

Exercise – Hello Docker!

Objective

In this exercise we will look at installing and creating some Docker containers from the Docker hub.

Installing Docker

For Docker we will need another virtual machine running in aws. Create one with the following options:

- Image: Amazon linux
- Type: t2.medium
- Storage: 20 gig
- Tag: Name = Docker
- Security group: Open all tcp ports for access from anywhere

Connect to your server using PuTTY and your key as usual.

Installing docker is quite easy as it is included in the package manager for this version of linux. To install and start the service type:

```
$ sudo bash
$ yum install -y docker
$ service docker start
$ sudo chkconfig docker on
```

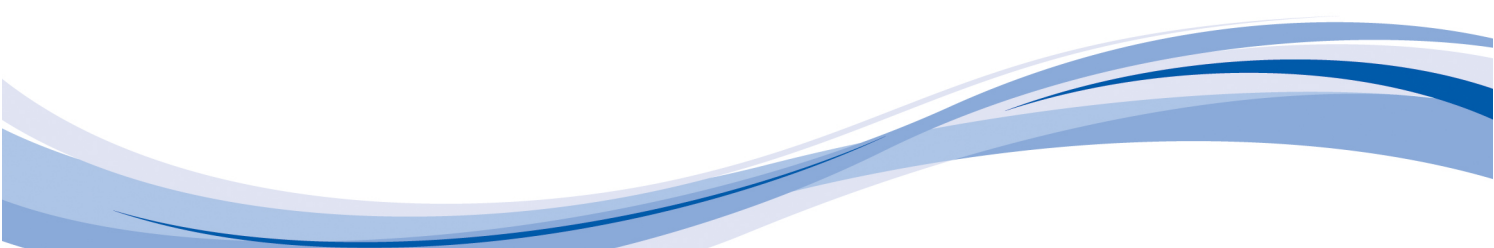
Part 2 - Hello World!

Every docker command needs to be run as the super user. So we can either put 'sudo' in front of every command, or we can switch to super user mode with 'sudo bash'.

Docker works by running commands inside containers. These are not the same as virtual machines. You can't turn on a container and leave it running, but you can start processes within them. The Docker hub is full of different containers that Docker and other people have provided which run different services on different operating systems. run the following:

```
$ docker run ubuntu:latest /bin/echo 'Hello World'
```

The format of the run command is always:



```
$ docker run [flags] [image:tag] [command] [parameters]
```

If we break this up into the different parts we can see what each statement is doing:

Docker	The Docker binary
run	We want to run a container. This will look locally first to see if we have the right container, if we don't it will head off to the repository and pull it down for us
ubuntu:latest	The name of the container we want to run "ubuntu" and the tag for it "latest". User defined docker containers have the format: "username/containername:tag"
/bin/echo	The command we want to run inside the container. We don't need to use the fully qualified name.
'Hello world'	Parameters being passed to the echo command

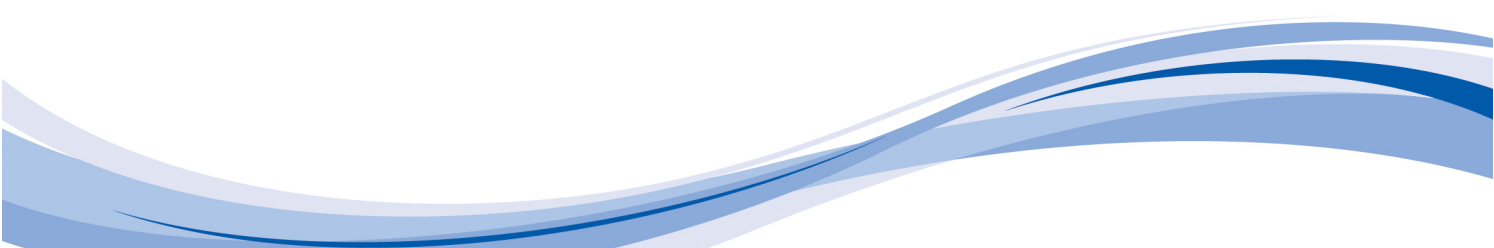
Output:

```
[on host]$ docker run ubuntu:latest /bin/echo 'Hello World!'
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu

d54efb8db41d: Pull complete
f8b845f45a87: Pull complete
e8db7bf7c39f: Pull complete
9654c40e9079: Pull complete
6d9ef359eaaa: Pull complete
Digest: sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d35
Status: Downloaded newer image for ubuntu:latest

Hello World
```

If you don't give docker a tag it will look for an image tagged "latest" by default. It has pulled down the image from the Docker hub including all the layers that make up this image. Now if we want to use the image again it is stored locally, so it will be a lot quicker.



```
[on host]$ docker run ubuntu echo 'hello world'
hello world
```

You can see the list of docker images available on your machine by using

```
[on host]$ docker images
```

Part 3 - Connecting a terminal

We can connect directly to a docker container by using the following command:

```
[on host]$ docker run -t -i ubuntu:latest /bin/bash
```

This will get the ubuntu image requested and run the /bin/bash command in the container. We have used two flags.

- -t Open a pseudo terminal to the container that we can use
- -i Start the session in interactive mode. By doing this we can type commands directly into the terminal and see their effect.

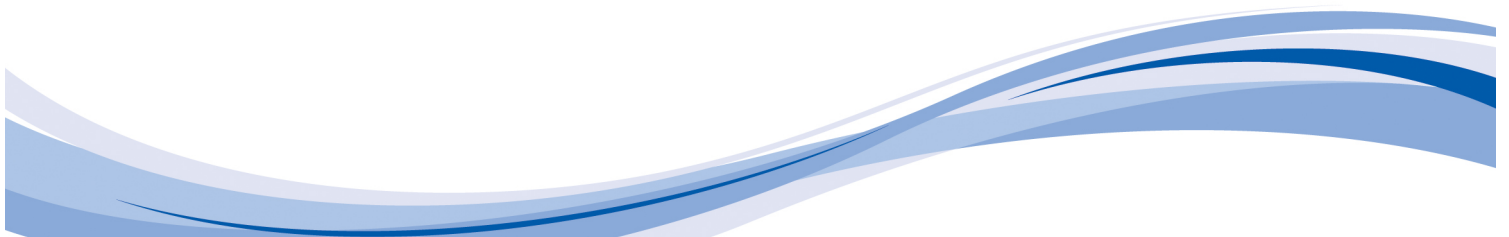
These can be combined as -ti

```
[on host]$ docker run -ti ubuntu /bin/bash
root@15b014b00f80:/$ echo 'hello world'
hello world
root@15b014b00f80:/$
```

The number after the @ symbol is the container ID that has been assigned by Docker.

Exit the container

```
root@15b014b00f80:/$ exit
exit
```



Part 4 - Pulling down pre-made containers

The Docker hub has thousands of public containers available for use. You will need to sign up for an account if you want to push your own images to the hub at <https://hub.docker.com>

You can search for available images via the command line tool as well as looking on website.

```
[on host]$ docker search jenkins
or
[on host]$ docker search hello-springboot
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
gazgeek/springboot-helloworld	0	[OK]	kizzie/hello-springboot	0

There will be an image called **kizzie/hello-springboot**, pull down that image and run it with the following:

```
[on host]$ docker pull kizzie/hello-springboot
[on host]$ docker run -d -P kizzie/hello-springboot
```

This pull down the image for hello-springboot and then run the container. We have two new flags here.

- **-d** Run as detached mode. This will run the Docker container the background allowing us to use the terminal for other things.
- **-P** Opens the ports that are exposed in the container, it will randomly assign an host port that is being forwarded to the container. If you want to define this yourself then use a non-capitalised p, such as `-p 8080:8080` forwards port 8080 on the host to port 8080 in the container.

It returns a very long hash code, this is the current container ID that has been generated. We can see all the information about the containers running with

```
[on host]$ docker ps
```

Find the line with the springboot container running and look for the ports shown. It should be of the form:

```
[on host]$ docker run -d -P kizzie/hello-springboot
cc56701ce14cb6f339bb60d4e88421dd996965e6f7a41bae0b3cea3fe4b2bd
[on host]$ docker ps
cc56701ce14c      kizzie/hello-springboot:latest      "/bin/sh
-c 'mvn spr      3 seconds ago      Up 1 seconds
0.0.0.0:49156->8080/tcp      sad_poitras
```

Port **49156** on the host machine has been forwarded to port **8080** on the container.

There is some delay between the container starting and the web address responding. We can have a look at what is going on inside the container with the logs command.

```
[on host]$ docker logs -f sad_poitras
```

The final part of the command, sad_poitras, is the name given to the container by docker. It will randomly assign names unless you specify one with the `--name` flag on the run command. (For example: `docker run -d -P --name springboot kizzie/hello-springboot`).

If you don't want to use the name when referring to the container then you can use the first part of the container id. For example "`docker logs cc56`" will pick up the correct container for use.

The `-f` flag tells docker we want to keep the connection open and see what is happening. When you see the line ending:

```
"Started HelloSpringbootApplication in 5.097 seconds (JVM running for 36.092) "
```

Then you should be able to access the website via the browser. It is running a tomcat server so if you go to the server root you should see the tomcat starter page. To see the hello-scalatra project go to the webaddress:

```
http://<your ip>:<your port>
```

It should say hello world!

Have a look at the docker hub for other images and try some out!

