



Romin Irani

[Follow](#)

My passion is to help developers succeed. ッ(ツ)ッ

Jul 19, 2015 · 8 min read

Docker Tutorial Series—Part 3—More on Images and Containers

This is part 3 of the [Docker Tutorial Series](#).

In this section, we shall get acquainted with various other Docker commands and parameters that will give you a good grasp of how to deal with Images, running containers, container networking to some extent and more.

To learn that, it is best to dive deep into running one of the popular servers of all time , the Apache Web Server and working with its Dockerized Image from the Docker Hub.

Running Apache Web Server

Now, let's try another experiment to help you understand how you will eventually envision your applications running inside Docker.

The title of this section indicates that we want to run the Apache Web Server. To do that, you should now start thinking that there must be a Docker image that someone must have created for Apache Web Server.

So, the first step is to do the following:

```
$ docker search httpd
```

This gives the following output:

```

docker@boot2docker:~$ docker search httpd
NAME                DESCRIPTION                STARS     OFFICIAL   AUT
httpd               The Apache HTTP Server Project  84       [OK]
centos/httpd
diegomarangoni/httpd-amf  Apache HTTP Server compiled within Apache ...  1
vbatts/slackware-httpd    1
villjaste/httpd          A micro-sized httpd. Start serving files i...  0
efrecon/httpd            0
jibot/httpd              0
kaorimatzi/httpd         0
ubeas/ubol-httpd         0
zanphu/httpd-ssh         0
roninkenji/httpd-docker  0
microwebapps/httpd-frontend-atomicapp  0
ishentras/httpd          0
microwebapps/httpd-config-fpm  micro-webapps config serving PHP-FPM web-apps  0
tsweb/httpd              0
microwebapps/httpd-config-ssl  micro-webapps config adding SSL support to...  0
microwebapps/httpd-config-static  micro-webapps config serving static files ...  0
microwebapps/httpd-config-generic-proxy  micro-webapps config allowing proxying fro...  0
blinkmobile/apache-httpd  Starting with an Ubuntu base image, this p...  0
dimotsai/httpd           0
kumarpraveen/nodejs-httpd  0
eeacms/httpd             0

```

We will go with the OFFICIAL image for httpd. So let's learn a new command i.e. docker pull

This command only pulls (downloads) the Docker image to your local repository and does not run it.

Give the following command at the boot2Docker prompt:

```
docker@boot2docker:~$ docker pull httpd
```

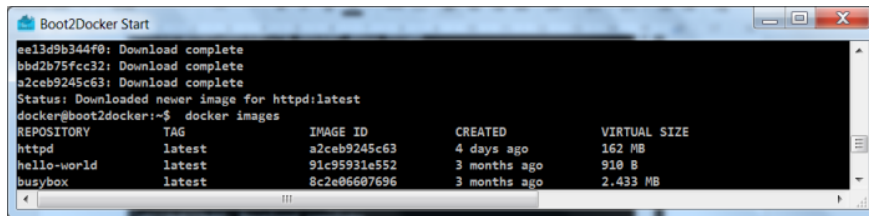
```

docker@boot2docker:~$ docker pull httpd
latest: Pulling from httpd

902b87aaec9: Pull complete
902b87aaec9: Download complete
9a61b6b1315e: Download complete
5c5ca7ea8ab9: Download complete
14cd4e2aa759: Download complete
89a04aa5aa59: Download complete
c44f556f8334: Download complete
1f99c5226c27: Download complete
d9119d522b8f: Download complete
d69bc311affe: Download complete
6920f7773389: Download complete
bee87a7c2cb7: Download complete
ee13d9b344f0: Download complete
bbd2b75fcc32: Download complete
a2ceb9245c63: Download complete
Status: Downloaded newer image for httpd:latest
docker@boot2docker:~$

```

This will download the httpd image. If you run a docker images command, you should see http listed in it.



```

Boot2Docker Start
ee13d9b344f0: Download complete
bbd2b75fcc32: Download complete
a2ceb9245c63: Download complete
Status: Downloaded newer image for httpd:latest
docker@boot2docker:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
httpd                latest             a2ceb9245c63       4 days ago         162 MB
hello-world          latest             91c95931e552       3 months ago       910 B
busybox              latest             8c2e06607696       3 months ago       2.433 MB

```

The [official documentation](#) for the httpd Docker image is something that you should look at. In fact make it a habit to go through the documentation page for each of the images present since it will give you clear instructions on how to run it and any other configuration details.

To make the default Apache Web Server run in the container, all we need to do is the following (Note that we are using the -d parameter i.e. running the Container in Detached mode). We have also given a name to our container via the—name parameter.

```

docker@boot2docker:~$ docker run -d --name MyWebServer
httpd
70c95e5394b4f4f03c0ca081249b181f02d49ca12d165d228ad0a9f
952b0bcf6

```

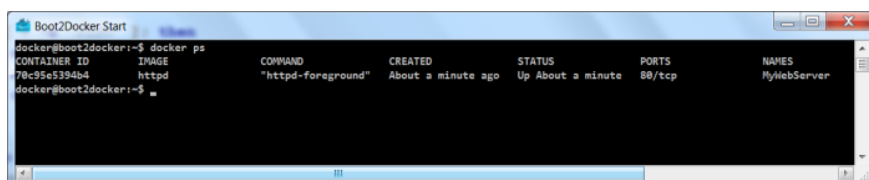
This will launch the default Apache HTTP Server based on the httpd image and since we have launched the container in detached mode, we got back a Container ID. Your ID need to be similar to the one that I got “70c95e5394b4f4f03c0ca081249b181f02d49ca12d165d228ad0a9f952b0bcf6”

Now, let us check if the Container is running. To do that we will use our familiar docker ps command as shown below:

```

docker@boot2docker:~$ docker ps

```



```

Boot2Docker Start
docker@boot2docker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
70c95e5394b4       httpd              "httpd-foreground" About a minute ago  Up About a minute  80/tcp             MyWebServer
docker@boot2docker:~$

```

Note that it is in running status. Also notice that the NAMES column now has the name that we specified for our Container i.e.

MyWebServer.

Detour: Stop the Container

If you wish to stop the Container at any point in time, you can now use the handy name that you provided instead of the CONTAINER_ID value in the docker stop command.

Try it now:

1. To stop the Container, give the following command:

```
$ docker stop MyWebServer
```

2. Try the :

```
$ docker ps command.
```

3. I also suggest to remove the Container via the following command:

```
docker rm MyWebServer
```

Great! Now, let us start it back again.

```
docker@boot2docker:~$ docker run -d --name MyWebServer  
httpd
```

```
<SOME_LARGE_CONTAINER_ID>
```

What is that PORTS column when you do a docker ps?

You will notice that there is an entry in the PORTS column. The value for that is “tcp:80”. This means that port 80 is being exposed by the Container. This looks reasonable since Apache Web Server default port for HTTP is 80. If you are adventurous enough, you can take a look at

Observe the output on your screen carefully. Even if you already have an instance of the boot2docker shell running and are not able to scroll up, don't worry. Simply launch another instance of boot2docker VM and it will get you into the shell again. Get used to having multiple shells open, that is a good way to keep navigating yourself through the various containers.

Now, observe carefully that it prints information about the Docker Daemon and how you can connect to it via any Docker client:

To connect the Docker client to the Docker daemon, please set:

```
export DOCKER_HOST=tcp://192.168.59.103:2376

export
DOCKER_CERT_PATH='C:\Users\irani_r\.boot2docker\certs\boot2docker-vm'

export DOCKER_TLS_VERIFY=1
```

Check out the IP Address of the Host. Also check out the port number : 2376 that Docker runs on (the port information is not relevant to us now, but it's good to know).

Also, see that the startup script printed out the IP Address for you :

```
192.168.59.103

connecting...
```

This output is a result of the \$(boot2docker ip) command that is present in the start.sh script file that you will find in the installation folder in which you installed Docker.

Another way, if you want to know the Host Address is to run the ifconfig command in Linux at the boot2docker prompt.

```
docker@boot2docker:~$ ifconfig
```

As an exercise, check out the output from `ifconfig` command above and search for the IP Address that we saw earlier. Alternately, try out `ifconfig eth1` and you should be able to find the host address.

docker@boot2docker:~\$ ifconfig eth1		
eth1	Link encap:Ethernet	HWaddr 08:00:27:52:7B:11
inet addr:192.168.59.103	Bcast:192.168.59.255	Mask:255.255.255.0
inet6 addr: fe80::a00:27ff:fe52:7b11/64	Scope:Link	
UP	BROADCAST	RUNNING MULTICAST MTU:1500 Metric:1
RX packets:3309	errors:0	dropped:0 overruns:0 frame:0
TX packets:248	errors:0	dropped:0 overruns:0 carrier:0
collisions:0	txqueuelen:1000	
RX bytes:334618 (326.7 KiB)	TX bytes:29485 (28.7 KiB)	

With that information in hand, let us launch the local browser and open the following URL:

<http://192.168.59.103>

Oops! No page appears. We were expecting at least the Apache Home page to appear over here but that does not seem to be the case.

This is because the default port that Apache Web Server runs on is port 80 but that port has not been exposed on the host side and as a result of that the web site is not accessible.

Show me the Proof?

Well I mentioned to you that port 80 is exposed by the Container but not by the Host. In other words, port 80 is the private port exposed by the Container but there does not seem to be a port mapped for the public facing host because of which we are seeing the problem.

It seems there is a docker command for that too. And its intuitively called port.

Try out the following command :

```
docker@boot2docker:~$ docker port MyWebServer
```

```
docker@boot2docker:~$
```

And the output is as expected i.e. there is no mapping done for the public facing port of the Host.

Luckily, help is on the way !

Random Port Mapping

First up, stop and remove the Container so that we can use the same Container Name i.e. MyWebServer.

```
docker@boot2docker:~$ docker stop MyWebServer
MyWebServer
```

```
docker@boot2docker:~$ docker rm MyWebServer
MyWebServer
```

```
docker@boot2docker:~$
```

Now, let us start the httpd Container with an extra parameter i.e. -P. What this parameter does is that it “Publish all exposed ports to random ports”. So in our case, the port 80 should get mapped to a random port, which is going to be the public port.

Execute the following command:

```
docker@boot2docker:~$ docker run -d--name MyWebServer -P httpd
60debd0d57bf292b0c3f006e4e52360feaa575e45ae3caea97637bb26
b490b10
```

Next, let us use the port command again to see what has happened:

```
docker@boot2docker:~$ docker port MyWebServer
```

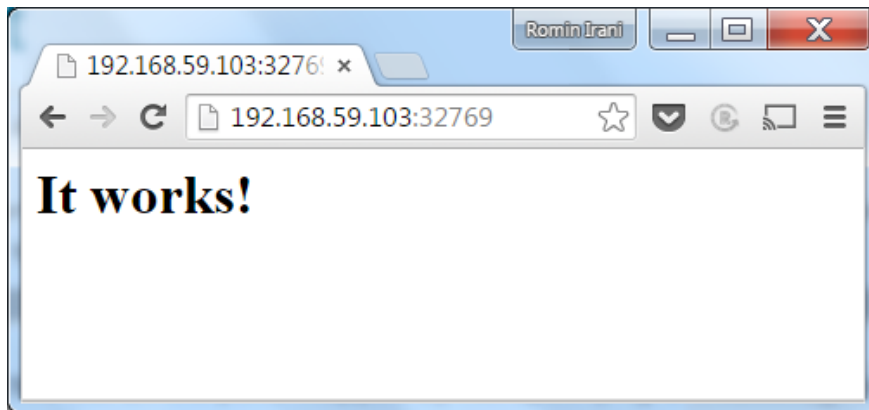
```
80/tcp -> 0.0.0.0:32769
```


We can see that port 80 has got mapped to port 32769. So if we access our web site at

```
http://<HostIP>/<HostPort>
```

it should work now.

So on my machine it is <http://192.168.59.103:32769> and it works!



Specific Port Mapping

So what if we wanted to map it to a port number other than 32769. You can do that via the `-p` (note the lowercase) parameter.

This parameter format is as follows:

```
-p HostPort:ContainerPort
```

For e.g. `-p 80:80` or `-p 8080:80`

The first parameter is the Host port and we are now making that 80. The second port is what Apache httpd exposes i.e. 80.

Let us check out everything again:

```
docker@boot2docker:~$ docker stop MyWebServer
MyWebServer

docker@boot2docker:~$ docker rm MyWebServer
MyWebServer

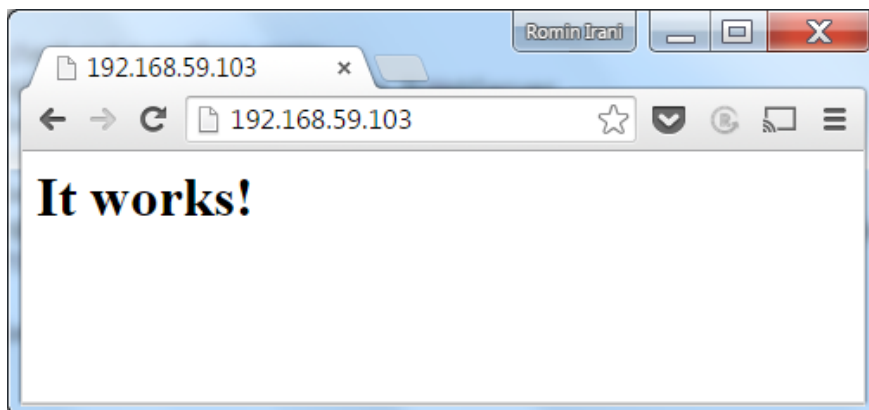
docker@boot2docker:~$ docker run -d --name MyWebServer -
p 80:80 httpd
02ce77550940cb1f37361b74af8913e46d6a507d06c2579b8a8b49e3
89b1e75f
```

Now, let us give the port command again:

```
docker@boot2docker:~$ docker port MyWebServer

80/tcp -> 0.0.0.0:80
```

Now you should be able to access your web site via port 80 (default port):



This brings us to an end of this section. You will still have a lot of questions in terms of Apache Web Server. Some of those would include where should you put your Web site files (HTML, CSS, etc) instead of this default one.

Those concerns are valid but the point in this session was to get you comfortable with various Docker commands, managing Containers, their ports and so on. When we come to the section for writing Docker files, things will get much clearer.