command will search the online Docker registry for a Image named *busybox*. On my machine, the output shown is as follows (only the top few rows are shown):

```
C:\Users\irani_r>docker search busybox
NAME                         DESCRIPTION                           STARS     OFFICIAL   AUTOMATED
busybox                      Busybox base image.                   360       [OK]
progrium/busybox                                                   51                   [OK]
radial/busyboxplus           Full-chain, Internet enabled, busybox made...  8          [OK]
azukiapp/busybox             This image is meant to be used as the base...  2          [OK]
odise/busybox-python                                               2                    [OK]
ofayau/busybox-libc32        Busybox with 32 bits (and 64 bits) libs        1          [OK]
skomma/busybox-data          Docker image suitable for data volume cont...  1          [OK]
socketplane/busybox                                               1                    [OK]
odise/busybox-curl                                               1                    [OK]
peelsky/zulu-openjdk-busybox                                     1                    [OK]
```

Let us understand the output here, by paying attention to the columns:
1) The first column is NAME and it gives you the name of the Docker image.
2) The second column is DESCRIPTION and it is obvious what that means.
3) The next column is STARS and if you noticed the list of images that matched the search term Docker have been listed in the descending order of the number of people who have starred the project. This is a very useful indicator of the popularity/correctness of the Image. Often if confused among which Docker image to go with, I usually pick the one with the most STARS.

The first column, to reiterate, was the NAME of the Docker image. This is a unique name and you must use this name for some of the commands given below.

So, let's say that we are fine with the **busybox** image name and now want to create an instance (Container) of this image. To do that, all we need to do is use the **docker run** command as given below:
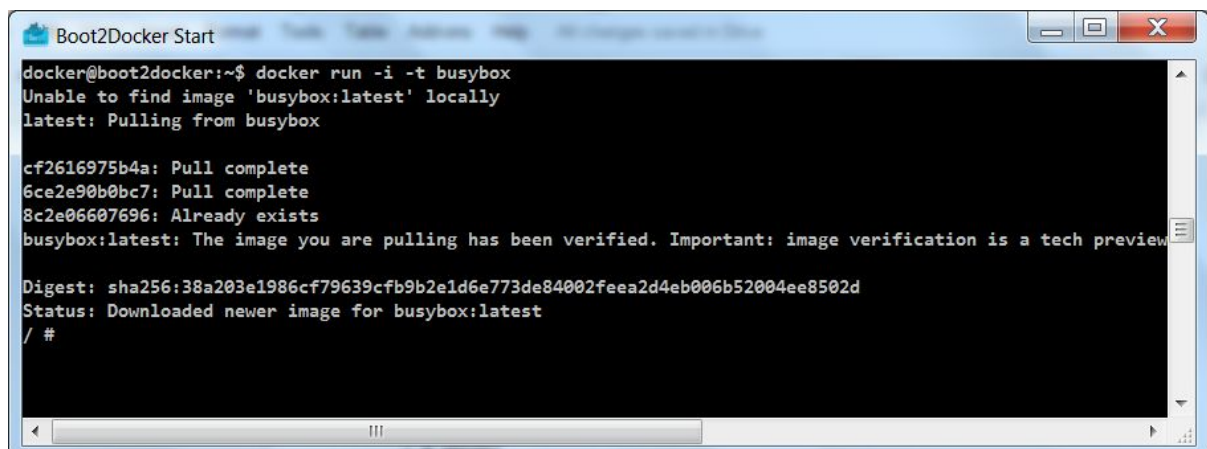
```
$ docker run -t -i busybox
```

The run command does something interesting and this will help you understand the Docker architecture, which we have seen earlier. The run command does the following:

1) It checks if you already have a busybox image in your local repository.
2) If it does not find that (which will be the case first time), it will pull the image from the Docker hub. Pulling the image is similar to downloading it and it could take a while to do that depending on your internet connection.
3) Once it is pulled successfully, it is present in your local repository and hence it is then able to create a container based on this image.
4) We have provided -i -t as the parameters to the run command and this means that it is interactive and attaches the tty input.

Note: If the image was present locally, it would have directly run the container for you.

On successful launch of the container, you will be led into the bash shell for busybox. To keep it simply for Windows users, we are now logged into the buxybox container and are at the command prompt, as shown below:
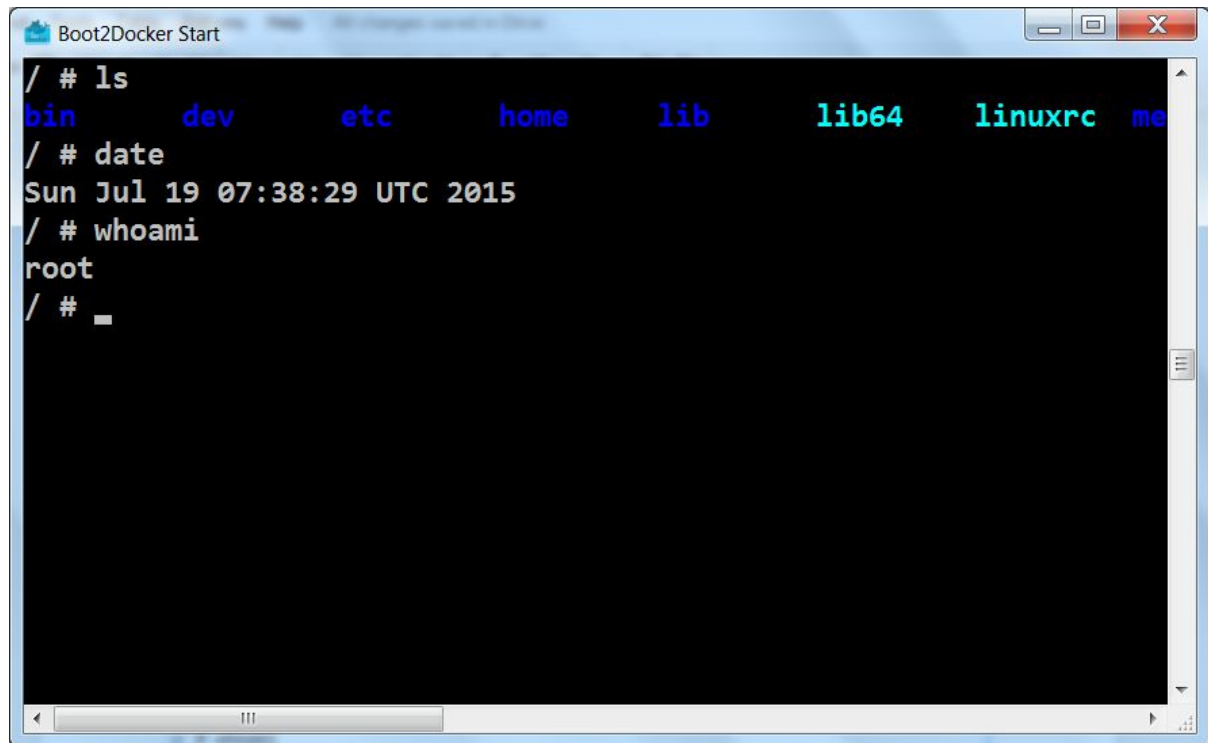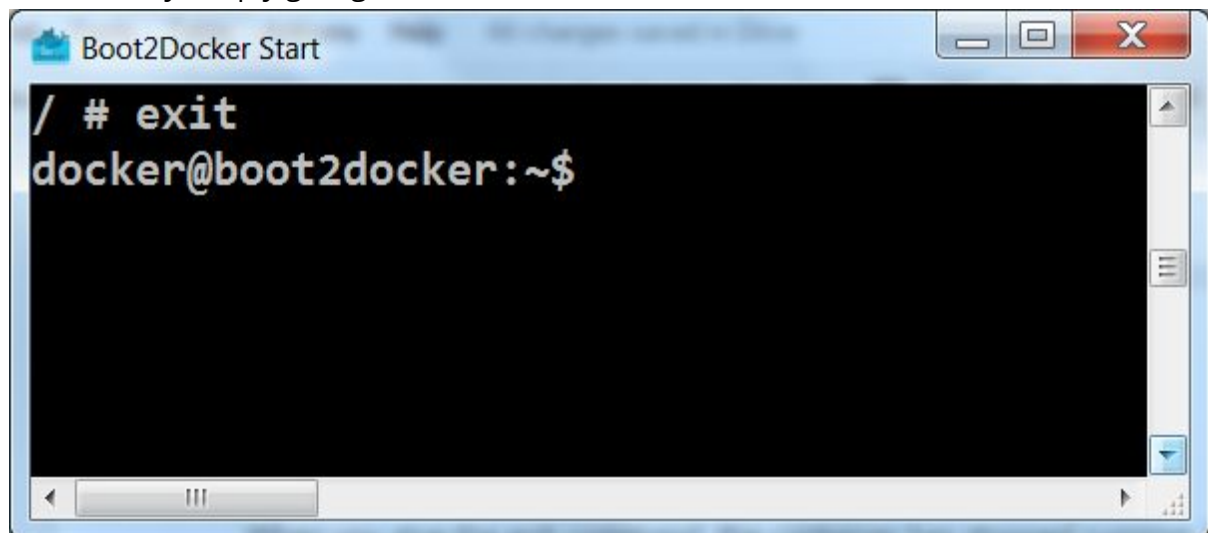
You can run a few commands as shown below:
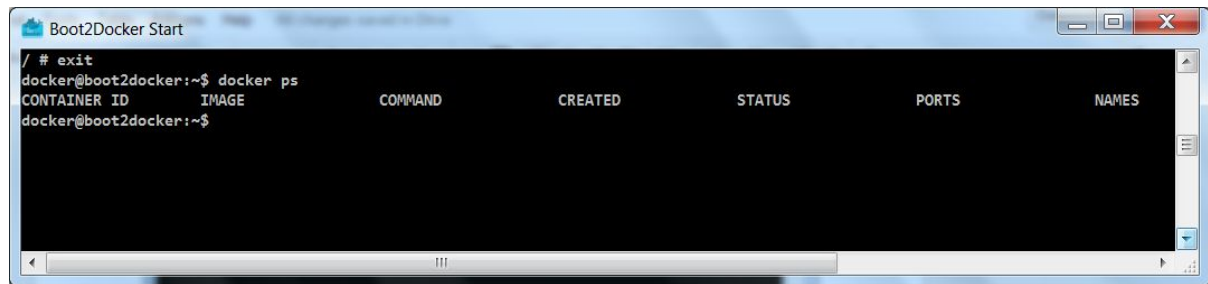
Note from the prompt that you are now inside the container. You can exit the container by simply giving the **exit** command.

When you give the exit command, the container has stopped running. To verify that, you can give another command as given below:

$ docker ps

This gives you a list of all the running containers. You will notice from the output that there are no container running.

<u>Try out the following:</u>
If you want to find out the containers that were running earlier but are not in the terminated state, you can use the -all flag for the `docker ps` command. Give it a try.

Get List of Docker Images

At this point in time, if you want to know what images are already present on your docker setup locally, try the following command:

```
$ docker images
```

You will find that it has the **busybox** image listed.

Note the columns that the output gives (2 important ones are given below):
1) REPOSITORY
2) TAG

The REPOSITORY column is obvious since it is the name of the Image itself. The TAG is important, you will find that the TAG value is mentioned as **latest**. But there was no indication given by us about that.

The fact is that when we gave the following command earlier:
```
$ docker run -i -t busybox
```

We only specified the name and by default if just the IMAGE name is specified , then it gets the latest image by default. The tag value 'latest' is sort of implicitly used by the Docker client in the absence of an explicit tag value provided by you.

In other words, you could have specified it as:

```
$ docker run -i -t busybox:latest
```

Similarly, there is a clear possibility that there will be multiple versions of any image present in the Docker Hub. We will see all that in a while, but for now, keep in mind that lets say there were the following versions available of busybox:

1) Image Name : busybox , Version TAG : 1.0
2) Image Name : busybox, Version TAG : 2.0
3) Image Name : busybox, Version TAG : 3.0
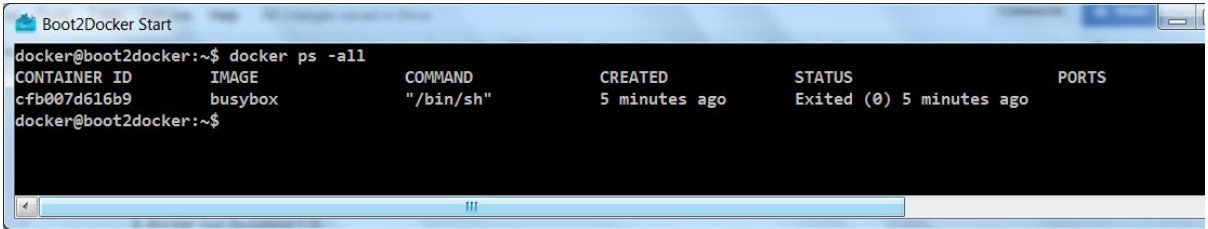
We could mention the version TAG as needed:

$ docker run busybox:1.0
$ docker run busybox:2.0

and so on.

Docker Containers

When you executed the docker ps command, you noticed that no container was running. This was because you exited out of the container. Which means that the container only exists as long as its parent process is running.

Now, let us do a docker ps -all command. This should show you the container that was launched a few minutes ago by you. For e.g. on my system, I see the following:
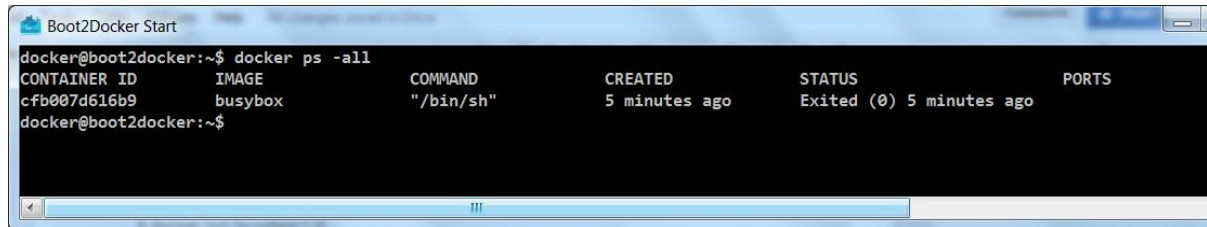

Notice the following columns:
- CONTAINER_ID : A Unique ID for the container that was launched.
- IMAGE : This was the IMAGE that you launched i.e. busybox
- COMMAND : Important stuff here. This was the default command that was executed when the container was launched. If you recollect, when the container based on busybox image was launched, it led you to the Unix Prompt i.e. the Shell was launched. And that is exactly what the program in /bin/sh does. This should give you a hint that in case you want to package your own Server in a Docker image, your default command here would typically be the command to launch the Server and put it in a listening mode. Getting it ?

Relaunch a Container

To start a stopped container, you can use the docker start command. All you need to do is give the Container ID to the docker start command.

So, look at the docker ps -all output and note down the CONTAINER_ID. On my machine, the docker ps -all command gives me the following output:

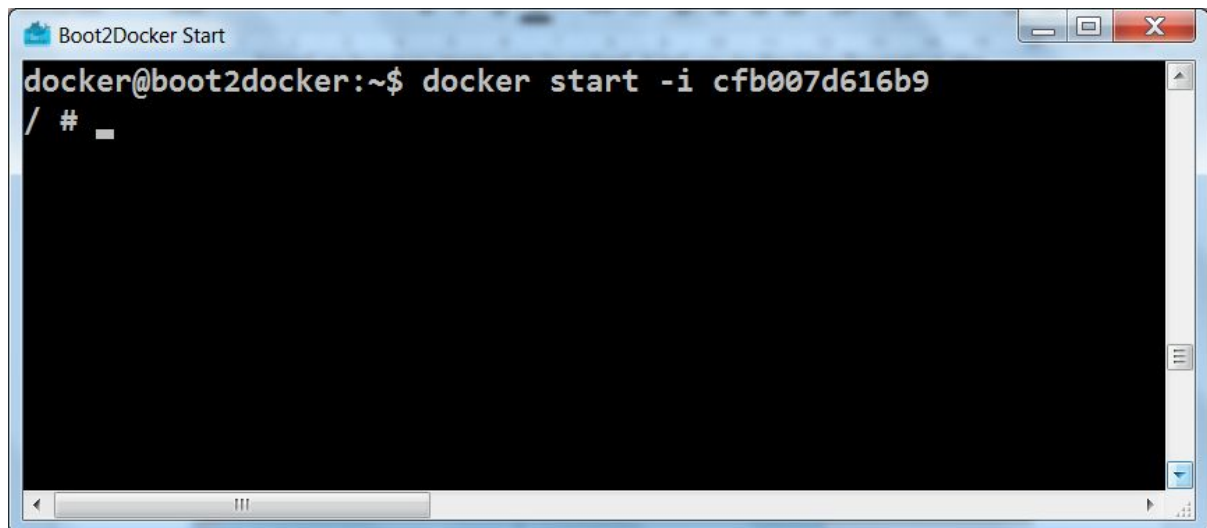I note down the CONTAINER_ID i.e. cfb007d616b9 and then give the following command:

```
$ docker start -i cfb007d616b9
```

Note the -i for going into interactive mode. You will find that if everything went fine, the Container was restarted and you were back again at the Prompt, as given below:

Type exit and come out of the container. We are back to where we were with no containers running.

Attach to a running Container

Now, its time for something interesting to help us understand some more commands. We will continue with our example around busybox Image.

First up, we will relaunch our container **without the -i (interactive) mode**.

Give the following command:
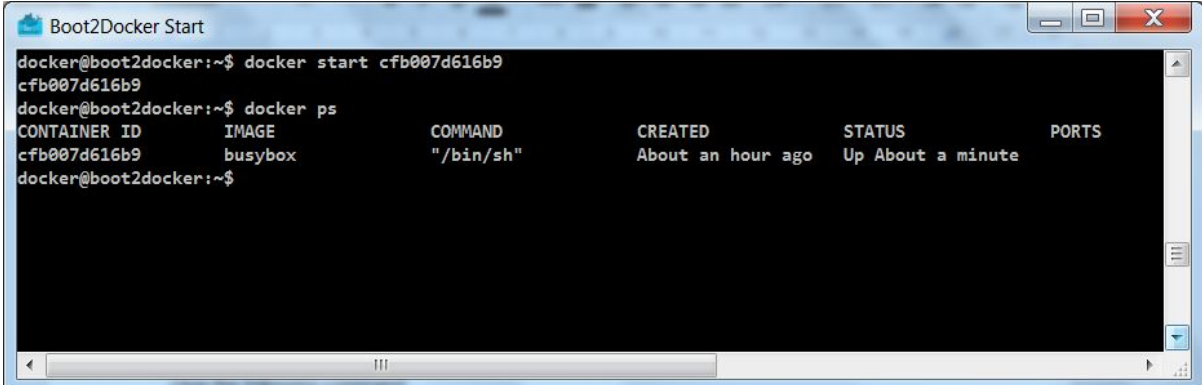
```
$ docker start cfb007d616b9
cfb007d616b9
```

Whoops ! What happened ?

The only output that we got was the CONTAINER ID back.

What has just happened is that the Container has got launched and all that docker client has done is give you the Container ID back.

Give the following command to check out the current running containers (the command should be familiar to you now):

```
$ docker ps
```


This gives us the output that the CONTAINER is running (Check the STATUS column. You will find that it says it is Up!)

We can attach to a running Container via the docker attach command. Let us attach to it:

```
$ docker attach cfb007d616b9
```
This will get you back to the Prompt i.e. you are now inside the busybox container. Type `exit` to exit the container and then try the `docker ps` command. There will be no running containers.

Note: If you want to stop a running container, you can give the `docker stop <ContainerId>` command. Try it.

Tip: You need not always give the full CONTAINER_ID value. Type a few letters from the start of the CONTAINER_ID and it should work. Neat, isn't it ?

Exercise for the Reader:

1. Try out some other images from the official Docker registry. For e.g. how about running Ubuntu in your Windows machine. Go ahead and try it. Just give the following command : `$ docker -i -t run ubuntu`
2. You might be wondering what happens if you use run without the -i -t. Give it a try.